Lecture Notes in Computer Science 5564

*Commenced Publication in 1973*
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Andrew V. Goldberg  Yunhong Zhou (Eds.)

# Algorithmic Aspects in Information and Management

5th International Conference, AAIM 2009
San Francisco, CA, USA, June 15-17, 2009
Proceedings

Springer

Volume Editors

Andrew V. Goldberg
Microsoft Research – Silicon Valley
1065 La Avenida, Mountain View, CA 94062, USA
E-mail: goldberg@microsoft.com

Yunhong Zhou
Rocket Fuel Inc.
350 Marine Parkway, Marina Park Center, CA 94065, USA
E-mail: yunhong.zhou@gmail.com

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.2, B.2.4, G.1, G.2, G.3, I.1, I.2

LNCS Sublibrary: SL 3 – Information Systems and Application, incl. Internet/Web and HCI

ISSN 0302-9743
ISBN-10 3-642-02157-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02157-2 Springer Berlin Heidelberg New York



Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12690664 06/3180 5 4 3 2 1 0

# Preface

The papers in this volume were presented at the 5th International Conference on Algorithmic Aspects in Information and Management (AAIM 2009), held June 15–17, 2009, in San Francisco, California. The topics cover mostly algorithmic applications in information management and management science.

A total of 41 papers were submitted to the conference. After an on-line paper review process, the Program Committee accepted 25 papers to be presented. The international Program Committee included Tetsuo Asano, Marshall Bern, Daniel Bienstock, Danny Z. Chen, Camil Demetrescu, Lisa Fleischer, Rudolf Fleischer, Martin Frer, Andrew Goldberg, Mordecai Golin, Monika Henzinger, Seok-Hee Hong, Ming-Yang Kao, Xiang-Yang Li, Mohammad Mahdian, Tom McCormick, Junfeng Pan, Rong Pan, Panos Pardalos, Tomasz Radzik, Rajeev Raman, Martin Scholz, Robert Schreiber, Dou Shen, Xiaodong Wu, Jinhui Xu, Qiang Yang, Huaming Zhang, Yunhong Zhou and Binhai Zhu. It is expected that many of the accepted papers will appear in a more complete form in scientific journals.

The submitted papers are from Algeria, Argentina, Australia, Chile, China, Czech Republic, Denmark, France, Germany, Hong Kong, India, Iran, Israel, Japan, Taiwan, UK and USA. Each paper was evaluated by at least three Program Committee members (four is also common), assisted in some cases by subreferees (listed in the proceedings). In addition to selected papers, the conference also included two invited presentations by Andrei Broder and Edward Chang.

We thank all the people who made this meeting possible: the authors for submitting papers, the Program Committee members for their excellent work, and the two invited speakers. Finally, we thank the Organizing Committee members whose hard work make this conference possible.

June 2009

Andrew Goldberg
Yunhong Zhou

# Organization

## Program Committee Chairs

| | |
|---|---|
| Andrew Goldberg | Microsoft Research, USA |
| Yunhong Zhou | Rocket Fuel Inc., USA |

## Program Committee Members

| | |
|---|---|
| Tetsuo Asano | JAIST, Japan |
| Marshall Bern | PARC, USA |
| Daniel Bienstock | Columbia University, USA |
| Danny Z. Chen | University of Notre Dame, USA |
| Camil Demetrescu | University of Rome "La Sapienza", Italy |
| Lisa Fleischer | Dartmouth, USA |
| Rudolf Fleischer | Fudan University, Shanghai, China |
| Martin Frer | Penn State, USA |
| Mordecai Golin | HKUST, Hong Kong |
| Monika Henzinger | EPFL & Google Switzerland |
| Seok-Hee Hong | University of Sydney, Australia |
| Ming-Yang Kao | Northwestern University, USA |
| Xiang-Yang Li | Illinois Institute of Technology, USA |
| Mohammad Mahdian | Yahoo! Research, USA |
| Tom McCormick | Sauder School of Business, UBC, Canada |
| Junfeng Pan | Google, USA |
| Rong Pan | HP Labs, USA |
| Panos Pardalos | University of Florida, USA |
| Tomasz Radzik | King's College London, UK |
| Rajeev Raman | University of Leicester, UK |
| Martin Scholz | HP Labs, USA |
| Robert Schreiber | HP Labs, USA |
| Dou Shen | Microsoft Adcenter Labs, USA |
| Xiaodong Wu | University of Iowa, USA |
| Jinhui Xu | SUNY Buffalo, USA |
| Qiang Yang | HKUST, Hong Kong |
| Huaming Zhang | University of Alabama in Huntsville, USA |
| Binhai Zhu | Montana State, USA |

## Organizing Committee

| | |
|---|---|
| Binhai Zhu | Chair, Montana State University, USA |
| Andrew Goldberg (Co-chair) | Microsoft Research, USA |
| Yunhong Zhou (Co-chair) | Rocket Fuel Inc., USA |
| Rong Pan | HP Labs, USA |

## Referees

| | | |
|---|---|---|
| Bin Cao | Yunlong Liu | Jing Shao |
| Xin Dou | Vincent Loechner | Shiguang Wang |
| Stanley Fung | Andrew McGregor | Yajun Wang |
| Tobias Harks | Matus Mihalak | Yanwei Wu |
| Samir Khuller | David Mount | Xiaohua Xu |

# Table of Contents

# Algorithmic Challenge in Online Advertising

Andrei Z. Broder

Yahoo! Research
2821 Mission College Blvd.
Santa Clara, CA 95054, USA

Computational advertising is an emerging new scientific sub-discipline, at the intersection of large scale search and text analysis, information retrieval, statistical modeling, machine learning, classification, optimization, and microeconomics. The central challenge of computational advertising is to find the "best match" between a given user in a given context and a suitable advertisement. The context could be a user entering a query in a search engine ("sponsored search") , a user reading a web page ("content match" and "display ads"), a user watching a movie on a portable device, and so on. The information about the user can vary from scarily detailed to practically nil. The number of potential advertisements might be in the billions. Thus, depending on the definition of "best match" this challenge leads to a variety of massive optimization and search problems, with complicated constraints.

This talk will give an introduction to this area focusing mostly on the algorithmic challenges encountered in practice.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, p. 1, 2009.

# Parallel Algorithms for Collaborative Filtering

Edward Y. Chang[1,2]

[1] Google Beijing Research, Beijing 100084, China
[2] University of California, Santa Barbara, CA 93106, USA
edchang@google.com

Collaborative filtering has been widely used to predict the interests of a user. Given a users past activities, collaborative filtering predicts the users future preferences. This talk presents techniques and discoveries of our recent parallelization effort on collaborative filtering algorithms. In particular, parallel association mining and parallel latent Dirichlet allocation will be presented and their pros and cons analyzed. Some counter-intuitive results will also be presented to stimulate future parallel optimization research.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, p. 2, 2009.

# On the Approximability of Some Haplotyping Problems*

John Abraham[1], Zhixiang Chen[1], Richard Fowler[1], Bin Fu[1], and Binhai Zhu[2]

[1] Department of Computer Science, University of Texas-American, Edinburg, TX 78739-2999, USA
{jabraham,chen,fowler,binfu}@panam.edu
[2] Department of Computer Science, Montana State University, Bozeman, MT 59717-3880, USA
bhz@cs.montana.edu

**Abstract.** In this paper, we study several versions of optimization problems related to haplotype reconstruction/identification. The input to the first problem is a set $C_1$ of haplotypes, a set $C_2$ of haplotypes, and a set $G$ of genotypes. The objective is to select the minimum number of haplotypes from $C_2$ so that together with haplotypes in $C_1$ they resolve all (or the maximum number of) genotypes in $G$. We show that this problem has a factor-$O(\log n)$ polynomial time approximation. We also show that this problem does not admit any approximation with a factor better than $O(\log n)$ unless P=NP. For the corresponding reconstruction problem, i.e., when $C_2$ is not given, the same approximability results hold.

The other versions of the haplotype identification problem are based on single individual haplotyping, including the well-known Minimum Fragment Removal (MFR) and Minimum SNP Removal (MSR), which have both shown to be APX-hard previously. We show in this paper that MFR has a polynomial time $O(\log n)$-factor approximation. We also consider Maximum Fragment Identification (MFI), which is the complementary version of MFR; and Maximum SNP Identification (MSI), which is the complementary version of MSR. We show that, for any positive constant $\epsilon < 1$, neither MFI nor MSI has a factor-$n^{1-\epsilon}$ polynomial time approximation algorithm unless P=NP.

## 1 Introduction

Haplotype inference and identification is an important problem in computational biology. For instance, human haplotype data are crucial in identifying certain diseases. In diploid organisms (such as human) there are two (usually not identical) copies of each chromosome. Consequently we can collect the conflated data from two corresponding regions (*genotype*), relatively easily. On the other hand, for complex diseases which are affected by more than a single gene, it is more informative to have *haplotype* data, i.e., those data from exactly one copy of a

* This research is partially supported by NSF Career Award 0845376.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 3–14, 2009.

chromosome. It is known that genotype data are much easier and cheaper to collect compared with haplotype data, and nowadays it is possible to obtain haplotype data directly from experiments [6].

Most part of genomes between two humans are identical. The sites of genomes that makes the difference among human population are Single Nucleotide Polymorphisms (SNPs). The values of a set of SNPs on a particular chromosome copy define a *haplotype*. Haplotyping an individual consists of determining a pair of haplotypes, one for each copy of a given chromosome.

The problem of haplotyping a population has been widely studied with various objective functions. The haplotyping problem seeks to determine the optimal pair of haplotypes, which can be derived from data which may have inconsistence. Many versions of this problem have been proven to be NP-hard. This problem has been studied in a series of papers [1,24,25,19,4,16,18,22] in recent years. The problem of haplotyping a population has been studies in [5,9,11]. There are several versions of haplotyping problems, like they have to fit a perfect phylogeny. Our first set of problems are mostly following the principle of haplotype inference with *maximum parsimony* [10,12,14,16,26]. While the second set of problems focus on single individual haplotyping problems, which have been studied before in [17,19,1].

In this paper, we study several versions of optimization problems related to haplotype reconstruction/identification. We first study the Single-Side Haplotype Identification problem. The input to the problem is a set $C_1$ of haplotypes, a set $C_2$ of haplotypes, and a set $G$ of genotypes. The objective is to select the minimum number of haplotypes from $C_2$ so that together with haplotypes in $C_1$ they resolve all (or the maximum number of) genotypes in $G$. The background of this model is that we assume that the haplotypes in $C_1$ is already known and may be from one of the ancestors for the genomes of a group of descendants.

We show that this problem has a factor-$O(\log n)$ polynomial time approximation. We also show that this problem does not admit any approximation with a factor better than $O(\log n)$ unless P=NP.

The Single-Side Haplotype Reconstruction problem is also studied. It is similar to the Single-Side Haplotype Identification problem, but the input does not contain the candidate set $C_2$. We obtain a factor-$O(\log n)$ polynomial time approximation, and also an $\Omega(\log n)$ approximation lower bound as those for the Single-Side Haplotype Identification problem.

Secondly, we study some other versions of the single individual haplotyping problems and derive strong inapproximability results. In these problems, the haplotype of an individual is determined directly using incomplete and/or imperfect fragments of sequencing data. So the input for these problems is a matrix, each row representing a fragment. The known versions of this problem include Minimum Fragment Removal (MFR), which has a complementary version Maximum Fragment Identification (MFI); and Minimum SNP Removal (MSR), which has a complementary version Maximum SNP Identification (MSI). MFR and MSR have already been shown to be APX-hard [1]. We show that MFR has a factor-$O(\log n)$ polynomial time approximation. On the other hand, for any positive

constant $\epsilon < 1$, neither MFI nor MSI has a factor-$n^{1-\epsilon}$ polynomial time approximation unless P=NP.

This paper is organized as follows. In Section 2, we present some necessary definitions. In Section 3, we present the approximability results for single-side haplotype reconstruction/identification. In Section 4, we present the approximability results for several versions of the single individual haplotyping problem. In Section 5, we conclude the paper with several open problems.

## 2 Preliminaries

We define necessary concepts in this section. Throughout this paper we assume that all the haplotypes and genotypes are within a fixed block.

Suppose that we are given some local chromosomes each of $m$ linked SNPs. A *genotype* is a sequence $g = g_1 g_2 \cdots g_m$ such that $g_i$ denotes the genotype at locus $i$ and $g_i = 0, 1$ or 2 denotes that this locus is homozygous wild type, homozygous mutant or heterozygous, respectively. A (complete) *haplotype* is a binary (0 or 1) sequence of length $m$. Two haplotypes $h_1 = h_{11}h_{12}\cdots h_{1m}$ and $h_2 = h_{21}h_{22}\cdots h_{2m}$ *resolve* a genotype $g = g_1 g_2 \cdots g_m$ if and only if $g_i$ =2 implies that $h_{1i} = 0$ and $h_{2i} = 1$, or $h_{1i} = 1$ and $h_{2i} = 0$; $g_i = 1$ implies that $h_{1i} = h_{2i} = 1$; and $g_i = 0$ implies that $h_{1i} = h_{2i} = 0$, for $1 \leq i \leq m$. Example, $g = 02120$ is resolved by $h_1 = 00110$ and $h_2 = 01100$.

For two sets of haplotypes $C$, $C'$, and a set of genotypes $G$, we say $C$ and $C'$ resolve $G$ if for every $g \in G$, there exist $c \in C$ and $c' \in C'$ such that $c$ and $c'$ resolve $g$.

**Definition 1**

- *The* **Single-Side Haplotype Identification** *problem can be formally defined as follows:*
  *INPUT: A set $C_1$ of $|C_1|$ haplotypes, another set $C_2$ of $|C_2|$ haplotypes, a set $G$ of $|G|$ genotypes, and an integer $k$.*
  *QUESTION: find a least size subset $C_2' \subseteq C_2$, $|C_2'| \leq k$, such that $C_1$ and $C_2'$ resolve all (or the maximum number of) genotypes in $G$?. We also use SSHI problem to represent Single-Side Haplotype Identification problem.*
- *For an integer $d > 1$, a $d$-SSHI problem is a SSHI problem $C_1, C_2$ and $G$ with each haplotype in $C_2$ resolving at most $d$ genotypes in $G$.*
- *The* **Single-Side Haplotype Reconstruction** *problem can be formally defined as follows:*
  *INPUT: A set $C_1$ of $|C_1|$ haplotypes, a set $G$ of $|G|$ genotypes, and an integer $k$.*
  *QUESTION: find a least size set $C_2'$ of haplotypes, $|C_2'| \leq k$, such that $C_1$ and $C_2'$ resolve all (or the maximum number of) genotypes in $G$?. We also use SSHR problem to represent Single-Side Haplotype Reconstruction problem.*
- *For an integer $d > 1$, a $d$-SSHR problem is a SSHR problem $C_1$ and $G$ with each haplotype in the solution resolving at most $d$ genotypes in $G$.*

For a minimization (maximization) problem $\Pi$, an algorithm $\mathcal{A}$ provides a *performance guarantee* $\alpha$ for $\Pi$ if for every instance of $\Pi$ with optimal solution value $OPT$ the solution value returned by $\mathcal{A}$ is at most $\alpha \times OPT$ (at least $OPT/\alpha$). Usually we simply say that $\mathcal{A}$ is a factor-$\alpha$ *approximation* for $\Pi$. Throughout this paper, we are only interested in approximation algorithms running in polynomial time.

We will show in Section 3 that both SSHI and SSHR are NP-complete; in fact, there are no polynomial time approximation with a factor $o(\log n)$, unless P=NP.

For single individual haplotyping, one needs to determine the haplotype of an individual by working directly on (usually incomplete and/or imperfect) fragments of sequencing data. In this case, a haplotype can be written as a string over the alphabet {**A**,**B**,**-**} with "**-**" meaning lack of information or uncertainty about the nucleotide at that site. For human, being a *diploid* organism, each has two copies of each chromosome, one each from the individual's father and mother. So the problem is even more complicated.

Given a matrix $M$ of haplotype fragments (rows), each column represents a SNP site and each cell of the matrix denotes the choice of nucleotide seen at that SNP site on that fragment. A cell of $M$ can have a value **A** or **B** when the data is complete and error-free; otherwise, it can have a *hole*, denoted by **-**. Two rows $i_1$ and $i_2$ of $M$ *conflict* if there exists a column $j$ such that $M[i_1, j] \neq M[i_2, j]$ and $M[i_1, j], M[i_2, j] \in \{\mathbf{A}, \mathbf{B}\}$. $M$ is *feasible* if and only if the rows of $M$ can be partitioned into two groups such that rows in each group are mutually conflict-free.

Among the following problems, MFR and MSR were first studied in [17,19,1]. Readers are referred to [4,3] for some other problems on single individual haplotyping.

**Definition 2**

- *MFR (Minimum Fragment Removal): Given a SNP matrix, remove the minimum number of fragments (rows) so that the resulting matrix is feasible.*
- *MSR (Minimum SNP Removal): Given a SNP matrix, remove the minimum number of SNPs (columns) so that the resulting matrix is feasible.*
- *MFI (Maximum Fragment Identification): Given a SNP matrix, select the maximum number of fragments (rows) so that the resulting matrix is feasible.*
- *MSI (Maximum SNP Identification): Given a SNP matrix, select the maximum number of SNPs (columns) so that the resulting matrix is feasible.*

MFR and MSR were both shown to be APX-hard, with restricted versions (i.e., when there is no gap, or sequences of "-" between non-hole values) being polynomially solvable [1]. We show that MFR admits a factor-$O(\log n)$ approximation. For MFI and MSI, which can be thought of as the complementary versions of MFR and MSR, we show much stronger inapproximability results, i.e., they cannot be approximated with a factor $n^{\epsilon}$ for any constant $0 < \epsilon < 1$.

## 3 Approximability for Single-Side Haplotype Identification/Reconstruction

In this section, we cover the first set of problems on haplotype identification and reconstruction.

**Theorem 1.** *There exists a factor-$O(\log n)$ polynomial time approximation for the Single-Side Haplotype Identification problem.*

*Proof.* We convert this problem into the set cover problem. The $O(\log n)$-factor approximation algorithm for the set cover problem [2,15,20] brings an $O(\log n)$-factor approximation for the Single-Side Haplotype Identification problem.

Assume that $C_1$, $C_2$ and $G$ are the three input sets for the Single-Side Haplotype Identification problem (see definition 1). A set cover problem is derived as follows.

Let $S = G$. For each $h_i \in C_2$, let $S_i$ be the set of all $g_j \in G$ such that $h_i$ and $h'$ resolve $g_j$ for some $h' \in C_1$. The input of the set cover problem is $S, S_1, \cdots, S_m$, where $m = |C_2|$.

It is easy to see that if the Single-Side Haplotype Identification problem with input $C_1, C_2$, and $G$ has a solution $C_2' \subseteq C_2$ with size $|C_2'|$ if and only if the set cover problem with input $S, S_1, \cdots, S_m$ has a solution of $|C_2'|$ sets from $S_1, \cdots, S_m$.

Since the set cover problem has an $O(\log n)$-factor polynomial time approximation, we have an $O(\log n)$-factor approximate algorithm for the Single-Side Haplotype Identification problem. □

**Theorem 2.** *There is no factor-$o(\log n)$ polynomial time approximation for the Single-Side Haplotype Identification problem unless P=NP.*

*Proof.* We prove this theorem by showing that Set Cover can be reduced to SSHI. It is known that Set Cover cannot be approximated with a factor-$o(\log n)$ polynomial time approximation [23].

Assume that $X$, $S_1, \cdots, S_m$ with ($S_i \subseteq X$ for $i = 1, \cdots, m$) are the input for Set Cover, which seeks to find a least number of sets among $S_1, \cdots, S_m$ to cover the elements in the base set $X$. A SSHI instance can be constructed as follows:

Assume that $X$ contains $n$ elements $e_1, \cdots, e_n$. $G$ contains $g_1, \cdots, g_n$, where $g_i = (22)^{i-1}00(22)^{n-i}2$ for $i = 1, \cdots, n$. $C_2$ contains $f_1, \cdots, f_m$ such that each $f_i$ corresponds to a subset $S_i$, and $f_i = a_1b_1 \cdots a_nb_n0$ with $a_jb_j = 01$ for $x_j \notin S_i$ and $a_jb_j = 00$ for $x_j \in S_i$. For $C_1$, each set $S_i$ adds $|S_i|$ sequences $h_{i,1}, \cdots, h_{i,|S_i|}$ to it. Assume that $S_i = \{e_{i_1}, \cdots, e_{i_t}\}$. Then $h_{i,j} = a_1b_1 \cdots a_nb_n1$, where $a_jb_j = 10$ for $e_j \notin S_i$, $a_tb_t = 11$ for $e_t \in S_i$ with $t \neq j$, and $a_jb_j = 00$ for $e_j \in S_i$.

Note that each $h_{i,j}$ can only combine with $f_i$ to resolve some $g_t$. We have the following example based on the above construction.

Input for Set Cover:

$$X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$$

$$S_1 = \{x_1, x_3\}$$
$$S_2 = \{x_1, x_2, x_3\}$$
$$S_3 = \{x_2, x_3\}$$
$$S_4 = \{x_3, x_5\}$$
$$S_5 = \{x_4, x_6\}$$

The derived SSHI instance:

$$g_1 = 00\ 22\ 22\ 22\ 22\ 22\ 2$$
$$g_2 = 22\ 00\ 22\ 22\ 22\ 22\ 2$$
$$g_3 = 22\ 22\ 00\ 22\ 22\ 22\ 2$$
$$g_4 = 22\ 22\ 22\ 00\ 22\ 22\ 2$$
$$g_5 = 22\ 22\ 22\ 22\ 00\ 22\ 2$$
$$g_6 = 22\ 22\ 22\ 22\ 22\ 00\ 2$$

$$f_1 = 00\ 01\ 00\ 01\ 01\ 01\ 0$$
$$h_{11} = 00\ 10\ 11\ 10\ 10\ 10\ 1$$
$$h_{12} = 11\ 10\ 00\ 10\ 10\ 10\ 1$$

$$f_2 = 00\ 00\ 01\ 01\ 00\ 01\ 0$$
$$h_{21} = 00\ 10\ 10\ 10\ 11\ 10\ 1$$
$$h_{22} = 11\ 00\ 10\ 10\ 11\ 10\ 1$$
$$h_{23} = 11\ 11\ 10\ 10\ 00\ 10\ 1$$

$$f_3 = 01\ 00\ 00\ 01\ 01\ 01\ 0$$
$$h_{31} = 10\ 00\ 11\ 10\ 10\ 10\ 1$$
$$h_{32} = 10\ 11\ 00\ 10\ 10\ 10\ 1$$

$$f_4 = 01\ 01\ 00\ 01\ 00\ 01\ 0$$
$$h_{41} = 10\ 10\ 00\ 10\ 11\ 10\ 1$$
$$h_{42} = 10\ 01\ 11\ 10\ 00\ 10\ 1$$

$$f_5 = 01\ 01\ 01\ 00\ 01\ 00\ 0$$
$$h_{51} = 10\ 10\ 10\ 00\ 10\ 11\ 1$$
$$h_{52} = 10\ 10\ 10\ 11\ 10\ 00\ 1$$

Assume that $S_{i_1}, \cdots, S_{i_k}$ be the optimal solution for the set cover problem. Then we have that the subset $C_2' = \{f_{i_1}, \cdots, f_{i_k}\}$ of $C_2$ such that $C_1$ and $C_2'$

resolve $G$. On the other hand, if a subset $\{f_{j_1}, \cdots, f_{j_u}\}$ of $C_2$ is an optimal solution for the SSHI problem, then $S_{j_1}, \cdots, S_{j_u}$ can cover the set $S$.

Therefore, for every integer $k \geq 1$, there exists $C_2' \subseteq C_2$ with $|C_2'| = k$ such that $C_1$ and $C_2'$ resolve $G$ if and only if there exists $S' \subseteq S$, with $|S'| = k$, which covers elements in $X$.

With respect to the above example, we have
$C_1 = \{h_{11}, h_{12}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}, h_{41}, h_{42}, h_{51}, h_{52}\}$, $C_2 = \{f_1, f_2, f_3, f_4, f_5\}$, and $G = \{g_1, g_2, g_3, g_4, g_5, g_6\}$. The optimal solution is $\{f_2, f_4, f_5\}$. □

**Theorem 3.** *There exists a polynomial time algorithm for the 2-SSHI problem.*

*Proof.* This problem can be converted into 2-set cover problem, which has a polynomial time solution via the polynomial time algorithm for the matching problem. □

**Theorem 4.** *There exists a $\frac{4}{3}$-factor polynomial time approximation for the 3-SSHI problem.*

*Proof.* By Duh and Fürer's algorithm [7], we have a $\frac{4}{3}$-factor approximate algorithm for the 3-set cover problem. Therefore, there is a $\frac{4}{3}$-factor approximate algorithm for the 3-SSHI problem. □

We next sketch how the above results for SSHI can hold for SSHR.

**Theorem 5.** *There exists a factor-$O(\log n)$ polynomial time approximation for the SSHR problem.*

*Proof.* We convert the SSHR problem into an instance of SSHI and use the approximation algorithm for SSHI to solve the SSHR problem. Assume that the input of the SSHR problem is a set of $C_1$ of haplotypes, and a set $G$ of genotypes. We construct the set $C_2$ to contain all haplotypes $h$ such that there exists a haplotype $h' \in C_1$ and a genotype $g \in G$, and together with $h'$, $h$ resolves $g$. It is easy to see that $C_2$ contains at most $|C_1||G|$ haplotypes.

The newly constructed SSHI instance has input $C_1, C_2$ and $G$. It is easy to see that the optimal solution of two problems are the same. By Theorem 1, we have a factor-$O(\log n)$ polynomial time approximation for the SSHR problem. □

**Theorem 6.** *There is no $o(\log n)$-factor polynomial time approximation for the SSHR problem unless P=NP.*

*Proof.* The proof follows from that of Theorem 2. Assume that $X$, $S_1, \cdots, S_m$ with ($S_i \subseteq X$ for $i = 1, \cdots, m$) are the input of a set cover problem $P$. We construct another SSHR instance $Q$ such that $P$ has a solution of $k$ subsets iff $Q$ has a solution of $k$ haplotypes. $Q$ is constructed as that in Theorem 2 with $C_1$ containing the same set of $h_{i,j}$ and $G$ containing the same set of genotypes. We do not construct those $f_i$ ($1 \leq i \leq m$), i.e., $C_2 = \emptyset$.

Assume that a haplotype $f$ and $h_{i,j}$ resolves some genotype $g_t$. For the pair 00 in $h_{i,j}$, the corresponding pair in $f$ has to be 00 since each $g_t$ has exactly one 00

pair. For the pair 11 in $h_{i,j}$, the corresponding pair in $f$ has to be 00 so that they can handle a 22 pair in $g_t$. For the pair 10 in $h_{i,j}$, the corresponding pair in $f$ has to be 01. Therefore, $f$ must correspond to the $f_i$ constructed in Theorem 2 via set $S_i$. Therefore, any solution should be selected from $\{f_1, \cdots, f_m\}$, otherwise, it is impossible to match those $h_{i,j}$ to resolve genotypes in $G$. □

**Theorem 7.** *There exists a polynomial time algorithm for 2-SSHR.*

*Proof.* This problem can be converted into 2-set cover problem, which has a polynomial time solution via the polynomial time algorithm for the matching problem. □

**Theorem 8.** *There exists a $\frac{4}{3}$-factor polynomial time approximation for 3-SSHR.*

*Proof.* By Duh and Fürer's algorithm [7], we have a $\frac{4}{3}$-factor approximate algorithm for the 3-set cover problem. Therefore, there is a $\frac{4}{3}$-factor approximate algorithm for the 3-SSHR problem. □

## 4 Approximability for Some Single Individual Haplotyping Problems

In this section we show some lower and upper bounds for the approximation of some single individual haplotyping problems by reducing some other well known problems to them.

Define 2-Disjoint-Clique as the problem of deciding whether an undirected graph contains two disjoint cliques of total size (number of vertices) at least $k_2$, for a given $k_2 > 0$. We show with the following lemma that 2-Disjoint-Clique is just as hard as Clique, even to approximate.

**Lemma 1.** *2-Disjoint-Clique is NP-complete; moreover, for any constant $\epsilon > 0$, there is no polynomial time $n^{1-\epsilon}$-factor polynomial time approximation for it unless P=NP.*

*Proof.* We reduce Clique to 2-Disjoint-Clique. It is known that Clique cannot be approximated with a factor-$n^{1-\epsilon}$ polynomial time approximation [13]. Let $G = (V, E)$ be an undirected graph which is the input to Clique. We construct $G'$ as two copies of $G$, with vertices and edges relabelled. It is easy to see that $G$ has a clique of size $k$ iff $G'$ has two disjoint cliques of size $2k$. Therefore, the lemma follows. □

**Theorem 9.** *For any constant $\epsilon > 0$, there is no factor-$n^{1-\epsilon}$ polynomial time approximation for the MFI problem unless P=NP.*

*Proof.* We reduce the 2-Disjoint-Clique problem to MFI. Assume that $G = (V, E)$ is an undirected graph. We construct a SNP matrix $M$ as follows. Each fragment (row) corresponds to a vertex of $G$. Each column checks whether two vertices of $G$ do not define an edge in $E$. So, each column of $M$ is indexed by

a pair $\langle u, v\rangle$ of vertices in $G$ and each row is indexed by a vertex $w$ of $G$. For two vertices $u$ and $v$ with $u < v$, put **A** at row $u$ and **B** at row $v$ in the column $\langle u, v\rangle$. All the remaining positions of column $\langle u, v\rangle$ hold the character $-$. By this construction, if there is no edge between $u$ and $v$, the corresponding rows (i.e., row $u$ and row $v$ in $M$) will conflict. Then it is easy to see that there are two disjoint cliques of size $k$ in $G$ iff there are $k$ rows of $M$ which define a feasible sub-matrix. □

**Theorem 10.** *For any constant $\epsilon > 0$, there is no factor-$n^{\epsilon}$ polynomial time approximation for the MSI problem unless P=NP.*

*Proof.* We reduce the Independent Set problem to MSI. Note that Independent Set is the complementary version of Clique, so the inapproximability results for the two problems are similar [13]. The actual reduction is similar to that in [1]. Assume that $G = (V, E)$ is an undirected graph. The matrix $M$ is of size $(2|E| + 1) \times |V|$. Each vertex of $G$ corresponds to a column in $M$. For each edge $e_i = (u, v)$ in $E$, row $i$ and row $2i$ has character **A** at column $u$, row $i$ has **A** at column $v$, and row $2i$ has **B** at column $v$. The special row $2|E| + 1$ contains **B** at every position. It is easy to see that there are $k$ independent vertices in $G$ iff $M$ has $k$ columns which define a feasible matrix.

Assume that $C$ is a set of columns selected from the matrix $M$ such that the resulting matrix formed by the columns of $C$ is feasible. For every edge $(u, v)$ in $E$, it is impossible that both columns $u$ and $v$ are in $C$. Otherwise, the matrix formed by $C$ is not feasible. Thus, $C$ induces a set of independent set of vertices in the graph $G$.

Let $I$ be a set of independent vertices in $G$. Then we claim that the sub-matrix $M'$ formed by the set of columns with indices from $I$ is feasible. For two rows $i$ and $2i$, with $e_i = (u, v)$, as $(u, v)$ is in $E$, one of $u$ and $v$ will not be in $I$. (Otherwise, due to the **B**'s in the last row, there is no way to make the resulting sub-matrix feasible.) Assume that $u$ is in $I$, we can then put row $i$ and $2i$ into different groups. Thus, the resulting matrix $M'$ is feasible. □

We next show that MFR admits a factor-$O(\log n)$ polynomial time approximation. This can be done through the MVDB problem.

Minimum Vertex-Deletion Bipartite Subgraph Problem (MVDB): Given an undirected graph $G = (V, E)$, find a minimum size subset $V' \subseteq V$ such that $G - V'$ is a bipartite graph.

**Theorem 11.** *The two problems MFR and MVDB are equivalent in terms of polynomial time approximation. In other words, it can be expressed in the following two facts:*

*(1) if there exists a polynomial time $f(n, m)$-factor approximation algorithm for MVDB with an input graph of $n$ vertices and $m$ edges, then there exists another polynomial time $f(n, O(n^2))$-approximation for MFR with an input matrix of $n$ fragments and $m$ SNPs, where $f(n, m)$ is nondecreasing function from $N \times N$ to $N$ for both variables.*

*(2) if there exists a polynomial time $g(n, m)$-factor approximation algorithm for MFR with an input matrix of $n$ fragments and $m$ SNPs, then there exists*

*another polynomial time $g(n, m)$-approximation for MVDB with an input graph of $n$ vertices and $m$ edges, where $g(n, m)$ is nondecreasing function from $n \times N$ to $N$ for both variables.*

*Proof.* We first prove part (1). Assume that $\mathcal{A}$ is an approximation algorithm for MVDB with approximation ratio $f(n, m)$ for a graph with $n$ vertices and $m$ edges. We will convert $\mathcal{A}$ into another approximation algorithm for MFR.

Let $M$ be an $n \times m$ SNP matrix for the MFR problem. A graph $G = (V, E)$ is constructed as follows. Each row of $M$ has a vertex in $V$. For every two rows $u$ and $v$ of $M$, put an edge $(u, v)$ in $E$ if there is a conflict between row $u$ and $v$. Therefore, the graph $G$ has $n$ vertices and at most $O(n^2)$ edges. Since $\mathcal{A}$ is an approximation algorithm for the MVDB problem, we obtain an polynomial time approximation algorithm for MFR with an approximation ratio $f(n, O(n^2))$.

Now we show part (2). Let $\mathcal{A}'$ be a polynomial time approximation for MFR with approximation ratio $g(n, m)$ for every input matrix of $n$ rows and $m$ columns. We convert $\mathcal{A}'$ into an approximation for the MVDB problem. Let $G = (V, E)$ be an input for the MVDB problem. We construct an instance $M$ for the MFR problem. The matrix $M$ has $|V|$ rows and $|E|$ columns. Each column is indexed by an edge $(u, v)$ in $E$ and each row is indexed by a vertex $u$ in $V$. For each edge $(u, v) \in E$, the column of $M$ with index $(u, v)$ has entry **A** at row $u$, entry **B** at row $v$ and hole $-$ at all of the remaining entries.

Assume that $V' \subseteq V$ is a subset of vertices in $V$ such that $G - V'$ is a bipartite graph. We can also remove those rows in $M$ with the index from $V'$ and make the remaining matrix feasible.

On the other hand, assume that $R$ is the subset of rows in $M$ such that removing those rows in $R$ makes the remaining matrix feasible. We can also remove the subset $V'$ of vertices, which correspond to the indexes of rows in $R$, and make the resulting graph $G - V'$ bipartite. Therefore, the approximation $\mathcal{A}'$ for MFR is converted into another approximation algorithm for MVDB with approximation $g(n, m)$. □

**Corollary 1.** *There exists a factor-$O(\log n)$ polynomial time approximation for the MFR problem, where $n$ is the number of rows in the input SNP matrix.*

*Proof.* It is known that there exists a polynomial time $O(\log n)$-factor approximation algorithm for the MVDB problem [8]. It follows from Theorem 11 that there is a factor-$O(\log n)$ polynomial time approximation for MFR. □

Since MVDB is APX-hard [21], Theorem 11 also implies that MFR is APX-hard, which was shown in [1] using a different reduction.

## 5 Conclusion

In this paper we study two classes of problems related to haplotyping. For the first set of problems, we investigate single-side haplotype identification and reconstruction. This is related to haplotype inference with maximum parsimony.

We show tight approximability bounds for the two problems Single-Side Haplotype Inference and Single-Side Haplotype Reconstruction. Another interesting problem is to decide whether the 3-SSHR problem is NP-complete.

For the second set of problems on single individual haplotyping, we investigate variants of the well-known APX-hard problems MFR and MSR, we show that there exists a polynomial time $O(\log n)$-factor approximation algorithm for the MFR problem. And for the complementary versions of MFR and MSR; namely, MFI and MSI, we show much stronger in approximability results (i.e., they cannot be approximated with a factor-$n^{\epsilon}$ polynomial time approximation, unless P=NP). An interesting problem is to find whether there exists a factor-$O(\log n)$ polynomial time approximation for the MSR problems.

## References


1. Bafna, V., Istrail, S., Lancia, G., Rizzi, R.: Polynomial and APX-hard cases of the individual haplotyping problem. Theoretical Computer Science 335, 109–125 (2005)
2. Chvátal, V.: A greedy heuristic for the set-covering problem. Math. Oper. Res. 4, 233–235 (1979)
3. Chen, Z., Fu, B., Sweller, R., Yang, B., Zhao, Z., Zhu, B.: Linear probabilistic algorithms for the singular haplotype reconstruction problem from SNP fragments. J. Computational Biology 15, 535–546 (2008)
4. Cilibrasi, R., van Iersel, L., Kelk, S., Tromp, J.: The complexity of the single individual SNP haplotyping problem. Algorithmica 49, 13–36 (2007)
5. Clark, A.: Inference of haplotypes from PCR-amplified samples of diploid populations. Molecular Biology Evolution 7, 111–122 (1990)
6. Douglas, J., Boehnke, M., Gillanders, E., Trent, J., Gruber, S.: Experimentally-driven haplotypes substantially increase the efficiency of linkage disequillibrium studies. Nat. Genetics 28, 361–364 (2001)
7. Duh, R.-c., Fürer, M.: Approximation of k-set cover by semi-local optimization. In: Proc. 29th ACM Symp. on Theory of Comput (STOC 1997), pp. 256–264 (1997)
8. Garg, N., Vazirani, V.V., Yannakakis, M.: Multiway cuts in directed and node weighted graphs. In: Shamir, E., Abiteboul, S. (eds.) ICALP 1994. LNCS, vol. 820, pp. 103–111. Springer, Heidelberg (1994)
9. Gusfield, D.: A practical algorithm for optimal inference of haplotype from diploid populations. In: ISMB 2000, pp. 183–189 (2000)
10. Gusfield, D.: Inference of haplotypes from samples of diploid populations: complexity and algorithms. J. Computational Biology 8, 305–323 (2001)
11. Gusfield, D.: Haplotyping as perfect phylogeny: Conceptual framework and efficient solutions. In: RECOMB 2002, pp. 166–175 (2002)
12. Gusfield, D.: Haplotype inference by pure parsimony. In: Baeza-Yates, R., Chávez, E., Crochemore, M. (eds.) CPM 2003. LNCS, vol. 2676, pp. 144–155. Springer, Heidelberg (2003)
13. Hästad, J.: Clique is hard to approximate within $n^{1-\epsilon}$. Acta Mathematica 182, 105–142 (1999)
14. Huang, Y.-T., Chao, K.-M., Chen, T.: An approximation algorithm for haplotype inference by maximum parsimony. J. Computational Biology 12, 1261–1274 (2005)
15. Johnson, D.: Approximation algorithms for combinatorial problems. J. Comput. System Sci. 9, 256–278 (1974)

16. Lancia, G., Pinotti, M.C., Rizzi, R.: Haplotyping populations by pure parsimony: complexity and algorithms. INFORMS Journal on computing 16, 348–359 (2004)
17. Lancia, G., Bafna, V., Istrail, S., Lippert, R., Schwartz, R.: SNPs Problems, Complexity and Algorithms. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 182–193. Springer, Heidelberg (2001)
18. Lancia, G., Rizzi, R.: A polynomial solution to a special case of the parsimony haplotyping problem. Operations Research letters 34, 289–295 (2006)
19. Lippert, R., Schwartz, R., Lancia, G., Istrail, S.: Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. Briefings in bioinformatics 3, 23–31 (2002)
20. Lóvasz, L.: On the ratio of optimal integral and fractional covers. Discrete Mathematics 13, 383–390 (1975)
21. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. ACM 41, 960–981 (1994)
22. Panconesi, A., Sozio, M.: Fast Hare: A fast heuristic for single individual SNP haplotype reconstruction. In: Jonassen, I., Kim, J. (eds.) WABI 2004. LNCS (LNBI), vol. 3240, pp. 266–277. Springer, Heidelberg (2004)
23. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and subconstant error-probability PCP characterization of NP. In: Proc. 29th ACM Symp. on Theory of Comput. (STOC 1997), pp. 475–484 (1997)
24. Rizzi, R., Bafna, V., Istrail, S., Lancia, G.: Practical algorithms and fixed-parameter tractability for the single individual SNP haplotyping problem. In: Guigó, R., Gusfield, D. (eds.) WABI 2002. LNCS, vol. 2452, pp. 29–43. Springer, Heidelberg (2002)
25. Wang, R.S., Wu, L.Y., Li, Z.P., Zhang, X.S.: Haplotype reconstruction from SNP fragments by minimum error correction. Bioinformatics 21, 2456–2462 (2005)
26. Wang, L., Xu, Y.: Haplotype inference by maximum parsimony. Bioinformatics 19, 1773–1780 (2003)

# On Acyclicity of Games with Cycles*

Daniel Andersson[1], Vladimir Gurvich[2], and Thomas Dueholm Hansen[1]

[1] Dept. of Computer Science, Aarhus University
{koda,tdh}@cs.au.dk
[2] RUTCOR, Rutgers University
gurvich@rutcor.rutgers.edu

**Abstract.** We study restricted improvement cycles (ri-cycles) in finite positional $n$-person games with perfect information modeled by directed graphs (digraphs) that may contain cycles. We obtain criteria of restricted improvement acyclicity (ri-acyclicity) in two cases: for $n = 2$ and for acyclic digraphs. We provide several examples that outline the limits of these criteria and show that, essentially, there are no other ri-acyclic cases. We also discuss connections between ri-acyclicity and some open problems related to Nash-solvability.

**Keywords:** Positional game, game form, improvement cycle, restricted improvement cycle, restricted acyclicity, Nash equilibrium, Nash-solvability.

## 1 Main Concepts and Results

### 1.1 Games in Normal Form

**Game Forms and Utility Functions.** Given a set of players $I = \{1, \ldots, n\}$ and a set of strategies $X_i$ for each $i \in I$, let $X = \prod_{i \in I} X_i$.

A vector $x = (x_i, i \in I) \in X$ is called a *strategy profile* or *situation*.

Furthermore, let $A$ be a set of outcomes. A mapping $g : X \to A$ is called a *game form*. In this paper, we restrict ourselves to *finite* game forms, that is, we assume that sets $I, A$ and $X$ are finite.

Then, let $u : I \times A \to \mathbb{R}$ be a utility function. Standardly, the value $u(i, a)$ (or $u_i(a)$) is interpreted as the payoff to player $i \in I$ in case of the outcome $a \in A$. In figures, the notation $a <_i b$ means $u_i(a) < u_i(b)$.

Sometimes, it is convenient to exclude ties. Accordingly, $u$ is called a *preference profile* if the mapping $u_i$ is injective for each $i \in I$; in other words, $u_i$ defines a complete order over $A$ describing the preferences of player $i \in I$.

A pair $(g, u)$ is called a *game in normal form*.

* The full version of this paper with complete proofs is available as the research report, [1]. This research was supported by the Center for Algorithmic Game Theory at Aarhus University, funded by the Carlsberg Foundation. The second author was partially supported also by DIMACS, Center for Discrete Mathematics and Theoretical Computer Science, Rutgers University, and by Graduate School of Information Science and Technology, University of Tokyo.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 15–28, 2009.

**Improvement Cycles and Acyclicity.** In a game $(g, u)$, an *improvement cycle (im-cycle)* is defined as a sequence of $k$ strategy profiles $\{x^1, \ldots, x^k\} \subseteq X$ such that $x^j$ and $x^{j+1}$ coincide in all coordinates but one $i = i(j)$ and, moreover, $u_i(x^{j+1}) > u_i(x^j)$, that is, player $i$ makes profit by substituting strategy $x_i^{j+1}$ for $x_i^j$; this holds for all $j = 1, \ldots, k$ and, standardly, we assume that $k + 1 = 1$.

A game $(g, u)$ is called *im-acyclic* if it has no im-cycles. A game form $g$ is called *im-acyclic* if for each $u$ the corresponding game $(g, u)$ is im-acyclic.

We call $x^{j+1}$ an improvement with respect to $x^j$ for player $i = i(j)$. We call it a *best reply* (BR) improvement if player $i$ can get no strictly better result provided all other players keep their strategies. Correspondingly, we introduce the concepts of a BR im-cycle and BR im-acyclicity. Obviously, im-acyclicity implies BR im-acyclicity but not vice versa.

**Nash Equilibria and Acyclicity.** Given a game $(g, u)$, a strategy profile $x \in X$ is called a *Nash equilibrium* (NE) if $u_i(x) \geq u_i(x')$ for each $i \in I$, whenever $x'_j = x_j$ for all $j \in I \setminus \{i\}$. In other words, $x$ is a NE if no player can get a strictly better result by substituting a new strategy ($x'_i$ for $x_i$) when all other players keep their old strategies. Conversely, if $x$ is not a NE then there is a player who can improve his strategy. In particular, he can choose a best reply. Hence, a NE-free game $(g, u)$ has a BR im-cycle. Let us remark that the last statement holds only for finite games, while the converse statement is not true at all.

A game $(g, u)$ is called *Nash-solvable* if it has a NE. A game form $g$ is called *Nash-solvable* if for each $u$ the corresponding game $(g, u)$ has a NE.

The main motivation for the study of im-acyclicity is to prove the existence of Nash equlibria. In addition, im-acyclicity leads to a natural algorithm for computing such equilibria: iteratively perform improvement steps until an equlibrium is reached.

### 1.2 Positional Games with Perfect Information

**Games in Positional Form.** Let $G = (V, E)$ be a finite directed graph (digraph) whose vertices $v \in V$ and directed edges $e \in E$ are called *positions* and *moves*, respectively. The edge $e = (v', v'')$ is a move from position $v'$ to $v''$. Let $out(v)$ and $in(v)$ denote the sets of moves from and to $v$, respectively.

A position $v \in V$ is called *terminal* if $out(v) = \emptyset$. Let $V_T$ denote the set of all terminals. Let us also fix a starting position $v_0 \in V \setminus V_T$. A directed path from $v_0$ to a terminal position is called a *finite play*.

Furthermore, let $D : V \setminus V_T \to I$ be a decision mapping, with $I$ being the set of players. We say that the player $i = D(v) \in I$ makes a decision (move) in a position $v \in D^{-1}(i) = V_i$. Equivalently, $D$ is defined by a partition $D : V = V_1 \cup \ldots \cup V_n \cup V_T$.

The triplet $\mathcal{G} = (G, D, v_0)$ is called a *positional game form*.

**Cycles, Outcomes, and Utility Functions.** Let $C$ denote the set of simple (that is, not self-intersecting) directed cycles in $G$.

The set of outcomes $A$ can be defined in two ways:

(i) $A = V_T \cup C$, that is, each terminal and each directed cycle is a separate outcome.
(ii) $A = V_T \cup \{C\}$, that is, each terminal is an outcome and all directed cycles constitute one special outcome $c = \{C\}$; this outcome will also be called *infinite play*.

Case (i) was considered in [3] for two-person games ($n = 2$). In this paper, we analyze case (ii) for $n$-person games.

*Remark 1.* Let us mention that as early as in 1912, Zermelo already considered two-person zero-sum games with repeated positions (directed cycles) in his pioneering work [12], where the game of chess was chosen as a basic example. In this game, repeating a position in a play results in a draw.

Note that players can rank outcome $c$ arbitrarily in their preferences. In contrast, in [2] it was assumed that infinite play $c \in A$ is the worst outcome for all players.

**Positional Games in Normal Form.** A triplet $\mathcal{G} = (G, D, v_0)$ and quadruple $(G, D, v_0, u) = (\mathcal{G}, u)$ are called a *positional form* and a *positional game*, respectively. Positional games can also be represented in *normal form*, as described below. A mapping $x : V \setminus V_T \to E$ that assigns to every non-terminal position $v$ a move $e \in out(v)$ from this position is called a *situation* or *strategy profile*.

A *strategy* of player $i \in I$ is the restriction $x_i : V_i \to E$ of $x$ to $V_i = D^{-1}(i)$.

*Remark 2.* A strategy $x_i$ of a player $i \in I$ is interpreted as a decision plan for every position $v \in V_i$. Note that, by definition, the decision in $v$ can depend only on $v$ itself but not on the preceding positions and moves. In other words, we restrict the players to their pure positional strategies.

Each strategy profile $x \in X$ uniquely defines a play $p(x)$ that starts in $v_0$ and then follows the moves prescribed by $x$. This play either ends in a terminal of $V_T$ or results in a cycle, $a(x) = c$ (infinite play). Thus, we obtain a game form $g(\mathcal{G}) : X \to A$, which is called the *normal form* of $\mathcal{G}$. This game form is standardly represented by an $n$-dimensional table whose entries are outcomes of $A = V_T \cup \{c\}$; see Figure 1.

The pair $(g(\mathcal{G}), u)$ is called the *normal form* of a positional game $(\mathcal{G}, u)$.

### 1.3 On Nash-solvability of Positional Game Forms

In [3], Nash-solvability of positional game forms was considered for case (i). An explicit characterization of Nash-solvability was obtained for the two-person game forms whose digraphs are *bidirected*: $(v', v'') \in E$ if and only if $(v'', v') \in E$.

In [2], Nash-solvability of positional game forms was studied (for a more general class of payoff functions, so-called *additive or integral* payoffs, yet) with the following additional restriction:

(ii') The outcome $c$, infinite play, is ranked as the worst one by all players.

Under assumption (ii'), Nash-solvability was proven in three cases:

(a) Two-person games ($n = |I| = 2$);
(b) Games with at most three outcomes ($|A| \leq 3$);
(c) Play-once games: each player controls only one position ($|V_i| = 1 \; \forall \; i \in I$).

However, it was also conjectured in [2] that Nash-solvability holds in general.

*Conjecture 1.* ([2]) A positional game is Nash-solvabile whenever (ii') holds.

This Conjecture would be implied by the following statement: *every im-cycle* $\mathcal{X} = \{x^1, \ldots, x^k\} \subseteq X$ *contains a strategy profile* $x^j$ *such that the corresponding play* $p(x^j)$ *is infinite.*

Indeed, Conjecture 1 would follow, since outcome $c \in A$ being the worst for all players, belongs to no im-cycle. However, the example of Section 2.2 will show that such an approach fails. Nevertheless, Conjecture 1 is not disproved. Moreover, a stronger conjecture was recently suggested by Gimbert and Sørensen, [5]. They assumed that, in case of terminal payoffs, condition (ii') is not needed.

*Conjecture 2.* A positional game is Nash-solvable if all cycles are one outcome.

They gave a simple and elegant proof for the two-person case. With their permission, we reproduce it in Section 5.

### 1.4 Restricted Improvement Cycles and Acyclicity

**Improvement Cycles in Trees.** Kukushkin [9,10] was the first to consider im-cycles in positional games. He restricted himself to trees and observed that even in this case im-cycles can exist; see example in Figure 1.

However, it is easy to see that unnecessary changes of strategies take place in this im-cycle. For example, let us consider transition from $x^1 = (x_1^1, x_2^2)$ to $x^2 = (x_1^1, x_2^3)$. Player 1 keeps his strategy $x_1^1$, while 2 substitutes $x_2^3$ for $x_2^2$ and gets a profit, since $g(x_1^1, x_2^2) = a_1$, $g(x_1^1, x_2^3) = a_2$, and $u_2(a_1) < u_2(a_2)$.

Yet, player 2 switches simultaneously from $a_4$ to $a_3$. Obviously, this cannot serve any practical purpose, since the strategy is changed outside the actual play.

In [9], Kukushkin also introduced the concept of *restricted improvements* (ri). In particular, he proved that positional games on trees become ri-acyclic if players are not allowed to change their decisions outside the actual play.

Since we consider arbitrary finite digraphs (not only trees), let us define accurately several types of restrictions for this more general case. The restriction considered by Kukushkin is what we call the *inside play restriction.*

**Inside Play Restriction.** Given a positional game form $\mathcal{G} = (G, D, v_0)$ and strategy profile $x^0 = (x_i^0, \, i \in I) \in X$, let us consider the corresponding play $p_0 = p(x^0)$ and outcome $a_0 = a(x^0) \in A$. This outcome is either a terminal, $a_0 \in V_T$, or a cycle, $a_0 = c$.

Let us consider the strategy $x_i^0$ of a player $i \in I$. He is allowed to change his decision in any position $v_1$ from $p_0$. This change will result in a new strategy profile $x^1$, play $p_1 = p(x^1)$, and outcome $a_1 = a(x^1) \in A$.

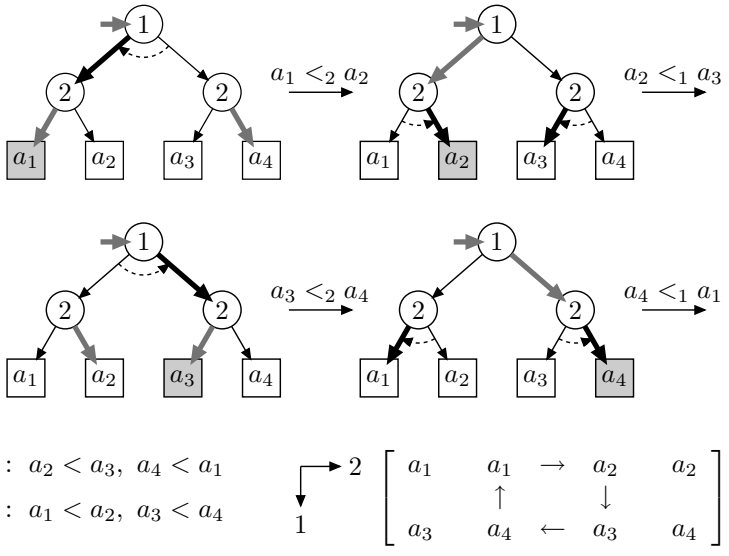



**Fig. 1.** Im-cycle in a tree. Bold arrows indicate chosen moves, and the black ones have changed from the previous strategy profile. The induced preference relations are shown on the left. The matrix on the right shows the normal form of the game.

Then, player $i$ may proceed, changing his strategy further. Now, he is only allowed to change the decision in any position $v_2$ that is located *after* $v_1$ in $p_1$, etc., until a position $v_m$, strategy profile $x^m$, play $p_m = p(x^m)$, and outcome $a_m = a(x^m) \in A$ appears; see Figure 2, where $m = 3$.

Equivalently, we can say that all positions $v_1, \ldots, v_m$ belong to one play.

Note that, by construction, obtained plays $\{p_0, p_1, \ldots, p_m\}$ are pairwise distinct. In contrast, the corresponding outcomes $\{a_0, a_1, \ldots, a_m\}$ can coincide and some of them might be the infinite play outcome $c \in A$.

Whenever the acting player $i$ substitutes the strategy $x_i^m$, defined above, for the original strategy $x_i^0$, we say that this is an *inside play deviation*, or in other words, that this change of decision in $x$ satisfies the *inside play restriction*.

It is easy, but important, to notice that this restriction, in fact, does not limit the power of a player. More precisely, if a player $i$ can reach an outcome $a_m$ from $x$ by a deviation then $i$ can also reach $a_m$ by an inside play deviation.

From now on, we will consider only such *inside play restricted* deviations and, in particular, only restricted improvements (ri) and talk about ri-cycles and ri-acyclicity rather than im-cycles and im-acyclicity, respectively.

**Types of Improvements.** We define the following four types of improvements:

**Standard improvement (or just improvement):** $u_i(a_m) > u_i(a_0)$;
**Strong improvement:** $u_i(a_m) > u_i(a_j)$ for $j = 0, 1, \ldots, m-1$;
**Last step improvement:** $u_i(a_m) > u_i(a_{m-1})$;

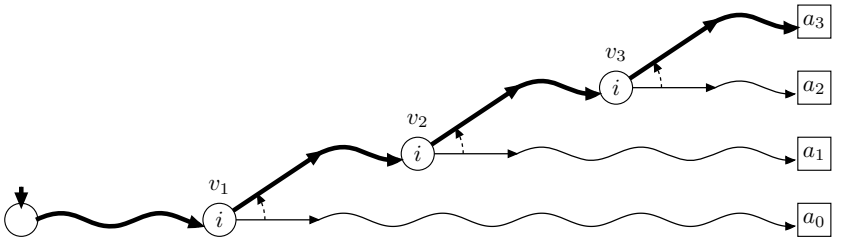



**Fig. 2.** Inside play restriction

**Best reply (BR) improvement:** $a_m$ is the best outcome that player $i$ can reach from $x$ (as we already noticed above, the inside play restriction does not restrict the set of reachable outcomes).

Obviously, each best reply or strong improvement is a standard improvement, and a strong improvement is also a last step improvement. Furthermore, it is easy to verify that no other containments hold between the above four classes. For example, a last step improvement might not be an improvement and vice versa.

We will consider ri-cycles and ri-acyclicity specifying in each case a type of improvement from the above list.

Let us note that any type of ri-acyclicity still implies Nash-solvability.

Indeed, if a positional game has no NE then for every strategy profile $x \in X$ there is a player $i \in I$ who can improve $x$ to some other profile $x' \in X$. In particular, $i$ can always choose a strong BR restricted improvement. Since we consider only finite games, such an iterative procedure will result in a strong BR ri-cycle. Equivalently, if we assume that there is no such cycle then the considered game is Nash-solvable; in other words, already strong BR ri-acyclicity implies Nash-solvability.

### 1.5 Sufficient Conditions for Ri-acyclicity

We start with Kukushkin's result for trees.

**Theorem 1.** *([9]). Positional games on trees have no restricted standard improvement cycles.*

After trees, it seems natural to consider acyclic digraphs. We asked Kukushkin [11] whether he had a generalization of Theorem 1 for this case. He had not considered it yet, but shortly thereafter produced a result that can be modified as follows.

**Theorem 2.** *Positional games on acyclic digraphs have no restricted last step improvement cycles.*

Let us notice that Theorem 1 does not result immediately from Theorem 2, since a standard improvement might be not a last step improvement.

Finally, in case of two players the following statement holds.

**Theorem 3.** *Two-person positional games have no restricted strong improvement cycles.*

Obviously, Theorem 3 implies Nash-solvability of two-person positional games; see Section 5 for an independent proof by Gimbert and Sørensen.

Theorems 2 and 3 are proved in Sections 3 and 4, respectively.

## 2 Examples of Ri-cycles; Limits of Theorems 1, 2, and 3

In this paper, we emphasize negative results showing that it is unlikely to strengthen one of the above theorems or obtain other criteria of ri-acyclicity.

### 2.1 Example Limiting Theorems 2 and 3

For both Theorems 2 and 3, the specified type of improvement is essential. Indeed, the example in Figure 3 shows that a two-person game on an acyclic digraph can have a ri-cycle. However, it is not difficult to see that in this ri-cycle, not all improvements are strong and some are not even last step improvements.

Thus, all conditions of Theorems 2 and 3 are essential.

Furthermore, we note that if in Theorem 3 we substitute BR improvement for strong improvement, the modified statement will not hold, see Figure 4.

By definition, every change of strategy must result in an improvement for the corresponding player. Hence, each such change implies an ordering of the two outcomes; in our figures, it appears as a label on the transition arrow between situations. An entire im-cycle implies a set of inequalities, which must be satisfiable in order to allow a consistent preference profile. Note that it is also sufficient to allow ties and have a partial ordering of the outcomes.

### 2.2 On *c*-free Ri-cycles

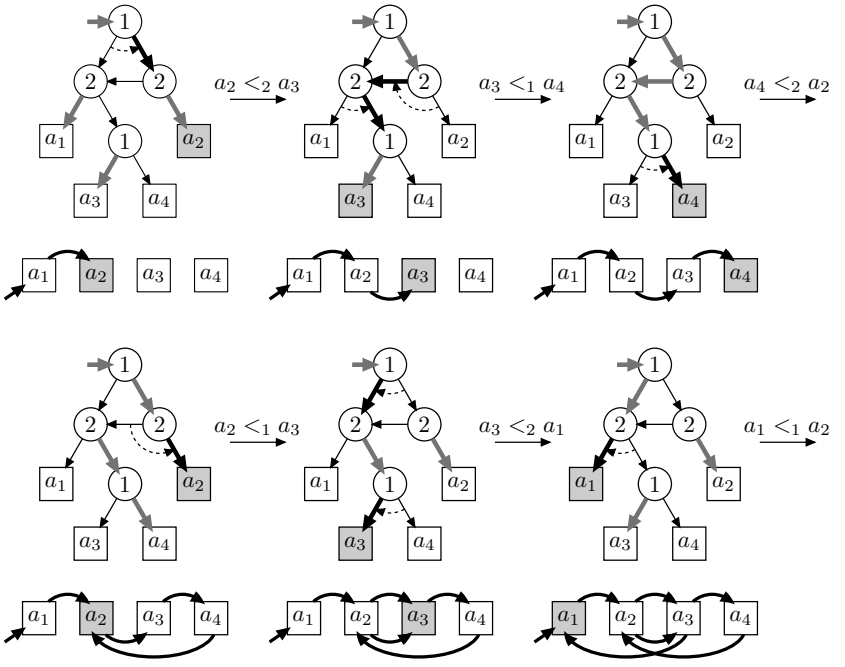
In Section 1.3, we demonstrated that Conjecture 1 on Nash-solvability would result from the following statement: **(i)** There are no *c*-free im-cycles.

Of course, (i) fails. As we know, im-cycles exist already in trees; see Figure 1.

However, let us substitute (i) by the similar but much weaker statement:
**(ii)** Every restricted strong BR ri-cycle contains a strategy profile whose outcome is infinite play.

One can derive Conjecture 1 from (ii), as well as from (i). Yet, (ii) also fails. Indeed, let us consider the ri-cycle in Figure 5. This game is play-once; each player controls only one position. Moreover, there are only two possible moves in each position. For this reason, every ri-cycle in this game is BR and strong.

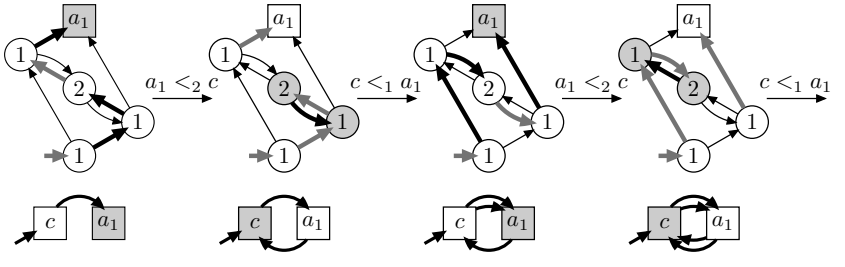
There are seven players ($n = 7$) in this example, yet, by teaming up players in coalitions we can reduce the number of players to four while the improvements remain BR and strong. Indeed, this can be done by forming three coalitions $\{1, 7\}, \{3, 5\}, \{4, 6\}$ and merging the preferences of the coalitionists. The required extra constraints on the preferences of the coalitions are also shown in Figure 5.




1 : $a_1 < a_2 < a_3 < a_4$

2 : $a_4 < a_2 < a_3 < a_1$

**Fig. 3.** 2-person ri-cycle in an acyclic digraph. Beneath each situation is a graph of outcomes with edges defined by the previous improvement steps; these will be of illustrative importance in the proof of Theorem 3.




1 : $c < a_1$

2 : $a_1 < c$

**Fig. 4.** 2-person BR ri-cycle in graph with cycles

It is easy to see that inconsistent (i.e., cyclic) preferences appear whenever any three players form a coalition. Hence, the number of coalitions cannot be reduced below 4, and it is, in fact, not possible to form 4 coalitions in any other way while keeping improvements BR and strong.

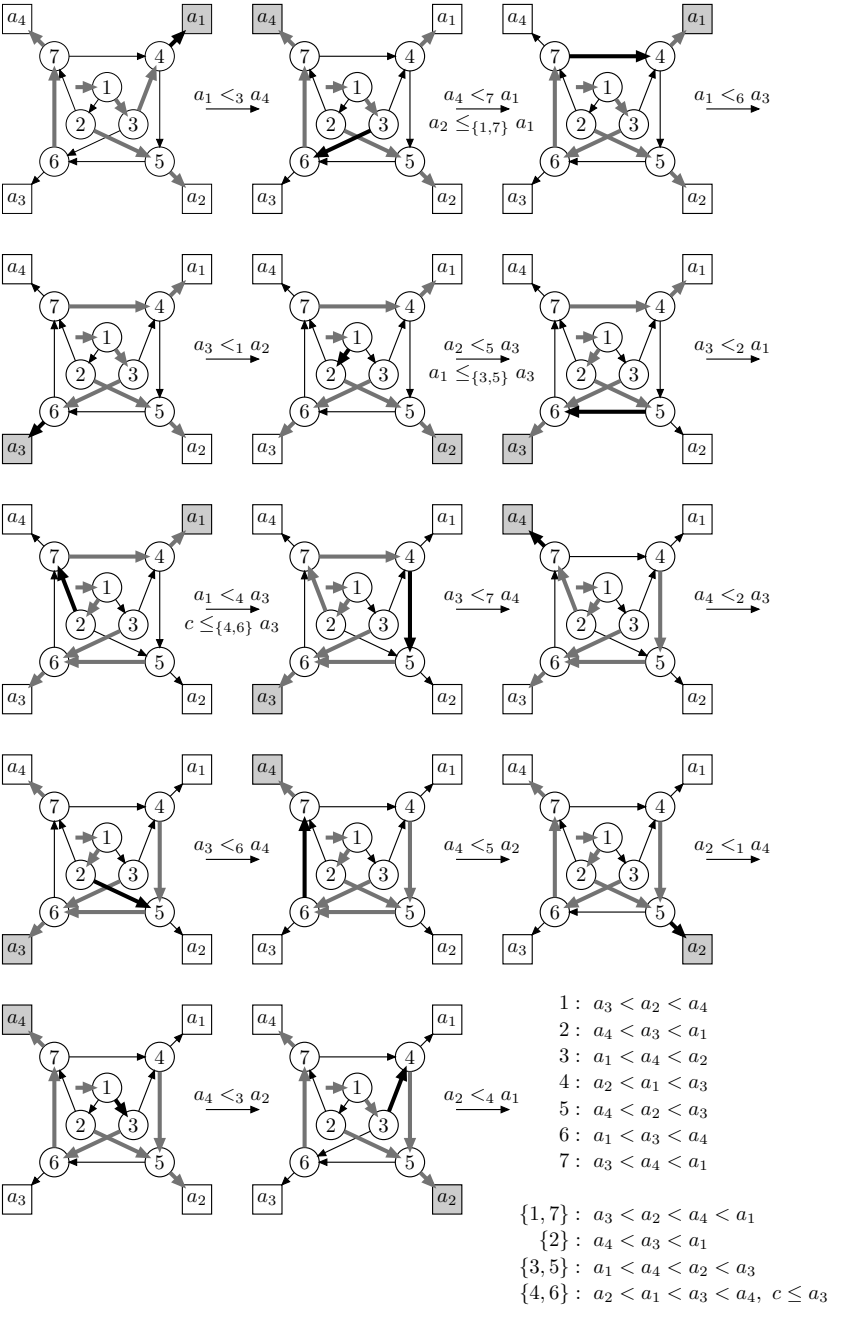
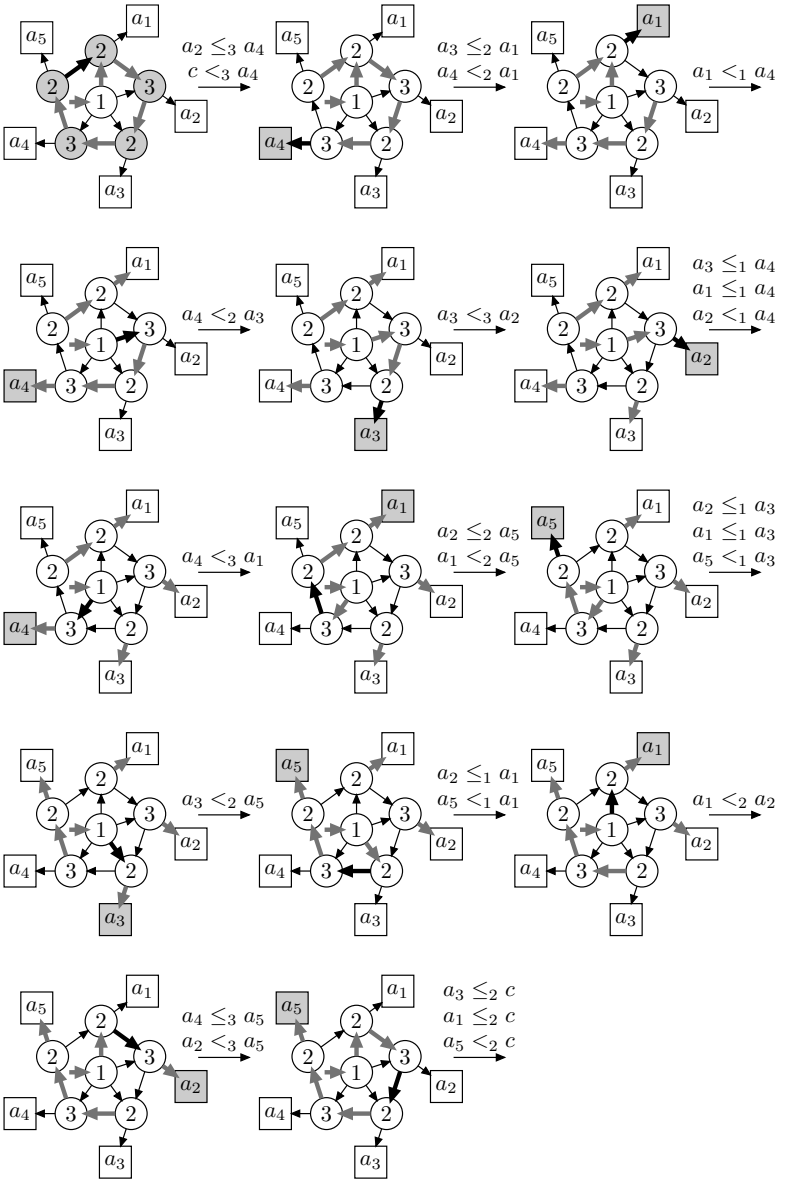



**Fig. 5.** $c$-free strong BR ri-cycle




Feasible total order:

1 : $a_5 < a_2 < a_1 < a_3 < a_4 < c$

2 : $a_4 < a_3 < a_1 < a_2 < a_5 < c$

3 : $c < a_3 < a_2 < a_4 < a_1 < a_5$

**Fig. 6.** Strong BR ri-cycle containing $c$ in a 3-person positional game

Obviously, for the two-person case, **(ii)** follows from Theorem 3.

*Remark 3.* We should confess that our original motivation fails. It is hardly possible to derive new results on Nash-solvability from ri-acyclicity. Although, ri-acyclicity is much weaker than im-acyclicity, it is still too much stronger than Nash-solvability. In general, by Theorems 3 and 2, ri-acyclicity holds for $n = 2$ and for acyclic digraphs. Yet, for these two cases Nash-solvability is known.

It is still possible that (ii) (and, hence, Conjecture 1) holds for $n = 3$, too. Strong BR ri-cycles in which the infinite play outcome $c$ occurs do exist for $n = 3$, however. Such an example is provided in Figure 6.

However, ri-acyclicity is of independent (of Nash-solvability) interest. In this paper, we study ri-acyclicity for the case when each terminal is a separate outcome, while all directed cycles form one special outcome. For the alternative case, when each terminal and each directed cycle is a separate outcome, Nash-solvability was considered in [3], while ri-acyclicity was never studied.

## 3 Proof of Theorem 2

Given a positional game $(\mathcal{G}, u) = (G, D, v_0, u)$ whose digraph $G = (V, E)$ is acyclic, let us order positions of $V$ so that $v < v'$ whenever there is a directed path from $v$ to $v'$. To do so, let us assign to each position $v \in V$ the length of a longest path from $v_0$ to $v$ and then order arbitrarily positions with equal numbers.

Given a strategy profile $x$, let us, for every $i \in I$, assign to each position $v \in V_i$ the outcome $a(v, x)$ which $x$ would result in starting from $v$ and the number $u_i(a(v, x))$. These numbers form a $|V \setminus V_T|$-dimensional vector $y(x)$ whose coordinates are assigned to positions $v \in V \setminus V_T$. Since these positions are ordered, we can introduce the inverse lexicographic order over such vectors $y$.

Let a player $i \in I$ choose a last step ri-deviation $x'_i$ from $x_i$. Then, $y(x') > y(x)$, since the last changed coordinate increased: $u_i(a_k) > u_i(a_{k-1})$. Hence, no last step ri-cycle can exist. □

## 4 Proof of Theorem 3

Let us consider a two-person positional game $\mathcal{G} = (G, D, v_0, u)$ and a strategy profile $x$ such that in the resulting play $p = p(x)$ the terminal move $(v, a)$ belongs to a player $i \in I$. Then, a strong improvement $x'_i$ results in a terminal $a' = p(x')$ such that $u_i(a') > u_i(a)$. (This holds for $n$-person games, as well.)

Given a *strong* ri-cycle $\mathcal{X} = \{x^1, \ldots, x^k\} \in X$, let us assume, without any loss of generality, that the game $(G, D, v_0, u)$ is *minimal* with respect to $\mathcal{X}$, that is, the ri-cycle $\mathcal{X}$ is broken by eliminating any move from $\mathcal{G}$. Furthermore, let $\mathcal{A}(\mathcal{X})$ denote the set of the corresponding outcomes: $A(\mathcal{X}) = \{a(x^j),\ j = 1, \ldots, k\}$. Note that several of the outcomes of the strategy profiles may be the same and that $A(\mathcal{X})$ may contain $c \in A$ (infinite play).

Let us introduce the directed multigraph $\mathcal{E} = \mathcal{E}(\mathcal{X})$ whose vertex-set is $A(\mathcal{X})$ and the directed edges are $k$ pairs $(a_j, a_{j+1})$, where $a_j = a(x^j),\ \ j = 1, \ldots, k$,

and $k+1=1$. It is easy to see that $\mathcal{E}$ is *Eulerian*, i.e., $\mathcal{E}$ is strongly connected and for each vertex its in-degree and out-degree are equal. An example of this construction is shown in Figure 3.

Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be the subgraphs of $\mathcal{E}$ induced by the edges corresponding to deviations of players 1 and 2, respectively. Then, $\mathcal{E}_1$ and $\mathcal{E}_2$ are acyclic, since a cycle would imply an inconsistent preference relation. In the example of Figure 3, the edges are partitioned accordingly (above and below the vertices), and the subgraphs are indeed acyclic.

Hence, there is a vertex $a^1$ whose out-degree in $\mathcal{E}_1$ and in-degree in $\mathcal{E}_2$ both equal 0. In fact, an outcome $a^1 \in A(\mathcal{X})$ most preferred by player 1 must have this property. (We do not exclude ties in preferences; if there are several best outcomes of player 1 then $a^1$ can be any of them.) Similarly, we define a vertex $a^2$ whose in-degree in $\mathcal{E}_1$ and out-degree in $\mathcal{E}_2$ both equal 0.

Let us remark that either $a^1$ or $a^2$ might be equal to $c$, yet, not both. Thus, without loss of generality, we can assume that $a^1$ is a terminal outcome.

Obviously, a player, 1 or 2, has a move to $a^1$. If 1 has such a move then it cannot be improved in $\mathcal{X}$, since $u_1(a_j) \le u_1(a^1)$ for all $j=1,\dots,k$ and $\mathcal{X}$ is a strong ri-cycle. Let us also recall that $a^1$ has no incoming edges in $\mathcal{E}_2$. Hence, in $\mathcal{X}$, player 2 never makes an improvement that results in $a^1$. In other words, a player who has a move to $a^1$ will make it either always, player 1, or never, player 2. In both cases we obtain a contradiction with the minimality of digraph $\mathcal{G}$. □

## 5 Nash-Solvability of Two-Person Game Forms

If $n=2$ and $c \in A$ is the worst outcome for both players, Nash-solvability was proven in [2]. In fact, the last assumption is not necessary: even if outcome $c$ is ranked by two players arbitrarily, Nash-solvability still holds. This observation was recently made by Gimbert and Sørensen.

A two-person game form $g$ is called:

*Nash-solvable* if for every utility function $u : \{1,2\} \times A \to \mathbb{R}$ the obtained game $(g,u)$ has a Nash equilibrium.

*zero-sum-solvable* if for each zero-sum utility function $(u_1(a)+u_2(a)=0$ for all $a \in A)$ the obtained zero-sum game $(g,u)$ has a Nash equilibrium, which is called a saddle point for zero-sum games.

$\pm$*-solvable* if zero-sum solvability holds for each $u$ that takes only values: $+1$ and $-1$.

Necessary and sufficient conditions for zero-sum solvability were obtained by Edmonds and Fulkerson [4] in 1970; see also [6]. Somewhat surprisingly, these conditions remain necessary and sufficient for $\pm$-solvability and for Nash-solvability, as well; in other words, all three above types of solvability are equivalent, in case of two-person game forms [7]; see also [8] and Appendix 1 of [3].

**Proposition 1.** *([5]). Each two-person positional game form in which all cycles form one outcome is Nash-solvable.*

*Proof.* Let $\mathcal{G}=(G,D,v_0,u)$ be a two-person zero-sum positional game, where $u : I \times A \to \{-1,+1\}$ is a zero-sum $\pm 1$ utility function. Let $A_i \subseteq A$ denote

the outcomes winning for player $i \in I = \{1, 2\}$. Without any loss of generality we can assume that $c \in A_1$, that is, $u_1(c) = 1$, while $u_2(c) = -1$. Let $V^2 \subseteq V$ denote the set of positions in which player 2 can enforce a terminal from $A_2$. Then, obviously, player 2 wins whenever $v_0 \in V^2$. Let us prove that player 1 wins otherwise, when $v_0 \in V^1 = V \setminus V^2$.

Indeed, if $v \in V^1 \cap V_2$ then $v' \in V^1$ for every move $(v, v')$ of player 2; if $v \in V^1 \cap V_1$ then player 1 has a move $(v, v')$ such that $v' \in V_1$. Let player 1 choose such a move for every position $v \in V^1 \cap V_1$ and an arbitrary move in each remaining position $v \in V^2 \cap V_1$. This rule defines a strategy $x_1$. Let us fix an arbitrary strategy $x_2$ of player 2 and consider the profile $x = (x_1, x_2)$. Obviously, the play $p(x)$ cannot come to $V_2$ if $v_0 \in V_1$. Hence, for the outcome $a = a(x)$ we have: either $a \in V^1$ or $a = c$. In both cases player 1 wins. Thus, the game is Nash-solvable. □

Let us recall that this result also follows immediately from Theorem 3.

Finally, let us briefly consider a refinement of the Nash equilibrium concept, the so-called *subgame perfect* equilibrium, where a strategy profile is an equilibrium regardless of the choice of starting position.

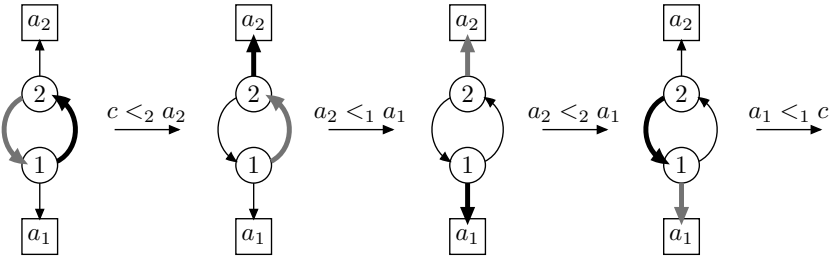



**Fig. 7.** Two-person game with no subgame perfect positional strategies. The improvements do not obey any inside play restriction, since there is no fixed starting position.

It is not difficult to see that already for two-person games a Nash equilibrium can be unique but not subgame perfect. Let us consider the example in Figure 7. There are only four different strategy profiles and for all of them there is a choice of starting position for which the profile is not an equilibrium.

## 6 Conclusions and Open Problems

Nash-solvability of $n$-person positional games (in which all directed cycles form a single outcome) holds for $n = 2$ and remains an open problem for $n > 2$. For $n = 2$, we prove strong ri-acyclicity, which implies Nash-solvability. Computing Nash equilibria efficiently is another interesting issue for further investigation.

For $n \geq 4$ there are examples of best reply strong $c$-free ri-cycles. Yet, it remains open whether such $c$-free examples exist for $n = 3$.

**Acknowledgements.** We are thankful to Gimbert, Kukushkin, and Sørensen for helpful discussions.

## References

1. Andersson, D., Gurvich, V., Hansen, T.D.: On acyclicity of games with cycles. DIMACS Technical Report 2009-09, Rutgers University
2. Boros, E., Gurvich, V.: On Nash-solvability in pure strategies of finite games with perfect information which have cycles. Math. Social Sciences 46 (2003)
3. Boros, E., Gurvich, V., Makino, K., Shao, W.: Nash-solvable bidirected cyclic two-person game forms, Rutcor Research Report 26-2007 and DIMACS Technical Report 2008-13, Rutgers University
4. Edmonds, J., Fulkerson, D.R.: Bottleneck Extrema, RM-5375-PR, The Rand Corporation, Santa Monica, Ca. (January 1968); J. Combin. Theory 8, 299–306 (1970)
5. Gimbert, H., Sørensen, T.B.: Private communications (July 2008)
6. Gurvich, V.: To theory of multi-step games USSR Comput. Math. and Math. Phys. 13(6), 143–161 (1973)
7. Gurvich, V.: Solution of positional games in pure strategies. USSR Comput. Math. and Math. Phys. 15(2), 74–87 (1975)
8. Gurvich, V.: Equilibrium in pure strategies. Soviet Mathematics Doklady 38(3), 597–602 (1988)
9. Kukushkin, N.S.: Perfect information and potential games. Games and Economic Behavior 38, 306–317 (2002)
10. Kukushkin, N.S.: Acyclicity of improvements in finite game forms (manuscript, 2009), `http://www.ccas.ru/mmes/mmeda/ququ/GF.pdf`
11. Kukushkin, N.S.: Private communications (August 2008)
12. Zermelo, E.: Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels. In: Proc. 5th Int. Cong. Math. Cambridge 1912, vol. II, pp. 501–504. Cambridge University Press, Cambridge (1913)

# Discrete Online TSP*

Mauro Aprea, Esteban Feuerstein, Gustavo Sadovoy,
and Alejandro Strejilevich de Loma

Departamento de Computación, Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires, Pabellón I, Ciudad Universitaria,
(1428) Capital Federal, Argentina

**Abstract.** In this paper we introduce a discrete version of the online traveling salesman problem (DOLTSP). We represent the metric space using a weighted graph, where the server is allowed to modify its route only at the vertices. This limitation directly affects the capacity of the server to react and increases the risk related to each decision. We prove lower bounds on the performance of deterministic online algorithms in different scenarios of DOLTSP, and we present distinct algorithms for the problem, some of them achieving the best possible performance. We measure the performance of the algorithms using competitive analysis, the most widely accepted method for evaluating online algorithms. Besides, we perform an empirical simulation on paths, generating a significant set of instances and measuring the quality of the solutions given by each algorithm. Our experiments show that algorithms with the best competitive ratio do not have the best performance in practice.

**Keywords:** TSP, discrete metric spaces, online algorithms.

## 1 Introduction

Numerous variations of the Vehicle Routing Problem (VRP) have been defined [1,2,3,4]. Many of these variations assume that the input is completely known when the solution is computed. However, there are many situations in which decisions must be made based on partial information, and the solution must be built or even executed before the input is completely known, what is usually known as *online optimization* [5,6]. Think for example of a salesman with a cellular phone, or a fleet of vehicles equipped with radios that must collect and deliver packages at different locations, and many other transportation problems in which the itinerary can be modified during its execution. In the online versions of VRPs, a sequence of requests is posed to an algorithm that has to decide how to move the servers to satisfy the requests without knowledge of future requests.

* Research supported in part by UBACyT projects X143 and X212, and by ANPCyT project PICT-2006-01600.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 29–42, 2009.

Previous works about online VRPs [7,8,9,10,1,11] have considered that the servers move in a *continuous* metric space. In this scenario the servers can change direction at any time while they are moving from one point to another. However, in some applications this may not be the case, and it is preferable to model the problem using a *discrete* metric space. For example, if a server is on a road network of freeways and a request arrives while the server is moving between two exits, the server has to proceed to the next exit before being able to change its plan. Online routing on discrete metric spaces appears a priori to be "harder" than the continuous counterpart, as in the former there are less opportunities to revise the plan, and thus the risk associated to each decision may be higher.

In this paper we consider the online version of the Traveling Salesman Problem (TSP) on discrete metric spaces. We call this variant *Discrete Online Traveling Salesman Problem* (DOLTSP). In an instance of DOLTSP, a server that travels at unit speed must visit the vertices of a graph in order to satisfy requests that are presented along time, finishing the service as early as possible. As in previous works about online routing, we consider two versions of the problem: one *Homing* (HDOLTSP), in which the journey of the server must finish at the same vertex where it starts, and one *Nomadic* (NDOLTSP), in which it can finish anywhere in the graph. Note that the associated offline problems are the same as in the continuous case, namely the *Vehicle Routing Problem with release times* [12].

We propose deterministic online algorithms for the two versions of DOLTSP, and we measure their performance using *competitive analysis* [13]. In this widely used framework the cost of an algorithm is compared to that of an *optimal offline adversary* that knows the whole input in advance. We consider two types of adversaries: a *standard* adversary with unrestricted power, and a *fair* adversary [10] that must keep the server inside the region where requests have already been presented. We analyze two classes of online algorithms: *zealous* algorithms [10] that keep working while there is something to do, and *cautious* algorithms that may halt the server even when there is pending work.

Most of our results hold on any (non trivial) graph. However, sometimes we focus our attention on *paths*, i.e. graphs in which the server can only move in two opposite directions which we refer as "left" (or negative) and "right" (or positive). Moreover, certain results are only valid for *halfpaths*, that are paths where the starting vertex is the leftmost one. Notice that paths and halfpaths are the discrete analogons of the real line and halfline, respectively. This family of graphs allows to capture many interesting applications, like the previously mentioned of a highway, an elevator, a stacker-crane moving on a track, or the radial movement of the read/write head in a hard disc drive.

A summary of our theoretical results is given in Table 1, where lower bounds that hold on any graph appear in boldface. The two lower bounds not in boldface are valid on halfpaths with a certain distribution of vertices (and then also on paths and trees). We found that, in general, DOLTSP is harder than its continuous counterpart. As expected, HDOLTSP is easier than NDOLTSP, the

**Table 1.** Summary of theoretical results

| | | zealous | | | cautious | | |
|---|---|---|---|---|---|---|---|
| | | lower | upper bound | | lower | upper bound | |
| | | bound | path | general | bound | halfpath | general |
| Homing | fair | **2** | 2 | 3 | $\approx$ **1.618** | $\approx$ 1.618 | $\approx$ 2.618 [8] |
| | standard | **2** | 2 | 3 | $\approx$ 1.707 | $\approx$ 1.707 | $\approx$ 2.618 [8] |
| Nomadic | fair | **3** | 3 | 3 | $\approx$ **1.839** | 2 | (unknown) |
| | standard | **3** | 3 | 3 | 2 | 2 | (unknown) |

fair adversary is weaker than the standard one, and zealous algorithms are weaker than cautious ones.[1]

Besides, we perform empirical simulations on paths. We find that in practice the zealous strategies we devised are better than the cautious ones. Nevertheless, the empirical studies also ratify the idea that waiting is profitable in a worst case sense.

## 2 Basic Definitions and Notation

### 2.1 DOLTSP

The input of DOLTSP consists of a graph $G = (V, E)$ with a positive length associated to each edge $e \in E$, a distinguished vertex $o \in V$ (the *origin*), and a sequence $\sigma$ of requests $r_i = (t_i, v_i)$, where $v_i \in V$, and $t_i \in \mathbb{R}_{\geq 0}$ is a *release time* representing the moment at which $r_i$ is presented. These moments form an ordered sequence in the sense that $t_i \leq t_j$ if $i < j$. At time 0 a server is located at the origin $o$, and must serve all requests. With this purpose it has to move through the edges of $E$ at unit speed and visit each vertex $v_i$ at some moment not earlier than $t_i$. The server cannot change direction while traversing and edge. We consider two variants of DOLTSP: in *Homing* DOLTSP (HDOLTSP) the server must return to the origin after serving all requests, while in *Nomadic* DOLTSP (NDOLTSP) the journey can finish anywhere.

For every pair of vertices $v, w \in V$ we denote with $d(v, w) = d(w, v)$ the distance between them, that is, the length of a shortest path joining them. We denote with $\bar{v}$ the distance of $v$ to the origin $o$. We assume that the graph $G$ is not trivial (it has at least one vertex apart from $o$), and that all lengths associated to edges satisfy the triangle inequality.

[1] In the Online Asymmetric Traveling Salesman Problem (OL-ATSP) a server moves in a not necessarily symmetric space [8]. This problem can be viewed as a generalization of DOLTSP, by replacing each edge of the graph by two directed arcs of the same length, and considering a particular notion of distance (the cost of changing direction while traversing an arc from $x$ to $y$, is the cost of reaching $y$ plus the cost of going back to $x$). Thus, the algorithms presented in [8] mantain their performance when they are used for DOLTSP. However, we obtain strictly better performances on paths and for NDOLTSP.

An algorithm for DOLTSP must decide the movements of the server with the goal of ending its work as soon as possible. An *online* algorithm has to execute each movement without knowledge of unreleased requests. On the contrary, an *offline* algorithm can decide based on the whole sequence of requests.

### 2.2 Competitive Analysis and Adversaries

*Competitive analysis* [13,14] is a type of worst case analysis where the performance of an algorithm for a problem is compared to that of an optimal offline algorithm. The measure of performance used in competitive analysis is the *competitive ratio*. We say that an algorithm `ALG` for DOLTSP is $\rho$-competitive if and only if for every sequence of request $\sigma$ we have $C_{\mathrm{ALG}}(\sigma) \leq \rho \cdot C_{\mathrm{OPT}}(\sigma)$, where $C_{\mathrm{ALG}}(\sigma)$ is the cost of `ALG` for $\sigma$, and $C_{\mathrm{OPT}}(\sigma)$ is the cost of an optimal offline algorithm that knows the whole input sequence in advance.

It is usually useful to see competitive analysis as a game between an *online player* and an *offline adversary*. The former tries to find a good solution for a sequence of requests generated by the latter, who knows the online strategy and tries to maximize the ratio between both costs. Thus, we use the terms *optimal offline algorithm* and *adversary* interchangeably.

Competitive analysis is sometimes criticized for its excessive pessimism [15]. With the aim of attenuating this situation, different alternative measures have been proposed. One of them is known as *comparative analysis*, in which the adversary is restricted in some sense. For online routing problems, a form of comparative analysis consists of using, instead of a *standard*, unrestricted adversary, a *fair* adversary that is required to move the server inside the region where requests have already been presented. Fair adversaries have been originally proposed in [10] for (continuous) OLTSP. For DOLTSP we define this class of adversaries as follows. At any given moment $t$, the *fair region* is the closure under shortest paths of the set of vertices formed by the origin and the vertices where requests have been presented. An adversary is *fair* if at every moment its server is in the subgraph induced by the fair region at that moment.

## 3 Zealous Algorithms

In this section we propose and analyze simple and intuitive online algorithms for DOLTSP. As we will see, these algorithms are members of a natural class of algorithms that keep working as long as there is something to do.

The first online algorithm is known as Replan (`REP`). This algorithm was well studied in the context of distinct online optimization problems, and it consists of adjusting the solution each time a new request is presented. In the case of DOLTSP, this means computing a new itinerary that allows the server to satisfy all pending requests in the least possible time. If new requests are presented when the server is not at a vertex of the graph, the new route is computed as soon as the server reaches a vertex.

We will analyze the performance of `REP` for the particular case of paths. For doing that, we need to introduce the concept of *extreme* vertices.

**Definition 1.** *Given an instance of DOLTSP on a path, at any given moment consider the set $S$ of vertices that* `REP` *has yet to visit. This set contains the vertices of unserved requests, and in HDOLTSP includes also the origin. We call* left (resp. right) extreme *the leftmost (resp. rightmost) vertex of $S$.*

Note that for HDOLTSP on paths, the left (resp. right) extreme is located in the left (resp. right) halfpath. However, one or both extremes could be the origin. This implies that at least one of the extremes is in the same halfpath where the server is. Among the extremes that are in the same halfpath that the server, let $F$ be the extreme that is farthest from the origin. It is easy to see that for HDOLTSP on paths the route computed by `REP` is as follows: move to extreme $F$, then to the other extreme, and finally to the origin.

In NDOLTSP we cannot assume any order between the extremes and the origin. However, the route computed by `REP` is simpler: move to the extreme that is nearest to the server, and then to the other extreme.

Knowing the routes computed by `REP` for DOLTSP on paths, we are ready to analyze its competitiveness. The following results show that `REP` is 2-competitive for HDOLTSP and 3-competitive for NDOLTSP. This is valid on any path (even a halfpath) against both fair and standard adversaries.

**Theorem 2.** *Algorithm* `REP` *is 2-competitive for HDOLTSP on any path against both fair and standard adversaries.*

*Proof.* Let $\sigma$ be any sequence of requests. Let $L$ and $R$ be respectively the leftmost and rightmost vertices that must be visited to serve all the requests of $\sigma$, including the origin. Clearly, at any moment, the left and right extremes are located between $L$ and $R$. More precisely, the left extreme is between $L$ and the origin, and the right extreme is between the origin and $R$. Besides, the server of `REP` is always between $L$ and $R$, because it only moves to serve a request or to return to the origin. Let $T$ be the moment in which the last request of $\sigma$ is presented. Two situations can occur.

1. At time $T$, the online server is at a vertex or traversing an edge that moves it away from the origin. To complete its work, the server will first move to an extreme, then to the other extreme, and finally to the origin, ending at most at time $T + 2\bar{L} + 2\bar{R}$. Since $T$ and $2\bar{L} + 2\bar{R}$ are lower bounds to the optimal offline cost, we have
$$\frac{C_{\mathrm{REP}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \leq \frac{T}{C_{\mathrm{OPT}}(\sigma)} + \frac{2\bar{L} + 2\bar{R}}{C_{\mathrm{OPT}}(\sigma)} \leq 2.$$
2. At time $T$, the online server is traversing an edge that brings it nearer to the origin. Let $v \neq o$ be the vertex where that move starts. We know that $v$ is between $L$ and $R$. There are two possibilities.
   (a) After the server leaves $v$, no new request is presented at $v$ or at any other vertex in the same halfpath farther away from the origin than $v$. As in the previous situation, the server will first move to an extreme, then to the other extreme, and finally to the origin, with a total cost of at most $T + 2\bar{L} + 2\bar{R}$. And again, this is at most twice the optimal offline cost.

(b) The last situation is when at least one of those requests is presented. Let $r$ be one of them, presented at time $t$ at vertex $x$. Clearly, the length of the edge that the online server is traversing is at most $\bar{x}$ and, as we said, the movement started before time $t$. Then, before time $t + \bar{x}$ the server arrives to a vertex and can replan its tour, ending its job at most at time $t + \bar{x} + 2\bar{L} + 2\bar{R}$. Since the optimal offline cost is lower bounded by $t + \bar{x}$ and by $2\bar{L} + 2\bar{R}$, we obtain

$$\frac{C_{\mathrm{REP}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \leq \frac{t + \bar{x}}{C_{\mathrm{OPT}}(\sigma)} + \frac{2\bar{L} + 2\bar{R}}{C_{\mathrm{OPT}}(\sigma)} \leq 2. \qquad \square$$

**Theorem 3.** *Algorithm* REP *is* 3*-competitive for NDOLTSP on any path against both fair and standard adversaries.*

*Proof.* Let $\sigma$ be any sequence of requests. Let $L$, $R$ and $T$ be as in the previous proof, and note that also in NDOLTSP the server of REP is always between $L$ and $R$. Until time $T$ the cost of the algorithm is obviously $T$. Consider the route of the server from this moment on. If the server is traversing and edge, it completes the movement, and then the server moves to its nearest extreme, with a total cost of at most $d(L, R)$, that is, the distance between $L$ and $R$. To complete its job, the server moves to the other extreme, again with a cost of at most $d(L, R)$. Since $T$ and $d(L, R)$ are lower bounds for the optimal offline cost, we have

$$\frac{C_{\mathrm{REP}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \leq \frac{T}{C_{\mathrm{OPT}}(\sigma)} + \frac{2d(L, R)}{C_{\mathrm{OPT}}(\sigma)} \leq 3 \qquad \square$$

Our second online algorithm for DOLTSP is called Zig-Zag (ZZG), and it is defined only on paths. The algorithm repeatedly moves the server to the left and to the right while there are pending requests in each direction. In HDOLTSP, while there are no pending requests, ZZG moves the server towards the origin.

The following results show that ZZG can achieve the same competitive ratios we obtained for REP. The proofs are very similar to those of Theorem 2 and Theorem 3.

**Theorem 4.** *Algorithm* ZZG *is* 2*-competitive for HDOLTSP on any path against both fair and standard adversaries.*

**Theorem 5.** *Algorithm* ZZG *is* 3*-competitive for NDOLTSP on any path against both fair and standard adversaries.*

Our last online algorithm for DOLTSP is called Delayed Replan (DREP), and it is very similar to REP. The only difference is that when a new request is presented, DREP delays the computation of a new optimal tour until the server is at the origin or it has just served a request. This implies that at any given moment the server of DREP is on a shortest path from $x$ to $y$, with $x$ and $y$ being the origin or vertices where requests have been presented. According to the following results, DREP is 3-competitive on any graph, in all versions of DOLTSP.

**Theorem 6.** *Algorithm* DREP *is* 3*-competitive for HDOLTSP on any graph against both fair and standard adversaries.*

*Proof.* Let $\sigma$ be any sequence of requests, and let $T$ be the moment in which the last request is presented. WLOG, assume that at time $T$ the server of DREP is on a shortest path from $x$ to $y$, with $x$ and $y$ being the origin or vertices where requests have been presented. Once the server reaches $y$, it will follow an optimal tour that serves all the pending requests ending at the origin. Since the server can always go from $y$ to the origin and then follow an optimal offline tour for $\sigma$, the cost of the last tour of DREP is at most $\bar{y} + C_{\mathrm{OPT}}(\sigma)$. Besides, the server of any algorithm must visit $x$ and $y$, and return to the origin, which implies $\bar{y} + d(x, y) \leq C_{\mathrm{OPT}}(\sigma)$. Putting all the above things together we obtain

$$\frac{C_{\mathrm{DREP}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \leq \frac{T + d(x, y) + \bar{y} + C_{\mathrm{OPT}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \leq 3. \qquad \square$$

**Theorem 7.** *Algorithm* DREP *is* 3*-competitive for NDOLTSP on any graph against both fair and standard adversaries.*

*Proof.* Let $\sigma$, $T$, $x$ and $y$ be as in the previous proof. Let $a$ be the ending vertex of an optimal offline tour for $\sigma$. Once the server reaches $y$, it will follow an optimal tour that serves all the pending requests. Since the server can always go from $y$ to either the origin or $a$, and then follow an optimal offline tour for $\sigma$, the cost of the last tour of DREP is at most $\min(\bar{y}, d(y, a)) + C_{\mathrm{OPT}}(\sigma)$. Finally, the server of any algorithm must visit $x$ and $y$, being $d(x, y) + d(y, a) \leq C_{\mathrm{OPT}}(\sigma)$ if OPT goes first to $x$, and $\bar{y} + d(y, x) \leq C_{\mathrm{OPT}}(\sigma)$ otherwise. Therefore we have

$$\frac{C_{\mathrm{DREP}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \leq \frac{T + d(x, y) + [\min(\bar{y}, d(y, a)) + C_{\mathrm{OPT}}(\sigma)]}{C_{\mathrm{OPT}}(\sigma)} \leq 3. \qquad \square$$

Algorithms REP, ZZG and DREP are members of a very natural class of algorithms, namely *zealous* algorithms. The idea behind them is simple: whenever there is pending work, do it without wasting time. Zealous algorithms were introduced in [10] for (continuous) OLTSP. As mentioned in that work, a formal definition of zealous algorithms requires some care. This is particularly true for DOLTSP because the server can change direction only at the vertices.

**Definition 8.** *An online algorithm for DOLTSP is called* zealous *if and only if each time the server is at any vertex, the following conditions are met.*

1. *If there are pending requests, the server moves to serve one of them or to the origin using a shortest path.*
2. *In HDOLTSP, if there are no pending requests, the server moves to the origin using a shortest path.*
3. *If the server arrived to the vertex while traveling to another one, it can change the planned tour only if a new request has been presented after leaving the previous vertex in the route.*

Now we have a precise description of zealous online algorithms for DOLTSP, we are able to prove lower bounds on their competitiveness. Our lower bounds are valid on any graph against both fair and standard adversaries.

**Theorem 9.** *No zealous online algorithm for HDOLTSP on any graph is better than* 2*-competitive against neither fair nor standard adversaries.*

*Proof.* Let $x$ be the closest vertex to the origin. At time 0 request $r_1$ is presented at vertex $x$. Since the online algorithm is zealous, it starts moving the server to $x$ immediately. Once $r_1$ is served at time $\bar{x}$, there are no pending requests, so the server starts moving to the origin. At this moment request $r_2$ is presented again at vertex $x$, but the server must arrive to the origin before it can return to $x$. Once $r_2$ is satisfied, the server goes again to the origin, with a total cost of at least $4\bar{x}$. The adversary can serve both requests at time $\bar{x}$ moving its server only once to $x$, and then returning it to the origin, with a total cost of at most $2\bar{x}$, which proves the claim. □

**Theorem 10.** *No zealous online algorithm for NDOLTSP on any graph is better than* 3*-competitive against neither fair nor standard adversaries.*

*Proof.* Let $x$ be the closest vertex to the origin. At time 0 request $r_1$ is presented at vertex $x$, and the zealous algorithm starts moving its server to $x$ immediately. At this moment request $r_2$ is presented at the origin, but the server must complete its movement to $x$ (at time $\bar{x}$), and then it starts returning to the origin. At this moment request $r_3$ is presented at vertex $x$, but the server must reach the origin (at time $2\bar{x}$), and then it goes again to $x$, with a total cost of $3\bar{x}$. The adversary can end its job at time $\bar{x}$, serving $r_2$ at time 0 at the origin, and the other requests at time $\bar{x}$ at vertex $x$, which completes the proof. □

Note that in most of the cases our zealous algorithms `REP`, `ZZG` and `DREP` achieve competitive ratios coincident with the lower bounds we have just presented. More precisely, `REP` and `ZZG` are optimal zealous algorithms for DOLTSP on paths, while `DREP` is optimal for the Nomadic problem on any graph. This implies that in general the competitiveness achievable by zealous algorithms for DOLTSP does not depend on the type of adversary: the lower bounds of Theorem 9 and Theorem 10 are the same against both fair and standard adversaries, and those lower bounds are achieved in most of the cases.

Another interesting observation is that HDOLTSP is easier than NDOLTSP, at least on paths, since optimal zealous algorithms for HDOLTSP on paths are 2-competitive, while for NDOLTSP we have 3-competitive optimal algorithms. This is not surprising, if we consider that online algorithms for HDOLTSP have an extra bit of information: the server must always end at the origin.

In [8] it was proved that a zealous algorithm called Plan At Home is 3-competitive for (continuous) Homing OL-ATSP. Since OL-ATSP can be viewed as a generalization of DOLTSP, that result can be applied to our problem, achieving the same upper bound as Theorem 6.

## 4 General Lower Bounds

Lower bounds shown in Sect. 3 use the fact that the online algorithms are zealous. In this section we remove this restriction and present lower bounds valid for any online algorithm for DOLTSP. Some of the lower bounds hold on any graph, while others need a special distribution of the vertices.

We will start with HDOLTSP (Theorem 11 and Theorem 12), and then we will consider NDOLTSP (Theorem 13 and Theorem 14).

**Theorem 11.** *No online algorithm for HDOLTSP on any graph is better than $\rho$-competitive against a fair adversary, with $\rho = \frac{1+\sqrt{5}}{2} \approx 1.618$.*

*Proof.* Let $x$ be the closest vertex to the origin. At time 0 request $r_1$ is presented at vertex $x$. The online server must visit $x$ and return to the origin at a certain moment, because we are in the Homing problem. Let $\tau \geq \bar{x}$ be the moment in which the online server starts moving to the origin after serving $r_1$. If $\tau + \bar{x} \geq 2\bar{x}\rho$, no more requests are presented. In this case the online server arrives to the origin not before time $\tau + \bar{x}$, while the adversary can move its server to $x$ and return it to the origin at time $2\bar{x}$, so we have

$$\frac{C_{\mathrm{ALG}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{\tau + \bar{x}}{2\bar{x}} \geq \frac{2\bar{x}\rho}{2\bar{x}} = \rho.$$

On the other hand, if $\tau + \bar{x} < 2\bar{x}\rho$, a new request $r_2$ is presented at time $\tau$ at vertex $x$. In this situation the adversary can serve both requests at time $\tau$, with a total cost of at most $\tau + \bar{x}$. The online cost is at least $\tau + 3\bar{x}$, because when the online server reaches the origin after serving $r_1$, it must visit again $x$ and return again to the origin. Therefore we have

$$\frac{C_{\mathrm{ALG}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{\tau + 3\bar{x}}{\tau + \bar{x}} = 1 + \frac{2\bar{x}}{\tau + \bar{x}} > 1 + \frac{2\bar{x}}{2\bar{x}\rho} = 1 + \frac{1}{\rho} = \rho. \qquad \square$$

**Theorem 12.** *There exists a family of halfpaths where no online algorithm for HDOLTSP is better than $\rho$-competitive against a standard adversary, with $\rho = \frac{2+\sqrt{2}}{2} \approx 1.707$.*

**Theorem 13.** *No online algorithm for NDOLTSP on any graph is better than $\rho$-competitive against a fair adversary, with $\rho = \frac{1+\sqrt[3]{19-3\sqrt{33}}+\sqrt[3]{19+3\sqrt{33}}}{3} \approx 1.839$.*

*Proof.* Let $x$ be the closest vertex to the origin. At time 0 request $r_1$ is presented at vertex $x$. Let $\tau_1 \geq 0$ be the moment in which the online server leaves the origin. If $\tau_1 + \bar{x} \geq \rho\bar{x}$, the sequence of requests ends. The online server arrives to $x$ not before time $\tau_1 + \bar{x}$, while the adversary can reach the vertex at time $\bar{x}$, and then we have

$$\frac{C_{\mathrm{ALG}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{\tau_1 + \bar{x}}{\bar{x}} \geq \frac{\rho\bar{x}}{\bar{x}} = \rho \ .$$

On the contrary, if $\tau_1 + \bar{x} < \rho\bar{x}$, request $r_2$ is presented at time $\tau_1$ at the origin. Let $\tau_2 \geq \tau_1 + \bar{x}$ be the moment in which the online server starts moving

to the origin for serving $r_2$. If $\tau_2 + \bar{x} \geq \rho(\tau_1 + \bar{x})$, no more requests are presented. In this case the cost of the online algorithm is at least $\tau_2 + \bar{x}$, because its server must arrive to the origin. Since the adversary can wait at the origin until time $\tau_1$ for serving $r_2$, and then move to $x$ for serving $r_1$, we obtain

$$\frac{C_{\mathrm{ALG}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{\tau_2 + \bar{x}}{\tau_1 + \bar{x}} \geq \frac{\rho(\tau_1 + \bar{x})}{\tau_1 + \bar{x}} = \rho.$$

Finally, if $\tau_1 + \bar{x} < \rho\bar{x}$ and $\tau_2 + \bar{x} < \rho(\tau_1 + \bar{x})$, request $r_3$ is presented at time $\tau_2$ at vertex $x$. In this situation the online cost is at least $\tau_2 + 2\bar{x}$, because the server must visit $x$ after it arrives to the origin. Once again the adversary can wait at the origin until time $\tau_1$ for serving $r_2$, and then move to $x$ for serving $r_1$, ending its job at time $\tau_2$ when $r_3$ is presented. Therefore we have

$$\frac{C_{\mathrm{ALG}}(\sigma)}{C_{\mathrm{OPT}}(\sigma)} \geq \frac{\tau_2 + 2\bar{x}}{\tau_2} > 1 + \frac{2\bar{x}}{\rho(\tau_1 + \bar{x}) - \bar{x}} > 1 + \frac{2\bar{x}}{\rho^2\bar{x} - \bar{x}} = 1 + \frac{2}{\rho^2 - 1} = \rho. \quad \square$$

**Theorem 14.** *There exists a family of paths where no online algorithm for NDOLTSP is better than 2-competitive against a standard adversary.*

*Proof.* Consider a path with at least two vertices $x > 0$ and $-x$. WLOG, assume that at time $\bar{x}$ the online server is in the left halfpath. At that moment, a single request at vertex $x$ is presented. The online cost is at least $2\bar{x}$, while the adversary can serve the request at time $\bar{x}$. $\square$

Notice that the last lower bound is valid on a certain group of paths that are not halfpaths. It is essentially the same result presented in [9] for (continuous) OLTSP. In the full version of this paper we prove that the same lower bound holds on a particular group of halfpaths.

Online Dial-a-Ride Problem (OLDARP) generalizes (continuous) OLTSP to the case in which requests are pairs of points and a server must take an object from the first point to the second point. It is interesting to note that the lower bounds of this section are very similar to the corresponding lower bounds known for OLDARP on the real line. For instance, the lower bounds of Theorem 11 and Theorem 12 are coincident with the lower bounds given in [11] and [7] for Homing OLDARP on the real halfline against fair and standard adversaries, respectively. While Theorem 11 uses the same idea presented in [11], we derived Theorem 12 in a completely different way. The relation between our lower bounds and those for OLDARP on the real line must be studied further. However, a possible explanation for this phenomenon could be that, even though OLDARP is defined on a continuous metric space, once the server picks up an object it cannot satisfy other requests until the object is delivered. A similar situation occurs in DOLTSP, where the server cannot change direction until it arrives to the next vertex. A summary of results for OLDARP can be found in [11].

## 5 Cautious Algorithms

In Sect. 3 we saw that REP and ZZG achieve the best competitive ratio for zealous online algorithms. However, those ratios are notably higher than the general lower bounds shown in Sect. 4. It would be nice to have online algorithms with competitiveness closer to these general lower bounds. In order to succeed, we must consider a distinct class of algorithms.

An online *cautious* algorithm may wait without moving its server even when there is pending work. New requests presented while the algorithm is waiting (and even the absence of them), give additional information that the algorithm can use to improve its performance. On the contrary, a zealous algorithm faced with the same sequence of requests would take an early decision that could be inappropriate a posteriori.

A key point in the design of a cautious online algorithm is to decide how much time the server should wait when there is pending work. A longer waiting increases the possibilities to obtain additional information. Nonetheless, the caution must not be against the main goal of the algorithm, which is to minimize the total time to complete its job.

Our cautious online algorithms aim at obtaining competitive ratios coincident with the general lower bounds of Sect. 4. Each time a new request arrives the algorithms compute how long the adversary needs to serve all the known requests. Then, cautious algorithms wait just till the moment in which extending the waiting time would prevent obtaining the desired competitiveness. A number of online algorithms that wait taking into account the cost of the adversary were considered for continuous problems related to DOLTSP, such as OLTSP [10,11], OLDARP [7] and OL-ATSP [8].

We devised two cautious online algorithms using the general scheme described above. The main difference between them is when they decide to wait. Our first cautious algorithm is called Wait-Before-Return (WBR). The algorithm is only defined for HDOLTSP on halfpaths. It serves as soon as possible pending requests away from the origin. The remaining requests are satisfied when the server returns to the origin. Before doing so, WBR waits as explained above.

The other cautious algorithm is Wait-Before-Begin (WBB). It is possible to use this algorithm for both HDOLTSP and NDOLTSP, on any path (though we only prove results for NDOLTSP on halfpaths). Each time the server is halted and a new request is presented, WBB computes an optimal route that serves all the pending requests. Before starting its tour, the algorithm waits according to the general scheme explained above.

A more detailed description of WBR and WBB can be found in the full version of this paper. The following results establish the competitiveness of the algorithms in different variants of DOLTSP.

**Theorem 15.** *Algorithm WBR is $\rho$-competitive for HDOLTSP on any halfpath, with $\rho = \frac{1+\sqrt{5}}{2} \approx 1.618$ against a fair adversary, and $\rho = \frac{2+\sqrt{2}}{2} \approx 1.707$ against a standard adversary.*

**Theorem 16.** *Algorithm* `WBB` *is* 2*-competitive for NDOLTSP on any halfpath against both fair and standard adversaries.*

Notice that the above upper bounds match the general lower bounds of Sect. 4, with the exception of `WBB` for NDOLTSP against a fair adversary. If we compare these lower and upper bounds, against those of Sect. 3 for zealous algorithms, we observe that zealous algorithms for DOLTSP are weaker than cautious algorithms.

Recall that in Sect. 3 we showed that for zealous online algorithms, HDOLTSP is easier than NDOLTSP, and there is no difference between the two types of adversaries. If we review the results of this section and those of Sect. 4, we can observe that for cautious online algorithms once again HDOLTSP is easier than NDOLTSP. Besides, for cautious algorithms we have better performances against a fair adversary than against a standard adversary. That is, for cautious algorithms the fair adversary is weaker than the standard one. This is not surprising taking into account that the fair adversary has a restricted power.

As for general graphs, Theorem 3.2 in [8] states that a cautious algorithm called SmartStart is $\frac{3+\sqrt{5}}{2}$-competitive for (continuous) Homing OL-ATSP. The algorithm is a variation of an algorithm presented in [7]. In the same spirit as our algorithm `WBB`, SmartStart waits at the origin until the moment in which waiting more would prevent him from being competitive. At that moment it starts an optimal tour that serves all the pending requests and returns to the origin. As we said before, OL-ATSP can be viewed as a generalization of DOLTSP, so we can derive the following result.

**Theorem 17.** *Algorithm SmartStart for Homing OL-ATSP [8] determines a* $\rho$*-competitive algorithm for HDOLTSP on any graph against both fair and standard adversaries, with* $\rho = \frac{3+\sqrt{5}}{2} \approx 2.618$.

The above upper bound is far from the lower bounds of Theorem 11 and Theorem 12 (Section 4). It is not clear for now whether the lower bounds can be improved, or a better cautious algorithm can be found. In trying to improve the lower bounds, requests outside a halfpath must be considered, since Theorem 15 states that `WBR` achieves a performance coincident with those lower bounds for HDOLTSP on halfpaths.

## 6 Empirical Analysis of Online Algorithms

We designed a set of tests in order to get empirical evidence about how online algorithms work in practice on paths. In our tests, we included all our competitive algorithms, as well as other algorithms with some intuitive improvements. One of these new algorithms is Statistic–Replan (`STR`), a subtle enhancement to regular `REP`. The only difference occurs in NDOLTSP when the server is idle at some vertex. Instead of waiting passively for a new request, `STR` moves the server to the vertex that is the most likely to receive the next request, assuming the

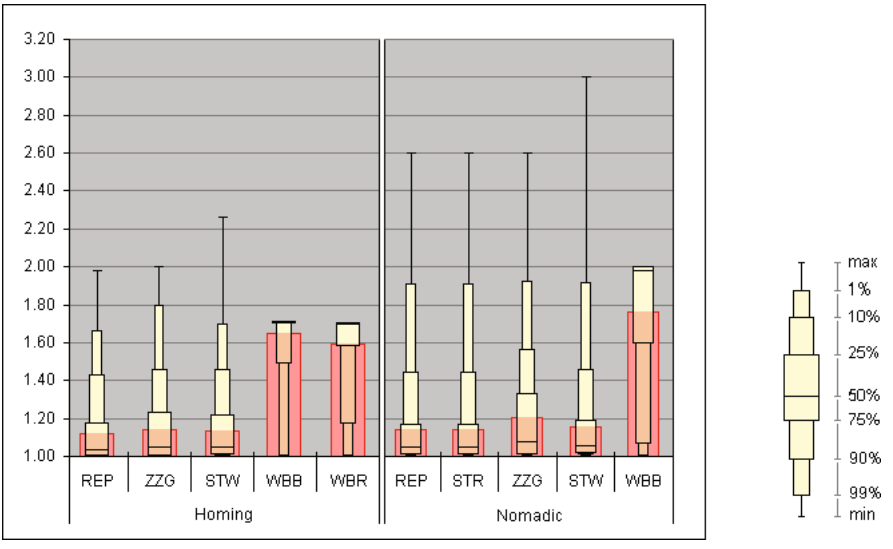



**Fig. 1.** Mean (*bar*) and distribution (*telescopic-plot*) of approximate ratios

existence of a pattern. Another algorithm we considered is Statistic–Wait-Before-Begin (`STW`), which ends the waiting time when it is likely enough that no more requests will be presented.

All the algorithms were tested using a large collection of instances designed to cover a representative set of cases. This collection takes into account different aspects of an instance: quantity and distribution of vertices, quantity and distribution (over time and over space) of requests, etc. A set of 7506 distinct scenarios was considered by combining different values for each aspect. A detailed description of the test set is given in the full version of this paper.

We executed all strategies over all instances on our test set. In each case we computed the *approximate ratio*, i.e. the ratio between the cost of the online solution and the optimal offline cost. Note that the approximate ratio of any strategy is upper bounded by its competitive ratio.

Figure 1 shows the average approximate ratio of all algorithms (as a bar), and the distribution of values among the different trials (as a telescopic-plot). One thing we can conclude is that, as suggested by the theoretical part of our study, HDOLTSP is easier than NDOLTSP. Besides, more than 50% of the times, the simplest algorithms generated quasi optimal solutions. Opposed to this, most approximate ratios of cautious strategies (`WBR` and `WBB`) exceed the threshold of 1.5. However, as we could expect, the worst case ratios of the simplest algorithms are much higher than those of the more sophisticated (competitive) ones. The cautious algorithm with best results is `STW`, although it was better than all zealous strategies only in 5% of the cases. The other cautious strategies (`WBR` and `WBB`) were surpassed by a zealous algorithm in 90% of the cases. With this evidence, we can conclude that waiting was not so profitable in practice. Finally,

our empirical study was useful to analyze how the different aspects of the instances affect their complexity. We found that the distribution of requests over time is the most influential aspect: it does not matter where vertices or requests are located; the later the requests become visible, the worse are the results that the online algorithm gets.

## References

1. de Paepe, W.E.: Complexity Results and Competitive Analysis for Vehicle Routing Problems. PhD thesis, Technische Universiteit Eindhoven (2002)
2. Deif, I., Bodin, L.: Extension of the Clarke and Wright algorithm for solving the vehicle routing problem with backhauling. In: Babson Conference on Software Uses in Transportation and Logistics Management, Babson Park, MA (1984)
3. Savelsbergh, M.W.P.: Local search in routing problems with time windows. Annals of Operations Research 4(1-4), 285–305 (1985)
4. Stein, D.: An asymptotic, probabilistic analysis of a routing problem. Mathematics of Operations Research 3(2), 89–101 (1978)
5. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
6. Fiat, A., Woeginger, G.J.: Online Algorithms: The State of the Art. LNCS, vol. 1442. Springer, Heidelberg (1998)
7. Ascheuer, N., Krumke, S.O., Rambau, J.: Online dial-a-ride problems: Minimizing the completion time. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 639–650. Springer, Heidelberg (2000)
8. Ausiello, G., Bonifaci, V., Laura, L.: The on-line asymmetric traveling salesman problem. Journal of Discrete Algorithms 6(2), 290–298 (2008)
9. Ausiello, G., Feuerstein, E., Leonardi, S., Stougie, L., Talamo, M.: Algorithms for the on-line travelling salesman. Algorithmica 29(4), 560–581 (2001)
10. Blom, M., Krumke, S.O., de Paepe, W., Stougie, L.: The online-TSP against fair adversaries. INFORMS Journal on Computing 13(2), 138–148 (2001)
11. Lipmann, M.: On-line Routing. PhD thesis, Technische Universiteit Eindhoven (2003)
12. Psaraftis, H.N., Solomon, M.M., Magnanti, T.L., Kim, T.U.: Routing and scheduling on a shoreline with release times. Management Science 36(2), 212–223 (1990)
13. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of ACM 28, 202–208 (1985)
14. Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D.: Competitive snoopy caching. Algorithmica 3, 79–119 (1988)
15. Koutsoupias, E., Papadimitriou, C.H.: Beyond competitive analysis. SIAM Journal on Computing 30(1), 300–317 (2000)

# On Approximating an Implicit Cover Problem in Biology

Mary V. Ashley[1], Tanya Y. Berger-Wolf[2], Wanpracha Chaovalitwongse[3], Bhaskar DasGupta[2], Ashfaq Khokhar[2], and Saad Sheikh[2]

[1] Department of Biological Sciences, University of Illinois at Chicago, 840 West Taylor Street, Chicago, IL 60607
ashley@eeb.uic.edu

[2] Department of Computer Science, University of Illinois at Chicago, 851 South Morgan Street, Chicago, IL 60607
{dasgupta,ssheikh,tanyabvv,ashfaq}@cs.uic.edu

[3] Department of Industrial Engineering, Rutgers University, PO Box 909, Piscataway, NJ 08855
wchaoval@rci.rutgers.edu

**Abstract.** In an implicit combinatorial optimization problem, the constraints are not enumerated explicitly but rather stated implicitly through equations, other constraints or auxiliary algorithms. An important subclass of such problems is the implicit set cover (or, equivalently, hitting set) problem in which the sets are not given explicitly but rather defined implicitly. For example, the well-known minimum feedback arc set problem is such a problem. In this paper, we consider such a cover problem that arises in the study of wild populations in biology in which the sets are defined implicitly via the Mendelian constraints and prove approximability results for this problem.

## 1 Introduction

In an *implicit* combinatorial optimization problem, the constraints are not enumerated explicitly but rather stated implicitly through equations, other constraints or auxiliary algorithms. Well-known examples of such optimization problems include convex optimization problems where the constraints are not given explicitly but rather can be queried implicitly through a separation oracle[1] or given by an auxiliary algorithm. This paper concerns the implicit set cover problems which are defined as follows. In the standard (unweighted) version of the set cover problem, we are given a collection of subsets $\mathcal{S}$ over an universe of elements $\mathcal{U}$ and the goal is to find a sub-collection of sets from $\mathcal{S}$ of minimum cardinality such that the union of these sets is precisely $\mathcal{U}$. A combinatorially equivalent

[1] For example, the ellipsoid method can be used to solve in polynomial time a linear programming problem with possibly exponentially many constraints provided we have a separation oracle that, given a tentative solution, in polynomial time either verifies that the solution is a feasible solution or provides a hyperplane separating the solution point from the feasible region.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 43–54, 2009.

version of the set cover problem is the so-called hitting set problem where one needs to pick instead a subset of the universe $\mathcal{U}$ of minimum cardinality which contains at least one element from every set. Set cover and hitting set problems are fundamental problems in combinatorial optimization whose computational complexities have been throughly investigated and well understood [15,27]. More general version of the problem could include generalizing the objective function to be minimized, namely the number of sets picked, by say having weighted sets and minimizing the sum of weights of the selected sets, or by defining a monotone objective function on the set system.

Implicit set cover (or hitting set) problems have the same standard setting, but the sets are not given explicitly but rather implicitly through some implicit combinatorial constraints. For example, the minimum feedback vertex set or the minimum feedback arc set problems are examples of such implicit hitting set problems. Such implicit set cover or hitting set problems can be characterized by not giving the collection of sets $\mathcal{S}$ explicitly but via an efficient (polynomial-time) *oracle* $\mathcal{O}$ that will supply members of $\mathcal{S}$ satisfying certain conditions. For example, the recent work of Richard Karp and Erick Moreno Centeno[2] considers some implicit hitting set problems with applications to multiple genome alignments in computational biology in which the oracle $\mathcal{O}$ provides a minimum-cardinality set (or a good approximation to it) from the collection $\mathcal{S}$ that is disjoint from a given set $Q$. In addition to standard polynomial-time approximation guarantees, one could also invoke other measures of efficiencies, such as number of access to the oracle $\mathcal{O}$ to obtain an optimal or near-optimal solution to the hitting set problem as used by Karp and Centeno.

In this paper, we consider an implicit (unweighted) set cover problem, which we call the MIN-PARENT problem, that arises in the study of wild population. Our problem in the setting described above is roughly as follows. Our oracle $\mathcal{O}$ returns, given a sub-collection of elements $\mathcal{U}' \subseteq \mathcal{U}$, if there is a set that includes $\mathcal{U}'$. Our specific objective function is motivated by the biological application and is a monotone function, namely including a new element in our collection does not decrease it. More precise formulations of our problems appear in the next section and will easily convince the reader that our problem is not captured by previous works or the recent work by Karp and Centeno.

## 2 Motivations

For wild populations, the growing development and application of molecular markers provides new possibilities for the investigation of many fundamental biological phenomena, including mating systems, selection and adaptation, kin selection, and dispersal patterns. The power and potential of the genotypic information obtained in these studies often rests in our ability to reconstruct genealogical relationships among individuals. These relationships include parentage, full and half-sibships, and higher order aspects of pedigrees [10,11,17]. In our motivation we are only concerned with full sibling relationships from single

[2] Richard Karp, personal communication.

generation sample of microsatellite markers. Several methods for sibling reconstruction from microsatellite data have been proposed [2,1,9,18,20,25,26,28]. Combinatorial approaches to sibling reconstruction using suitable parsimony assumptions have been studied in [5,7,8,13,14,23,24]. These approaches use the Mendelian inheritance rules to impose constraints on the genetic content possibilities of a sibling group. A formulation of the inferred combinatorial constraints in constructing a collection of groups of individuals that satisfy these constraints under the parsimony assumption of a minimum number of parents leads to the MIN-PARENT problems discussed in the paper.

## 3 Precise Formulations of MIN-PARENT Problems

An element (individual) $u$ is an ordered sequence $(u_1, u_2, \ldots, u_\ell)$ where each $u_j$ is a genetic trait (*locus*) and is represented by a multi-set $\{u_{j,0}, u_{j,1}\}$ of two (possibly equal) numbers (*alleles*) inherited from its parents. Biologically, each element corresponds to an individual in the sample of the wild population from the same generation. We have a universe $\mathcal{U}$ consisting of $n$ such elements. Certain sets of individuals in $\mathcal{U}$ can be *full siblings*, *i.e.* having the *same* pair of parents under the Mendelian inheritance rule. **These sets are specified in an implicit manner in the following way**. The Mendelian inheritance rule states that an individual $u = (u_1, u_2, \ldots, u_\ell) \in \mathcal{U}$ can be a child of a pair of individuals (*parents*), say $v = (v_1, v_2, \ldots, v_\ell)$ and $w = (w_1, w_2, \ldots, w_\ell)$, if and only if for each locus $j \in \{1, \ldots, \ell\}$ one element of $u_j$ is from $v_j$ and the other element of $u_j$ is from $w_j$. Finally, a subset $\mathcal{U}' \subseteq \mathcal{U}$ is a full sibling group if and only if there exists a pair of parents $v$ and $w$ such that every member of $\mathcal{U}'$ is a child of $v$ and $w$. Note that any pair of individuals can have the same pair of parents by the Mendelian constraints.

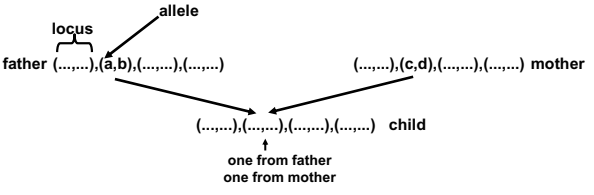



**Fig. 1.** Illustration of the Mendelian inheritance rule

Given these Mendelian constraints, our goal is to cover the universe $\mathcal{U}$ by a set of full-sibling groups under the parsimonious assumption of a minimum number of parents. Formally, the MIN-PARENT problem is defined as follows:

**Problem name:** MIN-PARENT$_{n,\ell}$
**Input:** An universe $\mathcal{U}$ of $n$ individuals each with $\ell$ loci.
**Valid Solutions:** A cover $\mathcal{A}$ of $\mathcal{U}$ such that each set $S \in \mathcal{A}$ in the cover is a sibling group.
**Notation:** Let $\mathcal{B}(\mathcal{A})$ denote a set of individuals (parents) such that every set $S$ (sibling group) in the cover has its two parents from $\mathcal{B}(\mathcal{A})$.
**Objective for minimization:** $|\mathcal{B}(\mathcal{U})| = \min_{\mathcal{A}} |\mathcal{B}(\mathcal{A})|$.

In the setting of the implicit set cover problems described before, our cover problem is as follows:

- Our sets (sibling groups) are defined implicitly by the Mendelian constraints; note that the number of such sets is possibly exponential and thus we cannot always enumerate them in polynomial time.
- Our polynomial time oracle $\mathcal{O}$ answers queries of the following type: *given a given subset $\mathcal{U}' \subseteq \mathcal{U}$ of the universe, does $\mathcal{U}'$ form a valid (sibling) set following the Mendelian constraints*[3]? It is easy to show a polynomial-time implementation of the oracle (*e.g.*, see [7]).

Finally, note that our objective function is obviously monotone since $\mathcal{U}' \subset \mathcal{U}$ implies $|\mathcal{B}(\mathcal{U}')| \leq |\mathcal{B}(\mathcal{U})|$. **A natural parameter of interest in covering problems the maximum size (number of elements) $a$ in any set**. For our problem, the parameter $a$ corresponds to maximum number of individuals of any sibling group.

We first show that the MIN-PARENT problem is MAX-SNP-hard even if $a = 3$. This leads us to the question about the computational complexity of the problem for arbitrary $a$. We will show that, for arbitrary $a$, it is hard to even find a minimum set of parents for a given sibling partition of the universe given a candidate set of parents. Formally, we can define the FIND-MIN-PARENT problem as follows:

**Problem name:** FIND-MIN-PARENT$_{n,\ell}$
**Input:**
- a partition $\mathcal{A}$ of a set $\mathcal{U}$ of $n$ elements, each with $\ell$ loci, such that each set $S$ in the partition $\mathcal{A}$ is a sibling set, and
- a set of elements (possible parents) $\mathcal{P}$.

**Notation:** Let $\mathcal{B}(\mathcal{A})$ denote a set of elements (parents) such that every set $S$ (sibling group) in the partition $\mathcal{A}$ has its two parents from $\mathcal{B}(\mathcal{A})$.
**Valid Solutions:** $\mathcal{B}(\mathcal{A}) \subseteq \mathcal{P}$.
**Objective for minimization:** $|\mathcal{B}(\mathcal{U})| = \min_{\mathcal{B}(\mathcal{A}) \subseteq \mathcal{P}} |\mathcal{B}(\mathcal{A})|$.

### 3.1 Standard Terminologies

Recall that a $(1+\varepsilon)$-*approximate solution* (or simply an $(1+\varepsilon)$-approximation) of a minimization (resp. maximization) problem is a solution with an objective value no larger (resp. no smaller) than $1+\varepsilon$ times (resp. $(1+\varepsilon)^{-1}$ times) the value of the optimum, and an algorithm achieving such a solution is said to have an *approximation ratio* of at most $1+\varepsilon$. A problem is $r$-inapproximable under a certain complexity-theoretic assumption means that the problem does not have a $r$-approximation unless the complexity-theoretic assumption is false.

L-reductions are a special kind of approximation-preserving reduction [21] that can be used to show MAX-SNP-hardness of an optimization problem. Given

[3] Note that if $\mathcal{U}'$ is not a valid set, the oracle $\mathcal{O}$ is unable to provide any hint about other possible candidates.

two optimization problems $\Pi$ and $\Pi'$, $\Pi$ L-reduces to $\Pi'$ if there are three polynomial-time procedures $T_1$,$T_2$, $T_3$ and two constants $a$ and $b > 0$ such that the following two conditions are satisfied:

**(1)** For any instance $I$ of $\Pi$, algorithm $T_1$ produces an instance $I' = f(I)$ of $\Pi'$ generated from $T_1$ such that the optima of $I$ and $I'$, $OPT(I)$ and $OPT(I')$, respectively, satisfy $OPT(I') \leq a \cdot OPT(I)$.

**(2)** For any solution of $I'$ with cost $c'$, algorithm $T_2$ produces another solution with cost $c''$ that is no worse than $c'$, and algorithm $T_3$ produces a solution of $I$ of $\Pi$ with cost $c$ (possibly from the solution produced by $T_2$) satisfying $|c - OPT(I)| \leq b \cdot |c'' - OPT(I')|$.

An optimization problem is MAX-SNP-hard if another MAX-SNP-hard problem L-reduces to that problem. Arora et al. [4] show that, assuming P≠NP, every MAX-SNP-hard problem is $(1+\varepsilon)$-inapproximable for some constant $\varepsilon > 0$ unless P=NP.

### 3.2 The Map

We show in Section 4.1 that MIN-PARENT$_{n,\ell}$ is MAX-SNP-hard even if $a = 3$, the smallest non-trivial value of $a$, and, for any $a$ and any integer constant $c > 0$, admits an easy $\left(\frac{a}{c} + \ln c\right)\sqrt{n}$-approximation using polynomial number of queries to the oracle $\mathcal{O}$ (and therefore in polynomial time). We show in Section 5 that, for arbitrary $a$, FIND-MIN-PARENT$_{n,\ell}$ is $2^{\log^{\varepsilon} n}$-inapproximable, for every constant $0 < \varepsilon < 1$, unless NP⊆DTIME($n^{polylog(n)}$).

## 4 Approximating MIN-PARENT

This section discusses lower and upper bounds on polynomial-time approximation of MIN-PARENT.

### 4.1 Inapproximability of MIN-PARENT for $a = 3$

**Lemma 1.** *MIN-PARENT$_{n,\ell}$ is MAX-SNP-hard even if $a = 3$.*

**Proof.** For notational simplification, when an individual has the multiset $\{a, a\}$ in a locus, we will refer to it by saying that the individual has a "**label**" of value $a$ in that locus. Our construction will ensure that all individuals have only labels at every locus. It is then easy to check that a set of individuals can be a sibling set if and only if at each locus they have labels with no more than two distinct values. In the sequel, we will use the terminologies "label $a$" and "locus $\{a, a\}$" interchangeably.

The (*vertex-disjoint*) triangle-packing (TP) problem is defined as follows. We are given an undirected connected graph $G$. A triangle is a cycle of 3 nodes. The goal is to find (pack) a maximum number of *node-disjoint* triangles in $G$. TP is known to be MAX-SNP-hard even if every vertex of $G$ has degree at most 4 [12]. Moreover, the proof in [12] show that the MAX-SNP-hard instances of TP in

their reduction produces an instance of TP with $n$ nodes in which an optimal solution has $\alpha n$ triangles for some constant $0 < \alpha < 1$.

We will provide an approximation preserving reduction from an instance graph $G$ of $n$ nodes of TP with nodes of $G$ having a maximum degree of 4 as obtained in [12] to MIN-PARENT$_{n,\ell}$. We introduce an individual **u** for every node $u$ of the graph $G$ and provide ordered label sequences for each node (individual) such that:

**(1)** Three individuals corresponding to a triangle of $G$ have at most two values in every locus and thus can be a sibling set.
**(2)** Three individuals that do not correspond to a triangle of $G$ have at least three values in some locus and thus cannot be a sibling set.
**(3)** Consider any *maximal* set of vertex disjoint triangles in $G$ and the corresponding sibling sets (each of size 3). Partition the remaining vertices of $G$ not covered by these triangles arbitrarily into pairs (groups of size 2) and consider the corresponding full sibling sets (each of size 2). Then, each sibling set in the above collection requires two *new* parents.
Note that since we have a maximal set of triangles, no three vertices in the set of pairs can form a triangle. Conversely, given any solution of the MIN-PARENT problem, we preprocess the solution (Algorithm $T_2$ in the definition of L-reduction) to get a canonical solution to ensure that no three individuals in the union of pairs can be s sibling set; this preprocessing obviously does not increase the number of sibling sets.

Note that, since any pair of individuals can be a full sibling set, the above properties imply that

> TP has a solution with $t$ triangles if and only if the 2-label cover can be solved with $2t + 2 \cdot \frac{n-3t}{2} = n - t$ parents.

The MAX-SNP-hardness now follows easily since $t$ is at most $\frac{n}{3}$ and an optimum solution of TP on $G$ has $\alpha n$ triangles for some constant $0 < \alpha < 1$. To be precise, in the notations of Section 3.1 for an L-reduction with $\Pi$ and $\Pi'$ being the TP and the MIN-PARENT problems, respectively, we have

**(1)** OPT(I')$= (1 - \alpha)n = \frac{1}{\alpha(1-\alpha)}\alpha n = \frac{1}{\alpha(1-\alpha)}$OPT(I).
**(2)** if $c'' = (1 - \alpha)n + x$ for some $x$ then $c = \alpha n - x$ and thus $|c - OPT(I)| = |c'' - OPT(I')|$.

Now, we describe the reduction.

Our first set of loci are as follows. The index of a locus, which we call the "coordinate", is defined by an "origin" node $a$. Thus, we will have at most $O(|V|)$ such loci. The respective label of an individual **v** at this coordinate is the distance from $a$ to $v$, assuming every edge has length 1.

Our second set of loci are as follows. We have such a locus **for every** set of vertices $\{u, v, w\}$ that **does not** form a triangle. Thus, we will have $O(|V|^3)$ such loci. Since the three vertices do not form a triangle, at least one pair of them, say $u$ and $v$, are not conected by an edge. As a result, the set of vertices $\{u, v, x\}$

do not form a triangle for any other vertex $x \notin \{u, v\}$. Our goal is to ensure that the vertices $u$, $v$ and $w$ cannot be a sibling group while not disallowing any other sibling groups that can be formed by a triangle in the graph. This is easy to do. Put the label 1 in this locus for the individual **u**, label 2 for individual **v** and label 3 for all other individuals.

It is easy to see that no two individuals are the same, *i.e.*, they differ in at least one locus.

First we need to check that Property **(1)** holds. The following is true with respect to the first set of loci. Consider a triangle $\{u, v, w\}$ and assume that $u$ has the minimum label value of $L$, *i.e.*, it is the nearest with respect to the origin node that defined this locus. Then labels of $v$ and $w$ are at least $L$ and at most $L + 1$, hence we have at most two labels. The second set of loci never disallows a sibling group corresponding to a triangle, so the property is not violated by them either.

The construction of the second set of loci implies that Property **(2)** is true.

Now, we need to verify Property **(3)**. There are three cases to verify.

First, consider the case when we have two sibling groups correspond to two triangles $T_1 = \{u, v, w\}$ and $T_2 = \{p, q, r\}$ in $G$. Note that since nodes in $G$ have a maximum degree of 4, any node of one triangle can be connected to at most two nodes in the other triangle.

The locus corresponding to the index with $u$ as the origin node has a label 0 for $u$ and a label 1 for $v$ and $w$. Thus, the sibling set $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ can be generated **only** by a pair of parents, say $A$ and $B$, each of which has the alleles $\{0, 1\}$ in the corresponding locus.

Since $u$ is connected to at most two nodes in $T_2$, it is connected to a node in $T_2$, say $r$. Then, $r$ must have a label $x$ in this locus which is at least 2. Thus, neither $A$ nor $B$ can be a parent of the sibling group $\{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$ since $x \notin \{0, 1\}$.

Second, consider the case when the we have two sibling groups corresponding to a triangle $T = \{u, v, w\}$ and a pair $P = \{p, q\}$. Consider the locus corresponding to the index with $u$ as the origin node. We have a label 0 for $u$ and a label 1 for $v$ and $w$. Thus, the sibling set $\{\mathbf{u}, \mathbf{v}, \mathbf{w}\}$ can be generated **only** by a pair of parents, say $A$ and $B$, each of which has the alleles $\{0, 1\}$ in the corresponding locus. If node $u$ is *not* connected to both nodes $p$ and $q$ then one of the nodes which is not connected to $u$, say $p$, must have a label $x$ in this locus which is at least 2. Thus, neither $A$ nor $B$ can be a parent of the sibling group $\{\mathbf{p}, \mathbf{q}\}$ since $x \notin \{0, 1\}$. Otherwise, it must be the case that $u$ is connected to both $p$ and $q$.

Repeating the same argument with $q$ as the origin node and then $r$ as the origin node shows that the only case that remains to be considered is when *each* of $u$, $v$ and $w$ is connected to *both* the nodes $p$ and $q$. But, then the induced subgraph of $G$ with vertices $u, v, w, p$ and $q$ is 5-clique. Since every node in $G$ has a degree of no more than 4, this implies that is subgraph is a connected component of $G$ separated from the rest of the graph, contradicting the fact that $G$ was a connected graph.

Finally, consider the case when we have two sibling groups corresponding to two pairs $P_1 = \{u, v\}$ and $P_2 = \{p, q\}$. Since we have a canonical solution of the

MIN-PARENT problem or a maximal set of triangles for the TP problem, node $u$ is not connected to at least one node in $P_2$, say $p$. The locus corresponding to the index with $u$ as the origin node has a label 0 for $u$ and a label 1 for $v$, but for $p$ we must have a label $x$ in this locus which is at least 2. Thus, the sibling set $\{\mathbf{u}, \mathbf{v}\}$ can be generated **only** by a pair of parents, say $A$ and $B$, each of which has the alleles $\{0, 1\}$ in the corresponding locus, but neither $A$ nor $B$ can be a parent of the sibling group $\{\mathbf{p}, \mathbf{q}\}$ since $x \notin \{0, 1\}$. ❑

### 4.2 A Simple Approximation Algorithm for MIN-PARENT$_{n,\ell}$

Note that we do not need to know the value of $a$ in the theorem below.

**Lemma 2.** *Let $a$ be the maximum size of any sibling set. Then, for any integer constant $c > 0$, MIN-PARENT admits an easy $\left(\frac{a}{c} + \ln c\right)\sqrt{n}$-approximation with polynomially many access to the oracle $\mathcal{O}$ (and, therefore in polynomial time).*

**Proof.** Our proof is similar to the analysis of a standard greedy algorithm for set cover problems [27].

Suppose that we have a subset $\mathcal{U}' \subset \mathcal{U}$ of the universe that is still not covered. We can enumerate all subsets of $\mathcal{U}'$ of size at most $c$ in $O(n^c)$ time and for each subset query the oracle $\mathcal{O}$ to find if any of these subsets of individuals are full siblings for the MIN-PARENT$_{n,\ell}$ problem. Thus we can assume that for every instance of the problem, either the maximum sibling set size is below $c$ and we can find such a group of maximum size, or we can find a sibling set of size $c$. Our algorithm simply selects such a set, removes the corresponding elements from $\mathcal{U}'$ and continues until all elements of $\mathcal{U}$ are covered.

Obviously, all subsets of a sibling set are valid sibling sets too. Let OPT be the minimum number of parents in an optimal solution of MIN-PARENT$_{n,\ell}$. Consider an optimum solution, make it disjoint by arbitrarily shrinking each full-sibling set and let $\alpha$ be the number of sets in this partition. Obviously, $\alpha \leq n/2$. Since no two full-sibling sets are produced by the same pair of parents (because of minimality), $\binom{\mathrm{OPT}}{2} \geq \alpha$ which implies OPT$> \sqrt{2\alpha}$. We distribute the cost of our solution among the sets of the optimum. When a set with $b$ elements is selected, we remove each of its element and charge the sets of the optimum $1/b$ for each removal. It is easy to see that a set with $a$ elements will get the sequence of charges with values at most $(\underbrace{1/c, \ldots, 1/c}_{a-c \text{ times}}, 1/(c-1), 1/(c-2), \ldots, 1)$ and these charges add to $\frac{a}{c} - 1 + \sum_{i=1}^{c} \frac{1}{i} = \frac{a}{c} + \sum_{i=2}^{c} \frac{1}{i} < \frac{a}{c} + \ln c$. Thus, we use at most $\left(\frac{a}{c} + \ln c\right)\alpha$ sibling groups. Each sibling group can be generated by at most two new parents. Thus, the total number of parents necessary to generated these sibling groups is at most $\left(\frac{a}{c} + \ln c\right)\sqrt{2\alpha}\,\mathrm{OPT} < \left(\frac{a}{c} + \ln c\right)\sqrt{n}\,\mathrm{OPT}$. ❑

## 5 Inapproximability of FIND-MIN-PARENT

**Lemma 3.** *For every constant $0 < \varepsilon < 1$, FIND-MIN-PARENT$_{n,\ell}$ is $2^{\log^{\varepsilon} n}$-inapproximable unless NP$\subseteq$DTIME($n^{polylog(n)}$).*

**Proof.** We first need the MINREP problem which is defined as follows. The problem is closely related to the "label cover" problem defined in [3].

We are given a bipartite graph $G = (A, B, E)$. Also is given a partition of $A$ into $|A|/\alpha$ equal-size subsets $A_1, A_2, \ldots, A_\alpha$ and a partition of $B$ into $|B|/\beta$ equal-size subsets $B_1, B_2, \ldots, B_\beta$. These partitions define a natural "bipartite super-graph" $H$ in the following manner. $H$ has a "super-vertex" for every $A_i$ (the left partition) and a "super-vertex" for every $B_j$ (the right partition). There exists an "super-edge" between the super-vertex $A_i$ and the super-vertex $B_j$ if and only if there exists $u \in A_i$ and $v \in B_j$ such that $\{u, v\}$ is an edge of $G$.

A pair of vertices $u$ and $v$ "witnesses" a super-edge $\{A_i, B_j\}$ provided $a \in A_i$, $b \in B_j$ and the edge $\{a, b\}$ exists in $G$. A set of vertices $S$ of $G$ witnesses a super-edge if there exists at least one pair of vertices in $S$ that witnesses the super-edge.

The goal of the MINREP problem is to find $A' \subseteq A$ and $B' \subseteq B$ such that $A \cup B$ witnesses *every* super-edge of $H$ and the size of the solution, namely $|A'| + |B'|$, is *minimum.*

For notation simplicity, let $n = |A| + |B|$. The following result is a consequence of Raz's parallel repetition theorem [16,22,19].

**Theorem 4.** [19,22] *Let $L \in NP$ and $0 < \varepsilon < 1$ be any fixed constant. Then, there exists a reduction running in quasi-polynomial time, namely in time $n^{\mathrm{polylog(n)}}$, that given an instance $x$ of $L$ produces an instance of MINREP such that:*

- *if $x \in L$ then MINREP has a solution of size at most at most $\alpha + \beta$;*
- *if $x \notin L$ then MINREP has a solution of size at least or a solution of size at least $(\alpha + \beta) \cdot 2^{\log^{\varepsilon} n}$.*

Thus, the above theorem provides a $2^{\log^{\varepsilon} n}$-inapproximability for MINREP under the complexity-theoretic assumption of NP$\not\subseteq$DTIME($n^{\mathrm{polylog(n)}}$).

Let $L$ be any language in NP. Use the above theorem to translate an instance $x$ of $L$ to an instance of MINREP as described in the above theorem. Now, we describe a translation of this instance of MINREP to an instance of FIND-MIN-PARENT$_{\mathcal{P},n,\ell}$.

We have a parent $p_v$ in $\mathcal{P}$ corresponding to every element $v \in A \cup B$. We have an individual $s_{a,b}$ in $\mathcal{U}$ for every edge $\{a, b\}$ in $G$. Thus, the number of possible parents in $\mathcal{P}$ is $n$ and the number of individuals in $\mathcal{U}$ is $O(n^2)$. It therefore suffices to prove a $2^{\log^{\varepsilon} |\mathcal{P}|}$-inapproximability since that implies as $2^{\log^{\varepsilon} |\mathcal{U}|}$-inapproximability.

Before describing our reduction, we need a generic construction of the following nature to simplify our description. We are given two elements $p_u, p_v \in \mathcal{P}$ and an element $s_{a,b} \in \mathcal{U}$. We want to add a new locus with appropriate allele values to ensure that $s_{a,b}$ **cannot** be a child of $p_u$ and $p_v$, but **no additional parent-child relationship is forbidden**. This is easy to do. Put the alleles $\{a, b\}$ in this locus for $p_u$ and $p_v$ and put the alleles $\{a, c\}$ in this locus for every individual (including $s_{a,b}$) in $(\mathcal{P} \cup \mathcal{U}) \setminus \{p_u, p_v\}$. It follows that $s_{a,b}$ cannot be a child of $p_u$ and $p_v$ since $c \notin \{a, b\}$, but no other child-parent combination is

forbidden since $\{a, c\}$ can be produced by the Mendelian rule either from $\{a, b\}$ and $\{a, c\}$ or from $\{a, c\}$ and $\{a, c\}$.

Now, we add additional loci to the individuals in $\mathcal{U} \cup \mathcal{P}$ in the following manner following the two rules:

**Rule** ($\star$)**:** For every edge $\{u, v\}$ of $G$ with $u \in A_i$ and $v \in B_j$ and for every pair of vertices $\{a, b\}$ such that $\{a, b\} \in E \setminus \{\, \{y, z\} \mid y \in A_i, z \in B_j, \{y, z\} \in E \,\}$ we add an additional locus using the generic construction to ensure that $s_{a,b}$ *cannot* be a child of $p_u$ and $p_v$.

**Rule** ($\star\star$)**:** For every pair of vertices $u$ and $v$ of $G$ such that $\{u, v\} \notin E$ and for every pair of vertices $a$ and $b$ of $g$ such that $\{a, b\} \in E$, we add an additional locus using the generic construction to ensure that the individual $s_{a,b} \in \mathcal{U}$ *cannot* be a child of the parents $p_u$ and $p_v$ in $\mathcal{P}$.

We build each individual in $\mathcal{U} \cup \mathcal{P}$ locus-by-locus in the above manner. Finally, our given sibling partition $\mathcal{A}$ of $\mathcal{U}$ puts all the individuals $s_{a,b}$ with $\{a, b\}$ witnessing the *same* super-edge in the same partition. Let the sibling set corresponding to the super-edge $\{A_i, B_j\}$ be $\mathcal{A}_{i,j} = \{\{s_{a,b}\} \mid \{a, b\}$ witnesses the super-edge $\{A_i, B_j\}\}$.

First, we need to verify that each of our sibling set is indeed a sibling set. Consider the sibling set $\mathcal{A}_{i,j}$. Pick any $u \in A_i$ and $v \in B_j$ such that $\{u, v\} \in E$, *i.e.*, $\{u, v\}$ witnesses the super-edge $\{A_i, B_j\}$. We claim that $p_u$ and $p_v$ are the parents for all individuals in $\mathcal{A}_{i,j}$. Indeed, the two rules allow this.

Suppose that MINREP has a solution of size $\gamma$. This generates a set of $\gamma$ parents for FIND-MIN-PARENT in an obvious manner: for every vertex $v$ in the solution of MINREP we pick the individual $p_v$ in the solution of FIND-MIN-PARENT. If the super-edge $\{A_i, B_j\}$ is witnessed by the edge $\{u, v\}$ in the solution of MINREP, then the sibling set $\mathcal{A}_{i,j}$ is generated by the parents $p_u$ and $p_v$.

Conversely, suppose that FIND-MIN-PARENT has a solution with $\gamma$ parents. We associate each parent $p_u$ to the corresponding vertex $u$ of $G$ in our solution of MINREP. Consider a super-edge $\{A_i, B_j\}$ and the associated sibling set $\mathcal{A}_{i,j}$. Suppose that $p_u$ and $p_v$ are the parents of this group. By Rule ($\star$), $\{u, v\} \in E$. By Rule ($\star\star$), one of $p_u$ and $p_v$, say $p_u$, must be from $A_i$ and the other one $p_v$ from $B_j$. Thus, the edge $\{u, v\}$ witnesses this super-edge. ❑

*Remark 1.* The inapproximability reduction works even if one does not specify the set $\mathcal{A}$ of sibling partition explicitly as part of input but allows all feasible partitions.

## References

1. Almudevar, A.: A simulated annealing algorithm for maximum likelihood pedigree reconstruction. Theoretical Population Biology 63, 63–75 (2003)
2. Almudevar, A., Field, C.: Estimation of single generation sibling relationships based on DNA markers. Journal of Agricultural, Biological, and Environmental Statistics 4, 136–165 (1999)

3. Arora, S., Lund, C.: Hardness of Approximations. In: Hochbaum, D. (ed.) Approximation Algorithms for NP-hard Problems. PWS Publishing (1996)
4. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and hardness of approximation problems. Journal of the ACM 45(3), 501–555 (1998)
5. Ashley, M.V., Caballero, I.C., Chaovalitwongse, W., DasGupta, B., Govindan, P., Sheikh, S., Berger-Wolf, T.Y.: KINALYZER, A Computer Program for Reconstructing Sibling Groups, to appear in Molecular Ecology Resources
6. Ashley, M., Berger-Wolf, T., Berman, P., Chaovalitwongse, W., DasGupta, B., Kao, M.-Y.: On Approximating Four Covering/Packing Problems. To appear in Journal of Computer and System Sciences
7. Berger-Wolf, T.Y., DasGupta, B., Chaovalitwongse, W., Ashley, M.V.: Combinatorial reconstruction of sibling relationships. In: Proceedings of the 6th International Symposium on Computational Biology and Genome Informatics, pp. 1252–1255 (2005)
8. Berger-Wolf, T.Y., Sheikh, S., DasGupta, B., Ashley, M.V., Caballero, I., Chaovalitwongse, W., Putrevu, S.L.: Reconstructing Sibling Relationships in Wild Populations. Bioinformatics (special issue of ISMB 2007) 23(13), i49–i56 (2007)
9. Beyer, J., May, B.: A graph-theoretic approach to the partition of individuals into full-sib families. Molecular Ecology 12, 2243–2250 (2003)
10. Blouin, M.S.: DNA-based methods for pedigree reconstruction and kinship analysis in natural populations. TRENDS in Ecology and Evolution 18(10), 503–511 (2003)
11. Butler, K., Field, C., Herbinger, C., Smith, B.: Accuracy, efficiency and robustness of four algorithms allowing full sibship reconstruction from DNA marker data. Molecular Ecology 13, 1589–1600 (2004)
12. Caprara, A., Rizzi, R.: Packing Triangles in Bounded Degree Graphs. Information Processing Letters 84(4), 175–180 (2002)
13. Chaovalitwongse, W., Berger-Wolf, T.Y., DasGupta, B., Ashley, M.V.: Set covering approach for reconstruction of sibling relationships. Optimization Methods and Software 22(1), 11–24 (2007)
14. Chaovalitwongse, W., Chou, C.-A., Berger-Wolf, T., DasGupta, B., Sheikh, S., Ashley, M., Caballero, I.: New Optimization Model and Algorithm for Sibling Reconstruction from Genetic Markers. To appear in INFORMS Journal of Computing
15. Feige, U.: A threshold for approximating set cover. JACM 45, 634–652 (1998)
16. Feige, U., Lovasz, L.: Two-prover one-round proof systems: their power and their problems. In: Proc. 24th annual ACM symposium on Theory of computing, pp. 733–744 (1992)
17. Jones, A.G., Ardren, W.R.: Methods of parentage analysis in natural populations. Molecular Ecology (12), 2511–2523 (2003)
18. Konovalov, D.A., Manning, C., Henshaw, M.T.: KINGROUP: a program for pedigree relationship reconstruction and kin group assignments using genetic markers. Molecular Ecology Notes, doi: 10.1111/j.1471-8286.2004.00796.x
19. Kortsarz, G., Krauthgamer, R., Lee, J.R.: Hardness of Approximating Vertex-Connectivity Network Design Problems. SIAM J. of Computing 33(3), 704–720 (2004)
20. Painter, I.: Sibship reconstruction without parental information. Journal of Agricultural, Biological, and Environmental Statistics 2, 212–229 (1997)
21. Papadimitriou, C.H., Yannakakis, M.: Optimization, approximation, and complexity classes. Journal of Computer and System Sciences 43(3), 425–440 (1991)
22. Raz, R.: A parallel repetition theorem. SIAM J. of Computing 27(3), 763–803 (1998)

23. Sheikh, S.I., Berger-Wolf, T.Y., Ashley, M.V., Caballero, I.C., Chaovalitwongse, W., DasGupta, B.: Error Tolerant Sibship Reconstruction in Wild Populations. In: Markstein, P., Xu, Y. (eds.) Computational Systems Bioinformatics, pp. 273–284. World Scientific Publishers, Singapore (2008)
24. Sheikh, S.I., Berger-Wolf, T.Y., Khokhar, A.A., DasGupta, B.: Consensus Methods for Reconstruction of Sibling Relationships from Genetic Data. In: 4th Multidisciplinary Workshop on Advances in Preference Handling, Chicago, IL (2008)
25. Smith, B.R., Herbinger, C.M., Merry, H.R.: Accurate partition of individuals into full-sib families from genetic data without parental information. Genetics 158, 1329–1338 (2001)
26. Thomas, S.C., Hill, W.G.: Sibship reconstruction in hierarchical population structures using markov chain monte carlo techniques. Genet. Res., Camb. 79, 227–234 (2002)
27. Vazirani, V.: Approximation Algorithms. Springer, Heidelberg (2001)
28. Wang, J.: Sibship reconstruction from genetic data with typing errors. Genetics 166, 1968–1979 (2004)

# Power Indices in Spanning Connectivity Games

Haris Aziz, Oded Lachish, Mike Paterson, and Rahul Savani

Computer Science Department, University of Warwick
CV4 7AL Coventry, UK
{h.aziz,oded,msp,rahul}@dcs.warwick.ac.uk

**Abstract.** The Banzhaf index, Shapley-Shubik index and other voting power indices measure the importance of a player in a coalitional game. We consider a simple coalitional game called the spanning connectivity game (SCG) based on an undirected, unweighted multigraph, where edges are players. We examine the computational complexity of computing the voting power indices of edges in the SCG. It is shown that computing Banzhaf values is #P-complete and computing Shapley-Shubik indices or values is NP-hard for SCGs. Interestingly, Holler indices and Deegan-Packel indices can be computed in polynomial time. Among other results, it is proved that Banzhaf indices can be computed in polynomial time for graphs with bounded treewidth. It is also shown that for any reasonable representation of a simple game, a polynomial time algorithm to compute the Shapley-Shubik indices implies a polynomial time algorithm to compute the Banzhaf indices. This answers (positively) an open question of whether computing Shapley-Shubik indices for a simple game represented by the set of minimal winning coalitions is NP-hard.

**Keywords:** Network connectivity, coalitional games, Banzhaf index, Shapley-Shubik index.

## 1 Introduction

In this paper, we study the natural problem of computing the influence of edges in keeping an unweighted and undirected multigraph connected. Game theorists have studied notions of efficiency, fairness and stability extensively. Therefore, it is only natural that when applications in computer science and multiagent systems require fair and stable allocations, social choice theory and cooperative game theory provide appropriate foundations. For example, a network administrator with limited resources to maintain the links in the network may decide to commit resources to links according to their connecting ability. A spy network comprises communication channels. In order to intercept messages on the channels, resources may be utilized according to the ability of a channel to connect all groups. In a social network, we may be interested in checking which connections are more important in maintaining connectivity and hence contribute more to social welfare.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 55–67, 2009.

Our model is based on undirected, unweighted and connected multigraphs. All the nodes are treated equally, and the importance of a edge is based solely on its ability to connect all the nodes. Using undirected edges is a reasonable assumption in many cases. For example, in a social network, relations are usually mutually formed.

We use a multigraph as a succinct representation of a simple coalitional game called the *spanning connectivity game (SCG)*. The players of the game are the edges of the multigraph. The importance of an edge is measured by computing its voting power index in the game. Voting power indices including the Banzhaf index and Shapley-Shubik index are standard ways to compute the importance of a player in a coalitional voting game. Intuitively, the Banzhaf value is the number of coalitions in which a player plays a critical role and the Shapley-Shubik index is the proportion of permutations for which a player is pivotal.

The whole paper is concerned with computing solutions for SCGs. In Section 2, a summary of related work is given. In Section 3, preliminary definitions related to graph theory and coalitional games are given, and we define SCGs. Section 4 presents hardness results for computing Banzhaf values and Shapley-Shubik indices. In Section 5, positive computational results for Banzhaf values and Shapley-Shubik indices are provided for certain graph classes. Section 6 presents a polynomial-time algorithm to compute Holler indices and Deegan-Packel indices. In Section 7, a summary of results is given and future work is discussed.

## 2 Related Work

Power indices such as the Banzhaf and Shapley-Shubik indices have been extensively used to gauge the power of a player in different coalitional games such as weighted voting games [17] and corporate networks [13]. These indices have recently been used in network flow games [7], where the edges in the graph have capacities and the power index of an edge signifies the influence that an edge has in enabling a flow from the source to the sink. Voting power indices have also been examined in vertex connectivity games [8] on undirected, unweighted graphs; there the players are nodes, which are partitioned into primary, standard, and backbone classes.

The study of cooperative games in combinatorial domains is widespread in operations research [11,14]. Spanning network games have been examined previously [19,23] but they are treated differently, with weighted graphs and *nodes* as players (not *edges*, as here). The SCG is related to the all-terminal reliability model, a non-game-theoretic model that is relevant in broadcasting [22,10]. Whereas the reliability of a network concerns the overall probability of a network being connected, this paper concentrates on resource allocation to the edges. A game-theoretic approach can provide fair and stable outcomes in a strategic setting.

# 3 Preliminaries

## 3.1 Graph Theory

**Definition 1.** *A* multigraph $G := (V, E, s)$ *consists of a simple underlying graph* $(V, E)$ *with a multiplicity function* $s : E \mapsto \mathbb{N}$ *where* $\mathbb{N}$ *is the set of natural numbers excluding* 0. *Let* $|V| = n$ *and* $|E| = m$. *For every underlying edge* $i \in E$, *we have* $s_i$ *edges in the multigraph. The multigraph has a total of* $M = \sum_{i \in E} s_i$ *edges.*

**Definition 2.** *A* subgraph $G' = (V', E')$ *of a graph* $G = (V, E)$ *is a graph where* $V'$ *is a subset of* $V$ *and* $E'$ *is a subset of* $E$ *such that the vertex set of* $E'$ *is a subset of* $V'$. *A subgraph* $H$ *is a* connected spanning subgraph *of a graph* $G$ *if it is connected and has the same vertex set as* $G$.

## 3.2 Coalitional Game Theory

**Definition 3.** *A* simple voting game *is a pair* $(N, v)$ *with characteristic function* $v : 2^N \to \{0, 1\}$ *where* $v(\emptyset) = 0$, $v(N) = 1$ *and* $v(S) \leq v(T)$ *whenever* $S \subseteq T$. *A coalition* $S \subseteq N$ *is* winning *if* $v(S) = 1$ *and* losing *if* $v(S) = 0$. *A simple voting game can alternatively be defined as* $(N, W)$ *where* $W$ *is the set of winning coalitions.*

For the sake of brevity, we will abuse the notation to sometimes refer to game $(N, v)$ as $v$. For each connected multigraph $(V, E, s)$, we define the SCG, *spanning connectivity game*, $(E, v)$ with players $E$ and valuation function $v$, defined as follows for $S \subseteq E$:

$$v(S) = \begin{cases} 1, \text{ if there exists a spanning tree } T = (V, E') \text{ such that } E' \subseteq S \\ 0, \text{ otherwise} \end{cases}$$

It is easy to see that the SCG $(E, v)$ is a simple game because the outcome is binary, $v$ is monotone, $v(\emptyset) = 0$ and $v(E) = 1$. We consider power indices and cooperative game solutions for the edges in the SCG.

**Definition 4.** *A player* $i$ *is* critical *in a coalition* $S$ *when* $v(S) = 1$ *and* $v(S \setminus \{i\}) = 0$. *For each* $i \in N$, *we denote the number of coalitions in which* $i$ *is critical in game* $v$ *by the* Banzhaf value $\eta_i(v)$. *The* Banzhaf Index *of player* $i$ *in game* $v$ *is*

$$\beta_i = \frac{\eta_i(v)}{\sum_{i \in N} \eta_i(v)} \ .$$

The *Shapley-Shubik index* is the proportion of permutations for which a player is *pivotal*. For a permutation $\pi$ of $N$, the $\pi(i)$th player is pivotal if coalition $\{\pi(1), \ldots, \pi(i-1)\}$ is losing but coalition $\{\pi(1), \ldots, \pi(i)\}$ is winning.

**Definition 5.** *The* Shapley-Shubik (SS) value *is the function* $\kappa$ *that assigns to any simple game* $(N, v)$ *and any voter* $i$ *a value* $\kappa_i(v)$ *where*

$$\kappa_i = \sum_{S \subseteq N} (|S| - 1)!(n - |S|)!(v(S) - v(S \setminus \{i\})).$$

*The* Shapley-Shubik (SS) index *of* $i$ *is defined by*

$$\phi_i = \frac{\kappa_i}{n!}.$$

The Banzhaf index and the Shapley-Shubik index are the normalized versions of the Banzhaf value and the Shapley-Shubik value respectively. Since the denominator of the Shapley-Shubik index is fixed, computing the Shapley-Shubik index and Shapley-Shubik value have the same complexity. This is not necessarily true for the Banzhaf index and Banzhaf value.

## 4 Complexity of Computing Power Indices

We define the problems of computing the power indices of the edges in the SCG. For any power index X (e.g. Banzhaf value, Banzhaf index, Shapley-Shubik index etc.) we define the problem SCG-X as follows:

**Problem:** SCG-X
**Instance:** Multigraph $G$
**Output:** For the SCG corresponding to $G$, compute X for all the edges

We represent a communication network as a multigraph, where an edge represents a connection that may or may not work. An edge is said to be *operational* if it works. For a given graph $G$, the reliability $Rel(G, \{p_i\})$ of $G$ is the probability that the operational edges form a connected spanning subgraph, given that each edge is operational with probability $p_i$ for $i = 1, \ldots m$.

**Problem:** Rational Reliability Problem
**Instance:** Multigraph $G$ and $p_i \in \mathbb{Q}$ for all $i$, $1 \leq i \leq m$
**Output:** Compute $Rel(G, \{p_i\})$

A special case of the reliability problem is when every edge has the same probability $p$ of being operational. This is called the *Functional Reliability Problem.* A connected spanning subgraph with $i$ edges will occur with probability $p^i(1-p)^{m-i}$.

**Definition 6.** *Let* $N_i$ *be the number of connected spanning subgraphs with* $i$ *edges. Then the required output of the Functional Reliability Problem is the* reliability polynomial

$$Rel(G, p) = \sum_{i=0}^{m} N_i p^i (1-p)^{m-i}.$$

**Problem:** Functional Reliability Problem
**Instance:** Multigraph $G$ and $p \in \mathbb{Q}$
**Output:** Compute the coefficients $N_i$ of the reliability polynomial for all $i$, $1 \leq i \leq m$.

Ball [10] points out that an algorithm to solve the Rational Reliability Problem can be used as a sub-routine to compute all the coefficients for the Functional Reliability Problem. Moreover he proved that computing the general coefficient $N_i$ is NP-hard and therefore computing the rational reliability of a graph is NP-hard. As we will see in Section 5, reliability problems have connections with computing power indices of SCG. We first prove that SCG-BANZHAF-VALUE is #P-complete.

**Proposition 1.** *SCG-BANZHAF-VALUE is #P-complete even for simple, bipartite and planar graphs.*

*Proof.* We present a reduction from the problem of counting connected spanning subgraphs. SCG-BANZHAF-VALUE is clearly in #P because a connected spanning subgraph can be verified in polynomial time. It is known that counting the total number of connected spanning subgraphs is #P-complete even for simple, bipartite and planar graphs( [9], p. 305). We now reduce the problem of computing the total number of connected spanning subgraphs to solving SCG-BANZHAF-VALUE. Take $G = (V, E)$ with $n$ nodes and $m$ edges. Transform graph $G$ into $G' = (V \cup \{n+1\}, E \cup \{m+1\})$ by taking any node and connecting it to a new node via a new edge. Then the number of spanning subgraphs in $G$ is equal to the Banzhaf value of edge $m+1$ in graph $G'$. This shows that SCG-BANZHAF-VALUE is #P-complete. □

Similarly, SCG-SS is NP-hard.

**Proposition 2.** *SCG-SS is NP-hard even for simple graphs.*

*Proof.* Let $N_i$ be the number of connected spanning subgraphs of $G$ with $i$ edges. We know that computing $N_i$ is NP-hard [10]. We show that if there is an algorithm polynomial in the number of edges to compute the Shapley-Shubik index of all edges in the graph, then each $N_i$ can be computed in polynomial time.

We obtain graph $G_0$ by the following transformation: for some node $v \in V(G)$, we link it by a new edge $x$ to a new node $v_x$. Then, by the definition of the Shapley-Shubik value, $\sum_{r=0}^{m} r!N_r(|E(G)| - r)! = \sum_{r=0}^{m} r!N'_r = \kappa_x(G_0)$, where we write $N'_r$ for $N_r(m-r)!$, for all $r$.

Similarly we can construct $G_i$ by adding a path $P_i$ of length $i$ to $v_x$ where $P_i$ has no edge or vertex intersection with $G$. Therefore

$$\sum_{r=0}^{m}(r+i)!N'_r = \kappa_x(G_i). \tag{1}$$

For $i = 0, \ldots, m$, we get an equation of the form of (1) for each $G_i$. The left-hand side of the set of equations can be represented by an $(m+1)\times(m+1)$ matrix

$A$ where $A_{ij} = (i + j - 2)!$. The set of equations is independent because $A$ has a non-zero determinant of $(1!2! \cdots m!)^2$ (see e.g. Theorem 1.1 [5]). If there is a polynomial time algorithm to compute the Shapley-Shubik index of each edge in a simple graph, then we can compute the right-hand side of each equation corresponding to $G_i$.

The biggest possible number in the equation is less than $(2m)!$ and can be represented efficiently. According to Stirling's formula, $m! \approx \sqrt{2\pi m}\left(\frac{m}{e}\right)^m$, the number $(2m)!$ can be represented by $km(log\ m)$ bits where $k$ is a constant. We can use Gaussian elimination to solve the set of linear equations in $\mathcal{O}(m^3)$ time. Moreover, each number that occurs in the algorithm can also be stored in a number of bits quadratic of the input size (Theorem 4.10 [21]). Therefore SCG-SS is NP-hard. □

**Comment 1.** *A representation of a simple game is considered* reasonable *if, for a simple game $(N, v)$, the new game $(N \cup \{x\}, v')$ where $v(S) = 1$ if and only if $v'(S \cup \{x\}) = 1$, can also be represented. Then the proof technique in Proposition 2 can be used to show that for any reasonable representation of the simple game, a polynomial time algorithm to compute the Shapley-Shubik indices implies a polynomial time algorithm to compute the Banzhaf indices. This answers (positively) the question from [3] of whether computing Shapley-Shubik indices for a simple game represented by the set of minimal winning coalitions is NP-hard.*

**Proposition 3.** *For a simple game represented by its minimal winning coalitions, computing the Shapley-Shubik indices is NP-hard.*

*Proof.* This follows from Comment 1 and the fact that computing the Banzhaf values for a simple game represented by its minimal winning coalitions is NP-hard [3]. If Shapley-Shubik indices can be computed, then this implies that number of winning coalitions can be computed.

## 5 Polynomial Time Cases

In this section, we present polynomial time algorithms to compute voting power indices for restricted graph classes including graphs with bounded treewidth. We first consider the trivial case of a tree. If the graph $G = (N, E)$ is a tree then there is a total of $n - 1$ edges and only the grand coalition of edges is a winning coalition. Therefore a tree is equivalent to a unanimity game. This means that each edge has a Banzhaf index and Shapley-Shubik index of $\frac{1}{n-1}$. In the case of the same tree structure but with multiple parallel edges, we refer to this multigraph as a *pseudo-tree*.

**Proposition 4.** *Let $G = (N, E, s)$ be a pseudo-tree such that the underlying edges are $1, \ldots, m$ with multiplicities $s_1, \ldots, s_m$. Then,*

$$\eta_{i_1} = \prod_{\substack{j=1 \\ j \neq i}}^{m} (2^{s_j} - 1), \text{ and so } \beta_{i_1} = \frac{\eta_{i_1}}{\sum_{k=1}^{m} s_k \eta_{k_1}}. \tag{2}$$

*Proof.* Note that $m = n - 1$ in this case. Suppose edge $i_1$ is a parallel edge corresponding to edge $i$ in the underlying graph. Edge $i_1$ is critical for a coalition $C$ if the coalition $C$ contains no edges parallel to $i_1$ but contains at least one sub-edge corresponding to each edge other than $i$. The number of such coalitions is $\prod_{\substack{j=1 \\ j\neq i}}^{m}(2^{s_j} - 1)$, which gives (2). □

**Proposition 5.** *Let $G = (N, E, s)$ be a pseudo-tree such that the underlying edges are $1, \ldots, m$ with multiplicities $s_1, \ldots, s_m$ where $s = \sum_{i=1}^{m} s_i$. Then the Shapley-Shubik indices can be computed in time polynomial in the total number of edges.*

*Proof.* Denote by $e_r$ the coefficient of $x^r$ in

$$\sum_{\substack{1\leq j\leq n-1 \\ j\neq i}} ((1 + x)^{s_j} - 1).$$

Then $e_r$ is the number of coalitions with $r$ edges which include at least one parallel edge for each underlying edge $j$ except $i$. Then, by definition of the Shapley-Shubik value, for $1 \leq k \leq s_i$,

$$\kappa_{i_k}(G) = \sum_{r=n-2}^{s-s_i} e_r r!(s - r - s_i)!.$$

Thus, the Shapley-Shubik indices can be computed in time polynomial in the total number of edges. □

We now consider graphs with bounded treewidth. Note that trees and pseudo-trees hve treewidth 1.

**Definition 7.** *For a graph $G = (V, E)$, a* tree decomposition *is a pair $(X, T)$, where $X = \{X_1, ..., X_n\} \subset 2^V$, and $T$ is a tree whose nodes are the subsets $X_i$ with the following properties:*

1. $\bigcup_{1\leq i\leq n} X_i = V$
2. *For every edge $(v, w) \in E$, there is a subset $X_i$ that contains both $v$ and $w$.*
3. *If $X_i$ and $X_j$ both contain a vertex $v$, then all nodes $X_z$ of the tree in the path between $X_i$ and $X_j$ also contain $v$.*

*The* width *of a tree decomposition is the size of its largest set $X_i$ minus one. The* treewidth *$tw(G)$ of a graph $G$ is the minimum width among all possible tree decompositions of $G$.*

**Proposition 6.** *If the reliability polynomial defined in Definition 6 can be computed in polynomial time, then the following problems can be computed in time polynomial in the number of edges:*

1. *the number of connected spanning subgraphs;*
2. *the Banzhaf indices of edges.*

*Proof.* We deal with each case separately.

1. By definition, $N_i$ is the number of connected spanning subgraph with $i$ edges. If all coefficients $N_i$ are computable in polynomial time, then the total number of connected spanning subgraphs $\sum_{i=0}^{m} N_i$ is computable in polynomial time.
2. We know that $\eta_i(G) = 2\omega_i(G) - \omega(G)$ (See [16]) where $\omega(G)$ is equal to the total number of winning coalitions and $\omega_i(G)$ is the number of winning coalitions including player $i$. Consider the graph $G$ where the probability of edge $i$ being operational is set to 1 whereas the probability of other edges being operational is set to 0.5. Then the reliability of the graph being connected is equal to the ratio of the number of connected spanning subgraphs that include edge $i$ to $2^{M-1}$, the total number of subgraphs that include $i$. Therefore, $\omega_i(v)$ the number of connected spanning subgraphs including edge $i$ can be computed in polynomial time too. □

**Corollary 1.** *Banzhaf indices of edges can be computed in polynomial time for graphs with bounded treewidth.*

*Proof.* This follows from the polynomial time algorithm to compute the reliability of a graph with treewidth $k$ for some fixed $k$ [2]. □

**Definition 8.** *Let $G = (V, E)$ be a graph with source $s$ and sink $t$. Then $G$ is a* series-parallel *graph if it may be reduced to $K_2$ by a sequence of the following operations:*

1. *replacement of a pair of parallel edges by a single edge that connects their common endpoints;*
2. *replacement of a pair of edges incident to a vertex of degree 2 other than $s$ or $t$ by a single edge.*

Graphs with bounded treewidth can be recognized in polynomial time [1]. Series-parallel graphs and 2-trees are well-known classes of graphs with constant treewidth. Other graph classes with bounded treewidth are cactus graphs and outer-planar graphs. We see that whereas computing Banzhaf values of edges in general SCGs is NP-hard, important graph classes can be recognized and their Banzhaf values computed in polynomial time.

When edges have special properties, their power indices may be easier to compute. We define a *bridge* in the graph to be an edge whose removal results in the graph being disconnected. A graph class is *hereditary* if for every graph in the class, every subgraph is also in the class.

**Proposition 7.** *If graph $G$ belongs to a hereditary graph class, for which the reliability polynomial of a graph can be computed in polynomial time, then the Shapley-Shubik index of a bridge can be computed in time polynomial in the total number of edges.*

*Proof.* Let graph $G = (V, E)$ be a graph where edge $k$ is a bridge which connects two components $A = (V_A, E_A)$ and $B = (V_B, E_B)$. Then $|E| = |E_A| + |E_B| + 1$.

If the reliability polynomial of $G$ can be computed in polynomial time, then the reliability polynomial for each of the components $A$ and $B$ can be computed. Then the Shapley-Shubik index of player $k$ is:

$$\phi_k(G) = \frac{\sum_{i=|V_A|-1}^{|E_A|} \sum_{j=|V_B|-1}^{|E_B|} N_i(A) N_j(B)(i+j)!(|E_A|+|E_B|-i-j)!}{|E|!}.$$

□

Our next result is that if the reliability of a simple graph can be computed then the Banzhaf indices of the corresponding multigraph can be computed. A naive approach would be to compute the Banzhaf values of each edge in a simple graph and then, for the corresponding parallel edges in the multigraph, divide the Banzhaf value of the overall edge by the number of parallel edges. However, as the following example shows, this approach is incorrect:

*Example 1.* Let $G = (V, E, s)$ be the multigraph in Figure 1. Then, $\eta_{4_1}(v_G) = 10$, $\eta_{1_1}(v_G) = 14$, and $\eta_2(v_G) = \eta_3(v_G) = 28$. Therefore $\beta_{4_1}(v_G) = \frac{10}{3\times10+2\times14+28+28} = \frac{5}{57}$. Moreover, $\beta_{1_1}(v_G) = \frac{7}{57}$ and $\beta_2(v_G) = \beta_3(v_G) = \frac{14}{57}$. If we examine the underlying graph of $G'$ in Figure 1, then $\eta_4(v'_G) = 4$ and $\eta_1(v'_G) = \eta_2(v'_G) = \eta_3(v'_G) = 2$ giving $\beta_4(G') = 2/5$ and $\beta_i(G') = 1/5$ for $i = 1, 2, 3$. Therefore, the Banzhaf values of edges in the underlying graph do not give a direct way of computing the Banzhaf values in the multigraph.

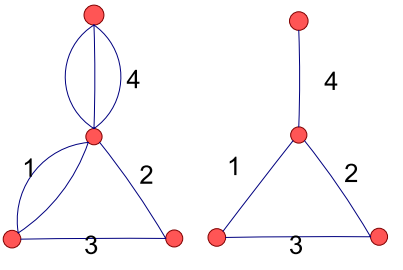



**Fig. 1.** Multigraph and its underlying graph

**Lemma 1.** *If there is an algorithm to compute the reliability of the underlying simple graph, then the algorithm can be used to compute the reliability of the corresponding multigraph.*

*Proof.* Let $G = (V, E, s)$ be a multigraph in which there are $s_i$ parallel edges $i_1, \ldots, i_{s_i}$ corresponding to edge $i$. Let $p_{i_j}$ be the probability that the $j$th parallel edge of edge $i$ is operational. In that case $Rel(G, p)$ is equal to $Rel(G', p')$, where $G'$ is the corresponding simple graph of $G$ and the probability $p_i$ that edge $i$ is operational is $1 - \prod_{j=1}^{s_i}(1 - p_{i_j})$. □

We now prove in Proposition 8 that if there is an algorithm to compute the reliability of the underlying simple graph $G$, then it can be used to compute

the Banzhaf indices of the edges in the corresponding multigraph of $G$. It would appear that the proposition follows directly from Lemma 1 and Proposition 6. However, one needs to be careful that the reliability computed is the reliability of the overall graph. Example 1 shows that computing the Banzhaf values of the edges in the underlying simple graph does not directly provide the Banzhaf values of the parallel edges in the corresponding graph.

**Proposition 8.** *For a multigraph $G$ and edge $i$, let $G'$ be the multigraph where all the other edges parallel to edge $i$ are deleted. Then if the reliability of $G'$ can be computed in polynomial time, then the Banzhaf value of edge $i$ in $G$ can be computed directly by analysing $G'$.*

*Proof.* Recall that $G$ is a multigraph with a total of $M$ edges. Given an algorithm to compute the reliability of $G'$, we provide an algorithm to compute the Banzhaf values of the parallel edges of edge $i$ in $G$. For graph $G'$, set the operational probabilities of all edges to 0.5 except $i$ which has an operation probability of $1 - 0.5^{s_i}$. and compute the overall reliability $r(G')$ of the graph. Then, by Lemma 1, $\omega(G)$ is $2^M r(G')$.

Now for $G'$, set the operational probabilities of all edges to 0.5 except $i$ which has an operation probability of 1. Let the reliability of $G'$ with the new probabilities be $r'(G')$. We see that $r'(G')$ is equal to $\omega_i(G')/2^{M-s_i}$. Then $\omega_i(G) = 2^{s_i-1}\omega_i(G') = 2^{M-1}r'(G')$. The Banzhaf value of $i$ is then $2\omega_i(G) - \omega(G)$. A similar approach gives Banzhaf values of other edges from which all the Banzhaf indices can be computed. □

## 6 Other Power Indices

Apart from the Banzhaf and Shapley-Shubik indices, there are other indices which are also used. Both the Deegan-Packel index [15] and the Holler index [20] are based on the notion of minimal winning coalitions. Minimal winning coalitions are significant with respect to coalition formation. The Holler index, $H_i$ of a player $i$ in a simple game is similar to the Banzhaf index except that only swings in minimal winning coalitions contribute toward the Holler index.

**Definitions 9.** *We define the* Holler value $M_i$ *as* $\{S \in W^m : i \in S\}$. *The* Holler index *(also called* the public good index*) is defined by* $H_i(v) = \frac{|M_i|}{\sum_{j \in N} |M_j|}$. *The* Deegan Packel index *for player $i$ in voting game $v$ is defined by* $D_i(v) = \frac{1}{|W^m|} \sum_{S \in M_i} \frac{1}{|S|}$.

**Proposition 9.** *For SCGs corresponding to multigraphs, Holler indices and Deegan-Packel indices can be computed in polynomial time.*

*Proof.* We use the fact that the number of trees in a multigraph can be computed in polynomial time, which follows from *Kirchhoff's matrix tree theorem* [18]. Given a connected graph $G$ with $n$ vertices, let $\lambda_1, \lambda_2, ..., \lambda_{n-1}$ be the non-zero eigenvalues of the Laplacian matrix of $G$ (the Laplacian matrix is the difference

of the degree matrix and the adjacency matrix of the graph). Kirchhoff proved that the number of spanning trees of $G$ is equal to any cofactor of the Laplacian matrix of $G$ [18]: $t(G) = \frac{1}{n}\lambda_1\lambda_2\cdots\lambda_{n-1}$. So now that we have a polynomial-time method to compute the number of spanning trees $t(G)$ of graph $G$, we claim this is sufficient to compute the Holler values of the edges. If an edge $i$ is a bridge, then it is present in every spanning tree and its Holler value is simply the total number of spanning trees. If $i$ is not a bridge then $M_i = t(G) - t(G \backslash e)$. Moreover, since the size of every minimal winning coalition is the same, namely $(n-1)$, the Holler indices and Deegan Packel Indices coincide for an SCG. □

## 7 Conclusion

This paper examined fairness-based cooperative game solutions of SCGs, for allocating resources to edges. In another recent paper, we have also looked at the computation of stability based cooperative game solutions of SCGs. A polynomial time algorithm is presented to compute the nucleolus [4]. This is a surprising result considering that the standard power indices are NP-hard to compute and also that the SCG is not convex in general. Therefore, the nucleolus may be a better alternative for resource allocation in SCGs.

We looked at the exact computation of power indices. In [6], an optimal randomized algorithm to compute Banzhaf indices and Shapley-Shubik indices with the required confidence interval and accuracy is presented. Since the analysis in [6] is not limited to a specific representation of a coalitional game, it can be used to approximate Banzhaf indices and Shapley-Shubik indices in SCGs.

The results of the paper are summarized in Table 1. This framework can be extended to give an ordering on the importance of nodes in the graph [12]. To convert a resource allocation to edges to one on nodes, the payoff for an edge is divided equally between its two adjacent nodes. The total payoff of a node is the sum of the payoffs it gets from all its adjacent edges. This gives a way to quantify and compare the centrality or connecting role of each node. It will be interesting to understand the properties of such orderings, especially for unique cooperative solution concepts such as the nucleolus, Shapley-Shubik and Banzhaf indices.

The complexity of computing the Shapley-Shubik index for an SCG with a graph of bounded treewidth is open. If this problem is NP-hard, it will answer

**Table 1.** Complexity of SCGs

| Problem | Input | Complexity |
|---|---|---|
| SCG-BANZHAF-VALUE | Simple, bipartite, planar graph | #P-complete |
| SCG-BANZHAF-INDEX | Simple graph | ? |
| SCG-BANZHAF-(VALUE/INDEX) | Multigraph with bounded treewidth | P |
| SCG-SS | Multigraph | NP-hard |
| SCG-SS | Multigraph with bounded treewidth | ? |
| SCG-H-(VALUE/INDEX) | Multigraph | P |
| SCG-DP-(VALUE/INDEX) | Multigraph | P |

the question posed in the conclusion of [6] on whether there are any domains where computing one of the Banzhaf index and Shapley-Shubik index is easy, whereas computing the other is hard.

## Acknowledgements

Partial support for this research was provided by DIMAP (the Centre for Discrete Mathematics and its Applications), which is funded by the UK EPSRC under grant EP/D063191/1. Rahul Savani also received partial support from EPSRC grant EP/D067170/1. Haris Aziz would also like to thank the Pakistan National ICT R&D Fund for funding his research.

## References

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods 8(2), 277–284 (1987)
2. Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Appl. Math. 23(1), 11–24 (1989)
3. Aziz, H.: Complexity of comparison of influence of players in simple games. In: Proceedings of the Second International Workshop on Computational Social Choice (COMSOC 2008), pp. 61–72 (2008)
4. Aziz, H., Lachich, O., Paterson, M., Savani, R.: Wiretapping: the nucleolus of connectivity (submitted, 2009)
5. Bacher, R.: Determinants of matrices related to the Pascal triangle. J. Théor. des Nombres Bordeaux 14, 19–41 (2002)
6. Bachrach, Y., Markakis, E., Procaccia, A.D., Rosenschein, J.S., Saberi, A.: Approximating power indices. In: AAMAS 2008: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, pp. 943–950 (2008)
7. Bachrach, Y., Rosenschein, J.S.: Computing the Banzhaf power index in network flow games. In: AAMAS 2007: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, pp. 1–7. ACM Press, New York (2007)
8. Bachrach, Y., Rosenschein, J.S., Porat, E.: Power and stability in connectivity games. In: AAMAS 2008: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems, pp. 999–1006 (2008)
9. Bailey, R.A.: Surveys in Combinatorics. Cambridge University Press, Cambridge (1997)
10. Ball, M.O.: Computational complexity of network reliability analysis: An overview. IEEE Transactions on Reliability 35(3), 230–239 (1986)
11. Borm, P., Hamers, H., Hendrickx, R.: Operations research games: A survey. TOP: An Official Journal of the Spanish Society of Statistics and Operations Research 9(2), 139–199 (2001)
12. Brandes, U., Erlebach, T.: Network Analysis: Methodological Foundations. Springer, Heidelberg (2005)
13. Crama, Y., Leruth, L.: Control and voting power in corporate networks: Concepts and computational aspects. European Journal of Operational Research 178(3), 879–893 (2007)

14. Curiel, I.: Cooperative Game Theory and Applications: Cooperative Games Arising from Combinatorial Optimization Problems. Springer, Heidelberg (1997)
15. Deegan, J., Packel, E.: A new index of power for simple n-person games. International Journal of Game Theory 7(2), 113–123 (1978)
16. Dubey, P., Shapley, L.S.: Mathematical properties of the Banzhaf power index. Mathematics of Operations Research 4(2), 99–131 (1979)
17. Faliszewski, P., Hemaspaandra, L.A.: The complexity of power-index comparison. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 177–187. Springer, Heidelberg (2008)
18. Godsil, C., Royle, G.: Algebraic Graph Theory. Springer, Heidelberg (2001)
19. Granot, D., Maschler, M.: Spanning network games. International Journal of Game Theory 27(4), 467–500 (1998)
20. Holler, M.: Forming coalitions and measuring voting power. Political Studies 30(2), 262–271 (1982)
21. Korte, B., Vygen, J.: Combinatorial Optimization: Theory and Algorithms, 3rd edn. Springer, Germany (2006)
22. Valiant, L.G.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8(3), 410–421 (1979)
23. van den Nouweland, A., Tijs, S., Maschler, M.: Monotonic games are spanning network games. International Journal of Game Theory 21(4), 419–427 (1993)

# Efficiently Generating $k$-Best Solutions to Procurement Auctions

Andrew Byde, Terence Kelly, Yunhong Zhou*, and Robert Tarjan

Hewlett-Packard Laboratories
Palo Alto, California, USA
{andrew.byde,terence.p.kelly,robert.tarjan}@hp.com,
yunhong.zhou@gmail.com

**Abstract.** Procurement executives often find it difficult to articulate their preferences and constraints regarding auctions, making it difficult to cast procurement decisions as straightforward optimization problems. This paper presents an efficient algorithm to aid decision support in such situations. Instead of trying to compute a single optimal solution for the auction winner determination problem, we generate many candidate solutions in ascending order of buyer expenditure. Standard techniques such as clustering and dominance pruning can then trim this list to a compact yet diverse menu of alternatives; other analyses can illuminate the cost of constraints and the competitive landscape. Our efficient solution-generation algorithm addresses sealed-bid procurement auctions with multiple suppliers and multiple types of goods available in multiple units. It supports multi-sourcing and volume discounts/surcharges in bids. Our algorithm may optionally incorporate certain classes of hard constraints, generating only solutions that satisfy them.

## 1 Introduction

The problem of clearing a sealed-bid auction—i.e., determining how goods and payments change hands among participants as a function of auction rules and bids—is conventionally known as the *winner determination problem* (WDP). For most kinds of auctions it is easy to define the WDP as a straightforward optimization problem, e.g., an integer linear program [1]. In practice, however, it can be difficult for auction participants to supply all of the inputs required to solve the WDP, particularly the constraints that define the space of permissible solutions and the preferences that allow a WDP solver to select the best solution.

Our primary focus in this paper is on the buyer's decision problem in sealed-bid procurement auctions, also known as reverse auctions. A procurement executive given seller bids in such an auction might "know the right solution when she sees one." However if she cannot articulate its *properties* in terms of hard constraints and soft tradeoffs among conflicting desiderata, a straightforward optimization formulation of the WDP does not by itself allow the auction to be cleared.

* Currently at Rocket Fuel, Inc.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 68–84, 2009.

Existing decision-support techniques such as scenario navigation and preference elicitation extend an optimization framework by requiring additional inputs from the decision maker, e.g., replies to elicitation queries. This paper considers an alternative framework that provides the buyer in a procurement auction with additional outputs rather than demanding additional inputs. Specifically, we use seller bids to generate a large list of alternatives from the most promising region of the solution space: the solutions that entail minimal buyer expenditure.

The key to our approach is generating $k$-best (i.e., $k$-cheapest) solutions to the auction WDP. An earlier paper described experiments based on real bids submitted to a real procurement auction; the results demonstrated the usefulness of our $k$-best solutions approach [2]. The present paper presents a far more efficient solution-generation algorithm, shows how to incorporate into the same decision framework the risk of supplier failure to deliver, and theoretically analyzes the relationship between solution rank $k$ and buyer expenditure.

Our overall approach is in principle applicable to auctions other than procurement auctions (see Section 2), but its efficiency depends on the precise form of the WDP considered—an unavoidable fact due to the NP-hardness of solving general WDPs [3]. When specialized for procurement auctions, in which portions of a procurement order must be assigned among sellers such that the total order is filled exactly, our algorithm can scale to practical problem sizes. We can incorporate certain kinds of hard constraints to prevent unacceptable solutions from being generated, and our approach allows multi-sourcing and the expression of volume discounts and surcharges in bids.

Once generated, the $k$-cheapest solutions to a procurement auction can be post-processed in several helpful ways. Ordinal preferences over solution attributes enable dominance pruning that yields a smaller Pareto frontier of solutions. The buyer may also reduce the number of candidate solutions by clustering them and considering only the cheapest in each cluster. Furthermore, the $k$-best solutions define prices on bundles of constraints: The price of any bundle of constraints satisfied by a generated solution is the cost difference between the cheapest satisfying solution and the cheapest unconstrained solution. These prices can focus the buyer's attention on the auction's most pressing tradeoffs. No restrictions on the mathematical form of constraints or preferences are necessary; arbitrary non-linearities pose no special difficulties. Finally, the $k$-cheapest solutions admit a wide range of informative visualizations. See [2] for a more detailed discussion of post-processing, including empirical results.

## 2 General Approach

Before refocusing attention on procurement auctions in Section 3, we briefly consider the fully general case of arbitrary sealed-bid combinatorial auctions/exchanges. This very general context makes it easy to sketch the basic ideas underlying our approach and explain why the restrictions of less general WDPs are necessary to obtain a computationally efficient solution generator.

The basic recipe for generating $k$-best solutions to any WDP follows from linking four observations [2]:

1. the WDP in combinatorial exchanges is a generalized knapsack problem [4];
2. dynamic programming can solve such problems [5];
3. dynamic programs are equivalent to shortest path problems [6]; and
4. we can generate the $k$ shortest paths in a graph [7].

We can therefore construct a graph whose *paths* correspond to WDP solutions and whose path *lengths* correspond to objective function values in the WDP optimization problem (e.g., path lengths might represent buyer expenditure in a procurement auction). By computing $k$-shortest paths on this graph we obtain $k$-best solutions to the WDP.

Unfortunately, while this approach is straightforward, there is good reason to believe that it cannot be computationally tractable for the fully general case of arbitrary combinatorial auction/exchange WDPs: Merely computing the *first-best* solution to a combinatorial auction is NP-hard [3]; computing $k$-best solutions can be no easier.

We gain more detailed insight into the computational complexity of combinatorial exchange WDPs by considering four natural measures of problem size: the number of *types* of goods, the number of *units* of each good available, the number of participating agents, and the length of agent bids. The computational difficulty of solving a fully general combinatorial exchange WDP is remarkably modest in terms of three of these four measures: Practical solvers with pseudo-polynomial time and memory requirements are available if the number of types of goods is a small constant [4]. The parameter responsible for intractability in the fully general context is the number of types of goods.

In this paper we present an approach that achieves computational efficiency by restricting attention to a class of procurement auctions, defined precisely in Section 3. The most important benefit of the restrictions that differentiate our procurement auctions from the fully general case of arbitrary combinatorial exchanges is that the procurement WDP admits efficient solvers whose computational demands scale pseudo-polynomially in all problem size parameters, including the number of good types. The $k$-best solution generation algorithms that we present in this paper have computational demands quadratic in a granularity parameter that divides the seller's demand for each type of good into equal shares. In practical procurement auctions this parameter is reasonably small, so the quadratic cost is acceptable.

The remainder of this paper is organized as follows: Section 3 formalizes our class of procurement auctions. Section 4 presents a $k$-best-solutions algorithm for the case of unconstrained solutions. Section 5 expands the scope of our method to incorporate constraints at the local level (e.g., no seller may supply more than 80% of any one item) and at the global level (e.g., at least three sellers must be involved in the global solution). Section 6 describes an extension of the notion of "cost" from the obvious monetary interpretation to a risk-based interpretation. Section 7 briefly summarizes the results of experiments on real bids from an actual material-parts procurement auction [2], demonstrating that our approach yields useful insights for procurement executives. Section 8 analyzes the distribution of buyer cost among the $k$-cheapest solutions assuming that

supplier bids are random variables constrained in reasonable ways. Section 9 reviews related work, and we conclude in Section 10.

## 3 Procurement Auctions

Businesses increasingly obtain goods through procurement auctions. Such auctions account for tens of billions of dollars of HP's expenditures in recent years [8], and US firms spend hundreds of billions of dollars via procurement auctions per year. In practice, buyer preferences typically encompass non-price solution attributes and side constraints, e.g., a desire to have 2–4 suppliers for each type of good; XOR constraints on winners, e.g., "supplier B must be excluded if A is chosen"; constraints on the total number of winning sellers; constraints on the distribution of expenditure across sellers. Many of these constraints are motivated by risk-management concerns related to delivery failure or delay, a topic to which we shall return in Section 6. Furthermore many are "soft" in the sense that the buyer would waive them in exchange for sufficiently large savings.

### 3.1 Definitions and Notation

Let $S$ denote the number of *sellers*, a term that we will use interchangeably with *suppliers*; we assume that $S \geq 2$. Let $I$ denote the number of *items* (distinct types of goods) that the buyer wishes to acquire; the overall procurement auction consists of $I$ single-item sub-auctions that are cleared simultaneously. Global granularity parameter $Q$ specifies the number of *quantiles* (shares of an item) that bids offer to supply. If $Q = 4$, for instance, then bids offer to supply 25%, 50%, 75%, or 100% of the total number of demanded units of each item. In Section 4.3 we shall consider a more general case, in which the number of quantiles depends on the item $i$; it makes no difference to the construction or complexity, so for the sake of clarity we will assume until then that the total number of quantiles is uniform across single-item auctions.

A vector of quantity assignments $\mathbf{q} = \{q_{i,s}\}$ is a *solution* (or *outcome*) of the auction if the constraint $\sum_{s=1}^{S} q_{i,s} = Q$ is satisfied for all items $i$. Our objective will be to rank such solutions in order of some *cost* function:

$$c(\mathbf{q}) = \sum_{i=1}^{I} \sum_{s=1}^{S} B_{is}(q_{i,s}), \tag{1}$$

where $B_{is}(q)$ is any non-negative function that is calculable from an assignment of a given quantity of a given item to a given seller, and for which $B_{is}(0) = 0$. The canonical example is the amount of money that seller $s$ demands for providing $q$ quantiles of item $i$, but others definitions are useful (see Section 6). When we refer to "cheapest" we will implicitly mean cost in this general sense.

The data $(I, S, Q, B)$ specifies a procurement auction WDP. For a given auction we will construct a weighted, directed acyclic graph $G$ with special source and sink vertices `s` and `t` such that the $k$-shortest paths from `s` to `t` correspond

to the $k$ cheapest solutions to the WDP. Eppstein [7] demonstrates that given a graph $G$ with $n$ vertices and $m$ edges, the $k$ shortest paths can be calculated implicitly in time $O(m + n \log n + k)$. The $n \log n$ term comes from Dijkstra's algorithm [9] for constructing the tree of shortest paths from `s` to each other vertex; if such a tree has already been constructed Eppstein's algorithm takes time $O(m + k)$. Happily, since our graph is directed and acyclic, it is possible to construct the shortest-path tree in time $O(m)$, so that our graph's $k$ shortest paths can be found implicitly in time $O(m + k)$. To extract explicit representations takes additional time proportional to the number of edges in each path, for which we will derive good bounds in Section 4.2.

Graph construction is significantly simpler in the unconstrained case, so we discuss this first; a full discussion of the constrained case is presented in Section 5.

## 4 Unconstrained Solutions

It is natural to decompose the problem by item, constructing a sub-graph for each item (with the appropriate correspondence between shortest paths and cheapest solutions), and then chaining the sub-graphs together; a concatenation of paths will correspond to a multi-item solution, and its length to total cost.

### 4.1 Single-Item Sub-Graph

To construct a single-item graph $G$, we consider a set of vertices with coordinates $(s, q)$. We choose the source `s` to be the vertex $(0, 0)$, the sink `t` to be $(S, Q)$, and the intermediate vertices to be all those for which $s = 1, \ldots, S-1$ and $q = 0, \ldots, Q$. Given these vertices for $G$, we add an edge from $(s, q)$ to $(s+1, q+q')$ with label $q'$ and length $B_{s+1}(q')$, whenever both of these vertices are in $G$. This edge corresponds to an assignment of quantity $q'$ to seller $s + 1$.

The graph for $S = Q = 3$ is shown in Figure 1. Each edge corresponds to assigning 0, 1, 2 or 3 of the 3 available quantiles to a particular seller; the "length" of each edge is the corresponding bid $B_{is}(q)$. The reader can verify that there are exactly ten paths from `s` to `t` (edges are directed, left to right), corresponding to the ten ways of allocating 3 quantiles among 3 sellers. In general the total number of distinct paths through the unconstrained single-item graph can be shown to be $R(S, Q) = (Q + S - 1)!/(Q!(S - 1)!)$, which is well known to be the number of ways of placing $Q$ indistinguishable balls into $S$ distinguishable cells.

**Lemma 1.** *Each path in $G$ from `s` to `t` corresponds, via the edge labeling, to a non-negative integer solution $\mathbf{q}$ of the equation $\sum_{s=1}^{S} q_s = Q$, and vice-versa. Furthermore the length of this path is exactly the cost to the buyer of the outcome $\mathbf{q}$.*

*Proof.* Suppose a path in $G$ has edge labels $q_s$. By induction on $s$, any path starting at `s` with the labels $q_s$, $k = 1, \ldots, s$ must end at $(s, \sum_{k \leq s} q_k)$, so the fact that the sink vertex has label $(S, Q)$ proves the equation. For the converse

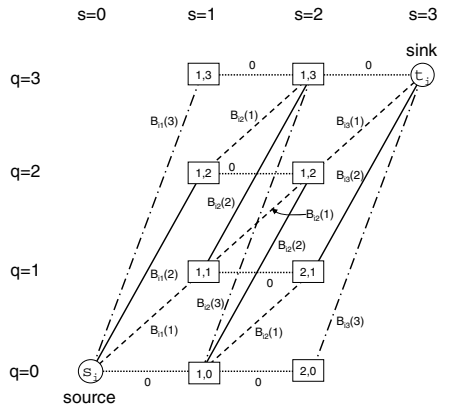



**Fig. 1.** Individual-item solutions graph for $S = Q = 3$. Each edge is directed, left to right. Finely dashed edges have label $q = 0$; roughly dashed edges have label $q = 1$; solid edges have label $q = 2$ and dash-cut edges have label $q = 3$. Edge lengths are shown next to each edge.

statement, the equation implies that the vertices $(s, \sum_{k \le s} q_k)$, $s = 0, \ldots, S$ are all necessarily in $G$; the path that links these vertices in order of $s$ clearly has edge labels $q_s$ by the definition of edge labels in $G$.

### 4.2 Multi-item Graph

Having understood the intuition behind a single-item subgraph, we present the formal definition of the unconstrained multi-item graph, as a concatenation of single-item graphs for each item:

**Definition 1.** *Let $(I, S, Q, B)$ be a WDP; the **unconstrained solutions graph** $G_u(I, S, Q, B)$ is defined as follows: The set of vertices is the set of tuples $(i, s, q)$, where $i = 1, \ldots, I$ and either*

- $(s, q) = (0, 0)$*; or*
- $s = 1, \ldots, S-1$ *and* $q = 0, \ldots, Q$*; or*
- $(s, q) = (S, Q)$.

*The set of edges is constructed by adding an edge from $(i, s, q)$ to $(i, s+1, q+q')$ with label $q'$ and length $B_{is+1}(q')$ whenever both vertices are in $G_u$; we also add connecting edges from $(i, S, Q)$ to $(i+1, 0, 0)$ for each $i = 1, \ldots, I-1$, with no label and length 0. We identify $(1, 0, 0)$ as the source $\mathtt{s}$ of $G_u$, and $(I, S, Q)$ as the sink $\mathtt{t}$.*

**Proposition 1.** *There is a one-to-one correspondence between solutions to the WDP of an auction $(I, S, Q, B)$ and paths in $G_u(I, S, Q, B)$ from $\mathtt{s}$ to $\mathtt{t}$.*

*Proof.* The proposition is a simple consequence of Lemma 1, since $G_u$ is a concatenation of single-item graphs.

**Complexity.** Each single-item sub-graph of $G_u$ has $(Q+1)(S-1)+2 = O(SQ)$ vertices and

$$(S-2)\left(\frac{(Q+1)(Q+2)}{2}\right) + 2(Q+1) \;=\; O(SQ^2)$$

edges. It follows that $n = O(I\,S\,Q)$, and $m = O(I\,S\,Q^2)$, so that the complexity of implicitly finding the $k$-shortest paths is $O(I\,S\,Q^2 + k)$. To explicitly extract a path requires additional computation in proportion to the number of edges in the path [7], which is $I \times S$, so explicit enumeration of the $k$-shortest paths takes time $O(I\,S(Q^2 + k))$.

### 4.3 The General Problem

For real-world auctions, instead of a fixed number of quantiles for all items, there is an exact number of units to acquire for each item. Quantile is a heuristic we use to obtain reasonable approximate solutions by dividing the number of units for each item into the same number of quantiles (rounding if necessary). In this section we consider *the general problem* where items have various desired number of units, and denote the procurement problem where all items have the same number of quantiles as *the simplified problem*.

For item $i$, let $Q_i$ denote the total number of units desired, for $i = 1, \ldots, I$. If $Q_i = Q$ for all $i$ the general problem degenerates into the simplified problem. For each item $i$, $Q_i$ is no longer very small, and it could be exponentially large (e.g., a large computer firm usually needs to procure millions of units for each computer part). For the general problem, we can solve it as in Section 4. We now require $\sum_{s=1}^{S} q_{i,s} = Q_i$ for each item $i$. We construct the single-item subgraph $G_i$ for each item $i$, identical to the construction in Section 4.1. Thus $G_i$ has $O(SQ_i)$ vertices and $O(SQ_i^2)$ edges. The multi-item graph $G$ is a concatenation of single-item subgraphs $G_i$ for all $i$, thus it has $O(S\sum_i Q_i)$ vertices and $O(S\sum_i Q_i^2)$ edges. Using Eppstein's algorithm, we obtain:

**Theorem 1.** *The general procurement problem (demanding $Q_i$ units of item $i$) is pseudo-polynomial-time solvable. We can compute the $k$-cheapest solutions in time $O(S\sum_i Q_i^2 + k)$ implicitly and time $O(S\sum_i Q_i^2 + kSI)$ explicitly.*

## 5 Constrained Solutions

Some constraints on a full solution can be imposed within the framework described in Section 4 by simply removing some edges from graph. Because the full graph is a chain of single-item graphs, any constraint that can be incorporated in this way can be factorized into constraints on each single-item auction, and for this reason we call them "local". An example of a local constraint is that no seller provide more than 80% of any item; this can be represented by removing all edges whose label $q$ is greater than $0.8Q$.

Since the incorporation of local constraints is so straightforward, we will turn our attention to *hard* constraints, whose satisfying global solutions are *not* the product of restricted sets of individual-item outcomes; the canonical example, which has great practical importance for risk management, is that of restricting the number of sellers involved in the global solution.

### 5.1 Hard Constraints

This section describes an approach for modifying the simple graph representation of Section 4 to incorporate certain types of hard constraints, in the sense that solutions that violate the constraints are not generated when the $k$-shortest paths algorithm operates on the modified graph. We call the expanded graph that encodes global constraints a *constrained solutions graph*. The method of this section is not generally efficient, but several useful global constraints do have efficient representations; we enumerate some in Section 5.3.

In the process of expanding the graph to permit structural representation of complex constraints we inevitably increase the complexity of the $k$-shortest paths algorithm, and so the question arises as to whether it is better to do so, or to generate a larger list of candidate solutions more quickly and filter out those that violate the constraints. In practice the approach described in this section scales best when the constraint is most stringent—i.e. when the proportion of all paths failing the constraint is significant. This is in contrast to approaches in the literature, such as in Villeneuve & Desaulniers [10], that rely on forbidding a relatively small set of paths, whose computation time scales with the set of *forbidden* paths rather than the set of *permitted* paths.

### 5.2 Constrained Solutions Graph

In this section we consider the problem of constructing a graph all of whose paths correspond to solutions satisfying some constraint $\mathcal{C}$. Our method is, as in Section 4, to construct a graph $G_c$ with vertices indexed by item-seller-quantity triples, but now with an auxiliary variable, $x \in X$, which represents the "state" of the solution so far constructed. By restricting those edges that are added to $G_c$ on the basis of their state, we can exclude paths that are bound to violate the constraint. $X$ will therefore represent the intermediate states in the evaluation of the acceptability of an outcome as the outcome is constructed by assigning quantities to suppliers.

We can formalize this description in the following way. We consider the edges $edges(G_u)$ of the unconstrained graph, and because of the need to bootstrap the evaluation of the auxiliary variable, pay particular attention to those edges $edges(\mathtt{s})$ that originate at the source vertex $\mathtt{s}$. A **representation function** is defined to be a tuple $(X, f)$, such that $f : edges(\mathtt{s}) \cup \big(X \times \big(edges(G_u) \setminus edges(\mathtt{s})\big)\big) \to X$ is a function mapping a source edge, or a non-source edge and a state value, to another state value. A representation

function can be extended from edges to paths starting at $\mathtt{s}$, by repeated application: for any sequence of edges $e_1, \ldots, e_m$ starting at $\mathtt{s}$ we define

$$f(e_1, \ldots, e_m) := f(f(\ldots f(f(e_1), e_2), \ldots, e_{m-1}), e_m) \tag{2}$$

By construction, paths in the unconstrained graph from $\mathtt{s}$ to $\mathtt{t}$ are in one-to-one correspondence with solutions to the WDP; we say that a function $(X, f)$ and the set of final states $X_{\mathtt{t}}$ **represents** a constraint $\mathcal{C}$ if the set of solutions obeying the constraint corresponds in this way to exactly the set of paths $(e_1, \ldots, e_m)$ such that $f(e_1, \ldots, e_m) \in X_{\mathtt{t}}$.

**Definition 2.** *Let $(I, S, Q, B)$ be a WDP, and $(X, f, X_{\mathtt{t}})$ a representation of a constraint $\mathcal{C}$ on the set of acceptable solutions. The **constrained solutions graph** for this WDP, $G_c(I, S, Q, B, X, f, X_{\mathtt{t}})$, is defined as follows:*

1. *Let the vertices of $G_c$ be $\mathtt{s} \cup \big(X \times G_u \setminus \{\mathtt{s}\}\big)$, with a special sink vertex $\mathtt{t}$ added (here $G_u(I, S, Q, B)$ is the unconstrained graph defined in Definition 1);*
2. *For each source edge $e$ in the unconstrained graph, from $\mathtt{s}$ to $v'$, add an edge in $G_c$ from $\mathtt{s}$ to $(f(e), v')$, with the label and weight of $e$;*
3. *For each non-source edge $e$ in the unconstrained graph, from $v$ to $v'$, and each state $x$, add an edge in $G_c$ from $(x, v)$ to $(f(x, e), v')$; and*
4. *Add an edge of weight zero between $(I, S, Q, x)$ and $\mathtt{t}$ whenever $x \in X_{\mathtt{t}}$.*

**Proposition 2.** *If $(X, f, X_{\mathtt{t}})$ represents a constraint $\mathcal{C}$, then there is a one-to-one correspondence between solutions to the WDP $(I, S, Q, B)$ satisfying $\mathcal{C}$, and paths in $G_c$ from $\mathtt{s}$ to $\mathtt{t}$.*

*Proof.* It is clear from 2 and 3 that any path in $G_c$ from $\mathtt{s}$ to $\mathtt{t}$ corresponds to a sequence of edges $e_1, \ldots, e_m$ in $G_u$ along which $f$ is iteratively evaluated; the penultimate vertex in $G_c$ must therefore be $(I, S, Q, f(e_1, \ldots, e_m))$. By the definition of the fact that $f$ represents the constraint, a solution obeys the constraint if and only if the corresponding path in the unconstrained graph satisfies $f(e_1, \ldots, e_m) \in X_{\mathtt{t}}$, which by 4 is true if and only if the corresponding path in $G_c$ goes from $\mathtt{s}$ to $\mathtt{t}$. Therefore there is a one-to-one correspondence between paths in $G_c$ from $\mathtt{s}$ to $\mathtt{t}$, and solutions to the WDP satisfying the constraint.

**Complexity.** It is obvious from the construction that the complexity of incorporating a constraint via a representation with state set $X$ is a factor of $|X|$ worse than that in Section 4.2. Thus the implicit cost is $O(|X|\, I\, S\, Q^2 + k)$ and the explicit cost is $O(|X|\, I\, S(Q^2 + k))$.

### 5.3 Examples

In this section we detail a selection of global constraints of increasing complexity, and their corresponding representations and constrained solutions graphs.

**Worst Case.** The first thing to notice is that *every* global constraint has a representation: Let $X$ be the set of all paths originating at $\mathtt{s}$ in $G_u$, and define $f(x, e)$ to be the concatenation of the edge $e$ onto the end of $x$, if it is defined, or the empty path otherwise. Clearly every path starting at $\mathtt{s}$ is mapped by repeated application of $f$ to a unique element of $X$, namely itself. For an arbitrary set of acceptable outcomes we can therefore let $X_{\mathtt{t}}$ be the set of corresponding paths from the unconstrained graph; only these paths will be connected to the sink vertex in $G_c$, and so only solutions obeying our arbitrary constraint will be generated by the $k$-shortest paths method in $G_u$. This representation is not useful, however, because the number of new vertices and edges required to create its constrained solutions graph scales very poorly.

**Constrained Number of Winners.** In order to clarify the general definitions in Section 5.2 above, in this section we give an explicit representation of the important constraint that the number of sellers allocated non-zero quantities in the global solution should lie in some range.

Suppose that our goal is to bound this number above by some value $\Sigma_f$. We proceed by letting $X$ be the collection of sets of sellers with at most $\Sigma_f$ elements, with a special element $fail$ to denote that the constraint is violated. We define the representation functions $f(e)$ and $f(x, e)$ as follows:

- If $e$ is a source edge, and has label $q > 0$ then $f(e) := \{s_1\}$, otherwise $f(e)$ is the empty set.
- If $e$ is not a source edge,
  - If it has zero weight, then define $f(x, e) = x$;
  - Otherwise the edge ends at a vertex of the form $(i, s, q)$; if $x \cup \{s\} \in X$ then define $f(x, e) := x \cup \{s\}$; otherwise define $f(x, e) := fail$.

As quantities are allocated to sellers, the state keeps an accurate record of the set of sellers so far allocated non-zero quantity, transitioning to state $fail$ if the number of sellers ever gets too high. To represent an upper bound on the number of sellers it is sufficient to let $X_{\mathtt{t}} = X \setminus \{fail\}$. A *lower* bound of $\sigma_f$ is representable by using $\Sigma_f = \sigma_f - 1$ in the above, and $X_{\mathtt{t}} = \{fail\}$ (only those solutions that fail to use less than or equal to $\sigma_f - 1$ sellers are acceptable). We can impose upper and lower bounds simultaneously by using the state set from the upper bound, and choosing $X_{\mathtt{t}} = \{x \in X : |x| \geq \sigma_f\}$. Because lower bounds thus have multiple representations, choosing a representation wisely is in general a tricky matter.

**Monotonic Predicates.** The most important feature of the cardinality constraint example in Section 5.3 is that the global constraint is evaluated over predicates of the form "is seller $s$ assigned non-zero total volume?" Such predicates have two very nice properties: Most importantly, they can be evaluated incrementally at each step by a simple OR over whether the seller has yet been included and whether the seller is included at the current step. This implies that the space of states need be no larger than $2^{|S'|}$, where $S'$ is the set of sellers under consideration in the global constraint. For example, a representation of

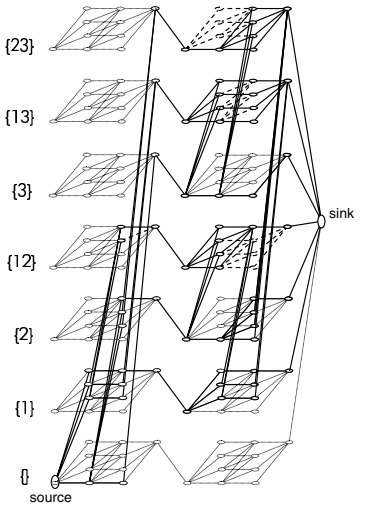



**Fig. 2.** A representation of the constrained graph $G_c(2, 3, 3, B, X)$ for state variable $x$ equal to the set of suppliers so far included in the solution. Dashed edges lead to a fail state, not shown; a copy of the unconstrained graph based on concatenated copies of Figure 1 for each element $x \in X$ is shown, grayed out, for reference.

the constraint "Either seller 1 is included, or seller 2 is included, but not both" exists with $|X| = 4$.

Secondly, the value of the predicate is monotonic in the sense that as quantities are assigned to sellers, once the predicate is true, it will remain true in all subsequent steps. This fact sometimes gives straightforward upper bounds on the state sets. For example, for the canonical representation of "Either seller 1 is included, or seller 2 is included, but not both", the state sets will clearly never contain a state in which both seller 1 and seller 2 are included: it is not necessary to wait until step $I$ to realize this. If the constraint had been "Either seller 1 is assigned an even number of shares, or seller 2 is assigned an even number of shares, but not both", this would not have been possible. Similarly, it is this monotonicity that allows the upper bound on $X$ in Section 5.3.

**Constraints on a second metric.** We can impose an arbitrary constraint with respect to a second cost metric $c'$ (i.e. one expressible as a sum of edge weights $B'$, as in Equation 1) by maintaining it as a state variable $x = c'$: updates to the state are calculable at the edge level by the fact that the cost is a sum of edge values.

In this case $X$ covers all reachable values of $c'$, and so might potentially be very large. A constraint is enforced in the straightforward way, by letting $X_{\mathtt{t}}$ be the set of values of the second metric that are acceptable.

**Monotonic constraints on a second metric.** If the second cost metric is *monotonic* in the sense that the edge weights $B'$ in Equation 1 are all of one sign,

and if the constraint is an inequality $c' \leq C$, $c' \geq C$, etc., then the representation of Section 5.3 can be simplified.

Without loss of generality, assume that the second cost metric is positive. For the case of an upper bound on a positive second cost metric, $c' \leq C$, we can restrict $X$ to be the reachable values of $c'$ that are also less than or equal to $C$, and add a failure state $fail$ as in Section 5.3. Then the constraint representation either accumulates $c'$ in the expected way, or diverts to $fail$ if the constraint is violated: the monotonicity of $c'$ guarantees that once broken the constraint is always broken. The set of final values $X_{\mathtt{t}}$ is equal to $X \setminus fail$.

Symmetrically, for the case of a lower bound on a positive second cost metric, $c' > C$, we can use the same state space, with the failure state replaced by a success state, *success*: the set of final values is then just $X_{\mathtt{t}} = \{success\}$.

**Quantized thresholds as a solution filter.** In Section 5.3, most interesting secondary cost metrics will have very many possible states. Even in cases where this multiplicity makes the full constrained graph impractical to construct, we can still construct a graph whose solutions all satisfy the constraint, but which excludes some solutions that do not. This reduces the computational burden on a final filtering stage, possibly at low cost in terms of the original graph construction.

The basic idea is to quantize the secondary edge costs $B'$ by some step parameter $\delta$. If $c'$ is a positive monotonic secondary cost metric, and the constraint is an upper bound $c' \leq C$, then by rounding all values of $B'$ *down* to the nearest multiple of $\delta$, we can assure ourselves of a smaller space of possible second metric values, while remaining confident that any path excluded corresponds to a solution violating the constraint. For a lower bound we round bid values up instead.

## 6 Pareto Optima Minimizing Both Cost and Risk

We have presented the objective function with respect to which solutions are ranked as being the monetary cost of procuring a particular bundle of quantities from various sellers, but another interesting example is provided by letting the cost of such an assignment be the negative logarithm of the probability of failure:

$$B_{is}(q) = -\log\left(Pr(\text{seller } s \text{ delivers}|\text{quantity} = q, \text{item} = i)\right). \tag{3}$$

Then the total cost of an assignment is a measurement of the *risk of failing to obtain all quantities required* (under the assumption of independence between deliveries). If we care only about minimizing risk of delivery, then it is equivalent to minimizing the total length of a path where $B_{is}(q)$ is defined in Equation 3. And of course we can compute *k-safest* solutions using the same overall approach that we have heretofore employed for computing $k$-cheapest solutions. However, since both monetary cost (expenditure) and delivery risk are important considerations in procurement auctions, it is natural to consider the bi-criteria

optimization problem minimizing both expenditure and risk, and to seek an algorithm to enumerate the Pareto optimal solutions in order of one metric or the other.

Next we show how to compute the Pareto optima path set. Write the unconstrained graph from Section 4 in terms of its vertices and edges: $G = G_u(I, S, B, Q) = (V, E)$. Recall from Section 4.2 that the number of vertices and edges are bounded by $n = O(ISQ)$ and $m = O(ISQ^2)$. For each edge $e \in E$, denote its cost $c_1(e)$ and risk $c_2(e)$. For each vertex $v \in V$ and any non-negative cost value $c \leq V_{\max}$, we use $L_v[c]$ to store the minimum risk distance from the source to vertex $v$ with cost distance exactly $c$. Here $V_{\max}$ is the maximum cost distance for any path from the source, corresponding to the maximum procurement cost to satisfy demand for all items. $L_v$ is an array with length $V_{max}$. It is easy to initiate $L_s$ for the source vertex $\mathtt{s}$. Next we use dynamic programming to compute $L_v[c]$ as follows:

$$L_v[c] = \min\{L_u[c - c_1(u, v)] + c_2(u, v) \mid (u, v) \in E\}.$$

The set of Pareto optima paths can be extracted from the array $L_{\mathtt{t}}$, where $\mathtt{t}$ is the destination vertex, through a linear walk of $L_{\mathtt{t}}$ from least to highest cost: the first Pareto path is given by the minimum $c$ value with $L_v[c]$ defined; at step $c$, if $L_v[c]$ is defined and $L_v[c]$ is smaller than the risk distance of any of the stored Pareto paths, the path corresponding to $L_v[c]$ is Pareto optimal, and we store it.

The total running time to compute the set of Pareto optima is dominated by the dynamic programming step to compute $L_v[c]$ for all $v \in V$, $c \leq V_{\max}$, and it takes time $O(mV_{\max})$ where $m = |E| = O(ISQ^2)$. This completes the description of our algorithm, which runs in time $O(ISQ^2V_{\max})$.

In general we cannot expect to do particularly well on the Pareto enumeration problem, because the procurement auction problem minimizing both expenditure and risk belongs to the bi-criteria shortest path problem, which is NP-hard based on a reduction from the *Partition Problem* (see Garey & Johnson [11], p. 214). The general multiple-objective shortest path problem is one of the most intensively studied problems in multiple-objective combinatorial optimization; here we mention only work most relevant to our setting. We are interested in computing the Pareto optima path set in an acyclic graph with two cost metrics. Henig [12] considered the bi-criteria optimization problem using a utility function to combine both metrics. For an acyclic graph with $n$ vertices, Warburton [13] gives a pseudo-polynomial-time exact algorithm (based on DP) with running time $O(n^2\hat{V}_{\max} \log n)$ where $\hat{V}_{\max}$ is the maximum possible path distance under both metrics. Using standard scaling and rounding techniques, it is converted into a fully-polynomial-time approximation scheme (FPTAS) to compute the *approximately efficient* Pareto optima set in time $O(n^3/\epsilon)$.

For our contributions, first, we show that the procurement auction problem minimizing both cost and risk can be modeled as bi-criteria shortest path problem in graph $G_u(I, S, Q, B)$. Second, we show that a dynamic programming approach simplifying the Walburton method can compute the set of Pareto optima in time $O(ISQ^2V_{\max})$.

## 7 Experiments

This section briefly summarizes the results of experiments described in detail in a preliminary paper [2] that employed an inefficient solution-generation algorithm inferior to the algorithm of the present paper.

We computed $k$-best solutions based on actual bids submitted to a multi-million-dollar, multi-item, multi-supplier HP material-parts procurement auction. We then explored the following questions:

1. Are the top $k$ solutions *affordable*?
2. Are the top $k$ solutions *diverse*?
3. Does *dominance pruning* aid multi-criteria decision problems?
4. Can our method *assign prices to bundles of side constraints*?

In all cases, our results were encouraging: The 100,000th-cheapest solution is only 0.054% more expensive than the 1st-cheapest solution, so considering $k$-best solutions is not prohibitively expensive for the buyer in this real procurement auction. Furthermore the $k$-best solutions are remarkably *diverse* in terms of how they apportion the buyer's expenditure across sellers, and moreover when we add randomly-generated volume discounts to the bids, this measure of diversity improves. When we consider the bi-criteria optimization problem in which the buyer wishes to minimize expenditure and also spread expenditure as evenly as possible across sellers, we find that the Pareto frontier of undominated solutions is small enough to admit consideration by a human decision-maker. Finally, our experiments show that our method can assign prices to bundles of side constraints, e.g., constraints on both the number of sellers included in a solution and the uniformity of expenditure across sellers.

## 8 Extreme Value Statistics

Our experiments have shown empirically that for a real auction, the $k$-cheapest solutions have very similar expenditure [2]. In this section we examine the same issue theoretically, by examining the important question of the probability, in the face of randomly distributed bids, of the cost of the $k$-cheapest solutions relative to the absolute cheapest. The question we therefore address is the likely tradeoff between cost and diversity in solutions.

For items $i = 1, \ldots, I$, let $X_i$ denote a random variable representing the total cost for item $i$; For $Q = 1$, $X_i$ denotes a random selection of supplier $j$ for all units of item $i$ with total cost $p_{ij}$. Let $Y = X_1 + X_2 + \ldots + X_I$, then $Y$ denotes the total procurement cost to obtain all the items with desired number of units. Let $a_i \equiv \min X_i$, $b_i \equiv \max X_i$, then $0 < a_i \leq b_i$ for each $i$, and $Y_{\min} = a_1 + a_2 + \ldots + a_I, \quad Y_{\max} = b_1 + b_2 + \ldots + b_I$. Here $Y_{\min}$ denotes the minimum cost to purchase all items with desired number of units, and it is the optimal solution for other solutions to compare with. We say a solution $Y$ is *$\epsilon$-approximately optimal* if $Y \leq (1+\epsilon)Y_{\min}$. Here we abuse the notation and use $Y$ to denote both the solution and its corresponding total cost.

**Assumption 1.** *$b_i \leq 2a_i$ for all items $i$.*

Given that bids are normally competitive, it is reasonable to assume that the highest unit-price for each item is at most twice as expensive as the lowest unit-price for each item. This implies that $Y_{\max} \leq 2Y_{\min}$. Suppose that items are sorted according to their minimum cost, i.e., $a_1 \leq a_2 \leq \ldots \leq a_I$, then

$$\frac{\sum_{i=1}^{\delta}(X_i - a_i)}{Y_{\min}} \leq \frac{\sum_{i=1}^{\delta} a_i}{\sum_{i=1}^{I} a_i} \leq \frac{\delta}{I}.$$

Pick $\delta$ such that $\delta \leq \epsilon I$, then any supplier selected for the first $\delta$ items together with minimum cost for items $i > \delta$ consist of a solution with total cost at most $(1+\epsilon)Y_{\min}$. Recall that $R = R(S,Q)$ is defined to be the total number of solutions for each item; for the special case of $Q = 1$, $R$ is equal to $S$, the total number of suppliers.[1] The total number of $\epsilon$-approximately optimal solutions is $R^{\delta} = R^{\epsilon I} = 2^{\epsilon I \log R}$.

**Assumption 2.** *There is an aggressive new entrant who matches the minimum price for a significant fraction of all the items.*

Suppose that new entrant $j_0$ matches the min-price for a fraction $f_0$ of all the items. For each of these items, there are at least two choices of suppliers with min-cost, thus the total number of min-cost solutions grows by a factor of $2^{f_0 I}$. Combining results using Assumptions 1 and 2, we obtain:

**Theorem 2.** *There are at least $R^{\epsilon I} 2^{f_0 I} = 2^{\epsilon I \log R + f_0 I}$ $\epsilon$-approximately optimal solutions assuming that for each item the max-cost bid is at most twice the min-cost one, and a new entrant matches the min-cost bid of existing suppliers for a fraction $f_0$ of all items.*

## 9 Related Work

Decision support in auctions is an important problem in practice and has inspired much research, primarily on *preference elicitation* and *scenario navigation*.

Preference elicitation techniques typically represent a decision maker's preferences as a latent utility function with a specified functional form and unknown coefficients, and then repeatedly query the decision maker to refine estimates of these coefficients (e.g., by asking her to choose between two alternatives). Preference elicitation is applicable to auctions [14], and can preserve privacy and shorten bids [15] and aid uncertain decision makers [16]. However, most approaches place strong restrictions on the mathematical form of the utility function and may require auction participants to reply to exponentially many queries. Furthermore, revealed preferences may be intransitive. Our approach does not suffer from any of these difficulties.

[1] The number of solutions for item $i$ is $\Theta(S^Q)$, thus much larger than $S$ when $Q > 1$.

Scenario navigation typically employs mixed-integer program solvers to find price-optimal solutions under different constraints. This approach requires the buyer in a procurement auction to specify a different set of constraints for each scenario—a potentially tedious exercise. By contrast, in its simplest form our method does not require explicit modeling of side constraints.

The close relationship between combinatorial auction/exchange WDPs and generalized knapsack problems is described in [4], which furthermore exploits this connection to develop a general multi-unit combinatorial exchange WDP solver that offers attractive computational properties. Specifically, the time and memory required are pseudo-polynomial (indeed, linear) in all problem parameters *except* that they are exponential in the number of good types. Our present contribution exploits the special properties of procurement auctions to achieve good scalability in terms of *all* problem-size parameters.

Eppstein [7] surveys $k$-shortest paths problems and algorithms, and indeed applies his shortest paths algorithm to the solution of the 0-1 knapsack problem, generating what can be seen as a special case of our single-item graph. Encoding constraints in graphs so that a $k$-shortest paths algorithm generates only satisfying paths has also been explored. Villeneuve and Desaulniers [10] describe an approach based on string-matching algorithms; as noted above, this method is not suitable for our problem. Coutinho-Rodrigues et al. employ $k$-shortest paths computations with interactive elicitation queries to explore the Pareto frontier in bi-criteria optimization [17].

To the best of our knowledge, ours is the first systematic method of generating $k$-best solutions to auction WDPs. An early paper described an inefficient solution generation algorithm that required exponential time and memory [2]. The present paper makes the overall method more practical by greatly improving the computational efficiency of solution generation.

## 10 Conclusions

This paper has described an efficient method for computing $k$-cheapest solutions to procurement auction WDPs. It supports multi-sourcing, volume discounts and surcharges, and it scales pseudo-polynomially (in fact at most quadratically) with respect to all problem size parameters. Furthermore, the constrained solutions graph can accommodate many useful global hard constraints with only a modest increase in computational complexity. We have presented analytical results on the number of "reasonably cheap" solutions, complementing previous empirical results addressing the same issue. Finally, we have shown that $k$-safest solutions may be computed using the same framework as $k$-cheapest solutions, and we have presented an algorithm for computing solutions on the Pareto frontier of the bi-criteria cost/risk problem. Taken together, the contributions of this paper enable a promising approach to decision support for practical procurement auctions.

## Acknowledgments

We thank Kemal Guler for early support and encouragement. Alex Zhang and Claudio Bartolini also provided valuable feedback.

## References

1. Andersson, A., Tenhunen, M., Ygge, F.: Integer programming for combinatorial auction winner determination. In: Proc. 4th Int'l Conf. on Multi-Agent Systems (ICMAS), pp. 39–46 (July 2000)
2. Kelly, T., Byde, A.: Generating k-best solutions to auction winner determination problems. ACM SIGecom Exchanges 6(1), 23–34 (2006); Presents an inefficient generation algorithm and extensive experimental results; see [18] and the present paper for an efficient algorithm
3. Rothkopf, M.H., Pekec, A., Harstad, R.M.: Computationally manageable combinatorial auctions. Management Science 44(8), 1131–1147 (1998)
4. Kelly, T.: Generalized knapsack solvers for multi-unit combinatorial auctions. In: Faratin, P., Rodríguez-Aguilar, J.-A. (eds.) AMEC 2004. LNCS (LNAI), vol. 3435. Springer, Heidelberg (2006); Also available as HP Labs TR HPL-2004-21
5. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Heidelberg (2004)
6. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. Prentice-Hall, Englewood Cliffs (1993)
7. Eppstein, D.: Finding the k shortest paths. SIAM Journal on Computing 28(2), 652–673 (1998)
8. Beckett, J.: The business of bidding: Reinventing auctions for better results (September 2005), `http://www.hpl.hp.com/news/2005/jul-sep/auctions.html`
9. Dijkstra, E.W.: A note on two problems in connection with graphs. Numerische Mathematik 1, 83–89 (1959)
10. Villeneuve, D., Desaulniers, G.: The shortest path problem with forbidden paths. European Journal of Operations Research 165, 97–107 (2005)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
12. Henig, M.: The shortest path problem with two objective functions. European Journal of Operational Research 25(2), 281–291 (1985)
13. Warburton, A.: Approximation of Pareto optima in multiple-objective, shortest-path problems. Operations Research 35(1), 70–79 (1987)
14. Cramton, P., Shoham, Y., Steinberg, R. (eds.): Combinatorial Auctions, ch. 10. MIT Press, Cambridge (2006)
15. Lahaie, S.M., Parkes, D.C.: Applying learning algorithms to preference elicitation. In: Proc. ACM E-Commerce Conf., 180–188 (May 2004)
16. Boutilier, C., Sandholm, T., Shields, R.: Eliciting bid-taker non-price preferences in (combinatorial) auctions. In: Proc. AAAI (2004)
17. Coutinho-Rodrigues, J., Climaco, J., Current, J.: An interactive bi-objective shortest path approach: searching for unsupported nondominated solutions. Computers & Operations Research 26, 789–798 (1999)
18. Byde, A., Kelly, T.: Efficiently generating k-best solutions for procurement auctions. Technical Report HPL-2006-145, HP Labs (October 2006); Presents a more efficient generation algorithm than that of [2]

# Integer Polyhedra for Program Analysis

Philip J. Charles[1], Jacob M. Howe[2], and Andy King[1,3]

[1] University of Kent, Canterbury, CT2 7NF, UK
[2] City University London, EC1V 0HB, UK
[3] Portcullis Computer Security Limited, Pinner, HA5 2EX, UK

**Abstract.** Polyhedra are widely used in model checking and abstract interpretation. Polyhedral analysis is effective when the relationships between variables are linear, but suffers from imprecision when it is necessary to take into account the integrality of the represented space. Imprecision also arises when non-linear constraints occur. Moreover, in terms of tractability, even a space defined by linear constraints can become unmanageable owing to the excessive number of inequalities. Thus it is useful to identify those inequalities whose omission has least impact on the represented space. This paper shows how these issues can be addressed in a novel way by growing the integer hull of the space and approximating the number of integral points within a bounded polyhedron.

## 1 Introduction

The aim of this work is to take algorithms from computation geometry and linear programming and apply them to solve problems arising in program analysis using polyhedra. In abstract interpretation convex polyhedra have long been used to abstract the sets of values that variables may take [6]. This has proven to be attractive in program analysis because, as well as prescribing range constraints on variables, polyhedra can also describe interactions between variables.

Polyhedral analyses sometimes need to consider integrality [16], for instance, to derive invariants between integral objects such as loop counters and pointer offsets [20]. In such analyses variables are discrete, whereas polyhedra are defined over real or rational numbers. Further, polyhedra cannot express non-linear relationships; in this case, the non-linearity is either projected out or approximated in an ad hoc way. These drawbacks impede the accurate analysis of programs. In terms of tractability, polyhedra can be too expressive in some situations; an analysis can become overwhelmed by large systems of (non-redundant) inequalities. This paper presents a synthesis of solutions to the three problems introduced above: integrality, non-linearity and tractability.

The target of this work is abstract interpretation based analyses, such as those performed by [5]. In such an analysis a fixpoint in the meet semi-lattice of polyhedra over the variables of interest is calculated, where this fixpoint describes the values and relationships between program variables. The smaller the fixpoint (when the polyhedra is interpreted as a set of points), the more information it contains. In particular, if the polyhedra describes the values of integers

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 85–99, 2009.

(and no floating point variables), then tightening to the integer hull provides a systematic way of strengthening an analysis.

The starting point of the work is that a polyhedron can be grown to describe the integer solutions of a system of constraints. The process is incremental in nature. First an integer solution to the system of constraints is found. Then a second distinct integer solution is found whose distance is maximal from the first and the convex hull of this point and the previous space is taken. Iterating this mechanism, one of the inequalities that bounds the current space is chosen and a solution is found at maximal normal distance from the inequality. This process is repeated until all inequalities have been considered, giving the integer hull [10]. Computing the integer hull for arbitrary systems of even linear inequalities is NP-hard, limiting the size of problems likely to be solvable and motivating approximation techniques. Observe that the technique above gives a series of integer polyhedra approximating the solution from below, converging on the precise solution. It will also be seen that an approximation from above can be extracted from the algorithm. It is important to note that the input set of constraints is not necessarily linear, thus this approach addresses two of the three problems: integrality and non-linearity.

A potential drawback of the above technique is that the resulting integer polyhedron may involve an unmanageably large number of inequalities. This motivates a systematic technique for relaxing a polyhedron by reducing the number of inequalities. This is achieved by calculating a Monte Carlo approximation of the number of integer points that a constraint bars from a polyhedron. The least contributing constraints are relaxed. This approach provides a way of curbing the growth of inequalities and computing an integer approximation whose number of defining constraints does not exceed some bound, addressing the problem of tractability.

This paper brings together a number of threads in program analysis and computational geometry and the contributions of the paper are summarised:

- The algorithm of [10] that grows the integer hull and allows anytime approximation from below and above is presented and elucidated.
- This algorithm can be adapted to calculate the integer hull of a projection of the input constraint system onto a subset of its variables. When running to completion this method can be used for over-approximating the integer solutions of a set of non-linear constraints, an approximation problem which thus far has not been satisfactorily addressed.
- A method to determine which constraints contribute little to the enclosed space, hence are candidates for relaxation, is presented. This is parameterised by the method used to determine this contribution and a Monte Carlo approximation technique is discussed in detail.
- The integer hull algorithm and one approach to relaxation have been implemented and the results of an empirical evaluation are presented. The results are promising for the use of the algorithms in program analysis.

## 2 Growing Integer Hulls

This section details the calculation of the convex hull of the integer solutions – the so-called integer hull [18] – of a system of constraints $C$ defined over totally ordered variables $x_1, \ldots, x_n$. The integer hull is approximated from below, growing it from a point by giving an inequality $c$ to an oracle that will return a point $p$ satisfying $C$, but not $c$; inequality $c$ bounds the current hull, $W$. The convex hull of $p$ and $W$ is then calculated. This approach was first seen in [10] and is also remarked upon in [4]. Here, the algorithm is detailed with particular attention given to the maintenance of the inequalities describing the hull. Further attention is paid to the novel use of this algorithm in the context of program analysis, especially the way in which it can deal with non-linear constraints.

### 2.1 An Integer Hull Algorithm

The following three procedures detail the integer hull algorithm implemented in Section 4. The main loop of the algorithm is contained in the second procedure, worklisthull. The first procedure, integerhull below, sets up the problem:

```
 1: procedure integerhull(C)
 2: p := maximise(C, x_1);
 3: p = null then return null; end if
 4: Ineqs := {x_i ≤ p_i, −x_i ≤ −p_i | 1 ≤ i ≤ n};
 5: Ps := {p};
 6: Cons := sort(Ineqs);
 7: lastrank := 0;
 8: rank := rank(Ineqs);
 9: while Cons ≠ [] do
10:    p' := null;
11:    while Cons ≠ [] ∧ p' = null do
12:       Cons ≡ f :: Rest, f ≡ c · x ≤ d;
13:       p' := maximise(C ∧ ¬f, c · x);
14:       Cons := Rest;
15:    end while
16:    if p' = null then
17:       return Ineqs;
18:    else
19:       lastrank := rank;
20:       Ineqs' := convexhull(p', Ineqs);
21:       rank := rank(Ineqs');
22:       if rank > lastrank then
23:          Ineqs := Ineqs';
24:          Cons := sort(Ineqs);
25:          Ps := Ps ∪ {p'};
26:       else
27:          return worklisthull(Ps, Ineqs, C);
28:       end if
29:    end if
30: end while
```

The purpose of this procedure is to calculate a first approximation (importantly, a simplex) of the integer hull that reaches the dimension of the final solution. Note that the integer hull might well be a hyperplane of lower dimension than $n$, the number of varibles. On line 2, a first integer point in the hull is calculated. This uses the auxiliary function maximise$(C, c)$ that takes a system of constraints $C$ and a cost function $c$ and returns an integer solution to $C$ that maximises $c$. If no such point exists it returns *null*. The choice of the first cost function is arbitrary. $Ineqs$ is a set of linear inequalities describing the current approximation and $Ps$ is the set of points so far calculated. $Ineqs$ is then sorted by the number of points in $Ps$ that lie on the boundary of an inequality. This ensures that the next discovered point will raise the dimension, if full

dimensionality is not yet reached. The dimension of a set of inequalities is determined by function rank. The next point is determined on line 13 and (the topological closure of) the convex hull of this point with the previous hull is calculated on line 20 using an appropriate method. This process is repeated through lines 9 to 30 until either the integer hull is calculated or full dimensionality is reached.

Before passing on to worklisthull, it is worth noting that replacing lines 18-28 (and the rank variables and calculations) of integerhull with

$Ineqs :=$ convexhull$(\boldsymbol{p}', Ineqs)$;
$Cons :=$ sort$(Ineqs)$;

will give a complete algorithm not using the following two procedures. This will be referred to later as integerhull′. This is essentially what is given in [10] and the additional procedures detail the use of simplicial faces to control the generation of new inequalities (which in integerhull′ result from the call to convexhull).

Procedure worklisthull is passed a set of points, a set of inequalities describing their integer hull and the input constraints. This procedure works on a simplicial input and the final point calculated by the integerhull is not included in the set of points; worklisthull provides the main loop and is given below:

```
 1: procedure worklisthull(Ps, Ineqs, C)
 2: Hull = dimred(Ps, Ineqs);
 3: Worklist = faces(Ps, Ineqs);
 4: while Worklist ≠ φ do
 5:    f(Vs, ineq) ∈ Worklist;
 6:    p := maximise(C ∧ ¬ineq, c · x) where ineq ≡ (c · x ≤ d);
 7:    if p = null then
 8:       Hull := Hull ∪ {ineq};
 9:       Worklist := Worklist \ {f(Vs, ineq)}
10:    else
11:       Ps := Ps ∪ {p};
12:       Worklist := hull(p, Worklist, Ps);
13:    end if
14: end while
15: return Hull;
```

In worklisthull a worklist of consists of faces, where a face $f(Vs, ineq)$ is a set of integer points $Vs$, with $|Vs|$ equal to the dimension of the integer hull, and inequality $ineq$ with each point in $Vs$ lying on the boundary of $ineq$. Each face is a simplex of dimension $|Vs| - 1$. $Hull$ represents the inequalities in the integer hull. It is initialised with any dimension reducing inequalities, determined by auxiliary dimred – every point in $Ps$ will be on the boundary of such an inequality. The auxiliary faces sets up the initial worklist. Whilst there are faces in the worklist a face $f(Vs, ineq)$ is selected and the oracle is asked for a point $\boldsymbol{p}$ satisfying $C$, but not $ineq$, line 6. If there is no such point, then $ineq$ is added to $Hull$, line 7. If there is, line 10, the procedure hull determines a new worklist, replacing any face not satisfied by $\boldsymbol{p}$ with a set of new faces whose determining

points will include the new point. Note that the call to hull will remove the current face from the worklist.

Procedure hull, below, takes the place of a convex hull calculation in worklisthull:

```
1: procedure hull(p, Worklist, Ps)
2: NewWorklist = φ;
3: for all f(Vs, ineq) ∈ Worklist do
4:   if p ⊨ ineq then
5:     NewWorklist := NewWorklist ∪ {f(Vs, ineq)};
6:   else
7:     for all v ∈ Vs do
8:       Vs' := (Vs \ {v}) ∪ {p};
9:       ineq' := ineq(Vs', v);
10:      if ∀q ∈ Ps.q ⊨ ineq' then
11:        NewWorklist := NewWorklist ∪ {f(Vs', ineq')};
12:      end if
13:    end for
14:  end if
15: end for
16: return NewWorklist;
```

The procedure will retain any face satisfied by the new point $\boldsymbol{p}$, line 4 ($\models$ denotes the satisfaction relation). An unsatisfied face forms the base of a simplicial cone whose pinacle is $\boldsymbol{p}$. The faces of this cone are the simplicies obtained by replacing one of the points defining the base by $\boldsymbol{p}$, line 8. The inequality for this new face can be calculated (see below), from these points, plus the discarded point, line 9. Finally, the worklist need only retain faces that are currently satisfied by all discovered points, others are discarded, line 10.

The plane through a set of $d$ independent points, $\boldsymbol{p_1}, ..., \boldsymbol{p_d}$, can be calculated in constant time for fixed $d$ by solving the parametric description of the plane using Gaussian elimination. That is, $plane = \boldsymbol{p_1} + \sum_{i=2}^{d} \lambda_i . vec(\boldsymbol{p_1}, \boldsymbol{p_i})$ where $vec(\boldsymbol{p_i}, \boldsymbol{p_j})$ is the vector from point $\boldsymbol{p_i}$ to point $\boldsymbol{p_j}$. Set up a matrix where the first $d-1$ columns are given by $vec(\boldsymbol{p_1}, \boldsymbol{p_{i+1}})$, the next $d$ columns are the unit vectors for each dimension and the final column is the entries of $\boldsymbol{p_1}$. Use Gaussian elimination to set the first $d-1$ entries of the final row to 0 and read off the equation of the plane from the entries in the remaining columns of this row. The discarded point can then be used to determine the desired inequality.

**Anytime Approximations.** During execution of worklisthull at any point the accumulated inequalities of $Hull$ and $Worklist$ determine an integer polyhedron that is an underapproximation of the integer hull, allowing anytime approximation from below. Further, note that at any point $Hull$ is a potentially unbounded polyhedron (but not necessarily an integer polyhedron) that is an over-approximation of the integer hull, allowing anytime approximation from above. Algorithms with anytime approximation are paricularly attractive for program analysis when attempting to bound the time the analysis takes. Both under and over-approximations are useful, depending the whether analysis is for properties that definitely hold, or potentially hold.

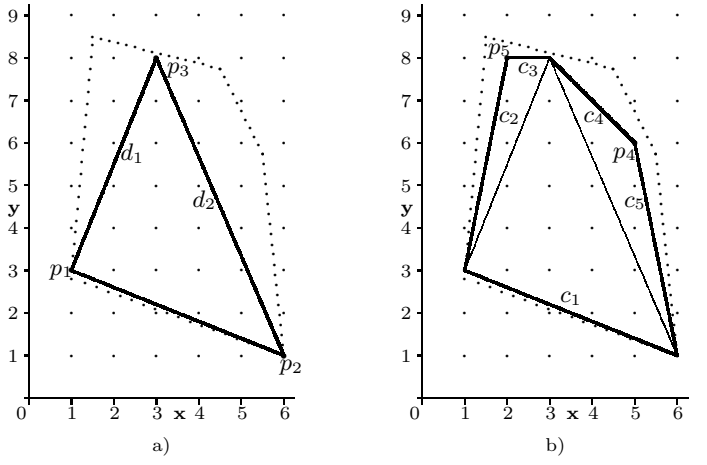



**Fig. 1.** Integer hull of a set of linear constraints

*Example 1.* Consider the following linear constraints, $C = \{-11x+y \leq -8, 2x+8y \leq 71, 8x+4y \leq 67, 19x+2y \leq 116, -4x-11y \leq -35\}$. This is represented by the dotted lines in Fig. 1. The initial call to maximise gives $p_1$, subsequent calls from integerhull give the points $p_2$ and $p_3$ and the simplicial under-approximation of the integer hull given by the continuous lines in Fig. 1 a).

In worklisthull, the face with inequality $c_1 \equiv -2x - 5y \leq -17$ will be first selected. The call maximise$(C, c_1)$ will return *null* and $c_1$ will be added to $Hull$. Next consider $face(\{(6,1),(3,8)\}, d_2)$, giving the call to maximise$(C, d_2)$ that will return $(5,6)$. In hull, the face with inequality $d_1$ is satisfied by the new point and will remain in the worklist. The new faces are $face(\{(5,6),(3,8)\}, c_4)$ and $face(\{(6,1),(5,6)\}, c_5)$. To determine $c_4$ consider the matrix on the left, which with one elimination step gives that on the right:

$$\begin{bmatrix} -2 & 1 & 0 & 5 \\ 2 & 0 & 1 & 6 \end{bmatrix} \qquad \begin{bmatrix} -2 & 1 & 0 & 5 \\ 0 & 1 & 1 & 11 \end{bmatrix}$$

This allows the result to be read off: $c_4 \equiv x+y \leq 11$ (the point (6,1) has been used to determine the inequality). Similarly, $c_5 \equiv 5x+y \leq 31$. Further iterations give $c_2 \equiv -5x + y \leq -2$ and $c_3 \equiv y \leq 8$. Since there are no further external points satisfying $C$ these will be added to $Hull$ which will finally be returned.

Note that a tightening has been achieved. The input $C$ projected onto $x$ gives range [0.984,6] whereas the integer hull gives [1,6], and for $y$ [1,8.5] becomes [1,8].

### 2.2 Working in a Projected Space and Non-linear Constraints

This section builds upon two observations on the algorithm presented in the previous section to highlight its suitability for use in program analysis. The first observation is that the algorithm is easily adapted to compute the integer hull of

a $k$-dimensional projection of constraints $C$, that is, the smallest polyhedron that contains those points $\langle v_1, \ldots, v_k \rangle$ for which $C$ possesses a corresponding integer solution $\langle v_1, \ldots, v_n \rangle$. Restriction to a subset of variables of interest is an operation commonly required for program analysis. The adaptation is simply achieved by restricting the points $\boldsymbol{p}$, determined by calls to maximise, to the variables of interest. The second observation is that $C$ may contain non-linear constraints, as long as the point oracle can deal with these. Both of these observations are illustrated with a well-known problem from program analysis.

Although the seminal paper on polyhedral analysis [6] identified the problem of approximating non-linear constraints, a widely accepted solution to the problem has not been found. Consider the example of [6, Sect. 4.2.1] to illustrate how to compute a polyhedral approximation of a non-linear assignment. Specifically, suppose the constraint $S = \{-x + y \leq 1, -y \leq -1, -x - y \leq -5\}$, holds when the non-linear assignment $y := xy$ is encountered. The problem is how to systematically compute a polyhedral approximation of the ensuing non-linear space. This problem can be addressed by augmenting $S$ with the constraint $y' = xy$, thereby raising the dimension, then symbolically projecting out $y$, and replacing $y'$ with $y$. This gives the shaded space in Fig. 2(a) defined by $\{-x \leq -2, y \leq x + x^2, -y \leq -x, -y \leq x^2 - 5x, y \leq 32767\}$. This approach presupposes that a symbolic projection algorithm is known for the system of augmented constraints, which of course, is not guaranteed in general [7]. Note that the $y \leq 32767$ constraint is imposed by an underlying 16-bit representation where variables range over $[-32768, 32767]$; other machine representations would likewise ensure that integer variables can only lie within a finite range. Note too that the manually derived non-linear constraint suggested in [6, Sect. 4.2.1] omits the inequality $-x \leq -2$ that is necessary to exclude the origin. This inequality follows from a linear combination of $-x+y \leq 1$ and $-x-y \leq -5$, and illustrates the subtlety of manually abstracting non-linear constraints.

Now consider a run of the algorithm where $C$ is instantiated to $S \wedge y' = xy$ and the variables are totally ordered as in the sequence $x, y', y$. Putting $k = 2$ then eliminates the variable $y$ so that the algorithm computes the integer hull of the projection of $C$ onto the $x, y'$ plane. An initial solution $\boldsymbol{u} = \langle 32767, 32767, 1 \rangle$ is computed at line 2 of integerhull. The projection of $\boldsymbol{u}$ onto the $x, y'$ plane is merely $\boldsymbol{u}' = \langle 32767, 32767 \rangle$ which can be represented as a system of inequalities $\{x \leq 32767, -x \leq -32767, y' \leq 32767, -y' \leq -32767\}$, which defines the polyhedra $P$ at line 2 and can be seen in Fig. 2(b).

On the first iteration $f$ is chosen to be $-x \leq -32767$, then the cost function at line 8 is $-x$. The net effect is to find the solution $\boldsymbol{v} = \langle 2, 6, 3 \rangle$ that minimises the $x$ coordinate whilst satisfying $C \wedge \neg f \equiv C \wedge x < 32767$. Projecting $\boldsymbol{v}$ onto the $x, y'$ plane yields $\boldsymbol{v}' = \langle 2, 6 \rangle$. Extending the polyhedra $P$ in Fig. 2(a) with this point by computing the convex hull at line 9 gives the line segment $\{-x \leq -2, x \leq 32767, 32765y' = 32761x + 131068\}$ depicted as $P$ in Fig. 2(b).

On the second iteration, $f$ is chosen to be $32765y' \leq 32761x + 131068$, leading to the triangle depicted in Fig. 2(c). At this stage $P$ has reached full dimensionality and worklisthull will be called. Here, a call to maximise with *ineq*

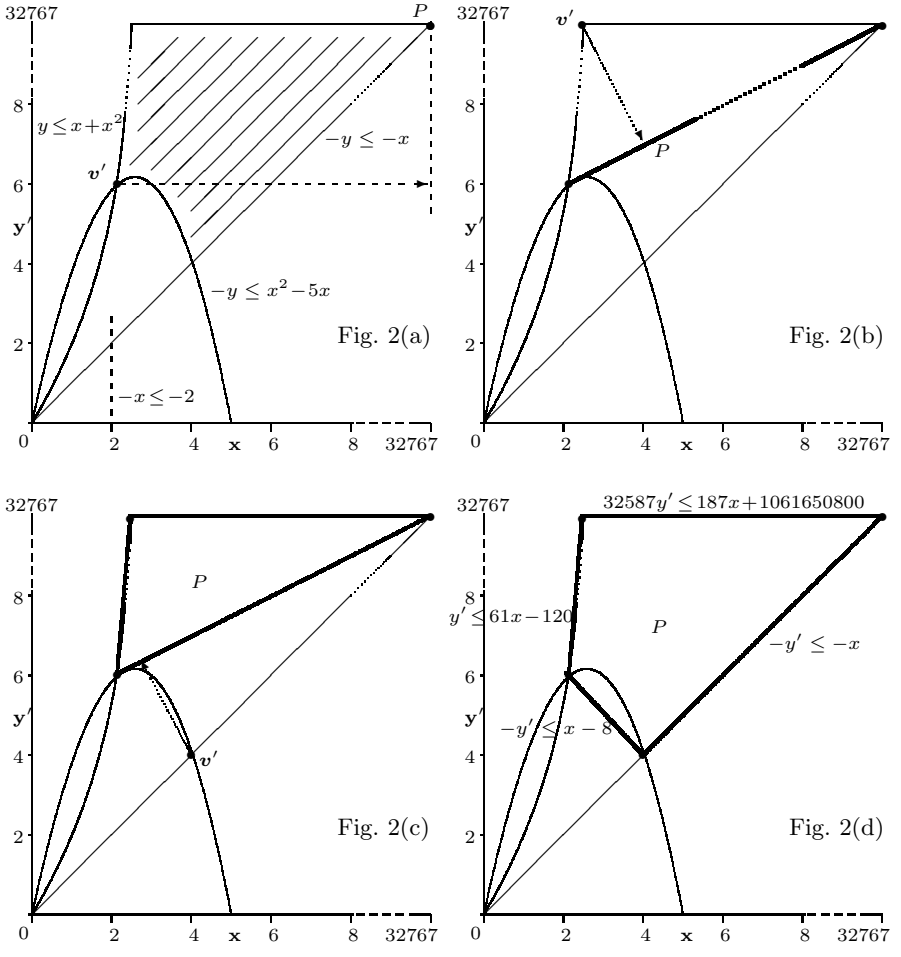



**Fig. 2.** Polyhedral approximation of a feasible region defined by non-linear constraints

as $-32765y' \leq -32761x - 131068$ will lead to the polyhedra in Fig. 2(d). After this, further iterations will fail to find further points and $P$ will be returned. Notice that $P$ is formulated in terms of $y'$ which represents the value of $y$ after the assignment. The state of the $x, y$ variables after the assignment is obtained by merely replacing $y'$ with $y$. Again notice that symbolic computation of the projection has been replaced by an integer hull calculation.

## 3 Curbing Growth of the Integer Hull

Earlier it was noted that the integer hull algorithm presented allows for anytime approximation from above and below. This is advantageous when the algorithm fails to perform quickly enough, however an approach that allows selected inequalities to be dropped is desirable for a different kind of problem. As the

polyhedron representing a solution space grows, the number of inequalities may also grow. This growth can be curbed by either relaxing inequalities in the final polyhedron or dropping them on-the-fly whilst the integer hull algorithm proceeds. These two approaches will henceforth be referred to as off-line and on-line integer hull relaxation. Here, the focus is on off-line relaxation. These techniques can be used to limit the growth of a system of inequalities, ameliorating any ensuing tractability issues (with possible precision cost).

Suppose that $S = \{0 \leq x, 0 \leq y, y \leq -x + 6\}$ describes the relative values of variables $x$ and $y$ when the non-linear assignment $y := xy$ is reached. Applying the integer hull algorithm to the system of non-linear constraints $S \wedge y' = xy$ with the variable ordering $x, y', y$ and $k = 2$ gives the projected integer hull, $H$, enclosing only those integer points that satisfy both the linear inequality $-y' \leq 0$ and the non-linear inequality $y' \leq -x^2 + 6x$. $H$ is defined by 7 inequalities $c_1$: $-y' \leq 0$, $c_2$: $y' \leq 5x$, $c_3$: $y' \leq 3x+2$, $c_4$: $y' \leq x+6$, $c_5$: $y' \leq -x+12$, $c_6$: $-3x+20$, $c_7$: $y' \leq -5x + 30$. Rank these inequalities according to some measure of their suitability for relaxation, and discard as appropriate. For example, if the ranking was $c_3, c_6, c_4, c_5, c_2, c_7, c_1$ then relaxing the highest two ranked would give a small increase the volume, but this slightly larger polytope contains no additional integer points. Reranking, this process could be continued to a final result given by $c_2, c_7, c_1$.

This approach is parameterised by the function ranking the inequalities. The method chosen here for ranking is to calculate a Monte Carlo approximation, [15], of the volume of $H \wedge \neg c_i$ that represents the increase in volume resulting from the relaxation. A bounding box is constructed and sampled until the sampling error ($\sigma/\sqrt{n}$, where $\sigma^2 = (r^2 + n^2)/(r + n)^2$, $r$ is the number of samples in the region and $n$ the number of samples not in the region) is beneath a given value. The proportion of the sample within the polytope multiplied by the volume of the bounding box gives an approximation to the volume, as required.

Alternative rankings are possible: the volume of a polytope (with rational vertices) can be computed in polynomial time [2] and, rather surprisingly, so can the number of integer points in such a polytope [3,8,22], which is exactly what is required when describing integral properties. However, despite their complexity these remain difficult problems, particularly in high dimension and sampling based methods seem more suitable to guiding the quick relaxation of constraints.

A natural generalisation of off-line relaxation is on-line relaxation which discards inequalities as soon as their number exceeds some pre-defined threshold in the main loop of the integer hull algorithm. This approach is problematic for the integerhull algorithm given earlier as relaxation will lead to faces whose vertices are not known and the hull method will not work. However, the integerhull′ method with its reliance on a more general convex hull algorithm can incorporate this – simply follow the call to convexhull with as many relaxation steps as required. Anecdotal evidence suggests that this might useful, particularly in discarding inequalities with large coefficients that are both problematic for performance and less likely to be useful for program analysis.

## 4 Experimental Evaluation

The algorithm described in Section 2 and the off-line approximation technique described in Section 3 were implemented and tested.

As mentioned in the introduction, the target of this work is abstract interpretation based analyses, where a fixpoint describing the values and relationships between program variables is calculated. This fixpoint is a point in the meet semi-lattice of polyhedra over the variables. Fixpoints arise because of loops: the values of the variables after an iteration of a loop serve as the values that are input to the next iteration. The semantic equations that express the values that variables can assume are thus recursive. A fixpoint of these equations can be interpreted as expressing invariants that hold over all iterations of a loop. The fixpoint may not necessarily be the unique least fixpoint; the requirement for correctness is merely that if the fixpoint summarises values that hold in one iteration, then it also summarises values that hold in the next. By tightening to integer polyhedra at certain analysis points the analysis is strengthened. The tightening to the integer hull could be applied with differing levels of granularity: after each domain operation, at the end of the fixpoint calculation, or after the analysis of each loop structure. The benchmarks best represent the last of these.

The benchmarks come from the Stanford Invariant Generator (Sting) and FAST [17,1]. The Sting analyser discovers linear invariants of transition systems that represent iterations of loops where all the variables are integer. The benchmarks are invariants generated by Sting and are representive of the program analysis problems that this work is aimed at. The implementation is in Java and the experiments were run on a single core of a MacBook with a 2.4GHz Intel Core 2 Duo processor and 4GB of memory.

**Calculating Integer Hulls.** The algorithm is coded in Java, with the oracle provided by the CBC MILP solver [13]. CBC is coded in C++ and called via the Java Native Interface. The integer hull is only defined for bounded problems and as noted earlier inequalities need to be augmented with variable bounds. In these experiments the problem constraints were augmented with variable bounds of [0,64]. The results can be seen in Fig. 3: for each named benchmark, Var gives the number of variables in the benchmarks, Ineqs the number of input constraints (including the variable bounds), Time gives the execution time in seconds for calculating the integer hull, Opts the number of calls to the ILP solver and Sol Size the number of inequalities in the integer hull.

The *barvinok* package for integer point counting [22] and the *Polylib* package for manipulating integer polyhedra [23] have been used to check the integer hull calculations given in this paper. *barvinok* has been run on the input constraints and the calculated integer hull to check that the number of lattice points are indentical for both and *Polylib* has been used to convert the calculated constraints to a vertices and rays representation, the test being that the vertices are all at integer points.

**Off-line Approximation.** Fig. 4 tabulates results from applying the Monte Carlo approximation of Section 3 to the results of the integer hull calculations.

| Problem | Var | Ineqs | Time | Opts | Sol Size |
|---|---|---|---|---|---|
| barber.inv | 8 | 29 | 3.417 | 207 | 15 |
| berkeley-nat.inv | 4 | 13 | 0.075 | 29 | 9 |
| berkeley.inv | 4 | 11 | 0.069 | 28 | 9 |
| cars.inv | 5 | 19 | 0.15 | 61 | 13 |
| efm.inv | 6 | 22 | 0.111 | 39 | 12 |
| efm1.inv | 6 | 21 | 0.086 | 19 | 9 |
| heap.inv | 5 | 16 | 0.08 | 26 | 10 |
| lifo-nat.inv | 7 | 24 | 0.297 | 87 | 13 |
| lifo.inv | 4 | 14 | 0.047 | 15 | 6 |
| robot.inv | 3 | 10 | 0.031 | 10 | 5 |
| scheduler-2p.invl1 | 7 | 27 | 0.15 | 46 | 15 |
| scheduler-2p.invl2 | 7 | 27 | 0.279 | 65 | 17 |
| scheduler-3p.invl1 | 10 | 40 | 10.881 | 273 | 42 |
| scheduler-3p.invl2 | 10 | 46 | 194.022 | 1037 | 125 |
| scheduler-3p.invl3 | 10 | 38 | 23.034 | 388 | 26 |
| see-saw.inv | 2 | 6 | 0.022 | 10 | 5 |
| swim-pool-1.inv | 9 | 32 | 0.255 | 62 | 16 |
| swim-pool.inv | 9 | 31 | 0.248 | 57 | 15 |
| train-beacon.invlate1 | 3 | 11 | 0.026 | 12 | 7 |
| train-beacon.invonbrake | 3 | 10 | 0.037 | 14 | 6 |
| train-beacon.invontime | 3 | 12 | 0.027 | 14 | 8 |
| train-beacon.invstopped | 3 | 11 | 0.026 | 12 | 7 |
| train-rm03.inv | 6 | 20 | 0.12 | 38 | 12 |

**Fig. 3.** Benchmarking of the integer hull algorithm

Benchmarks with no bounded relaxation have been omitted. The first set of results gives data on ranking and relaxing one constraint: T1 is the time taken in seconds, Best is the number of sample points needed to calculate the volume associated with the constraint dropped, Total is the total number of sample points and Max is the largest sample size needed to approximate a volume arising from a single constraint. The second set of results details relaxing as many constraints as possible, recalculating the ranking at each step: TM is the time taken in seconds, Cons is the number of constraints in the input, Size is the final number of constraints and Sam is total number of samples taken in this process.

**Discussion.** The results are promising. The implementation (not tuned to the problems) returns the integer hull for all benchmarks up to 10 dimensions in an acceptable time. These are the first experiments of this kind performed on program analysis benchmarks (indeed, the authors know of no integer hull benchmarking work at all). However, at 10 dimensions and beyond performance degenerates (the implementation was unable to solve nine further suitable benchmarks over more than 10 variables in a reasonable time). This is in part because some calls to the ILP oracle become slow, in part owing to the amount of factoring performed and in part owing to the growth in the number of simplicies to be handled. The largest benchmarks in the suite have 15 variables; this is real loop data and being able to handle between 10 to 20 variable problems would

| Problem | T1 | Best | Total | Max | Volume | TM | Cons | Size | Sam |
|---|---|---|---|---|---|---|---|---|---|
| berkeley-nat.inv | 0.061 | 100 | 401 | 101 | 260.0 | 0.089 | 9 | 8 | 405 |
| berkeley.inv | 0.049 | 107 | 518 | 108 | 5.981 | 0.084 | 9 | 8 | 514 |
| cars.inv | 0.099 | 134 | 1403 | 190 | 0.0 | 0.455 | 13 | 8 | 4596 |
| efm.inv | 0.114 | 200 | 635 | 200 | 0.0 | 0.255 | 12 | 10 | 973 |
| efm1.inv | 0.080 | 200 | 200 | 200 | 0.0 | 0.127 | 9 | 8 | 200 |
| heap.inv | 0.077 | 200 | 594 | 200 | 0.0 | 0.179 | 10 | 8 | 984 |
| lifo-nat.inv | 0.132 | 184 | 560 | 184 | 5.565 | 0.235 | 13 | 12 | 525 |
| robot.inv | 0.022 | 148 | 148 | 148 | 11000.372 | 0.032 | 5 | 4 | 149 |
| scheduler-2p.invl1 | 0.155 | 200 | 1368 | 200 | 0.0 | 0.709 | 15 | 10 | 3673 |
| scheduler-2p.invl2 | 0.198 | 200 | 1586 | 200 | 0.0 | 1.076 | 17 | 10 | 6776 |
| scheduler-3p.invl1 | 1.025 | 200 | 6170 | 200 | 0.0 | 16.282 | 42 | 14 | 92934 |
| scheduler-3p.invl2 | 47.755 | 200 | 22848 | 200 | 0.0 | 1556.5 | 125 | 14 | 1326519 |
| scheduler-3p.invl3 | 0.499 | 200 | 2572 | 200 | 0.0 | 4.312 | 26 | 14 | 17962 |
| see-saw.inv | 0.012 | 107 | 452 | 142 | 2.505 | 0.027 | 5 | 3 | 639 |
| swim-pool-1.inv | 0.223 | 200 | 800 | 200 | 0.0 | 0.745 | 16 | 13 | 1598 |
| swim-pool.inv | 0.208 | 200 | 400 | 200 | 0.0 | 0.521 | 15 | 13 | 600 |
| train-beacon.invlate1 | 0.026 | 109 | 331 | 112 | 1.431 | 0.045 | 7 | 6 | 349 |
| train-beacon.invonbrake | 0.030 | 104 | 206 | 104 | 3.308 | 0.038 | 6 | 5 | 214 |
| train-beacon.invontime | 0.034 | 134 | 716 | 134 | 30.09 | 0.100 | 8 | 5 | 1205 |
| train-beacon.invstopped | 0.031 | 114 | 346 | 120 | 1.263 | 0.064 | 7 | 5 | 441 |
| train-rm03.inv | 0.095 | 179 | 710 | 182 | 4.148E9 | 0.305 | 12 | 9 | 1219 |

**Fig. 4.** Off-line relaxation of constraints

allow the analysis of many programs. The authors believe that further work on the implementation will yield improvements in scalability. Size of the constraint coefficients is also a problem as the bounding box increases in size – most benchmarks run equally well with larger bounding boxes, but not all.

With two or three exceptions approximation results give the desired behaviour – sensible constraints to relax can be quickly identified. The slower benchmarks indicate a need to augment ranking with a timeout. A further issue is that when relaxing more than one constraint, dropping one or other equally ranked constraint can change the total number of constraints relaxed.

## 5 Related Work

The algorithm presented in Section 2 was first outlined by Hartmann in [4,10] and is also mentioned in [9]. However, its relationship with projection, non-linearity and program analysis have not previously been commented on. An alternative algorithm has been proposed by Meister based on the concept of periodic polyhedra [14]. Eisenbrand's work [9] also deals with approximating the integer hull from above, a counterpart to the work here that is also useful for program analysis. In terms of implementation, the *barvinok* package for integer

point counting [22] includes an integer hull algorithm also based on [10]; initial experimentation suggests that this does not scale as well as the implementation described here. The *iB4e* system [11] implements the beneath/beyond convex hull algorithm over reals and uses similar concepts; it is also presented with an oracle for solving LP problems, this could be an ILP solver.

The work on polynomial algorithms for lattice point counting from Barvinok and others [2,3,8,12,22] gives precisely what is required for assessing the importance of a candidate constraint for relaxation. This work has been extended to count points in the projection of a constrained space [21]. The results of these systems are impressive, but are still slower than sampling based techniques for estimating the number of integer points in a polytope as used in the approximation thread of this work. Recent work on loop nest analysis [19,22] utilises the algorithmic results on point counting. However, abstract interpretation based analysis requires constraints between variables, not lattice point counts.

## 6 Conclusion

This paper has presented work on the application of algorithms from computational geometry and linear programming to data arising in program analysis. An existing algorithm has been detailed and elucidated, and features that make its adaptation to problems of non-linearity and integrality in program analysis easy and natural have been identified. It has also been implemented and empirically evaluated. The results of this evaluation underline that this novel approach to dealing with integer variables and non-linear constraints in program analysis is promising. The approach is coupled with a method for approximating the increase in volume associated with relaxing a constraint from a system in order to control the size of that system. This too has been implemented and again the results suggests that the methods will be of importance in program analysis.

The implementation described in this paper represents the state-of-the-art for integer hull calculation. The results are promising, but also invite further work on increasing the speed of the implementation and the size of problem that can be dealt with. Future work will focus on improvements to the current implementation, whilst also investigating alternative approaches avoiding large numbers of calls to an ILP solver. The current implementation is not tuned to program analysis benchmarks and a further line of work is to investigate whether the structure of these lead to practical or theoretical improvements.

*Acknowledgements.* The authors thank Tim Hopkins, Mathias Köppe and Axel Simon for useful discussions. They would further like to thank the anonymous referees for their insightful comments and pointers to related work. The work was supported by EPSRC projects EP/E033105/1 and EP/E034519/1. Andy King's

secondment to Portcullis is supported by a Royal Society Industrial Fellowship. Jacob Howe would like to thank the University of St Andrews for supporting him as a visiting scholar.

## References

1. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Fast Acceleration of Symbolic Transition Systems. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 118–121. Springer, Heidelberg (2003)
2. Barvinok, A.: Computing the Volume, Computing Integral Points, and Exponential Sums. In: Computational Geometry, pp. 161–170. ACM Press, New York (1992)
3. Barvinok, A.: A Polynomial Time Algorithm for Counting Integral Points in Polyhedra When the Dimension is Fixed. Mathematics of Operations Research 19(4), 769–779 (1994)
4. Cook, W., Hartmann, M., Kannan, R., McDiarmid, C.: On Integer Points in Polyhedra. Combinatorica 12(1), 27–37 (1992)
5. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: The ASTREÉ analyzer. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 21–30. Springer, Heidelberg (2005)
6. Cousot, P., Halbwachs, N.: Automatic Discovery of Linear Restraints Among Variables of a Program. In: Principles of Programming Languages, pp. 84–96. ACM Press, New York (1978)
7. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties and Algorithms. Springer, Heidelberg (1996)
8. Deloera, J.A., Hemmecke, R., Tauzer, J., Yoshida, R.: Effective Lattice Point Counting in Rational Convex Polytopes. Journal of Symbolic Computation 38(4), 1273–1302 (2004)
9. Eisenbrand, F.: Gomory-Chvátal Cutting Planes and the Elementary Closure of Polyhedra. PhD thesis, Universität des Saarlandes (2000)
10. Hartmann, M.E.: Cutting Planes and the Complexity of the Integer Hull. PhD thesis, School of Operations Research and Industrial Engineering, Cornell University (1988); Technical Report 819
11. Huggins, P.: iB4e: A Software Framework for Parametrizing Specialized LP Problems. In: Iglesias, A., Takayama, N. (eds.) ICMS 2006. LNCS, vol. 4151, pp. 245–247. Springer, Heidelberg (2006)
12. Köppe, M.: A Primal Barvinok Algorithm Based on Irrational Decompositions. SIAM Journal on Discrete Mathematics 21(1), 220–236 (2007)
13. Lougee-Heimer, R.: The Common Optimization INterface for Operations Research. IBM Journal of Research and Development 47(1), 57–66 (2003)
14. Meister, B.: Periodic Polyhedra. In: Duesterwald, E. (ed.) CC 2004. LNCS, vol. 2985, pp. 134–149. Springer, Heidelberg (2004)
15. Ong, H.L., Huang, H.C., Huin, W.M.: Finding the Exact Volume of a Polyhedron. Advances in Engineering Software 34, 351–356 (2003)
16. Quinton, P., Rajopadhye, S.V., Risset, T.: On Manipulating $\mathbb{Z}$-polyhedra using a Canonical Representation. Parallel Processing Letters 7(2), 181–194 (1997)
17. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constraint Based Linear Relations Analysis. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 53–68. Springer, Heidelberg (2004)
18. Schrijver, A.: Theory of Linear and Integer Programming. Wiley, Chichester (1986)

19. Seghir, R., Loechner, V.: Memory Optimization by Counting Points in Integer Transformations of Parametric Polytopes. In: Compilers, Architectures, and Synthesis for Embedded Systems, pp. 74–82. ACM Press, New York (2006)
20. Simon, A.: Value-Range Analysis of C Programs. Springer, Heidelberg (2008)
21. Verdoolaege, S., Beyls, K., Bruynooghe, M., Catthoor, F.: Experiences with Enumeration of Integer Projections of Parametric Polytopes. In: Bodik, R. (ed.) CC 2005. LNCS, vol. 3443, pp. 91–105. Springer, Heidelberg (2005)
22. Verdoolaege, S., Seghir, R., Beyls, K., Loechner, V., Bruynooghe, M.: Counting Integer Points in Parametric Polytopes Using Barvinok's Rational Functions. Algorithmica 48(1), 37–66 (2007)
23. Wilde, D.K.: A Library for Doing Polyhedral Operations. Technical Report PI-785, IRISA (1993)

# Line Segment Facility Location in Weighted Subdivisions*

Yam Ki Cheung and Ovidiu Daescu

Department of Computer Science,
University of Texas at Dallas,
Richardson , TX 75080, USA
{ykcheung,daescu}@utdallas.edu

**Abstract.** In this paper we present approximation algorithms for solving the *line segment facility location problem in weighted regions*. The weighted region setup is a more realistic model for many facility location problems that arise in practical applications. Our algorithms exploit an interesting property of the problem, that could possibly be used for solving other problems in weighted regions.

## 1 Introduction

In this paper we consider a geometric optimization problem that we call the *line segment facility location in weighted regions*. We study the problem in a weighted subdivision $R = \{R_1, R_2, \ldots, R_{n'}\}$ of the plane with $n$ vertices, with each region $R_i \in R$ having associated a positive integer weight $w_i$. Without loss of generality, we assume that $R$ is triangulated. The Euclidean length of a line segment $st$ is denoted by $|st|$. We denote by $S(st)$ the weighted length of a segment $st \in R$. In general, the weighted length of a segment $st$ in $R$ is defined as the sum of the weighted lengths of $st$ within each region it intersects. That is, $S(st) = \sum_{st \cap R_j \neq \emptyset} w_j * d_j(st)$, where $d_j(st)$ is the Euclidean length of $st$ within region $R_j$.

The problem we study is defined as follows: Given a set of $l$ points $P = \{s_1, s_2, \ldots, s_l\} \in R$, find a facility $L$, which is a line segment, such that each point in $P$ can be connected to $L$ via an orthogonal link and some *cost* $C(L)$ associated with the facility location is minimized (see Fig. 1 for an example). We consider two versions of this problem, depending on how $C(L)$ is defined. For the first version ($P_1$), $C(L) = S(L)$. That is, we want to minimize the weighted length of $L$. For the second version ($P_2$), $C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$, where $L_i$ is the orthogonal link from $s_i$ to $L$. That is, we want to minimize the sum of weighted lengths of $L$ and the orthogonal links from points in $P$ to $L$.

The facility location problem is a well-studied problem in operations research and computer science literature [12,13,14,9]. The problem has many formulations, resulting in various definitions for the objective function. In our formulation, there is a cost for opening a facility and also for constructing orthogonal

* This research was partially supported by NSF grant CCF-0635013.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 100–113, 2009.

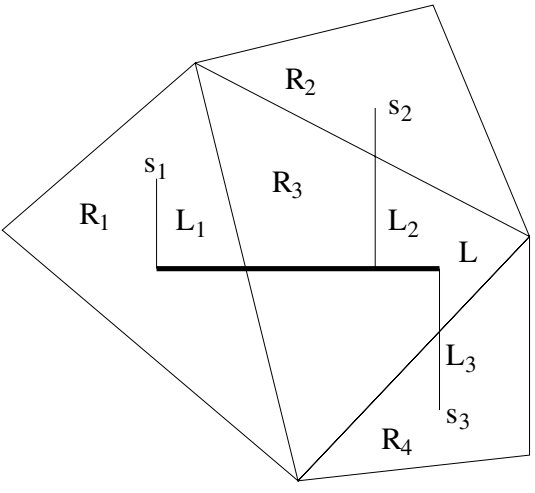



**Fig. 1.** A line segment facility $L$ and the orthogonal links from $P = \{s_1, s_2, s_3\}$ to $L$

links that connect customers to this facility. The goal of $P_1$ is to minimize the cost of building the facility, while $P_2$ takes into account the cost of building the facility as well as the cost of building orthogonal links to connect a set of sites to it. One may imagine that the facility $L$ is an oil pipeline and the points in $P$ are oil wells. The oil field is divided into weighted regions based on its characteristics (cost of digging, ownership rights, etc.). From each well, a spur pipeline is to be connected directly to $L$, in straight line. Given the x- and y-coordinates of the wells, the goal is to find the optimal location for the main pipeline that minimizes the total cost.

### 1.1 Related Work

In unweighted environment, Megiddo and Tamir [14] solved the problem of finding a line minimizing the sum of the Euclidean length of orthogonal links to the line more than two decades ago. They showed that an optimal line facility can be found in $O(n^2 \log n)$ time by proving that there exists an optimal line that passes through at least two of the points in $P$. Later, Imai et. al. [12] proved that in $L_1$-metric the optimal solution can be computed in linear time. Very recently, Cheung and Daescu [9] showed that the weighted version of this problem, i.e. $C(L) = \sum_{i=1}^{l} S(L_i)$, can be solved by dividing the problem into a number ($O(l^2 n^2)$ in the worst case) of 1-variable subproblems. The objective function of each subproblem has the form

$$\sum_{i=1}^{l} S(L_i) = \sqrt{1 + m_o^2}\left(\sum_{j=1}^{M} \frac{c_j}{m_o - m_j} + \sum_{i=1}^{l} \frac{b_i m_o + a_i}{m_o^2 + 1} + C\right),$$

for some constants $a_i, b_i, c_i$, and $C$, where $l$ is the number of points in $P$, $n$ is the number of vertices of $R$, and $m_o$ is the slope of orthogonal links. To this end, they proved a key property of the problem, specifically that there exists an optimal solution that passes through a set of "critical" points.

The proposed problem is closely related to weighted region shortest path problems, which have been investigated in computational geometry for about two decades. Using Snell's refraction low and the continuous Dijkstra algorithm, Mitchell and Papadimitriou [15] present an $O(n^8 \log \frac{nN'\rho}{\epsilon})$ time algorithm, where $N'$ is the largest integer coordinate of vertices of $R$ and $\rho$ is the ratio of the maximum weight to the minimum weight of the regions in $R$. Aleksandrov et al. [2,3] provide two logarithmic discretization schemes, that place Steiner points along edges or bisectors of angles of regions in $R$, forming geometric progressions. The placement of Steiner points depends on an input parameter $\epsilon > 0$ and the geometry of the subdivision. The $(1+\epsilon)$-approximation algorithms in [2,3] take $O(\frac{n}{\epsilon}(\frac{1}{\sqrt{\epsilon}} + \log n) \log \frac{1}{\epsilon})$ and $O(\frac{n}{\sqrt{\epsilon}} \log \frac{n}{\epsilon} \log \frac{1}{\epsilon})$ time, respectively. Sun and Reif [17] give an algorithm, called BUSHWHACK, which constructs a discrete graph $G$ by placing Steiner points along edges of the subdivision. By exploiting the geometric property of an optimal path, BUSHWHACK computes an approximate path more efficiently as it accesses only a subgraph of $G$. Combined with the logarithmic discretization scheme introduced in [2], BUSHWHACK takes $O(\frac{n}{\epsilon}(\log \frac{1}{\epsilon} + \log n) \log \frac{1}{\epsilon})$ time. Recently, Aleksandrov et al. [1] give a query algorithm that can find an $\epsilon$-approximate shortest path between any two points in $O(\bar{q})$ time, where $\bar{q}$ is a query time parameter. The preprocessing time of this algorithm is $O(\frac{(g+1)n^2}{\epsilon^{2/3}\bar{q}} \log \frac{n}{\epsilon} \log^4 \frac{1}{\epsilon})$, where $g$ is the genus of the discrete graph constructed by the discretization scheme. Cheng et al. [7] give an algorithm to approximate optimal paths in anisotropic regions, which is a generalized case of weighted regions. Their algorithm takes $O(\frac{\rho^2 \log \rho}{\epsilon^2} n^3 \log(\frac{\rho n}{\rho}))$ time, where $\rho \geq 1$ is a constant such that the convex distance function of any region contains a concentric Euclidean disk with radius $1/\rho$. In weighted regions, the time complexity of the algorithm is improved to $O(\frac{\rho \log \rho}{\epsilon} n^3 \log(\frac{\rho n}{\epsilon}))$, where $\rho$ is the ratio of the maximum weight to the minimum weight. Very recently, Cheng et al. [8] also provided a query version of this algorithm that gives an approximate optimal path from a fixed source (in an anisotropic subdivision) in $O(\log \frac{\rho n}{\epsilon})$ time. The preprocessing time is $O(\frac{\rho^2 n^4}{\epsilon^2}(\log \frac{\rho n}{\epsilon})^2)$.

Chen et al. [5] give a fundamental study on the optimal weighted link problem, where the goal is to minimize the weighted length of a line segment $st$ connecting two given regions, $R_s$ and $R_t$, of $R$. In [5], it has been proven that the problem to minimize the cost $S(st) = \sum_{st \cap R_j \neq \emptyset} w_j * d_j(st)$ can be reduced to a number of ($O(n^2)$ in the worst case) global optimization problems, each of which asks to minimize a 2-variable function over a convex domain. In [10,11], it has been shown that to minimize the objective function $S(st)$, $st$ must pass through a vertex of $R$. By considering each candidate vertex, they reduced the problem to solving $O(n)$ optimization problems at each vertex. The objective function for the optimization problem can be expressed in the form $\sqrt{1+m^2}(d_0 + \sum_{i=1}^{k} \frac{a_i}{m+b_i})$,

where $d_0$, $a_i$, and $b_i$ are constants, $k = O(n)$, and $m$ is the slope of the line supporting $st$. The feasible domain $D_m$ is a slope interval for the line passing through the vertex. Hence, the objective function of each subproblem contains only one variable and can be solved by computing the roots of a polynomial of degree $O(n)$ over the given domain $D_m$. Unfortunately, this approach is only practical when the number of fractional terms in $S(st)$ is small. In [11], a faster approach is suggested, which aims to compute an approximate solution by bisecting the slope interval and pruning out subproblems that cannot lead to an optimal solution. Some of the results in [5] were later on generalized in [6] to more difficult problems, where the line is replaced by a parallel or cone like beam.

### 1.2 Our Results

Our main contributions are in describing how to partition the problem into a polynomial number of 1-variable subproblems, and how to obtain the expression of the objective functions for these subproblems.

For problem $P_1$, we show that the objective function $C(L)$ can be expressed in the form of a sum of fractional terms, which resembles those in [5,10]. The problem can be reduced $O(l^2n^2)$ 1-variable optimization problems, each with $O(n)$ fractional terms, which can be solved by either computing exact roots of polynomials or by approximation algorithms. Using point-line duality transforms [16], the feasible domain for each subproblem is an arc or a line segment in the dual plane.

Then, we show that problem $P_2$ can be solved using a similar, but more complex, approach.

Finally, we outline a prune-and-search approximation algorithm for solving both problems, that is very simple to implement. We expect an approximate solution to be obtained fast in practice, since the prune-and-search algorithm handles all subproblems at once, rather than one by one.

## 2 $P_1$: Minimize the Weighted Length of $L$

In this section, we consider the problem of minimizing the weighted length of $L$, i.e. $C(L) = S(L)$, such that the points in $P$ can access $L$ via orthogonal links. We first derive the general objective function. Then, we show that this optimization problem can be reduced to a number of 1-variable subproblems. These subproblems can be solved by the prune-and-search approximation algorithm described in Section 4.

### 2.1 The Rotating Calipers

Notice that we want to set the line segment facility $L$ as short as possible, while each point in $P$ is able to connect to $L$ via an orthogonal link. Let $CH(P)$ denote the convex hull of $P$. A line $e$ is a *line of support* of $CH(P)$ if $e$ is tangent to $CH(P)$. The length of $L$ is minimized when $L$ is a segment bounded by two parallel lines of support enclosing $CH(P)$ and orthogonal to $L$. See Fig. 2 for

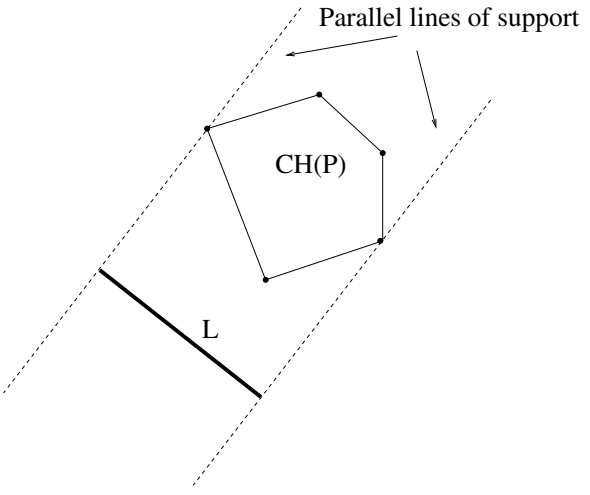



**Fig. 2.** $L$ is bounded by two parallel lines of support enclosing $CH(P)$

an illustration. A pair of vertices of $CH(P)$, $\{p_i, p_j\}$, is an *antipodal pair* if it admits two parallel lines of support enclosing $CH(P)$. Applying the *rotating calipers* procedure [18], we can generate all possible antipodal pairs ($h$ pairs in the worst case) for $O(h)$ slope intervals in $O(h)$ time, where $h$ is the number of vertices of $CH(P)$.

The rotating calipers procedure resembles rotating a pair of dynamically adjustable calipers around $CH(P)$ [18]. First, we choose an initial direction such as the y-axis. Let the two calipers be $e_1, e_2$ respectively. The antipodal pair $\{p_i, p_j\}$ of the initial slope can be found in $O(h)$ time. See Fig. 3 for an illustration. As we rotate the calipers, the calipers touch the same pair of antipodal vertices until one of the caliper touches another vertex of $CH(P)$. When at $p_i, p_j$, to generate the next antipodal pair we consider the angles, $\theta_i$ and $\theta_j$, that $e_1, e_2$ make with edges $p_ip_{i+1}$ and $p_jp_{j+1}$, respectively. If $\theta_j < \theta_i$, we rotate the calipers by an angle $\theta_j$, and $\{p_{j+1}, p_i\}$ becomes the next antipodal pair, and vice versa. This process is continued until we come full circle to the starting position.

### 2.2 Optimization of $S(L)$

Given a slope $m$, let the antipodal pair that admits the two parallel lines of support of slope $-1/m$ be $p_i = (a, b)$ and $p_j = (c, d)$, respectively. Also, let $L$ be a line segment with end points on the two parallel lines of support and orthogonal to them. Let the sequence of edges intersected by $L$ be $Seq(L) = (e_1, e_2, \ldots, e_k)$, where $k = O(n)$, $e_j : y = m_jx + p_j$, for $j = 1, 2, \ldots, k$, and $e_1$ and $e_k$ are the two parallel lines of support. We have

$$S(L) = \sqrt{1+m^2}\sum_{j=1}^{k-1} w_i|x_{j+1} - x_j| = \sqrt{1+m^2}\sum_{j=1}^{k-1} w_j(\frac{p_{j+1}-p}{m-m_{j+1}} - \frac{p_j-p}{m-m_j}),$$

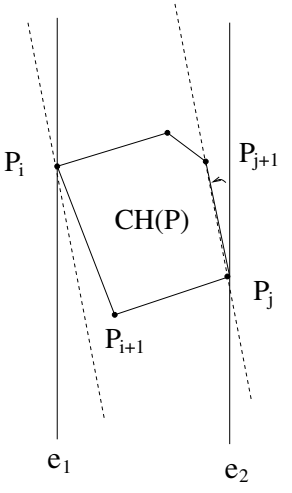



**Fig. 3.** Rotating the calipers around $CH(P)$

where $m$ and $p$ are the slope and intercept of $L$ respectively, $x_j$ is the $x$-coordinate of the intersection between $L$ and $e_j$, and $w_j$ is the weight of the region bounded by edges $e_j$ and $e_{j+1}$. Since $e_1$ and $e_k$ are the two parallel lines of support, we have $e_1 : y = -1/m(x-a)+b$ and $e_k : y = -1/m(x-c)+d$. It follows that $S(L) = \sqrt{1+m^2}(\sum_{j=2}^{k-2} w_j(\frac{p_{j+1}-p}{m-m_{j+1}} - \frac{p_j-p}{m-m_j}) + w_1(\frac{p_2-p}{m-m_2} - \frac{b+a/m-p}{m+1/m}) + w_{k-1}(\frac{d+c/m-p}{m+1/m} - \frac{p_{k-1}-p}{m-m_{k-1}}))$, and thus

$$S(L) = \sqrt{1+m^2}(\sum_{j=2}^{k-2} \frac{c_j + d_j p}{m-m_j} + \frac{Cmp+Dm+E}{1+m^2}),$$

where $C, D, E, c_j$, and $d_j$ are all constants.

Let $Q$ denote the set of intersection points between the lines of support above and the edges of $R$. Let $V$ be the set of vertices of $R$.

**Lemma 1.** *For a fixed slope $m$, there exists an optimal solution $L$ such that $L \cap (V \cup Q) \neq \emptyset$, where $Q$ is the set of intersection points between the lines of support at slope $-1/m$ and the edges of $R$. That is, $L$ passes through a point in $V \cup Q$.*

*Proof.* With $m$ fixed, $S(L)$ becomes a linear function of $p$. Thus, there is a direction of improvement for $S(L)$, as we slide it parallel to itself. The direction of improvement may change only when $L$ intersects a new region of $R$, and thus the expression of $S(L)$ changes. This can happen only if one of the following two situation occurs: (1) $L$ passes a vertex of $R$ or (2) an endpoint of $L$ passes an edge of $R$. Since that endpoint of $L$ is also on a line of support, the second case implies the endpoint passes a point in $Q$. □

**Corollary 1.** *For a fixed slope $m$ we can find the optimal $L$ in $O(n^2)$ time and $O(n)$ space.*

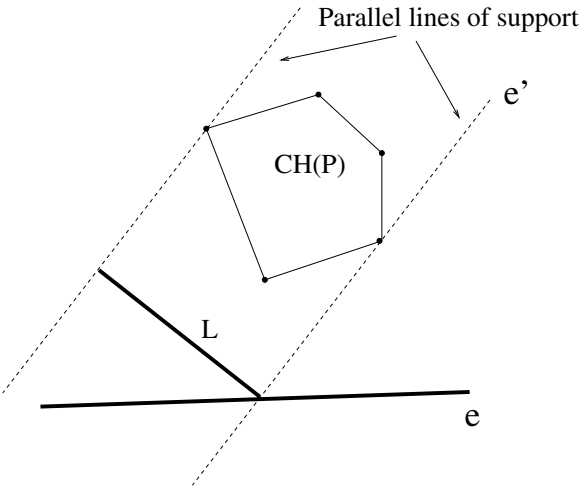



**Fig. 4.** $L$ intersects a line of support at an edge

Next, we analyze the problem in the dual space. We use the duality transform which maps a line $y = mx+p$ on the plane onto a point $(m, p)$ in the dual plane and maps a point $(a, b)$ onto the line $y = -ax + b$ on the dual plane. Note that $L$ can be represented by the line supporting it.

Observe that if we change the slope $m$ of $L$ then $seq(L)$ changes, i.e. the expression of the objective function $C(L) = S(L)$ changes, only if one of the following events occurs:
(1) $L$ sweeps through a vertex of $R$, or
(2) $L$ sweeps through an intersection point between a line of support and an edge of $R$, or
(3) $L$ rotates through a slope such that the antipodal pair changes.

We introduce three sets of curves or lines to properly partition the dual space such that each cell corresponds to all segments in the primal space which have the same functional expression for the weighted length $S(L)$.

First, we transform all vertices of $R$ to the dual space. Let $A_v$ denote this arrangement, which consists of $n$ lines.

Next, consider that $L$ intersects a line of support $e'$ at an edge $e$. As the line of support $e'$ rotates, the intersection between $L$ and $e'$ moves along edge $e$.See Fig. 4 for an illustration.

Let the vertex of $CH(P)$ that is on $e'$ be $(a, b)$ and let $e : y = m_e x + p_e$. We have

$$e' : (y - b) = -\frac{1}{m}(x - a)$$

$$y = -\frac{1}{m}x + \frac{a}{m} + b$$

$$L : y = mx + p$$

For the intersection point of $L$ and $e'$, we have:

$$-\frac{1}{m}x + \frac{a}{m} + b = mx + p$$

$$x = \frac{bm - pm + a}{m^2 + 1}$$

Thus, $L$ and $e'$ intersect at $u : (\frac{bm-pm+a}{m^2+1}, \frac{bm^2+am+p}{m^2+1})$. Since $u \in e$, we have

$$\frac{bm^2 + am + p}{m^2 + 1} = m_e \frac{bm - pm + a}{m^2 + 1} + p_e$$

$$bm^2 + am + p = m_e(bm - pm + a) + p_e(m^2 + 1)$$

$$(b - p_e)m^2 + (a + m_e p - m_e b)m + (p - m_e a - p_e) = 0$$

$$p = -\frac{(b - p_e)m^2 + (a - m_e b)m - m_e a - p_e}{m_e m + 1}$$

Since there are $O(h)$ antipodal pairs of vertices and $O(n)$ edges of $R$, we have a total of $O(hn)$ such curves. Let $A'_c$ denote the arrangement of these curves. Note that two such curves can intersect at most three times and one such curve can intersect with a line at most two times.

Finally, let $M = \{m_1, m_2, \ldots, m_h\}$ be the set of slopes of edges in $CH(P)$ and let $M' = \{-1/m : m \in M\}$. Recall that in the rotating calipers procedure, the antipodal pair changes only when the calipers are rotating through the slopes in $M$, or equivalently when $L$ rotates through the slopes in $M'$, since $L$ is orthogonal to the calipers. Let $A'_r$ be the arrangement of $O(h)$ vertical lines $\{x^* = m : m \in M'\}$ in the dual space.

The overall partition is $A_v \cup A'_c \cup A'_r$, which consists of $O(n)$ lines and $O(hn)$ curves. See Fig. 5 for an illustration.

**Lemma 2.** *$A_v \cup A'_c \cup A'_r$ partitions the dual space such that each cell corresponds to all segments in the primal space which have the same functional expression for the weighted length.*

*Proof.* Let $L$ and $L'$ be two line facilities such that $S(L)$ and $S(L')$ have different functional expressions for the weighted length. Due to order preserving property of the dual transform, their corresponding dual points $L^*$ and $L'^*$ must be separated by at least one line or curve in $A_v$, $A'_c$ or $A'_r$. The proof follows. □

Using the algorithm in [4], the arrangement $A_v \cup A'_c \cup A'_r$ can be computed in $O(nh \log(nh) + k)$ time and $O(nh + k)$ space, where $k$ is the number of cells, which is $O(h^2 n^2)$ in the worst case.

**Lemma 3.** *A global minimum of $C(L)$ can be found at non-vertical boundaries of the partition, i.e. on the lines or the curves in $A_v \cup A'_c$.*

*Proof.* When the slope $m$ of $L$ is fixed, the objective function $C(L)$ is linear and monotonic with respect to the intercept $p$ of $L$. Then, a minimum of $C(L)$ can be found on the non-vertical boundary of the partition. □

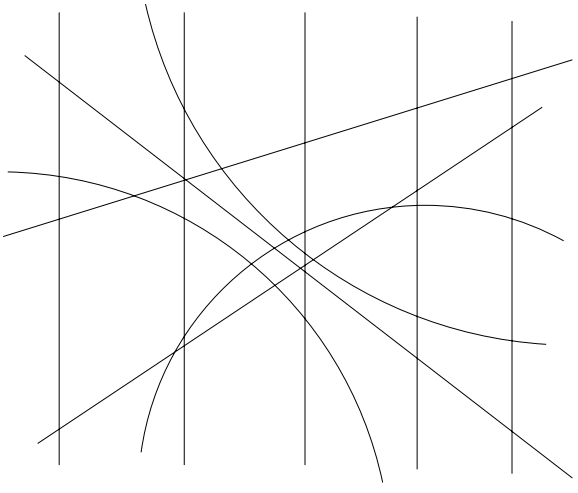


**Fig. 5.** Partition of the dual space

**Lemma 4.** *$P_1$ can be reduced to $O(h^2n^2)$ 1-variable subproblems. The domain for each subproblem is a piece of non-vertical boundary on the partition $A_v \cup A'_c \cup A'_r$.*

*Proof.* The complexity of the overall partition $A_v \cup A'_c \cup A'_r$ is $O(h^2n^2)$. The result follows from Lemma 2 and Lemma 3. □

We now analyze the subproblems. If the domain of a subproblem is on $A_v$, $L$ passes through a vertex. Let $v : (x_i, y_i) \in R$ be that vertex. We have,

$$L : y - y_i = m(x - x_i),$$

$$p = y_i - mx_i,$$

$$\begin{aligned} S(L) &= \sqrt{1+m^2}\left(\sum_{j=2}^{k-2} \frac{c_j + d_j p}{m - m_j} + \frac{Cmp + Dm + E}{1+m^2}\right) \\ &= \sqrt{1+m^2}\left(\sum_{j=2}^{k-2} \frac{c_j + d_j(y_i - mx_i)}{m - m_j} + \frac{Cm(y_i - mx_i) + Dm + E}{1+m^2}\right) \\ &= \sqrt{1+m^2}\left(\sum_{j=2}^{k-2} \frac{c'_j}{m - m_j} + \frac{D'm + E'}{1+m^2} + C'\right), \end{aligned}$$

where $m_j, c'_j, C', D', E'$ are all constants.

If the domain of the subproblem is on $A'_c$, $L$ intersects with a line of support on an edge $e$. Let $m_e$ and $p_e$ be the slope and intercept of $e$, respectively, and

let the vertex of $CH(P)$ on the line of support be $(a, b)$. Substituting

$$p = -\frac{(b - p_e)m^2 + (a - m_e b)m - m_e a - p_e}{m_e m + 1}$$

into $S(L)$, we have

$$S(L) = \sqrt{1 + m^2}(\sum_{j=2}^{k-2} \frac{c'_j m + d'_j}{(m - m_j)(1 + m_e m)} + \frac{C'm^2 + D'm + E'}{(1 + m^2)(1 + m_e m)} + C''),$$

where $m_j, m_e, c'_j, d'_j, C', C'', D', E'$ are constants.

We will show in Section 4 how to minimize $S(L)$ using a prune-and-search algorithm.

## 3 $P_2$: Minimize the Sum of Weighted Length of Orthogonal Links and $L$

In this section, we address the second optimization problem $P_2$, which asks to minimize the sum of weighted length of orthogonal links and $S(L)$, i.e. $C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$. We solve this optimization problem by (1) partitioning the problem in the dual space, (2) finding the objective function on the boundary of the partition and (3) applying the prune-and-search algorithm in Section 4 to approximate the solution.

The problem of minimizing the sum of weighted length of orthogonal links is solved in [9]. In [9], the dual space is partitioned by introducing three sets of curves or lines. The first set consists of dual lines of points in $P$. Let $A_p$ denote the arrangement of the $l$ dual lines. Next, all lines intersecting with some orthogonal link at some edge of $R$ are transformed to the dual space. This yields a set of $O(ln)$ curves. Let $A_c$ denote the arrangement of this set of curves. The last set of lines introduced to the dual space is

$$\{x^* = -1/m_{ij} : i = 1, 2, \ldots, l \text{ and } j = 1, 2, \ldots, n\},$$

where $m_{ij}$ is the slope of the line passing through point $s_i \in P$ and vertex $v_j \in R$. Let $A_r$ denote the arrangement of this set of lines. The optimization problem is reduced to a number of 1-variable subproblems. The domain for each subproblem is a piece of arc or segment on $A_p$ or $A_c$. The objective function for each subproblem can be expressed in the form

$$\sum_{i=1}^{l} S(L_i) = \sqrt{1 + m_o^2}(\sum_{j=1}^{M} \frac{c_j}{m_o - m_j} + \sum_{i=1}^{l} \frac{b_i m_o + a_i}{m_o^2 + 1} + C),$$

for some constants $a_i, b_i, c_i, C$, where $m_o$ is the slope of orthogonal links. Substituting $m_o = -1/m$,

$$\sum_{i=1}^{l} S(L_i) = \sqrt{1 + m^2}(\sum_{j=1}^{M} \frac{-c_j}{m_j m + 1} + \sum_{i=1}^{l} \frac{a_i m - b_i}{m^2 + 1} + \frac{C}{m}).$$

Notice that $A'_c$, defined in the previous section, is a subset of $A_c$, since each line of support supports an orthogonal link from a point in $P$ to $L$. It follows that $A_p \cup A_c \cup A_r \cup A_v \cup A'_r$ partitions the dual space such that each cell corresponds to all segments in the primal space which have the same functional expression for the objective function, i.e. $C(L) = \sum_{i=1}^{l} S(L_i) + S(L)$. The partition can be computed in $O(ln\log(lh) + k)$ time and $O(lh + k)$ space, where $k$ is the number cells, which is $O(l^2n^2)$ in worst case, using the algorithm in [4].

**Lemma 5.** *An optimal solution for $P_2$ can be found on the non-vertical boundary of the partition $A_p \cup A_c \cup A_r \cup A_v \cup A'_r$.*

*Proof.* Similar to Lemma 3. □

**Lemma 6.** *$P_2$ can be divided into $O(l^2n^2)$ subproblems, where the domain for each subproblem is a piece of non-vertical boundary of the partition $A_p \cup A_c \cup A_r \cup A_v \cup A'_r$, i.e. an arc or a segment on $A_p$, $A_c$, or $A_v$.*

*Proof.* Similar to Lemma 4. □

Next, we derive the overall expression for $C(L)$. If the domain of the subproblem is on $A_p$ or $A_v$, i.e. $L$ passes through a point $s_i \in P$ or a vertex in $R$, we have:

$$\begin{aligned} C(L) &= \sum_{i=1}^{l} S(L_i) + S(L) \\ &= \sqrt{1+m^2}\left(\sum_{j=1}^{M} \frac{-c_j}{m_j m + 1} + \sum_{i=1}^{l} \frac{a_i m - b_i}{m^2+1} + \frac{C}{m}\right) \\ &+ \sqrt{1+m^2}\left(\sum_{j=2}^{k-2} \frac{c'_j}{m - m'_j} + \frac{D'm + E'}{1+m^2} + C'\right) \\ &= \sqrt{1+m^2}\left(\sum_{j=1}^{M}\left(\frac{c''_j}{m_j m + 1} + \frac{c'_j}{m - m_j}\right) + \frac{D''m + E''}{1+m^2} + \frac{C}{m} + C'\right), \end{aligned}$$

where $m_j, c_j, c'_j$, $c''_j$, $C$, $C'$, $D''$, and $E''$ are constants.

If the domain of the subproblem is on $A_c$, i.e. $L$ intersects with an orthogonal link or a line of support on an edge, we have:

$$\begin{aligned} C(L) &= \sum_{i=1}^{l} S(L_i) + S(L) \\ &= \sqrt{1+m^2}\left(\sum_{j=1}^{M} \frac{-c_j}{m_j m + 1} + \sum_{i=1}^{l} \frac{a_i m - b_i}{m^2+1} + \frac{C}{m}\right) \\ &+ \sqrt{1+m^2}\left(\sum_{j=2}^{k-2} \frac{c'_j m + d'_j}{(m - m'_j)(1 + m_e m)} + \frac{C'm^2 + D'm + E'}{(1+m^2)(1+m_e m)} + C'''\right) \end{aligned}$$

$$= \sqrt{1+m^2}\left(\sum_{j=1}^{M}\left(\frac{c''_j}{m_j m+1}+\frac{c'_j m+d'_j}{(m-m_j)(1+m_e m)}\right)\right.$$
$$\left.+\frac{C''m^2+D''m+E''}{(1+m^2)(1+m_e m)}+\frac{C}{m}+C'''\right),$$

where $m_j, m_e, c_j, c'_j$, and $c''_j$ are constants.

Once the objective functions are derived for all subproblems, the optimal solution can be approximated within an $\epsilon$-factor by applying the prune-and-search algorithm described in the next section.

## 4 A Prune-and-Search Approximation Algorithm

We present a prune-and-search approximation algorithm combined with the subdivision approach. The general outline is as follows:

(1) Find upper and lower bounds for each subproblem of interest.
(2) Calculate $U^{min}$, which is the minimum of upper bounds of all subproblems in the queue.
(3) For each subproblem, if its lower bound is greater than $U^{min}$ then prune it.
(4) Maintain a priority queue of active subproblems based on their lower bounds.
(5) Get the first candidate $I_i$ from the priority queue and find a coarse approximation by placing Steiner points and sampling.
(6) A solution $L$ is accepted if $S(L) \leq (1+\epsilon)S_i^{min}$, where $\epsilon$ is a parameter defining the quality of the approximation and $S_i^{min}$ is the lower bound of $I_i$.
(7) If no acceptable solution has been found, we further divide $I_i$ into smaller subproblems.
(8) Find the upper bound and lower bound of all new subproblems and add them back to the priority queue. Go to step (3).

Next, we give more details of the prune-and-search algorithm for problem $P_1$ (a similar analysis can be done for $P_2$). Given a subproblem $I_i$, all line segments in the feasible domain $D_i$ intersect with the same set of regions of $R$. Let the set of regions intersected by segments in $D_i$ be $\{R_{i_1}, R_{i_2}, \ldots, R_{i_k}\}$. We can set the lower bound of $I_i$, $S_i^{min}$, as $\sum_{q=1}^{k} w_{i_q} * min_{L\in D_i}(L \cap R_{i_q})$. The value of $min_{L\in D_i}(L \cap R_{i_q})$ can be computed in $O(1)$ time. The upper bound of $I_i$ could be any sample value of the objective function in the given domain.

**Lemma 7.** *The prune process is safe.*

*Proof.* Consider a subproblem $I_i$. If the lower bound $S_i^{min} \geq U^{min}$ then the subproblem that produces $U^{min}$ can give us a solution at least as good as the best possible solution given by $I_i$. Hence, there is no need to consider $I_i$ in future steps. □

## 5 Conclusions

In this paper, we discussed two versions of the line segment facility location problem in weighted regions. For the first version ($P_1$), we want to minimize the

weighted length of $L$, while each point in a set of points $P$ is able to connect to $L$ via an orthogonal link. For the second version ($P_2$), we want to minimize the sum of weighted lengths of $L$ and the orthogonal links from points in $P$ to $L$. Each version of the problem can be solved in similar fashion. We partition the problem in the dual space such that the optimal solution is located on the boundary of the partition. We show that $P_1$ can be reduced to $O(h^2n^2)$ 1-variable subproblems, while $P_2$ can be reduced to $O(l^2n^2)$ 1-variable subproblems, where $h$ is the number of vertices of the convex hull of $P$, $l$ is the number of points in $P$, and $n$ is the number of vertices in the weighted subdivision. Approximate solutions for $P_1$ and $P_2$ can be found by applying a prune-and-search algorithm.

We conclude by mentioning the following open problems.

- Is it possible to obtain a combinatorial solution for $P_1$, $P_2$, or the problem of minimizing the sum of orthogonal links?
- Other interesting versions are when $L$ reduces to a point or when we do not require the connecting links to be orthogonal to $L$.

## References

1. Aleksandrov, L., Djidjev, H.N., Guo, H., Maheshwari, A., Nussbaum, D., Sack, J.R.: Algorithms for approximate shortest path Queries on weighted polyhedral surfaces. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 98–109. Springer, Heidelberg (2006)
2. Aleksandrov, L., Maheshwari, A.A., Sack, J.R.: Approximation algorithms for geometric shortest path problems. In: Proc. 32nd Annual ACM Symposium on Theory of Computing, pp. 286–295 (2000)
3. Aleksandrov, L., Maheshwari, A., Sack, J.R.: Determining approximate shortest paths on weighted polyhedral surfaces. Journal of the ACM 52(1), 25–53 (2005)
4. Amato, N.M., Goodrich, M.T., Ramos, E.A.: Computing the arrangement of curve segments: Divide-and-conquer algorithms via sampling. In: Proc. 11th Annual CAM-SIAM Symposium on Discrete Algorithms, pp. 705–706 (2000)
5. Chen, D.Z., Daescu, O., Hu, X., Wu, X., Xu, J.: Determining an optimal penetration among weighted regions in two and three dimensions. Journal of Combinatorial Optimization 5(1), 59–79 (2001)
6. Chen, D.Z., Hu, X., Xu, J.: Computing Optimal Beams in Two and Three Dimensions. Journal of Combinatorial Optimization 7(2), 111–136 (2003)
7. Cheng, S.W., Na, H.S., Vigneron, A., Wang, Y.: Approximate shortest paths in anisotropic regions. SIAM Journal on Computing 38, 802–824 (2008)
8. Cheng, S.W., Na, H.S., Vigneron, A., Wang, Y.: Querying approximate shortest paths in anisotropic regions. In: Proc. 23rd Symposium on Computational Geometry, pp. 84–91 (2007)
9. Cheung, Y., Daescu, O.: Line Facility Location in Weighted Regions. In: Proc. 4th Intl. Conf. on Algorithmic Aspects in Information and Management, pp. 109–119 (2008)
10. Daescu, O.: Improved optimal weighted links algorithms. In: Proc. ICCS, 2nd International Workshop on Computational Geometry and Applications, pp. 227–233 (2002)
11. Daescu, O., Palmer, J.: Minimum Separation in Weighted Subdivisions. International Journal of Computational Geometry and Applications 19(1), 33–57 (2009)

12. Imai, H., Kato, K., Yamamoto, P.: A linear-time algorithm for linear $L_1$ approximation of points. Algorithmica 4(1), 77–96 (1989)
13. Goemans, M.X., Skutella, M.: Cooperative facility location games. Journal of Algorithms 50(2), 194–214 (2004)
14. Megiddo, N., Tamir, A.: Finding Least-Distance Lines. SIAM Journal of Algebraic Discrete Methods 4(2), 207–211 (1983)
15. Mitchell, J.S.B., Papdimitriou, C.H.: The weighted region problem: Finding shortest paths through a weighted planer subdivision. Journal of the ACM 38(1), 18–73 (1991)
16. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer, New York (1985)
17. Sun, Z., Reif, J.H.: On finding approximate optimal path in weighted regions. Journal of Algorithms 58(1), 1–32 (2006)
18. Toussaint, G.T.: Solving geometric problems with the rotating calipers. In: Proc. 2nd IEEE Mediterranean Electrotechnical Conference (MELECON 1983), pp. 1–4 (1983)

# Algorithms for Placing Monitors in a Flow Network
## (Preliminary Version)

Francis Chin[1,⋆], Marek Chrobak[2,⋆⋆], and Li Yan[2,⋆⋆]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
[2] Department of Computer Science, University of California, Riverside, CA 92521

**Abstract.** In the Flow Edge-Monitor Problem, we are given an undirected graph $G = (V, E)$, an integer $k > 0$ and some unknown circulation $\psi$ on $G$. We want to find a set of $k$ edges in $G$, so that if we place $k$ monitors on those edges to measure the flow along them, the total number of edges for which the flow can be uniquely determined is maximized. In this paper, we first show that the Flow Edge-Monitor Problem is NP-hard, and then we give two approximation algorithms: a 3-approximation algorithm with running time $O((m+n)^2)$ and a 2-approximation algorithm with running time $O((m+n)^3)$, where $n = |V|$ and $m = |E|$.

## 1 Introduction

We study the *Flow Edge-Monitor Problem* (FlowMntrs, for short), where the objective is to find $k$ edges in an undirected graph $G = (V, E)$ with an unknown circulation $\psi$, so that if we place $k$ flow monitors on these edges to measure the flow along them, we will maximize the total number of edges for which the value and direction of $\psi$ is uniquely determined by the flow conservation property. Intuitively, the objective is to maximize the number of bridge edges in the subgraph induced by edges not covered by monitors. (For a more rigorous definition of the problem, see Section 2.)

Consider, for example, the graph and the monitors shown in Figure 1. In this example we have $k = 4$ monitors represented by rectangles attached to edges, with measured flow values and directions shown inside. Thus we have $\psi(2, 3) = 4$, $\psi(3, 8) = 2$, $\psi(6, 4) = 7$ and $\psi(1, 2) = 1$. From the flow conservation property, we can then determine that $\psi(3, 5) = 2$, $\psi(8, 6) = 2$, $\psi(7, 5) = 3$ and $\psi(5, 6) = 5$. Thus with 4 monitors we can determine flow values on 8 edges.

**Our results.** We first show that the FlowMntrs problem is NP-hard. Next, we study polynomial-time approximation algorithms. We introduce an algorithm called $\sigma$-Greedy that, in each step, places up to $\sigma$ monitors in such

⋆ Research supported in parts by grant (HKU 7113/07E).
⋆⋆ Research supported by NSF Grant CCF-0729071.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 114–128, 2009.

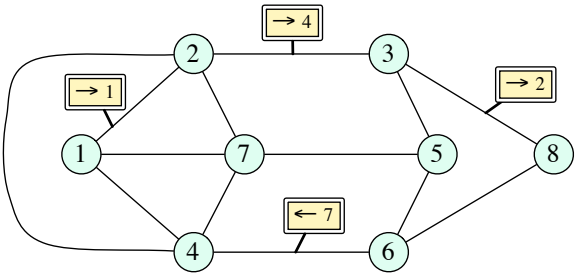



**Fig. 1.** A graph with 4 monitors

a way that the number of edges with known flow is maximized. We then prove that 1-Greedy is a 3-approximation algorithm and that 2-Greedy is a 2-approximation algorithm. The running times of these two algorithms are $O((m+n)^2)$ and $O((m+n)^3)$, respectively, where $n = |V|$ and $m = |E|$. In both cases, our analysis is tight. In fact, our approximation results are stronger, as they apply to the weighted case, where the input graph has weights on edges, and the objective is to maximize the total weight of the edges with known flow.

**Related work.** A closely related problem was studied by Gu and Jia [4] who considered a traffic flow network with directed edges. They observed that $m-n+1$ monitors are necessary to determine the flow on all edges of a strongly connected graph, and that this bound can be achieved by placing flow monitors on edges in the complement of a spanning tree. (The same bound applies to connected undirected graphs.) Khuller et al. [5] studied an optimization problem where pressure meters may be placed on nodes of a flow network. An edge whose both endpoints have a pressure meter will have the flow determined using the pressure difference, and other edges may have the flow determined via flow conservation property. The goal is to compute the minimum number of meters needed to determine the flow on every edge in the network. They showed that this problem is NP-hard and MAX-SNP-hard, and that a local-search based algorithm achieves 2-approximation. For planar graphs, they have a polynomial-time approximation scheme. The model in [5] differs from ours in that it assumes that the flow satisfies Kirchhoff's current and voltage laws, while we only assume the current law (that is, the flow preservation property). This distinction is reflected in different choices of "meters": vertex meters in [5] and edge monitors in our paper. Recall that, as explained above, minimizing the number of *edge monitors* needed to determine the flow on all edges is trivial, providing a further justification for our choice of the objective function.

The FlowMntrs problem is also related to the classical $k$-cut and multi-way cut problems [6,8,1], where the goal is to find a minimum-weight set of edges that partitions the graph into $k$ connected components. One can view our monitor problem as asking to maximize the number of connected components obtained from removing the monitor edges and the resulting bridge edges.

## 2 Preliminaries

We now give formal definitions. Let $G = (V, E)$ be an undirected graph. We assume that $G$ is simple, that is, it does not have multiple edges or loops, although our algorithms work for multi-graphs with loops as well. Throughout the paper, we use $n = |V|$ to denote the number of vertices in $G$ and $m = |E|$ to be the number of edges. We will typically use letters $u, v, x, y, ...$, possibly with indices, to denote vertices, and $a, b, e, f, ...$ to denote edges. If an edge $e$ has endpoints $x, y$, we write $e = \{x, y\}$.

A *circulation* on $G$ is a function $\psi$ that assigns a flow value and a direction to any edge in $E$. (We use the terms "circulation" and "flow" interchangeably, slightly abusing the terminology.) Denoting by $\psi(u, v)$ the flow on $e = \{u, v\}$ from $u$ to $v$, we require that $\psi$ satisfies the following two conditions (i) $\psi$ is anti-symmetric, that is $\psi(u, v) = -\psi(v, u)$ for each edge $\{u, v\}$, and (ii) $\psi$ satisfies the flow conservation property, that is $\sum_{\{u,v\}\in E} \psi(u, v) = 0$ for each vertex $v$.

A *bridge* in $G$ is an edge whose removal increases the number of connected components of $G$. Let $Br(G)$ be the set of bridges in $G$. The flow value on any bridge of $G$ must be 0, so, without loss of generality, throughout the paper we will be assuming that the input graph does not have any bridges. In other words, each connected component of $G$ is 2-edge-connected. (Recall that, for an integer $c \geq 1$, a graph $H$ is called $c$-edge-connected, if $H$ is connected and it remains connected after removing any $c - 1$ edges from $H$.)

Suppose that some circulation $\psi$ is given for all edges in some set $M \subseteq E$, and not for other edges. We have the following observation:

**Observation 1.** *For $\{u, v\} \in E - M$, $\psi(u, v)$ is uniquely determined if and only if $\{u, v\} \in Br(G - M)$.*

We can now define the *gain* of $M$ to be $gain(G, M) = |M \cup Br(G - M)|$, that is, the total number of edges for which the flow can be determined if we place monitors on the edges in $M$. We will refer to the edges in $M$ as *monitor* edges, while the bridge edges in $Br(G-M)$ will be called *extra* edges. If $G$ is understood from context, we will write simply $gain(M)$ instead of $gain(G, M)$.

The *Flow Edge-Monitor Problem* (FlowMntrs) can now be defined formally as follows: given a graph $G = (V, E)$ and an integer $k > 0$, find a set $M \subseteq E$ with $|M| \leq k$ that maximizes $gain(G, M)$.

**The weighted case.** We consider the extension of FlowMntrs to weighted graphs, where each edge $e$ has a non-negative weight $w(e)$ assigned to it, and the task is to maximize the *weighted gain*. More precisely, if $M$ are the monitor edges, then the formula for the (weighted) gain is $gain(M) = \sum_{e\in M\cup B} w(e)$, for $B = Br(G - M)$. We will denote this problem by WFlowMntrs.

Throughout the paper, we denote by $M^*$ some arbitrary, but fixed, optimal monitor edge set. Let $B^* = Br(G - M^*)$ be the set of extra edges corresponding to $M^*$. Then the optimal gain is $gain^*(G, k) = w(M^* \cup B^*)$.

We now claim that in the weighted case we can restrict our attention to graphs whose all connected components are 3-edge-connected. More specifically,

we show that any weighted 2-edge-connected graph $G = (V, E)$ can be converted in linear time into a 3-edge-connected weighted graph $G' = (V', E')$ such that:

(i) $gain^*(G, k) = gain^*(G', k)$, and
(ii) If $M' \subseteq E'$ is a set of $k$ monitor edges in $G'$, then in linear time one can find a set $M \subseteq E$ of $k$ monitor edges in $G$ with $gain(G, M) = gain(G', M')$.

We now show the construction of $G'$. A 2-*cut* is a pair of edges $\{e, e'\}$ whose removal disconnects $G$. Write $e \simeq e'$ if $\{e, e'\}$ is a 2-cut. It is known, and quite easy to show, that relation "$\simeq$" is an equivalence relation on $E$. The equivalence classes of $\simeq$ are called *edge groups*.

Suppose that $G$ has an edge group $F$ with $|F| = q$, for $q \geq 2$, and let $H_1, ..., H_q$ be the connected components of $G - F$. Then $F = \{e_1, ..., e_q\}$, where, for each $i$, $e_i = \{u_i, v_i\}$, $u_i \in H_i$ and $v_i \in H_{i+1}$ (for $i = q$ we assume $q + 1 \equiv 1$). For $i = 1, ..., q - 1$, contract edge $e_i$ so that vertices $u_i$ and $v_i$ become one vertex, and then assign to edge $e_q = \{u_q, v_q\}$ weight $\sum_{i=1}^{q} w(e_i)$. We will refer to $e_q$ as the *deputy edge* for $F$. Figure 2 illustrates the construction.

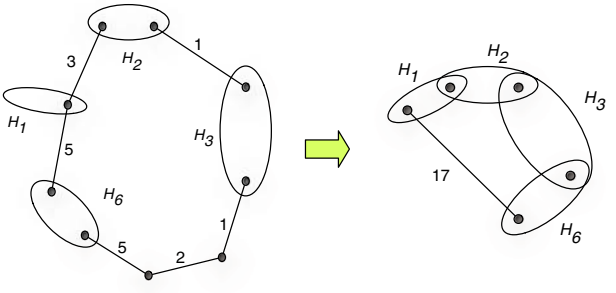



**Fig. 2.** Contracting edge groups

Let $G' = (V', E')$ be the resulting weighted graph. By the construction, $G'$ is 3-edge-connected. All edge groups can be computed in linear time (see, [7], for example), so the whole transformation can be done in linear time as well.

It remains to show that $G'$ satisfies conditions (i) and (ii). If $M$ is any monitor set, and if $M$ has two or more monitors in the same edge group, we can remove one of these monitors without decreasing the gain of $M$. Further, for any monitor edge $e$ of $M$, we can replace $e$ by the deputy edge of the edge group containing $e$, without changing the gain. This implies that, without loss of generality, we can assume that the optimal monitor set $M^*$ in $G$ consists only of deputy edges. These edges remain in $G'$ and the gain of $M^*$ in $G'$ will be exactly the same as its gain in $G$. This shows the "$\geq$" inequality in (i). The "$\leq$" inequality follows from the fact that any monitor set in $G'$ consists only of deputy edges from $G$. The same argument implies (ii) as well.

**The kernel graph.** Consider a graph $G = (V, E)$ and a monitor edge set $M$, and let $B = Br(G - M)$. The *kernel graph associated with $G$ and $M$* is defined as

a weighted graph $G_M = (V_M, E_M)$, where $V_M$ is the set of connected components of $G - M - B$, and $E_M$ is determined as follows: For any edge $\{u, v\} \in M \cup B$, where $u, v \in V$, let $x$ and $y$ be the connected components of $G - M - B$ that contain, respectively $u$ and $v$. Then we add edge $\{x, y\}$ to $E_M$. The weights are preserved, that is $w(\{x, y\}) = w(\{u, v\})$. We will say that this edge $\{x, y\}$ *represents* $\{u, v\}$ or *corresponds to* $\{u, v\}$. In fact, we will often identify $\{u, v\}$ with $\{x, y\}$, treating them as the same object. Note that $G_M$ is a multigraph, as it may have multiple edges and loops (even though $G$ is a simple graph).

Figure 3 shows the kernel graph corresponding to the graph and the monitor set in the example from Figure 1 (all edge weights are 1):

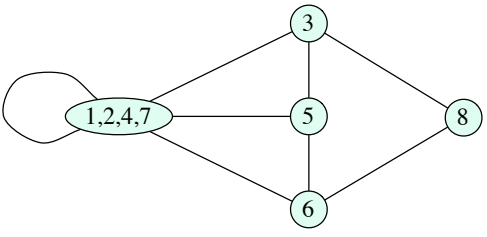



**Fig. 3.** The kernel graph for the example in Figure 1. The loop in vertex $\{1, 2, 4, 7\}$ represents edge $\{2, 1\}$.

Note that we have $|E_M| \leq k + |V_M| - cc(G_M)$, where $cc(H)$ denotes the number of connected components of a graph $H$. This can be derived directly from the definitions: The edges in $G_M$ that represent extra edges are the bridges in $G_M$ and therefore they form a forest in $G_M$. This implies that the number of extra edges is at most $|V_M| - cc(G_M)$, and the inequality follows.

In the paper, we will use the concept of kernel graphs only with respect to some optimal monitor set. Let $M^*$ be some arbitrary, but fixed, optimal monitor edge set. To simplify notation, we will write $G^* = (V^*, E^*)$ for the kernel graph associated with $M^*$, that is $G^* = G_{M^*}$, $V^* = V_{M^*}$ and $E^* = E_{M^*}$. In this notation, we have $gain^*(G, k) = w(E^*)$. In the analysis of our algorithms, we will be comparing the weights of edges collected by the algorithm against the edges in the kernel graph $G^*$.

## 3 Proof of NP-hardness of FlowMntrs

We show that the FlowMntrs is NP-hard (even in the unweighted case), via a reduction from the Clique problem. We start with a simple lemma whose proof is omitted. (In the lemma, we assume that $\binom{1}{2} = 1(1-0)/2 = 0$.)

**Lemma 1.** *Let $a_1, a_2, \ldots, a_s$ be $s$ positive integers such that $\sum_{i=1}^{s} a_i = n$, for a fixed integer $n$. Then $\sum_{i=1}^{s} \binom{a_i}{2}$ is maximized if and only if $a_j = n - s + 1$ for some $j$ and $a_i = 1$ for all $i \neq j$.*

**Theorem 2.** FLOWMNTRS *is* NP*-hard.*

*Proof.* In the CLIQUE problem, given an undirected graph $G = (V, E)$ and an integer $q > 0$, we wish to determine if $G$ has a clique of size at least $q$. CLIQUE is well-known to be NP-complete (see [2]). We show how to reduce CLIQUE, in polynomial-time, to DECFLOWMNTRS, the decision version of FLOWMNTRS, defined as follows: Given a graph $G = (V, E)$ and two integers, $k, l > 0$, is there a set $M$ of $k$ edges in $G$ for which $|Br(G - M)| \geq l$?

The reduction is simple. Suppose we have an instance $G = (V, E), q$ of CLIQUE. We can assume that $G$ is connected and $q \geq 3$. Let $n = |V|$ and $m = |E|$. We map this instance into an instance $G, k, l$ of DECFLOWMNTRS, where $k = m - \binom{q}{2} - l$ and $l = n - q$. This clearly takes polynomial time. Thus, to complete the proof, it is sufficient to prove the following claim:

$(*)$ $G$ has a clique of size $q$ iff $G$ has a set $M$ of $k$ edges for which $|Br(G-M)| \geq l$.

We now prove $(*)$. The main idea is that, by the choice of parameters $k$ and $l$, the monitors and extra edges in the solution of the instance of DECFLOWMNTRS must be exactly the edges outside the size-$q$ clique of $G$.

$(\Rightarrow)$ Suppose that $G$ has a clique $C$ of size $q$. Let $G'$ be the graph obtained by contracting $C$ into a single vertex and let $T$ be a spanning tree of $G'$. We then take $M$ to be the set of edges of $G'$ outside $T$. Thus the edges in $T$ will be the bridges of $G - M$. Since $G'$ has $n - q + 1$ vertices, $T$ has $l = n - q$ edges, and $M$ has $m - \binom{q}{2} - l = k$ edges.

$(\Leftarrow)$ Suppose there is a set $M$ of $k$ monitor edges that yields a set $B$ of $l'$ extra edges, where $l \leq l' \leq n - 1$. We show that $G$ has a clique of size $q$.

Let $s$ be the number of connected components of $G - M - B$, and denote by $a_1, a_2, ..., a_s$ the cardinalities of these components (numbers of vertices). Since $|B| = l'$, we have $s \geq l' + 1$. Also, $\sum_{i=1}^{s} a_i = n$ and $\sum_{i=1}^{s} \binom{a_i}{2} + k + l' \geq m$. Therefore, using Lemma 1, and the choice of $k$ and $l$, we have

$$\binom{n-l'}{2} + l' \;\geq\; \binom{n-s+1}{2} + l' \;\geq\; \sum_{i=1}^{s} \binom{a_i}{2} + l' \;\geq\; m - k \;=\; \binom{n-l}{2} + l.$$

By routine calculus, the function $f(x) = \frac{1}{2}(n - x)(n - x - 1) + x$ is decreasing in interval $[0, n - 1]$, and therefore the above derivation implies that $l' \leq l$, so we can conclude that $l' = l$. This, in turn, implies that all inequalities in this derivation are in fact equalities. Since the first inequality is an equality, we have $s - 1 = l' = l = n - q$. Then, since the second inequality is an equality, Lemma 1 implies that $a_j = q$ for some $j$ and $a_i = 1$ for all $i \neq j$. Finally, the last inequality can be an equality only if all the connected components are cliques. In particular, we obtain that the $j$th component is a clique of size $q$.

## 4 Algorithm $\sigma$-GREEDY

Fix some integer constant $\sigma \geq 1$. Let $G = (V, E)$ be the input graph with weights on edges. For simplicity, we will assume that $G$ is connected. (A full argument

will appear in the final version.) As explained in Section 2, we can then also assume that $G$ is 3-edge-connected.

Algorithm $\sigma$-GREEDY that we study in this section works in $\lceil k/\sigma \rceil$ steps and returns a set of $k$ monitor edges. In each step, it assigns $\sigma$ monitors to a set $P$ of $\sigma$ remaining edges that maximizes the gain in this step, that is, the total weight of the monitor edges and the resulting bridges. A more rigorous description is given in Figure 4, which also deals with special cases when the number of monitors or edges left is less than $\sigma$.

**Algorithm** $\sigma$-GREEDY
  $G_0 = (V, E_0) \leftarrow G = (V, E)$
  $M_0 \leftarrow \emptyset$
  $X_0 \leftarrow \emptyset$
  **for** $t \leftarrow 1, 2, ..., \lceil k/\sigma \rceil$
    **if** $E_{t-1} = \emptyset$
      **then** return $M = M_{t-1}$ and halt
    $\sigma' \leftarrow \sigma$
    **if** $t = \lfloor k/\sigma \rfloor + 1$
      **then** $\sigma' = k \bmod \sigma$
    **if** $|E_{t-1}| \leq \sigma'$
      **then** $P \leftarrow E_{t-1}$
      **else**
        find $P \subseteq E_{t-1}$ with $|P| = \sigma'$
          that maximizes $w(P \cup Br(G_{t-1} - P))$
        $Y_t \leftarrow P \cup Br(G_{t-1} - P)$
        $X_t \leftarrow X_{t-1} \cup Y_t$
        $E_t \leftarrow E_{t-1} - Y_t$
        $G_t \leftarrow (V, E_t)$
    $M_t \leftarrow M_{t-1} \cup P$
  return $M = M_{\lceil k/\sigma \rceil}$

**Fig. 4.** Pseudo-code for Algorithm $\sigma$-GREEDY. $Y_t$ represents the edges collected by the algorithm in step $t$, with $P \subseteq Y_t$ being the set of monitor edges and $Y_t - P$ the set of extra edges. $M_t$ represents all monitor edges collected up to step $t$ and $X_t$ represents all edges collected up to step $t$.

Note that each step of the algorithm runs in time $O(m^\sigma(n+m))$, by trying all possible combinations of $\sigma$ edges in the remaining graph $G_{t-1}$ to find $P$. Hence, for each fixed $\sigma$, Algorithm $\sigma$-GREEDY runs in polynomial time.

### 4.1 Analysis of 1-GREEDY

For $\sigma = 1$, Algorithm 1-GREEDY is: At each step, choose an edge whose removal creates a maximum number of bridges, and place a monitor on this edge. Then remove this edge and the resulting bridges. We show that this algorithm has approximation ratio 3.

**Analysis.** For simplicity, assume that the input graph $G$ is connected. As explained in previous sections, we can assume that $G$ is in fact 3-edge-connected.

Fix the value of $k$, and some optimal solution $M^*$, and let $G^* = (V^*, E^*)$ be the corresponding kernel graph. To avoid cluttered notation, we will identify each edge in $E^*$ with its corresponding edge in $E$, thus thinking of $E^*$ as a subset of $E$. For example, when we say that the algorithm collected some $e \in E^*$, we mean that it collected the edge in $E$ represented by $e$.

Recall that $gain^*(G, k) = w(E^*)$, where $w(E^*)$ is the sum of weights of the edges in $E^*$. Thus we need to show that our algorithm's gain is at least $\frac{1}{3}w(E^*)$.

Intuitively, since $G$ is 3-edge-connected, each vertex in $G^*$ has degree at least 3, so $|E^*| \geq \frac{3}{2}|V^*|$. Thus $k = |E^*| - |V^*| + 1 > \frac{1}{3}|E^*|$. 1-Greedy collects at least $k$ edges. Since 1-Greedy maximizes the gain at each step, its total gain will be at least the total weight of the $\frac{1}{3}|E^*|$ heaviest edges in $E^*$. (This does not mean, however, that 1-Greedy will collect the $\frac{1}{3}|E^*|$ heaviest edges.)

We now give a more rigorous argument. The proof is by amortized analysis. We will analyze consecutive steps of the algorithm, while maintaining a dynamic set $L_t$ of edges. Initially, we set $L_0 = E^*$. As the algorithm collects edges in each step $t$, we will also remove edges from $L_{t-1}$, so that $L_t \subseteq L_{t-1}$ for $t \geq 1$. In addition, this set $L_t$ will satisfy the following conditions for each step $t = 1, 2, ...$:

(L1.1) $L_t \cap X_t = \emptyset$; that is, all edges in $L_t$ are available to the algorithm after step $t$;
(L1.2) $w(Y_t) \geq \frac{1}{3}w(L_{t-1} - L_t)$; that is, our gain at each step is at least one third of the total weight of all the edges removed from $L_{t-1}$; and
(L1.3) $|L_{t-1}| - |L_t| \geq \min\{3, |L_{t-1}|\}$.

We claim that the conditions above imply that 1-Greedy's approximation ratio is 3. Since $k \geq \frac{1}{3}|E^*|$, from (L1.3) and amortization, we have $L_k = \emptyset$. Then, again by amortization, (L1.2) implies that $w(X_k) \geq \frac{1}{3}w(E^*)$, as claimed.

It thus remains to show how to update $L_t$ to maintain (L1.1), (L1.2) and (L1.3). Suppose that these conditions hold up to step $t-1$. Let $\gamma = |Y_t \cap L_{t-1}|$ be the number of edges collected in step $t$ that are in $L_{t-1}$. We first set $L_t \leftarrow L_{t-1} - Y_t$. Next, if $\gamma < 3$, we further remove arbitrary $3 - \gamma$ edges from $L_t$ (If it so happens that $L_t$ has fewer than $3 - \gamma$ edged, then we remove all edges from $L_t$.) Since we have removed all $Y_t$ from $L_{t-1}$, (L1.1) is preserved. Moreover, since we either remove at least 3 edges or $L_t = \emptyset$, (L1.3) is preserved as well. Finally, by the algorithm and (L1.1), $w(Y_t)$ is at least as large as the weight of each of the $3 - \gamma$ additional edges removed from $L_{t-1}$, which implies (L1.2).

Summarizing the argument above, we obtain:

**Theorem 3.** *Algorithm* 1-Greedy *is a polynomial-time* 3*-approximation algorithm for the* WFlowMntrs *problem.*

With a somewhat more careful analysis, one can show that the approximation ratio of 1-Greedy is actually $3(1-1/k)$, which matches our lower bound example below. Also, we remark that the proof above is perhaps unnecessarily technical,

but we introduce it here on purpose, as a stepping stone to a much more involved analysis of Algorithm 2-Greedy in the next section.

**A tight-bound example.** We now present an example showing that our analysis of 1-Greedy is tight. Graph $G$ consists of one connected component with $2k-2$ vertices, in which each vertex has degree 3 and each edge has weight 1, and the other connected component that has only two vertices connected by $k+2$ edges each of weight $1+\epsilon$. Fig. 5 shows the construction for $k=5$.

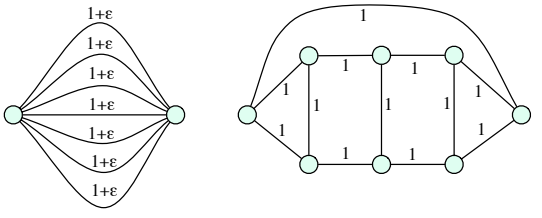



**Fig. 5.** Lower bound example for 1-Greedy, with $k=5$

1-Greedy will be collecting edges from the 2-vertex component on the left, ending up with $k$ edges and total gain $(1+\epsilon)k$. The optimum solution is to put $k$ monitors in the cubic component on the right, thus gaining all $3k-3$ edges from this component. For $\epsilon \to 0$, the approximation ratio tends to $3(1-1/k)$.

### 4.2 Analysis of 2-Greedy

Let $G=(V,E)$ be the input graph with weight on edges. As in the previous section we will assume that $G$ is 3-edge-connected. For $\sigma=2$, Algorithm 2-Greedy, at each step, collects two edges whose total weight combined with the weight of all resulting bridges, is maximized among all possible choices of two edges. Ties are broken arbitrarily. We place monitors on these two edges, and then remove them from $G$, as well as the resulting bridges. The exceptional situations, when $k$ is odd, or we run out of edges, etc., are handled as in Figure 4.

**Analysis.** We can assume that the algorithm never runs out of edges (that is, $E_{t-1} \neq \emptyset$ for each step $t$), for otherwise it computes the optimum solution. For simplicity, we will assume that $k$ is even. If $k$ is odd, the proof below can be shown to work by taking into account the gain of the algorithm in the last step when it has only one monitor. We also fix some optimal solution $M^*$, and let $G^*=(V^*,E^*)$ be the corresponding kernel graph. Recall that $gain^*(G,k) = w(E^*)$; thus we need to show that our algorithm's gain is at least $\frac{1}{2}w(E^*)$.

Before we delve into formal proof, we give a high level description of our approach. We start with a set $L$ which contains two copies of every edge in $E^*$, so that $w(L) = 2w(E^*) = 2gain^*(G,k)$. It thus suffices to show that the algorithm's gain in $k/2$ steps is at least $w(L)/4$. In the analysis of one step we remove some copies of edges from $L$, while guaranteeing that their total weight is at most 4 times 2-Greedy's gain in this step. Then we show that after $k/2$

steps $L$ becomes empty. Hence the algorithm must have collected a set of edges with total weight at least $w(L)/4$, as needed. Below we give the complete proof.

As in the previous section, the proof is by amortized analysis. If $e = \{x, y\}$ is an edge, then by $e^x$ and $e^y$ we will denote two copies of $e$, and we will call them *arcs*. We also say that arcs $e^x$ and $e^y$ are *associated* with $e$. One can think of these arcs as directed edges, with $e^x = (x, y)$ directed from $x$ to $y$, and $e^y$ directed in the opposite direction. Each arc $e^x$ has weight $w(e^x) = w(e)$. In our analysis, we maintain a dynamic set $L_t$ of arcs, for each step $t$, that will satisfy invariants similar to those in the previous section. Initially, $L_0$ contains two copies of each edge, that is $L_0 = \{e^x, e^y : e = \{x, y\} \in E^*\}$. Note that $|L_0| = 2|E^*|$ and $w(L_0) = 2w(E^*)$. As we analyze each step $t$ of the algorithm, we will be removing some arcs from $L_{t-1}$, so that we will have $L_t \subseteq L_{t-1}$ for each $t = 1, 2, ..., k/2$. $L_t$ will not contain any arcs associated with edges already collected by the algorithm in the first $t$ steps. For an edge in $E^*$, $L_t$ may contain both its associated arcs, one of the two arcs, or neither of them.

Let $x$ be a vertex of degree-3 in $G^*$ and $e$, $f$ and $g$ be the three edges incident to $x$. We will say that $x$ is a *tripod at step t* if none of edges $e$, $f$, $g$ have been collected by 2-Greedy when step $t-1$ completes. Arcs $e^x$, $f^x$, $g^x$ are then called *tripod arcs* or *arcs of* $x$. We say that $x$ is a *bipod at step t* if exactly one of these edges has been collected by 2-Greedy. (Note that the case when exactly two of these edges was collected is not possible since any two collected by the algorithm implies the algorithm also gets the third one.) If, say, $g$ is the one edge collected by 2-Greedy, then $e^x$ and $f^x$ are called *bipod arcs* or *arcs of* $x$.

Let $x$ be a tripod at step $t$. If $A \subseteq L_0$ is a set of arcs, we say that $x$ *is in* $A$ if its three arcs $e^x, f^x, g^x$ are in $A$. Similarly, if $x$ is a bipod at time $t$, we say that $x$ *is in* $A$ if both its arcs $e^x, f^x$ are in $A$. Let $\xi(A)$ be the total number of tripods and bipods in $A$. It will be convenient to have some name for arcs in $A$ that are neither tripod arcs nor bipod arcs. We refer to such arcs as *loose arcs in* $A$. Note that for an edge $e = \{x, y\}$, arc $e^x \in A$ could be a loose arc while the opposite arc $e^y$ is a tripod arc or a bipod arc.

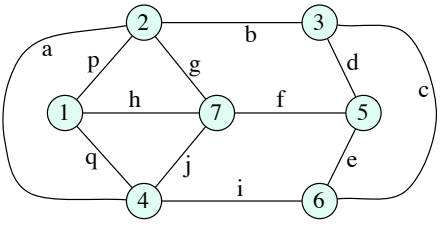



**Fig. 6.** Example of a graph $G^*$

Consider, for example, the graph $G^*$ in Figure 6, where all weights are 1. For this graph, we would have $L_0 = \{a^2, a^4, b^2, b^3, c^3, c^6, \ldots, p^1, p^2, q^1, q^4\}$. The tripods of $L_0$ are vertices 1, 3, 5 and 6, so $\xi(L_0) = 4$. By definition of bipod, the initial set $L_0$ does not contain any bipods. All arcs in $L_0$ that do not belong to a tripod are loose arcs, for example $q^4$, $a^4$, $j^7$, etc. If the algorithm picks

edges $b$, $d$ in step 1, $c$ will become a bridge, so the gain is 3. In the analysis, all arcs associated with these edges will be removed from $L_0$. Assuming that in the analysis we will not remove any arcs associated with edges $f$, $e$ and $i$, $L_1$ will contain two bipods, 5 and 6.

In our analysis, we will assume for now that $|L_0| - \xi(L_0)$ is even; later we will explain how to modify the proof to cover the case when this quantity is odd. As mentioned earlier, the overall idea of the proof is to construct a decremental sequence $L_0 \supseteq L_1 \supseteq L_2 \supseteq ...$ that satisfies appropriate conditions, from which we can derive a bound on the gain of 2-Greedy. In fact, we will actually construct the sets $\Delta_t \subseteq L_{t-1}$ of arcs to be removed at each step, so that $L_t = L_{t-1} - \Delta_t$. We claim that for any given step $t$, $1 \le t \le k/2$. there exists a set of arcs $\Delta_t \subseteq L_{t-1}$ that satisfies the following conditions:

(D2.1) If $e^x \in L_{t-1}$ and $e \in Y_t$ then $e^x \in \Delta_t$.
(D2.2.) $w(Y_t) \ge \frac{1}{4} w(\Delta_t)$.
(D2.3) Either (a) $|\Delta_t| - \xi(\Delta_t) \ge 8$ and $|\Delta_t| - \xi(\Delta_t)$ is even, or (b) $|\Delta_t| - \xi(\Delta_t) < 8$ and $\Delta_t \cup \Delta_{t+1} = L_{t-1}$ (in other words, we will remove all arcs in this and next step).
(D2.4) For any bipod $x$ in $L_{t-1}$, if one arc of $x$ is in $\Delta_t$, then both of them are in $\Delta_t$. For any tripod $x$ in $L_{t-1}$, if two of the arcs of $x$ are in $\Delta_t$, then all three arcs are in $\Delta_t$.

Recall that $Y_t$ is the set of edges collected by the algorithm at step $t$, including the two monitor edges and the resulting bridges. Condition (D2.1) ensures that $L_t$ contains only arcs whose associated edges are available to the algorithm right after step $t$. Condition (D2.2) states that 2-Greedy's gain in this step is at least $\frac{1}{4}$th of the total weight of arcs removed from $L_{t-1}$. Condition (D2.3) says that we remove a sufficient number of arcs from $L_{t-1}$ at step $t$, guaranteeing that we will empty $L_t$ at or before step $k/2$. The parity condition in (D2.3) and condition (D2.4) are of more technical nature, and their significance will become apparent in the construction of $\Delta_t$ below.

Before explaining how to construct such a set $\Delta_t$, we first show that the existence of $\Delta_t$ implies the 2-approximation of 2-Greedy. As explained earlier, we define $L_t = L_{t-1} - \Delta_t$, for any $t = 1, 2, ....$ Denoting by $\eta_d$ the number of vertices in $G^*$ of degree $d$, we have

$$\begin{aligned} |L_0| - \xi(L_0) &= \textstyle\sum_{d\ge 3} d\eta_d - \eta_3 \;=\; 2\eta_3 + \sum_{d\ge 4} d\eta_d \;\le\; 2\sum_{d\ge 3}(d-2)\eta_d \\ &= 4\cdot(\tfrac{1}{2}\textstyle\sum_{d\ge 3} d\eta_d - \sum_{d\ge 3}\eta_d) \;=\; 4(|E^*| - |V^*|) \;\le\; 4k-4. \quad (1) \end{aligned}$$

From invariant (D2.3), by amortization over all steps, we have $|L_{k/2}| - \xi(L_{k/2}) \le \max\{0\,,\, |L_0| - \xi(L_0) - 4k\} \le 0$. On the other hand, for all $t$, whenever $L_t \ne \emptyset$, we have $|L_t| > \xi(L_t)$. We thus conclude that $L_{k/2} = \emptyset$. This, together with condition (D2.2) and amortization, implies that $w(X_{k/2}) \ge \frac{1}{4} w(L_0) = \frac{1}{2} w(E^*) = \frac{1}{2} gain^*(G,k)$. Here $X_t = \bigcup_{j=1}^{t} Y_t, t = 1, 2, \ldots, k/2$ is the set of edges collected by the algorithm up to and include step $t$. Thus 2-Greedy approximates the optimum solution within a factor of 2.

To complete the analysis, it remains to show how to construct a set $\Delta_t$ that satisfies conditions (D2.1) to (D2.4). Suppose we have already constructed sets $\Delta_s$, for $s = 1, 2, ..., t-1$. Given the definition of $L_0$, the assumption that $|L_0| - \xi(L_0)$ is even, as well as conditions (D2.1), (D2.3) together imply that $L_{t-1}$ satisfies the following three conditions:

(L2.1) If $e^x \in L_{t-1}$ then $e \notin X_{t-1}$.
(L2.2) $|L_{t-1}| - \xi(L_{t-1})$ is even.

We start with a high level idea. We first include in $\Delta_t$ all arcs in $L_{t-1}$ associated with edges in $Y_t$, the set of edges collected at step $t$. This will satisfy condition (D2.1). Note that removing additional arcs from the dynamic set $L$ will not violate (D2.1). The total weight of these arcs will be at most $2w(Y_t)$, since every edge is associated with at most two arcs and every arc removed is associated with some edge in $Y_t$. Thus, at least so far, condition (D2.2) holds as well. To satisfy condition (D2.3), we may need to include more arcs to $\Delta_t$. This requires that we keep a delicate balance between the number of additional arcs to be included and their total weight: If we include too many arcs, we may violate (D2.2) because the total weight of arcs in $\Delta_t$ is too large. On the other hand, if we include too few arcs, we may not be able to satisfy (D2.3) which requires $|\Delta_t| - \xi(\Delta_t)$ to be at least 8.

Thus we will have $\Delta_t = \Delta'_t \cup \Delta''_t$, where $\Delta'_t$ and $\Delta''_t$ are the arcs from $L_{t-1}$ removed in Stage 1 and Stage 2, respectively. We now describe these two stages.
*Stage 1: removing affected arcs.* We set $\Delta'_t$ to be the set of all arcs $e^x \in L_{t-1}$ such that $e \in Y_t$. These arcs can be grouped into the following four categories:

Case (I): Loose arcs. Any loose arc $e^x \in L_{t-1}$ associated with $e \in Y_t$, contributes 1 to $|\Delta'_t| - \xi(\Delta'_t)$.
Case (II): Triples of tripod arcs. Suppose that $x$ is a tripod in $L_{t-1}$ and $e^x$, $f^x$, $g^x$ are its arcs. If $e, f, g \in Y_t$, then the arcs of $x$ will contribute 2 to $|\Delta'_t| - \xi(\Delta'_t)$.
Case (III): Single tripod arcs. Suppose that $x$ is a tripod in $L_{t-1}$ and $e^x$, $f^x$, $g^x$ be its arcs. If $e \in Y_t$ but $f, g \notin Y_t$, then $e^x$ contributes 1 to $|\Delta'_t| - \xi(\Delta'_t)$.
Case (IV): Pairs of bipod arcs. Suppose that $x$ is a bipod in $L_{t-1}$ and $e^x$, $f^x$ are its arcs. If $e, f \in Y_t$, then the two bipod arcs of $x$ will together contribute 1 to $|\Delta'_t| - \xi(\Delta'_t)$.

Note that the above four cases exhaust all possibilities. Clearly (D2.1) holds. (D2.2) is true because each edge in $Y_t$ is associated with at most two arcs. (D2.4) follows from the algorithm since, if it collects one edge of a bipod, then it also collects the other; if it collects two of three edges of a tripod, then the third one is collected as well. So only (D2.3) needs further attention. In Stage 2 we shall include additional arcs in $\Delta_t$ to satisfy (D2.3).

*Stage 2: removing additional arcs.* Now we choose a set $\Delta''_t \subseteq L_{t-1} - \Delta'_t$ of additional arcs that will be removed from $L_{t-1}$. Let $L' = L_{t-1} - \Delta'_t$, and define $\delta' = (|L_{t-1}| - \xi(L_{t-1})) - (|L'| - \xi(L')) = |\Delta'_t| - \xi(\Delta'_t)$. We have two cases, depending on the value of $\delta'$.
Case 1: $\delta'$ is even. If $\delta' \geq 8$, then we are done. Now consider subcases $\delta' = 2, 4, 6$.

Case 1.1: $\delta' = 2$. In this sub-case, $\Delta'_t$ may contain either one tripod, two independent bipods, or one bipod and one arc that are independent, or two independent arcs. Here tripods and bipods are with respect to $L_{t-1}$, arc may be a tripod arc or a loose arc, and independent simply means they do not contain two arcs associated with the same edge. In every case we have $w(\Delta'_t) \leq w(Y_t)$ due to the algorithm. Now we need to bring down $|L| - \xi(L)$ further by 6. Recall $L$ is the dynamic set of arcs that we are maintaining in the analysis of the algorithm. It is easy to see that every tripod in $L' = L_{t-1} - \Delta'_t$ has weight at most $w(Y_t)$ and each counts 2 toward $|L| - \xi(L)$. So if there are enough tripods in $L'$, then we may include 3 tripods in $\Delta''_t$ and we are done. If we run out of tripods, then we may use bipods and loose arcs. Any two independent loose arcs or bipods have total weight no more than $w(Y_t)$ and bring down $|L| - \xi(L)$ by 2. One potential issue is that a bipod(call it *new* bipod) might result from removing a tripod arc in constructing $\Delta'_t$ in Stage 1. Since $\delta' = 2$, this new bipod together with the tripod arc will have total weight no more than $w(Y_t)$ and bring down $|L| - \xi(L)$ by 2, while the other arc or bipod in $\Delta'_t$ together with the other arc or bipod(could be a new bipod too) chosen in stage 2 so far will have weight no more than $w(Y_t)$(Either they both belong to the same tripod in $L_{t-1}$ or they are they are independent, as all arcs or bipods in stage 1 are independent of bipods or arcs in stage 2) and also bring down $|L| - \xi(L)$ by 2. Overall we are removing arcs with total weight at most $2w(Y_t)$ and bring down $|L| - \xi(L)$ by 4, and this is the same as if the new bipod were a bipod in $L_{t-1}$. We may further look for two tripods or two pairs of independent bipods(cannot be new bipods) or loose arcs until we have $\Delta''_t$ such that $\delta'' = |\Delta''_t| - \xi(\Delta''_t) = 6$ and $w(\Delta''_t) \leq 3w(Y_t)$. If the above completes successfully, then we have $w(\Delta_t) = w(\Delta'_t) + w(\Delta''_t) \leq w(Y_t) + 3w(Y_t) = 4w(Y_t)$ and $\delta = \delta' + \delta'' = 8$ as desired.

It is also possible that we do not have a pair of independent bipods or loose arcs while $\delta''$ is still less than 6. And this happens after we removed arcs in $\Delta'_t$ and $\Delta''_t$ from the dynamic set $L$. We first note that this can happen only when $|L| - \xi(L) \leq 3$. The reason is that, as long as $|L| - \xi(L) \geq 4$, and $L$ contains no tripods, we know the count of bipods and the count of loose arcs combined must be at least 4. But a bipod can depend on at most two bipods or loose arcs, hence it must be independent of one of the three bipods or loose arcs. So to be in this situation we have $|L| - \xi(L) \leq 3$ and $L$ has no tripods. The set $L$ contains either 3 interlocked bipods forming a 3-cycle, or two dependent bipods or loose arcs. And it is not hard to see that $w(L) \leq 3w(Y_t)$. If we have one more step to go, then in the next step $t+1$ the algorithm gets a gain of at least the weight of one bipod or loose arc that is in $L$, and we shall have $w(Y_{t+1}) \geq w(L)/4$ since there are at most 3 bipods or loose arcs, and the set $L$ becomes empty after those dependent bipods and arcs are removed. Otherwise we are in the last step, step $k/2$. Recall inequality (1) is actually $|L_0| - \xi(L_0) \leq 4(k-1)$. Given that we have kept $|L| - \xi(L)$ down by at least 8 in all previous steps except the last step, we must have $|L_{k/2-1}| - \xi(L_{k/2-1}) \leq 4$. It follows that after stage 1 of step $k/2$, we shall have $|L| - \xi(L)$ no more than 2. In this case we can simply include all the dependent bipods or loose arcs in $\Delta''_t$. Their total weight is no

more than $3w(Y_t)$ and we have emptied the set $L$, while maintained (D2.2) since $w(\Delta_{k/2}) = w(\Delta'_{k/2}) + w(\Delta''_{k/2}) \leq w(Y_t) + 3w(Y_t) \leq 4w(Y_t)$ as desired.
Case 1.2: $\delta' = 4$ and $w(\Delta'_t) \leq 2w(Y_t)$. If we can find 2 tripods in $L' = L_{t-1} - \Delta'_t$, then we are done since each tripod contributes 2 to $\delta''$ and its weight is no more than $w(Y_t)$. Otherwise we may use bipods and loose arcs. Because all bipods and arcs in $L' = L_{t-1} - \Delta'_t$ are independent of bipods or arcs in $\Delta'_t$, we can pair up bipods or arcs in $\Delta'_t$ and bipods or arcs in $L'$. Each pair is either from the same tripod in $L_{t-1}$ or the two in that pair are independent with no new bipods, hence each pair contributes 2 to $\delta''$ and its weight is no more than $w(Y_t)$. So we are able to construct $\Delta''_t$ such that $w(\Delta'_t) + w(\Delta''_t) \leq 4w(Y_t)$ and $\delta' + \delta'' = 8$. Additional pairs of indepdent bipods or loose arcs may be needed to construct $\Delta''_t$ just described. In case we do not have independent pairs, we argue as in $\delta' = 2$ subcase that we are able to empty the set $L$ in the next step.

$\delta' = 6$ and we have $w(\Delta'_t) \leq 2w(Y_t)$. We simply include one tripod, or two bipods or loose arcs in $\Delta''_t$. The total weight is no more than $2w(Y_t)$, and we have $\delta'' = 2$. Overall we have $w(\Delta_t) = w(\Delta'_t) + w(\Delta''_t) \leq 2w(Y_t) + 2w(Y_t) = 4w(Y_t)$ and $\delta = \delta' + \delta'' = 6 + 2 = 8$ as desired. If we are not able to find any tripods, bipods and loose arcs, then effectively we have already emptied the dynamic set $L$ while maintained (D2.2), that is, total weight of arcs removed from the set $L$ at each step is no more than 4 times the gain of the algorithm at that step.

The analysis of the case when $\delta'$ is odd, as well as other the remaining cases, will appear in the full paper. Summarizing, we obtain our main result.

**Theorem 4.** *Algorithm* 2-Greedy *is a polynomial-time* 2*-approximation algorithm for the Weighted Flow Edge-Monitor Problem.*

**A tight-bound example.** The example is essentially the same as the one for 1-Greedy (see Figure 5), except that the edges in the 2-vertex component on the left side have now weights $1.5 + \epsilon$. 2-Greedy will collect edges from this 2-vertex component, so its total gain will be $(1.5 + \epsilon)k$, while the optimum gain is $3k - 3$. For $\epsilon \to 0$ and $k \to \infty$, the ratio tends to 2.

## 5 Final Comments

The most intriguing open question is what is the approximation ratio of $\sigma$-Greedy in the limit for $\sigma \to \infty$. We can show that this limit is not lower than 1.5, and we conjecture that 1.5 is indeed the correct answer.

A natural question to ask is whether our results can be extended to directed graphs. It is not difficult to show that this is indeed true; both the NP-hardness proof and 2-approximation can be adapted to that case.

Another direction to pursue would be to study the extension of our problem to arbitrary linear systems of equations. Here we can put $k$ "monitors" on $k$ variables of the system to measure their values. The objective is to maximize the number of variables whose values can be uniquely deduced.

## References

1. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The Complexity of Multiway Cuts (Extended Abstract). In: 24th ACM Symposium on Theory of Computing, pp. 241–251 (1992)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
3. Goldschmidt, O., Hochbaum, D.S.: Polynomial Algorithm for the k-Cut Problem. In: 29th Annual IEEE Symposium on Foundations of Computer Science, pp. 444–451 (1988)
4. Gu, W., Jia, X.: On a Traffic Control Problem. In: 8th International Symposium on Parallel Architectures, Algorithms and Networks, pp. 510–515 (2005)
5. Khuller, S., Bhatia, R., Pless, R.: On Local Search and Placement of Meters in Networks. SIAM J. on Comput. 32, 470–487 (2003)
6. Saran, H., Vazirani, V.V.: Finding k-cuts within Twice the Optimal. In: 32nd Annual Symposium on Foundations of Computer Science, pp. 743–751 (1991)
7. Tsin, Y.H.: A Simple 3-Edge-Connected Component Algorithm. Theor. Comp. Sys. 40, 125–142 (2007)
8. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001)

# Three Results on Frequency Assignment in Linear Cellular Networks
## (Extended Abstract)

Marek Chrobak[1,⋆] and Jiří Sgall[2,⋆⋆]

[1] Department of Computer Science, University of California, Riverside, CA 92521, USA
[2] Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, CZ-11800 Praha 1, Czech Republic

**Abstract.** In the frequency assignment problem we are given a graph representing a wireless network and a sequence of requests, where each request is associated with a vertex. Each request has two more attributes: its arrival and departure times, and it is considered active from the time of arrival to the time of departure. We want to assign frequencies to all requests so that at any time no two active requests associated with the same or adjacent nodes use the same frequency. The objective is to minimize the number of frequencies used.

We focus exclusively on the special case of the problem when the underlying graph is a linear network (path). For this case, we consider both the offline and online versions of the problem, and we present three results. First, in the incremental online case, where the requests arrive over time, but never depart, we give an algorithm with an optimal (asymptotic) competitive ratio $\frac{4}{3}$. Second, in the general online case, where the requests arrive and depart over time, we improve the current lower bound on the (asymptotic) competitive ratio to $\frac{11}{7}$. Third, we prove that the offline version of this problem is $\mathbb{NP}$-complete.

## 1 Introduction

*The frequency assignment problem.* In a wireless network, the coverage area is divided into cells, with each cell covered by a transmitter. Each user within a given cell is assigned a unique frequency for communicating with the transmitter. In order to avoid interferences, it is also necessary to ensure that any pair of adjoining cells uses different sets of frequencies. The set of available frequencies is a limited resource; thus the frequency assignment policy needs to attempt to minimize the total number of assigned frequencies.

In the *static* setting, if the set of users in each cell is fixed, we can model this problem as a variation of graph coloring, where the network is represented

⋆ Research supported by NSF Grant CCF-0729071.
⋆⋆ Partially supported by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR) and grant IAA100190902 of GA AV ČR.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 129–139, 2009.

by an undirected graph $G$, whose vertices represent the cells and edges connect adjacent cells. We are given an integer demand $w_v \geq 0$ (the number of users in cell $v$) for each vertex $v$. We wish to assign a set $C_v$ of $w_v$ colors to each vertex $v$, such that $C_u \cap C_v = \emptyset$ for all adjacent vertices $u, v$. The objective is to find such a color assignment that minimizes the total number of colors used.

It is natural to consider graphs $G$ with a regular structure. For example, in the literature it is commonly assumed that the cells are regular hexagons in the plane, in which case $G$ is a triangular grid graph. As shown by McDiarmid and Reed [4] the offline frequency assignment problem for such graphs is $\mathbb{NP}$-hard. They also gave a polynomial-time $\frac{4}{3}$-approximation algorithm for this version. Another $\frac{4}{3}$-approximation algorithm was designed independently by Narayanan and Shende [5].

*Dynamic setting.* Of course, in practice, the set of users in each cell is dynamic – users arrive and leave over time. Thus it is natural to study the frequency assignment in the *dynamic* setting. Here, each frequency request (user) has three attributes: a node $v$ where the request is issued, the arrival time and the departure time. The request is *active* between its arrival and departure times. We need to assign frequencies to all requests in such a way that at each time step, frequencies assigned to active requests in the same or adjacent nodes are different. The objective function is the total number of frequencies used.

*Online algorithms.* For the dynamic setting described above, online algorithms are of particular significance, since in realistic applications users' arrivals and departures are unknown and unpredictable. In the online scenario, the arrival and departure events come over time, and the online algorithm has to react immediately; once a frequency is assigned to a request, it cannot be changed.

For an online frequency assignment algorithm $\mathbf{A}$, let $\mathbf{A}(I)$ be the number of frequencies used by $\mathbf{A}$ on an instance $I$, and let $opt(I)$ be the minimum number of frequencies required for $I$. We define $\mathbf{A}$ to be *$R$-competitive*, if there exists a constant $B$ independent of $I$ such that $\mathbf{A}(I) \leq R \cdot opt(I) + B$. When $B = 0$, we say that the ratio $R$ is *absolute*; otherwise, if we need to distinguish the cases, we say that the ratio $R$ is *asymptotic*. In this paper, we study the asymptotic competitive ratio.

In the *incremental* version of this problem, it is assumed that requests, once issued, last forever. This version corresponds to the static version of the offline problem, and it has been intensely studied. Chan *et al.* [1] derive some bounds on the asymptotic competitive ratio for hexagonal-cells graphs, proving that the optimal ratio is between 1.5 and 1.9126. For the absolute ratio, they give a tight bound of 2. More generally, for $\xi$-colorable graphs, they show an upper bound of $(\xi + 1)/2$.

Again, special classes of graphs can be studied. One natural and simple case, and the focus of our study, is that of a linear network (*path*), where the nodes are represented by integer points on the real line and two nodes are adjacent if they are at distance 1. For this case, it is known that the asymptotic competitive ratio is between $\frac{4}{3} \approx 1.333$ and 1.5, while the optimal absolute ratio is equal to 1.5, see [2].

In the general version of the online problem departures are allowed, as in the dynamic version of the offline problem. For the problem on the path, Chan *et al.* [2] proved that the asymptotic competitive ratio is between $\frac{14}{9} \approx 1.556$ and $\frac{5}{3} \approx 1.667$, while the absolute competitive ratio is 1.5. Both upper bounds are achieved by a simple greedy strategy which always assigns the smallest frequency that can be used at that time.

*Our work.* We first consider the incremental case of the online problem on the path. We give an upper bound of 4/3, matching the lower bound. The idea of our algorithm is based on a technique that spreads the requests evenly between four mutually overlapping sets of points, such that within each set we can allocate frequencies optimally with respect to this set. As each point belongs to three sets, the ratio 4/3 will follow.

Next, we consider the general online case, where departures are allowed. For this case we prove a lower bound of $11/7 \approx 1.571$, improving the currently best bound of $14/9 \approx 1.556$ in [2]. Table 1 summarizes the known bounds for online algorithms for the path.

**Table 1.** Competitive ratios for the case of a path. Bounds from this paper are in boldface; all other bounds are from [2].

| Case | incremental | | with departures | |
|---|---|---|---|---|
| Ratio | absolute | asymptotic | absolute | asymptotic |
| lower bound | 1.5 | $4/3 \approx 1.333$ | $5/3 \approx 1.667$ | $\mathbf{11/7 \approx 1.571}$ |
| upper bound | 1.5 | $\mathbf{4/3 \approx 1.333}$ | $5/3 \approx 1.667$ | $5/3 \approx 1.667$ |

In the incremental case (on the path), it is easy to determine the value of the optimum: It is simply equal to the maximum number of frequencies on two adjacent nodes. This turns out to be false when departures are allowed. Our third result is that computing this optimum is $\mathbb{NP}$-hard. This proof uses a reduction from 3-coloring of planar graphs, and it shows that even deciding if three frequencies are sufficient is hard. Thus also achieving a better absolute approximation ratio than 4/3 is $\mathbb{NP}$-hard. This result complements the $\mathbb{NP}$-hardness result in [4] and the offline upper bounds in [4,5]. It also indicates that establishing the optimal competitive ratio for this case is likely to be more challenging, since it implies that there is probably no simple way to keep track of the optimum solution.

## 2 Preliminaries

We identify points on the path with integers, $v = \ldots, -2, -1, 0, 1, \ldots$. Frequencies are denoted by positive integers. At each step, a request can be issued at some point $v$. We then need to assign a frequency to this request that is different from all frequencies already assigned to the active requests at points $v - 1$, $v$ and $v + 1$. The objective is to minimize the number of frequencies used at the same time.

For each point $v$, let $L_v$ be the (dynamic) set of frequencies assigned to $v$ by the algorithm and $\ell_v = |L_v|$. Thus $\ell_v$ is simply the number of active requests at $v$ at a given time. A frequency $f$ is called *admissible* for $v$ if $f \notin L_{v-1} \cup L_v \cup L_{v+1}$.

To estimate the performance of an algorithm, we measure the number of frequencies used. This is equivalent to measuring the maximum used frequency, as is done in some literature. Clearly, the number of frequencies cannot exceed the maximum frequency. On the other hand, any algorithm can be modified to use only consecutive integers. This is trivial in the incremental case, as we can simply renumber the frequencies in the order in which they appear. Similarly, in the general case, if the online algorithm is about to use a frequency $f$ not used by any active request at this time, we change it to use the lowest unused frequency $f'$, and fix the mapping $f \mapsto f'$ for as long as there are some active requests with frequency $f$.

*The optimum.* As observed in [2], in the static case – and thus also in the incremental online case – the (offline) optimum number of frequencies is

$$\omega = \max_v \{\ell_v + \ell_{v+1}\}.$$

Indeed, the "$\geq$" bound is trivial. To see that "$\leq$" bound holds, we can assign frequencies to nodes as follows: If $v$ is even, assign to it frequencies $1, 2, \ldots, \ell_v$, and if $v$ is odd, assign to it frequencies $\omega, \omega - 1, \ldots, \omega - \ell_v + 1$. Then no two adjacent nodes will be assigned the same frequency.

## 3 An Upper Bound for the Incremental Online Case

*Algorithm* FourBuckets. We partition all available frequencies $1, 2, 3, \ldots$ into four disjoint infinite *buckets* denoted $B_\sigma$, for $\sigma = 0, 1, 2, 3$. The frequencies in bucket $B_\sigma$ are denoted $1^\sigma, 2^\sigma, \ldots$, and are assumed to be ordered in this way, with $1^\sigma$ being the lowest one. How the partition into buckets is defined is not important. For example, one such partition can be achieved by defining $x^\sigma = \sigma + 4x - 3$, for each integer $x \geq 1$. For any node $v$ and $\sigma \in \{0, 1, 2, 3\}$, we say that $\sigma$ is *associated* with $v$ if $\sigma \not\equiv v \pmod 4$. Thus each node has three out of four buckets associated with it.

Suppose that a request is issued at a node $v$. Choose any $\sigma \in \{0, 1, 2, 3\}$ associated with $v$ that minimizes $|L_v \cap B_\sigma|$, and assign to this request the lowest frequency $f^\sigma$ from $B_\sigma$ admissible at $v$.

*Analysis.* The general idea is this: By the assignment of buckets to nodes, each bucket is associated with groups of exactly three consecutive nodes on the path. In each bucket, the algorithm is equivalent to the greedy algorithm, which is optimal for paths of three vertices. At each node, the algorithm spreads the requests evenly among the three buckets associated with this node, so each bucket gets about one third of all requests at each node. This implies that the number of frequencies used by the algorithm in each bucket is about $\frac{1}{3}$ of the optimum. Multiplying by the number of buckets, we conclude that the competitive ratio is $\frac{4}{3}$.

Now we give a formal argument. Let $L_{\sigma,v} = |L_v \cap B_\sigma|$ be the set of frequencies in $B_\sigma$ assigned to requests at a node $v$, and $\ell_{\sigma,v} = |L_{\sigma,v}|$ be the cardinality of $L_{\sigma,v}$. For any subset $X \subseteq B_\sigma$, we use notation $\max^\sigma(X)$ for the maximum integer $h$ such that $h^\sigma \in X$; for the empty set we put $\max^\sigma(\emptyset) = 0$. The following lemma subsumes the proof that the greedy algorithm is optimal for paths of three vertices.

**Lemma 1.** *For each $\sigma$ and $v$, $\max^\sigma(L_{\sigma,v}) \le \ell_{\sigma,v} + \max\{\ell_{\sigma,v-1}, \ell_{\sigma,v+1}\}$.*

*Proof.* Without loss of generality, we can assume that $\sigma = 0$ and $v \in \{0,1,2,3\}$.

For $v = 0$, since $\sigma$ is not associated with $v$, we have $L_{0,0} = \emptyset$, so $\max^0(L_{0,0}) = 0$, and the lemma holds trivially.

Suppose $v = 1$, and let $h = \max^0(L_{0,1})$. By the algorithm, each frequency $1^0, \ldots, h^0$ is either in $L_{0,1}$ or $L_{0,2}$, as otherwise the algorithm would not use $h^0$. Of course, no frequency is in both sets. So $h \le \ell_{0,1} + \ell_{0,2}$.

The case $v = 3$ is symmetric to the previous one.

Finally, consider the case $v = 2$, and let $h = \max^0(L_{0,2})$. Let $\bar{h} = \max^0(L_{0,1} \cup L_{0,3})$; by symmetry, we can assume $\bar{h} = \max^0(L_{0,1})$. Like in the previous case, each frequency $1^0, \ldots, \bar{h}^0$ is in either $L_{0,1}$ or $L_{0,2}$, but not in both. When $h^0$ is assigned, each frequency $\bar{h}^0 + 1, \ldots, h^0$ is in $L_{0,1} \cup L_{0,2} \cup L_{0,3}$, as otherwise we would not use $h^0$. However, by the definition of $\bar{h}$, none of these frequencies is in $L_{0,1} \cup L_{0,3}$, thus all of them are in $L_{0,2}$. Therefore $h \le \ell_{0,1} + \ell_{0,2}$ again.

**Theorem 1.** *Algorithm* FourBuckets *is asymptotically $\frac{4}{3}$-competitive for the incremental frequency assignment on a path.*

*Proof.* Consider any node $v$ and any $\sigma$. By the algorithm, if $\sigma$ is not associated with $v$ then $\ell_{\sigma,v} = 0$. On the other hand, if $\sigma$ and $\sigma'$ are associated with $v$ then $\ell_{\sigma,v} \le \ell_{\sigma',v} + 1$. This implies that for each $\sigma$ associated with $v$ we have $\ell_{\sigma,v} \le \lceil \ell_v/3 \rceil \le \frac{1}{3}(\ell_v + 2)$. Therefore, using Lemma 1, we get

$$\begin{aligned}
\max^\sigma(L_{\sigma,v}) &\le \ell_{\sigma,v} + \max\{\ell_{\sigma,v-1}, \ell_{\sigma,v+1}\} \\
&\le \tfrac{1}{3}(\ell_v + 2) + \tfrac{1}{3}\max\{\ell_{v-1} + 2, \ell_{v+1} + 2\} \\
&\le \tfrac{1}{3}[\ell_v + \max\{\ell_{v-1}, \ell_{v+1}\} + 4] \\
&\le \tfrac{1}{3}(\omega + 4).
\end{aligned}$$

Thus $|B_\sigma| = \max_v \max^\sigma(L_{\sigma,v}) \le \frac{1}{3}(\omega + 4)$ as well, and we can conclude that the total number of frequencies used in all four buckets is at most $\frac{4}{3}(\omega + 4)$.

## 4 A Lower Bound for the General Online Case

To obtain an improved lower bound, we modify the idea from [2, Section 4.2], which we first describe informally.

Suppose that we start with $k$ requests on vertices 0 and 2 each. Next, we can remove $k/2$ appropriate requests from each of these vertices so that the online algorithm uses $k$ distinct frequencies in total for the remaining requests. Then we

issue $k/2$ requests on vertex 1. The online algorithm will use $3k/2$ frequencies, while the optimum is $k$. This gives us a lower bound of $\frac{3}{2}$.

The first observation, explored in [2], is that if initially the two $k$-tuples of requests use slightly distinct sets of frequencies, then the idea above can be refined to give a better lower bound; this is described below in Procedure FinishOff.

Furthermore, if we start with two $k$-tuples of requests on vertices 0 and 2, we can use the key ingredient from [2] (see Procedure Expand below) to create two $k$-tuples of requests at distance 2 apart that are served by two sets of frequencies that differ more than the initial two sets. To this end, issue $k$ requests on point 5. If many new frequencies are used, we are done. Otherwise we can remove requests from 0 and 2 and then add $k$ requests at 1 so that they use many distinct frequencies from the $k$ requests at 5. Now, remove all requests from 0 and 2. The last trick is to issue $k$ requests at 3, and these must use at least half as many distinct frequencies either from the requests at 1 or from the requests at 5. Remove all requests either from 5 or 1, whichever vertex has more frequencies in common with vertex 3. As it turns out, we end up with two vertices, each having $k$ active request, whose frequency sets differ by more than the two initial sets on vertices 0 and 2.

In [2], the $\frac{14}{9} \approx 1.556$ lower bound is obtained by running first Procedure Expand followed by Procedure FinishOff. We improve the bound by iterating Procedure FinishOff. A somewhat careful argument is needed to show that the overall optimum is still $k$. Optimizing the parameters, we get the lower bound of $\frac{11}{7} \approx 1.571$.

**Theorem 2.** *No deterministic online algorithm for general online frequency assignment on a path with 8 vertices has competitive ratio smaller than* $\frac{11}{7} \approx 1.571$.

*Proof.* Let $R = \frac{11}{7}$ be our target competitive ratio. Let $\rho > 0$ be small and $k$ be a sufficiently large positive integer. We give an adversary strategy which for a given online algorithm $\mathcal{A}$ generates a sequence on which $\mathcal{A}$ uses at least $(R-\rho)k$ frequencies while the optimum is $k$. By taking $\rho$ small and $k$ large, we obtain the desired lower bound.

We first describe the overall adversary strategy and the two procedures FinishOff and Expand. Then we verify that the optimum is $k$.

At the beginning, the adversary simply issues $k$ requests at vertex 0 and $k$ requests at vertex 2. The rest of the adversary strategy is divided into phases. The invariant at the beginning of each phase is that there are two nodes $v$ and $v+2$ with $|L_v| = |L_{v+2}| = k$, and that there are no other active requests. In each phase, the adversary proceeds as follows. Let $s$ and $\delta$ be such that the online algorithm now uses $|L_v \cup L_{v+2}| = s = (1+\delta)k$ frequencies. If $\delta \geq \frac{1}{7} - \rho$, then the adversary completes the sequence by executing Procedure FinishOff. Otherwise, it uses Procedure Expand; this either completes a sequence or ends in a configuration with $k$ requests on each of two points $u$ and $u+2$ for some $u$, in which case we continue with the next phase.

Procedure FinishOff: Let $U$ be the $\lceil k - \frac{1}{2}s \rceil$ lowest frequencies in $L_v \cap L_{v+2}$ and let $U'$ be the $\lceil k - \frac{1}{2}s \rceil$ highest frequencies in $L_v \cap L_{v+2}$. The adversary removes

the requests using frequencies $U$ from $v$ and the requests using frequencies $U'$ from $v+2$. Next, he makes $\lceil k - \frac{1}{2}s \rceil$ requests on $v+1$. These requests will have to be assigned frequencies other than those left at $v$ and $v+2$. The remaining frequencies at $v$ and $v+2$ are distinct, with $\lfloor \frac{1}{2}s \rfloor$ frequencies at each vertex. Thus the current total number of frequencies used is at least $k + \lfloor \frac{1}{2}s \rfloor \geq \frac{1}{2}(3+\delta)k - 1$. Note that if $\delta \geq \frac{1}{7} - \rho$ then the number of frequencies is at least $(\frac{11}{7} - \frac{1}{2}\rho)k - 1 \geq (R - \rho)k$ for a large $k$.

Procedure EXPAND: Since the path has eight vertices, it must contain either vertex $v+5$ or $v-3$. Without loss of generality we suppose that it contains $v+5$; the other case is symmetric.

Issue $k$ requests on $v+5$. If $|L_v \cup L_{v+2} \cup L_{v+5}| \geq Rk$, we stop the input sequence. In the remaining case we have $|L_v \cup L_{v+2} \cup L_{v+5}| \leq Rk$. This implies two things. First, denoting $r = (2 + \delta - R)k$, we get

$$\begin{aligned} |(L_v \cup L_{v+2}) \cap L_{v+5}| &= |(L_v \cup L_{v+2})| + |L_{v+5}| - |L_v \cup L_{v+2} \cup L_{v+5}| \\ &\geq (1+\delta)k + k - Rk \;=\; r. \end{aligned}$$

Second, for each $u \in \{v, v+2\}$, we obtain

$$\begin{aligned} |L_u \cap L_{v+5}| &= |L_u| + |L_{v+5} - |L_u \cup L_{v+5}| \\ &\geq |L_u| + |L_{v+5}| - |L_v \cup L_{v+2} \cup L_{v+5}| \\ &\geq (2-R)k \;\geq\; \tfrac{1}{2}r, \end{aligned}$$

where the last inequality uses the fact that $\delta < \frac{1}{7}$ whenever we use EXPAND. Therefore there are sets $U \subseteq L_v \cap L_{v+5}$ and $U' \subseteq L_{v+2} \cap L_{v+5}$ such that $U \cap U' = \emptyset$ and $|U| = |U'| = \lfloor \frac{1}{2}r \rfloor$. Remove from $v$ all the requests that do not use frequencies in $U$ and from $v+2$ all the requests that do not use frequencies in $U'$. Then issue $k - \lfloor \frac{1}{2}r \rfloor$ requests on $v+1$. These requests will have to be allocated frequencies that are not in $U \cup U'$, while those in $U \cup U' \subseteq L_{v+5}$ are still used at $v+5$. Thus at this point we have $|L_{v+1} - L_{v+5}| \geq \lfloor \frac{1}{2}r \rfloor$. Then delete all the remaining requests from $v$ and $v+2$ and issue $\lfloor \frac{1}{2}r \rfloor$ more requests on $v+1$. As a result, we have $|L_{v+1} \cup L_{v+5}| = k + z$ for $z \geq \lfloor \frac{1}{2}r \rfloor$.

Next, the adversary makes $k$ requests on $v+3$. Then we have $\max\{|L_{v+1} \cup L_{v+3}|, |L_{v+3} \cup L_{v+5}|\} \geq k + \frac{1}{2}z$. Without loss of generality, assume $|L_{v+1} \cup L_{v+3}| \geq k + \frac{1}{2}z$. Finally, the adversary removes all requests from $L_{v+5}$. Note that at this time we have $|L_{v+1}| = |L_{v+3}| = k$, $|L_{v+1} \cup L_{v+3}| \geq k + \frac{1}{2}z \geq (\frac{3}{2} + \frac{1}{4}\delta - \frac{1}{4}R)k - 1$, and that no vertex other than $v+1, v+3$ has any active requests. This completes the description of Procedure EXPAND.

To finish the description of the adversary strategy, we need to show that it finishes the sequence after finitely many phases. Suppose that we have a phase that is not a final one. Thus the phase starts with $\delta = \frac{1}{7} - \rho - \epsilon$ for some $\epsilon > 0$ and uses EXPAND. At the end, the number of used frequencies is at least

$$\begin{aligned} (\tfrac{3}{2} + \tfrac{1}{4}(\tfrac{1}{7} - \rho - \epsilon) - \tfrac{1}{4})k - 1 &= (1 + \tfrac{1}{7} - \tfrac{1}{4}(\rho + \epsilon))k - 1 \\ &\geq (1 + \delta')k, \end{aligned}$$

for $\delta' = \delta + \frac{1}{2}\rho$ and sufficiently large $k$ (independent of $\epsilon$). Thus, for any $\rho > 0$ and a sufficiently large $k$, after a fixed number of phases, $\delta$ increases above $\frac{1}{7} - \rho$, at which point the adversary uses FinishOff and completes the sequence.

The description of the strategy shows that at the our the online algorithm $\mathcal{A}$ uses at least $(R - \rho)k$ frequencies.

To finish the proof, it remains to show that the optimum number of frequencies is $k$. Given the instance produced by the adversary strategy, we will maintain an offline solution (that is, a dynamic frequency assignment). In addition to using only given $k$ frequencies, this offline frequency assignment will maintain the following invariant:

($*$) Let $U$ and $U'$ be the sets of frequencies from the description of the procedure FinishOff or Expand in the current phase. Consider the corresponding sets of requests, i.e., the requests at $v$ using frequencies $U$ in the online algorithm and the requests at $v+2$ using frequencies $U'$. Then these two sets of requests use the same set of frequencies.

At the beginning of the first phase, after the first $2k$ requests, we can guarantee the invariant while using only $k$ frequencies total, since we can assign the frequencies arbitrarily. Next, we check that for both FinishOff and Expand we can serve the sequence with $k$ frequencies and maintain the invariant.

In FinishOff, after removing the requests corresponding to $U$ and $U'$, the same frequencies are used at $v$ and $v + 2$, by invariant ($*$). So we have enough admissible frequencies for the new requests at $v + 1$ (among the $k$ original frequencies). This is the last phase, so there is no invariant to be maintained.

In Expand, first we assign the requests at $v + 5$ arbitrarily, but using the same $k$ frequencies as at $v$ and $v + 2$. If the sequence does not stop now, then, by invariant ($*$), after removing the requests not corresponding to $U$ and $U'$, the same frequencies are used at $v$ and $v + 2$. So we have enough admissible frequencies for the first batch of requests at $v+1$. Next we remove the remaining requests at $v$ and $v + 2$, thus we have admissible frequencies for the remaining requests at $v + 1$. Finally, for the $k$ requests at $v + 3$, we may assign the $k$ frequencies arbitrarily. In particular, we can guarantee the invariant for the next phase.

Thus the optimum uses only $k$ frequencies and the competitive ratio is at least $R = \frac{11}{7}$, completing the proof of the lower bound.

We remark that the value $R = \frac{11}{7}$, as well as the choice of the breakpoint $\delta = \frac{1}{7}$, is optimal for the strategy described in the proof.

## 5 $\mathbb{NP}$-Completeness

We have seen that in the incremental version, computing the optimum is easy. Now we show that this is not the case once we allow dynamic requests.

Let FAL stand for the decision version of the frequency allocation problem for the path: "Given a sequence of requests and an integer $k$, determine whether these requests can be served with at most $k$ frequencies".

**Theorem 3.** *The problem* FAL *is* $\mathbb{NP}$*-complete, even for any fixed* $k \geq 3$.

*Proof.* We reduce the 3-coloring problem for planar graphs (3CPG) to FAL. Suppose that $G$ is a planar graph. We construct a sequence $I$ of requests such that $I$ can be served with 3 frequencies if and only if $G$ has a 3-coloring.

Using the result of [3], we can embed $G$ in a plane so that (i) each vertex is represented by a vertical line segment, and (ii) each edge $(u, v)$ is represented by a horizontal line segment whose endpoints connect the vertical segments representing $u$ and $v$, without intersecting any other vertical segments. (See Figure 1.) This embedding is also known as visibility representations of planar graphs (see, for example, [6,7]). We align all segments so that coordinates of all their endpoints are integral and multiples of 6.

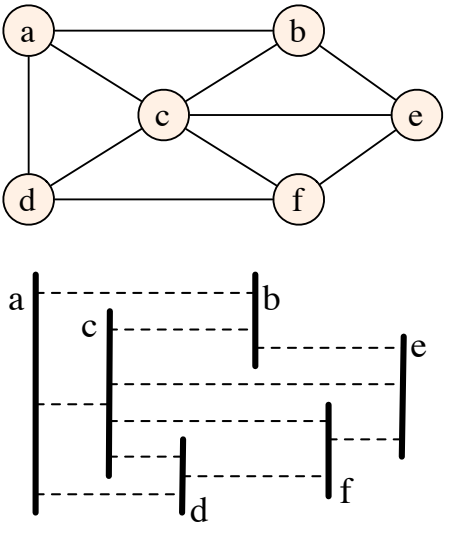



**Fig. 1.** A graph and its embedding

Now we construct an instance $I$ of FAL. The x-axis of the plane will correspond to the path and the y-axis will represent time. It is convenient to think of requests in $I$ as vertical intervals in the plane. We will denote each such interval by $(x, y', y'')$, where $x'$ is its x-coordinate and $y', y''$ the bottom and top y-coordinates. As a request, it is a request at point $x$ arriving at time $y'$ and departing at time $y''$.

For each vertex in $G$ represented by a vertical segment $(x, y', y'')$, the instance contains the request $(x, y', y'')$. For each edge represented by a segment from $(x', y)$ to $(x'', y)$, $x'' > x'$, we build a gadget consisting of a number of requests. Note that $x'' - x'$ is even and at least 6, by our assumptions. We add two requests $(x'+1, y+4, y+5)$, one request $(x'+2, y+2, y+5)$, and two requests

$(x' + 2, y, y + 3)$. Furthermore we will add requests $(x' + i, y, y + 1)$ for any $i$, $2 < i < x'' - x'$; we add one such request if $i$ is odd and two such requests if $i$ is even. By the construction and alignment of the segments, the requests from different edge gadgets do not interfere. (See Figure 2.)

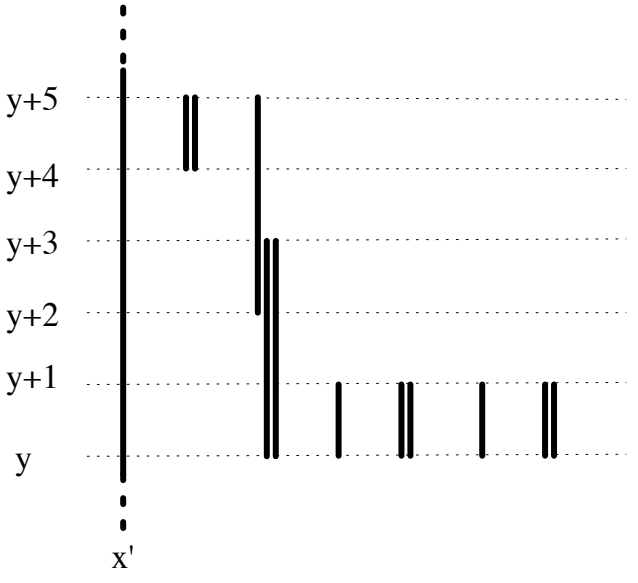



**Fig. 2.** An edge gadget

The edge gadget guarantees that the assignment of frequencies to (the requests corresponding to) its endpoints can be extended to an assignment of frequencies to the request in the edge gadget using only three frequencies in total if and only if the two vertex colors are distinct. Consider again an edge represented by a segment from $(x', y)$ to $(x'', y)$, $x'' > x'$. Suppose that the request containing point $(x', y)$ (i.e., the first endpoint of an edge) is assigned frequency $f$. We show that if only three frequencies $f, f', f''$ are used, the assignment is essentially unique. The two requests $(x'+1, y+4, y+5)$ must use $f'$ and $f''$. Now the request $(x' + 2, y + 3, y + 5)$ must use $f$, and the two requests $(x' + 2, y, y + 3)$ must use $f'$ and $f''$. We continue by induction to show that for $i$ odd, $(x' + i, y, y + 1)$ uses $f$ and, for $i$ even, the two requests $(x' + i, y, y + 1)$ use $f'$ and $f''$. This implies that $(x'' - 1, y, y + 1)$ uses $f$. Overall, the assignment is valid if and only if the request containing $(x'', y)$ is not colored by $f$.

This implies that $G$ is 3-colorable if and only if the instance $I$ of FAL has a frequency allocation with only 3 frequencies. The extension to $k$-colorability for $k \geq 3$ is straightforward: In the gadget, instead of each pair of identical request, use $k - 1$ of the same requests.

## 6 Final Comments

The most outstanding open problem is to establish the optimal asymptotic competitive ratio for the dynamic case (with expirations) on the path. The current gap is between 11/7 and 5/3.

We would like to point out that the idea of Algorithm FourBuckets can be generalized as follows. Suppose that $G$ has $k$ induced subgraphs $G_1, \ldots, G_k$ with the following properties: (1) Each vertex of $G$ belongs to exactly $\ell$ subgraphs $G_i$, (2) Each subgraph $G_i$ does not contain a 4-path (that is, each $G_i$ is a collection of disjoint stars. Then $G$ has a $k/\ell$-competitive algorithm. In fact, this can be generalized further: if, instead of (2), we require that all $G_i$ have an $R$-competitive algorithm, then $G$ will have an $kR/\ell$-competitive algorithm. As of now, however, we have not been able to apply it to improve upper bounds for other types of graphs.

## Acknowledgments

We are grateful to Dan Král and Jan Kratochvíl for discussions on graph drawing.

## References

1. Chan, J.W.-T., Chin, F.Y.L., Ye, D., Zhang, Y.: Online frequency allocation in cellular networks. In: Proc. 19th Symp. on Parallel Algorithms and Architectures (SPAA), pp. 241–249 (2007)
2. Chan, J.W.-T., Chin, F.Y.L., Ye, D., Zhang, Y., Zhu, H.: Frequency allocation problem for linear cellular networks. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 61–70. Springer, Heidelberg (2006)
3. Duchet, P., Hamidoune, Y.O., Las Vergnas, M., Meyniel, H.: Representing a planar graph by vertical lines joining. Discrete Mathematics 46, 319–321 (1983)
4. McDiarmid, C., Reed, B.: Channel assignment and weighted colouring. Networks 36, 114–117 (2000)
5. Narayanan, L., Shende, S.M.: Static frequency assignment in cellular networks. Algorithmica 29(3), 396–409 (2001)
6. Rosenstiehl, P., Tarjan, R.: Rectilinear planar layouts and bipolar orientations of planar graphs. Discrete & Computational Geometry 1, 343–344 (1986)
7. Tamassia, R., Tollis, I.G.: A unified approach a visibility representation of planar graphs. Discrete & Computational Geometry 1, 321–341 (1986)

# Link Distance and Shortest Path Problems in the Plane*

Atlas F. Cook IV and Carola Wenk

University of Texas at San Antonio
Department of Computer Science
One UTSA Circle, San Antonio, TX 78249-0667
acook@cs.utsa.edu, carola@cs.utsa.edu

**Abstract.** We develop algorithms to compute Voronoi diagrams, shortest path maps, and the Fréchet distance in the plane with polygonal obstacles. Distances between points are measured either by link distance or by Euclidean shortest path distance.

**Keywords:** Voronoi Diagram, Shortest Path Map, Fréchet Distance, Link Distance.

## 1 Introduction

We develop algorithms to compute Voronoi diagrams, shortest path maps, and the Fréchet distance in the plane with polygonal obstacles. Our twist on these problems is to measure the distance between two points either by link distance or by Euclidean shortest path distance. The *link distance* [3,9,14,16,17] between two points is the minimum number of edges necessary to connect the points with a polygonal path that avoids a set of obstacles. Our motivation for studying link distance and shortest path problems is that they often serve as building blocks for other techniques.

### 1.1 Related Work

Related work uses various preprocessing schemes to support $O(\log k)$ time link distance queries between two points. In a simple polygon with $k$ vertices, fixed source queries are supported by Suri [17] after $\Theta(k)$ preprocessing, and queries between any two points are supported by Arkin, Mitchell, and Suri [3] and Efrat et al. [9] after $\Theta(k^3)$ preprocessing. In a polygonal domain[1] with $k$ vertices, Mitchell, Rote, and Woeginger [16] support queries from a fixed source after $\Theta(k^4)$ preprocessing.

Related work on Euclidean shortest path problems also supports $O(\log k)$ query times. In a simple polygon, Guibas et al. [11] support shortest path queries between any two points after $\Theta(k)$ preprocessing. In a polygonal domain, Hershberger and Suri [13] support queries from a fixed source after $\Theta(k \log k)$ preprocessing. Chiang and Mitchell [4] support queries between any two points in a polygonal domain after $O(k^{11})$ preprocessing.

* This work has been supported by the National Science Foundation grant NSF CAREER CCF-0643597. Previous versions of this work have appeared as technical reports [6,7].

[1] A *polygonal domain* is a plane with polygonal obstacles.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 140–151, 2009.

### 1.2 Terminology

Throughout this paper, $N$ is the total complexity of objects in the plane such as polygonal curves, points, or line segments that are not considered to be obstacles. By contrast, $\mathcal{O} = \{o_1, o_2, ..., o_k\}$ denotes $k$ obstacle vertices in the plane. $\pi(s,t)$ denotes a shortest path between points $s$ and $t$, and $d(s,t)$ is the Euclidean length of $\pi(s,t)$. $\pi_L(s,t)$ and $d_L(s,t)$ denote similar concepts for link distance. $\pi(s,o_i) \circ t$ denotes the shortest path from $s$ to $o_i$ concatenated with the line segment from $o_i$ to $t$. $\pi(s,t) = s \circ t$ implies that $s$ has line of sight to $t$. $\lambda_s(k)$ is a near-linear function that is defined by the length of a Davenport-Schinzel sequence [1]. $\overline{ab}$ is a line segment with endpoints $a$ and $b$.

The *Fréchet distance* [2] is a similarity metric for *continuous* shapes that is defined for two $N$ complexity polygonal curves $A, B : [0,1] \rightarrow \mathbb{R}^2$ as

$$\delta_F(A,B) = \inf_{\alpha,\beta:[0,1]\rightarrow[0,1]} \sup_{t\in[0,1]} d(\ A(\alpha(t)), B(\beta(t))\ )$$

where $\alpha$ and $\beta$ range over continuous non-decreasing reparameterizations, and $d$ is a distance metric for points. A variant called the *weak Fréchet distance* permits all continuous reparameterizations. For a given constant $\varepsilon \geq 0$, *free space* is $\{(s,t) \mid s \in A,\ t \in B,\ d(s,t) \leq \varepsilon\}$. A *free space cell* is the parameter space defined by two line segments $\overline{ab} \in A$ and $\overline{cd} \in B$, and the free space inside the cell is all points $\{(s,t) \mid s \in \overline{ab}, t \in \overline{cd}, d(s,t) \leq \varepsilon\}$.

### 1.3 Our Results

Link distance is fundamentally different from Euclidean distance and has a wealth of applications including robotic motion, wireless communications, geographic information systems, VLSI, computer vision, solid modeling, image processing, and even water pipe placement. These applications are naturally modeled by link distance because turns are costly while straight line movements are inexpensive.

Table 1 summarizes our results. In section 2.1, we present *tight* bounds and efficient algorithms to compute two types of link-based Voronoi diagrams in the plane. Surprisingly, this seems to be the first time that link-based Voronoi diagrams have been studied. In section 2.2, we define a link-based shortest path map that allows the query to be any point on a line segment. Although there are related Euclidean results [4,11] that support queries from more than one point in the plane, we are not aware of any related work that supports *link distance* queries from a continuous set of source points. In sections 2.3 and 2.4, we compute the link-based Fréchet distance in a simple polygon and a polygonal domain. A novelty of our approach is that it allows the link-based Fréchet distance to be computed in the same time and space as the decision problem. This is intriguing because the only other known scenarios where it is possible to shave off the logarithmic Fréchet optimization factor are for the weak or discrete Fréchet distance. In these cases, the free space diagram can be treated as a planar graph [2], and any linear-time shortest path algorithm (e.g., [12]) can be applied to this planar graph to solve the optimization problem. We also give a lower bound for the complexity of the link-based free space diagram in a polygonal domain.

We now describe our Euclidean shortest path results. Sections 3.1 and 3.2 define structures that encode all shortest paths between two line segments in a polygonal domain. Section 3.3 explores the Fréchet distance in a polygonal domain and gives a lower

**Table 1. Our results.** The shortest path map SPM($\overline{ab}$,$\mathbb{R}^2$) supports queries from $s \in \overline{ab}$ to $t \in \mathbb{R}^2$; SPM($\overline{ab}$,$\overline{cd}$) supports queries from $s \in \overline{ab}$ to $t \in \overline{cd}$. $P$ is a simple polygon with $k$ vertices; $D$ is a polygonal domain with $k$ vertices. $N$ is the complexity of any objects that are not obstacles. An asterisk * indicates that in addition to the exact runtimes that are shown, we also have approximation algorithms.

| **Link Distance Results** | | Time | Space |
|---|---|---|---|
| Site-Based Voronoi Diagram | $P$ | $O(N^2k \log Nk)$ | $\Theta(N^2k)$ |
| Distance-Based Voronoi Diagram | $P$ | $O(N(N+k) \log N \log k)$ | $\Theta(N(N+k))$ |
| SPM($\overline{ab}$,$\mathbb{R}^2$) | $P$* | $O(k^2)$ | $O(k^2)$ |
| SPM($\overline{ab}$,$\overline{cd}$) | $D$* | $O(k^6\lambda_6(k))$ | $O(k^7)$ |
| Fréchet Distance | $P$* | $O(kN+N^2)$ | $O(k+N^2)$ |
| | $D$* | $O(N^2k^7 \log kN)$, $\Omega(N^2k^4)$ | $O(Nk^2+k^4)$ |
| **Euclidean Results** | | | |
| Fréchet Distance | $P$ [5] | $O(k+N^2 \log Nk \log N)$ | $O(N^2+k)$ |
| | $D$ | $O(N^2k^4 \log Nk \log k)$, $\Omega(N^2k^2)$ | $O(Nk+k^4)$ |
| Weak Fréchet Distance | $D$ | $O(N^2k^3\lambda_6(k) \log Nk)$, $\Omega(N^2k^2)$ | $O(Nk+k^4)$ |
| SPM($\overline{ab}$,$\overline{cd}$) | $D$ | $O(k^4\lambda_6(k))$, $\Omega(k^2)$ | $O(k^5)$ |

bound for the free space diagram. Our motivation for studying this problem is that teaming up two people for safety reasons is common practice in many real-life situations, ranging from scouts in summer camp, to fire fighters and police officers, and even to astronauts exploring the moon. In all of these applications, two team members need to coordinate their movement to stay within "walking distance" so that fast assistance can be offered in case of an emergency. The Fréchet distance is an ideal model for this scenario. Section 3.4 contains a lower bound and an algorithm for a shortest path map that supports all possible queries between a pair of line segments in a polygonal domain.

## 2 Link Distance Problems

### 2.1 Voronoi Diagrams

Let $P$ be a simple polygon with vertices $o_1, ..., o_k$. Let $\mathcal{S} = s_1, ..., s_N$ be a set of point or line segment sites in $P$. Unlike traditional Euclidean Voronoi diagrams, each region in a link distance Voronoi diagram can be associated with multiple nearest sites. We define a *site-based* Voronoi diagram $\mathcal{V}_s(\mathcal{S})$ as a partition of $P$ such that all points within a region have the same *set* of nearest neighbor sites with respect to link distance. We define a *distance-based* Voronoi diagram $\mathcal{V}_d(\mathcal{S})$ as a partition of $P$ such that all points within a region have the same link distance to a nearest site. A distance-based Voronoi diagram is composed of interior-disjoint layers for $i = 0...k$ such that the $i$th layer is $\mathfrak{L}_i = \{t \in P \mid i = \min_{s \in \mathcal{S}} d_L(s,t)\}$. Note that all points in a layer need not share the same set of nearest neighbor sites. Related work [10] has considered the complexity of the arrangement of visibility polygons for a set of sites but did not explore

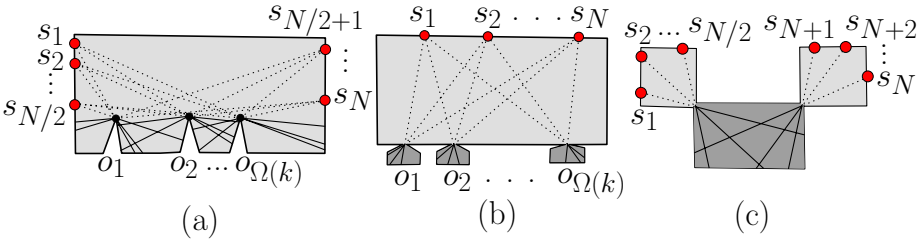



**Fig. 1.** Edges in $\xi$ are shown as solid lines. (a) $\mathcal{V}_s(\mathcal{S})$ has $\Omega(N^2k)$ complexity. $\mathcal{V}_d(\mathcal{S})$ can have (b) $\Omega(Nk)$ or (c) $\Omega(N^2)$ edges defining layer $\mathfrak{L}_1$. For both (b) and (c), layer $\mathfrak{L}_1$ consists of the lightly shaded area plus the solid visibility polygon edges, and layer $\mathfrak{L}_2$ consists of the remaining (disconnected) heavily shaded areas in $P$.

this arrangement for all link distances from 0 to $k$. Candidate edges for $\mathcal{V}_s(\mathcal{S})$ and $\mathcal{V}_d(\mathcal{S})$ can be defined by precomputing SPM($s_j$) [17] for each site $s_j \in \mathcal{S}$ in $O(Nk)$ total time and space. Let $\xi$ be the set of $O(Nk)$ edges defined by SPM($s_1$),...,SPM($s_N$).

**Theorem 1.** *$\mathcal{V}_s(\mathcal{S})$ has $\Theta(N^2k)$ complexity and can be constructed in $O(N^2k)$ expected time and $O(N^2k \log Nk)$ deterministic time. Link distance and path queries are supported in $O(\log Nk + K)$ time, where $K$ is the complexity of any returned path.*

*Proof.* Consider the arrangement of all edges in $\xi$. $\mathcal{V}_s(\mathcal{S})$ can have $\Omega(N^2k)$ complexity because $\Omega(k)$ pairs of adjacent vertices $o_i,\ o_{i+1} \in P$ can each define an $\Omega(N^2)$ arrangement such that $\Omega(N)$ edges pass through $o_i$ and $\Omega(N)$ edges pass through $o_{i+1}$ (see Figure 1a). $\mathcal{V}_s(\mathcal{S})$ has $O(N^2k)$ complexity because each of the $O(Nk)$ edges in $\xi$ can intersect at most three faces in each of SPM($s_1$),...,SPM($s_N$) [3]. The arrangement for $\xi$ can be constructed with an incremental algorithm [1] in $O(N^2k)$ expected time or in $O(N^2k \log Nk)$ deterministic time. Since some edges in this arrangement should not appear in $\mathcal{V}_s(\mathcal{S})$, a breadth-first postprocessing step should merge adjacent faces with link distance $i$ that are separated by a suboptimal edge with link distance $j > i$. □

Compared to $\mathcal{V}_s(\mathcal{S})$, the *distance-based* Voronoi diagram $\mathcal{V}_d(\mathcal{S})$ has asymptotically superior complexity bounds because the points in a face of $\mathcal{V}_d(\mathcal{S})$ need not have the same set of nearest neighbor sites.

**Theorem 2.** *$\mathcal{V}_d(\mathcal{S})$ has $\Theta(N(N+k))$ complexity (for $k \geq 2$) and can be constructed in $O(N(N+k) \log N \log k)$ time. Link distance and path queries are supported in $O(\log Nk + K)$ time, where $K$ is the complexity of any returned path.*

*Proof.* $\mathcal{V}_d(\mathcal{S})$ has $\Omega(Nk)$ complexity because each of the $N$ sites can define $\Omega(k)$ thin line of sight wedges that contribute to a layer (see Figure 1b). $\mathcal{V}_d(\mathcal{S})$ has $\Omega(N^2)$ complexity because $\Omega(N)$ thin line of sight wedges defined by $s_1, \ldots, s_{N/2}$ can each intersect $\Omega(N)$ thin wedges defined by $s_{N/2+1}, \ldots, s_N$ (see Figure 1c). We now show that $\mathcal{V}_d(\mathcal{S})$ has $O(N(N+k))$ complexity. Let $P_d^i$ be the set of points at link distance $d$ from site $s_i$ so that layer $\mathfrak{L}_d$ equals $\bigcup_{i=1}^{N} P_d^i$. $P_d^i$ cannot *enter* $P_d^j$ in more than one connected interval without creating a hole in a simple polygon; thus, for each pair of sites, there are $O(1)$ entrance vertices. If $P_d^i$ *exits* $P_d^j$ in more than one interval, then

a vertex must separate these intervals. Thus, the number of exit vertices defined by each site $s_i$ is $O(N+k)$. A divide-and-conquer approach can construct all layers in $O(N(N+k)\log N \log kN)$ total time and $O(N(N+k))$ space. □

### 2.2 Shortest Path Maps

We develop link-based shortest path maps that support $O(\log k)$ time queries from any source point $s \in \overline{ab}$. Approximations accurate to within 1, 2, and 3 links are also explored. Previous link-based work in a simple polygon with $k$ vertices builds a shortest path map SPM($s$) to support all queries from a fixed source $s$ after $\Theta(k)$ preprocessing [17] and builds another shortest path map SPM($\mathbb{R}^2, \mathbb{R}^2$) to support queries between any two points after $\Theta(k^3)$ preprocessing [3,9]. Our SPM($\overline{ab}$,$\mathbb{R}^2$) structure supports queries from any point $s \in \overline{ab}$ to any point $t \in \mathbb{R}^2$ after $O(k^2)$ preprocessing. Related work in a link-based polygonal domain with $k$ vertices uses $\Theta(k^4)$ preprocessing to build SPM($s$) [16]. Our SPM($\overline{ab}$,$\overline{cd}$) structure supports all link-based queries from any $s \in \overline{ab}$ to $t \in \overline{cd}$.

We use the following terminology of [3]: the *combinatorial structure* of a shortest path map is a listing of the combinatorial structures of its edges. The combinatorial structure of a shortest path map edge $\mathcal{E}$ is a vertex-edge pair $(v, e)$ such that $\mathcal{E}$ has one endpoint on an obstacle vertex $v$ and has its other endpoint on an obstacle edge $e$. As the source point $s$ varies along $\overline{ab}$, the position of $\mathcal{E}$'s endpoint on $e$ is parameterized homographically by $g(s) = \frac{A+Bs}{C+Ds}$ for constants $A, B, C, D$ (see Figure 2). We define an *edgelet* $\alpha$ as a maximal line segment such that the shortest path map for every source point $s \in \alpha$ has the same combinatorial structure.

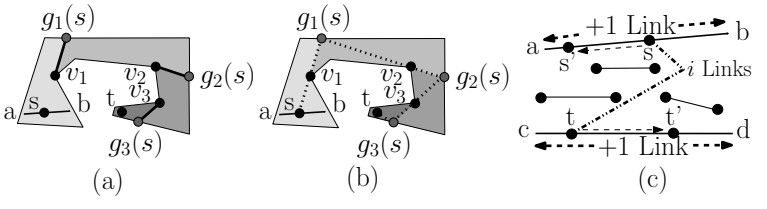



**Fig. 2.** (a) Every shortest path map edge can be described as a vertex-edge pair. (b) All but the last link of $\pi_L(s,t)$ can be made to overlap a shortest path map edge. (c) All $d_L(s,t)$ for $s \in \overline{ab}$, $t \in \overline{cd}$ equal either $i$, $i+1$, or $i+2$, where $i = \min_{s \in \overline{ab},\ t \in \overline{cd}} d_L(s,t)$.

**Theorem 3.** *A link-based shortest path map SPM($\overline{ab}$,$\mathbb{R}^2$) can be constructed in a simple polygon $P$ with $k$ vertices in $O(k^2)$ time and space. SPM($\overline{ab}$,$\mathbb{R}^2$) allows $d_L(s,t)$, $\pi_L(s,t)$ to be returned for any $s \in \overline{ab}$ and $t \in \mathbb{R}^2$ in $O(\log k + K)$ time, where $K$ is the complexity of any returned path. These queries can also be answered to within* one *link of optimal after $O(k)$ preprocessing.*

*Proof.* Partition $\overline{ab}$ into $O(k)$ edgelets by computing $\overline{ab} \cap \text{SPM}(o_j)$ [17] for each obstacle vertex $o_j \in P$. Each intersection induces at most three edgelets on $\overline{ab}$ by [3], and we construct a shortest path map SPM($\alpha_i$) [17] for each edgelet. A query $d_L(s,t)$

consists of a one dimensional coordinate for $s$ and a two dimensional coordinate for $t \in P$. $s \in \overline{ab}$ lies in an edgelet $\alpha_i$, and the edges of SPM($\alpha_i$) are parameterized by $s \in \alpha_i$. $t$ is a point inside SPM($\alpha_i$). To speed up queries, we compute a non-parameterized triangulation of SPM($\alpha_i$) that permits evaluating at most two parameterized edges of SPM($\alpha_i$) at query time.

SPM($\alpha_i$) contains $O(k)$ parameterized edges that are represented as vertex-edge pairs $(v, e)$. By [3], there are at most two parameterized shortest path map edges $(v_1, e)$, $(v_2, e) \in$ SPM($\alpha_i$) that can intersect the interior of a fixed boundary edge $e \in P$. As $s$ varies over $\alpha_i$, the edge $(v_1, e)$ touches a subsegment $\overline{\sigma\varsigma} \in e$ such that the three points $v_1$, $\sigma$, and $\varsigma$ define a triangle $\Delta_1$. Similarly, the parameterization for $(v_2, e)$ defines a second triangle $\Delta_2$. Triangulating $\Delta_1 \cup \Delta_2$ yields a constant number of triangles that can be associated with $(v_1, e)$,$(v_2, e)$. No other parameterized shortest path map edges can intersect these triangles for any $s \in \alpha_i$ because shortest path map edges in a simple polygon never cross each other.

A query $d_L(s, t)$ is handled in $O(\log k)$ time by identifying the edgelet $\alpha_i$ containing $s$, point locating the triangle containing $t$ in SPM($\alpha_i$), and evaluating the at most two parameterized edges associated with this triangle. $\pi_L(s, t)$ is calculated by following a chain of predecessors. *Approximate* queries use a shortest path map SPM($\overline{ab}$) [17] to return $\min_{s' \in \overline{ab}} d_L(s', t)$. This value always equals either $d_L(s, t)$ or $d_L(s, t) + 1$ (see Figure 2c). The approximate path is $s \circ \pi_L(s', t)$. □

**Lemma 1.** *In a polygonal domain, the intersection of a line segment $\overline{cd}$ with a link-based shortest path map SPM(s) of $\Theta(k^4)$ complexity [16] can be constructed without precomputing SPM(s) in $O(k^2\alpha(k)\log^2 k)$ time and $O(k^2)$ space.*

*Proof.* Mitchell, Rote, and Woeginger [16] represent all points with link distance $j$ to a source by the union of $O(k^2)$ triangles. Since at most three distances define SPM($s$) $\cap\, \overline{cd}$ (see Figure 2c), SPM($s$) $\cap\, \overline{cd}$ is defined by the union of three sets of $O(k^2)$ triangles. Both the triangles and their intersection with $\overline{cd}$ can be computed in $O(k^2\alpha(k)\log^2 k)$ time and $O(k^2)$ space using the algorithm of [16]. □

Many problems in a polygonal domain with $k$ vertices are difficult to compute. For example, Chiang and Mitchell [4] support Euclidean shortest path queries between any two points in a polygonal domain after $O(k^{11})$ preprocessing, and Mitchell, Rote, and Woeginger [16] construct a link-based shortest path map from a fixed source in $\Theta(k^4)$ time and space. Theorem 4 presents the first shortest path map in a polygonal domain that supports link distance queries from a *continuous* set of source points.

**Theorem 4.** *A link-based shortest path map SPM($\overline{ab}$,$\overline{cd}$) can be constructed in a polygonal domain in $O(k^7)$ space and either $O(k^7)$ expected time or $O(k^6\lambda_6(k))$ deterministic time. $d_L(s, t)$, $\pi_L(s, t)$ can be returned for any $s \in \overline{ab}$, $t \in \overline{cd}$ in $O(\log k + K)$ time, where $K$ is the complexity of $\pi_L(s, t)$.* Approximate *queries are supported for $s \in \overline{ab}$, $t \in \mathbb{R}^2$ to within* one *link of optimal after $O(k^4)$ preprocessing and to within* two *links of optimal after $O(k^{\frac{7}{3}}\log^{3.11} k)$ time and $O(k)$ space preprocessing.*

*Proof.* SPM($\overline{ab}$,$\overline{cd}$) is a partition of the parameter space defined by $\overline{ab}$ and $\overline{cd}$. $\overline{ab}$ can be partitioned into $O(k^3)$ edgelets by Lemma 1. For each edgelet $\alpha_i$, a shortest path map

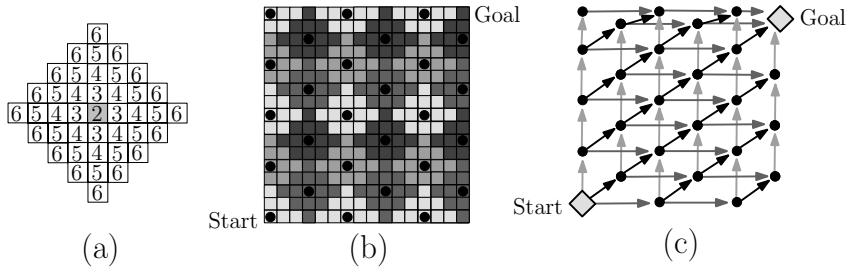



**Fig. 3.** (a) A bundle of cells. Each cell is marked with its maximum error from a representative distance in the shaded node. (b) $O(\frac{N^2}{\mathcal{B}^2})$ bundles cover the free space diagram and define (c) a directed acyclic graph.

edge $\mathcal{E} \in \text{SPM}(\alpha_i)$ is a vertex-edge pair $(v, e)$ that is homographically parameterized by $s \in \alpha_i$ such that $\mathcal{E} \cap \overline{cd}$ defines a constant complexity algebraic curve. Constructing such a curve for each choice of $v$ and $e$ yields $O(k^2)$ curves and $O(k^4)$ arrangement complexity. Since there are $O(k^3)$ edgelets, SPM($\overline{ab}$,$\overline{cd}$) has $O(k^3 \cdot k^4)$ complexity. Approximate queries use SPM($\overline{ab}$) [16] to return $\min_{s' \in \overline{ab}} d_L(s', t)$ exactly or approximately. □

### 2.3 Fréchet Distance in a Simple Polygon

**Lemma 2.** *Although the free space in a link-based cell $\mathcal{C}$ for the Fréchet distance in a simple polygon need not be convex, it is $x$-monotone, $y$-monotone, and connected.*

*Proof Sketch.* Monotonicity follows because the link distance from a point $s$ to each $t \in \overline{cd}$ defines a piecewise constant function that is decreasing-increasing bitonic. Connectivity follows by a point-sliding argument. □

**Theorem 5.** *The link-based Fréchet distance in a simple polygon can be computed* exactly *in $O(kN + N^2)$ time and $O(k + N^2)$ space. It can also be computed to within $\pm\mathcal{B}$ links of optimal in $O(\frac{kN}{\mathcal{B}} + \frac{N^2}{\mathcal{B}^2})$ time and $O(k + \frac{N^2}{\mathcal{B}^2})$ space (for $\mathcal{B} \geq 2$).*

*Proof.* To approximate the Fréchet distance, we compute diamond-shaped[2] *bundles* of $O(\mathcal{B}^2)$ adjacent free space cells (see Figure 3). After defining a representative distance for each bundle with a shortest path map of [17], a directed acyclic graph with $O(\frac{N^2}{\mathcal{B}^2})$ nodes and edges is used to represent all monotone paths through the bundles (see Figure 3c). A breadth first search on this graph yields an approximation for the Fréchet distance to within $\pm\mathcal{B}$ links of optimal. The *exact* Fréchet distance can be computed by applying the approximation algorithm with $\mathcal{B} = 2$ and running the *exact* decision problem twice. To solve the exact decision problem, we use shortest path maps [17] to compute free space on cell boundaries. Since the free space in each cell is $x$-monotone, $y$-monotone, and connected by Lemma 2, reachability information can be propagated via dynamic programming [2,5]. □

[2] A rectangular bundle of cells can also be used but yields a slightly poorer approximation.

The bundles technique can also be used to approximate the traditional Euclidean Fréchet distance (without obstacles) between polygonal curves $A$ and $B$ in $\mathbb{R}^d$. If $l_{\max}$ is the length of the longest line segment in $A \cup B$, then the result is accurate to within $\pm\mathcal{B}\cdot l_{\max}$ of the optimal distance after $O(\frac{N^2}{\mathcal{B}^2})$ processing.

### 2.4 Fréchet Distance in a Polygonal Domain

The Fréchet distance is more difficult to compute in a polygonal domain than in a simple polygon because the free space inside a cell can be disconnected. We use the shortest path map SPM($\overline{ab}, \overline{cd}$) from Theorem 4 to compute the Fréchet distance.

**Theorem 6.** *The link-based Fréchet distance $\delta_F(A, B)$ between polygonal curves $A$ and $B$ in a polygonal domain can be calculated in $O(N^2k^7 \log kN)$ time and $O(Nk^2 + k^4)$ space, and the free space diagram can have $\Omega(N^2k^4)$ complexity. Approximations are available to within 1, 2, and 3 links of $\delta_F(A, B)$ in $O(N^2k^2\ \alpha(k) \log^2 k)$ time and $O(k^2 + N^2)$ space, $O(Nk^{\frac{7}{3}} \log^{3.11} k + N^2k)$ time and $O(k + N^2)$ space, and $O(Nk^{\frac{7}{3}} \log^{3.11} k + N^2 \log k)$ time and $O(k + N^2)$ space.*

*Proof Sketch.* Let $i = \min_{s\in\overline{ab},\ t\in\overline{cd}} d_L(s, t)$. Since all link distances in a cell are either $i$, $i+1$, or $i+2$, even picking a random distance in a cell yields an accurate approximation for the cell. After approximating $\delta_F(A, B)$, its optimal value can be computed via $O(1)$ iterations of the exact decision problem. The exact decision problem is computed by representing each of the $O(N^2)$ free space cells by the SPM($\overline{ab}$,$\overline{cd}$) structure of Theorem 4 and combining dynamic programming [2] with a plane sweep to propagate reachability information in $O(N^2k^7 \log kN)$ time. $O(Nk^2 + k^4)$ space is sufficient to store one edgelet of a cell at a time plus the cell boundaries on the two rows required for dynamic programming [2]. See [7] for our lower bound on the complexity of the free space diagram. □

## 3 Euclidean Shortest Path Problems in a Polygonal Domain

Sections 3.1 and 3.2 introduce shortest path structures in a polygonal domain. These structures are used to compute the Fréchet distance in section 3.3 and a shortest path map in section 3.4.

### 3.1 Dynamic Spotlights

Line of sight shortest paths from $s \in \overline{ab}$ to $t \in \overline{cd}$ are represented by a structure that we call a *dynamic spotlight* (see Figure 4). Suppose that $\Delta_{s,o_i,o_j}$ is a triangle with apex $s \in \overline{ab}$, sides supported by $\overline{so_i}$ and $\overline{so_j}$, and base on $\overline{cd}$. Let $\mathcal{I}_{ab} \in \overline{ab}$ be a maximal connected interval such that $\Delta_{s,o_i,o_j}$ contains no obstacles in its interior for any $s \in \mathcal{I}_{ab}$. A dynamic spotlight is defined as $\mathcal{L}_{\mathcal{D}}(\mathcal{I}_{ab}, o_i, o_j) = \{(s, t) \mid \pi(s, t) = s \circ t,\ s \in \mathcal{I}_{ab},\ t \in \Delta_{s,o_i,o_j} \cap \overline{cd}\}$. As $s$ varies over $\mathcal{I}_{ab}$, the maximal interval $\Delta_{s,o_i,o_j} \cap \overline{cd}$ that is directly visible from $s$ changes continuously and defines $\mathcal{L}_{\mathcal{D}}$. The *free space* for $\mathcal{L}_{\mathcal{D}}$ is $\{(s, t) \in \mathcal{L}_{\mathcal{D}} \mid ||s - t|| \leq \varepsilon\}$. Both $\mathcal{L}_{\mathcal{D}}$ and its free space are contained in a cell $C$ whose parameter space is defined by $\overline{ab}$ and $\overline{cd}$. Each dynamic spotlight can be

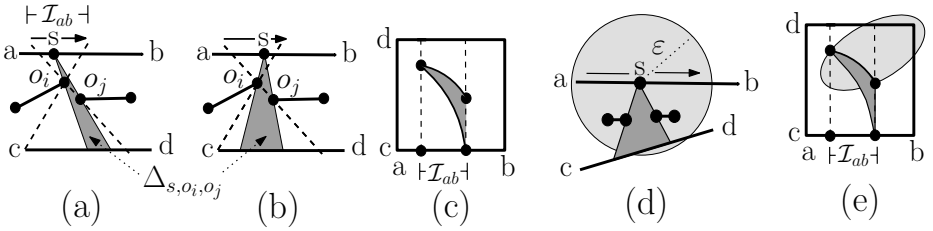



**Fig. 4.** (a),(b) A dynamic spotlight $\mathcal{L}_{\mathcal{D}}(\mathcal{I}_{ab}, o_i, o_j)$ for different positions of $s \in \mathcal{I}_{ab}$. $\Delta_{s,o_i,o_j} \cap \overline{cd}$ changes continuously and defines (c) $\mathcal{L}_{\mathcal{D}}$ in a cell $C$. (d),(e) Free space for $\mathcal{L}_{\mathcal{D}}$ is the intersection of a lightly-shaded free space ellipse with $\mathcal{L}_{\mathcal{D}}$.

associated with a unique visibility graph edge; thus, a cell $C$ contains $O(k^2)$ non-empty dynamic spotlights. Note that each spotlight is interior-disjoint from all other spotlights in $C$ because there is at most one line of sight path between any $s \in \overline{ab}$ and $t \in \overline{cd}$.

**Lemma 3.** *The free space for the $O(k^2)$ dynamic spotlights in a cell $C$ has $O(k^2)$ complexity and can be computed in $O(k^2 \log k)$ time and $O(k^2)$ space.*

*Proof Sketch.* Plane sweep techniques are used to compute the dynamic spotlights. Free space is defined by intersecting each dynamic spotlight with a free space ellipse [2] (see Figure 4e). □

### 3.2 Static Spotlights

Paths from $s \in \overline{ab}$ to $t \in \overline{cd}$ that have their final turn at an obstacle vertex $o_j$ have the form $\pi(s, o_j) \circ t$ and are represented by a structure that we call a *static spotlight* (see Figure 5). Let $\mathcal{I}_{ab}$ be an interval on the partition of $\overline{ab}$ that is defined by SPM($o_j$) $\cap\ \overline{ab}$. Note that $\pi(s, o_j)$ has the same combinatorial structure for all $s \in \mathcal{I}_{ab}$. Let $\mathcal{I}_{cd}$ be one of the $O(k)$ maximal connected intervals on $\overline{cd}$ with line of sight to $o_j$. A static spotlight is defined as $\mathcal{L}_{\mathcal{S}}(\mathcal{I}_{ab}, o_j, \mathcal{I}_{cd}) = \{(s,t) \mid s \in \mathcal{I}_{ab},\ t \in \mathcal{I}_{cd}\}$. A cell $C$ defined by line segments $\overline{ab}$ and $\overline{cd}$ contains $O(k^3)$ non-empty static spotlights because each obstacle vertex defines $O(k^2)$ possible $(\mathcal{I}_{ab}, \mathcal{I}_{cd})$ pairs. The *free space* for $\mathcal{L}_{\mathcal{S}}$ is $\{(s,t) \in \mathcal{L}_{\mathcal{S}} \mid d(s, o_j) + ||o_j - t|| \leq \varepsilon\}$ (see Figure 5).

Unlike dynamic spotlights, static spotlights represent shortest path *candidates* that can include suboptimal paths. This follows because a static spotlight for $o_j$ forces a path from $s$ to $t$ to go through $o_j$ even when $\pi(s,t)$ does not have its final turn at $o_j$. However, every shortest path $\pi(s,t)$ must be represented by at least one dynamic or static spotlight because shortest paths can only turn at obstacle vertices [3,14].

**Lemma 4.** *The free space for the $O(k^3)$ static spotlights in a cell $C$ has $O(k^3)$ complexity and can be computed in $O(k^3)$ expected time or $O(k^2 \lambda_6(k))$ deterministic time.*

*Proof.* For a fixed obstacle vertex $o_j$, SPM($o_j$) $\cap\ \overline{ab}$ partitions $\overline{ab}$ into intervals $\mathcal{I}_{ab,1}$, ... , $\mathcal{I}_{ab,O(k)}$ [15]. These intervals induce $O(k)$ vertical slabs in the cell $C$. Within each slab, the free space inequality $d(s, o_j) + ||o_j - t|| \leq \varepsilon$ describes a *semi-algebraic set* of constant description complexity (see Figure 5c). The overlay of the vertical slabs for

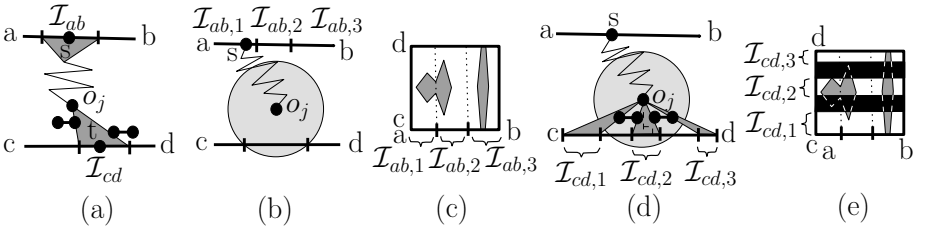



**Fig. 5. (a)** A static spotlight represents paths of the form $\pi(s, o_j) \circ t$. **(b)** A circle centered at $o_j$ with radius $\varepsilon - d(s, o_j)$ defines a candidate set of free space points in the **(c)** cell $C$. **(d)** Line of sight from $o_j$ can be restricted to visible intervals $\mathcal{I}_{cd,1}, ..., \mathcal{I}_{cd,O(k)} \in \overline{cd}$ by cutting out **(e)** darkly-shaded horizontal slabs.

$o_i$ and $o_j$ has $O(k)$ complexity, and within each of the $O(k)$ overlayed slabs the semi-algebraic sets for $o_i$ and $o_j$ have $O(1)$ intersections. Thus, the union of semi-algebraic sets has $O(k \cdot k^2)$ complexity over all pairs of obstacle vertices.

We now enforce line of sight information from each $o_j$ onto the maximal intervals $\mathcal{I}_{cd,1}, ..., \mathcal{I}_{cd,O(k)} \in \overline{cd}$ that are visible from $o_j$ (see Figures 5d and 5e). For each of the $O(k)$ intervals on $\overline{cd}$ that is *not* directly visible to $o_j$, we cut out a horizontal slab from the semi-algebraic set for $o_j$. Each of these $O(k)$ horizontal slabs intersects the semi-algebraic set for $o_j$ $O(k)$ times, yielding $O(k^2)$ vertices per $o_j$. Thus, the free space has $O(k^3)$ complexity. It can be computed using an incremental algorithm [1]. □

### 3.3 Fréchet Distance

**Lemma 5.** *Free space in a cell $C$ has $O(k^4)$ complexity and can be constructed for any $\varepsilon \geq 0$ in $O(k^4)$ expected time or $O(k^3 \lambda_6(k))$ deterministic time.*

*Proof.* The free space in a cell $C$ is the union of the free spaces for the dynamic and static spotlights (see Lemmas 3 and 4). Each of the $O(k^2)$ dynamic spotlights can intersect the semi-algebraic set for $o_j$ (see Figure 5c) $O(k)$ times, for a total of $O(k \cdot k^3)$ intersections of this type over all $o_j$. Each of the $O(k^2)$ dynamic spotlights can also intersect the $O(k^2)$ line of sight enforcing horizontal slabs (see Figure 5e). Hence, the arrangement of the dynamic and static spotlights has $O(k^4)$ complexity. The arrangement can be computed with an incremental algorithm [1]. □

**Theorem 7.** *The Fréchet distance for polygonal curves $A$ and $B$ in a polygonal domain with $k$ vertices can be computed in $O(N^2 k^4 \log k \log Nk)$ time, and the free space diagram can have $\Omega(N^2 k^2)$ complexity. $N$ is the complexity of $A$ and $B$. The* weak *Fréchet distance can be computed in $O(N^2 k^4 \log Nk)$ expected time or $O(N^2 k^3 \lambda_6(k) \log Nk)$ deterministic time. Both approaches use $O(Nk + k^4)$ space.*

*Proof.* The decision problem is answered by propagating reachability information through all cells with a plane sweep. The *weak* decision problem can be answered by marking a set of connected free space components as reachable. By Lemma 5, the free space diagram has $O(N^2 k^4)$ free space vertices. Each of these vertices is parameterized by $\varepsilon$ as an algebraic curve that has constant degree and description complexity.

By applying parametric search [2] to these curves and using the $O(N^2k^4 \log k)$ runtime for the decision problem, the Fréchet distance can be computed in $O(N^2k^4 \log k \log Nk)$ time. The weak Fréchet distance can be computed similarly. The space bound follows by storing one cell at a time plus the cell boundaries on the two rows required for dynamic programming [2]. Our lower bound for the free space diagram can be found in [6]. □

### 3.4 Two-Segment Shortest Path Map

Chiang and Mitchell [4] support queries between any two points in a polygonal domain with $k$ vertices after $O(k^{11})$ preprocessing. Using our spotlight structures, we define a shortest path map SPM($\overline{ab}$, $\overline{cd}$) that supports optimal query time from any source point $s \in \overline{ab}$ to any destination point $t \in \overline{cd}$ after $O(k^4\lambda_6(k))$ preprocessing.

All dynamic spotlights encode $\pi(s,t) = s \circ t$ paths and contribute to SPM($\overline{ab}$, $\overline{cd}$). Unlike dynamic spotlights which are always interior-disjoint and consist entirely of shortest paths, static spotlights can overlap and encode suboptimal paths. The main task to construct SPM($\overline{ab}$, $\overline{cd}$) is to "clip" the static spotlights into an interior-disjoint set of optimal paths. Regions of overlap between all static spotlights defined by $o_j$ and $o_l$ can be resolved with a hyperbolic *bisector* $B_{jl}(s) = \{(s,t) \mid d(s,o_j) + ||o_j - t|| = d(s,o_l) + ||o_l - t||\}$ that is commonly used in additively weighted Voronoi diagrams.

**Theorem 8.** *SPM($\overline{ab}$, $\overline{cd}$) can be constructed in $O(k^5)$ expected time or $O(k^4\lambda_6(k))$ deterministic time and $O(k^5)$ space. After this preprocessing, queries $d(s,t)$, $\pi(s,t)$ for any $s \in \overline{ab}$ and $t \in \overline{cd}$ take $O(\log k + K)$ time, where $K$ is the complexity of any returned path. SPM($\overline{ab}$, $\overline{cd}$) can have $\Omega(k^2)$ complexity.*

*Proof Sketch.* We partition the parameter space defined by $\overline{ab}$ and $\overline{cd}$ such that for all points $(s,t)$ in a region the shortest path $\pi(s,t)$ has the same combinatorial structure. This arrangement is defined by inserting a static spotlight bisector for each pair of obstacles into the arrangement of the dynamic and static spotlights (see Lemmas 4 and 5). The resulting arrangement has $O(k^5)$ complexity by a slab argument and can be constructed with an incremental algorithm [1]. The $\Omega(k^2)$ lower bound follows from the lower bound in Theorem 7. □

## 4 Conclusion

We develop algorithms to compute Voronoi diagrams, shortest path maps, and the Fréchet distance in the plane with polygonal obstacles. We also develop *tight* bounds for our Voronoi diagrams. It would be interesting to extend our shortest path map structures SPM($\overline{ab}$,$\mathbb{R}^2$) and SPM($\overline{ab}$,$\overline{cd}$) to support queries between any two points in the plane and to compare the resulting runtimes with the $O(k^{11})$ Euclidean runtime of Chiang and Mitchell [4]. It would also be interesting to extend our Fréchet distance approach to a set of more than two polygonal curves (see related work by Dumitrescu and Rote [8]). Results on the Hausdorff distance were omitted from this version due to space concerns.

## References


1. Agarwal, P.K., Sharir, M.: Davenport–Schinzel Sequences and Their Geometric Applications. In: Handbook of Computational Geometry, pp. 1–47. Elsevier, Amsterdam (2000)
2. Alt, H., Godau, M.: Computing the Fréchet distance between two polygonal curves. International Journal of Computational Geometry and Applications 5, 75–91 (1995)
3. Arkin, E.M., Mitchell, J.S.B., Suri, S.: Optimal link path queries in a simple polygon. In: 3rd Symposium on Discrete Algorithms (SODA), pp. 269–279 (1992)
4. Chiang, Y., Mitchell, J.S.B.: Two-point Euclidean shortest path queries in the plane. In: 10th Symposium on Discrete Algorithms (SODA), pp. 215–224 (1999)
5. Cook IV, A.F., Wenk, C.: Geodesic Fréchet distance inside a simple polygon. In: 25th Symposium on Theoretical Aspects of Computer Science (STACS) (2008)
6. Cook IV, A.F., Wenk, C.: Geodesic Fréchet distance with polygonal obstacles. Technical Report CS-TR-2008-010, University of Texas at San Antonio (2008)
7. Cook IV, A.F., Wenk, C.: Min-link shortest path maps and Fréchet distance. Technical Report CS-TR-2008-011, University of Texas at San Antonio (2008)
8. Dumitrescu, A., Rote, G.: On the Fréchet distance of a set of curves. In: 16th Canadian Conference on Computational Geometry (CCCG), pp. 162–165 (2004)
9. Efrat, A., Guibas, L.J., Har-Peled, S., Lin, D.C., Mitchell, J.S.B., Murali, T.M.: Sweeping simple polygons with a chain of guards. In: 11th Symposium on Discrete Algorithms (SODA), pp. 927–936 (2000)
10. Gewali, L., Meng, A., Mitchell, J.S.B., Ntafos, S.: Path planning in $0/1/\infty$ weighted regions with applications. In: 4th Symposium on Computational Geometry (SoCG), pp. 266–278 (1988)
11. Guibas, L.J., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.E.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica 2, 209–233 (1987)
12. Henzinger, M.R., Klein, P., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. Journal of Computer and System Sciences 55(1), 3–23 (1997)
13. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. SIAM Journal on Computing 28(6), 2215–2256 (1999)
14. Maheshwari, A., Sack, J.-R., Djidjev, H.N.: Link distance problems. In: Handbook of Computational Geometry (1999)
15. Mitchell, J.S.B.: Geometric shortest paths and network optimization. In: Handbook of Computational Geometry (1998)
16. Mitchell, J.S.B., Rote, G., Woeginger, G.J.: Minimum-link paths among obstacles in the plane. In: 6th Symposium on Computational Geometry (SoCG), pp. 63–72 (1990)
17. Suri, S.: A linear time algorithm for minimum link paths inside a simple polygon. Computer Vision and Graphical Image Processing (CVGIP) 35(1), 99–110 (1986)

# ORCA Reduction and ContrAction Graph Clustering*

Daniel Delling, Robert Görke, Christian Schulz, and Dorothea Wagner

Faculty of Informatics, Universität Karlsruhe (TH)
{delling,rgoerke,cschul,wagner}@informatik.uni-karlsruhe.de

**Abstract.** During the last years, a wide range of huge networks has been made available to researchers. The discovery of natural groups, a task called *graph clustering*, in such datasets is a challenge arising in many applications such as the analysis of neural, social, and communication networks.

We here present ORCA, a new graph clustering algorithm, which operates locally and hierarchically contracts the input. In contrast to most existing graph clustering algorithms, which operate globally, ORCA is able to cluster inputs with hundreds of millions of edges in less than 2.5 hours, identifying clusterings with measurably high quality. Our approach explicitly avoids maximizing any single index value such as *modularity*, but instead relies on simple and sound structural operations. We present and discuss the ORCA algorithm and evaluate its performance with respect to both clustering quality and running time, compared to other graph clustering algorithms.

## 1 Introduction

In the exploration and the analysis of large and complex networks such as the World Wide Web, social and natural networks and recommendation systems or protein dependencies, *graph clustering* has become a valuable tool[1]. The majority of algorithms for graph clustering are based on the paradigm of intra-cluster density versus inter-cluster sparsity. Several formalizations of this intuition have been proposed and evaluated, an overview of such techniques is given in [12] and [14]. Due to the increasing availability of digitized network data, computational puissance and storage media and upcoming trends such as time-expanded clustering [23], networks comprising up to several millions of vertices are today's subjects of research. However, despite of technical advances, instances of this size still pose algorithmic challenges, and render techniques that are successfully applied on smaller problems infeasible.

**Our Contribution.** In this work we present the ORCA reduction and contraction algorithm, a locally operating, fast graph clustering algorithm, which is

* This work was partially supported by the DFG under grant WA 654/15-1 and by the EU under grant DELIS (contract no. 001907).

[1] We use the two terms *network* and *graph* interchangeably.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 152–165, 2009.

capable of handling huge instances that state-of-the-art methods cannot cope with and which exhibits remarkably good results of community detection. The emphasis is on the feasibility of applying the algorithm on huge problem instances. ORCA is designed to rely on simple structural observations that immediately translate to intra-cluster density and inter-cluster sparsity, avoiding the direct maximization of some index. On several publicly available networks we evaluate the performance of ORCA with respect to running time and several quality measures for clusterings. We show that ORCA scales well and solve instances with half a billion edges, and yields good clustering quality.

This paper is organized as follows. After introducing related work and assets of local methods, necessary notation is given. In Section 1, we describe ORCA. In Section 3 we show our findings on its general feasibility and on parameter choices. Our experiments in Section 4 compare ORCA to related approaches on a number of instances. We conclude with a discussion and future work in Section 5.

**Related Work.** It is common knowledge that there is no single best strategy for graph clustering, which justifies the plethora of existing approaches. Moreover, most quality indices for graph clusterings have turned out to be NP-hard to optimize and rather resilient to effective approximations, see e.g., [11,31,8], allowing only heuristic approaches for optimization. Other approaches often rely on specific strategies with high running times, e.g., the iterative removal of central edges [27], or the direct identification of dense subgraphs [19]. Provably good methods with a decent running time include such that have a spectral embedding of the vertices as the basis for a geometric clustering [13], *min-cut tree* clustering [21], a technique which guarantees certain bounds on bottlenecks and an approach which relies on random walks in graphs staying inside dense regions with high probability [30]. Related to the latter is an effective bottom-up strategy called *walktrap* [28] that iteratively updates a distance measure based on local random walks, which governs hierarchical agglomerations.

The greedy maximization of the quality index *modularity* via iterative agglomeration of vertices into growing clusters [15] caused a surge of follow-up studies on related methodologies (see [11] or [14] for an overview). A variant thereof, which abandons global greedyness, has recently been presented in [10]. Here a significant speedup is achieved by only locally deciding about agglomeration and hierarchically reducing the graph repeatedly. This conceptually simple but effective local method of greedy *modularity* maximization constructs consecutive hierarchy levels of a clustering by letting each vertex decide to which neighboring cluster/vertex to affiliate, and then contracts each stable affiliation. It is worth noting that this approach is similar to ORCA, on a rough scale. However, while this technique is explicitly designed to maximize *modularity*—which it achieves quite well—and thus solely relies on one measure as the single criterion, ORCA builds a clustering without this bias towards *modularity*. Although *modularity* has proven to be a rather reliable quality measure, it is known to behave artificially to some extent.

Methods that potentially identify overlapping clusters, or leave nodes unclustered (see, e.g., [25] and [19]), are a slightly different field and thus not discussed herein.

A more generous overview of the field calls for a few words about what graph clustering is not. This context of *graph partitioning* strongly differs from general graph clustering in that the number and possibly the size of clusters are crucial input parameters. Note furthermore that graph clustering is related but essentially different from the field of *data clustering* where data points are embedded in a high dimensional feature space and no explicit edge structure is present.

**Making a Case for Local Methods.** Many widespread clustering algorithms iterate some global mechanism a linear number of times, which is particularly typical for classic bottom-up agglomerative approaches (e.g., greedy index maximization [15] or the walktrap [28]), or they include some direct technique that is both time and space consuming (e.g., global Markov chains [30] or iterative conductance cutting [24]). Operating locally in graphs avoids these issues, if local operations are simple and bounded in number. Apart from this and their obvious eligibility for parallelization, more facts encourage local approaches. First, heuristics that maximize a clustering quality index are known to exhibit what has been coined *scaling behavior* [9,11], which can roughly be described as a technique not reproducing results after, say, doubling an instance. Local methods can avoid such effects. Second, a limited set of local operations on a graph, e.g., iterating over incident edges, allows for fast data structures that grant further speed-ups and fit most graphs into the main memory of a server with 32GB of RAM. Third, local strategies are better suited for dynamization. They potentially miss some global structure but since it is natural to assume that local changes on graphs are of local semantics only, local decisions on the clustering *should* suffice.

**Notation.** We assume that $G = (V, E, \omega)$ is an undirected, weighted, and simple graph[2] with the edge weight function $\omega\colon E \to [0, 1]$. We set $|V| =: n, |E| =: m$ and $\mathcal{C} = \{C_1, \ldots, C_k\}$ to be a partition of $V$. We call $\mathcal{C}$ a *clustering* of $G$ and sets $C_i$ *clusters*. A clustering is *trivial* if either $k = 1$, or all clusters contain only one element, i.e., are *singletons*. We identify a cluster $C_i$ with its node-induced subgraph of $G$. Then $E(\mathcal{C}) := \bigcup_{i=1}^{k} E(C_i)$ are *intra-cluster* edges and $E \setminus E(\mathcal{C})$ *inter-cluster* edges, with cardinalities $m(\mathcal{C})$ and $\overline{m}(\mathcal{C})$, respectively. We denote the number of non-adjacent intra-cluster (inter-cluster) pairs of vertices as $m(\mathcal{C})^c$ ($\overline{m}(\mathcal{C})^c$). A node $v$'s (standard) *neighborhood* is $N(v) := \{w \in V \mid \{v, w\} \in E\}$, and the set of vertices within distance $d$ of $v$ is denoted as the *d-neighborhood* $N_d(v) = \{w \in V \mid w \neq v, \mathrm{dist}(v, w) \leq d\}$, where $\mathrm{dist}(v, w)$ denotes the length of the shortest path between $v$ and $w$. since disconnected clusters are unreasonable.

[2] We call elements of $V$ *vertices*, and reserve the term *nodes* (or *super-nodes*) for entities that potentially embody several vertices during some contraction stage. However, the reader is absolutely fine if this distinction eludes her or him. Links between vertices/nodes are uniformly called *edges*.

**Quality Indices.** We measured the quality of clusterings with a range of quality indices, discussed, e.g., in [12], however, we set our focus on the indices *coverage* [12], *performance* [30], *inter-cluster conductance* [24] and *modularity* [27] in this work, since they are the most studied ones. In brief, *coverage* is the fraction of intra-cluster edges, and *performance* is the fraction of correctly classified vertex pairs, w.r.t. the set of edges. *Modularity* compares the coverage of a clustering to the same value after rearranging edges randomly and *inter-cluster conductance* returns the worst (i.e. the thickest) bottleneck created by separating a cluster from the rest of the graph. All these definitions generalize in a natural way as to take edge weights $\omega(e)$ into account, for a discussion thereof see [12], [26] and [22]. For further discussions of these indices we refer the reader to the given references, and simply state their formal (unweighted) definitions:

$$\mathrm{perf}(\mathcal{C}) := \frac{m(\mathcal{C}) + \overline{m}(\mathcal{C})^c}{\frac{1}{2}n(n-1)} \qquad \mathrm{mod}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} - \frac{1}{4m^2}\sum_{C\in\mathcal{C}}\left(\sum_{v\in C}\deg(v)\right)^2$$

$$\mathrm{cov}(\mathcal{C}) := \frac{m(\mathcal{C})}{m} \qquad \mathrm{icc}(\mathcal{C}) := 1 - \max_{C\in\mathcal{C}} \frac{\sum_{v\in C}\deg(v) - 2E(C)}{\min\Big(\sum_{v\in C}\deg(v), \sum_{v\in V\setminus C}\deg(v)\Big)}$$

## 2 The ORCA-Algorithm

The general approach of ORCA is as follows: Preliminarily prune the graph of irrelevant vertices, then, iteratively identify dense neighborhoods and contract them into super-nodes; after contraction repeat the second step on the next hierarchy level or, if this fails, remove low-degree nodes and replace them by shortcuts. Do this until the whole graph is contracted. Due to the widely agreed on fact that no quality function can be elected *best* in general, an important design goal for ORCA was to refrain from having any decision base on such an index. Instead we only rely on fundamental and indisputable structural properties such as the 2-core, the similarity of a subgraph to a clique and local sparsity. The following sections detail each step of ORCA in the order of their execution, things are then put together in Section 2. We postpone technical details of our implementation and our data structures to Section 4.

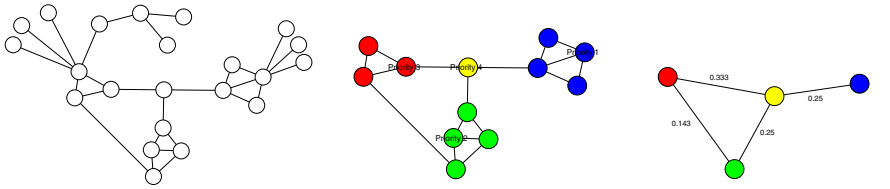
**Core-2 Reduction.** The initial preprocessing step of ORCA is a simple reduction of the instance to its 2-core. Introduced in [29], the 2-core of a graph is the maximal vertex-induced subgraph in which each vertex has at least degree 2. Note that the running time of this procedure CORE-2 REDUCTION is linear in $m + n$. The rationale behind this pruning step is as follows. Vertices in the 1-coreshell are tree-like appendices, which are highly ambiguous to cluster sensibly anyway (see Figure 1(a)). Since in a reasonably modeled real-world network such appendices should not be large, we make the straightforward assumption

that any tree appendix is to be clustered together with its anchor vertex in the 2-core, which is done in a postprocessing step. Depending on the nature of the input, this step can significantly reduce the size of the actual problem instance.

**Local Search for Dense Regions.** We now describe an integral part of ORCA, the elementary detection of *dense regions*. Roughly speaking, a dense region $R \subseteq V$ is a set of $c$ nodes within distance $d$ of some seed node $v$, such that each node $w \in R$ is within distance at most $d$ of at least $|N_d(v)|/\gamma$ other nodes of $N_d(v)$. This step is employed repeatedly and iteratively as will be described in the next section. Each call of the procedure DENSE-REGION-LOCAL (we omit pseudocode) is parameterized by a seed node $v$ and two positive reals $\gamma$ and $d$ which set the required degree of density and the size of the neighborhood to be explored, respectively. Low values of $\gamma$ impose a stricter criterion on density, which leads to DENSE-REGION-LOCAL returning smaller regions. First, the dense region is initialized with $v$; then each node $w$ within distance $d$ or less from $v$, in turn has each node $x \in N_d(w)$ increment its *seen*-attribute. For each node this attribute thus stores how many nodes of $N_d(v)$ it considered a $d$-neighbor. In a second step, this procedure now adds each node $w \in N_d(v)$ to the dense region, which has been seen by at least a $\gamma$-fraction of the nodes in $N_d(v)$, and returns the assembled region as in Figure 1(b). Note that allowing nodes in any $N(w)$ into a region might produce undesirable "holes". The time complexity of this procedure is highly dependent on $d$. Setting $d = 1$ at most $\Delta$ nodes each have their at most $\Delta$ neighbors increment their attribute, yielding $O(\Delta^2)$.

**Contraction of Dense Regions.** The second elementary operation on the graph is the contraction of a subgraph into a single *super-node*. The main goal of CONTRACTION is to reduce the size of the problem instance by summarizing parts that have already been solved; Figure 1 illustrates its effect. The contraction of a node-induced subgraph of $G$ is straightforward. A new node replaces the subgraph, and is receives former adjacencies to other nodes are replaced by new edges, weighted by their average adjacency to the region. A rough upper bound on the running time of such a CONTRACTION clearly is $O(m)$, since each edge is touched only once. An amortized analysis of the time complexity of a series of calls of Algorithm DENSE-REGION-LOCAL and CONTRACTION will be given in the next section.

**Global Dense Region Detection.** While procedure DENSE-REGION-LOCAL identifies a dense region, and procedure CONTRACTION reduces it to a super-node the following algorithm, called DENSE-REGION-GLOBAL, orchestrates the calls to these local operations. Roughly speaking, a single run of DENSE-REGION-GLOBAL assigns each node to a prioritized dense region (Figure 1(b)), and then abstracts the graph to the next hierarchy level by replacing each dense region by a super-node (Figure 1(c)). The crucial observation is that DENSE-REGION-GLOBAL reduces the size of the instance very quickly and in a meaningful way, paving the way for further and more far-reaching clustering steps.




(a) The input instance, first it will be reduced to its 2-core (b) Now, dense regions are identified (colors) (c) Each dense region is contracted

**Fig. 1.** Dense regions (by colors) are contracted in the order of contraction priority

Given parameters $\gamma$ and $d$ as above, DENSE-REGION-GLOBAL first calls for each node $v$ in the graph DENSE-REGION-LOCAL using $v$ as the seed node. Each dense region returned is then inserted into a priority queue with a priority key that expresses how significant the region actually is, as indicated in Figure 1(b). This key is computed by evaluating the following simple function $\psi$ that measures the average edge weight mass incident with a node in the region:

$$\psi : \mathcal{P}(V) \to [0,1] \qquad D \mapsto \frac{\sum_{e \in E(D)} \omega(e)}{|D|} \quad , \quad D \subseteq V$$

An alternative approach to accomplish this, which we have yet to examine, is given in [33]. After determining and queuing for each node $v \in V$ its dense region, regions are popped from the queue and contracted. Since we seek a proper partition of nodes, we first have to reassemble dense regions excluding all nodes that are assigned to dense regions with a higher priority by tagging them as invalid. Experiments showed that reordering the queue after such exclusions is costly and yields a minimal gain in quality, thus initial priorities are kept.

In total, $n$ calls of DENSE-REGION-LOCAL account for $O(n\Delta^2)$ and $n$ priority queue operations require $O(n \log n)$ time. During the course of all CONTRACTION operations each edge is touched at most twice, which yields an amortized time of $O(m)$. Summing up, DENSE-REGION-GLOBAL is in $O(m + n(\Delta^2 + \log n))$.

**Densification via Shortcuts.** While initially, low degree nodes or appendices are pruned and assigned to clusters in a canonical way (see CORE-2 REDUCTION), this might not be desirable for super-nodes incorporating thousands of elementary vertices. However, such low degree elements are potentially incompatible with a given choice of the threshold parameter $\gamma$ Thus, we *densify* a graph, by replacing a low-degree node $v$ with a clique construction of *shortcuts* among its neighbors as in Figure 2. Similar to nodes removed during the CORE-2 REDUCTION, such a node is then potentially affiliated with the node it is most strongly connected to.

Algorithm SHORTCUTS loops through all nodes $v$ with the minimum degree $\delta$, ensure that all pairs $\{v_1, v_2\}$ of nodes adjacent to $v$ become connected and removes $v$. The weight on the edge between $\{v_1, v_2\}$ is then set to its new *conductivity*, a concept borrowed from electrical circuits: To the old weight, which is 0 if the edge was not present, the term $1/(\frac{1}{\omega(\{v_1,v\})} + \frac{1}{\omega(\{v_2,v\})})$ is added that


**Fig. 2.** SHORTCUTS using $\delta = 2$, a shortcut between nodes 1 and 2, replaces node 3

expresses the conductivity of the path $\pi = v_1, v, v_2$. The rationale is that this adjustment maintains conductivities between all neighbors while densifying the graph structurally, again enabling the detection of dense regions. Analyzing the time complexity very roughly yields a worst case complexity of $O(n \cdot \Delta^2)$.

**Putting Things Together.** This section details the overall approach of ORCA, i.e., Algorithm 1 which repeatedly calls all necessary procedures. After the CORE-2 REDUCTION, for as long as there are more than two nodes left in the graph, DENSE-REGION-GLOBAL and SHORTCUTS iteratively reduce and contract the graph. If at any time no significant contraction is possible (Line 4), SHORTCUTS removes low degree nodes and compactifies the graph (Line 5). After each successful global contraction stage we store the current clustering (Line 6). ORCA returns the whole clustering hierarchy alongside evaluated quality indices for manual choice of the preferred clustering. Additionally a recommendation is given, based on quality indices. In practice, procedure SHORTCUTS is hardly ever called, and no value of $\delta > 2$ was ever used, leaving SHORTCUTS with a marginal impact on running times. Only with ill-modeled graphs, pathological examples or unreasonable choices of $\gamma$ does this procedure ever operate on a graph with size comparable to the input, usually it is only called after a series of contraction steps. We discuss good choices for the two parameters $\gamma$ and $d$ in the following section.

**Algorithm 1.** ORCA

```
   Input: G = (V, E, ω), d, γ ∈ ℝ+
1  CORE-2 REDUCTION(G)
2  while |V| > 2 do
3  |   DENSE-REGION-GLOBAL(G, γ, d)
4  |   if |V_old| > 0.25|V| then
5  |   |   SHORTCUTS(G,δ)
6  |   else Store current clustering
```

The total running time of ORCA derives from its subroutines, and factor $h$, the number of iterations of the main loop or, in other words, the depth of the clustering hierarchy, which is $n$ in the worst case but always below $\log n$ in practice. SHORTCUTS However, since this work is on practical performance, we refrain from a detailed analysis and close with our observation that empirically the total running time of ORCA sums up to $O(\log n(m + n(\log n + \Delta^2)))$.

**Engineering ORCA.** We here shortly present two small optimizations for ORCA. It turns out that for particular graphs with a few high-degree vertices the running time of ORCA is dominated by the $\Delta^2$ term. Hence, we use a little tweak to reduce running times: After the CORE-2 REDUCTION, we remove all vertices with a degree greater than $4 \cdot \sqrt{n}$ from the graph, as these global hubs hardly

indicate local density. Later we assign such a vertex to the cluster which contains most of its neighbors.

At later iteration steps, it is possible that the current clustering still contains many singleton elementary vertices. In order to reduce these undesirable clusters, we assign each singleton vertex to the cluster it is connected to most strongly.

## 3 Parameters and Feasibility

This brief section yields insights on reasonable choices for the parameters $\gamma$ and $d$ and corroborates the feasibility of ORCA on two toy examples. Parameter testing was conducted with the aid of two random generators that served as a source for graphs with an implanted clustering structure.

**Parameter Estimation.** We employed two generators for random test instances: first, an *Attractor Generator*, which is based on assigning vertices, randomly placed in the plane, to clusters in a Voronoi fashion and connecting them with probability based on distance and cluster affiliation; and second, a *Significant Gaussian Generator* which partitions the vertex set into clusters and then interconnects vertices similar to the Erdős-Rényi model, using intra- and inter-cluster edge-probabilities. We refer the reader to [16] for details on these generators, where they are evaluated and shown to produce reasonable and variable pre-clustered graphs with a tunable clarity. In a broad study on smaller graphs with 50 to 1000 vertices (step size 50), we varied the density parameter of the Attractor Generator from 0.5 (mostly disconnected stars) to 2.5 (almost a clique) in steps of 0.1, and we varied the intra-edge probabilities of the Significant Gaussian Generator between 0.1 (very sparse) and 0.7 (almost cliques) in steps of 0.1, having the ratio of inter-cluster edges range between 0.1 and 0.5 (0.05 step size). For each such setup we performed 30 experiments and evaluated the results of ORCA with respect to *performance*, *coverage* and *modularity*.

The results of this parameter exploration revealed that setting depth $d$ to 1 for unweighted graphs is the best choice in general. The main reason for this is that a broader candidate neighborhood encourages "holes" inside clusters which at a later stage cannot be repaired. Parameter $\gamma$, proved to be feasible for values between 2 and 10 for sparse graphs, with low values working best in general.

**Two Toy Examples.** In the following we show clustering results for two graphs, one of which is well known in the clustering community, and one that very fundamentally incorporates a clustering hierarchy. The latter graph is clearly organized into 16 small groups which themselves are organized into four groups, it was proposed in [25], as a basic benchmark for hierarchy detection. Figure 3 shows ORCA's results, a clear success. The second example was compiled by Wayne Zachary [32] while doing a study on the social structure of friendship between the members of a university sports club. The two real-world factions are indicated by color in Figure 4. Using $\gamma = 2$ and $d = 1$, ORCA clusters this graph as illustrated in Figure 5, where hierarchy levels 1 to 3 are shown. Level 3 misclassifies only a single vertex (vertex number 10, in the original numbering).

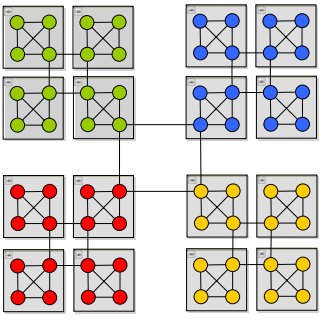


**Fig. 3.** Hierarchy levels 1 (grouping) and 3 (colors) found by ORCA

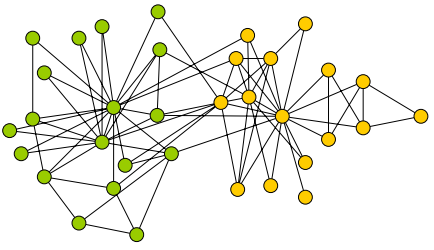
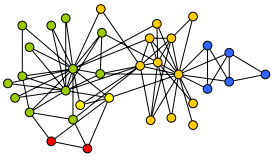
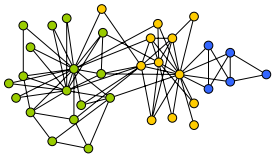


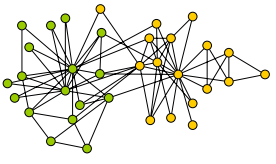
**Fig. 4.** Zachary in reality (cov = 0.87, perf = 0.62, icc = 0.87, mod = 0.37)



(a) cov = 0.73, perf = 0.79, icc = 0.22, mod = 0.39



(b) cov = 0.82, perf = 0.71, icc = 0.75, mod = 0.40



(c) cov = 0.87, perf = 0.62, icc = 0.87, mod = 0.37

**Fig. 5.** Hierarchy levels 1 to 3 (left to right), using $\gamma = 2$ and search depth $d = 1$

## 4 Experiments

*Implementation Details.* Another field with huge datasets in algorithm engineering is the development of fast shortest path algorithms (see [17]). There we made the experience that in most cases, the loss with respect to running times stemming from external libraries is rather high. As the goal of ORCA was the development of a fast clustering algorithm, our implementation is written in C++, using only the STL at some points. As priority queue we use a binary heap, and we represent the graph as an adjacency array. In the following we report running times and quality achieved by ORCA, using fixed parameters $\gamma = 2$ and $d = 1$. For measuring the quality of a clustering, we evaluate the score achieved by coverage, performance, inter-cluster conductance, and modularity (see Section 1). Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.3. It is clocked at 2.6 GHz, has 32 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.2, using optimization level 3.

*Inputs.* We use three different types of inputs. Small world graphs, webgraphs and road networks. The first group contains three graphs. The first dataset represents the Internet on the router level, it is taken from the CAIDA webpage [2]. The second graph is a citation network, obtained from crawling citeseer [1]. The final dataset is a co-authorship [7] network, which is obtained from DBLP [3]. The

second group of inputs are webgraphs taken from [6]. Nodes represent webpages, edges represent hyperlinks. We use four graphs, namely cnr-2000, eu-2005, in-2004 and uk-2002. The final group of inputs are road networks taken from the DIMACS homepage [18]. We use three graphs, the first one represents Florida, the second one central USA while the last one is the full road network of the US. Sizes are given in Tables 1-3.

**Hierarchy of Clusterings.** We first evaluate the clustering hierarchy computed by ORCA. Figure 6 shows the score of all quality indices and the number of clusters for all levels of the hierarchy. Due to space limitations, we restrict ourselves to one representative of each group of our inputs. As on higher hierarchy levels, the number of clusters decreases, coverage increases. It turns out that inputs are too large (contain degeneracies) for the worst-case index inter-cluster conductance to yield reasonable insights. Interestingly, modularity first increases and later decreases, yielding a clear preference. For sparse graphs performance is known to favor fine clusterings, but the point where performance severely decreases agrees with what modularity favors. Summarizing, ORCA produces a reasonable clustering hierarchy from which a user can choose his favorite. A good choice seems to be a level, where performance, coverage, and modularity score best.

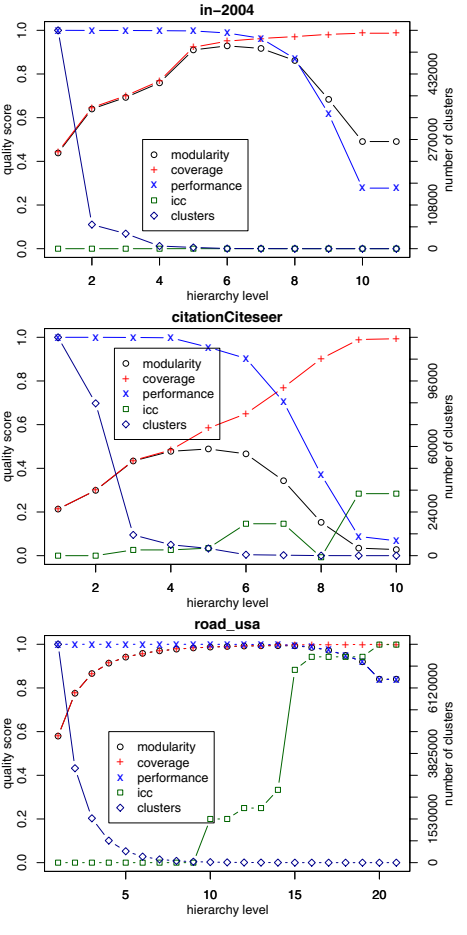



**Fig. 6.** Quality of the clustering hierarchy computed by ORCA. The inputs are the webgraph in-2004, the small world citation network, and the road network of the US.

**Comparison.** Next, we compare our results with competing graph clustering algorithms. We evaluate the *global* greedy modularity algorithm [15], the new local variant [10], and walktrap [28]. We omit a number of other promising approaches, e.g., via simulated annealing [20], which are computationally too demanding for these instance sizes. The implementations of global greedy and walktrap are taken from the *igraph* library [5], the code for local greedy is taken from [4]. Note that in the following we only report the

clustering with maximum modularity for ORCA, quality scores of other levels can be found in Figure 6.

*Small World Graphs.* Tab. 1 reports running times and quality score achieved by ORCA and competing algorithms applying our three small world inputs. We observe excellent running times for ORCA with feasible quality scores. Moreover, we observe that in terms of running time, global greedy and walktrap cannot compete with the local algorithms. While this is to be expected, note that they do not achieve better quality scores either. Comparing ORCA with local greedy we observe that ORCA tends to produce finer clusterings. Roughly speaking, quality scores are worse than for local greedy but still feasible. For the instance citation, ORCA fails to find a very good clustering, this is mainly due to many high degree hubs—milestone papers or major surveys, where ORCA seems to take too many simplification steps (see Engineering ORCA above).

*Webgraphs.* Next, we focus on the scalability of our approach. The webgraphs we have taken from [6] have similar properties but different sizes. It turns out that the global greedy algorithm needs too much memory to be executed while walktrap takes too much time. Hence, we compare ORCA with the local greedy algorithm only, Tab. 2 reports running times and quality scores. At a glance we observe that ORCA provides good clusterings within reasonable computation times. All graphs are clustered in less than 2.5 hours. Only for eu-2005, we achieve a modularity score of less than 0.85, and do not agglomerate long enough to find a better clustering. Interestingly, intercluster conductance is always almost zero for ORCA. This stems from the fact that, intercluster conductance, being a worst-case quality index, always considers a clustering with at least one singleton a very poor clustering. While this may make sense for small inputs, such a worst-case index is not reliable for large inputs. As observed in Fig. 6, in most cases clusterings on a higher level score higher values. Comparing ORCA with local greedy, we observe that ORCA has longer running times but achieves comparable quality scores on these large inputs, neglecting icc. On cnr-2000 and

**Table 1.** Running times and quality of the algorithms on small world graphs

| Instance | $n/m$ | Algorithm | time [s] | clusters | icc | perf. | cov. | mod. |
|---|---|---|---|---|---|---|---|---|
| caida Router | 190 914 / 607 610 | global greedy | 0:20 | 1672 | 0.5667 | 0.9397 | 0.9052 | 0.7639 |
| | | Walktrap | 0:23 | 24952 | 0.0000 | 0.9858 | 0.7540 | 0.6693 |
| | | local greedy | < 0:01 | 442 | 0.6410 | 0.9720 | 0.8944 | 0.8440 |
| | | ORCA | < 0:01 | 492 | 0.2105 | 0.9578 | 0.7113 | 0.6500 |
| co-Authors | 299 067 / 977 676 | global greedy | 1:15 | 2930 | 0.5000 | 0.9187 | 0.8638 | 0.7413 |
| | | Walktrap | 0:55 | 37669 | 0.0000 | 0.9790 | 0.7089 | 0.6432 |
| | | local greedy | < 0:01 | 269 | 0.6196 | 0.9813 | 0.8486 | 0.8269 |
| | | ORCA | < 0:01 | 2038 | 0.1733 | 0.9954 | 0.7274 | 0.7212 |
| citations | 268 495 / 1 156 647 | global greedy | 2:08 | 1927 | 0.2857 | 0.8253 | 0.9106 | 0.6650 |
| | | Walktrap | 0:51 | 16822 | 0.0000 | 0.9690 | 0.7449 | 0.6824 |
| | | local greedy | < 0:01 | 147 | 0.5983 | 0.9544 | 0.8593 | 0.8037 |
| | | ORCA | < 0:01 | 4201 | 0.0000 | 0.9973 | 0.5649 | 0.5100 |

**Table 2.** Running times and quality of the algorithms on webgraphs

| Instance | $n/m$ | Algorithm | time [s] | clusters | icc | perf. | cov. | mod. |
|---|---|---|---|---|---|---|---|---|
| cnr-2000 | 325 556 | local greedy | 8 | 242 | 0.8571 | 0.9799 | 0.9971 | 0.9130 |
| | 5 565 376 | ORCA | 28 | 110 | 0.0002 | 0.9632 | 0.9427 | 0.8567 |
| eu-2005 | 862 664 | local greedy | 23 | 326 | 0.7668 | 0.9643 | 0.9708 | 0.9376 |
| | 32 778 307 | ORCA | 307 | 217 | 0.0002 | 0.9458 | 0.7965 | 0.7014 |
| in-2004 | 1 382 908 | local greedy | 36 | 1004 | 0.0000 | 0.9931 | 0.9234 | 0.9094 |
| | 27 560 318 | ORCA | 313 | 740 | 0.0002 | 0.9877 | 0.9503 | 0.9288 |
| uk-2002 | 18 520 486 | local greedy | 432 | 6280 | 0.0000 | 0.9981 | 0.5693 | 0.5671 |
| | 529 444 599 | ORCA | 8807 | 66595 | 0.0000 | 0.9995 | 0.8758 | 0.8749 |

**Table 3.** Running times and quality of the algorithms on road networks

| Instance | $n/m$ | Algorithm | time [s] | clusters | icc | perf. | cov. | mod. |
|---|---|---|---|---|---|---|---|---|
| florida | 1 070 376 | local greedy | 15 | 541 | 0.9845 | 0.9978 | 0.9971 | 0.9948 |
| | 2 687 902 | ORCA | 37 | 48 | 0.9609 | 0.9954 | 0.9913 | 0.9866 |
| central | 14 081 816 | local greedy | – | – | – | – | – | – |
| | 33 866 826 | ORCA | 1116 | 343 | 0.9319 | 0.9943 | 0.9966 | 0.9909 |
| usa | 23 900 746 | local greedy | – | – | – | – | – | – |
| | 58 389 712 | ORCA | 1317 | 209 | 0.9424 | 0.9980 | 0.9954 | 0.9933 |

eu-2005 local greedy has a slight advantage in terms of quality indices while on in-2004 and uk-2002 ORCA yields higher values. On these two instances, ORCA outperforms the local greedy method in terms of *modularity*—especially on uk-2002 by a surprisingly large margin. Although the latter technique merges groups of nodes until no more improvement in *modularity* can be attained, it seems to fundamentally run past the innate clustering structure of this network, since ORCA identifies ten times as many clusters, with both a significantly higher *coverage* and *modularity*. At this point it is worth noting that the size of the local greedy clustering monotonously scales with the number of nodes (except for the smallest instance). This is paralleled by the predictable and artificial behavior of the *modularity* index, favoring a balance of (small) degree sums within clusters over *coverage*. This might be the reason for the algorithm's behavior on uk-2002, which seems better clustered much finer.

*Road Networks.* Unfortunately, walktrap and global greedy are way too slow for this input and the implementation of the local greedy algorithm crashes with a segmentation fault, for reasons unknown to us. Hence, we conclude that ORCA is currently the only graph clustering algorithm working on large instances of such kinds of inputs. As observable in Tab. 3, both running times and quality scores are excellent. All quality indices score a value higher than 0.94. We need less than 22 minutes to construct all levels of the hierarchy. Note that although usa is almost double the size of central, and ORCA clusters the former even coarser; running times are very similar. Together with the very high quality values, usa seems to be an easy instance.

## 5 Conclusion

We presented a fast graph clustering algorithm, called Orca. Unlike previous approaches, Orca works in a local sense: it iteratively contracts dense regions to supernodes which become the clustering of the current iteration step. It turns out that Orca clusters graphs up to a size of 530 million edges in less than 2 and half hours on standard server hardware. Only [10]—recently developed independently from us—can compete with Orca in terms of feasibility on huge networks. However, the scalability of our approach seems better since Orca can also cluster big road networks, where the latter approach fails. In terms of quality the two algorithms both compete with other state-of-the-art algorithms, and between them, no general assertion which one to prefer can be made. While [10] is faster, the obtained clustering is strongly dependent on the behavior of the index *modularity*, of which Orca is independent. For huge instances the choice ultimately depends on the application and whether artifacts specific to *modularity* are acceptable. Future work on Orca includes the adaptation of better rules for network hubs and a dynamization, which, given the clustering of some snapshot and a graph update, recomputes only affected parts of the clusterings.

## References

1. CiteSeer, Scientific Literature Digital Library, `http://citeseer.ist.psu.edu`
2. CAIDA - Cooperative Association for Internet Data Analysis, `http://www.caida.org`
3. DBLP - DataBase systems and Logic Programming (2007), `http://dblp.uni-trier.de`
4. Finding communities in large networks (2008), `http://findcommunities.googlepages.com`
5. IGRAPH - The igraph library (2008), `http://cneurocvs.rmki.kfki.hu/igraph`
6. Laboratory for Web Algorithmics (2008), `http://law.dsi.unimi.it`
7. An, Y., Janssen, J., Milios, E.E.: Characterizing and Mining the Citation Graph of the Computer Science Literature. Knowledge and Information Systems 6(6), 664–678 (2004)
8. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti-Spaccamela, A.: Complexity and Approximation - Combinatorial Optimization Problems and Their Approximability Properties, 2nd edn. Springer, Heidelberg (2002)
9. Barthélémy, M., Fortunato, S.: Resolution limit in community detection. Proc. of the National Academy of Science of the USA 104(1), 36–41 (2007)
10. Blondel, V., Guillaume, J.-L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. JSM: Theory and Experiment 2008(10) (2008)
11. Brandes, U., Delling, D., Gaertler, M., Görke, R., Höfer, M., Nikoloski, Z., Wagner, D.: On Modularity Clustering. IEEE TKDE 20(2), 172–188 (2008)
12. Brandes, U., Erlebach, T. (eds.): Network Analysis: Methodological Foundations. LNCS, vol. 3418. Springer, Heidelberg (2005)
13. Brandes, U., Gaertler, M., Wagner, D.: Engineering Graph Clustering: Models and Experimental Evaluation. ACM JEA 12(1.1), 1–26 (2007)
14. Castellano, C., Fortunato, S.: Community Structure in Graphs. To appear as chapter of Springer's Encyclopedia of Complexity and Systems Science (2008) arXiv:0712.2716v1

15. Clauset, A., Newman, M.E.J., Moore, C.: Finding community structure in very large networks. Physical Review E 70(066111) (2004)
16. Delling, D., Gaertler, M., Wagner, D.: Generating Significant Graph Clusterings. In: Proc. of ECCS 2006 (2006)
17. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering Route Planning Algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics of Large and Complex Networks. LNCS. Springer, Heidelberg (to appear, 2009)
18. Demetrescu, C., Goldberg, A.V., Johnson, D.S. (eds.): 9th DIMACS Implementation Challenge - Shortest Paths (November 2006)
19. Derényi, I., Palla, G., Vicsek, T.: Clique Percolation in Random Networks. Physical Review Letters 94 (2005)
20. Duch, J., Arenas, A.: Community Detection in Complex Networks using Extremal Optimization. Physical Review E 72(027104), 1–4 (2005)
21. Flake, G.W., Tarjan, R.E., Tsioutsiouliklis, K.: Graph Clustering and Minimum Cut Trees. Internet Mathematics 1(4), 385–408 (2003)
22. Gaertler, M., Görke, R., Wagner, D.: Significance-Driven Graph Clustering. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 11–26. Springer, Heidelberg (2007)
23. Gaertler, M., Görke, R., Wagner, D., Wagner, S.: How to Cluster Evolving Graphs. In: Proc. of ECCS 2006 (2006)
24. Kannan, R., Vempala, S., Vetta, A.: On Clusterings: Good, Bad, Spectral. Journal of the ACM 51(3), 497–515 (2004)
25. Lancichinetti, A., Fortunato, S., Kertéz, J.: Detecting the Overlapping and Hierarchical Community Structure of Complex Networks (2008), http://arxiv.org/abs/0802.1218
26. Newman, M.E.J.: Analysis of Weighted Networks. Physical Review E 70(056131), 1–9 (2004)
27. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Physical Review E 69(026113) (2004)
28. Pons, P., Latapy, M.: Computing Communities in Large Networks Using Random Walks. Journal of Graph Algorithms and Applications 10(2), 191–218 (2006)
29. Seidman, S.B.: Network Structure and Minimum Degree. Social Networks 5, 269–287 (1983)
30. van Dongen, S.M.: Graph Clustering by Flow Simulation. PhD thesis, University of Utrecht (2000)
31. Wagner, D., Wagner, F.: Between Min Cut and Graph Bisection. In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 744–750. Springer, Heidelberg (1993)
32. Zachary, W.W.: An Information Flow Model for Conflict and Fission in Small Groups. Journal of Anthropological Research 33, 452–473 (1977)
33. Wang, B., Phillips, J.M., Schreiber, R., Wilkinson, D., Mishra, N., Tarjan, R.: Spatial Scan Statistics for Graph Clustering. In: Proc. of SDM 2008, Atlanta (2008)

# Equiseparability on Terminal Wiener Index

Xiaotie Deng and Jie Zhang*

Department of Computer Science,
City University of Hong Kong, Hong Kong, P.R. China

**Abstract.** Wiener index as one of the oldest chemical index has been well studied. It has been extensive used in Computational Biology, Preliminary screening of drugs and Complex Network. Based on variable Wiener index, I.Gutman et al [6] introduced the concept of equiseparable pairs of trees and chemical trees, meanwhile they gave a rule on how to construct such equiseparable pairs. D.Vukičević and I.Gutman [8] proved almost all trees and chemical trees have equiseparable mates, which is a disadvantageous property of many molecular-structure graph-based descriptors. Recently, I.Gutman et al [9] proposed the concept of *Terminal Wiener Index*, which equals to the summation of distance between all pairs of pendent vertices of trees. Following this line, we explore the properties of terminal Wiener index, and show the fact that there still exist pairs of trees and chemical trees which can not be distinguished by it, therefore we give some general methods to construct equiseparable pairs and compare the methods in the case of Wiener index. More specifically, we show that terminal Wiener index is degenerative to some extent.

**Keywords:** Equiseparability, terminal Wiener index, chemical tree.

## 1 Introduction

There are many chemical indices proposed as molecular-structure descriptors until now, one of the oldest and well studied chemical index is the *Wiener index* which was given by H.Wiener [1] in 1947. It can be expressed as:

$$W(G) = \sum_{1 \le i < j \le n} d(v_i, v_j) \tag{1}$$

where $d(v_i, v_j)$ is the distance between vertices $v_i$ and $v_j$ in a graph $G$, the summation goes over all pairs of vertices of the given graph. For trees, H.Wiener got a very useful formula to calculate the Wiener index:

$$W(T) = \sum_{e \in T} n_1(e|T) \cdot n_2(e|T) \tag{2}$$

* Corresponding author, zhangjie3@student.cityu.edu.hk

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 166–174, 2009.

where $n_1(e|T)$ and $n_2(e|T)$ are the number of vertices of $T$, lying on the two sides of $e$. The summation on the right-hand side of the equation goes over all edges of the tree $T$. Obviously, if the tree $T$ has $n$ vertices, then for all of its edges,

$$n_1(e|T) + n_2(e|T) = n$$

Based on Wiener index, a general index called *Variable Wiener Index* has been proposed [3,4]:

$$W_\lambda(T) = \sum_{e \in T} [n_1(e|T) \cdot n_2(e|T)]^\lambda \tag{3}$$

where $\lambda$ is an adjustable parameter.

**Definition 1.** *([6]). Assuming $n_1(e|T) \leq n_2(e|T)$, two trees $T'$ and $T''$ of order $n$ are said to be* equiseparable *if their edges $e'_1, e'_2, \cdots, e'_{n-1}$ and $e''_1, e''_2, \cdots, e''_{n-1}$ can be labeled so that the equality $n_1(e'_i|T') = n_1(e''_i|T'')$ holds for all $i = 1, 2, \cdots, n-1$.*

Wiener index has been extensive used in Computational Biology, Preliminary screening of drugs and Complex Network. For example, it is a measurement of average distance in network [10,11]. In the design of economical networks, spanning trees of connected graph with smallest Wiener index are very important in practice [12]. In Chemistry, Wiener index measures the van der Waals surface area of an alkane molecule, which explains the correlations found between $W$ and a great variety of physico-chemical properties of alkanes [5]. But if two or more chemical trees are equiseparable, then those compounds will have similar physico-chemical properties which can not be distinguished by Wiener index. It is a main drawback of many chemical index structure-descriptors.

I.Gutman et al [6] pointed out that there does exist pairs of isomeric alkanes whose variable Wiener index coincide for all values of the parameter $\lambda$. Some former studies [6,7] showed how to construct such equiseparable chemical trees. As another point of view, D.Vukičević et al [8] gave a proof on almost all trees and chemical trees[1] have equiseparable mates.

In [13], E.A.Smolenskii et al made use of terminal distance matrices to encode molecular structures. The proposed reduced vector is less degenerative than some other molecular codes. Based on those applications, I.Gutman et al [9] proposed the concept of *Terminal Wiener Index*, which equals to the summation of distance between all pairs of pendent vertices[2] of trees, i.e.

$$TW(T) = \sum_{1 \leq i < j \leq k} d(v_i, v_j) \tag{4}$$

where $v_i$ and $v_j$ are pendent vertices of tree $T$, $d(v_i, v_j)$ is the distance between them, and the sum goes over all pairs of such pendent vertices.

[1] A tree is a chemical tree if its maximum degree is at most 4.

[2] In this paper, pendent vertices means leaves of the tree.

Similar to the proof of (2), I.Gutman got another way to calculate the terminal Wiener index.

$$TW(T) = \sum_{e \in T} p_1(e|T) \cdot p_2(e|T) \tag{5}$$

where $p_1(e|T)$ and $p_2(e|T)$ are the number of pendent vertices of $T$, lying on the two sides of $e$, and the summation embraces all the $n-1$ edges of $T$. We will use $p_1(e)$, $p_2(e)$ instead of $p_1(e|T)$, $p_2(e|T)$ when there is no confusion.

Similar to Wiener index, we define the variable terminal Wiener index so that it can have more molecular-structure descriptive power.

**Definition 2.** *Variable terminal Wiener index is defined as follows:*

$$TW_\lambda(T) = \sum_{e \in T} [p_1(e) \cdot p_2(e)]^\lambda \tag{6}$$

*where $\lambda$ is an adjustable parameter.*

Unfortunately, with this more powerful index, there still exit pairs of trees and chemical trees whose variable terminal Wiener index coincide for all values of the parameter $\lambda$. We can see it from the example in Figure 1, where $T_1$ and $T_2$ have same variable terminal Wiener index, $5 \cdot 2^\lambda$.

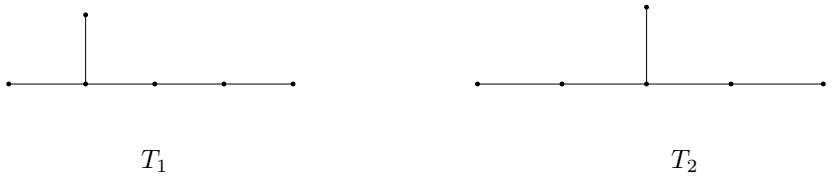



**Fig. 1.**

Based on this fact, we define the equiseparability w.r.t. terminal Wiener index.

**Definition 3.** *Assuming $p_1(e) \leq p_2(e)$, two trees $T'$ and $T''$ of order $n$ are said to be* equiseparable *w.r.t. terminal Wiener index if their edges $e'_1, e'_2, \cdots, e'_{n-1}$ and $e''_1, e''_2, \cdots, e''_{n-1}$ can be labeled so that the equality $p_1(e'_i|T') = p_1(e''_i|T'')$ holds for all $i = 1, 2, \cdots, n-1$.*

In Section 2, we explore different rules for constructing equiseparable trees w.r.t Wiener index and terminal Wiener index. In Section 3, we give a formal proof of the fact that terminal wiener index has the degenerative phenomenon as the wiener index. We conclude in section 4.

## 2 Rules for Constructing Equiseparable Trees with Respect to Terminal Wiener Index

First, we show that the methods of constructing equiseparable trees w.r.t Wiener index in [6,7] can be extended to construct equiseparable trees w.r.t terminal Wiener index.

In [6], I.Gutman got some rules for constructing equiseparable chemical trees w.r.t. Wiener index. But they are in fact special cases of the method obtained in [7], which can be stated as:

**Theorem 1.** *([7]). Let $T$, $X$ and $Y$ be arbitrary trees, each with more than two vertices. Let the tree $T_1$ be obtained from $T$ by identifying the vertices $u$ and $s$, and by identifying the vertices $v$ and $t$. Let $T_2$ be obtained from $T$ by identifying the vertices $u$ and $t$, and by identifying the vertices $v$ and $s$. Then if $X$ and $Y$ have equal number of vertices, the trees $T_1$ and $T_2$ are equiseparable. See Fig.2.*

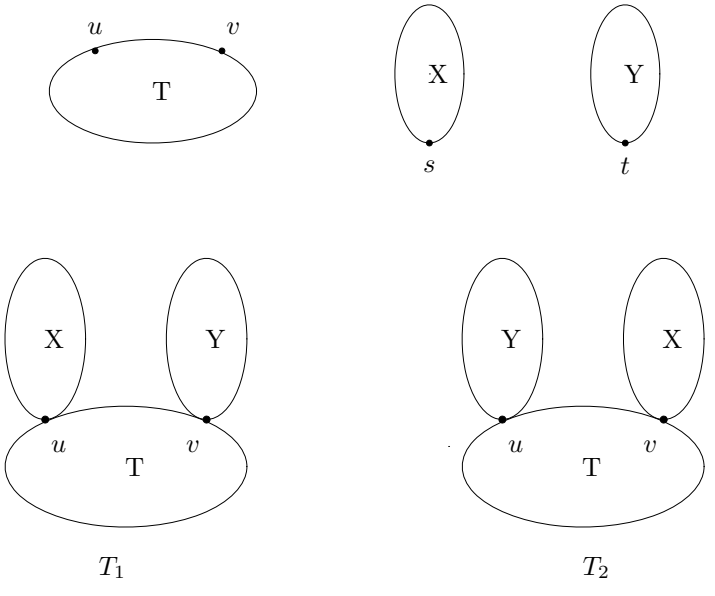



**Fig. 2.**

If we revise the condition felicitously, then Theorem 1 can be extended to construct equiseparable trees w.r.t. terminal Wiener index.

**Theorem 2.** *Let trees $T_1$ and $T_2$ be constructed the same as they are in Fig.2. If $p_x - p_s = p_y - p_t$, then the trees $T_1$ and $T_2$ are equiseparable w.r.t. terminal Wiener index. $p_x$ and $p_y$ denote the number of pendent vertices of fragments $X$ and $Y$, respectively. $p_s$ is equal to 1 if $s$ is a pendent vertex of $X$, otherwise it is equal to 0. $p_t$ is defined similar to $p_s$.*

*Proof.* We prove it by classifying the edges of $T_1$ and $T_2$ into four types and each type of edges satisfy Definition 3.

(1) For edges belonging to $T$, lying on the same side of $u$ and $v$. For example, edge $e'$ of $T_1$ and $e''$ of $T_2$, both lying on the left of $u$. We have $p_1(e'|T_1) = p_1(e''|T_2) = p_1(e|T)$, $p_2(e'|T_1) = p_2(e''|T_2) = p_2(e|T) + p_x + p_y - k$, where $k$ is

a constant which equals the number of pendent vertices among $\{u, v, s, t\}$. So this type of edges can be labeled so that $p_1(e'|T_1) = p_1(e''|T_2)$ always holds. The same applies to edges lying on the right of $v$.

(2) For edges belonging to $X$. Obviously there is a bijection between the edges of fragment $X$ of $T_1$ and the edges of fragment $X$ of $T_2$, so this type of edges can also be labeled so that $p_1(e'|T_1) = p_1(e''|T_2)$ always holds.

(3) For edges belonging to $Y$. It is the same as case (2).

(4) For edges belonging to $T$, lying between the vertices $u$ and $v$. According to whether vertices $u, v, s, t$ are pendent vertices of their corresponding fragments, this case can be divided into $2^4 = 16$ subcases. We only check three typical subcases here, others can be proved similarly.

(4.1) None of them is pendent vertex.

Then we have $p_1(e'|T_1) = p_1(e|T) + p_x$, $p_2(e'|T_1) = p_2(e|T) + p_y$ and $p_1(e''|T_2) = p_1(e|T) + p_y$, $p_2(e''|T_2) = p_2(e|T) + p_x$. Combined with $p_s = 0$ and $p_t = 0$, we get that the equality $p_x - p_s = p_y - p_t$ implies the edges lying between $u$ and $v$ satisfy Definition 3.

(4.2) One of them is pendent vertex, for example, $s$ is a pendent vertex of $X$.

Then we have $p_1(e'|T_1) = p_1(e|T) + p_x - 1$, $p_2(e'|T_1) = p_2(e|T) + p_y$ and $p_1(e''|T_2) = p_1(e|T) + p_y$, $p_2(e''|T_2) = p_2(e|T) + p_x - 1$. Combined with $p_s = 1$ and $p_t = 0$, we get that the equality $p_x - p_s = p_y - p_t$ implies the edges lying between $u$ and $v$ satisfy Definition 3.

(4.3) Two of them are pendent vertices, for example, $s$ is a pendent vertex of $X$ while $v$ is a pendent vertex of $T$.

Then we have $p_1(e'|T_1) = p_1(e|T) + p_x - 1$, $p_2(e'|T_1) = p_2(e|T) + p_y - 1$ and $p_1(e''|T_2) = p_1(e|T) + p_y$, $p_2(e''|T_2) = p_2(e|T) + p_x - 2$. Combined with $p_s = 1$ and $p_t = 0$, we get that the equality $p_x - p_s = p_y - p_t$ implies the edges lying between $u$ and $v$ satisfy Definition 3.

After checking all 16 subcases we get that edges lying between $u$ and $v$ can be labeled so that $p_1(e'|T_1) = p_1(e''|T_2)$ always holds.

Aggregating these four cases, we can see that if $p_x - p_s = p_y - p_t$, then $p_1(e'_i|T') = p_1(e''_i|T'')$ holds for all $i = 1, 2, \cdots, n-1$, which implies that trees $T_1$ and $T_2$ are equiseparable.

On the other hand, trees are equiseparable w.r.t. Wiener index does not imply they are equiseparable w.r.t. terminal Wiener index, since that the terminal Wiener index is the sum of the distance between all pairs of pendent vertices but not pairs of vertices. For example, the trees $T_1$ and $T_2$ in Fig.3 are equiseparable w.r.t. Wiener index but not equiseparable w.r.t terminal Wiener index. So, it is worth to find some general rules for constructing equiseparable trees w.r.t terminal Wiener index only.

The following theorem and corollary are rules to construct equiseparable trees w.r.t terminal Wiener index but not Wiener index.

**Theorem 3.** *Let $Z$ be an arbitrary tree, $u$ is a vertex of $Z$, tree $T_1$ is obtained by identifying the vertices $u$ and $i$, $T_2$ is obtained by identifying the vertices $u$ and $j$. If $X$ and $Y$ have equal number of pendent vertices, then the trees $T_1$ and $T_2$ are equiseparable. See Fig.4.*

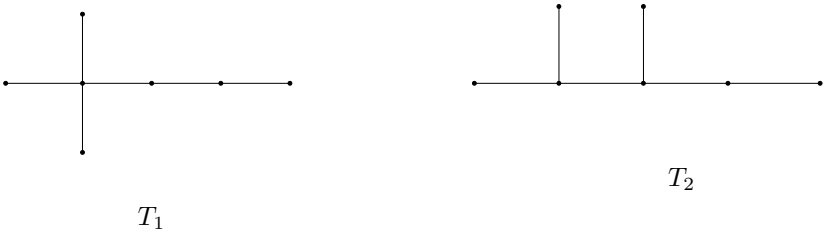



**Fig. 3.**

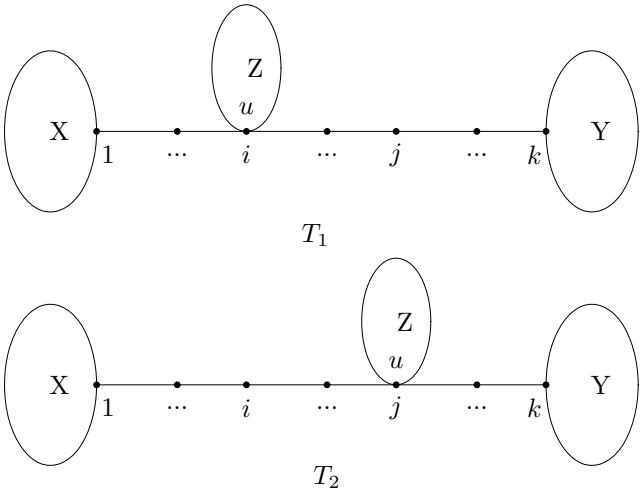



**Fig. 4.**

*Proof.* Suppose the number of pendent vertices of fragments $X$, $Y$ and $Z$ are $p_x$, $p_y$ and $p_z$, respectively. If $u$ is a pendent vertex of $Z$ then $k$ is equal to 1, otherwise $k$ is equal to 0.

For each pair of edges $e'$ of $T_1$ and $e''$ of $T_2$ which lying on the left of vertex $i$, the number of pendent vertices sit on the two sides of these edges are $p_x$ and $p_y + p_z - k$, respectively. So the edges lying on the left of vertex $i$ can be labeled so that $p_1(e'_i|T_1) = p_1(e''_i|T_2)$ always holds.

The same applies to the edges which lying on the right of vertex $j$ and belonging to fragment $Z$, so we only need to consider the edges lying between vertices $i$ and $j$.

For the edge $e'$ of $T_1$ which lying between $i$ and $j$, the number of pendent vertices sit on the two sides of $e'$ are $p_x + p_z - k$ and $p_y$; For the edge $e''$ of $T_2$ which lying between $i$ and $j$, the number of pendent vertices sit on the two sides of $e''$ are $p_x$ and $p_y + p_z - k$.

Since $p_x = p_y$, we can label the edges $e'_1, e'_2, \cdots, e'_{n-1}$ of $T_1$ and $e''_1, e''_2, \cdots, e''_{n-1}$ of $T_2$, so that the equality $p_1(e'_i|T_1) = p_1(e''_i|T_2)$ holds for all $i = 1, 2, \cdots, n-1$. Therefore $T_1$ and $T_2$ are equiseparable w.r.t. terminal Wiener index.

Note that since $TW(T)$ only depends on the distance between pairs of pendent vertices, the position of fragment $Z$ can be arbitrary lying on the path from 1 to $k$. But for Wiener index, things are different. Fragments $X$ and $Y$ having equal number of vertices is not sufficient for equiseparability when fragment $Z$ moving arbitrary between vertex $i$ and $j$, we can see it from two trees in Fig.1.

Theorem 3 can be extended to the circumstances when they are more than one fragment on the path from 1 to $k$.

**Corollary 1.** *If the fragments $X$ and $Y$ have equal number of pendent vertices, the fragments $Z_1, Z_2, \cdots, Z_t$ moving without changing the distance between them, then the resulting two (chemical) trees are equiseparable w.r.t. terminal Wiener index. See Fig.5 for illustration.*

The proof of Corollary 1 is omitted here.

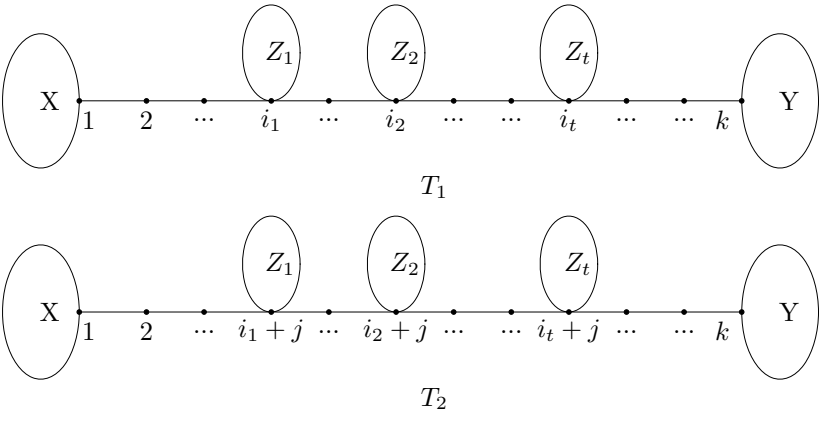



**Fig. 5.**

## 3 Degeneracy of Terminal Wiener Index

D.Vukičević and I.Gutman [8] developed a powerful technic to prove almost all trees and chemical trees have equiseparable mates w.r.t. Wiener index, the proof of the chemical trees case is omitted since it is more complicated than the case of trees. In this section, we show that the terminal Wiener index is degenerative by proving almost all chemical trees have equiseparable mates.

Obviously, a necessary condition for two trees to be equiseparable is that they have same number of pendent vertices. Let $T$ be an $n$-vertex tree with $k$ pendent

vertices, $e_1, e_2, \cdots, e_{n-1}$ are its edges, $k = 2, 3, \cdots, n-1$. Assume that for each edge $e$ of tree $T$, $p_1(e|T) \leq p_2(e|T)$. Let $t_i(T)$ denote the numbers $p_1(e_i|T)$ be equal to $i$, $i = 1, 2, \cdots, n-1$, i.e., $t_i(T)$ is the number of edges that, when cutting off the edges $e_i$, lying on the side of $e_i$ which has lesser pendent vertices. We can see that the sequence $(t_1(T), t_2(T), \cdots, t_{\lfloor k/2 \rfloor}(T))$ does not dependent on the labeling of the edges of $T$.

**Theorem 4.** *The terminal Wiener index is degenerative in the sense that almost all chemical trees have equiseparable mates.*

*Proof.* Let $U_n$ and $CU_n$ be the number of trees and chemical trees of order $n$ having no equiseparable mates, $T_n$ and $CT_n$ be the number of distinct $n$-vertices trees and chemical trees. From the definition of equiseparable trees, we can see that two trees are equiseparable if and only if their corresponding sequence coincide. Then what we need to prove is

$$\lim_{n\to\infty} \frac{CU_n}{CT_n} = 0$$

For $CU_n$, it is obviously that $CU_n \leq U_n$.

Let $k$ be fixed, consider tree $T$ of order $n$ with $k$ pendent vertices. Since $T$ has $n-1$ edges, which means the sum of the sequence $(t_1(T), t_2(T), \cdots, t_{\lfloor k/2 \rfloor}(T))$ equals $n-1$, the number of such trees is equal to

$$\binom{\lfloor k/2 \rfloor + n - 2}{\lfloor k/2 \rfloor - 1}$$

with $k \in \{2, 3, \cdots, n-1\}$, we have

$$U_n = \sum_{k=2}^{n-1} \binom{\lfloor k/2 \rfloor + n - 2}{\lfloor k/2 \rfloor - 1} = \begin{cases} \sum\limits_{k=2}^{(n-1)/2} \binom{k+n-2}{k-1} & \text{for } n \text{ is odd;} \\ \sum\limits_{k=2}^{n/2-1} \binom{k+n-2}{k-1} & \text{for } n \text{ is even} \end{cases}$$

by using simple combinatorial identity, we get

$$CU_n \leq \begin{cases} \binom{n-1+\frac{n-1}{2}}{\frac{n-1}{2}-1} - 1 & \text{for } n \text{ is odd;} \\ \binom{n-1+\frac{n}{2}-1}{\frac{n}{2}-2} - 1 & \text{for } n \text{ is even} \end{cases}$$

For $CT_n$, R.Otter [2] obtained an asymptotic value, i.e., for $3 \leq m < \infty$

$$T_n \sim \frac{\alpha^{2.5}\beta^3 a_{m-3}}{4\sqrt{\pi}} \frac{\alpha^{-n}}{n^{2.5}}$$

where $m$ is the maximum degree in $T_n$, $\alpha, \beta$ and $a$ are constant for any fixed $m$. Specifically, for $m = 4$, i.e., for chemical trees,

$$CT_n \sim k \cdot \frac{\alpha^{-n}}{n^{2.5}}, \;\; k = 0.656319, \;\; \alpha = 0.3551817$$

Obviously, $CT_n$ is exponential on $n$. Then, we have

$$\lim_{n \to \infty} \frac{CU_n}{CT_n} = 0.$$

## 4 Conclusion

We demonstrated some rules to construct equiseparable trees w.r.t. terminal Wiener index and compared them in the case of Wiener index. We also showed that terminal Wiener index is degenerative as Wiener index. An interesting problem raised naturally is that the difference in constructing equiseparable trees, i.e., what is the sufficient condition for trees to be equiseparable w.r.t Wiener index but not equiseparable w.r.t terminal Wiener index, and vice versa. This may be difficult to solve.

## References

1. Wiener, H.: Structural determination of paraffin boiling points, J. Am. Chem. Soc. 69, 17–20 (1947)
2. Otter, R.: The number of trees. The Annals of Mathematics, Second Series 49(3), 583–599 (1948)
3. Lučić, B., Miličević, A., Nikolić, S., Trinajstić, N.: On variable Wiener index. Indian J. Chem. 42A, 1279–1282 (2003)
4. Gutman, I., Vukičević, D., Žerovnik, J.: A class of modified Wiener indices. Croat. Chem. Acta 77, 103–109 (2004)
5. Gutman, I., Körtvélyesi, T.: Wiener indices and molecular surfaces. Z. Naturforsch. 50a, 669–671 (1995)
6. Gutman, I., Arsić, B., Furtula, B.: Equiseparable chemical trees. J. Serb. Chem. Soc. 68, 549–555 (2003)
7. Gutman, I., Furtula, B., Vukičević, D., Arsić, B.: Equiseparable molecules and molecular graphs. Indian J. Chem. 43, 7–10 (2004)
8. Vukičević, D., Gutman, I.: Almost all trees and chemical trees have equiseparable mates. J. Comput. Chem. Jpn. 3(3), 109–112 (2004)
9. Gutman, I., Furtula, B., Pettrović, M.: Terminal Wiener index. J. Math. Chem., doi: 10.1007/s10910-008-9476-2
10. Wuchtya, S., Stadler, P.F.: Centers of complex networks. Journal of Theoretical Biology 223, 45–53 (2003)
11. Jing, Z., Hong, Y., Jianhua, L., Cao, Z.W., Yixue, L.: Complex networks theory for analyzing metabolic networks. Chinese Science Bulletin., doi: 10.1007/s11434-006-2015-2
12. Dobrynin, A.A., Entringer, R., Gutman, I.: Wiener Index of Trees: Theory and Applications. Acta Applicandae Mathematicae 66, 211–249 (2001)
13. Smolenskii, E.A., Shuvalova, E.V., Maslova, L.K., Chuvaeva, I.V., Molchanova, M.S.: Reduced matrix of topological distances with a minimum number of independent parameters: distance vectors and molecular codes. J. Math. Chem., doi: 10.1007/s10910-008-9393-4

# Effective Tour Searching for TSP by Contraction of Pseudo Backbone Edges

Changxing Dong, Gerold Jäger, Dirk Richter, and Paul Molitor

Computer Science Institute
University of Halle-Wittenberg
D-06120 Halle (Saale), Germany
{dong,jaegerg,richterd,molitor}@informatik.uni-halle.de

**Abstract.** We introduce a reduction technique for the well-known TSP. The basic idea of the approach consists of transforming a TSP instance to another one with smaller size by contracting pseudo backbone edges computed in a preprocessing step, where pseudo backbone edges are edges which are likely to be in an optimal tour. A tour of the small instance can be re-transformed to a tour of the original instance. We experimentally investigated TSP benchmark instances by our reduction technique combined with the currently leading TSP heuristic of Helsgaun. The results strongly demonstrate the effectivity of this reduction technique: for the six VLSI instances *xvb13584*, *pjh17845*, *fnc19402*, *ido21215*, *boa28924*, and *fht47608* we could set world records, i.e., find better tours than the best tours known so far. The success of this approach is mainly due to the effective reduction of the problem size so that we can search the more important tour subspace more intensively.

## 1 Introduction

The traveling salesman problem (TSP) is a well known and intensively studied problem [1,11,16,29] which plays a very important role in combinatorial optimization. It can be simply stated as follows. Given a set of cities and the distances between each pair of them, find a shortest cycle visiting each city exactly once. Formally, for a complete, undirected and weighted graph with $n$ vertices, find a shortest Hamiltonian cycle. The size of the problem instance is denoted by $n$. If the distance between two cities does not depend on the direction, the problem is called *symmetric TSP* (STSP). In this paper we consider only the STSP. Although TSP is easy to understand, it is hard to solve, even $\mathcal{NP}$-hard. The hardness of a TSP instance depends on the graph structure, but also very strongly on the size $n$. The latter dependence comes from the immense search space, i.e., the number of tours is $(n-1)!/2$ for the STSP. Therefore it is very hard to effectively find good tours for very large problem instances. We distinguish two classes of algorithms for the STSP, namely heuristics and exact algorithms. For the exact algorithms the program package *Concorde* [1,33], which combines techniques of linear programming and branch-and-cut, is the

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 175–187, 2009.

currently leading code. Concorde has exactly solved many benchmark instances, the largest one has even size 85,900 [2]. On the other hand, in the field of STSP heuristics Helsgaun's code [12,13,34] (LKH), which is an effective implementation of the Lin-Kernighan heuristic [17], is one of the best packages. This code found the currently best tours for the most not exactly solved TSP benchmark instances [27,28,30,31] including the famous *World TSP* instance [32].

As the search space for the STSP is huge, we can only traverse a tiny part of it in reasonable time. An interesting observation is that tours with good quality are likely to share many edges, which we call *pseudo backbone edges* [26]. In contrast, *backbone edges* are edges which are contained in each optimal tour [15,19,25]. In this paper we try to improve TSP heuristics by exploiting this observation. Our main idea is to considerably reduce the size of the TSP instance by contraction of edges, which have high probability to appear in an optimal tour. Our approach works as follows. First, we find a number of tours with good quality. Second, we draw out from these tours the pseudo backbone edges. Third, we compute the maximal paths from the pseudo backbone edges, and contract each maximal path to an edge. In this way, we create a new TSP instance with smaller size. This reduced instance can be attacked more effectively and combined with each possible TSP heuristic. In our experimental investigation, we used Helsgaun's implementation of the Lin-Kernighan-Heuristic [34]. This improved version contains many new efficient features such as general $k$-OPT moves, different types of partitioning, tour merging [3], iterative partial transcription [18] and backbone-guided search [24,26].

The concept of edge fixing *without* backbone contraction was already used by Lin, Kernighan [17]. Fischer and Merz [6] extended this idea to size reduction, but paid more attention to the different reduction heuristics and the enhancement to evolutionary algorithms than in our approach. Further related ideas are Cook and Seymour's tour merging algorithm [3], which merges a given set of starting tours to get an improved tour, and compression in LKH, which is similar to contraction, but works for subproblems of partitions. All these approaches are in some sense a heuristic parallel to FPT kernelization (for an overview over the theory of parameterized complexity see [4,7,14,20]), albeit TSP is not a problem of this class. One special concept which is applied during FPT kernelization is data reduction and iterative compression which reduce a hard instance to a smaller equivalent problem kernel [10]. Note that our approach is closely related to this kernelization technique.

Our experiments led to impressive results, e.g., we found record tours for six VLSI instances [31]: *xvb13584*, *pjh17845*, *fnc19402*, *ido21515*, *boa28924*, and *fht47608*, where some of the previous record tours had not been improved for several years.

The paper is structured as follows. Our pseudo backbone contraction approach is described in Section 2 and the experimental results are presented in Section 3. Finally, we give conclusions and suggestions for future work in Section 4.

## 2 Pseudo Backbone Contraction

### 2.1 The Phases of Pseudo Backbone Contraction

Let a TSP instance be given as a complete graph $G = (V, E)$, where $V$ is the set of vertices, and $E$ the set of edges. Our approach undergoes the following phases:

1. Find a set $\Omega$ of good tours for the given instance. We call these tours *starting tours*.
2. Collect the pseudo backbone edges, i.e., the edges that appear in all starting tours. Formally, we have a set $B := \{e \in E : e \in \cap_{T_i \in \Omega} T_i\}$. Let $V_B := \{v \in V : v \in e, e \in B\}$ and $\overline{V}_B := V \setminus V_B$.
3. Construct all maximal paths consisting only of pseudo backbone edges contained in $B$, where a path is called *maximal* with property $\mathcal{P}$, if it meets $\mathcal{P}$ and cannot be extended by an edge without violating $\mathcal{P}$. Each maximal path is contracted to an edge, the end points of which are that of the path. We call such an edge a *p-edge* (path edge). The $p$-edge of a path with only one pseudo backbone edge is just the edge itself. We denote the set of all the end points of the $p$-edges by $V_C$.
4. Construct a new TSP instance $H = (W, F)$, where $W = \overline{V}_B \cup V_C, F = W \times W$.
5. Find a good tour for the new TSP instance $H$ subject to the condition that all $p$-edges must be in the tour, i.e., the $p$-edges are *fixed*. Note that the length of a $p$-edge can be chosen arbitrarily, as it is fixed.
6. Obtain a tour for the original instance by re-transforming the tour of the new instance.

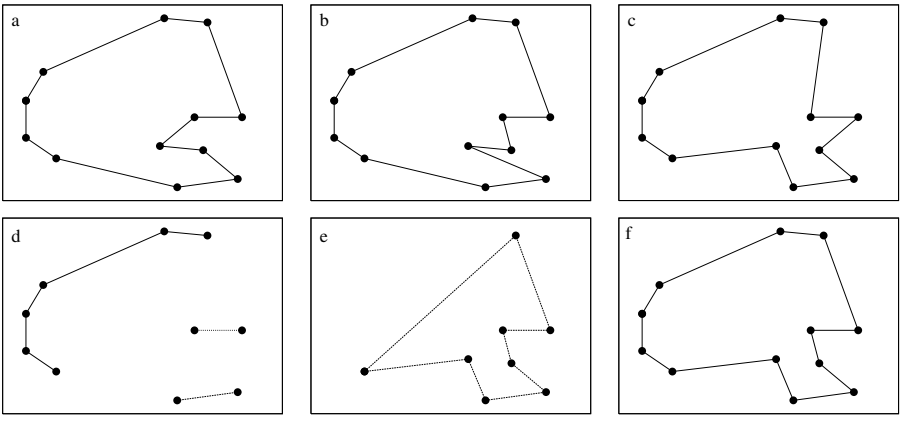



**Fig. 1.** Demonstration of the pseudo backbone edge contraction

### 2.2 Illustration of the Approach

Our approach is illustrated in Fig. 1 by a small TSP instance. This instance has 12 points in a two dimensional Euclidean plane, which means the distances between the points are given by the Euclidean metric. By the three starting tours in Fig. 1a, 1b, and 1c, we receive the pseudo backbone edges (Fig. 1d). From the maximal paths consisting only of pseudo backbone edges, only one has a length greater than 1. Only this path contributes to the size reduction. This pseudo backbone edge is contracted to a $p$-edge. After contracting, we receive a new instance with 8 points (Fig. 1e). The three $p$-edges are fixed while searching tours for the new instance. In Fig. 1e an optimal tour for the new instance is shown. After expanding the $p$-edges in the tour of the new instance, we receive a tour for the original instance (Fig. 1f). For this instance the final tour is optimal.

### 2.3 Why Contracting?

An alternative to this approach is fixing without backbone contraction. In this case all the pseudo backbone edges are forced to appear in every tour. Thus the search space is considerably cut, although the size of the problem is not reduced. In contrast, the main feature of contraction is the reduction of the problem size. This reduction has great influence to the effectivity of tour searching. The reason is that all edges incident to the vertices in $V \setminus W$ do not appear in the new instance, whilst by only fixing many of them should be traversed – without any chance to find better tours. Note that also the time used for the edge length computations becomes smaller by the reduction of the problem size.

### 2.4 Solving the Reduced TSP-Instance

Local search is an improvement heuristic, which means it steadily improves the best tour found. Certainly we need an initial tour constructed by other heuristics, e.g., the nearest neighbor heuristic or a random heuristic. Local search transforms a tour to another one by exchanging $k$ tour edges with $k$ non-tour edges. This is called a $k$-OPT move. To improve a tour, the sum of the length of the $k$ tour edges must be larger than the sum of the length of the $k$ non-tour edges. A tour which cannot be improved by $k$-OPT moves for $k \leq r$ is called *$r$-optimal*. Note that an $n$-optimal tour is optimal. As finding an $r$-optimal tour has complexity $\mathcal{O}(n^r)$, we are forced to constrain the search space such that we obtain good tours in tolerable time. LKH uses several parameters for this purpose. The most important ones beside the value of $r$ are the maximal number $s$ of edges incident to each vertex to be considered while searching tours and the number $t$ of independent runs. In this context we define a tour as *approximated $r$-optimal*, if it is $r$-optimal for the given maximal number $s$ of edges incident to each vertex. The larger $r$, $s$, and $t$ are, the larger is the search space. With large search space we can find better tours against cost of time. The parameter values that make the search space large are called strong parameter values, otherwise weak.

In our approach, LKH was used for searching both the starting tours of the original instances and tours for the reduced instances. To collect the starting tours, we could not afford strong parameter values because of the time limit. Therefore, the search for the starting tours was associated with relative weak parameter values and the search for the reduced instances with stronger ones. More exactly, for the reduced instances we used $r = 8, 10$, $s = 5, 6$ and $t = 10, 20$, while the standard parameter values are $r = 5$, $s = 5$, and $t = 10$. It is just the reduction by contraction that makes the choice of strong parameter values possible. Note that LKH with standard parameter values finds optimal solutions frequently for small instances, e.g., most instances from TSPLIB. Since the reduced instances in this work have sizes not larger than 18,242 and stronger searching parameter values are applied to them, we have not tried to solve them with the program package Concorde [33]. In addition, the reduced instances have fixed edges to be treated specifically which is not implemented in Concorde.

### 2.5 Starting Tours

As starting tours we used tours received by previous experiments with many different variants of LKH. The main distinctions between these variants are that the candidate edges are chosen by different criteria, which are mostly tolerances (for an overview over the theory of tolerances see [8,9]). Note that the number of starting tours we used is different for each instance. In particular, it depends on the size of the TSP instance.

## 3 Experimental Results

We ran the programs on several computers to attack different instances simultaneously, where for solving the reduced instances we used LKH-2.0 and for computing the starting tours the older version LKH-1.3. The time limit for the reduced instances was set to two weeks. Mostly, we used the default parameter values of LKH, whereas we varied only the already mentioned parameters $r, s, t$. The main goal of this work is to find tours as good as possible, therefore we pay less attention to the running time. Furthermore the running times strongly depend on the instances, i.e, smaller size does not necessarily mean less time. Unfortunately our running times cannot be compared with those of previous experiments listed at [29], as in most cases no times were given there. To get a feeling for the running times we also applied LKH to several large original TSP instances with the same parameter values $r$, $s$, $t$ as applied to the reduced instances. After more than one month we could not find any better tours for the original instances.

All experimental information such as starting tours, parameter values of LKH, running times and machines are available at [35].

### 3.1 Investigated Instances

All the TSP instances investigated in this work are shown in Table 1, where their sizes can be seen in their names. These instances come from three different

**Table 1.** TSP instances investigated and the best tour lengths found before and by us

| | Instance | Src | Best | Ours | | Instance | Src | Best | Ours |
|---|---|---|---|---|---|---|---|---|---|
| 1 | dsj1000 | 1 | 18659688 | = | 27 | ei8246 | 2 | 206171 | = |
| 2 | pr1002 | 1 | 259045 | = | 28 | dga9698 | 3 | 27724 | = |
| 3 | u1060 | 1 | 224094 | = | 29 | kz9976 | 2 | 1061881 | = |
| 4 | pcb1173 | 1 | 56892 | = | 30 | xmc10150 | 3 | 28387 | = |
| 5 | rl1304 | 1 | 252948 | = | 31 | fi10639 | 2 | 520527 | = |
| 6 | rl1323 | 1 | 270199 | = | **32** | **xvb13584** | **3** | **37084** | **37083** |
| 7 | nrw1379 | 1 | 56638 | = | 33 | brd14051 | 1 | 469385 | 469392 |
| 8 | fl1400 | 1 | 20127 | 20188 | 34 | xrb14233 | 3 | 45462 | 45464 |
| 9 | u1432 | 1 | 152970 | = | 35 | xia16928 | 3 | 52850 | = |
| 10 | u1817 | 1 | 57201 | = | **36** | **pjh17845** | **3** | **48094** | **48093** |
| 11 | rl1889 | 1 | 316536 | = | 37 | frh19289 | 3 | 55798 | = |
| 12 | u2152 | 1 | 64253 | = | **38** | **fnc19402** | **3** | **59288** | **59287** |
| 13 | xqc2175 | 3 | 6830 | = | **39** | **ido21215** | **3** | **63519** | **63518** |
| 14 | pr2392 | 1 | 378032 | = | 40 | fma21553 | 3 | 66527 | = |
| 15 | pcb3038 | 1 | 137694 | = | 41 | lsb22777 | 3 | 60977 | 60983 |
| 16 | pia3056 | 3 | 8258 | = | 42 | xrh24104 | 3 | 69294 | = |
| 17 | xqe3891 | 3 | 11995 | = | 43 | irx28268 | 3 | 72607 | = |
| 18 | bgb4355 | 3 | 12723 | = | 44 | icx28698 | 3 | 78090 | 78095 |
| 19 | rl5915 | 1 | 565530 | = | **45** | **boa28924** | **3** | **79624** | **79623** |
| 20 | rl5934 | 1 | 556045 | = | 46 | pbh30440 | 3 | 88313 | 88314 |
| 21 | tz6117 | 2 | 394718 | = | 47 | xib32892 | 3 | 96767 | 96780 |
| 22 | xsc6880 | 3 | 21535 | = | 48 | fry33203 | 3 | 97240 | 97242 |
| 23 | bnd7168 | 3 | 21834 | = | 49 | ics39603 | 3 | 106821 | 106826 |
| 24 | lap7454 | 3 | 19535 | = | **50** | **fht47608** | **3** | **125124** | **125119** |
| 25 | ym7663 | 2 | 238314 | = | 51 | fna52057 | 3 | 147802 | 147818 |
| 26 | ida8197 | 3 | 22338 | = | | | | | |

areas: TSPLIB instances [21,28] (**Src 1**), national instances [30] (**Src 2**) and VLSI instances [31] (**Src 3**). In column **Best** the length of the best tour found so far is given for each instance. The length of our best tour can be seen in column **Ours**. If for an instance we found a tour with the same length as that in column **Best**, an equal sign "=" is used in column **Ours**. The six VLSI instances *xvb13584*, *pjh17845*, *fnc19402*, *ido21215*, *boa28924*, and *fht47608*, for which we found new best tours, are highlighted in the table.

### 3.2 The Effectivity of the Pseudo Backbone Contraction

From the column **Ours** of Table 1 we observe the effectivity of the pseudo backbone contraction approach. For all but two (*fl1400*, *brd14051*) of the TSPLIB and national instances (**Src**=1, 2), we found tours that are equally good as that of the best known ones, most of them are optimally solved. For the VLSI instances we have all three cases: we found better, equally good or worse tours in comparison with the best tours known before this work. Actually for the instance *fht47608* we found three tours that are better than the previously best known

**Table 2.** Reduction data of the investigated TSP instances

| | Instance | BEP(%) | #Tours | NewSize | Impr | Grade(%) |
|---|---|---|---|---|---|---|
| 1 | dsj1000 | 95.40 | 35 | 88 | – | 53.41 |
| 2 | pr1002 | 98.20 | 35 | 34 | – | 55.88 |
| 3 | u1060 | 90.09 | 35 | 143 | – | 74.13 |
| 4 | pcb1173 | 94.63 | 35 | 118 | – | 54.24 |
| 5 | rl1304 | 94.33 | 35 | 136 | – | 55.15 |
| 6 | rl1323 | 95.16 | 35 | 120 | – | 54.17 |
| 7 | nrw1379 | 88.18 | 35 | 301 | – | 54.49 |
| 8 | fl1400 | 32.50 | 35 | 1002 | -24 | 94.41 |
| 9 | u1432 | 52.51 | 35 | 799 | – | 85.23 |
| 10 | u1817 | 73.75 | 35 | 730 | – | 65.48 |
| 11 | rl1889 | 92.80 | 35 | 245 | – | 55.92 |
| 12 | u2152 | 72.96 | 35 | 826 | – | 70.58 |
| 13 | xqc2175 | 79.91 | 66 | 733 | 0 | 59.75 |
| 14 | pr2392 | 95.40 | 35 | 212 | – | 52.36 |
| 15 | pcb3038 | 86.44 | 35 | 754 | – | 54.77 |
| 16 | pia3056 | 75.26 | 66 | 1307 | 0 | 57.92 |
| 17 | xqe3891 | 77.87 | 66 | 1447 | 0 | 59.57 |
| 18 | bgb4355 | 73.69 | 66 | 1917 | 0 | 59.83 |
| 19 | rl5915 | 91.46 | 35 | 905 | 14 | 55.91 |
| 20 | rl5934 | 91.94 | 35 | 882 | 43 | 54.31 |
| 21 | tz6117 | 77.90 | 66 | 2185 | 11 | 61.92 |
| 22 | xsc6880 | 75.74 | 66 | 2850 | 0 | 58.60 |
| 23 | bnd7168 | 70.69 | 64 | 3482 | 0 | 60.37 |
| 24 | lap7454 | 78.84 | 65 | 2715 | 0 | 58.12 |
| 25 | ym7663 | 85.03 | 66 | 2029 | 3 | 56.58 |
| 26 | ida8197 | 77.50 | 66 | 3132 | 0 | 58.91 |
| 27 | ei8246 | 84.00 | 67 | 2363 | 0 | 55.86 |
| 28 | dga9698 | 77.41 | 67 | 3694 | 0 | 59.34 |
| 29 | kz9976 | 84.13 | 66 | 2895 | 4 | 54.72 |
| 30 | xmc10150 | 75.67 | 66 | 4146 | 2 | 59.58 |
| 31 | fi10639 | 78.13 | 35 | 4099 | 21 | 56.79 |
| **32** | **xvb13584** | **65.89** | **100** | **7577** | **1** | **61.17** |
| 33 | brd14051 | 76.63 | 35 | 5719 | 8 | 57.44 |
| 34 | xrb14233 | 77.17 | 15 | 5637 | 2 | 57.65 |
| 35 | xia16928 | 65.50 | 57 | 9392 | 3 | 62.20 |
| **36** | **pjh17845** | **70.71** | **39** | **8887** | **1** | **58.83** |
| 37 | frh19289 | 82.91 | 8 | 5877 | 1 | 56.12 |
| **38** | **fnc19402** | **69.79** | **59** | **9754** | **8** | **60.11** |
| **39** | **ido21215** | **64.65** | **58** | **12233** | **9** | **61.31** |
| 40 | fma21553 | 69.09 | 59 | 10965 | 11 | 60.77 |
| 41 | lsb22777 | 80.95 | 6 | 7717 | 4 | 56.24 |
| 42 | xrh24104 | 66.63 | 60 | 13073 | 6 | 61.54 |
| 43 | irx28268 | 74.15 | 23 | 12398 | 13 | 58.94 |
| 44 | icx28698 | 67.42 | 58 | 15200 | 11 | 61.52 |
| **45** | **boa28924** | **67.13** | **59** | **15518** | **5** | **61.28** |
| 46 | pbh30440 | 81.46 | 9 | 10009 | 6 | 56.40 |
| 47 | xib32892 | 65.55 | 54 | 18242 | 8 | 62.13 |
| 48 | fry33203 | 79.66 | 8 | 11817 | 8 | 57.17 |
| 49 | ics39603 | 84.26 | 3 | 11434 | 79 | 54.54 |
| **50** | **fht47608** | **83.04** | **8** | **14568** | **30** | **55.43** |
| 51 | fna52057 | 82.00 | 5 | 16620 | 30 | 56.40 |

tour. The reasons that we could not find better tours for some instances can be described as follows. First, we gave a time limit of two weeks to the calculation. Second, we used mostly standard parameter values for all the instances, i.e., we did not do any tuning work except for parameters $r$, $s$, and $t$. The third reason, which is the most important one, is that some of the pseudo backbone edges may be no real backbones, since the starting tours were found also by local search. And finally, the best tours of some instances may be optimal and it is not possible to improve them. Note that even for the case that we found only worse tours, the length differences between our best tours and the best ones known so far are usually very small. In summary, our approach based on pseudo backbone edge contraction gives satisfiable, in some cases excellent results.

### 3.3 Discussion

Now let's consider some details about our approach. In Table 2 we give some data about the reduction for the investigated instances. The column **BEP(%)** gives the percentage of pseudo backbone edges with respect to the size of the instance. The number of the starting tours is given in column **#Tours**. After the reduction we have an instance with smaller size, where the new size can be seen in column **NewSize**. The column **Impr** gives the improvement by the reduction which is the length difference between the best starting tour and the best tour we found by pseudo backbone contraction. If the best tour in the set of starting tours is optimal, then we certainly could not improve it at all. In this case, a "–" symbol is shown in this column, if our approach finds also an optimal tour. Note that there is only one instance, for which we found only worse tours (*fl1400*). The last column **Grade** will be explained later.

We analyzed the distribution of all edges in the starting tours for every instance. Some of these edges appear in all the starting tours, therefore they are just the pseudo backbone edges. Some of them may appear in only one starting tour. These edges have the smallest probability to keep in an optimal tour. Other edges appear between these two extreme cases. Fig. 2 shows the frequency distribution of all tour edges for the instance *boa28924*. From this figure we observe that about 200 edges appear in exactly 40 (different) of the 59 starting tours and about 20,000 edges are pseudo backbone edges, i.e., appear in each of the 59 starting tours. Note that the $y$-axis is in logarithmic scale. Interestingly, the shape of this curve is typical for most of the investigated instances. From this curve it is easy to understand the high BEP values. This curve also shows a large frequency difference between that of the pseudo backbone edges ($t = 59$ in the figure) and that of the almost pseudo backbone edges for $t = 58$. As in most instances BEP is large it suffices to reduce the original instances by contracting the pseudo backbone edges, whereas almost pseudo backbone edges have not to be considered. Fig. 3 shows the pseudo backbone edges of the instance *nrw1379*, for which we found an optimal tour. From this figure, we observe that there are very long tour segments which can be contracted to corresponding $p$-edges. This means that all vertices in the middle of the segments are not included in the reduced instance. Note that for this instance all these segments are contained

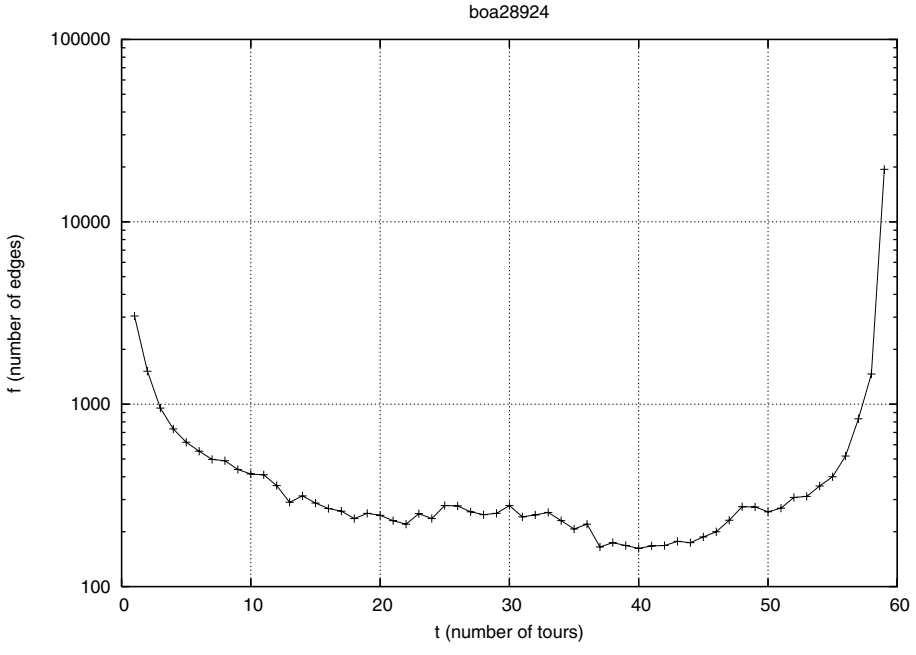



**Fig. 2.** Frequency distribution of starting tour edges for the TSP instance *boa28924*

in an optimal tour. Fig. 4 shows all $p$-edges from Fig. 3. It can be seen that the original instance can be strongly reduced. The new instance obtained after the contraction of the pseudo backbone edges has a size of 301, which is much smaller than the original size 1379. Because of the smaller size, we can search the more "important" and "difficult" areas more intensively by choosing stronger parameters.

The number of starting tours ranges from 3 to 100 (**#Tours**). For the just discussed instance *nrw1379* we have 35 starting tours. It is interesting to point out that the number of starting tours is not the most important factor. For example, for the instance *fht47608*, for which we could find three new best tours in two weeks, we have only 8 starting tours. Instead, the quality of the starting tours and also the independence among the tours play an important role. This can also be seen from the contrast of the BEP values between *fht47608* and *ei8246*. The former has slightly smaller BEP than the latter, although the latter has many more starting tours.

The length distribution of the tour segments of the pseudo backbone edges determines the new size of the reduced instance. Let $b$ be the number of pseudo backbone edges and $d$ the new size, then for $b > n/2$, which is the case for all investigated instances except one, we have:

$$n - (b - 1) \leq d \leq 2(n - b) < n.$$

The above equation follows from the following reasoning. If all the pseudo backbone edges lie in only one path, then by contracting this path to a $p$-edge,

**Fig. 3.** Tour segments for the TSP instance *nrw1379* showing only the backbone edges

we have $b-1$ vertices fewer in the new instance than in the original instance. On the other hand, we have exactly $n-b$ non-pseudo backbone edges for constructing a tour from the $b$ pseudo backbone edges. We have at most $2(n-b)$ vertices in the reduced instance, if all these non-backbone edges are disjoint. At this point, we introduce the grade of reduction as

$$\gamma = \frac{n-b+1}{d}$$

and calculate the grade of the contraction for each instance. They are listed in the last column **Grade** of Table 2. The larger this value is, the longer are the average paths consisting of only pseudo backbone edges. For most of the instances the grade of contraction has a value between 50% and 60%, but we also see a few large reduction grades. The instance *fl1400* has even a reduction grade of 94%, which indicates few but very long pseudo backbone edge paths for this instance.

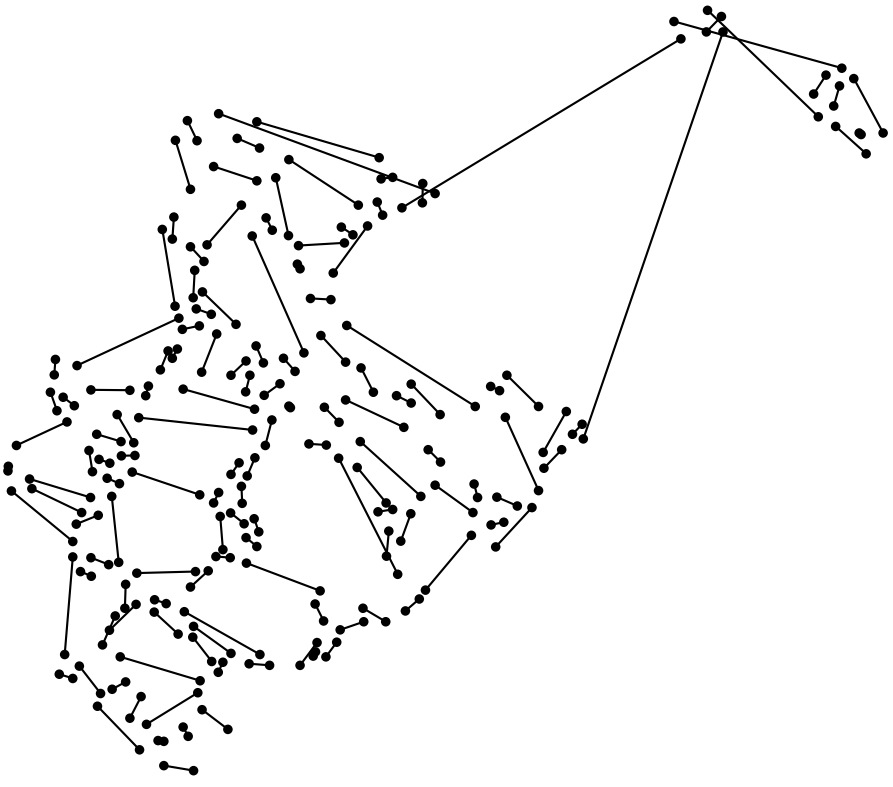


**Fig. 4.** Contracted pseudo backbone edges ($p$-edges) of TSP instance *nrw1379*

## 4 Conclusions and Current Work

Our approach of effectively finding good tours by contracting pseudo backbone edges is justified by the excellent results. For all instances except one we could find improved tours or tours with the same length. Especially we found better tours than the best ones known so far for six VLSI instances with sizes from 13,584 to 47,608, where some of the previous record tours had not been improved for several years.

A natural generalization of our idea would be to fix also edges, which do not appear in *all* starting tours, but in almost all of them. By this idea, it would be no problem, if among the starting tours also a bad one would appear.

Our current work concentrates on enhancing the central idea of our approach presented in this paper in order to attack even larger TSP instances with more than 100,000 vertices. In the present work we use the contraction of pseudo backbone edges only once, but for larger TSP instances the approach has to be extended. In [5] we suggest to apply the contraction idea in a iterative manner. This iterative approach which doesn't require starting tours is a dynamic and

automatic process, i.e., the number of the contraction steps is dynamically determined during the run of the program depending on the hardness of the instance. In each iteration, the pseudo backbone edges are computed by a window based technique in which the TSP instance is tiled in non-disjoint sub-instances. We hope that a further development of this approach makes it possible in (near) future to successfully attack the *World TSP* instance or further large TSP instances.

## Acknowledgement

This work is supported by German Research Foundation (DFG) with the grant number MO 645/7-3.

## References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem. A Computational Study. Princeton University Press, Princeton (2006)
2. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J., Espinoza, D., Goycoolea, M., Helsgaun, K.: Certification of an optimal Tour through 85,900 cities. Oper. Res. Lett. 37(1), 11–15 (2009)
3. Cook, W., Seymour, P.: Tour Merging via Branch-Decomposition. INFORMS J. Comput. 15(3), 233–248 (2003)
4. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
5. Ernst, C., Dong, C., Jäger, G., Molitor, P., Richter, D.: Finding Good Tours for Huge Euclidean TSP Instances by Iterative Backbone Contraction (submitted for Publication, 2009)
6. Fischer, T., Merz, P.: Reducing the Size of Traveling Salesman Problem Instances by Fixing Edges. In: Cotta, C., van Hemert, J. (eds.) EvoCOP 2007. LNCS, vol. 4446, pp. 72–83. Springer, Heidelberg (2007)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Berlin (2006)
8. Goldengorin, B., Jäger, G., Molitor, P.: Some Basics on Tolerances. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 194–206. Springer, Heidelberg (2006)
9. Goldengorin, B., Jäger, G., Molitor, P.: Tolerances Applied in Combinatorial Optimization. J. Comput. Sci. 2(9), 716–734 (2006)
10. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. SIGACT News 38(1), 31–45 (2007)
11. Gutin, G., Punnen, A.P. (eds.): The Traveling Salesman Problem and Its Variations. Kluwer, Dordrecht (2002)
12. Helsgaun, K.: An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal Oper. Res. 126(1), 106–130 (2000)
13. Helsgaun, K.: An Effective Implementation of K-opt Moves for the Lin-Kernighan TSP Heuristic. Writings on Computer Science 109 (2007)
14. Hüffner, F.: Algorithms and Experiments for Parameterized Approaches to Hard Graph Problems. PhD Thesis, Friedrich-Schiller-University Jena, Germany (2007)

15. Kilby, P., Slaney, J.K., Walsh, T.: The Backbone of the Travelling Salesperson. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), pp. 175–180 (2005)
16. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.): The Traveling Salesman Problem - A Guided Tour of Combinatorial Optimization. John Wiley & Sons, Chicester (1985)
17. Lin, S., Kernighan, B.W.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. Oper. Res. 21, 498–516 (1973)
18. Möbius, A., Freisleben, B., Merz, P., Schreiber, M.: Combinatorial Optimization by Iterative Partial Transcription. Phys. Rev. E 59(4), 4667–4674 (1999)
19. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyanski, L.: Determining Computational Complexity for Characteristic Phase Transitions. Nature 400, 133–137 (1998)
20. Niedermeier, R.: Invitation to Fixed-Parameter Tractability. Oxford University Press, Oxford (2006)
21. Reinelt, G.: TSPLIB – a Traveling Salesman Problem Library. ORSA J. Comput. 3, 376–384 (1991)
22. Ribeiro, C.C., Toso, R.F.: Experimental Analysis of Algorithms for Updating Minimum Spanning Trees on Graphs Subject to Changes on Edge Weights. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 393–405. Springer, Heidelberg (2007)
23. Richter, D.: Toleranzen in Helsgauns Lin-Kernighan-Heuristik für das TSP. Diploma Thesis, Martin-Luther-University Halle-Wittenberg, Germany (2006)
24. Richter, D., Goldengorin, B., Jäger, G., Molitor, P.: Improving the Efficiency of Helsgaun's Lin-Kernighan Heuristic for the Symmetric TSP. In: Janssen, J., Prałat, P. (eds.) CAAN 2007. LNCS, vol. 4852, pp. 99–111. Springer, Heidelberg (2007)
25. Slaney, J.K., Walsh, T.: The Backbones in Optimization and Approximation. In: Nebel, B. (ed.) Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), pp. 254–259 (2001)
26. Zhang, W., Looks, M.: A Novel Local Search Algorithm for the Traveling Salesman Problem that Exploits Backbones. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), pp. 343–350 (2005)
27. DIMACS Implementation Challenge: `http://www.research.att.com/~dsj/chtsp/`
28. TSPLIB: `http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html`
29. TSP Homepage, `http://www.tsp.gatech.edu/`
30. National Instances from the TSP Homepage, `http://www.tsp.gatech.edu/world/summary.html`
31. VLSI Instances from the TSP Homepage, `http://www.tsp.gatech.edu/vlsi/summary.html`
32. World, T.S.P.: from the TSP Homepage, `http://www.tsp.gatech.edu/world/`
33. Source Code of [1] (Concorde), `http://www.tsp.gatech.edu/concorde/index.html`
34. Source Code of [12] (LKH), `http://www.akira.ruc.dk/~keld/research/LKH/`
35. Additional Information about Experiments of this Paper, `http://www.informatik.uni-halle.de/ti/forschung/toleranzen/kantenkontraktion`

# Optimal Auctions Capturing Constraints in Sponsored Search*

Esteban Feuerstein[1], Pablo Ariel Heiber[1], Matías Lopez-Rosenfeld[1], and Marcelo Mydlarz[2]

[1] Departamento de Computación, FCEyN, Universidad de Buenos Aires, Argentina
{efeuerst,pheiber,mlopez}@dc.uba.ar
[2] Yahoo! Research, Santiago, Chile
marcem@yahoo.com

**Abstract.** Most sponsored search auctions use the *Generalized Second Price* (GSP) rule. Given the GSP rule, they try to give an *optimal* allocation, an easy task when the only need is to allocate ads to slots. However, when other practical conditions must be fulfilled –such as budget constraints, exploration of the performance of new ads, etc.– optimal allocations are hard to obtain. We provide a method to optimally allocate ads to slots under the practical conditions mentioned above. Our auctions are stochastic, and can be applied in tandem with different pricing rules, among which we highlight two: an intuitive generalization of GSP and VCG payments.

## 1 Introduction

In the framework of sponsored search, advertisers compete in an auction to place their ads on a web page. Each advertiser places a bid, and the search engine decides, based on the bids and other public or private parameters, which ads will be published and where. In the widely used pay-per-click model, each advertiser is charged only when her ad receives a click. The position in which an ad is displayed has an impact on its likelihood of being clicked, and advertisers' preferences follow accordingly. Nevertheless, it is generally assumed that all clicks (independently of the ads' position) have the same value for an advertiser; consequently, search engines establish a price for a click that is not conditioned on the position where the ad is presented.

The most widely used mechanism for sponsored search auctions combines the sort-by-revenue allocation and Generalized Second Price (GSP) rules: bidders are ranked according to the revenue the auctioneer expects to obtain from them, while the price associated to each of the winning bidders –which are precisely the top-ranked bidders– is the minimum amount each of them needs to bid in order to maintain their position in the ranking [3,10].

* This research was funded by a Yahoo! Research Alliance Grant, and in part by UBACYT project X436 "Algoritmos de selección y asignación y mecanismos de pago para la publicidad online en Internet".

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 188–201, 2009.

Although in a basic setting the aforementioned tandem sort-by-revenue/GSP yields good revenue for the auctioneer, actual implementations must address many extra features, for instance: (a) Click-through-rates and expected revenues are not really known a priori, so they must be learned somehow by the auctioneer. In order to avoid leaving out ads with high potential revenue, there is a need to alternate among ads with high, small and unknown revenue expectation. This is known as the explore/exploit trade-off [25]. In terms of the allocation mechanism, the implementation of that trade-off may be seen as adding extra constraints to the problem, for example stating that each ad must receive at least a certain fraction of the impressions. (b) Users may aim at different meanings of a same query. Hence, the overall clickability of the published ads, and likewise the auctioneer's revenue may increase when the set of published ads covers a wide range of meanings, independently of the revenue expectation of each ad considered separately [14,30]. (c) The publication of certain ads may have a (possibly negative) influence on the click-probability of other ads. Therefore, the set of ads with the highest aggregate click probability is not necessarily the set of the best individual ads. These are called "contextual effects" [14,30]. (d) In the process of optimizing the performance of an auction, we must decide how many ads to display. This number influences the revenue in several conflicting ways. On the one hand, the more ads published the higher the probability that a user finds one that suits her needs. On the other hand, as the number of ads increases the fraction of the user's attention that each ad attracts decreases; moreover, there is also evidence that the user experience suffers [4]. (e) Bidders usually set budget constraints, i.e., upper bounds on the amount they are willing to spend for a keyword or set of keywords over a time period. These kind of constraints have been studied in [21,1,2]. (f) Advertisers may be allowed to place special requests such as being displayed only in certain positions.

Instead of performing ad-hoc modifications to allocations and pricing rules to model each of these extra features, it is useful to have an auction mechanism general enough to easily adapt to a changing environment. We provide such a mechanism, which simplifies and improves the usability of sponsored search auctions, by means of stochastic auctions. Stochastic auctions are auctions in which the allocation or the pricing rule (or both) are random variables. These auctions may be preferable to deterministic ones for several reasons: (1) they are less prone to vindictive and/or strategic bidding, since strategic behavior is impaired by the non-deterministic nature of the output [20]; (2) the fact that *anyone* can eventually win the auction contributes to have a wider advertisers base and therefore higher revenue in the medium term [16]; (3) they bear higher diversity of ads, which improves user experience and increases aggregate click through rates [14,30]; (4) they provide an implicit mechanism to implement an explore/exploit trade-off [12]; (5) they are in general less vulnerable to fraudulent behavior [26].

The method we propose, based on mathematical programming, creates a stochastic auction that achieves the best allocation with respect to some objective, that satisfies the constraints of the problem. Concretely, we provide an

algorithm $\mathcal{M}$, which can be subdivided into two parts: (1) a template algorithm $\mathcal{A}$ that, given as input typical parameters (the advertisers' bids, estimations of the ad- and/or position-CTRs), possibly a set of constraints (e.g., budget or variety restrictions), and an objective function $O$, produces an *equivalence class of stochastic allocations* that satisfy the constraints, and are best possible according to $O$, and (2) a drawing algorithm that allocates ads according to the probabilities of some stochastic allocation in the class obtained through $\mathcal{A}$.

The method just described can be combined with several pricing rules, as we examine in Section 4. This combination yields, for a class of pricing rules which we will call *a priori*, optimal auctions –according to different objectives. A priori pricing rules include, among others, First-price and GSP. Other pricing rules we can use include VCG payments and Myerson's optimal truthful mechanism, which requires an assumption about the bidders' valuations (see Section 4).

We also introduce a natural extension of GSP for stochastic allocations, the *Extended Generalized Second Price* (EGSP) rule: prices are not only associated to the top-ranked bidders, but to *all* the bidders with a positive probability of being allocated a slot. Prices are computed in the same way as in GSP, therefore the prices associated to the top-ranked bidders coincide under both pricing rules.

EGSP is an a-priori pricing and can be coupled with different stochastic allocation rules, in particular those obtained using algorithm $\mathcal{M}$. This combination becomes a way of extending mechanisms currently in use towards a framework where a rich set of constraints can be explicitly included.

The computational complexity of algorithm $\mathcal{A}$ depends on the objective function and the type of restrictions. A key observation is that, thanks to the stochastic nature of the allocations, there is no need to impose integrality restrictions. In particular, there are many interesing objective functions and restrictions that are linear (see Section 3), and yield polynomial time algorithms[1].

As opposed to auctions currently in use, which (to the best of our knowledge) can only handle constraints and objective functions in an ad-hoc way, $\mathcal{M}$ can handle many of those constraints and objectives seamlessly, providing an optimal allocation for many pricing rules, including EGSP. Indeed, $\mathcal{M}$ combined with EGSP brings the same or better allocations than sort-by-revenue/GSP auctions. Another advantage of $\mathcal{M}$ is related to the nature of the stochastic allocations involved: we are able to optimize over a (continuous) polytope, as opposed to a discrete lattice, where optimization is computationally inefficient.

In summary, we present an extension to the most popular pricing rule in sponsored search, and a method to derive best stochastic allocation rules based on mathematical programming under different pricings. Neither of these contributions is a break-through result, yet their combination provides a powerful way to obtain optimal auctions in some real-life settings of sponsored search. Another

[1] Even when the types of restrictions entail a non-polynomial running time algorithm, if the size of the problem is reasonably small, $\mathcal{M}$ may still be used in practice; e.g., by combining our approach with techniques for subdividing query-bidder graphs into smaller instances [8].

application is their use as benchmarks to measure the impact of the introduction of constraints on the overall performance of an auction.

**Related work.** The subject of including budgets in the design of sponsored auctions has received a lot of attention recently, for example in [7] and [1]. Mehta, Saberi, Vazirani and Vazirani [21] explore the problem from a competitive analysis point of view: they aim at optimizing the total revenue for a set of queries in an on-line manner, by trying to consume the maximum amount of each bidder's budget through a sequence of queries, of which neither the total length nor the frequency of each are known in advance, obtaining an optimal $(1 - 1/e)$-competitive algorithm. Mahdian, Nazerzadeh and Saberi [19] consider the same framework, and present an algorithm that takes advantage of good estimations on the frequencies of keywords, while maintaining a good worst-case competitive ratio in case that those estimates are incorrect.

Linear Programming and Stochastic Algorithms have been used before in the framework of mechanism design. Just to cite a recent example in the framework of truthful mechanism design for combinatorial problems, Lavi and Swamy [17] propose a way to convert LP-based approximation algorithms into stochastic mechanisms that give approximate solutions to the winner determination problem and are truthful in expectation (i.e., all players maximize their expected utility by revealing their true values). One of the consequences of that work is a truthful approximation algorithm for both multi-unit auctions and multi-unit combinatorial auctions, which are problems related to ours. The focus there is, however, on a different aspect of the problem, more related to computational complexity of one-time auctions.

Much related to our approach, Abrams, Mendelevitch and Tomlin [2] use LP trying to optimize sponsored search auctions subject to budget constraints. The difference with our work is manifold. Firstly, they impose restrictions on allocations: no ad can appear in a worse position than another ad with lower ranking. Secondly, while our formulation supports diverse pricing rules, theirs gets restricted to one: the price associated to each bidder is the minimum price needed to beat the ad allocated to the next slot. Thirdly, their model is more involved than ours, and requires more elaborated LP techniques such as delayed column generation; our model is then computationally far more efficient. Finally, it lacks the extra flexibility (given by stochastic allocations) for *easily* including other kinds of restrictions.

LP has been used in the framework of on-line advertising under the more traditional pay-per-impression model, for example in [23,29]. Finally, the use of stochastic auctions for sponsored search has been recently considered in [20], [12], [13] and [6].

## 2 The Model

**Assumptions and notation.** The setting we consider involves $n$ risk-neutral bidders that compete for slots, but no bidder can win more than one; the number of slots is not set in advance. Each bidder $i$ has a private value $v_i$ for each click

received, and for which she places a bid of $b_i$. Following each query, the auctioneer decides which ads will be published along with their order.

A *bid vector* is a vector $\mathbf{b} = (b_1, \ldots, b_n) \in \mathbb{R}^n$, while $\mathcal{B}$ denotes the set of all bid vectors. An *allocation* $s$ is an ordered subset of the ads to be displayed on a particular occurrence of a query; its size is denoted by $|s|$ (the number of ads to be displayed).

Unless stated otherwise, we assume that the click probability of an ad is not influenced by the identity of the other published ads. We denote by $CTR_{i,j,k}$ the expected click probability of ad $i$ when presented in the $j$-th position while a total of $k$ ads are displayed. The expected click-through rate of an ad $i$ in an allocation $s$, denoted by $CTR_{i,s}$, is then $CTR_{i,j,|s|}$, where $j$ is the position of $i$ in $s$. A *stochastic allocation* $S$ is a random variable with some probability distribution over allocations. The expected click through rate of ad $i$ in $S$, denoted by $CTR_{i,S}$, is the sum over all possible allocations $s$ of $CTR_{i,s}P(S = s)$ (where $P(S = s)$ denotes the probability that allocation $s$ is chosen).

Let $\mathcal{S}$ be the set of all stochastic allocations. An *allocation rule* is a function $\sigma : \mathcal{B} \to \mathcal{S}$. A *pricing* $\mu = \langle \mu_1, \ldots, \mu_n \rangle \in \mathbb{R}^n$ is a vector of prices per click for each bidder. A *stochastic pricing* is a probability distribution over pricings. Let $\mathcal{M}$ be the set of all stochastic pricings. A *pricing rule* is a function $p : \mathcal{B} \to \mathcal{M}$. An *auction* is a pair $(a, p)$, where $a$ is an allocation rule and $p$ is a pricing rule.

**Basic Model.** Now we present a basic model that uses mathematical programming in the framework of sponsored search auctions. First we define an equivalence relation over stochastic allocations, along with a polyhedron whose feasible region is the set of those equivalence classes. Then we provide an efficient algorithm that, given a point $x$ in the polyhedron, obtains a stochastic allocation $S$ from the equivalence class of $x$, followed by an allocation according to the probability distribution of $S$. Finally, we show that many typical auction measures (such as social welfare or the auctioneer's expected revenue) may be described as linear functions over the polyhedron, being therefore possible to efficiently compute the optimal equivalence class of allocations for that measure using linear programming.

We say that two stochastic allocations $S$ and $S'$ are equivalent if, and only if, for each ad $i$, each position $j$ and each $k$, the probability that ad $i$ is assigned to slot $j$ when displaying $k$ ads is the same under $S$ as under $S'$. That two equivalent stochastic allocations are not necessarily equal is shown in the following simple example. Consider two stochastic allocations $S_1$ and $S_2$; $S_1$ allocates three bidders in order $(1, 2, 3)$, $(2, 3, 1)$, and $(3, 1, 2)$ each allocation with probability $1/3$, and $S_2$ allocates them in order $(1, 3, 2)$, $(3, 2, 1)$, and $(2, 1, 3)$ each with probability $1/3$ as well. While $S_1$ and $S_2$ are different, both have the same probability of allocation for each combination of advertiser and position (there are always three ads displayed), and thus $CTR_{i,S_1} = CTR_{i,S_2}$ for each ad $i$.

If two stochastic allocations are equivalent, the expected CTR of each ad coincides in both stochastic allocations and consequently, (noting that the price per click charged to any bidder is independent of the slot assigned to her) both the expected revenue for each bidder and for the auctioneer are the same as well in both auctions (under equal pricings).

**The Stochastic Allocations Polyhedron.** By regarding $y_k$ as the probability of having $k$ ads displayed, and $x_{i,j,k}$ as the probability of ad $i$ being displayed on position $j$ when a total of $k$ ads are displayed ($1 \leq i \leq n$, $1 \leq j \leq k \leq n$), we define the Stochastic Allocations Polyhedron (SAP) by

$$x_{i,j,k} \geq 0 \qquad \text{for each } i,j,k \quad (1) \qquad \sum_{k=1}^{n} y_k = 1 \qquad (2)$$

$$\sum_{i=1}^{n} x_{i,j,k} \leq y_k \qquad \text{for each } j,k \quad (3) \qquad \sum_{j=1}^{k} x_{i,j,k} \leq y_k \qquad \text{for each } i,k. \quad (4)$$

For convenience we also define an extension of SAP (SAP-e), by adding non-negative (slack) variables $x_{i,j,k}$ for $j > k$, and replacing inequalities (3) and (4) with

$$\sum_{i=1}^{n} x_{i,j,k} = y_k \qquad \text{for each } j,k \quad (3a) \qquad \sum_{j=1}^{n} x_{i,j,k} = y_k \qquad \text{for each } i,k. \quad (4a)$$

It is easy to see that the feasible region of SAP-e may be partitioned into subsets such that each of these subsets is associated with one feasible solution of SAP (by dropping the slack variables). While each feasible solution of SAP represents a class of stochastic allocations, SAP-e will prove useful for technical purposes.

Note that if we replace $y_k$ by 1, and remove the third coordinate of the variables $x_{i,j,k}$, then (1), (3a) and (4a) describe the *bipartite perfect matching polytope* [11,27,18]. As in the bipartite matching polytope, we state in the next lemma that SAP extremes are also integral.

**Lemma 1.** *Every extreme of SAP (SAP-e) is integral.*

**A drawing algorithm.** Each solution of SAP (or SAP-e) can be associated with a set of *equivalent* stochastic allocations. Given such solution $(x_{i,j,k}, y_k)_{1 \leq i,j,k \leq n}$ in SAP-e, we show next how to obtain a stochastic allocation in its equivalence class.

For each $k$ such that $y_k > 0$ we define the $n \times n$ matrix $Z^{(k)} = (z_{i,j}^{(k)})_{1 \leq i,j \leq n}$ by $z_{i,j}^{(k)} = x_{i,j,k}/y_k$. From this definition and restrictions (3a) and (4a) follows that each row and column of $Z^{(k)}$ sums up to 1, that is, $Z^{(k)}$ is a doubly stochastic matrix. In consonance with the Birkhoff-von Neumann theorem [5], $Z^{(k)}$ is a convex combination of permutation matrices. Accordingly, we give the following probabilistic algorithm that produces an allocation given a point in SAP:

Choose $k$ with probability $y_k$.
Construct $Z^{(k)}$.
Find permutation matrices $P_l$ and positive numbers $\lambda_l$
  such that $\sum_l \lambda_l = 1$ and $Z^{(k)} = \sum_l \lambda_l P_l$.
Choose a permutation matrix $P_l$ with probability $\lambda_l$.
For $j = 1$ to $k$
  Let $i$ be such that $P[i, j] = 1$.
  Display ad $i$ in position $j$.

We can see that with the preceding algorithm the probability of displaying exactly $k$ ads is $y_k$, and the probability that ad $i$ is displayed in position $j$ while having $k$ ads on display is $y_k z_{i,j}^{(k)} = x_{i,j,k}$. Therefore, the stochastic allocation that results from the application of the algorithm to a point in SAP belongs to the equivalence class of the point (the stochastic allocation selected from the equivalence class depends on the convex combination found, which is not necessarily unique).

Now our procedure is clear: apply an instance of the template algorithm $\mathcal{A}$ in order to obtain a solution of SAP, and feed that solution to the drawing algorithm in order to obtain an allocation. Since the convex combination for any given matrix $Z^{(k)}$ can be attained in polynomial time, the drawing algorithm takes polynomial time as well. As long as the instance of $\mathcal{A}$ also runs in polynomial time, so will our procedure.

Lemma 1 implies that we are modeling an assignment problem, which can be solved with faster methods than using linear programming plus the drawing algorithm. Nevertheless, as it will become clear in in the next section, this model provides an extra flexibility that enables the inclusion of different extensions.

## 3 Optimizing over SAP and Extensions

In order to round up the description of our model, we note that we can optimize any function over SAP. In particular, we consider linear functions, that yield linear programs. A natural instance of such functions is the social welfare, which can be maximized if we have the bidders' private values (or an estimation): $\sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{k} x_{i,j,k} CTR_{i,j,k} v_i$. Alternatively, we can maximize the expected revenue of the auctioneer, $\sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{k} x_{i,j,k} CTR_{i,j,k} \mu_i$. We can also maximize any linear combination of measures, therefore being able to tweak the trade-off between different objectives. We give an example of this kind of objective functions at the end of the next subsection.

We note that the model depends on some parameters of the environment, mainly the pricing and the click probabilities. As for the pricing, the vector $\mu = \langle \mu_1, \ldots, \mu_n \rangle$ that we use may be a function of the bids and eventually other variables (like the click probabilities themselves), but we restrict $\mu$ to not depend on the allocation rule[2]. In other words, when the objective function depends on

[2] We could also use $\mu_{i,j,k}$, allowing the price to depend on the number of ads displayed and their positions.

the price (e.g., the revenue function), the model is general enough to represent any stochastic sponsored search auction in which the prices are calculated "a priori" of the assignment. This includes classical pricing rules such as first-price or appropriate variants of the second-price rule. In Section 4 we present a natural extension of the broadly used Generalized Second Price (GSP) rule [10], which is suitable for our auctions. Note that we make no equilibrium analysis in these cases.

Since the extremes of SAP are binary solutions, maximizing any continuous objective function may be seen as an assignment problem. It is only natural then that we can obtain the same solutions with less sophisticated methods than linear programming. Nevertheless, the power of the model presented in Section 2 lies on its flexibility: it may be combined with different objective functions, restricted by adding different types of constraints, and extended by adding new variables that represent other features of the auction. Next we show a few practical instances of these extensions.

We restrict our analysis to linear programming, therefore both the objective functions and constraints considered are linear. We also note that although we consider each extension separately, they can be combined according to the features being modeled.

### 3.1 Variety Constraints

The first extension to the basic model that we consider are *variety constraints*. They are introduced as a means of granting each bidder some minimum reward in terms of impressions or click probability.

We consider two types of variety constraints: 1. Each ad is granted some probability of appearance on each occurrence of the query, 2. Each ad is granted a minimum expected click probability on each occurrence of the query. Both of them can be modeled by linear constraints to restrict the solution set of SAP. The first one may be appealing to some advertisers who are interested in impressions rather than clicks. However, the usual model for sponsored search considers that bidders only get profit on clicks, so we focus on type 2 constraints. In order to define type 2 constraints we make use of the *separability assumption* [3]: the CTR may be separated into two factors, one advertisement-specific, the *ad-CTR*, and the other position-specific, the *position-CTR*. Formally, denoting by $a_i$ the ad-CTR of ad $i$, and by $w_{j,k}$ the position-CTR of slot $j$ when $k$ ads are displayed, the separability assumption states that $CTR_{i,j,k} = a_i w_{j,k}$. For convenience, we assume without loss of generality that the weights are sorted in such a way that $CTR_{i,j,k} \geq CTR_{i,j+1,k}$ ($w_{j,k} \geq w_{j+1,k}$). We also assume that $CTR_{i,j,k} \geq CTR_{i,j,k+1}$ ($w_{j,k} \geq w_{j,k+1}$), since having additional ads displayed can only reduce the visibility of the others. If the maximum number of ads to be displayed is $m$, then we set $w_{j,k} = 0$ for all $k > m$.

By denoting with $l_i^a$ and $l_i^p$ the lower bounds on bidder $i$'s expected impression probability and position-CTR, respectively, constraints of type 1 and 2 can be respectively expressed by

$$\sum_{k=1}^{n}\sum_{j=1}^{k} x_{i,j,k} \geq l_i^a \quad \text{for each } i \quad \text{and} \quad \sum_{k=1}^{n}\sum_{j=1}^{k} w_{j,k}x_{i,j,k} \geq l_i^p \quad \text{for each } i.$$

Since the lower bounds are part of the input to $\mathcal{M}$, different alternatives are possible. They are beyond the scope of this article. Nevertheless, we observe that for constraints of type 1, no feasible solution exists unless $\sum_{i=1}^{n} l_i^a < k$, for the largest $k$ satisfying $w_{k,k} > 0$. It is less evident that for constraints of type 2 and assuming $l_i^p > l_{i+1}^p$, no feasible solution exists unless there is $k$ such that for each $1 \leq t \leq n$, $\sum_{i=1}^{t} l_i^p \leq \sum_{i=1}^{t} w_{i,k}$; we omit the proof due to space limitations.

Alternatively, the need of periodically publishing every ad can be expressed via another objective function such as $\sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{k} x_{ijk}(w_{jk}a_i\mu_i + E_i)$, where $E_i$ is a measure of the benefit of exploring ad $i$, provided by an external source devoted to manage the explore/exploit trade-off [25].

### 3.2 Budget Constraints

We introduce now *budget constraints*, describing how much bidders are willing to spend. In order to model these constraints, we slightly modify our polyhedron, increasing its dimension. Nevertheless, all the results presented thus far can be easily extended to support this change.

Instead of working with a single query, we consider now a set of queries $Q$; each bidders may place bids on many (possibly different) queries; Thus, we need to add to the variables and constants presented in Section 2 a new subindex $q$ ranging over $Q$. For instance, we will have variables $y_{k,q}$ and $x_{i,j,k,q}$ whenever bidder $i$ bids on $q$; a priori prices $\mu_{i,q}$ may be part of the input. The basic restrictions of Section 2 become

$$x_{i,j,k,q} \geq 0 \quad \text{for each } q,i,j,k \qquad \sum_{k=1}^{n} y_{k,q} = 1 \quad \text{for each } q$$

$$\sum_{i=1}^{n} x_{i,j,k,q} \leq y_{k,q} \quad \text{for each } q,j,k \qquad \sum_{j=1}^{n} x_{i,j,k,q} \leq y_{k,q} \quad \text{for each } q,i,k.$$

Each query $q \in Q$ is expected to occur $c_q$ times during a certain time window; each bidder $i$ may set a maximum budget $B_i^{(q)}$ for $q$ and/or a maximum overall budget $B_i$. Note that some of the budgets may be set to infinity by dropping the associated restriction. Bidders that do not participate in a given query can be modeled with a 0 price, as they should never be displayed for that query.

Naturally, the objective function must be modified accordingly; e.g., the revenue maximization goal would be $\sum_q \sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{k} c_q x_{i,j,k,q} CTR_{i,j,k,q}\mu_{i,q}$.

We state now the restrictions that *in expectation* preclude bidders from going over their budgets: the expected payments of each bidder should not exceed her budget during a time window.

$$\sum_{q}\sum_{k=1}^{n}\sum_{j=1}^{k} c_q x_{i,j,k,q} CTR_{i,j,k,q}\mu_{i,q} \leq B_i \qquad \text{for each } i$$

$$\sum_{k=1}^{n}\sum_{j=1}^{k} c_q x_{i,j,k,q} CTR_{i,j,k,q}\mu_{i,q} \leq B_i^{(q)} \qquad \text{for each } i, q.$$

Note that in practice it would be possible to display an ad whose budget has been exhausted. In such cases we may choose to replace this ad by any other one, without significantly affecting the expected revenue. A possible way of reducing the incidence of such situations is to use $B_i - \varepsilon$ instead of $B_i$ in the LP constraints. We note that artificially retaining bidders with exhausted budgets may result in some illegitimate price hikes for other bidders, depending on the pricing rule. This effect however can be controlled if needed.

### 3.3 Contextual Effects and Other Restrictions

We consider now situations where the click probability of an ad is influenced by the other ads displayed. These are called *externalities* or *contextual effects*, and have been considered recently in [14,15]. The latter argues through experimental evidence that contextual effects do exist in sponsored search, and quantifies them.

One possible way of modeling contextual effects in our framework is by grouping together ads that have negative effects on each other's clickability and introducing a new kind of restriction, that aims at avoiding the joint publication of ads in the same group. In order to establish incompatibilities among groups of similar ads, *ads are partitioned into incompatibility groups* (each ad in exactly one group). The restriction states that *at most one ad of each group can be displayed* at the same time. This approach may be useful, for instance, when a query has different meanings – so users that search for that query may have different intentions– and the auctioneer tries to cover all the range without increasing the total number of ads shown.

In this extension, apart from adding restrictions to SAP, we also need to refine the drawing algorithm given in Section 2 that produces the allocation. This can be done with minor adjustments. Let $m$ be the number of incompatibility groups. Since we will assign $m$ groups instead of $n$ bidders to the slots, we set $n$ to be $m$. We need to ensure that each group receives, for each particular $k$, an aggregated probability of exactly $y_k$ of being assigned some position. Letting $G$ be the set of groups, this is captured by

$$\sum_{i\in g}\sum_{j=1}^{m} x_{i,j,k} = y_k \qquad \text{for each } g \in G, k.$$

We can maximize different objective functions over this new polyhedron, looking for good "group allocations". Concrete ad allocations will be produced by a modified version of the drawing algorithm. For each $k$ such that $y_k > 0$, we

construct an $m \times m$ matrix $Z^{(k)}$ that is used to choose group permutations instead of bidder permutations, and then for each selected group $g$, choose a bidder from $g$ and assign it to the slot. $Z^{(k)}$ is defined by $z^{(k)}_{l,j} = \sum_{i \in G_l} x_{i,j,k}/y_k$ for $1 \leq l, j \leq m$. Note that again $Z^{(k)}$ is a doubly stochastic matrix, so we apply the Birkhoff-von Neumann theorem as in Section 2. The new drawing algorithm will then be:

Choose $k$ with probability $y_k$.
Construct $Z^{(k)}$.
Find permutation matrices $P_l$ and positive numbers $\lambda_l$
  such that $\sum_l \lambda_l = 1$ and $Z^{(k)} = \sum_l \lambda_l P_l$.
Choose a permutation matrix $P_l$ with probability $\lambda_l$.
For $j = 1$ to $k$
  Let $g$ be such that $P[g, j] = 1$.
  Choose ad $i$ of group $g$ with probability $\frac{x_{i,j,k}}{\sum_{i \in g} x_{i,j,k}}$.
  Display ad $i$ in position $j$.

It follows immediately that the probability of ad $i$ of group $l$ being placed on position $j$ when $k$ ads are displayed is $y_k z^{(k)}_{l,j} x_{i,j,k} / \sum_{i \in G_l} x_{i,j,k} = x_{i,j,k}$.

Grouping can also be used in other ways, such as setting the minimum number of ads to display from each group. However, this would require a deeper modification in the drawing algorithm.

## 4 Pricing Rules

Auctions are constituted by two main components: the allocation rule and the pricing rule. So far we have focused on obtaining optimal allocations (according to some criteria) that satisfy a number of restrictions. In this section, we focus on the pricing rules that can be combined with those allocation rules.

We divide our analysis in two directions: first we analyze pricing rules that are best suited for our optimization framework, and then we consider incentive-compatible (truthful) pricing rules. In the first case we do not make a game-theoretic analysis, we assume that bids do not change in response to the allocation rule. With truthful pricings, however, we can assume that the bids are in equilibrium since by definition every bidder maximizes her expected revenue by bidding her true value.

**A priori pricing rules and EGSP.** When the prices associated to the ads may depend on the ranking, but not on the allocation, these prices may be part of the input of $\mathcal{M}$, so as to find the best allocation with respect to some objective that depends on them[3]. We call these pricing rules *a priori*. Several well-known and

[3] In the general case, we do allow a *light* dependence on the allocation: we only associate a non-zero price when an item is allocated; nevertheless, in the pay-per-click model this distinction disappears since an ad that is not displayed cannot be clicked and therefore will not be charged.

widespread pricing rules are indeed a priori; most notorious examples of this class are First-Price and the Generalized Second Price (GSP) [3,10]. While the former may be applied in our framework, it has been dropped from sponsored-search settings due to its instability. On the other hand, GSP has become the most widely used rule in that framework, enjoying good properties such as *envy-free* equilibria (see [10] for details).

We introduce a natural extension of GSP for stochastic allocations, the *Extended Generalized Second Price* (EGSP) rule. Like in GSP, EGSP assumes that the auctioneer ranks bidders according to some function on their bids while each winning bidder pays (for a click) the minimum price needed to retain her position in the ranking. However, with EGSP prices are not only associated to the top-ranked bidders, but to *all* the bidders with a positive probability of being allocated a slot. Since prices are computed in the same way as in GSP, the prices associated to the top-ranked bidders coincide under both pricing rules. Note that, as we are dealing with stochastic allocations, the ranking order is not necessarily the order in which ads are displayed each time; nevertheless, the resulting stochastic allocation rule will tend to allocate more/better slots to ads with a higher ranking.

Another extension of GSP to an allocation rule different than the simple sort-by-revenue rule has been proposed in [2]: given allocations that are subsets of the ads ordered by ranking, the price associated to each bidder is the minimum price needed to beat the ad allocated to the following slot (the price associated to the last ad is the reserve price). It is easy to see that, given any set of bids, EGSP charges strictly more than the pricing rule in [2].

**Incentive Compatibility.** Although the variations over the GSP rule currently in use in sponsored search auctions are not truthful, there are many reasons that make truthfulness a desirable property, which can be summarized in the fact that advertisers can define their optimal bids by themselves, without the need of invoking consultants or gurus, driving more resources to the sponsored search business, for the benefit of advertisers, auctioneer and users.

A natural way to incorporate truthfulness into our framework consists of the classical VCG approach [28,9], that is, the incentive-compatible pricing rule corresponding to the allocation obtained through $\mathcal{M}$ that is individually rational and makes no positive transfers [24]. For instance, when bidder $i$'s bid is $b_i$ (interpreted as values $v_i$, since in a truthful auction we can assume that each bidder bids her own private valuation), and the objective funcion is the social welfare $\sum_{i=1}^{n}\sum_{k=1}^{n}\sum_{j=1}^{k} x_{i,j,k}CTR_{i,j,k}b_i$, we associate to each bidder a price equal to the difference between social welfare value obtained with and without her participation. In this way, in order to compute the price for the $n$ bidders we run the mathematical program $n+1$ times. In practice the computation can be done "on demand", that is, only when an ad indeed receives a click.

Another approach to truthfulness within our framework is the following: given a distribution $F_i$ on the valuation of each bidder $i$ (known or inferred through historical data), we apply Myerson's incentive-compatible mechanism [22,16],

which maximizes the (expected) revenue of the auctioneer[4]. For an explanation on how to apply this mechanism, see for example [24].

**Further Research.** We are currently studying "a priori" pricing methods and their consequences, in particular the existence of equilibra for stochastic auctions under EGSP.

Another interesting research subject are auctions in which the expected position-CTR of the $i$-th ranked bidder is set to a value $p_i$. Such auctions, though stochastic in nature, behave like deterministic ones, so paired with EGSP will have interesting properties, such as the existence of envy-free equilibria [10].

**Acknowledgments.** We thank Ofer Mendelevitch and John Tomlin for providing us with data for the simulations.

## References

1. Abrams, Z.: Revenue maximization when bidders have budgets. In: SODA, pp. 1074–1082 (2006)
2. Abrams, Z., Mendelevitch, O., Tomlin, J.: Optimal delivery of sponsored search advertisements subject to budget constraints. In: ACM Conference on Electronic Commerce, pp. 272–278 (2007)
3. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: ACM Conference on Electronic Commerce, pp. 1–7 (2006)
4. Bhargava, H.K., Feng, J.: Paid placement strategies for internet search engines. In: WWW 2002: Proceedings of the 11th international conference on World Wide Web (2002)
5. Birkhoff, G.: Tres observaciones sobre el algebra lineal. Univ. Nac. Tucumán. Revista, 147–151 (1946)
6. Borgs, C., Chayes, J., Etesami, O., Immorlica, N., Jain, K., Mahdian, M.: Dynamics of bid optimization in online advertisement auctions. In: 16th International World Wide Web Conference (WWW 2007) (2007)
7. Borgs, C., Chayes, J.T., Immorlica, N., Mahdian, M., Saberi, A.: Multi-unit auctions with budget-constrained bidders. In: ACM Conference on Electronic Commerce, pp. 44–51 (2005)
8. Carrasco, J.J., Fain, D.C., Lang, K.J., Zhukov, L.: Clustering of bipartite advertiser-keyword graph. In: Workshop on Large Scale Clustering at IEEE International Conference on Data Mining (2003)
9. Clarke, E.H.: Multipart pricing of public goods. Public Choice, 17–33 (1971)
10. Edelman, B.G., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. Stanford Graduate School of Business Research Paper No. 1917 Available at SSRN (2005), `http://ssrn.com/abstract=861164`
11. Edmonds, J.: Paths, trees, and flowers. Canad. J. Math. 17, 449–467 (1965)
12. Feuerstein, E., Heiber, P., Martínez-Viademonte, J., Baeza-Yates, R.: New stochastic algorithms for placing ads in sponsored search. In: Proc. 5th Latin American Web Congress (LA-WEB 2007) (2007)

[4] This may affect the incentives of the advertisers in previous stages of the mechanism.

13. Feuerstein, E., Heiber, P., Mydlarz, M.: Truthful stochastic and deterministic auctions for sponsored search. In: Proc. 6th Latin American Web Congress (LA-WEB 2008) (2008)
14. Ghosh, A., Mahdian, M.: Externalities in online advertising. In: 17th International World Wide Web Conference (WWW 2008) (2008)
15. Gunawardana, A., Meek, C.: Aggregators and contextual effects in search ad markets. In: WWW Workshop on Targeting and Ranking for Online Advertising (2008)
16. Klemperer, P.: Auction theory: A guide to the literature. Journal of Economic Surveys 13(3), 227–286 (1999)
17. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. In: FOCS, pp. 595–604 (2005)
18. Lovász, L., Plummer, M.D.: Matching Theory. In: Annals of Discrete Mathematics. North-Holland, Amsterdam (1986)
19. Mahdian, M., Nazerzadeh, H., Saberi, A.: Allocating online advertisement space with unreliable estimates. In: ACM Conference on Electronic Commerce, pp. 288–294 (2007)
20. Meek, C., Chickering, D.M., Wilson, D.B.: Stochastic and contingent-payment auctions. In: Workshop on Sponsored Search Auctions - ACM Conference on Electronic Commerce (EC 2005) (2005)
21. Mehta, A., Saberi, A., Vazirani, U., Vazirani, V.: Adwords and generalized on-line matching. In: Proc. Symposium on Foundations of Computer Science (2005)
22. Myerson, R.: Optimal auction design. Mathematics of Operations Research 6, 58–73 (1981)
23. Nakamura, A., Abe, N.: Improvements to the linear programming based scheduling of web advertisements: World wide web electronic commerce, security and privacy. In: zurko, M.e., greenwald, a. (guest eds.) Electronic Commerce Research, vol. 5, pp. 75–98 (2005)
24. Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V.V. (eds.): Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
25. Pandey, S., Olston, C.: Handling advertisements of unknown quality in search advertising. In: Proc. Twentieth Annual Conference on Neural Information Processing Systems (NIPS), Vancouver, Canada (2006)
26. Penemberg, A.: Click fraud threatens web. Wired news, October 13 (2004)
27. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & Sons, Chichester (1986)
28. Vickrey, W.: Counterspeculation, auctions, and competitive sealed tenders. The Journal of Finance 16(1), 8–37 (1961)
29. Wiggins, R., Tomlin, J.: Bid optimization for internet graphical ad auction systems via special ordered sets. Yahoo! Research Technical Report YR-2007-004 (2007)
30. Zhu, X., Goldberg, A., Van Gael, J., Andrzejewski, D.: Improving diversity in ranking using absorbing random walks. In: Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAAC-HLT) (2007)

# A Note on Estimating Hybrid Frequency Moment of Data Streams

Sumit Ganguly

Indian Institute of Technology, Kanpur

**Abstract.** We consider the problem of estimating the hybrid frequency moment of matrix data that is updated point-wise in arbitrary order by a data stream. In this model, data is viewed to be organized in the form of a matrix $(A_{i,j})_{1 \leq i,j, \leq n}$. The entries $A_{i,j}$ are updated coordinate-wise (both increments and decrements are allowed), in arbitrary order and possibly multiple times. The hybrid frequency moment $F_{p,q}(A)$ is defined as $\sum_{j=1}^{n} \left(\sum_{i=1}^{n} |A_{i,j}|^p\right)^q$ and is a generalization of the frequency moment of one-dimensional data streams.

Prior work [10] presented a nearly space-optimal algorithm for estimating $F_{p,q}$ for $p \in [0,2]$ and $q \in [0,1]$. Here, we complement that work by presenting a nearly space-optimal algorithm for estimating $F_{p,q}$ for $p \in [0,1]$ and $q \in [0,2]$.

## 1 Introduction

The data stream model of computation is an abstraction for a variety of monitoring applications. A problem of basic utility and relevance in this setting is the following *hybrid frequency moments estimation* problem. Consider a networking application where a stream of packets with schema (*src-addr*, *dest-addr*, *nbytes*, *time*) arrives at a router. The problem is to warn against the following scenario arising out of a possible distributed denial of service attack, where, a few destination addresses receive messages from an unusually large number of distinct source addresses. This can be quantified as follows: let $A$ be an $n \times n$ matrix where $A_{i,j}$ is the count of the number of messages from node $i$ to node $j$. Then $(A_{i,j})^0$ is 1 if $i$ sends a message to $j$ and is 0 otherwise. Thus, $\sum_{i=1}^{n} A_{i,j}^0$ counts the number of distinct sources that send at least one message to $j$. Define the hybrid moment $F_{0,2}(A) = \sum_{j=1}^{n} (\sum_{i=1}^{n} A_{i,j}^0)^2$. In an attack scenario, $F_{0,2}(A)$ becomes large compared to its average value. Thus, it is advantageous to track the values of $F_{0,2}(A)$. However, since $n$ can be very large (e.g., in the millions), it is not feasible to store and update the traffic matrix $A$ at network line speeds. We propose instead to use the data streaming approach to this problem, namely, to design a sub-linear space data structure that, (a) processes updates to the entries of $A$, and, (b) provides a randomized, approximate algorithm for estimating $F_{0,2}(A)$.

Quantities such as $F_{0,2}(A)$ are known as the hybrid moment of a matrix $A$ [10]. They are more generally defined [16] as follows. Given an $n \times n$ integer

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 202–211, 2009.

matrix $A$ with columns $A_1, A_2, \ldots, A_n$, the hybrid frequency moment $F_{p,q}(A)$ is the $q$th moment of the $n$-dimensional vector $[F_p(A_1), F_p(A_2), \ldots, F_p(A_n)]$. That is,

$$F_{p,q}(A) = \sum_{j=1}^{n} \left( \sum_{i=1}^{n} |A_{i,j}|^p \right)^q = \sum_{j=1}^{n} (F_p(A_j))^q \ .$$

*Data Stream Model.* We will be interested in algorithms in the data stream model, that is, the input is abstracted as a potentially infinite sequence $\sigma$ of records of the form $(position, i, j, \Delta)$, where, $i, j \in \{1, 2, \ldots, n\}$ and $\Delta \in \mathbb{Z}$ is the change to the value of $A_{i,j}$. The *position* attribute is simply the sequence number of the record. Each input record $(position, i, j, \Delta)$ changes $A_{i,j}$ to $A_{i,j} + \Delta$. In other words, the value $A_{i,j}$ is the sum of the changes made to the $(i, j)$th entry since the inception of the stream:

$$A_{i,j} = \sum_{(position\ ,i,j,\Delta)\in\sigma} \Delta, \quad 1 \leq i, j \leq n \ .$$

In this paper, we consider the problems of estimating $F_{p,q}$ and allow general matrix streams, that is, matrix entries may be positive, zero or negative.

*Prior work.* Hybrid frequency moments $F_{p,q}(A)$ are a generalization of the frequency moment $F_p(a)$ of an $n$-dimensional vector $a$, defined as $F_p(a) = \sum_{j=1}^{n} |a_i|^p$. The problem of estimating $F_p(a)$ has been studied in the data stream model where the input is a stream of updates to the components of $a$. This problem has been influential in the development of algorithms for data streams. As terminology, we will say that a randomized algorithm computes an $\epsilon$-approximation to a real valued quantity $L$, provided, it returns $\hat{L}$ such that $|\hat{L} - L| < \epsilon L$, with probability $\geq \frac{3}{4}$.

Alon, Matias and Szegedy [1] present a seminal randomized sketch technique for $\epsilon$-approximation of $F_2(a)$ in the data streaming model using space $O(\epsilon^{-2} \log F_1(a))$ bits. Using the techniques of [1], it is easily shown that deterministically estimating $F_p(a)$ for any real $p \geq 0$ requires $\Omega(n)$ space [1]. Hence, work in the area of sub-linear space estimation of moments has considered only randomized algorithms. Estimation of $F_0(a)$ was first considered by Flajolet and Martin in [9]; the work in [1] presents a modern version of this technique for estimating $F_0(a)$ to within a constant multiplicative factor and using space $O(\log n)$. Gibbons and Tirthapura [11] present an $\epsilon$-approximation algorithm using space $O(\epsilon^{-2} \log F_1(a))$; this is further improved in [3]. The use of $p$-stable sketches was proposed by Indyk [12] for estimating $F_p(a)$, for $0 < p \leq 2$, using space $\tilde{O}(\epsilon^{-2}(\log F_1(a)))$. Indyk and Woodruff [13] present a near optimal space algorithm for estimating $F_p$, for $p > 2$. Woodruff [20] presents an $\Omega(\epsilon^{-2})$ space lower bound for the problem of estimating $F_p$, for all $p \geq 0$, implying that the stable sketches technique is space optimal up to logarithmic factors. A space lower bound of $\Omega(n^{1-2/p})$ was shown for the problem $F_p$ in a series of developments [1,2,5]. Cormode and Muthukrishnan [8] present an algorithm for obtaining

an $\epsilon$-approximation for $F_{0,2}(A)$ using space $\tilde{O}(\sqrt{n})$. In [10], a bi-linear stable sketches technique is presented that estimates $F_{p,q}(A)$, for $p \in [0,2]$ and $q \in [0,1]$ in $\tilde{O}(1)$ space.

*Contributions.* We present randomized algorithms for the problem of estimating hybrid moments $F_{p,q}(A)$ of a matrix $A$ in the data stream model. We consider the range $p \in [0,1]$ and $q \in [0,2]$. We present a novel variation of the stable sketches technique to obtain a $\tilde{O}(1)$ space algorithm for estimating $F_{p,q}$ in this range. This gives the first poly-logarithmic space complexity algorithm for estimating $F_{p,q}$ in this interesting range, as motivated by the example in the introduction.

## 2 Preliminaries and Previous Work

In this section, we review salient properties of stable distributions and briefly review Indyk's [12] and Li's [14] techniques for estimating moments of one-dimensional vectors in the data streaming model. We use the notation $y \sim D$ to denote that a given random variable $y$ that follows a distribution $D$.

### 2.1 Stable Sketches

Indyk [12] proposed the use of stable sketches for estimating $F_q(a)$, $q \in [0,2]$, in the streaming model. A stable sketch is a linear combination $X = \sum_{i=1}^{n} a_i s_i$, where each $s_i$ is drawn at random from the stable distribution $S(q,1)$, where, the first parameter in $S(q,1)$ is the stability parameter and the second parameter is the scale factor (set to 1). The random variables $s_i$'s are independent of each other. By property of stable distributions,

$$X \sim S\left(q, (F_q(a))^{1/q}\right).$$

The problem now reduces to the estimation of the scale parameter of the distribution of $X$. Indyk proposed keeping $t = O(\frac{1}{\epsilon^2})$ independent stable sketches $X_1, X_2, \ldots, X_t$ and returning

$$\hat{F}_q(a) = C_I \cdot \text{median}_{r=1}^{t} |X_r|^q.$$

Li [14] uses the geometric means estimator

$$\hat{F}_q(a) = C_L \prod_{r=1}^{t} |X_r|^{q/t}$$

and shows that this is asymptotically unbiased for a proper choice of the constant $C_L$. Both estimators satisfy

$$|\hat{F}_q(a) - F_q(a)| \leq \epsilon \hat{F}_q(a), \text{ with probability } \frac{7}{8}.$$

We will jointly refer to Indyk's median estimator or Li's geometric means estimator for $F_p(a)$ as $\text{StableEst}^q(\{X_1, X_2, \ldots, X_t\})$, where, $q \in (0, 2]$ is the stability index.

Estimation of hybrid moments generalizes the problem of estimating the regular moment $F_p(a)$ for an $n$-dimensional vector $a$. In particular, for any $p$, $F_{p,1}(A) = F_p(a)$ where $a$ is the $n^2$-dimensional vector obtained by stringing out the matrix $A$ row-wise (or column-wise). Therefore, $F_{p,1}(A)$ can be estimated using standard techniques for estimating $F_p$ of one-dimensional vectors, that is, using space $\tilde{O}(\epsilon^{-2})$ sketches.

### 2.2 Bi-linear Stable Sketches

We now review the bilinear stable sketches technique [10] for estimating $F_{p,q}$ in the range $p \in [0, 2]$ and $q \in [0, 1]$ using bilinear stable sketches.

We first consider $p \in (0, 2]$ and $q \in (0, 1]$. Consider two families of fully independent stable variables $\{x_{i,j} : 1 \leq i \leq j \leq n\}$ and $\{\xi_j : 1 \leq j \leq n\}$, where, $x_{i,j} \sim S(p, 1)$ and $\xi_j \sim S(q, 1)$. A $p, q$ bi-linear stable sketch is defined as

$$X = \sum_{j=1}^{n} \sum_{i=1}^{n} A_{i,j} x_{i,j} \xi_j^{1/p} \ .$$

Corresponding to each stream update $(pos, i, j, \Delta)$, the bi-linear sketch is updated as follows: $X := X + \Delta \cdot x_{i,j} \cdot \xi_j^{1/p}$.

A collection of $s_1 s_2$ bi-linear sketches $\{X_{u,v} \mid 1 \leq u \leq s_1, 1 \leq v \leq s_2\}$ is kept such that for each distinct value of $v$, the family of sketches $\{X_{u,v}\}_{u=1,2,\ldots,s_1}$ uses the independent family of stable variables $\{x_{i,j}(u, v)\}$ but uses the same family of stable variables $\{\xi_i(v)\}$. That is,

$$X(u, v) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{i,j} x_{i,j}(u, v)(\xi_i(v))^{1/p}, \ u = 1, \ldots, s_1, v = 1, \ldots, s_2. \qquad (1)$$

Note that this construction is possible, since, for $0 < q \leq 1$, there exist stable distributions $S(q, 1)$ with non-negative support. Thus, $\xi_j \sim S(q, 1)$ is non-negative and $\xi_j^{1/p}$ is non-negative. The estimate $\hat{F}_{p,q}$ is obtained using the following steps.

1. $\hat{Y}(v) = \text{StableEst}^{(p)}(\{X(u, v)\}_{u=1,\ldots,s_1}), \quad v = 1, 2, \ldots s_2$ .
2. $\hat{F}_{p,q} = \text{StableEst}^{(q)}(\{\hat{Y}(v) \mid v = 1, \ldots, s_2\})$

The correctness of the estimator is shown in [10] and is summarized below.

**Lemma 1 ([10]).** *For each $0 < p \leq 2$ and $0 < q < 1$, the estimator* $\text{BilinStable}(p, q, s_1, s_2, \{X(u, v)\}_{u \in [1,s_1], v \in [1,s_2]})$ *with parameters $s_2 = \Theta(\frac{1}{q^2 \epsilon^2})$ and $s_1 = \Theta(\frac{1}{p^2 \epsilon^2} \log \frac{1}{\epsilon q})$ satisfies $|\hat{F}_{p,q} - F_{p,q}| \leq \epsilon F_{p,q}$ with probability $\frac{3}{4}$.* □

*Reducing random bits.* We can now use a technique of Indyk [12] to reduce the number of random bits from $n^2$ bits to $O(S \log(nS))$, where, $S$ is the space used by the algorithm assuming access to fully independent random bits.

The following lemma summarizes the continuity property of $F_{p,q}$ as a function of $p$ and $q$. This allows the estimation of quantities at the boundary points of $p = 0$ and $q = 0, 1$ by estimating for $p' = \epsilon/(5 \log F_{1,1})$ and $q' = \epsilon/5$ and $1 - \epsilon/5$ respectively.

**Lemma 2 ([10]).** *For every $\epsilon < 1/8$, $p \geq 0$ and $0 \leq q \leq 2$*

$$F_{p',q'} \geq F_{p,q} \geq (1 - 5\epsilon) F_{p',q'}$$

*where, $p' = \max(p, t)$, $q' = \max(q, \epsilon)$ and $t \leq \frac{\epsilon}{\log F_{1,1}}$.* □

## 3 Estimating $F_{p,q}$ Using the Envelope Technique

In this section, we present an algorithm for estimating $F_{p,q}$ in the region $0 < p < 1$ and $0 < q \leq 2$. By the continuity properties of $F_{p,q}$ discussed in Lemma 2, this will imply an $\epsilon$-close estimator for the boundary points, namely, $p = 0, 1$ and $q = 0$.

### 3.1 Algorithm

Assume that $0 < p < 1$ and $0 < q < 2$. Let $1 \leq u \leq s_1$ and $1 \leq v \leq s_2$, where, $s_1$ and $s_2$ are specified later. We keep $2s_1s_2$ sketches as follows. Let

$$\begin{aligned} x_{i,j}(u, v) &\sim S(p, 1), \quad i, j \in [n], u \in [s_1], v \in [s_2] \\ y_j(v) &\sim S(q, 1), \quad j \in [n], v \in [s_2]. \end{aligned}$$

The random variables are assumed independent (later we will use Indyk's technique [12] for reducing the number of random bits). For each $v \in [s_2]$, we construct two families of sketches $\{X_{u,v,+}\}_{1 \leq u \leq s_1}$ and $\{X_{u,v,-}\}_{1 \leq u \leq s_1}$ as follows.

$$\begin{aligned} X_{u,v,+} &= \sum_{j : y_j(v) \geq 0} A_{i,j} x_{i,j}(u, v) |y_j(v)|^{1/p}, \quad u \in [s_1], v \in [s_2] \\ X_{u,v,-} &= \sum_{j : y_j(v) < 0} A_{i,j} x_{i,j}(u, v) |y_j(v)|^{1/p}, \quad u \in [s_1], v \in [s_2]. \end{aligned}$$

Next, we derive two families of sketches.

$$\begin{aligned} Y_{v,+} &= \text{StableEst}_{u=1}^{s_2}(p, \{X_{u,v,+}\}), \quad v = 1, 2, \ldots, s_2, \text{ and} \\ Y_{v,-} &= \text{StableEst}_{u=1}^{s_2}(p, \{X_{u,v,-}\}), \quad v = 1, 2, \ldots, s_2. \end{aligned}$$

Finally, the estimate $\hat{F}_{p,q}(A)$ is obtained as

$$\hat{F}_{p,q}(A) = \left( \frac{\text{Quantile}_{v=1}^{s_2}(Q, \{|Y_{v,+} - Y_{v,-}|\}_{v=1,\ldots,s_2})}{\text{Quantile}(Q, |S(p, 1)|)} \right)^q$$

where, (1) Quantile$(Q, \{x_1, \ldots, x_r\})$ returns the element among $x_1, \ldots, x_r$ that forms its $Q$th quantile, and, (2) Quantile$(Q, |S(p,1)|)$ is the $Q$th quantile of the absolute value of a variable $\sim S(p,1)$. Although any value of quantile $Q$ may be chosen as observed by Ping Li [15], for simplicity, $Q$ is chosen so that $G(Q) \in [0.2, 1.0]$, where, $G$ is the probability function for the absolute value of a $q$-stable distributed variable.

### 3.2 Analysis

By construction, we have for each fixed $v$,

$$X_{u,v,+} \sim S\left(p, \left(\sum_{j:y_j(v)\geq 0}^{n} F_p(A_j)|y_j(v)|\right)^{1/p}\right) \quad \text{and}$$
$$X_{u,v,-} \sim S\left(p, \left(\sum_{j:y_j(v)<0}^{n} F_p(A_j)|y_j(v)|\right)^{1/p}\right).$$

If $s_1 = O(\epsilon'^{-2}p^{-2}q^2)$, then, by properties of StableEst, for each $v \in [s_2]$, we have,

$$Y_{v,+} = \alpha_v^+ \sum_{j:y_j(v)\geq 0} F_p(A_j)|y_j(v)|, \text{ and } Y_{v,-} = \alpha_v^- \sum_{j:y_j(v)<0} F_p(A_j)|y_j(v)|.$$

where, $1 - \epsilon'/(4q) \leq \alpha_v^+, \ \ \alpha_v^- \leq 1 + \epsilon'/(4q)$ with high probability. Therefore,

$$Y_v = Y_{v,+} - Y_{v,-} = \sum_{j=1}^{n} F'_{p,v}(A_j)y_j(v) \tag{2}$$

where $F'_{p,v}(A_j)$ is defined to be a small displacement from $F_p(A_j)$ as follows.

$$F'_{p,v}(A_j) = \begin{cases} (F_p(A_j)\alpha_v^+ & \text{if } y_j(v) \geq 0 \\ F_p(A_j)\alpha_v^- & \text{if } y_j(v) < 0. \end{cases}$$

Since $1 - \epsilon'/(4q) \leq \alpha_v^+, \alpha_v^- \leq 1 + \epsilon'/(4q)$, then,

$$|F'_{p,v}(A_j) - F_p(A_j)| \leq \frac{\epsilon'}{2q} F_p(A_j), \text{ for each } v = 1, 2, \ldots, s_2.$$

For simplicity of notation define

$$F'_{p,q,v}(A) \stackrel{\text{def}}{=} \sum_{j=1}^{n} (F'_{p,v}(A_j))^q$$

From (2), we obtain $Y_v = \sum_{i=1}^{n} F'_{p,v}(A_j) \sim S\left(q, \left(F'_{p,q,v}(A)\right)^{1/q}\right)$.

Let $G = G_q$ denote the cumulative probability function of the random variable $|z|$ where $z \sim S(q,1)$. As noted in [12], a straightforward application of Chernoff's bound shows that if $z_1, \ldots, z_s$ are independent, $z_i \sim S(q,1)$, $s = O(\epsilon'^{-2}\log(1/\delta))$ and

$$Z = \text{median}_{v=1}^{s}|Z_v|^q / \text{median}^q(|S(q,1)|),$$

then, $\mathsf{Pr}\{G(Z) \in F_q(a)[1/2-\epsilon', 1/2+\epsilon']\} \geq 1-\delta$ . We will use a slight generalization of this property here.

For $0 < a < b < 1$, let $K_q(a,b)$ be the Lipschitz constant for the probability function $G_q$ in the neighborhood $x \in G^{-1}[a,b]$.

**Lemma 3.** *Suppose* $s_2 = \Omega\left(\frac{(K_q(0.1,0.8))^2 q^2}{\epsilon^2}\right)$ *and* $1 - \frac{\epsilon'}{8q} \leq \alpha_v^+, \alpha_v^- \leq 1 + \frac{\epsilon'}{8q}$, *for each* $v = 1, 2, \ldots, s_2$. *Then,*

$$\mathsf{Pr}\{\mathit{Quantile}_{v=1}^{s_2}(Q, \{|Y_v|^q\}) \in F_{p,q}(A)[1-\epsilon, 1+\epsilon]\} \geq \frac{31}{32}.$$

*where, Q is chosen so that* $Q \in [0.25, 0.65]$.

*Proof.* Let $R$ be the interval $[(1-\epsilon')F_{p,q}(A), (1+\epsilon')F_{p,q}(A)]$. Since, $1-\epsilon'/(8q) \leq \alpha_v^+, \alpha_v^- \leq 1+\epsilon'/(8q)$, we have, $|F'_{p,v}(A_j) - F_p(A_j)| \leq \frac{\epsilon'}{(4q)}F_p(A_j)$ . Therefore,

$$\left|F'_{p,q,v}(A_j) - F_{p,q}(A)\right| \leq \frac{\epsilon'}{2}F_{p,q}(A) \tag{3}$$

Consider $Y_v \sim S\big(q, (F'_{p,q,v}(A_j))^{1/q}\big)$. Let $G$ denote the cumulative probability function of $S(q,1)$. Define $Z_v = \frac{Y_v}{(F'_{p,q,v})^{1/q}}$. Then, $Z_v \sim S(q,1)$ and

$$\mathsf{Pr}\left\{\frac{|Y_v|^q}{F'_{p,q,v}} \in [(G^{-1}(Q-\epsilon'))^q, (G^{-1}(Q+\epsilon'))^q]\right\} = 2\epsilon'. \tag{4}$$

It would be sufficient to require that $[(G^{-1}(Q-\epsilon'))^q, (G^{-1}(Q+\epsilon'))^q]$ has length at most $\epsilon/4$. That is

$$(G^{-1}(Q+\epsilon'))^q - (G^{-1}(Q-\epsilon'))^q \leq \epsilon/4.$$

This is implied by (using Taylor series expansion for $G^{-1}(Q \pm \epsilon)$ up to the first differential term)

$$\epsilon' = \frac{\epsilon}{8qG'(\xi)} \leq \frac{\epsilon}{8qK_q} \tag{5}$$

where, $\xi$ is the value that minimizes $G'(x)$ in the neighborhood $[Q-\epsilon', Q+\epsilon']$. With this condition, (4) becomes

$$\mathsf{Pr}\left\{\left|\frac{|Y_v^q|}{G^{-1}(Q)} - F_{p,q}(A)\right| \leq F_{p,q}(A)(2\epsilon' + \epsilon/2)\right\} \geq 2\epsilon' \tag{6}$$

A straightforward application of Chernoff's bound shows that if $s_2 = \Omega(\epsilon'^{-2})$, then,

$$\mathsf{Pr}\left\{\left|\text{Quantile}_{v=1}^{s_2}\left(Q, \frac{{Y_v}^q}{G^{-1}(Q)}\right) - F_{p,q}(A)\right| \leq F_{p,q}(A)(2\epsilon' + \epsilon/2)\right\} \geq 31/32.$$

□

We therefore have the following lemma. The notation $K_q(Q-\epsilon, Q+\epsilon)$ denotes the Lipschitz constant for the function $G^{-1}$ in the $\epsilon$-neighborhood of $Q$.

**Lemma 4.** *For each value of $0 < p < 1$ and $0 < q < 2$, there exists an algorithm that returns $\hat{F}_{p,q}$ satisfying $|\hat{F}_{p,q} - F_{p,q}| \geq 15/16$ using $O\left(\frac{K_q^2(Q-\epsilon,Q+\epsilon)K_p^2(Q+\epsilon,Q+\epsilon)}{\epsilon^4 p^2 q^2}\log\frac{1}{\epsilon' p'}\right)$ bi-linear stable sketches.*

*Proof.* We can obtain $1 - \frac{\epsilon'}{8q} \leq \alpha_v^+, \alpha_v^- \leq 1 + \frac{\epsilon'}{8q}$, for each $v = 1, 2, \ldots, s_2$, with total error probability of $1/32$, or, individual error probability of $1/(32s_2)$ provided,

$$s_1 = \Theta\left(\frac{(K_p(Q-\epsilon, Q+\epsilon))^2}{\epsilon'^2 p^2}\log\frac{1}{s_2}\right) \quad .$$

The number of sketches required is $s_1 \cdot s_2$. By Lemma 3, if $s_2 = \Theta(\frac{K_q(Q-\epsilon,Q+\epsilon)^2 q^2}{\epsilon^2})$, then, with the above choice of $s_1$, the premises of Lemma 3 are satisfied, with probability 31/32. By union bound, the probability of correct answer is $1 - 1/32 - 1/32 = 15/16$. This proves the lemma. □

We can now use a previously suggested technique of Indyk [12] to conclude that (1) each sketch can be represented using $O(\log(nF_{1,1}(A)))$ bits, and, (2) the random bits can be reduced to $O(S\log(n^2 S)$ using Nisan's pseudo-random generator that fools Turing machines using space $S$ bits. We thus obtain the following theorem.

**Theorem 1.** *For each value of $0 \leq p \leq 1$ and $0 \leq q \leq 2$, quantile value $0 < Q < 1$, and $0 < \epsilon < 1/5$, there exists an algorithm that returns $\hat{F}_{p,q}$ satisfying $|\hat{F}_{p,q} - F_{p,q}| \geq 15/16$ using $O(S(\log(nF_{1,1}(A)\log(nS\log(nF_{1,,1}(A)))$ bits, where,*
*$S = O\left(\frac{K_q^2(Q-\epsilon,Q+\epsilon)K_p^2(Q-\epsilon,Q+\epsilon)q^2}{\epsilon^4 p^2}\log\frac{1}{\epsilon p}\right)$ bilinear stable sketches.* □

*Lower Bounds.* The problem of estimating $F_{p,q}$ by reducing the problem of estimating the $pq$th one-dimensional moment $F_{p\cdot q}$ to $F_{p,q}$ as follows [19].[1] Consider an $n$-dimensional vector $a$ and view it as the first row of the $n \times n$ matrix $A$, the rest of whose entries are zeros. Then, by definition, $F_{p,q}(A) = F_{p\cdot q}(a)$. For $pq \in [0, 2]$, $F_{pq}$ has a lower bound of $\Omega(1/\epsilon^2)$ [20]. This implies that the bilinear stable sketches technique presented for the range $p \in [0, 1]$ and $q \in [0, 2]$ is close to optimal, up to polynomial factors in $1/\epsilon$.

[1] The author thanks David Woodruff for pointing this out.

## 4 Conclusion

We present a novel technique for estimating the hybrid frequency moment $F_{p,q}(A)$ of an $n \times n$ dimensional matrix $A$ whose entries are updated by a data stream. Our technique requires space $\tilde{O}(q^2/(p^2\epsilon^4)$ and is nearly space-optimal. The problem of obtaining matching upper or lower bounds for the problem is left open.

## References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating frequency moments. J. Comp. Sys. and Sc. 58(1), 137–147 (1998)
2. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. In: Proceedings of ACM STOC, Princeton, NJ, pp. 209–218 (2002)
3. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D., Trevisan, L.: Counting distinct elements in a data stream. In: Rolim, J.D.P., Vadhan, S.P. (eds.) RANDOM 2002. LNCS, vol. 2483, pp. 1–10. Springer, Heidelberg (2002)
4. Bhuvanagiri, L., Ganguly, S.: Estimating entropy over data streams. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 148–159. Springer, Heidelberg (2006)
5. Chakrabarti, A., Khot, S., Sun, X.: Near-Optimal Lower Bounds on the Multi-Party Communication Complexity of Set Disjointness. In: Proceedings of International Conference on Computational Complexity (CCC), Aarhus, Denmark (2003)
6. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002)
7. Cormode, G., Muthukrishnan, S.: An Improved Data Stream Summary: The Count-Min Sketch and its Applications. J. Algorithms 55(1), 58–75 (2005)
8. Cormode, G., Muthukrishnan, S.: Space Efficient Mining of Multigraph Streams. In: Proceedings of ACM International Symposium on Principles of Database Systems (PODS) (2005)
9. Flajolet, P., Martin, G.N.: Probabilistic Counting Algorithms for Database Applications. J. Comp. Sys. and Sc. 31(2), 182–209 (1985)
10. Ganguly, S., Bansal, M., Dube, S.: Estimating hybrid frequency moments of data streams. In: Preparata, F.P., Wu, X., Yin, J. (eds.) FAW 2008. LNCS, vol. 5059, pp. 55–66. Springer, Heidelberg (2008)
11. Gibbons, P.B., Tirthapura, S.: Estimating simple functions on the union of data streams. In: Proceedings of ACM SPAA, Heraklion, Crete, Greece, pp. 281–291 (2001)
12. Indyk, P.: Stable Distributions, Pseudo Random Generators, Embeddings and Data Stream Computation. In: Proceedings of IEEE FOCS, pp. 189–197 (2000)
13. Indyk, P., Woodruff, D.: Optimal Approximations of the Frequency Moments. In: Proceedings of ACM Symposium on Theory of Computing STOC, Baltimore, Maryland, USA, June 2005, pp. 202–298 (2005)
14. Li, P.: Very Sparse Stable Random Projections, Estimators and Tail Bounds for Stable Random Projections (manuscript) (2006)

15. Li, P.: Computationally Efficient Estimators for Dimension Reductions Using Stable Random Projections (June 2008) arXiv:0806.4422v1 [cs.LG]
16. McGregor, A.: Open Problem. In: Data Streams And Related Topics: IITK Workshop On Algorithms For Data Streams (2006), http://www.cse.iitk.ac.in/users/sganguly/openproblems.pdf
17. Nisan, N.: Pseudo-Random Generators for Space Bounded Computation. In: Proceedings of ACM Symposium on Theory of Computing STOC, May 1990, pp. 204–212 (1990)
18. Nolan, J.P.: Stable Distributions. ch. 1 (2006), http://academic2.american.edu/~jpnolan
19. Woodruff, D.P.: Personal Communication (August 2008)
20. Woodruff, D.P.: Optimal space lower bounds for all frequency moments. In: Proceedings of ACM Symposium on Discrete Algorithms (SODA), pp. 167–175 (2004)

# Two-Level Push-Relabel Algorithm for the Maximum Flow Problem

Andrew V. Goldberg

Microsoft Research – Silicon Valley
1065 La Avenida, Mountain View, CA 94062
goldberg@microsoft.com

**Abstract.** We describe a two-level push-relabel algorithm for the maximum flow problem and compare it to the competing codes. The algorithm generalizes a practical algorithm for bipartite flows. Experiments show that the algorithm performs well on several problem families.

## 1 Introduction

The maximum flow problem is classical combinatorial optimization problem with applications in many areas of science and engineering. For this reason, the problem has been studied both from theoretical and practical viewpoints for over half a century. The problem is to find a maximum flow from the source to the sink (a minimum cut between the source and the sink) given a network with arc capacities, the source, and the sink. We denote the number of vertices and arcs in the input network by $n$ and $m$, respectively. Some time bounds also depend on the maximum arc capacity $U$. When $U$ appears in the bound, we assume that the capacities are integral.

A theoretical line of research led to development of augmenting path, network simplex, blocking flow, and push-relabel methods and to a sequence of improved bounds; see [17] for a survey. Under the assumption $\log U = O(\log n)$ [12], the bound of $O(\min\{n^{2/3}, m^{1/2}\}m \log(n^2/m) \log U)$ is achieved by the binary blocking flow algorithm [15]. The best strongly polynomial bound of $O(nm \log_{m/(n \log n)} n)$ is achieved by the algorithm of [21].

From the practical point of view, good implementations of Dinitz blocking flow method [6,19] proved superior to the network simplex and the augmenting path algorithms. The blocking flow method remained the method of choice until the development of the push-relabel method [16]. For a long time, the HI-PR implementation [7] of the highest-level push-relabel method served as a benchmark for maximum flow algorithms. This implementation uses both the global update (see e.g. [13]) and gap [9] heuristics.

Mazzoni et al. [22] introduced a number of variations of the push-relabel method, including the partial augment-relabel method, which on the basis of limited experiments they claimed to be computationally superior. A variant of the method has been further studied in [24]. Recently, the author [18] developed an efficient implementation, PAR, of the algorithm and conducted more extensive experiments confirming that it outperforms HI-PR.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 212–225, 2009.

The partial augment-relabel algorithm moves flow along paths of $k$ arcs. In the efficient implementations the best value of $k$ is less then ten but greater than two. However, the idea of looking ahead or moving flow two steps has been used for several problems closely related to the maximum flow problem. In the context of the minimum-cost flow, [13] suggests an algorithm that pushes flow into a vertex that can push flow further without being relabeled. In the context of the assignment problem, an operation that moves a flow excess two levels at a time appears in [14]. For bipartite graphs, maximum flow algorithms that move flow excess two steps at a time and leave no excess on one side of the graph achieve better theoretical bounds [1] and practical performance [23].

The main contribution of this paper is a maximum flow algorithm that pushes flow on two adjacent levels of the network. The resulting algorithm is different from the partial augment-relabel algorithm with $k = 2$ and can be viewed as a generalization of the bipartite graph algorithm of [1] to general graphs. We present an efficient implementation, P2R, of this algorithm. Another contribution is an experimental evaluation of the algorithms, which compares P2R with several other codes, including a version of PAR that is an improved implementation of the algorithm used in [18]. The experiments show that P2R is comparable to PAR.

Our work makes progress towards unifying previous work on practical maximum flow algorithms and getting a better understanding of what works best in practice. The push-relabel method leads to practical algorithms for related problems, such as minimum-cost flows [13], assignment problem [14], bipartite matching [8] and parametric flows [2]. It is possible that some of the techniques discussed in this paper will lead to improved algorithms for some of the related problems as well.

## 2 Definitions and Notation

The input to the maximum flow problem is $(G, s, t, u)$, where $G = (V, A)$ is a directed graph, $s, t \in V$, $s \neq t$ are the *source* and the *sink*, respectively, $u : A \Rightarrow [1, \ldots, U]$ is the *capacity function*, and $U$ is the *maximum capacity*.

Let $a^R$ denote the *reverse* of an arc $a$, let $A^R$ be the set of all reverse arcs, and let $A' = A \cup A^R$. A function $g$ on $A'$ is *anti-symmetric* if $g(a) = -g(a^R)$. Extend $u$ to be an anti-symmetric function on $A'$, i.e., $u(a^R) = -u(a)$.

A flow $f$ is an anti-symmetric function on $A'$ that satisfies *capacity constraints* on all arcs and *conservation constraints* at all vertices except $s$ and $t$. The capacity constraint for $a \in A$ is $0 \leq f(a) \leq u(a)$ and for $a \in A^R$ it is $-u(a^R) \leq f(a) \leq 0$. The conservation constraint for $v$ is $\sum_{(u,v)\in A} f(u, v) = \sum_{(v,w)\in A} f(v, w)$. The *flow value* is the total flow into the sink: $|f| = \sum_{(v,t)\in A} f(v, t)$.

A *cut* is a partitioning of vertices $S \cup T = V$ with $s \in S, t \in T$. The capacity of a cut is defined by $u(S, T) = \sum_{v\in S, w\in T, (v,w)\in A} u(S, T)$. The max-flow, min-cut theorem [11] says that the maximum flow value is equal to the minimum cut capacity.

A *preflow* is a relaxation of a flow that satisfies capacity constraints and a relaxed version of conservation constraints $\sum_{(u,v)\in A} f(u,v) \geq \sum_{(v,w)\in A} f(v,w)$. We define the flow *excess* of $v$ by $e_f(v) = \sum_{(u,v)\in A} f(u,v) - \sum_{(v,w)\in A} f(v,w)$. For a preflow $f$, $e_f(v) \geq 0$ for all $v \in V \setminus \{s,t\}$.

The *residual capacity* of an arc $a \in A'$ is defined by $u_f(a) = u(a) - f(a)$. Note that if $f$ satisfies capacity constraints, then $u_f$ is non-negative. The *residual graph* $G_f = (V, A_f)$ is the graph induced by the arcs in $A'$ with strictly positive residual capacity.

An *augmenting path* is an $s$–$t$ path in $G_f$.

A *distance labeling* is an integral function $d$ on $V$ that satisfies $d(t) = 0$. Given a preflow $f$, we say that $d$ is valid if for all $(v,w) \in E_f$ we have $d(v) \leq d(w) + 1$. Unless mentioned otherwise, we assume that a distance labeling is valid with respect to the current preflow in the graph.

We say that an arc $(v,w)$ is *admissible* if $(v,w) \in A_f$ and $d(w) < d(v)$, and denote the set of admissible arcs by $A_d$.

## 3 The Push-Relabel Method

The push-relabel method maintains a preflow and a distance labeling, which are modified using two *basic operations*:

**Push**$(v,w)$ applies if $e_f(v) > 0$ and $(v,w) \in A_d$. It chooses $\delta : 0 < \delta \leq \min\{u_f(v,w), e_f(v)\}$, increases $f(v,w)$ and $e_f(w)$ by $\delta$ and decreases $e_f(v)$ and $f((v,w)^R)$ by $\delta$. A push is *saturating* if after the push $u_f(v,w) = 0$ and *non-saturating* otherwise.

**Relabel**$(v)$ applies if $d(v) < n$ and $v$ has no outgoing admissible arcs. A relabel operation increases $d(v)$ to the maximum value allowed: $1 + \min_{(v,w)\in A_f} d(w)$ or to $n$ if $v$ has no outgoing residual arcs.

The method can start with any feasible preflow and distance labeling. Unless mentioned otherwise, we assume the following simple initialization: $f$ is zero on all arcs except for arcs out of $s$, for which the flow is equal to the capacity; $d(s) = n$, $d(t) = 0$, $d(v) = 1$ for all $v \neq s,t$. For a particular application, one may be able to improve algorithm performance using an application-specific initialization. After initialization, the method applies push and relabel operations until no operation is applicable.

When no operation applies, the set of all vertices $v$ such that $t$ is reachable from $v$ in $G_f$ defines a minimum cut, and the excess at the sink is equal to the maximum flow value. For applications that need only the cut, the algorithm can terminate at this point. For applications that need the maximum flow, we run the second stage of the algorithm.

One way to implement the second stage is to first reduce flow around flow cycles to make the flow acyclic, and then to return flow excesses to the source by reducing arc flows in the reverse topological order with respect to this acyclic graph. See [25]. Both in theory and in practice, the first stage of the algorithm dominates the running time.

The *current arc* data structure is important for algorithm efficiency. Each vertex maintains a current arc $a(v)$. Initially, and after each relabeling of $v$, the arc is the first arc on $v's$ arc list. When we examine $a(v)$, we check if it is admissible. If not, we advance $a(v)$ to the next arc on the list. The definition of basic operations implies that only relabeling $v$ can create new admissible arcs out of $v$. Thus as $a(v)$ advances, arcs behind it on the list are not admissible. When the arc advances past the end of the list, $v$ has no outgoing admissible arcs and therefore can be relabeled. Thus the current arc data structure allows us to charge to the next relabel operation the searches for admissible arcs to apply push operations to.

### 3.1 HI-PR Implementation

Next we review the HI-PR implementation [7] of the push-relabel algorithm. It uses highest-label selection rule, and global update and gap heuristics. We say that a vertex $v \neq s, t$ is *active* if $d(v) < n$ and $e_f > 0$.

The method uses a *layers of buckets* data structure. Layers correspond to distance labels. Each layer $i$ contains two buckets, *active* and *inactive*. A vertex $v$ with $d(v) = i$ is in one of these buckets: in the former if $e_f(v) > 0$ and in the latter otherwise. Active buckets are maintained as a singly linked list and support insert and extract-first operations. Inactive buckets are maintained as doubly linked lists and support insert and delete operations. The layer data structure is an array of records, each containing two pointers – to the active and the inactive buckets of the layer. A pointer is `null` if the corresponding bucket is empty. We refer to the active and inactive buckets of layer $i$ by $\alpha_i$ and $\beta_i$, respectively.

To implement the highest-label selection, we maintain the index $\mu$ of the highest layer with non-empty active bucket. The index increases if an active vertex is inserted into a layer higher than the current value of $\mu$, which can happen during a relabel operation.

At each step of the algorithm, we examine $\alpha_\mu$. If it is empty, we decrease $\mu$ or terminate if $\mu = 0$. Otherwise, let $v$ be the first active vertex of $\alpha_\mu$. We look for an admissible arc out of $v$. If such an arc $(v, w)$ is found, we push flow along this arc. In addition to changing $f$, the push can have two side-effects. First, $e_f(w)$ may change from zero to a positive value, making $w$ active. We delete $w$ from $\beta_{d(w)}$ and insert it into $\alpha_{d(w)}$. Second, $e_f(v)$ can decrease to zero, making it inactive. We extract $v$ from the head of $\alpha_{d(v)}$ and insert it into $\beta_{d(v)}$. If no admissible arc out of $v$ exists, we relabel $v$. This increases $d(v)$. We extract $v$ from the head of $\alpha_{d'(v)}$, where $d'(v)$ is the old distance label of $v$, and insert it into $\alpha_{d(v)}$. In this case we also increase $\mu$ to $d(v)$. Then we proceed to the next step.

The gap heuristic [10] is based on the following observation. Suppose for $0 < i < n$, no vertex has a distance label of $i$ but some vertices $w$ have distance labels $j : i < j < n$. The validity of $d$ implies that such $w$'s cannot reach $t$ in $G_f$ and can therefore be deleted from the graph until the end of the first phase of the algorithm.

Layers facilitate the implementation of the gap heuristic. We maintain the invariant that the sequence of non-empty layers (which must start with layer zero containing $t$) has no gaps. During relabeling, we check if a gap is created, i.e., if increasing distance label of $v$ from its current value $d(v)$ makes both buckets in layer $d(v)$ empty. If this is the case, we delete $v$ and all vertices at the higher layers from the graph, restoring the invariant, and set $\mu = d(v) - 1$. Note that deleting a vertex takes constant time, and the total cost of the gap heuristic can be amortized over the relabel operations.

Push and relabel operations are local. On some problem classes, the algorithm substantially benefits from the *global relabeling* operation. This operation performs backwards breadth-first search from the sink in $G_f$, computing exact distances to the sink and placing vertices into appropriate layer buckets. Vertices that cannot reach the sink are deleted from the graph until the end of the first phase. Global update places the remaining vertices in the appropriate buckets and resets their current arcs to the corresponding first arcs. HI-PR performs global updates after $O(m)$ work has been done by the algorithm; this allows amortization of global updates.

The following improvements to the implementation of global relabeling have been proposed in [18]: (i) incremental restart, (ii) early termination, and (iii) adaptive amortization. Suppose flows on arcs at distance $D$ or less have not change. The incremental restart takes advantage of this fact: We can start the update from layer $D$ as lower layers are already in breadth-first order. This change can be implemented very efficiently as the only additional information we need is the value of $D$, which starts at $n$ after each global update, and is updated to $min(d(w), D)$ each time we push flow to a vertex $w$. The early termination heuristic stops breadth-first search when all vertices active immediately before the global update have been placed in their respective layers by the search.

With incremental restart and early termination, global updates sometimes cost substantially less than the time to do breadth-first search of the whole graph, and an amortization strategy can be used to trigger a global update. Our new implementation of PAR uses a threshold that is different from that used in [18]. Every time we do a global update, we set the threshold $T$ to $T = S + C$ where $S$ is the number of vertices scanned during the global update and $C$ is a constant that represents the cost of calling the global update routine. The next global update is performed when $WF > T$, where $W$ is the number of vertex scans since the last global update and $F$ is the global update frequency parameter. In our experiments we use $C = 500$ and $F = 0.2$.

Note that buckets have constant but non-trivial overhead. For instances where heuristics do not help, the buckets slow the code down by a constant factor. When the heuristics help, however, the improvement can be asymptotic.

### 3.2 PAR Implementation

The *partial augment-relabel (PAR)* algorithm is a push-relabel algorithm that maintains a preflow and a distance labeling. The algorithm has a parameter $k$. At each step, the algorithm picks an active vertex $v$ and attempts to find an

admissible path of $k$ vertices starting at $v$. If successful, the algorithm executes $k$ push operations along the path, pushing as much flow as possible. Otherwise, the algorithm relabels $v$.

PAR looks for augmenting paths in the depth-first manner. It maintains a current vertex $x$ (initially $v$) with an admissible path from $v$ to $x$. To extend the path, the algorithm uses the current arc data structure to find an admissible arc $(x, y)$. If such an arc exists, the algorithm extends the path and makes $y$ the current vertex. Otherwise the algorithm shrinks the path and relabels $x$. The search terminates if $x = t$, or the length of the path reaches $k$, or $v$ is the current vertex and $v$ has no outgoing admissible arcs.

As in the push-relabel method, we have the freedom to choose the next active vertex to process. Our PAR implementation uses layers and highest-level selection. The gap heuristic is identical to that used in HI-PR. After experimenting with different values of $k$ we used $k = 4$ in all of our experiments. Results for $2 \leq k \leq 6$ would have been similar.

Note that HI-PR relabels only active vertices currently being processed, and as a side-effect we can maintain active vertices in a singly-linked list. PAR can relabel other vertices as well, and we may have to move an active vertex in the middle of a list into a higher-level list. Therefore PAR uses doubly-linked lists for active as well as inactive vertices. List manipulation becomes slower, but the overall effect is very minor. No additional space is required as the inactive list is doubly-linked in both implementations and a vertex is in at most one list at any time.

Our new implementation of PAR includes two optimizations. The first optimization is to use regular queue-based breadth-first search (instead of the incremental breadth-first search described in Section-3.1) for the first global update. This is because the first update usually looks at the majority of the vertices, and the queue-based implementation is faster. The second optimization is a more careful implementation of the augment operation that, when flow is pushed into and then out of a vertex, moves the vertex between active and inactive lists only if the vertex status changed from the time before the first push to the time after the second one. The effects of these optimizations are relatively minor, but noticeable on easy problems, which includes some practical vision instances and some of the DIMACS problems.

## 4 The P2R Algorithm

The *two-level push-relabel algorithm (P2R)* at each step picks a vertex $u$ to process and applies the *two-level push* operation to it. Although $u$ can be any active vertex, the implementation discussed in this paper uses the highest label selection. It uses the same data structures and heuristics as PAR.

The two-level push operation works as follows. Using the current arc data structure, we examine admissible arcs out of $u$. If $(u, v)$ is such an arc, and $v$ is the sink, we push flow on $(u, v)$. If $v$ is not a sink, we make sure $v$ has an outgoing admissible arc, and relabel $v$ if it does not.

When we find an admissible arc $(u, v)$ such that $v$ has an outgoing admissible arc, we consider two cases. If $v$ is active, push the maximum possible amount from $u$ to $v$, and then push flow on admissible arcs out of $v$ until either $v$ is no longer active or $v$ has no outgoing admissible arcs. In the latter case, relabel $v$.

If $v$ has no excess, we proceed in a way that avoids activating $v$, i.e., we do not push to $v$ more flow than $v$ can push out along its admissible arcs. To keep the same asymptotic complexity, we do it in such a way that the work can be amortized over distance label increases of $v$. First, we set $\delta = \min(u_f(u, v), e_f(u))$ and set $\Delta = \delta + e_f(v)$. Then we compute the amount $C$ that $v$ can push along its admissible arcs as follows. We start with $C = 0$ and examine arcs of $v$ starting from the current arc. Each time we see and admissible arc $(v, w)$, we add $u_f(v, w)$ to $C$. We stop either when we reach the end of the arc list of $v$ or when $C \geq \Delta$.

In the former case, we push $C - e_f(v)$ units of flow on $(u, v)$, push $C$ units of flow out of $v$, and relabel $v$. Note that the push on $(u, v)$ may move less than $\delta$ units of flow and therefore neither get rid of the excess at $u$ nor saturate $(u, v)$. However, we can charge the work of this push to the relabeling of $v$.

In the latter case, we push $\delta$ units of flow on $(u, v)$ and then push $\delta$ units of flow out of $v$ on admissible arcs starting from the current arc of $v$ and advancing the current arc to the last arc used for pushing flow. Note that while doing so, we examine the same arcs as we did when computing $C$. Therefore the work involved in computing $C$ is amortized over the current arc advances.

P2R does something different from a generic push-relabel algorithm in two places. First, a push from $u$ to $v$ may move less flow. Second, extra work is involved in computing $C$. As has been mentioned above, non-standard work can be amortized over other work done by the algorithm. Therefore generic push-relabel bounds apply to P2R. We also believe that the $O(n^2\sqrt{m})$ bound for the highest-label push-relabel algorithm [5,26] can be matched but omit details due to the lack of space.

Note that if we apply P2R to a bipartite graph, the algorithm will maintain the invariant that except in the middle of the two-level push operation, all excess is on one side of the network. In this sense, the algorithm generalizes the bipartite flow algorithm of [1].

## 5 Experimental Results

We test code performance on DIMACS problem families [20] (see also [19]) and on problems from vision applications.[1] We use RMF-Long, RMF-Wide, Wash-Long, Wash-Wide, Wash-Line, and Acyc-Dense problem families. To make sure that the performance is not affected by the order in which the input arcs are listed, we do the following. First, we re-number vertex IDs at random. Next, we sort arcs by the vertex IDs. The vision problems have been made available at `http://vision.csd.uwo.ca/maxflow-data/` and include instances from stereo vision, image segmentation, and multiview reconstruction.

[1] Due to space restrictions we omit problem descriptions. See the references.

The main goal of our experiments is to compare P2R to PAR and HI-PR. We use the latest version, 3.6, of HI-PR, version 0.43, of PAR and version 0.45 of P2R. We also make a comparison to an implementation of Chandran and Hochbaum [4]. A paper describing this implementation is listed on authors' web sites as "submitted for publication" and no preprint is publicly available. The authors do make their code available, and gave several talks claiming that the code performs extremely well. These talks were about version 3.1 of their code, which we refer to as CH. Recently, an improved version, 3.21, replaced the old version on the web site. We also compare to this version, denoted as CH-n. Finally, for vision problem we compare to the code BK of Boykov and Kolmogorov [3]. As code is intended for vision applications and does not work well on the DIMACS problems, we restrict the experiments to the vision problems.

Our experiments were conducted on an HP Evo D530 machine with 3.6 HGz Pentium 4 processor, 28 KB level 1 and 2 MB level 2 cache, and 2GB of RAM. The machine was running Fedora 7 Linux. C codes HI-PR, PAR, P2R, BK, and CH-n were compiled with the gcc compiler version 4.1.2 using "-O4" optimization option. C++ code CH was compiled with the g++ compiler using "-O4" optimization.

For synthetic problems, we report averages over 10 instances for each problem size. In all tables and plots, we give running time in seconds. For our algorithms, we also give scan count per vertex, where the scan count is the sum of the number of relabel operations and the number of vertices scanned by the global update operations. This gives a machine-independent measure of performance.

### 5.1 Experiments with DIMACS Families

Figures 1 – 6 give performance data for the DIMACS families. We discuss the results below.

*P2R vs. PAR vs. HI-PR.* First we note that P2R and PAR outperform HI-PR, in some cases asymptotically so (e.g., RMF and Acyc-Dense families).

Compared to each other, P2R and PAR performance is very similar. The latter code is faster more often, but by a small amount. The biggest difference is on the Acyc-Dense problem family, where P2R is faster by roughly a factor of 1.5.

Note that for P2R and PAR, many DIMACS problem families are easy. For the biggest Wash-Long, Wash-Line, and Acyc-Dense problems, these codes perform less than two scans per vertex, and for RMF-Long – around six. Note that Acyc-Dense are highly structured graphs, and the fact that P2R does about 50% fewer operations than PAR is probably due to the problem structure and is not very significant.

The "wide" instances are harder, but not too hard. For Wash-Wide problems, the number of scans per vertex grows slowly with the problem size, and stops growing between the two largest problem sizes. Even for the largest problems with over eight million vertices, the number of scans per vertex is around 25. RMF-Wide problems are the hardest and the number of scans per vertex slowly

grows with the problem size, but even for the largest problem with over four million vertices, the number of scans per vertex is in the 70's.

*Comparison with CH.* On RMF-Long problem families, the CH codes are asymptotically slower, and CH-n is somewhat faster than CH. On the RMF-Wide family, CH is significantly slower than CH-n. Performance of both codes is asymptotically worse than that of P2R and PAR. In particular, for larger problem the CH codes exhibit very large variance from one instance to another, which is not the case for the push-relabel codes. CH is always slower than P2R and PAR, losing by an order of magnitude on the largest problem. CH-n is faster than P2R and PAR by about a factor of 1.3 on the smallest problem, but slower by about a factor of 3.4 on the largest one.

On Wash-Long problem family, P2R and PAR are faster than the CH codes, although the latter are within a factor of two. It looks like the CH codes are asymptotically slower, but the difference in the growth rates are small. On Wash-Wide problem family, CH and CH-n are the fastest codes, but P2R and PAR never lose by much more than a factor of two. On Wash-Line problems, the four codes are within a factor of two of each other, but while P2R and PAR show clear linear growth rate, CH and CH-n running times grow erratically with the problem size.

On Acyc-Dense problems, where P2R is about 1.5 times faster than PAR, CH and CH-n performance falls in the middle of the push-relabel codes: the CH codes are almost as fast as P2R for the smallest problem and about as fast as PAR for the largest one.

### 5.2 Vision Instances

Stereo vision problems three problem sequences: tsukuba has 16, sawtooth – 20, venus – 22 subproblems. As suggested in [3], we report the total time for each problem sequence (Table 1), but for operation counts we report the average over subprobelms for each sequence (Table 2). On these problems, BK is the fastest code by a large margin. PAR is about a factor of six slower, and P2R is 10–20% slower than PAR. However, PAR does improve on HI-PR by a factor of two to three. CH performance is extremely poor; it loses to BK by over three orders of magnitude. CH-n is faster than PAR and P2R, but by less than a factor of two.

Operation counts show that PAR and P2R perform between 8 and 14 scans per vertex on the stereo problems. This suggests that BK performance is not too far from optimal, as the push-relabel codes need to reduce the number of scans per vertex to about two in order to catch up with BK.

On multiview instances, PAR and P2R perform similarly to each other, and about a factor of two better than HI-PR. The comparison of these codes to BK is non-conclusive. We consider four multiview instances: camel and gargoyle, each in two sizes, small and medium. BK is faster than P2R/PAR on camel, by about a factor of four on the smaller problem and by less than a factor of two on the larger one. BK is slower on gargoyle instances by a factor of four to six.

CH crashes on multiview problems. CH-n fails feasibility and optimality self-checks on the smaller problems and fails to allocate sufficient memory for the larger problems.

Next we consider segmentation instances. PAR and P2R performance is very close, and two to three times better than that of HI-PR. CH-n is faster by less than a factor of two – much less for liver and babyface instances. BK is competitive with PAR and P2R on most problems except for liver-6-10 and babyface-6-10, where BK is about 2.5 and 4 times faster, respectively. CH crashes on the segmentation problems.

## 6 Concluding Remarks

We introduce the two-level push-relabel algorithm, P2R, and show that its performance is close to that of our improved implementation of PAR and superior to that of HI-PR. The new algorithm is also significantly better than the CH code and better overall than the CH-n code. For vision problems, P2R is worse than the BK algorithm for stereo problems, but competitive for multiview and segmentation problems.

An interesting question is why P2R and PAR perform better than HI-PR. Partial intuition for this is as follows. The motivation behind PAR given in [22] is that it mitigates the ping-pong effect (e.g., a push from $u$ to $v$ immediately followed by a relabeling of $v$ and a push back to $u$). In PAR, a flow is pushed along the partial augmenting path before any vertex can push the flow back. In P2R, the flow pushed from $u$ to $v$ is pushed to $v$'s neighbors before $v$ would push flow back to $u$. In addition, P2R avoids activating some vertices, which has a theoretical motivation in the bipartite case [1] and seems to help in general.

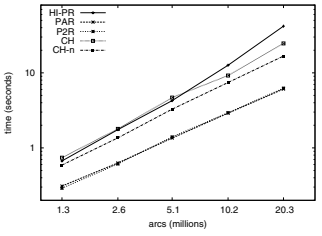
The main idea behind P2R is to push flow on length two paths, and not to create new excesses in the middle of such paths. This idea generalizes several previous algorithms. It would be interesting to investigate this idea in related contexts, such as minimum-cost flow, bipartite matching, assignment, and parametric flow problems.

## References

1. Ahuja, R.K., Orlin, J.B., Stein, C., Tarjan, R.E.: Improved algorithms for bipartite network flow problems. SIAM J. Comp. (to appear)
2. Babenko, M., Derryberry, J., Goldberg, A.V., Tarjan, R.E., Zhou, Y.: Experimental evaluation of parametric maximum flow algorihtms. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 256–269. Springer, Heidelberg (2007)
3. Boykov, Y., Kolmogorov, V.: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. IEEE transactions on Pattern Analysis and Machine Intelligence 26(9), 1124–1137 (2004)
4. Chandran, B., Hochbaum, D.: A computational study of the pseudoflow and push-relabel algorithms for the maximum flow problem (submitted for publication) (2007)
5. Cheriyan, J., Maheshwari, S.N.: Analysis of Preflow Push Algorithms for Maximum Network Flow. SIAM J. Comput. 18, 1057–1086 (1989)

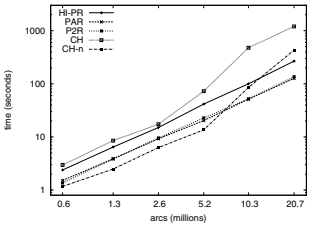
6. Cherkassky, B.V.: A Fast Algorithm for Computing Maximum Flow in a Network. In: Karzanov, A.V. (ed.) Collected Papers, Combinatorial Methods for Flow Problems, vol. 3, pp. 90–96. The Institute for Systems Studies, Moscow (1979); English translation appears in AMS Trans. 158, 23–30 (1994) (in Russian)
7. Cherkassky, B.V., Goldberg, A.V.: On Implementing Push-Relabel Method for the Maximum Flow Problem. Algorithmica 19, 390–410 (1997)
8. Cherkassky, B.V., Goldberg, A.V., Martin anbd, P., Setubal, J.C., Stolfi, J.: Augment or push? a computational study of bipartite matching and unit capacity flow algorithms. Technical Report 98-036R, NEC Research Institute, Inc. (1998)
9. Derigs, U., Meier, W.: Implementing Goldberg's Max-Flow Algorithm — A Computational Investigation. ZOR — Methods and Models of Operations Research 33, 383–403 (1989)
10. Derigs, U., Meier, W.: An Evaluation of Algorithmic Refinements and Proper Data-Structures for the Preflow-Push Approach for Maximum Flow. In: ASI Series on Computer and System Sciences, vol. 8, pp. 209–223. NATO (1992)
11. Ford, L.R., Fulkerson, D.R.: Maximal Flow Through a Network. Canadian Journal of Math. 8, 399–404 (1956)
12. Gabow, H.N.: Scaling Algorithms for Network Problems. J. of Comp. and Sys. Sci. 31, 148–168 (1985)
13. Goldberg, A.V.: An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. J. Algorithms 22, 1–29 (1997)
14. Goldberg, A.V., Kennedy, R.: An Efficient Cost Scaling Algorithm for the Assignment Problem. Math. Prog. 71, 153–178 (1995)
15. Goldberg, A.V., Rao, S.: Beyond the Flow Decomposition Barrier. J. Assoc. Comput. Mach. 45, 753–782 (1998)
16. Goldberg, A.V., Tarjan, R.E.: A New Approach to the Maximum Flow Problem. J. Assoc. Comput. Mach. 35, 921–940 (1988)
17. Goldberg, A.V.: Recent developments in maximum flow algorithms. In: Arnborg, S. (ed.) SWAT 1998. LNCS, vol. 1432, pp. 1–10. Springer, Heidelberg (1998)
18. Goldberg, A.V.: The Partial Augment–Relabel Algorithm for the Maximum Flow Problem. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 466–477. Springer, Heidelberg (2008)
19. Goldfarb, D., Grigoriadis, M.D.: A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow. Annals of Oper. Res. 13, 83–123 (1988)
20. Johnson, D.S., McGeoch, C.C.: Network Flows and Matching: First DIMACS Implementation Challenge. In: AMS, Proceedings of the 1-st DIMACS Implementation Challenge (1993)
21. King, V., Rao, S., Tarjan, R.: A Faster Deterministic Maximum Flow Algorithm. J. Algorithms 17, 447–474 (1994)
22. Mazzoni, G., Pallottino, S., Scutella, M.G.: The Maximum Flow Problem: A Max-Preflow Approach. Eur. J. of Oper. Res. 53, 257–278 (1991)
23. Negrus, C.S., Pas, M.B., Stanley, B., Stein, C., Strat, C.G.: Solving maximum flow problems on real world bipartite graphs. In: Proc. 11th International Workshop on Algorithm Engineering and Experiments, pp. 14–28. SIAM, Philadelphia (2009)
24. Iossa, A., Cerulli, R., Gentili, M.: Efficient Preflow Push Algorithms. Computers & Oper. Res. 35, 2694–2708 (2008)
25. Sleator, D.D., Tarjan, R.E.: A Data Structure for Dynamic Trees. J. Comput. System Sci. 26, 362–391 (1983)
26. Tuncel, L.: On the Complexity of Preflow-Push Algorithms for Maximum-Flow Problems. Algorithmica 11, 353–359 (1994)

## Appendix: Experimental Data



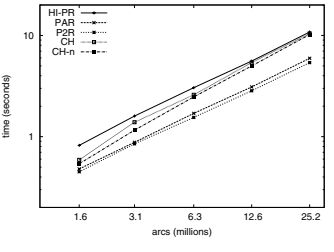

| n | 0.3 M | 0.5 M | 1.1 M | 2.0 M | 4.1 M |
|---|---|---|---|---|---|
| m | 1.3 M | 2.6 M | 5.1 M | 10.2 M | 20.3 M |
| HI-PR | 8.16 | 11.25 | 13.25 | 20.04 | 33.97 |
| PAR | 5.20 | 5.32 | 5.47 | 5.86 | 5.92 |
| P2R | 5.10 | 5.40 | 5.95 | 6.19 | 6.11 |

**Fig. 1.** RMF-Long problem data: time (plot), scans/vertex (table)



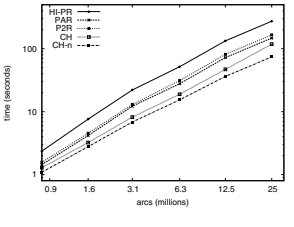

| n | 0.1 M | 0.3 M | 0.5 M | 1.0 M | 2.1 M | 4.2 M |
|---|---|---|---|---|---|---|
| m | 0.6 M | 1.3 M | 2.6 M | 5.2 M | 10.3 M | 20.7 M |
| HI-PR | 41.39 | 52.93 | 60.90 | 76.55 | 88.64 | 102.43 |
| PAR | 44.19 | 50.57 | 55.54 | 58.74 | 69.39 | 76.28 |
| P2R | 41.44 | 47.64 | 53.32 | 60.12 | 64.20 | 72.51 |

**Fig. 2.** RMF-Wide problem data: time (plot), scans/vertex (table)



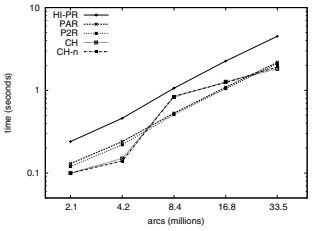

| HI-PR sc/n | 3.98 | 3.62 | 3.26 | 2.68 | 2.55 |
|---|---|---|---|---|---|
| PAR sc/n | 2.71 | 2.12 | 2.02 | 1.66 | 1.56 |
| P2R sc/n | 2.60 | 2.30 | 1.93 | 1.69 | 1.55 |

**Fig. 3.** Wash-Long problem data: time (plot), scans/vertex (table)



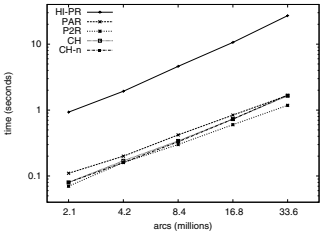

| n | 0.13 M | 0.26 M | 0.52 M | 1.0 M | 2.1 M | 4.2 M | 8.4 M |
|---|---|---|---|---|---|---|---|
| m | 0.4 M | 0.8 M | 1.6 M | 3.1 M | 6.3 M | 12.5 M | 25.0 M |
| HI-PR | 16.20 | 19.99 | 25.51 | 30.46 | 32.32 | 39.92 | 39.22 |
| PAR | 8.72 | 12.00 | 14.14 | 18.5 | 19.15 | 24.61 | 23.96 |
| P2R | 9.20 | 12.68 | 15.10 | 18.84 | 20.38 | 26.28 | 25.47 |

**Fig. 4.** Wash-Wide problem family: time (plot), scans/vertex (table)




| n | 41 K | 66 K | 104 K | 165 K | 262 K |
|---|---|---|---|---|---|
| m | 2.1 M | 4.2 M | 8.4 M | 16.8 M | 33.5 M |
| HI-PR sc/n | 2.07 | 2.06 | 2.05 | 2.05 | 2.04 |
| PAR sc/n | 1.13 | 1.11 | 1.10 | 1.09 | 1.07 |
| P2R sc/n | 1.08 | 1.07 | 1.06 | 1.05 | 1.04 |

**Fig. 5.** Wash-Line problem data: time (plot), scans/vertex (table)




| n | 2.0 K | 2.9 K | 4.1 K | 5.8 K | 8.2 K |
|---|---|---|---|---|---|
| m | 2.1 M | 4.2 M | 8.4 | 16.8 M | 33.6 M |
| HI-PR sc/n | 8.81 | 9.12 | 10.50 | 11.25 | 12.39 |
| PAR sc/n | 1.87 | 1.81 | 1.93 | 1.91 | 1.93 |
| P2R sc/n | 1.31 | 1.48 | 1.33 | 1.32 | 1.33 |

**Fig. 6.** Acyclic-Dense problem data: time (plot), scans/vertex (table)

**Table 1.** Stereo vision data – running times. Problems ordered by size $(n + m)$.

| name | n | m | HI-PR | PAR | P2R | CH | CH-n | BK |
|---|---|---|---|---|---|---|---|---|
| BVZ-tsukuba | 110,594 | 513,467 | 8.07 | 3.60 | 3.91 | 643.62 | 2.80 | 0.59 |
| BVZ-sawtooth | 164,922 | 796,703 | 12.45 | 6.81 | 8.00 | 3,127.23 | 5.45 | 1.01 |
| BVZ-venus | 166,224 | 795,296 | 23.65 | 10.19 | 11.02 | 2,707.32 | 7.23 | 1.86 |
| KZ2-tsukuba | 199,822 | 1,341,101 | 30.39 | 11.81 | 13.09 | 4,020.49 | 6.85 | 1.82 |
| KZ2-sawtooth | 294,936 | 1,956,194 | 31.88 | 14.57 | 16.69 | 13,472.85 | 11.70 | 2.77 |
| KZ2-venus | 301,610 | 2,026,283 | 61.64 | 21.31 | 26.75 | 12,898.89 | 15.84 | 4.49 |

**Table 2.** Stereo vision data – operation counts

| name | n | m | HI-PR | PAR | P2R |
|---|---|---|---|---|---|
| BVZ-tsukuba | 110,594 | 513,467 | 10.37 | 8.15 | 9.67 |
| BVZ-sawtooth | 164,922 | 796,703 | 14.73 | 9.55 | 10.84 |
| BVZ-venus | 166,224 | 795,296 | 19.34 | 11.87 | 13.01 |
| KZ2-tsukuba | 199,822 | 1,341,101 | 10.84 | 6.63 | 8.05 |
| KZ2-sawtooth | 294,936 | 1,956,194 | 20.96 | 12.34 | 13.29 |
| KZ2-venus | 301,610 | 2,026,283 | 20.03 | 9.65 | 12.51 |

**Table 3.** Multiview reconstruction – running times

| name | n | m | HI-PR | PAR | P2R | CH | CH-n | BK |
|---|---|---|---|---|---|---|---|---|
| gargoyle-sml | 1,105,922 | 5,604,568 | 4.39 | 2.72 | 2.28 | dnf | dnf | 12.29 |
| camel-sml | 1,209,602 | 5,963,582 | 8.50 | 4.14 | 4.41 | dnf | dnf | 1.14 |
| gargoyle-med | 8,847,362 | 44,398,548 | 125.03 | 47.5 | 52.32 | dnf | dnf | 193.58 |
| camel-med | 9,676,802 | 47,933,324 | 159.53 | 67.03 | 72.41 | dnf | dnf | 43.61 |

**Table 4.** Multiview reconstruction – operation counts

| name | n | m | HI-PR | PAR | P2R |
|---|---|---|---|---|---|
| gargoyle-sml | 1,105,922 | 5,604,568 | 8.93 | 8.61 | 6.77 |
| camel-sml | 1,209,602 | 5,963,582 | 15.91 | 12.61 | 13.14 |
| gargoyle-med | 8,847,362 | 44,398,548 | 29.25 | 19.95 | 19.49 |
| camel-med | 9,676,802 | 47,933,324 | 33.76 | 25.33 | 24.83 |

**Table 5.** Segmentation data – running times. Instances sorted by size $(n + m)$.

| name | $n$ | $m$ | HI-PR | PAR | P2R | CH | CH-n | BK |
|---|---|---|---|---|---|---|---|---|
| bone-xyzx-6-10 | 491,522 | 2,972,389 | 1.06 | 0.35 | 0.34 | dnf | 0.20 | 0.28 |
| bone-xyzx-6-100 | 491,522 | 2,972,389 | 1.08 | 0.37 | 0.35 | dnf | 0.25 | 0.33 |
| bone-xyz-6-10 | 983,042 | 5,929,493 | 2.39 | 0.82 | 0.80 | dnf | 0.50 | 0.85 |
| bone-xyz-6-100 | 983,042 | 5,929,493 | 2.46 | 0.85 | 0.86 | dnf | 0.58 | 1.32 |
| bone-xyzx-26-10 | 491,522 | 12,802,789 | 2.88 | 0.90 | 0.93 | dnf | 0.60 | 0.85 |
| bone-xyzx-26-100 | 491,522 | 12,802,789 | 3.16 | 0.93 | 0.89 | dnf | 0.58 | 1.01 |
| bone-xy-6-10 | 1,949,698 | 11,759,514 | 6.65 | 1.80 | 1.94 | dnf | 1.19 | 1.77 |
| bone-xy-6-100 | 1,949,698 | 11,759,514 | 6.90 | 1.95 | 2.01 | dnf | 1.36 | 3.10 |
| bone-xyz-26-10 | 983,042 | 25,590,293 | 6.36 | 2.01 | 1.99 | dnf | 1.20 | 2.52 |
| bone-xyz-26-100 | 983,042 | 25,590,293 | 6.75 | 2.13 | 2.10 | dnf | 1.39 | 3.4 |
| liver-6-10 | 4,161,602 | 25,138,821 | 34.88 | 18.73 | 20.77 | dnf | 14.18 | 7.59 |
| liver-6-100 | 4,161,602 | 25,138,821 | 46.74 | 20.32 | 21.87 | dnf | 17.59 | 19.68 |
| babyface-6-10 | 5,062,502 | 30,386,370 | 52.55 | 27.88 | 31.58 | dnf | 22.77 | 6.90 |
| babyface-6-100 | 5,062,502 | 30,386,370 | 71.37 | 28.74 | 33.70 | dnf | 37.85 | 15.15 |

**Table 6.** Segmentation data – operation counts

| name | $n$ | $m$ | HI-PR | PAR | P2R |
|---|---|---|---|---|---|
| bone-xyzx-6-10 | 491,522 | 2,972,389 | 3.97 | 2.42 | 2.28 |
| bone-xyzx-6-100 | 491,522 | 2,972,389 | 4.03 | 2.47 | 2.33 |
| bone-xyz-6-10 | 983,042 | 5,929,493 | 4.63 | 2.75 | 2.62 |
| bone-xyz-6-100 | 983,042 | 5,929,493 | 4.76 | 2.88 | 2.82 |
| bone-xyzx-26-10 | 491,522 | 12,802,789 | 3.96 | 2.36 | 2.37 |
| bone-xyzx-26-100 | 491,522 | 12,802,789 | 4.45 | 2.38 | 2.27 |
| bone-xy-6-10 | 1,949,698 | 11,759,514 | 5.96 | 2.89 | 2.91 |
| bone-xy-6-100 | 1,949,698 | 11,759,514 | 6.01 | 3.08 | 2.93 |
| bone-xyz-26-10 | 983,042 | 25,590,293 | 4.59 | 2.72 | 2.67 |
| bone-xyz-26-100 | 983,042 | 25,590,293 | 4.83 | 2.82 | 2.79 |
| liver-6-10 | 4,161,602 | 25,138,821 | 13.95 | 14.80 | 14.07 |
| liver-6-100 | 4,161,602 | 25,138,821 | 17.88 | 15.81 | 14.69 |
| babyface-6-10 | 5,062,502 | 30,386,370 | 20.78 | 20.17 | 19.03 |
| babyface-6-100 | 5,062,502 | 30,386,370 | 26.75 | 20.14 | 19.66 |

# A More Relaxed Model for Graph-Based Data Clustering: $s$-Plex Editing

Jiong Guo*, Christian Komusiewicz**, Rolf Niedermeier, and Johannes Uhlmann***

Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{jiong.guo,c.komus,rolf.niedermeier,johannes.uhlmann}@uni-jena.de

**Abstract.** We introduce the $s$-Plex Editing problem generalizing the well-studied Cluster Editing problem, both being NP-hard and both being motivated by graph-based data clustering. Instead of transforming a given graph by a minimum number of edge modifications into a disjoint union of cliques (Cluster Editing), the task in the case of $s$-Plex Editing is now to transform a graph into a disjoint union of so-called $s$-plexes. Herein, an $s$-plex denotes a vertex set inducing a (sub)graph where every vertex has edges to all but at most $s$ vertices in the $s$-plex. Cliques are 1-plexes. The advantage of $s$-plexes for $s \geq 2$ is that they allow to model a more relaxed cluster notion ($s$-plexes instead of cliques), which better reflects inaccuracies of the input data. We develop a provably efficient and effective preprocessing based on data reduction (yielding a so-called problem kernel), a forbidden subgraph characterization of $s$-plex cluster graphs, and a depth-bounded search tree which is used to find optimal edge modification sets. Altogether, this yields efficient algorithms in case of moderate numbers of edge modifications.

## 1 Introduction

The purpose of a clustering algorithm is to group together a set of (many) objects into a relatively small number of clusters such that the elements inside a cluster are highly similar to each other whereas elements from different clusters have low or no similarity. There are numerous approaches to clustering and "there is no clustering algorithm that can be universally used to solve all problems" [16]. To solve data clustering, one prominent line of attack is to use graph theory based methods [14]. In this line, extending and complementing previous work on cluster graph modification problems, we introduce the new edge modification problem $s$-Plex Editing.

In the context of graph-based clustering, data items are represented as vertices and there is an edge between two vertices iff the interrelation between the

* Partially supported by the DFG, Emmy Noether research group PIAF, NI 369/4, and research project DARE, GU 1023/1.
** Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.
*** Supported by the DFG, research project PABI, NI 369/7.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 226–239, 2009.

two corresponding items exceeds some threshold value. Clustering with respect to such a graph then means to partition the vertices into sets where each set induces a dense subgraph (that is, a *cluster*) of the input graph whereas there are no edges between the vertices of different clusters. In this scenario, the algorithmic task then typically is to transform the given graph into a so-called cluster graph by a minimum number of graph modification operations [14]. Herein, a *cluster graph* is a graph where all connected components form clusters and a graph modification is to insert or delete an edge. One of the most prominent problems in this context is the NP-hard Cluster Editing problem (also known as Correlation Clustering) [14, 2], where, given a graph $G$ and an integer $k \geq 0$, one wants to transform $G$ into a graph whose connected components all are cliques, using at most $k$ edge insertions and deletions. In this work, with the NP-hard $s$-Plex Editing problem, we study a more relaxed and often presumably more realistic variant of Cluster Editing: Whereas in the case of Cluster Editing the clusters shall be cliques, in the case of $s$-Plex Editing we only demand them to be $s$-plexes. A vertex subset $S \subseteq V$ of a graph $G = (V, E)$ is called *$s$-plex* if the minimum vertex degree in the induced subgraph $G[S]$ is at least $|S| - s$. Note that a clique is nothing but a 1-plex. Replacing cliques by $s$-plexes for some integer $s \geq 2$ allows one to reflect the fact that most real-world data are somewhat "spurious" and so the demand for cliques may be overly restrictive in defining what a cluster shall be (also see [5] concerning criticism of the overly restrictive nature of the clique concept).

**Problem formulation.** In the following, we call a graph an *$s$-plex cluster graph* if all its connected components are $s$-plexes.

$s$-Plex Editing
**Input:** An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
**Question:** Can $G$ be modified by up to $k$ edge deletions and insertions into an $s$-plex cluster graph?

Indeed, seen as an optimization problem, the goal is to minimize the number of edge editing operations. Note that 1-Plex Editing is the same as Cluster Editing. Compared to Cluster Editing, $s$-Plex Editing with $s \geq 2$ is a more flexible tool for graph-based data clustering: For increasing $s$, the number of edge modifications should decrease. This important advantage of $s$-Plex Editing reflects the observation that fewer edge modifications mean that we introduce fewer "errors" into our final cluster solution, because the computed $s$-plex cluster graph is closer to the original data. This is in accordance with the natural hypothesis that the less one perturbs the input graph the more robust and plausible the achieved clustering is (maximum parsimony principle, also see Böcker et al. [3] for making this point in terms of Cluster Editing). Figure 1 presents a simple example comparing Cluster Editing (that is, 1-Plex Editing) with 2-Plex Editing and 3-Plex Editing in terms of the (number of) necessary editing operations.

**Previous work and motivation.** The $s$-plex concept was introduced in 1978 by Seidman and Foster [13] in the context of social network analysis. Recently, a

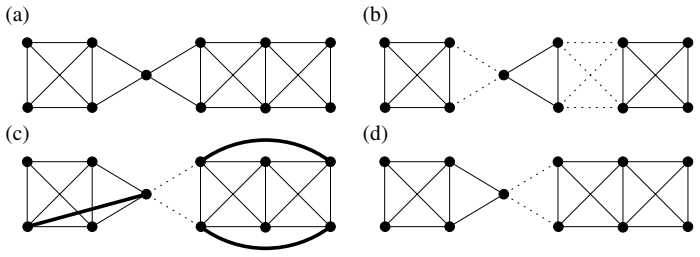



**Fig. 1.** An example for different optimal modifications that are applied to (a) an input graph using (b) Cluster Editing (equivalently, 1-Plex Editing), (c) 2-Plex Editing, and (d) 3-Plex Editing. Deleted edges are dashed, inserted edges are bold.

number of theoretical and experimental studies explored (and confirmed) the usefulness of $s$-plexes in various contexts [1, 6, 10, 11]. Finding maximum-cardinality $s$-plexes is NP-hard [1] and further hardness results in analogy to clique finding hold as well [10]. Hence, there is no hope for polynomial-time algorithms.

Cluster Editing has recently been intensively studied from the viewpoints of polynomial-time approximability as well as parameterized algorithmics. As to approximability, the currently best known approximation factor is 2.5 [17]. Considering the parameter $k$ defined as the number of allowed edge modifications, a search tree of size $O(1.83^k)$ [3] has been developed and several studies concerning provably efficient and effective preprocessing by data reduction (which is called problem kernelization in the context of parameterized algorithmics [12]) have been performed [7, 8]. Parameterized algorithms have led to several successful experimental studies mainly in the context of biological network analysis [3, 4]. The parameterized algorithms only run fast in case of moderate values of the parameter $k$, the number of allowed edge editing operations. Hence, it is desirable to have the parameter $k$ small not only for the sake of not too much perturbing the input graph but also for the sake of obtaining efficient solving algorithms. We mention in passing that slightly modifying a proof of Shamir et al. [14] for Cluster Editing one can show that $s$-Plex Editing is NP-complete for each specific choice of $s$ as well.

**Our contributions.** We develop a polynomial-time preprocessing algorithm that allows to provably simplify input instances of $s$-Plex Editing to smaller ones. More specifically, the corresponding data reduction rules, given an instance $(G = (V, E), k)$ of $s$-Plex Editing with $s \geq 2$, in polynomial time construct an equivalent reduced instance $(G' = (V', E'), k')$ with $V' \subseteq V$, $k' \leq k$, and $|V'| \leq (4s^2-2)\cdot k+4(s-1)^2$. In other words, the number of vertices of the reduced graph only depends on $s$ and $k$ (in fact, in case of $s$ being a constant, it is linear in $k$), implying that if $k$ is small then the data reduction will greatly simplify the instance basically without loosing information. In terms of parameterized algorithmics, the reduced instance gives a problem kernel. Moreover, we provide a graph-theoretic characterization of $s$-plex cluster graphs by means of forbidden induced subgraphs. In particular, we obtain a linear-time recognition algorithm

for $s$-plex cluster graphs for every constant $s$. This is a result of independent graph-theoretic interest and is also of decisive algorithmic use for clustering: Based on the forbidden subgraph characterization of $s$-plex cluster graphs, we show that $s$-Plex Editing can be solved in $O((2s + \lfloor\sqrt{s}\rfloor)^k \cdot s \cdot (|V| + |E|))$ time (which is linear for constant values of $s$ and $k$). Moreover, interleaving the problem kernelization and the search tree leads to a running time of $O((2s + \lfloor\sqrt{s}\rfloor)^k + |V|^4)$.

Due to the lack of space, many technical details are deferred to a full version of this paper.

## 2 Preliminaries

We only consider *undirected* graphs $G = (V, E)$, where $n := |V|$ and $m := |E|$. The *(open) neighborhood* $N_G(v)$ of a vertex $v \in V$ is the set of vertices that are adjacent to $v$ in $G$. The *degree* of a vertex $v$, denoted by $\deg_G(v)$, is the cardinality of $N_G(v)$. For a set $U$ of vertices, $N_G(U) := \bigcup_{v \in U} N_G(v) \setminus U$. We use $N_G[v]$ to denote the *closed* neighborhood of $v$, that is, $N_G[v] := N_G(v) \cup \{v\}$. For a set of vertices $V' \subseteq V$, the *induced subgraph* $G[V']$ is the graph over the vertex set $V'$ with edge set $\{\{v, w\} \in E \mid v, w \in V'\}$. For $V' \subseteq V$ we use $G - V'$ as an abbreviation for $G[V \setminus V']$ and for a vertex $v \in V$ let $G - v$ denote $G - \{v\}$. A vertex $v \in V(G)$ is called a *cut-vertex* if $G - v$ has more connected components than $G$.

Parameterized algorithmics [12] aims at a multivariate complexity analysis of problems without giving up the demand for finding optimal solutions. This is undertaken by studying relevant problem parameters and their influence on the computational hardness of problems. The hope lies in accepting the seemingly inevitable combinatorial explosion for NP-hard problems, but confining it to the parameter. Hence, the decisive question is whether a given parameterized problem is *fixed-parameter tractable (FPT)* with respect to a parameter $k$. In other words, one asks for the existence of a solving algorithm with running time $f(k) \cdot \text{poly}(n)$ for some computable function $f$. A core tool in the development of parameterized algorithms is polynomial-time preprocessing by *data reduction rules*, often yielding a *problem kernel* [9, 12]. Herein, the goal is, given any problem instance $G$ with parameter $k$, to transform it in polynomial time into a new instance $G'$ with parameter $k'$ such that the size of $G'$ is bounded from above by some function only depending on $k$, $k' \leq k$, and $(G, k)$ is a yes-instance iff $(G', k')$ is a yes-instance.

## 3 Data Reduction and Kernelization

In this section, we indicate that $s$-Plex Editing for $s \geq 2$ admits a problem kernel with $(4s^2 - 2) \cdot k + 4(s-1)^2$ vertices. Since $s$-Plex Editing is a generalization of Cluster Editing, the first idea coming to mind in order to achieve a linear kernelization is to adapt an approach developed by Guo [8]. However,

since the "critical clique" concept used there does not work for $s$-PLEX EDITING, we need a more sophisticated strategy; correspondingly, the accompanying mathematical analysis requires new tools.

The first data reduction rule is obvious and its running time is $O(n+m)$:

**Reduction Rule 1:** Remove connected components that are $s$-plexes from $G$.

Our problem kernelization consists of only one further, technically complicated data reduction rule. Roughly speaking, the idea behind this rule is that a data reduction can be performed if there is a vertex with a "dense local environment" that is only weakly connected to the rest of the graph. Unfortunately, the proof details are technical and require quite some mathematical machinery.

We start with explaining in more detail the purpose of introducing the second data reduction rule. Let $G_{\text{opt}}$ denote the $s$-plex cluster graph resulting from applying a solution $S$ with $|S| \leq k$ to the graph $G = (V, E)$, and let $K_1, \ldots, K_l$ be the $s$-plexes in $G_{\text{opt}}$. The vertex set $V$ can be partitioned into two subsets, namely, $X$, the set of vertices that are endpoints of the edges modified by $S$, and $Y := V \setminus X$. For each $s$-plex $K_i$, let $X_i := X \cap K_i$ and $Y_i := Y \cap K_i$. Clearly, $|X| \leq 2k$. To achieve a problem kernel with $O(k)$ vertices for constant $s$, it remains to bound $|Y|$. To this end, we use a function linear in $|X_i|$ to bound $|Y_i|$ for each $1 \leq i \leq l$. If $|Y_i| \leq (s-1) \cdot |X_i|$ or $|Y_i| \leq 2(s-1)$ for all $i$, then we are done; otherwise, we have at least one $s$-plex $K_i$ with $|Y_i| > \max\{(s-1) \cdot |X_i|, 2(s-1)\}$. Because the vertices in $Y_i$ are not affected by the edge modifications in $S$, the fact that $K_i$ is an $s$-plex implies that every vertex in $X_i$ is adjacent to at least $|Y_i| - s + 1$ vertices in $Y_i$ in the input graph $G$. With $|Y_i| > (s-1) \cdot |X_i|$, there has to be a vertex $u \in Y_i$ with $X_i \subseteq N_G(u)$ by the pigeonhole principle. Moreover, if $|Y_i| > 2(s-1)$, then every vertex in $Y_i$ has, in $G$, distance at most two to $u$: Suppose that this is not true. Let $x$ be a vertex in $Y_i$ with distance at least three to $u$. Then, since $K_i$ is an $s$-plex, we must have $|Y_i \setminus N_G[u]| \leq s-1$ (since otherwise $u$ would be non-adjacent to more than $s-1$ vertices) as well as $|N_G[u] \cap Y_i| \leq s-1$ (since otherwise $x$ would be non-adjacent to more than $s-1$ vertices—the vertices $N_G[u] \cap Y_i$ are non-adjacent to $x$ since $x$ has distance at least three to $u$), contradicting $|Y_i| > 2(s-1)$.

Let us summarize our findings: If we do not apply a second data reduction rule to $G$, then there can be arbitrarily large $s$-plexes $K_i$ in $G_{\text{opt}}$, in particular, $|Y_i| > \max\{(s-1) \cdot |X_i|, 2(s-1)\}$. However, then, there must be a vertex $u \in Y_i$ satisfying the following conditions:

C1. $X_i \subseteq N_G(u)$,
C2. $N_G(u) \subseteq K_i$,
C3. $|Y_i \setminus N_G[u]| \leq s-1$, and
C4. all vertices in $Y_i \setminus N_G[u]$ have distance two to $u$ in $G$.

Thus, if $|Y_i|$ is very large, then $|N_G[u]|$ is very large and we need a data reduction rule to reduce $N_G[u]$. This is exactly what the second rule does.

To simplify notation, let $\hat{s} = s - 1$ and write $N(u)$ and $N[u]$ for $N_G(u)$ and $N_G[u]$, respectively. Let $N_2(u)$ denote the set of vertices that have, in $G$, distance two to $u$. Further, we partition $N_2(u)$ into two sets, where the first

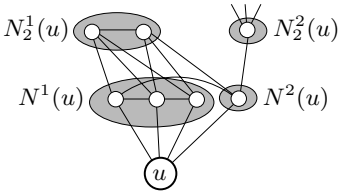



**Fig. 2.** An illustration of the partitions of $N_2(u)$ and $N(u)$ for $\hat{s} = 2$. Vertex $u$ satisfies the first two preconditions of Reduction Rule 2.

set $N_2^1(u)$ consists of vertices *tightly coupled* with $u$:

$$N_2^1(u) := \{v \in N_2(u) : \ |N(v) \cap N(u)| \geq |N[u]| - \hat{s}\},$$
$$N_2^2(u) := N_2(u) \setminus N_2^1(u).$$

Analogously, $N(u)$ is also partitioned into two sets,

$$N^1(u) := \{v \in N(u) : \big(N(v) \subseteq N[u] \cup N_2^1(u)\big) \wedge \big(|N[v]| \geq |N[u] \cup N_2^1(u)| - \hat{s}\,\big)\},$$
$$N^2(u) := N(u) \setminus N^1(u).$$

Figure 2 illustrates the above definitions. It is easy to see that the sets $N^1(u)$, $N^2(u)$, $N_2^1(u)$, and $N_2^2(u)$ can be computed in $O(n^2)$ time for any vertex $u$.

Following the above analysis, we need a data reduction rule which shrinks the set of the vertices tightly coupled with a vertex $u$ that has a very special neighborhood: There can be many vertices in $N^1(u)$, but only few (at most $\hat{s}$) tightly coupled vertices from $N_2(u)$, that is, the $N_2^1(u)$-vertices. Further, these $N_2^1(u)$-vertices are only adjacent to vertices in $N(u)$ or to $N_2^1(u)$-vertices. Reduction Rule 2 applies in this situation and replaces $N^1(u) \cup N_2^1(u)$ by smaller "simulating" cliques.

---

**Reduction Rule 2**
If there is a vertex $u$ for which
(1) $|N_2^1(u)| \leq \hat{s}$
(2) $\forall v \in N_2^1(u) : \ (N(v) \subseteq N(u) \cup N_2^1(u)) \wedge (|N[v]| \geq |N[u] \cup N_2^1(u)| - \hat{s})$, and
(3) $|A| > \alpha$, where $A := \{u\} \cup N^1(u) \cup N_2^1(u)$ and $\alpha := 2\,\hat{s} \cdot (|N^2(u)| + |N_2^2(u)| + \hat{s})$,

then replace $A$ by a clique $C$ with $\alpha$ vertices for even $|A|$ or $\alpha + 1$ vertices for odd $|A|$. Further, perform the following case distinction for every vertex $v \in N^2(u)$. Herein, for a vertex set $U$ and a vertex $w \notin U$, let $U_w := U \cap N(w)$ and $\overline{U_w} := U \setminus U_w$.

**Case 1.** If $|A_v| - |\overline{A_v}| \geq |N^2(u)| + |N_2^2(u)|$, then connect $v$ to $|C| - \min\{\hat{s}, |\overline{A_v}|\}$ many vertices of $C$ and decrease the parameter $k$ by $\max\{|\overline{A_v}| - \hat{s}, 0\}$.
**Case 2.** If $|\overline{A_v}| - |A_v| \geq |N^2(u)| + \hat{s}$, then decrease the parameter $k$ by $|A_v|$.
**Case 3.** If $|N^2(u)| + |N_2^2(u)| > |A_v| - |\overline{A_v}| > -|N^2(u)| - \hat{s}$, then insert edges between $v$ and the vertices in $C$ such that $|C_v| - |\overline{C_v}| = |A_v| - |\overline{A_v}|$ and decrease the parameter $k$ by $\max\{|A_v| - |C_v|, 0\}$.

---

To show the correctness of Reduction Rule 2, we need to prove that the input graph $G$ has a solution $S$ of size at most $k$ iff the graph $G'$ resulting by one application of this rule has a solution $S'$ of size at most $k'$, where $k'$ is the new parameter after the application of the rule. To this end, we need two claims. The first one (Lemma 1) says that if a vertex $u$ satisfies the three preconditions of Reduction Rule 2 then all vertices in $A$ as defined in the rule should be completely contained in one $s$-plex $K$ in the $s$-plex cluster graph generated by some optimal solution and $K \subseteq A \cup N^2(u)$. The second claim (Lemma 2) says that, after replacing $A$ by a clique $C$, all vertices in $C$ are tightly coupled with the vertices in $N^2(u)$. Using the second claim, we can show that there must be a vertex in $C$ satisfying the preconditions of the first claim. Thus, according to the first claim, there exists an optimal solution of $G'$ which generates an $s$-plex $K'$ with $C \subseteq K'$ and $K' \subseteq C \cup N^2(u)$. Then, by focussing on the edge modifications in $S$ and $S'$ that generate $K$ and $K'$, respectively, we can directly compare the sizes of $S$ and $S'$ and, thereby, establish the "iff"-relation between $G$ and $G'$.

**Lemma 1.** *Let $u$ be a vertex satisfying the first two preconditions of Reduction Rule 2 and $|A| \geq \alpha$ with $A$ and $\alpha$ defined as in Reduction Rule 2. Then, there exists always an optimal solution generating an $s$-plex cluster graph where*
*(1) the set $A$ is completely contained in one $s$-plex $K$ and*
*(2) $K \subseteq A \cup N^2(u)$.*

Now, before coming to the second claim (Lemma 2), we discuss in more detail the strategy behind. Based on Lemma 1, we can conclude that, with respect to a vertex $u$ which satisfies the preconditions of Reduction Rule 2, it remains to decide which vertices of $N^2(u)$ should build, together with $A$, an $s$-plex in the resulting $s$-plex cluster graph. Herein, Reduction Rule 2 distinguishes three cases. In the first two cases, a vertex $v \in N^2(u)$ has either much more or much less neighbors in $A$ than outside of $A$ (Cases 1 and 2). We can then easily decide whether $v$ should be in the same $s$-plex with $A$ (Case 1) or not (Case 2) and make the corresponding edge modifications. However, in the third case, where the "neighborhood size difference" is not so huge for a vertex $v \in N^2(u)$, the decision whether or not to put $v$ in the same $s$-plex with $A$ could be influenced by the global structure outside of $N(u) \cup N_2(u)$. To overcome this difficulty, Reduction Rule 2 makes use of the simulating clique $C$ which should play the same role as $A$ but has a bounded size. Moreover, for every vertex $v \in N^2(u)$ the construction of $C$ in Reduction Rule 2 guarantees that after its application $v$ again adheres to the same case (Cases 1–3, distinguishing according to the neighborhood size difference of $v$) as it has before. The second claim shows then that $C$ plays the same role as $A$.

**Lemma 2.** *Let $u$ be a vertex satisfying the three preconditions of Reduction Rule 2 and let $G'$ denote the graph resulting from applying Reduction Rule 2 once to $u$. Then, in $G'$, with the described implementation of Reduction Rule 2, each vertex in clique $C$ has at most $\hat{s}$ non-adjacent vertices in $N_{G'}(C)$.*

With these two claims, we can prove the correctness and the running time of Reduction Rule 2.

**Lemma 3.** *Reduction Rule 2 is correct and can be carried out in $O(n^3)$ time.*

Finally, we prove the main theorem in this section. Note that the running time upper bound is a pure worst-case estimation; improvements are conceivable.

**Theorem 1.** $s$-Plex Editing *admits a problem kernel with $(4s^2-2)\cdot k+4(s-1)^2$ vertices for $s \geq 2$. It can be computed in $O(n^4)$ time.*

*Proof.* Let $G_{\text{opt}}$ denote the $s$-plex cluster graph resulting from applying a solution $S$ with $|S| \leq k$ to the input graph $G = (V, E)$, and let $K_1, \ldots, K_l$ be the $s$-plexes in $G_{\text{opt}}$. The vertices $V$ of $G_{\text{opt}}$ can be partitioned into two subsets, namely, $X$, the set of vertices that are endpoints of the edges modified by $S$, and $Y := V \setminus X$. For an $s$-plex $K_i$, let $X_i := X \cap K_i$ and $Y_i := Y \cap K_i$. As stated in the beginning of this section, we know that $|X| \leq 2k$. Moreover, if $|Y_i| > \max\{\hat{s}\cdot|X_i|, 2\,\hat{s}\}$ for some $i$, then there must be a vertex $u \in Y_i$ that satisfies conditions C1–C4. By $N_2^1(u) \subseteq Y_i$ and $N[u] \subseteq K_i$, vertex $u$ fulfils the first two preconditions of Reduction Rule 2. Since $|Y_i| \leq |N^1(u) \cup N_2^1(u) \cup \{u\}|$, this implies either $|Y_i| \leq \alpha := 2\,\hat{s}\cdot(|N^2(u)| + |N_2^2(u)| + \hat{s})$ or Reduction Rule 2 can be applied to $u$. If we assume that the input graph is reduced with respect to both data reduction rules, then the former case applies. Note that $N^2(u) \cup N_2^2(u) \subseteq X$ and, for every deleted edge, each of its two endpoints in $X$ might be counted twice, once in $N^2(v)$ for a vertex $v \in K_i \cap Y$ and once in $N_2^2(w)$ for another vertex $w \in K_j \cap Y$ with $i \neq j$. Hence, considering all $s$-plexes, we then have

$$\sum_{1\leq i\leq l} |Y_i| \leq \sum_{1\leq i\leq l} \max\{2\,\hat{s}, \hat{s}\cdot|X_i|, 4\,\hat{s}\cdot(|X_i| + \hat{s}))\} \overset{(***)}{\leq} 8\,\hat{s}\,k + 4\,\hat{s}^2\cdot(k+1).$$

The inequality (***) follows from $|X| \leq 2k$ and the fact that deleting at most $k$ edges from a connected graph results in at most $k+1$ connected components. Together with $|X| \leq 2k$, we obtain a problem kernel with $|X| + |Y| \leq 4\,\hat{s}^2(k+1) + 8\,\hat{s}\,k + 2k = (4s^2-2)k + 4(s-1)^2$ vertices.

The running time $O(n^4)$ follows directly from Lemma 3 and the fact that Reduction Rule 2 can be applied at most $n$ times. □

## 4 Forbidden Subgraph Characterization and Search Tree

This section presents a forbidden subgraph characterization of $s$-plex cluster graphs for any $s \geq 1$ as well as an exact search tree algorithm that makes use of this characterization. We provide a characterization of $s$-plex cluster graphs by means of *induced* forbidden subgraphs. More specifically, we specify a set $\mathcal{F}$ of graphs such that a graph $G$ is an $s$-plex cluster graph iff $G$ is $\mathcal{F}$-*free*, that is, $G$ does not contain any induced subgraph from $\mathcal{F}$. If $s = 1$, where all connected components of the cluster graph are required to form cliques, the only forbidden subgraph is a path induced by three vertices [14]. By way of contrast, if $s \geq 2$, we face up to exponentially in $s$ many forbidden subgraphs. To cope with this, we develop a characterization of these subgraphs that still allows us to derive

**Input:** $G = (V, E)$ from $\mathcal{C}(s, i)$ with $i \cdot (i+1) > s$
**Output:** An induced subgraph $G' \in \mathcal{C}(s, i')$ of $G$ with $i' < i$
1 Let $v = \text{argmin}_{w \in V}\{\deg_G(w)\}$
2 **if** $\deg_G(v) < i$ **then**
3 **return** a connected graph induced by $N_G[v]$ and further $s$ arbitrary vertices
4 Let Cutvertices be the set of cut-vertices of $G$
5 **if** $(N_G(v) \setminus \text{Cutvertices}) \neq \emptyset$ **then**
6 **return** graph $G - w$ for an arbitrary $w \in (N_G(v) \setminus \text{Cutvertices})$
7 Let $N_G(v) = \{u_1, u_2, \ldots, u_i\}$
8 Let $U_j \subseteq V$ be the vertices not reachable from $v$ in $G - u_j$, for $1 \leq j \leq i$
9 Let $r = \text{argmin}_{j=1,\ldots,i}\{|U_j|\}$
10 **return** $G - (U_r \setminus \{w\})$ for an arbitrary vertex $w \in U_r$

**Fig. 3.** Algorithm A to compute smaller forbidden subgraphs

efficient algorithms. More specifically, we show that $s$-plex cluster graphs are characterized by forbidden subgraphs with $O(s)$ vertices and that if a graph is not an $s$-plex cluster graph then a forbidden subgraph can be found in $O(s \cdot (n+m))$ time.

The starting point for the forbidden subgraph characterization are the connected graphs that contain a vertex that is non-adjacent to $s$ vertices. These graphs clearly are no $s$-plex cluster graphs. Let $\mathcal{C}$ denote the set of connected graphs. Define $\mathcal{C}(s, i) := \{G = (V, E) \in \mathcal{C} \mid (|V| = s+1+i) \wedge (\exists v \in V : \deg_G(v) = i)\}$ and $\mathcal{F}(s, i) := \bigcup_{j=1}^{i} \mathcal{C}(s, j)$. The following lemma shows that the graphs in $\mathcal{F}(s, n-s-1)$ are forbidden.

**Lemma 4.** *A graph $G$ is an $s$-plex cluster graph iff $G$ is $\mathcal{F}(s, n-s-1)$-free.*

Next, we show that instead of studying graphs with $O(n)$ vertices, we can focus on graphs with $O(s)$ vertices by presenting an algorithm (Algorithm A, see Fig. 3) shrinking the size of large forbidden subgraphs. More precisely, we show that if the forbidden subgraph $G \in \mathcal{C}(s, i)$ with $i \cdot (i+1) > s$ then we can always remove at least one vertex from $G$ and still obtain a forbidden induced subgraph. For brevity, let $T_s$ be the maximum integer satisfying $T_s \cdot (T_s + 1) \leq s$, that is, $T_s = \lfloor -0.5 + \sqrt{0.25 + s} \rfloor$.

**Lemma 5.** *Given a graph $G = (V, E) \in \mathcal{C}(s, i)$ such that $i > T_s$, Algorithm A (Fig. 3) computes in $O(|V| + |E|)$ time an induced subgraph $G' \in \mathcal{C}(s, i')$ of $G$, with $i' < i$.*

*Proof.* Consider lines 1 to 3 of the algorithm. If a vertex $v$ in $G$ has degree less than $i$, then we can clearly find a graph from $\mathcal{C}(s, \deg_G(v))$ by choosing $N_G[v]$ and a set $S \subseteq V$ of $s$ further (arbitrary) vertices such that $G[N_G[v] \cup S]$ is connected. This is doable in linear time by breadth-first search starting at $v$.

Consider lines 4 to 6. If one of the neighboring vertices of $v$, say $w$, is no cut-vertex, then we can delete $w$ from $G$ obtaining a graph from $\mathcal{C}(s, i-1)$. Note that cut-vertices can be computed in linear time [15].

Consider lines 7 to 9. All neighboring vertices $N_G(v) = \{u_1, u_2, \ldots, u_i\}$ of $v$ are cut-vertices and the minimum vertex degree is $i$ with $i \cdot (i+1) > s$. On the one hand, note that $|U_j| \geq i$ for every $1 \leq j \leq i$ since the minimum vertex degree of $G$ is $i$ and since for every vertex $w \in U_j$ it holds that $N_G(w) \subseteq (U_j \cup \{u_j\}) \setminus \{w\}$. On the other hand, since $\sum_{j=1}^{i} |U_j| \leq s$, there must exist at least one $r$, $1 \leq r \leq i$, with $|U_r| \leq s/i < i \cdot (i+1)/i = i+1$. Therefore, $|U_r| = i$. Moreover, since the minimum vertex degree in $G$ is $i$, $U_r \cup \{u_r\}$ forms a clique of size $i+1$ and thus by deleting all but one vertex of $U_r$ we obtain a graph from $\mathcal{C}(s, 1)$. Note that Tarjan's algorithm [15] also computes a so-called *block-tree*. With the help of this data structure, the sets $U_j$ can be easily computed in linear time. □

We can iteratively use Algorithm A to compute an induced subgraph of $G'$ from $\mathcal{C}(s, i')$ with $i' \leq T_s$. This results in the following forbidden subgraph characterization that is—in contrast to the one of Lemma 4—tight concerning the number of vertices of the forbidden subgraphs.

**Theorem 2.** *A graph $G = (V, E)$ is an $s$-plex cluster graph if and only if $G$ is $\mathcal{F}(s, T_s)$-free.*

*Proof.* On the one hand, due to Lemma 4 we know that an $s$-plex cluster graph is $\mathcal{F}(s, n-s-1)$-free and, hence, $\mathcal{F}(s, T_s)$-free. On the other hand, if $G$ contains a forbidden subgraph from $\mathcal{C}(s, i')$ with $i' \cdot (i'+1) > s$, and, hence, according to Lemma 4 is not an $s$-plex cluster graph, then we can iteratively use Algorithm A (Figure 3) to find a forbidden subgraph from $\mathcal{C}(s, i)$ with $i \leq T_s$. □

To show our main result, we develop an $O(s \cdot (n+m))$-time algorithm (Algorithm B, see Figure 4) to find a forbidden subgraph from $\mathcal{F}(s, s)$. Since the number of vertices in such a subgraph is upper-bounded by $O(s)$, we can then apply Algorithm A iteratively ($O(s)$ times) to obtain a forbidden subgraph from $\mathcal{F}(s, T_s)$. Overall, this approach yields linear-time for any constant $s$.

**Lemma 6.** *Algorithm B (Figure 4) is correct and has running time $O(s \cdot (|V| + |E|))$.*

*Proof.* Consider lines 1 to 3. If $\deg_G(u) \leq s$, then we can clearly find a set $S \subseteq V$ of $s$ vertices such that $G[N_G[u] \cup S]$ is connected. This graph is in $\mathcal{C}(s, i')$ for an $i' \leq s$.

In the following, we need the observation that if one of the neighboring vertices of $v$ is a cut-vertex, then there exists at least one vertex in $G'$ with degree at most $s$. This can be seen as follows. Assume that $x \in N_G(v)$ is a cut-vertex and let $U \subseteq V$ denote the vertices not reachable from $v$ in $G - x$. Since a vertex $w \in U$ can only be adjacent to vertices in $U \cup \{x\}$ and $|U| \leq s$, we have that $\deg_G(w) \leq s$.

According to this observation, when entering line 5 of Algorithm B, we know that none of the vertices in $N_G(v) = \{u_1, u_2, \ldots, u_i\}$ is a cut-vertex. To make use of the observation, the remaining part of the algorithm is devoted to finding a set of vertices from $N_G(v)$ whose removal leads to a connected graph

**Input:** $G = (V, E)$ from $\mathcal{C}(s, i)$ with $i > s$
**Output:** An induced subgraph $G' \in \mathcal{C}(s, i')$ of $G$ with $i' \leq s$
1 Let $u = \text{argmin}_{w \in V'}\{\deg_G(w)\}$
2 **if** $\deg_G(u) \leq s$ **then**
3 **return** a connected graph induced by $N_G[u]$ and further $s$ arbitrary vertices
4 Let $v \in V'$ be a vertex with $\deg_G(v) = i$
5 Let $N_G(v) = \{u_1, u_2, \ldots, u_i\}$
6 Let $K = \{K_1, K_2, \ldots, K_l\}$ with $l \leq s$
denote the connected components of $G - N_G[v]$
7 Construct an auxiliary bipartite graph $H = (W_N, W_K, F)$ with
8 $W_N := \{w_{u_j} \mid 1 \leq j \leq i\}$,
9 $W_K := \{w_{K_q} \mid 1 \leq q \leq l\}$, and
10 $F := \{\{w_{u_j}, w_{K_q}\} \mid \exists\{u_j, v'\} \in E \text{ with } v' \in K_q\}$
11 Let $r := \arg\min_{q=1,\ldots,l}\{\deg_H(w_{K_q})\}$
12 Let $CC := \{u_j \mid w_{u_j} \in N_H(w_{K_r})\}$
13 Let $\hat{G} = G - (CC \setminus \{w\})$ for an arbitrary vertex $w \in CC$
14 Let $v' = \text{argmin}_{w \in V(\hat{G})}\{\deg_{\hat{G}}(w)\}$
15 **return** a connected graph induced by $N_{\hat{G}}[v']$ and further $s$ arbitrary vertices

**Fig. 4.** Algorithm B to compute in linear time a forbidden subgraph with $O(s)$ vertices

in which one neighbor of $v$ is a cut-vertex. To this end, one builds an auxiliary bipartite graph $H = (W_N, W_K, F)$ (lines 5-10). As to the running time needed for the construction of $H$, note that the degree of a vertex in $W_N$ is at most $s$ since $G - (N_G(v) \cup \{v\})$ contains exactly $s$ vertices and, hence, $W_K$ has size at most $s$. Thus, to construct $F$, we can iterate over the edge set $E$ and, given an edge $\{u_j, v'\}$ with $v' \in K_q$, we can decide in $O(s)$ time whether the edge $\{w_{u_j}, w_{K_q}\}$ is contained in $F$. Thus, the bipartite auxiliary graph $H$ can be constructed in $O(s \cdot (|V| + |E|))$ time.

Consider lines 11 to 13. By choosing a "component vertex" $w_{K_r}$ of minimum degree, we ensure that the set $CC$ is a minimum-cardinality set of vertices from $N_G(v)$ separating at least one connected component in $K$ from $v$. In particular, $CC$ separates the vertices in $K_r$ from $v$. Let $w$ be an arbitrary vertex of $CC$. By the deletion of all but one vertex from $CC$ (line 13), we ensure that the graph $\hat{G} = G - (CC \setminus \{w\})$ is still connected and contains at least one cut-vertex, namely $w$. Hence, according to the observation above, $\hat{G}$ contains a vertex of degree at most $s$. Let $v'$ be a minimum-degree vertex of $\hat{G}$ (line 14). As a consequence, $\deg_{\hat{G}}(v') \leq s$ and we can clearly find a set $S \subseteq V(\hat{G})$ of $s$ vertices such that $G' := \hat{G}[N_{\hat{G}}[v'] \cup S]$ is connected. Note that $G''$ is contained in $\mathcal{C}(s, \deg_{\hat{G}}(v')) \subseteq \mathcal{F}(s, s)$.

Altogether, the running time is $O(s \cdot (|V| + |E|))$. □

Summarizing, we obtain a linear-time algorithm for finding an induced forbidden subgraph if $s$ is a constant.

**Theorem 3.** *Let $G = (V, E)$ be a graph that is not an $s$-plex cluster graph. Then, a forbidden subgraph from $\mathcal{F}(s, T_s)$ can be found in $O(s \cdot (n + m))$ time.*

*Proof.* Let $C = (W, F)$ be a connected component of $G$ that is not an $s$-plex. Let $v$ be a vertex of minimum degree in $C$. Clearly, by breadth-first search starting at $v$ we can find a set $S \subseteq W$ of $s$ vertices such that $G' := G[N_G[v] \cup S]$ is connected. Note that $G' \in \mathcal{C}(s, \deg_{G'}(v))$. If $\deg_{G'}(v) > s$, then we can apply Algorithm $B$ (Figure 4) once to find an induced forbidden subgraph $G''$ from $\mathcal{F}(s, s)$. In order to find a forbidden subgraph from $\mathcal{F}(s, T_s)$, we apply Algorithm A (Figure 3) at most $O(s)$ times. □

Next, we present a search tree algorithm that is based on this forbidden subgraph characterization. To obtain an $s$-plex cluster graph, every forbidden subgraph has to be destroyed via edge modifications. To this end, we apply a branching strategy.

**Theorem 4.** *$s$-Plex Editing can be solved in $O((2s + \lfloor\sqrt{s}\rfloor)^k \cdot s \cdot (n + m))$ time.*

*Proof.* Given an instance $(G, k)$ of $s$-Plex Editing, we search in $G$ for a forbidden subgraph from $\mathcal{F}(s, T_s)$. By Theorem 3, this can be done in $O(s \cdot (n+m))$ time. If $G$ does not contain a subgraph from $\mathcal{F}(s, i)$, then $G$ already is an $s$-plex cluster graph and we are done. Otherwise, let $S$ be a set of vertices inducing a forbidden subgraph $G[S] \in \mathcal{C}(s, i') \subseteq \mathcal{F}(s, i)$, where $i' \leq T_s$. In the following, let $v$ denote a vertex with $\deg_{G[S]}(v) = i'$. By the definition of $\mathcal{C}(s, i')$, such a vertex must exist. We now branch into the different possibilities to destroy the forbidden subgraph $G[S]$ and then recursively solve the instances that are created in the respective search tree branches.

For branching, we either insert edges incident to $v$ or delete edges in $G[S]$. It is sufficient to only consider these edge modifications since, if none of these is performed, then $G[S]$ remains connected and there are $s$ vertices in $G[S]$ that are not adjacent to $v$, contradicting the $s$-plex (cluster graph) definition.

First, we consider edge insertions between $v$ and vertices $u \in S \setminus N[v]$. Since $G[S] \in \mathcal{C}(s, i')$ and $\deg_{G[S]}(v) = i'$, we have $|S \setminus N[v]| = s$. Therefore, we branch into $s$ cases, inserting a different edge in each search tree branch. The parameter decreases by 1 in each branch.

Besides this, we consider edge deletions. Hence, in each remaining branch, there is at least one vertex $u \in S$ such that $u$ and $v$ are not connected, that is, they are in different connected components of the final $s$-plex cluster graph. We now show that for each $u \in S$ we can create a search tree branch in which at least one edge deletion is performed for the case that $u$ and $v$ are not connected in the final cluster graph. Let $S_l \subset S$ denote the vertices that have distance exactly $l$ to $v$ in $G[S]$. We first consider the vertices in $S_1$ (the neighbors of $v$ in $G[S]$), then the vertices in $S_2$, and so on.

For each $u \in S_1$, we create a search tree branch in which we disconnect $u$ and $v$. Clearly this means that we have to delete the edge $\{u, v\}$. To branch on the vertices in $S_2$, we can assume that the vertices from $N[v] = \{v\} \cup S_1$ end up in the same cluster, since we have already considered all possibilities of removing edges between $v$ and the vertices in $S_1$. Therefore, when considering the case that a vertex $u \in S_2$ and $v$ are not connected in the final cluster graph,

we must delete all edges between $u$ and its neighbors in $S_1$. At least one such edge must exist because $u \in S_2$. Therefore, for each case, we create a search tree branch in which the parameter is decreased by at least 1.

The case distinction is performed for increasing values of $l$, always assuming that $v$ and the vertices in $S_1 \cup S_2 \cup \ldots \cup S_{l-1}$ end up in the same cluster of the final cluster graph. Hence, when considering the case that $v$ and a vertex $u \in S_l$ end up in different clusters, we create a search tree branch in which the edges between $u$ and its neighbors in $S_{l-1}$ are deleted, and at least one of these edges must exist. Hence, we create $|S|-1 = s+i' \leq s+T_s$ branches in which edges are deleted. Together with the $s$ cases in which edge insertions are performed, we branch into $2s+T_s$ cases, and in each branch, the parameter is decreased by at least 1. Branching is performed only as long as $k > 0$. The search tree thus has size $O((2s+T_s)^k) = O((2s+\lfloor\sqrt{s}\rfloor)^k)$. Using breadth-first search, the steps at each search tree node can be performed in $O(s \cdot (n+m))$ time which results in the claimed running time bound. □

Using Theorems 1 and 4, by interleaving the problem kernelization and the search tree [12], we get:

**Theorem 5.** *$s$-Plex Editing can be solved in $O((2s+\lfloor\sqrt{s}\rfloor)^k + n^4)$ time.*

## 5 Conclusion

We initiated the study of the graph modification problem $s$-Plex Editing. We believe that $s$-Plex Editing may have practical relevance for graph-based data clustering in a similar way as its well-studied special case Cluster Editing. Our results lead to numerous opportunities for future research. First, from the viewpoint of algorithm theory, we concentrated on parameterized algorithms, leaving open the study of approximation algorithms. Second, we left unstudied the sometimes desirable case of having a specified number of clusters to be generated. As to applications, important issues of interest for future study would be to deal with weighted inputs or to try to obtain faster algorithms for special cases such as $s = 2$. A thorough empirical study as recently undertaken for Cluster Editing [4] is a natural next step for future work.

*Acknowledgement.* We are grateful to Falk Hüffner for inspiring discussions in the early phase of this research.

## References

[1] Balasundaram, B., Butenko, S., Hicks, I.V., Sachdeva, S.: Clique relaxations in social network analysis: The maximum k-plex problem (manuscript, 2006)
[2] Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Machine Learning 56(1-3), 89–113 (2004)
[3] Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: Going weighted: Parameterized algorithms for cluster editing. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 1–12. Springer, Heidelberg (2008)

[4] Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: Evaluation and experiments. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 289–302. Springer, Heidelberg (2008)
[5] Chesler, E.J., Lu, L., Shou, S., Qu, Y., Gu, J., Wang, J., Hsu, H.C., Mountz, J.D., Baldwin, N.E., Langston, M.A., Threadgill, D.W., Manly, K.F., Williams, R.W.: Complex trait analysis of gene expression uncovers polygenic and pleiotropic networks that modulate nervous system function. Nature Genetics 37(3), 233–242 (2005)
[6] Cook, V.J., Sun, S.J., Tapia, J., Muth, S.Q., Argüello, D.F., Lewis, B.L., Rothenberg, R.B., McElroy, P.D., The Network Analysis Project Team: Transmission network analysis in tuberculosis contact investigations. Journal of Infectious Diseases 196, 1517–1527 (2007)
[7] Fellows, M.R., Langston, M.A., Rosamond, F.A., Shaw, P.: Efficient parameterized preprocessing for cluster editing. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 312–321. Springer, Heidelberg (2007)
[8] Guo, J.: A more effective linear kernelization for Cluster Editing. Theoretical Computer Science 410(8), 718–726 (2009)
[9] Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1), 31–45 (2007)
[10] Komusiewicz, C., Hüffner, F., Moser, H., Niedermeier, R.: Isolation concepts for enumerating dense subgraphs. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 140–150. Springer, Heidelberg (2007)
[11] Memon, N., Kristoffersen, K.C., Hicks, D.L., Larsen, H.L.: Detecting critical regions in covert networks: A case study of 9/11 terrorists network. In: Proc. 2nd ARES, pp. 861–870. IEEE Computer Society, Los Alamitos (2007)
[12] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications, vol. 31. Oxford University Press, Oxford (2006)
[13] Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. Journal of Mathematical Sociology 6, 139–154 (1978)
[14] Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Applied Mathematics 144(1–2), 173–182 (2004)
[15] Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)
[16] Xu, R., Wunsch II, D.: Survey of clustering algorithms. IEEE Transactions on Neural Networks 16(3), 645–678 (2005)
[17] van Zuylen, A., Williamson, D.P.: Deterministic algorithms for rank aggregation and other ranking and clustering problems. In: Kaklamanis, C., Skutella, M. (eds.) WAOA 2007. LNCS, vol. 4927, pp. 260–273. Springer, Heidelberg (2008)

# Dynamic Position Auctions with Consumer Search

Scott Duke Kominers*

Departments of Economics and Mathematics
Harvard University
Cambridge, MA, USA
kominers@fas.harvard.edu
skominers@gmail.com

**Abstract.** Building upon the static model of Athey and Ellison [1], we demonstrate the efficient convergence of dynamic position auctions in the presence of consumer search. The entry of low-quality advertisers does not slow this convergence. Our methods are extensions of those introduced by Cary *et al.* [2]. The applicability of these methods in the presence of consumer search indicates the robustness of the approach and suggests that convergence of dynamic position auction models is demonstrable whenever the associated static equilibrium strategies are sufficiently well-behaved.

**Keywords:** Position auctions, dynamic auctions, consumer search, best-response bidding, convergence.

## 1 Introduction

"Position auctions," the mechanisms search engines use to allocate sponsored search links, are often modeled as games in which advertisers submit bids and then are assigned positions in descending bid order. The utility of each advertiser then depends upon the advertiser's bid, his per-click valuation, and the click-through rate of his position.

Early position auction models, such as those of Aggarwal *et al.*[3], Edelman *et al.* [4], and Varian [5] assumed positions' click-through rates to be entirely exogenous. New approaches have introduced the effects of consumer behavior, as in the models of Chen and He [6] and Athey and Ellison [1]. Position auction models that include consumer search reflect the setting of sponsored search more accurately than do earlier models and also allow for the examination of consumer welfare. However, these newer models have only been studied in static settings.

* The author was partially supported by a grant from the Harvard College Program for Research in Science and Engineering. He is especially grateful to Susan Athey for supervising the work and for her commentary and support. He also thanks Zachary Abel, Yiling Chen, Peter Coles, Noam D. Elkies, Drew Fudenberg, Andrea Hawksley, Dmitry Taubinsky, several anonymous referees, and participants in the Harvard EconCS Research Workshop for helpful comments and suggestions.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 240–250, 2009.

By contrast, some of the earlier models of position auctions have been extended to dynamic settings. Most recently in this vein, Cary *et al.* [2] showed the convergence of a dynamic formulation of the static model of Edelman *et al.* [4].

In Sections 2.1 and 2.2, we model a dynamic position auction in the presence of consumer search. This model builds upon the static framework of Athey and Ellison [1]. Then, in Sections 2.3 and 2.4, we extend the methods of Cary *et al.* [2] to our framework, proving that the dynamic position auction with consumer search converges to a unique equilibrium when advertisers play a particular best-response bidding strategy. The contribution of this result is twofold: First, it assures that the addition of consumer behavior does not detract from the eventual stability of the dynamic position auction. Second, it demonstrates a surprising robustness of the approach used by Cary *et al.* [2]—similar analysis yields the convergence of position auctions in significantly different settings. In Section 3, we discuss the extent to which this apparent robustness is structural and indicates general facts about the convergence behavior of position auctions.

## 2 A Dynamic Model with Consumer Search

### 2.1 Underlying Framework

We consider an auction in which $N$ advertisers bid on $M < N$ sponsored link positions. Each advertiser $\pi$ has a quality score $q_\pi$, interpreted as the probability of meeting an individual consumer's need. We assume that the quality scores $q_\pi$ are drawn independently from a public, atomless distribution with support on $[0, 1]$. An advertiser receives a payoff of 1 every time it meets a consumer's need. This assumption does not mean that the advertisers have the same per-click valuations: the expected per-click revenue of advertiser $\pi$ is $q_\pi \cdot 1 = q_\pi$. Throughout, we label the advertisers $\pi$ by $\{1, \ldots, N\}$ so that the quality scores $q_\pi$ satisfy $q_1 > \cdots > q_N$.

At all times, there is a continuum of consumers seeking to meet their needs by searching through the sponsored link list. Consumers are assumed to be ignorant of the positions' dynamics, so that their beliefs are independent of the dynamics of advertisers' bid updating.[1] Additionally, consumers are assumed to believe that the advertisers' bidding strategies are strictly monotone in their qualities.[2]

Each consumer $i$ must pay a search cost of $c_i$ for each click on a sponsored search link. The search costs $c_i$ are assumed to be distributed according to a public, atomless distribution with support on $[0, 1]$ and CDF $G$. Since search is costly and consumers believe the links to be sorted in order of descending quality, they search in a top-down manner. Consumers update their predictions about advertisers' qualities in a Bayesian manner: when a website does not meet consumer $i$'s need, she

[1] A typical consumer will only use a search engine to seek a given product once, hence no consumer has an opportunity to learn about the positions' dynamics.

[2] This is a reasonable assumption for our purposes. Indeed, Proposition 5 of Athey and Ellison [1] shows that the static position auction game in this framework has a symmetric pure strategy monotone equilibrium. In our model this equilibrium the unique equilibrium of both the static game in each round and the dynamic game.

reduces her estimate of the lower websites' qualities and continues searching if and only if the expected value of clicking on the next link exceeds $c_i$.[3]

Formally, an assignment of advertisers to positions is an injection

$$\mathcal{P} : \{1, \ldots, M\} \hookrightarrow \{1, \ldots, N\}$$

such that advertiser $\mathcal{P}(j)$ is assigned position $j$. We suppose that the advertisers with quality scores $q^{\mathcal{P}}_{\pi_1}, \ldots, q^{\mathcal{P}}_{\pi_M}$ are respectively assigned positions $1, \ldots, M$ in some assignment $\mathcal{P}$ of the advertisers to positions. If $z^{\mathcal{P}}_{\pi_1}, \ldots, z^{\mathcal{P}}_{\pi_M}$ are Bernoulli variables taking the value 1 with these probabilities, then a searching consumer whose need has not been met by the advertisers in the first $j \geq 1$ positions expects that the quality of the firm in position $j+1$ is

$$\bar{q}^{\mathcal{P}}_{j+1} := E(q_{\pi_{j+1}} \mid z^{\mathcal{P}}_{\pi_1} = \cdots = z^{\mathcal{P}}_{\pi_j} = 0).$$

The expected probability of the advertiser in the first position meeting a consumer's need is always $\bar{q}^{\mathcal{P}}_1 := E(q_\pi)$. All consumers can compute this value, as the distribution of advertiser quality is assumed to be public. From these definitions, it is apparent that $\bar{q}^{\mathcal{P}}_j > \bar{q}^{\mathcal{P}}_{j+1}$ for any $1 \leq j \leq M$. With this setup, we may compute directly that the advertiser assigned position $j$ will receive $(1 - q^{\mathcal{P}}_{\pi_1}) \cdots (1 - q^{\mathcal{P}}_{\pi_{j-1}}) \cdot G(\bar{q}^{\mathcal{P}}_j)$ clicks.[4]

### 2.2 Auction Model and Bidding Dynamics

We assume a dynamic setting with sequential rounds of play $t = 1, 2, \ldots$. In each round $t > 0$, the search engine will allocate all its positions through a generalized second-price auction:[5] the advertiser submitting the $j$-th highest bid in round $t$ is assigned position $j$ and is charged a per-click price $p^t_j$ equal to the $(j+1)$-st highest bid submitted in round $t$.[6]

[3] Since consumers are unable to learn about the positions' dynamics, we assume that consumers are slightly myopic: an individual consumer will maintain her belief that the advertisers below position $j$ have lower qualities than do the advertisers in positions $1, \ldots, j$, even if she discovers while searching that the advertisers are not sorted in order of quality.

[4] See Proposition 2 of Athey and Ellison [1].

[5] The exact details of the implementation of such an auction in the presence of consumer search are specified by Athey and Ellison [1]. The mechanism is analogous to a second-price ascending bid auction for the $M$ positions.

[6] Note that this is a *rank-by-bid* mechanism: although payment is made per-click, advertisers are ranked in strict bid order. This is in contrast to *rank-by-revenue* mechanisms, in which advertisers are ranked after their bids are weighted by their ads' expected click-through rates. We have focused on rank-by-bid mechanisms for two reasons. First, to effectively model a dynamic rank-by-revenue mechanism, we would have to model fluctuations in advertisers' expected click-through rates, and click-through rate dynamics are not well-understood from either empirical or theoretical perspectives. Second, as examples presented by Athey and Ellison [1] illustrate, rank-by-revenue mechanisms render equilibrium bidding behavior unclear in the presence of consumer search, even in the static position auction setting.

For convenience, we denote

$$\bar{q}_j^t := \bar{q}_j^{\mathcal{P}^t}, \quad \bar{q}_j := \bar{q}_j^{\mathcal{P}^*},$$

where $\mathcal{P}^t$ is the assignment of positions in round $t$ and $\mathcal{P}^*$ is the assignment of advertisers to positions in order of descending valuation. With our assumption that $q_1 \geq \cdots \geq q_N$, this means that $q_{\pi_j}^{\mathcal{P}^*} = q_j$. We observe that, by construction, $\bar{q}_j \geq \bar{q}_j^{\mathcal{P}}$ for any assignment $\mathcal{P}$ and $1 \leq j \leq M$.

We assume that each advertiser $\pi$ plays a best-response strategy in each round $t > 0$, submitting a bid under the assumption that all other advertisers will repeat their bids from round $t-1$ in round $t$.[7] Under this assumption, we define a *restricted balanced bidding* strategy.

**Definition 1.** *The* restricted balanced bidding *(RBB) strategy is the strategy for advertiser $\pi$ which, given the bids of the other advertisers in round $t$,*

- *targets the position $s_\pi^*$ which maximizes the utility of advertiser $\pi$ among the positions with no higher expected click-through rate than her current position $s_\pi$,*
- *chooses the bid $b_\pi^*$ for round $t+1$ so as to satisfy the equation*

$$(1 - q_\pi)G(\bar{q}_{s_\pi^*}^t)(q_\pi - p_{s_\pi^*}^t) = G(\bar{q}_{s_\pi^*-1}^t)(q_\pi - b_\pi^*).^8 \tag{1}$$

*(We define $G(\bar{q}_0^t) := 2G(\bar{q}_1^t)$, so that this strategy is well-defined in the first position.)*

The condition (1) is not ad-hoc—it arises as the local bidding condition in round $t+1$ of our model when all advertisers play according to an envy-free symmetric strictly monotone equilibrium strategy.[9] Indeed, if advertiser $\pi$ expects other advertisers to repeat their bids from round $t$, then she must bid as in (1) if she is to be indifferent between receiving position $s_\pi^* - 1$ at price $b_\pi^*$ and receiving position $s_\pi^*$ at price $p_{s_\pi^*}^t$.[10]

[7] So that advertisers' strategies are well-defined in the first round, we must assume a random assignment of advertisers to positions at the beginning of the game (round 0), with all initial bids set to 0.

[8] Instead, we could require that each advertiser $\pi$ updates her bidding with attention to long-run equilibrium click-through rates, choosing $b_\pi^*$ to satisfy

$$(1 - q_\pi)G(\bar{q}_{s_\pi^*})(q_\pi - p_{s_\pi^*}^t) = G(\bar{q}_{s_\pi^*-1})(q_\pi - b_\pi^*).$$

With this bidding assumption, the proof of Theorem 1 goes through directly. In this case, the assumption that consumers always search in a top-down manner is unnecessary, as this search behavior arises endogenously.

[9] *Locally envy-free* equilibria, introduced independently by Varian [5] and Edelman *et al.* [4], are those in which an advertiser cannot improve her payoff by exchanging bids with the advertiser assigned the position directly above her. The focus on these particular equilibria and the associated *envy-free bidding* strategies is common within the sponsored search literature.

[10] Recall that $p_{s_\pi^*}^t$ is the price of position $s_\pi^*$ in round $t+1$ if all advertisers other than $\pi$ repeat their bids from round $t$.

The requirement that advertisers target only positions with no higher expected click-through rate than their current positions is less natural. This is a technical condition which is necessary in order to obtain convergence in the synchronous bidding model. As we discuss in Section 2.4, our convergence results for synchronous bidding under RBB imply that convergence obtains in an asynchronous bidding model even when this technical condition is lifted. Therefore, this condition does not appear overly confining.

Our Definition 1 is analogous to that of the restricted balanced bidding strategy which Cary *et al.* [2] specify for the Edelman *et al.* [4] auction mechanism. The key differences between this strategy and that of Cary *et al.* [2] appear in equation (1): the click-through rate factors in equation (1) are round-dependent, and an additional factor of $1 - q_\pi$ arises from the consumer search process.[11]

### 2.3 Convergence of the Auction

At the unique fixed point of the RBB strategy, the advertisers choose their bids according to the recursive strategy presented in Proposition 5 of Athey and Ellison [1].[12] That is, the bids at the fixed point are given by

$$b_{\pi_j} = \begin{cases} q_{\pi_j} - \frac{G(\bar{q}_j)}{G(\bar{q}_{j-1})}(1 - q_{\pi_j})(q_{\pi_j} - b_{\pi_{j+1}}) & 1 < j \leq M, \\ q_{\pi_j} & M < j \leq N, \end{cases} \tag{2}$$

where the $j$-th highest bid $b_{\pi_j}$ is submitted by advertiser $\pi_j$. Our main result is the convergence of the dynamic position auction to this fixed point.

**Theorem 1.** *In a synchronous model of dynamic bidding in which each advertiser bids every round, the RBB strategy always converges to its fixed point within finitely many rounds.*

This result is analogous to the convergence result of Cary *et al.* [2] for the auction mechanism of Edelman *et al.* [4]. Our proof follows the approach of Cary *et al.* [2]. However, the addition of consumer search greatly increases the technical difficulty of the argument. Consequently, we deliver slightly weaker bounds on convergence time than do Cary *et al.* [2].

*Proof.* We denote $\gamma_j^t(q) := (1 - q)\frac{G(\bar{q}_j^t)}{G(\bar{q}_{j-1}^t)}$ and let

$$\gamma^*(q) := (1 - q) \max_{\mathcal{P}} \left[ \max_{j>0} \left( \frac{G(\bar{q}_j^{\mathcal{P}})}{G(\bar{q}_{j-1}^{\mathcal{P}})} \right) \right], \quad \gamma^{**} := \max_{1 \leq \pi \leq N} \gamma^*(q_\pi).$$

[11] Since consumers search in a top-down manner, advertiser $\pi$ expects to lose a fraction of clicks equal to $1 - q_\pi$ when she switches from position $s_\pi^* - 1$ to position $s_\pi^*$, as $1 - q_\pi$ is the expected fraction of consumers meeting their needs at position $s_\pi^* - 1$.

Additionally, once consumer search is included, the computation of the position $s_\pi^*$ is different from that of Cary *et al.* [2]. However, this fact does not materially affect our arguments.

[12] This is a direct consequence of the proof of Proposition 5 of Athey and Ellison [1], which shows that these bids satisfy the condition (1).

We observe that, by construction,

$$\gamma_j^t(q_\pi) \le \gamma^*(q_\pi) \le \gamma^{**} < 1$$

for any $t > 0$, $1 \le j \le M$, and $1 \le \pi \le N$.[13]

We begin with two simple lemmata which are respectively analogous (and proven similarly) to Lemma 1 and Lemma 2 of Cary *et al.* [2].

**Lemma 1.** *Advertiser $\pi$ prefers to target position $j$ over position $j-1$ in round $t+1$ if and only if $(1-\gamma_j^t(q_\pi))q_\pi + \gamma_j^t(q_\pi)p_j^t < p_{j-1}^t$.*

*Proof.* This follows from the fact that advertiser $\pi$ prefers to target position $j$ over position $j-1$ if and only if

$$(1-q_\pi)G(\bar{q}_j^t)(q_\pi - p_j^t) > G(\bar{q}_{j-1}^t)(q_\pi - p_{j-1}^t),$$

upon algebraic manipulation.

**Lemma 2.** *At every round $t$ such that $t > t_1 := 2 + \log_{\gamma^{**}}((1-\gamma^{**})(q_M - q_{M+1})/q_{M+1})$, we have*

$$\begin{cases} b_\pi > q_{M+1} & \pi < M+1, \\ b_\pi = q_\pi & \pi \ge M+1, \end{cases}$$

*where $b_\pi$ is the bid of advertiser $1 \le \pi \le N$.*

*Proof.* If $b$ is the $(M+1)$-st highest bid, then $b \le q_{M+1}$. If $b < q_{M+1}$ in some round, then in the next round any advertiser $\pi \in \{1, 2, \ldots, M+1\}$ will either bid $b'_\pi = q_\pi$ or target some position $j \in \{1, \ldots, M\}$ with bid

$$\begin{aligned} b'_\pi &:= (1-\gamma_j^t(q_\pi))q_\pi + \gamma_j^t(q_\pi)p_j \\ &\ge (1-\gamma_j^t(q_\pi))q_{M+1} + \gamma_j^t(q_\pi)b \\ &= b + (1-\gamma_j^t(q_\pi))(q_{M+1} - b) \\ &\ge b + (1-\gamma^{**})(q_{M+1} - b). \end{aligned}$$

In both of these cases, it is clear that $q_{M+1} - b'_\pi \le \gamma^{**}(q_{M+1} - b)$.

It follows that we will have

$$q_{M+1} - b < (1-\gamma^{**})(q_M - q_{M+1})$$

within at most $r \le \log_{\gamma^{**}}((1-\gamma^{**})(q_M - q_{M+1})/q_{M+1})$ rounds. Then, the bidders $\pi \in \{1, \ldots, M\}$ will bid at least

$$\begin{aligned} (1-\gamma_j^t(q_\pi))q_\pi + \gamma_j^t(q_\pi)p_j &\ge (1-\gamma_j^t(q_\pi))q_\pi + \gamma_j^t(q_\pi)b \\ &\ge b + (1-\gamma_j^t(q_\pi))(q_\pi - b) \\ &> b + (1-\gamma^{**})(q_M - q_{M+1}) > q_{M+1} \end{aligned}$$

in round $r+1$. In round $r+2$, advertiser $M+1$ will bid $q_{M+1}$ while advertisers $\pi \in \{1, \ldots, M\}$ bid above $q_{M+1}$.

[13] The last of these inequalities follows from the fact that $\bar{q}_j^{\mathcal{P}} > \bar{q}_{j+1}^{\mathcal{P}}$ for any $\mathcal{P}$ and $1 \le j \le M$, since then $\frac{G(\bar{q}_j^{\mathcal{P}})}{G(\bar{q}_{j-1}^{\mathcal{P}})} < 1$.

Lemma 2 proves that, within finitely many rounds, the set of advertisers competing for the $M$ positions will stabilize and that this set will be the collection of advertisers of maximal quality, $\{1, \ldots, M\}$. Furthermore, at this time, the $N - M$ advertisers $\{M + 1, \ldots, N\}$ will bid their values in every round. Thus, we may assume that these rounds have already elapsed; all that remains is to show that the bids for the $M$ actual positions eventually converge to the desired fixed point. Since the fixed point is unique, it suffices to prove convergence.

For any $j \in [0, M]$, we say that the advertisers assigned positions $[j + 1, M]$ are *stable* if their allocation is in order of decreasing quality and their prices satisfy equation (1).[14] If all $M$ positions are stable, then it is clear that we have reached the fixed point of the RBB strategy.

We suppose that, at some round $t > t_1$, the set $S = [s + 1, M]$ of stable positions is not the full set $[1, M]$. We let $P$ denote the set of advertisers in positions $[1, s]$ and denote the minimum bid of these advertisers by $b$. We define a partial order $\sqsupset$ on stable sets: $S' \sqsupset S$ if either $S \subsetneq S'$ or if the advertiser of minimum quality in $(S \cup S') \setminus (S' \cap S)$ belongs to $S'$.

In round $t + 1$, all advertisers in $S$ repeat their bids. We let the new lowest bid of advertisers in $P$ be $b'_\pi$, bid by advertiser $\pi$. We must consider three cases:

*Case 1:* $b'_\pi < p^t_s$. We let $j$ be the position targeted by $\pi$. By Lemma 1 and the definition of RBB, we have $p^t_j < (1 - \gamma^t_j(q_\pi))q_\pi + \gamma^t_j(q_\pi)p^t_j = b'_\pi < p^t_{j-1}$.

We denote by $\pi_j \in S$ the advertiser who assigned position $j$ in round $t$. By the stability of $S$, we have $p^t_{j-1} = (1 - \gamma^t_j(q_{\pi_j}))q_{\pi_j} + \gamma^t_j(q_{\pi_j})p^t_j$. Then, we have

$$p^t_{j-1} = (1 - \gamma^t_j(q_{\pi_j}))q_{\pi_j} + \gamma^t_j(q_{\pi_j})p^t_j > (1 - \gamma^t_j(q_\pi))q_\pi + \gamma^t_j(q_\pi)p^t_j,$$

from which it follows that

$$(q_{\pi_j} - q_\pi)\left(1 + \frac{G(\bar{q}^t_j)}{G(\bar{q}^t_{j-1})}\left((q_{\pi_j} - p^t_j) + q_\pi - 1\right)\right) > 0. \tag{3}$$

Since advertiser $\pi_j$ is assigned position $j$ in round $t$, we know that $q_{\pi_j} \geq p^t_j$. Furthermore, $0 < \frac{G(\bar{q}^t_j)}{G(\bar{q}^t_{j-1})} \leq 1$, so

$$\frac{G_j}{G_{j-1}}\left((q_{\pi_j} - p^t_j) + q_\pi - 1\right) > -1.$$

It follows that (3) holds if and only if $q_{\pi_j} > q_\pi$. Likewise, we find that $q_{\pi_{j-1}} < q_\pi$. Thus, $S' := \{\pi' \in S : q_{\pi'} < q_\pi\} \cup \{\pi\}$ is stable and $S' \sqsupset S$.

*Case 2:* $\pi$ *targets position* $s$. Then $\pi$ is allocated position $s$ and $S \cup \{\pi\} \sqsupset S$ is stable.

[14] This is analogous to the definition of *stability* given by Cary *et al.* [2].

*Case 3: $\pi$ targets some position $j \leq s-1$.* Then, $S$ remains stable and the minimum bid of advertisers in $P$ has increased. We will show that this case may occur only finitely many times between occurrences of Cases 1 and 2.

As in Section 2.1, we respectively denote the qualities of the advertisers in positions $1, \ldots, M$ by $q_{\pi_1}, \ldots, q_{\pi_M}$. We then let

$$\epsilon := \frac{G(\bar{q}_M)}{2G(\bar{q}_1)}(1-\gamma^{**}) \min_{\pi \neq \pi'} |q_\pi - q_{\pi'}| \left( \prod_{j=1}^{M} (1-q_j) \right)$$

and let $x := \log_{1/\gamma^{**}}((q_1 - q_{M+1})/\epsilon)$. We will see that at most $x$ instances of Case 3 may occur between instances of Cases 1 and 2.

**Lemma 3.** *If $p_{s-1} > q_\pi - \epsilon$ then advertiser $\pi$ prefers position $s$ to any position $j < s$.*

*Proof.* We have

$$\begin{aligned} q_\pi - p_s &= (1-\gamma^t_{s+1}(q_{\pi_{s+1}}))(q_\pi - q_{\pi_{s+1}}) + \gamma^t_{s+1}(q_{\pi_{s+1}})p_{s+1} \\ &\geq (1-\gamma^{**}) \min_{\pi \neq \pi'} |q_\pi - q_{\pi'}|. \end{aligned} \tag{4}$$

The ratio of the expected utility of position $k < s$ to that of position $s$ is less than

$$\begin{aligned} \frac{G(\bar{q}^t_k)(q_\pi - p_{s-1})}{\left(\prod_{j=k}^{s}(1-q_{\pi_j})\right) G(\bar{q}^t_s)(q_\pi - p_s)} &\leq \epsilon \frac{G(\bar{q}^t_k)}{\left(\prod_{j=k}^{s}(1-q_{\pi_j})\right) G(\bar{q}^t_s)(q_\pi - p_s)} \\ &\leq \epsilon \frac{G(\bar{q}_1)}{\left(\prod_{j=k}^{s}(1-q_{\pi_j})\right) G(\bar{q}^t_s)(q_\pi - p_s)} \leq 1, \end{aligned}$$

where the last inequality follows from (4), the fact that $G(\bar{q}^t_s) \geq G(\bar{q}_M)$ (since $t > t_1$), and the definition of $\epsilon$.

Now, we suppose that Case 3 occurs for $x$ consecutive rounds. We let $\pi$ be the advertiser in $P$ of minimal quality $q_\pi$ and denote by $b^{(t')}$ the minimal bid of advertisers in $P$ after $t'$ consecutive rounds of Case 3. If $\pi' \in P$ submits the minimal bid $b^{(t'+1)}$ in the next round, then

$$\begin{aligned} b^{(t'+1)} &\geq (1-\gamma^*(q_{\pi'}))q_{\pi'} + \gamma^*(q_{\pi'})b^{(t')} \\ &\geq (1-\gamma^*(q_{\pi'}))q_\pi + \gamma^*(q_{\pi'})b^{(t')} \\ &= q_\pi - \gamma^*(q_{\pi'})(q_\pi - b^{(t')}) \\ &\geq q_\pi - \gamma^{**}(q_\pi - b^{(t')}). \end{aligned}$$

After $x$ consecutive rounds of Case 3, then, we have

$$b^{(x)} \geq q_\pi - (\gamma^{**})^x(q_\pi - b^{(0)}).$$

Hence, we have that $b^{(x)} \geq q_\pi - \epsilon$. It follows from Lemma 3 that $\pi$ will target position $s$ in the next round, so the next round is an instance of Case 2. Thus, we have shown that Case 3 may occur only finitely many times between instances of Cases 1 and 2.

### 2.4 Remarks

In Theorem 1, the number of rounds until convergence is constant in $N$, holding $\max_{1\leq\pi\leq N} q_\pi$ fixed. Thus, the entry of low-quality advertisers will not slow the auction's convergence.

Following the analysis of Cary *et al.* [2], it is possible to extend this result further, proving an analogous convergence result for an asynchronous auction game in a less restricted strategy space.[15]

**Definition 2.** *The* balanced bidding *(BB) strategy is the strategy for advertiser $\pi$ which, given the bids of the other advertisers in round $t$,*

- *targets the position $s_\pi^*$ which maximizes the utility of advertiser $\pi$,*
- *chooses the bid $b_\pi^*$ for round $t+1$ so as to satisfy the equation*

$$(1-q_\pi)G(\bar{q}^t_{s_\pi^*})(q_\pi - p^t_{s_\pi^*}) = G(\bar{q}^t_{s_\pi^*-1})(q_\pi - b_\pi^*).^{16} \tag{5}$$

*(As in RBB, we define $G(\bar{q}_0^t) := 2G(\bar{q}_1^t)$, so that this strategy is well-defined in the first position.)*

The BB strategy is a natural envy-free bidding strategy. Unlike RBB, each advertiser $\pi$ playing BB may target any position. But like RBB, the bid condition (5) arises as the advertisers' envy-free condition in a symmetric strictly monotone equilibrium. As in RBB, the bid profile (2) is a unique fixed point of the BB strategy.[17]

Our next result, which follows from the proof of Theorem 1, shows that the dynamic position auction converges under BB in an asynchronous bidding model.[18] The bound on convergence time has the same form as that of Theorem 2 of Cary *et al.* [2].

**Theorem 2.** *In the asynchronous model in which advertisers bid in a (uniformly) random order and follow a balanced bidding strategy, the system converges to the bid profile* (2) *with probability* 1 *and expected convergence time*

$$O\left(t_1(N\log M) + N\log N + M^{2^M(1+x)}\right).$$

We omit the proof of this result, as it follows directly from the analysis used by Cary *et al.* [2] in the proof of their Theorem 2.

[15] The asynchronous bidding model seems more realistic than does synchronous bidding, since in reality advertisers may update their sponsored search bids at any time.

[16] As before, the result goes through if instead each advertiser $\pi$ updates her bidding with attention to long-run equilibrium click-through rates.

[17] This follows directly from Proposition 5 of Athey and Ellison [1].

[18] This model differs from that of Section 2 only in that advertisers update their bids asynchronously, bidding in a (uniformly) random order.

## 3 Discussion and Conclusion

We have shown the convergence of a dynamic position auction in the presence of consumer search. Our approach closely follows the Cary *et al.* [2] analysis of the Edelman *et al.* [4] position auction model. The apparent robustness of the Cary *et al.* [2] methods to multiple position auction models is surprising, especially since these models differ substantially in their treatments of consumers. We believe that this robustness is structural, arising not only from the model frameworks but also from facts about the behavior of position auctions.

To clarify our observations, we discuss the three key steps of the Cary *et al.* [2] method for proving position auction convergence:

1. restriction of the strategy space (as in Definition 1),
2. demonstration that the advertisers with the lowest valuations must eventually bid their values (as in Lemma 2),
3. analysis of the late-stage behavior of the advertisers who win positions in equilibrium (as in Cases 1–3).

Step 1 is required in order to ensure that the bidding equilibrium in each stage is uniquely selected; an assumption of this form is present in most game-theoretic models of position auctions. Additional restriction may be required in the dynamic setting, in order to render the analysis of Step 3 tractable.[19]

Since it is always weakly dominant for advertisers to bid their values, the advertisers who actually receive positions in any position auction should be those with the highest valuations among bidders. We therefore expect Step 2 to be possible in most position auction settings, irrespective of the specific model framework. Notably, an analogue of Lemma 2 should hold even if the equilibrium bidding strategy is not strictly monotone in advertisers' valuations.[20]

The most complex analysis arises in Step 3, which proves the convergence of the auction amongst the advertisers with the largest valuations. The crux of this argument is the demonstration that the advertisers who win positions in equilibrium do not bid over their equilibrium bid "too often."[21] Although this cannot always be ensured,[22] it seems likely to occur—for a sufficiently restricted strategy space—whenever the equilibrium strategy is monotone.

[19] Edelman *et al.* [4] and Athey and Ellison [1] directly study the "balanced bidding" strategies appropriate to their models. However, both Cary *et al.* [2] and we have found it necessary to require additional restrictions in the synchronous bidding model, as convergence may not arise when balanced bidding is used in this model. As the name suggests, restricted balanced bidding is a substrategy of the balanced bidding strategy. Since balanced bidding leads to a unique equilibrium, these two strategies are equivalent in equilibrium.

[20] Of course, the difficulty of actually proving such a result will depend upon modeling decisions.

[21] This is encapsulated in two parts of our proof: Case 1 and Lemma 3.

[22] For example, Cary *et al.* [2] and Bu *et al.* [7] have demonstrated how "bid cycling," in which advertisers bid a sequence of values repeatedly, may arise in position auctions.

The conditions we have suggested for Steps 2 and 3 hold in many position auction settings.[23] Consequently, convergence should be demonstrable in dynamic position auctions with sufficiently well-behaved static equilibrium strategies.

## References

1. Athey, S., Ellison, G.: Position auctions with consumer search. Working paper (2008)
2. Cary, M., Das, A., Edelman, B., Giotis, I., Heimerl, K., Karlin, A.R., Mathieu, C., Schwarz, M.: On best-response bidding in GSP auctions. NBER Working Paper W13788 (2008)
3. Aggarwal, G., Goel, A., Motwani, R.: Truthful auctions for pricing search keywords. In: Proceedings of the 7th ACM conference on Electronic Commerce, pp. 1–7 (2006)
4. Edelman, B., Ostrovsky, M., Schwarz, M.: Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. American Economic Review 97, 242–259 (2007)
5. Varian, H.R.: Position auctions. International Journal of Industrial Organization 25, 1163–1178 (2006)
6. Chen, Y., He, C.: Paid placement: Advertising and search on the internet. NET Institute Working Paper 06-02 (2006)
7. Bu, T.M., Deng, X., Qi, Q.: Dynamics of strategic manipulation in ad-words auction. In: 3rd Workshop on Sponsored Search Auctions (SSA2007), in conjunction with WWW 2007 (2007)

[23] The auction models of Edelman *et al.* [4], Athey and Ellison [1], and Bu *et al.* [7] are all examples.

# Nonlinear Optimization over a Weighted Independence System*

Jon Lee[1], Shmuel Onn[2,**], and Robert Weismantel[3]

[1] IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
[2] Technion - Israel Institute of Technology, 32000 Haifa, Israel
[3] Otto-von-Guericke Universität Magdeburg, D-39106 Magdeburg, Germany

**Abstract.** We consider the problem of optimizing a nonlinear objective function over a weighted independence system presented by a linear optimization oracle. We provide a polynomial-time algorithm that determines an $r$-best solution for nonlinear functions of the total weight of an independent set, where $r$ is a constant depending on certain Frobenius numbers of the weights and is independent of the size of the ground set. In contrast, we show that finding an optimal (0-best) solution requires exponential time even in a very special case of the problem.

**Keywords:** independence system, matroid, oracle computation, exponential running time, combinatorial optimization, integer programming, nonlinear optimization, approximation algorithm, Frobenius number.

## 1 Introduction

An *independence system* is a nonempty set of vectors $S \subseteq \{0,1\}^n$ with the property that $x \in \{0,1\}^n$ , $x \leq y \in S$ implies $x \in S$ . The general nonlinear optimization problem over a multiply-weighted independence system is as follows.

**Nonlinear optimization over a multiply-weighted independence system.** Given independence system $S \subseteq \{0,1\}^n$ , weight vectors $w^1, \ldots, w^d \in \mathbb{Z}^n$ , and function $f : \mathbb{Z}^d \to \mathbb{R}$ , find $x \in S$ minimizing the objective

$$f(w^1 x, \ldots, w^d x) \;=\; f\left(\sum_{j=1}^{n} w_j^1 x_j, \ldots, \sum_{j=1}^{n} w_j^d x_j\right) \;.$$

The representation of the objective in the above composite form has several advantages. First, for $d > 1$ , it can naturally be interpreted as *multi-criteria optimization*: the $d$ given weight vectors $w^1, \ldots, w^d$ represent $d$ different criteria, where the value of $x \in S$ under criterion $i$ is its $i$-th total weight

* This research was supported in part by the Mathematisches Forschungsinstitut Oberwolfach through the Research in Pairs Programme.
** The research of this author was also supported in part by a grant from the Israel Science Foundation and by the E. and J. Bishop Fund.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 251–264, 2009.

$w^i x = \sum_{j=1}^n w_j^i x_j$ ; and the objective is to minimize the "balancing" $f(w^1 x, \ldots, w^d x)$ of the $d$ given criteria by the given function $f$ . Second, it allows us to classify nonlinear optimization problems into a hierarchy of increasing generality and complexity: at the bottom lies standard linear optimization, recovered with $d = 1$ and $f$ the identity on $\mathbb{Z}$ ; and at the top lies the problem of minimizing an arbitrary function, which is typically intractable, arising with $d = n$ and $w_i = \mathbf{1}_i$ the $i$-th standard unit vector in $\mathbb{Z}^n$ for all $i$ .

The computational complexity of the problem depends on the number $d$ of weight vectors, on the weights $w_j^i$ , on the type of function $f$ and its presentation, and on the type of independence system $S$ and its presentation. For example, when $S$ is a *matroid*, the problem can be solved in polynomial time for any fixed $d$ , any $\{0, 1, \ldots, p\}$-valued weights $w_j^i$ with $p$ fixed, and any function $f$ presented by a *comparison oracle*, even when $S$ is presented by a mere *membership oracle*, see [1]. Also, when $S$ consists of the *matchings* in a given bipartite graph $G$ , the problem can be solved in polynomial time for any fixed $d$ , any weights $w_j^i$ presented in unary, and any *concave* function $f$ , see [2]; but on the other hand, for *convex* $f$ , already with fixed $d = 2$ and $\{0, 1\}$-valued weights $w_j^i$ , it includes as a special case the notorious *exact matching problem*, the complexity of which is long open [5].

In view of the difficulty of the problem already for $d = 2$ , in this article we take a first step and concentrate on *nonlinear optimization over a (singly) weighted independence system*, that is, with $d = 1$ , single weight vector $w = (w_1, \ldots, w_n) \in \mathbb{Z}^n$ , and univariate function $f : \mathbb{Z} \to \mathbb{R}$ . The function $f$ can be arbitrary and is presented by a *comparison oracle* that, queried on $x, y \in \mathbb{Z}$ , asserts whether or not $f(x) \leq f(y)$ . The weights $w_j$ take on values in a $p$-tuple $a = (a_1, \ldots, a_p)$ of positive integers. Without loss of generality we assume that $a = (a_1, \ldots, a_p)$ is *primitive*, by which we mean that the $a_i$ are distinct positive integers having greatest common divisor $\gcd(a) := \gcd(a_1, \ldots, a_p)$ that is equal to 1 . The independence system $S$ is presented by a *linear-optimization oracle* that, queried on vector $v \in \mathbb{Z}^n$ , returns an element $x \in S$ that maximizes the linear function $vx = \sum_{j=1}^n v_j x_j$ . It turns out that solving this problem to optimality may require exponential time (see Theorem 5), and so we settle for an approximate solution in the following sense, that is interesting in its own right. For a nonnegative integer $r$ , we say that $x^* \in S$ is an *$r$-best solution* to the optimization problem over $S$ if there are at most $r$ better objective values attained by feasible solutions. In particular, a 0-best solution is optimal. Recall that the *Frobenius number* of a primitive $a$ is the largest integer $\mathrm{F}(a)$ that is not expressible as a nonnegative integer combination of the $a_i$ . We prove the following theorem.

**Theorem 1.** *For every primitive $p$-tuple $a = (a_1, \ldots, a_p)$ , there is a constant $r(a)$ and an algorithm that, given any independence system $S \subseteq \{0, 1\}^n$ presented by a linear-optimization oracle, weight vector $w \in \{a_1, \ldots, a_p\}^n$ , and function $f : \mathbb{Z} \to \mathbb{R}$ presented by a comparison oracle, provides an $r(a)$-best solution to the nonlinear problem* $\min\{f(wx) \ : \ x \in S\}$ *, in time polynomial in $n$ . Moreover:*

1. *If $a_i$ divides $a_{i+1}$ for $i = 1, \ldots, p-1$ , then the algorithm provides an optimal solution.*
2. *For $p = 2$ , that is, for $a = (a_1, a_2)$ , the algorithm provide an $\mathrm{F}(a)$-best solution.*

In fact, we give an explicit upper bound on $r(a)$ in terms of the Frobenius numbers of certain subtuples derived from $a$ .

Because $F(2,3) = 1$ , Theorem 1 (Part 2) assures us that we can efficiently compute a 1-best solution in that case. It is natural to wonder then whether, in this case, an optimal (i.e., 0-best) solution can be calculated in polynomial time. The next result indicates that this cannot be done.

**Theorem 2.** *There is no polynomial time algorithm for computing an optimal (i.e., 0-best) solution of the nonlinear optimization problem* $\min\{f(wx) : x \in S\}$ *over an independence system presented by a linear optimization oracle with $f$ presented by a comparison oracle and weight vector $w \in \{2,3\}^n$.*

The next sections gradually develop the various necessary ingredients used to establish our main results. §2 sets some notation. §3 discusses a naïve solution strategy that does not directly lead to a good approximation, but is a basic building block that is refined and repeatedly used later on. §4 describes a way of partitioning an independence system into suitable pieces, on each of which a suitable refinement of the naïve strategy will be applied separately. §5 provides some properties of monoids and Frobenius numbers that will allows us to show that the refined naïve strategy applied to each piece gives a good approximation within that piece. §6 combines all ingredients developed in §3–5, provides a bound on the approximation quality $r(a)$ , and provides the algorithm establishing Theorem 1. §7 demonstrates that finding an optimal solution is provably intractable, proving a refined version of Theorem 2. §8 concludes with some final remarks and questions. Due to lack of space, some of the proofs are omitted here. More details can be found in our Oberwolfach preprint [4].

## 2 Some Notation

In this section we provide some notation that will be used throughout the article. Some more specific notation will be introduced in later sections. We denote by $\mathbb{R}$ , $\mathbb{R}_+$ , $\mathbb{Z}$ and $\mathbb{Z}_+$ , the reals, nonnegative reals, integers and nonnegative integers, respectively. For a positive integer $n$, we let $N := \{1, \ldots, n\}$ . The $j$-th standard unit vector in $\mathbb{R}^n$ is denoted by $\mathbf{1}_j$ . The *support* of $x \in \mathbb{R}^n$ is the index set $\mathrm{supp}(x) := \{j : x_j \neq 0\} \subseteq N$ of nonzero entries of $x$ . The *indicator* of a subset $J \subseteq N$ is the vector $\mathbf{1}_J := \sum_{j \in J} \mathbf{1}_j \in \{0,1\}^n$ , so that $\mathrm{supp}(\mathbf{1}_J) = J$ . The *positive* and *negative* parts of a vector $x \in \mathbb{R}^n$ are denoted, respectively, by $x^+, x^- \in \mathbb{R}^n_+$ , and defined by $x_i^+ := \max\{x_i, 0\}$ and $x_i^- := -\min\{x_i, 0\}$ for $i = 1, \ldots, n$ . So, $x = x^+ - x^-$ , and $x_i^+ x_i^- = 0$ for $i = 1, \ldots, n$ .

Unless otherwise specified, $x$ denotes an element of $\{0,1\}^n$ and $\lambda, \mu, \tau, \nu$ denote elements of $\mathbb{Z}_+^p$ . Throughout, $a = (a_1, \ldots, a_p)$ is a *primitive* $p$-tuple, by which we

mean that the $a_i$ are distinct positive integers having greatest common divisor $\gcd(a) := \gcd(a_1, \ldots, a_p)$ equal to 1 . We will be working with weights taking values in $a$ , that is, vectors $w \in \{a_1, \ldots, a_p\}^n$ . With such a weight vector $w$ being clear from the context, we let $N_i := \{j \in N \ : \ w_j = a_i\}$ for $i = 1, \ldots, p$ , so that $N = \biguplus_{i=1}^p N_i$ . For $x \in \{0,1\}^n$ we let $\lambda_i(x) := |\mathrm{supp}(x) \cap N_i|$ for $i = 1, \ldots, p$ , and $\lambda(x) := (\lambda_1(x), \ldots, \lambda_p(x))$ , so that $wx = \lambda(x)a$ . For integers $z, s \in \mathbb{Z}$ and a set of integers $Z \subseteq \mathbb{Z}$ , we define $z + sZ := \{z + sx \ : \ x \in Z\}$ .

## 3 A Naïve Strategy

Consider a set $S \subseteq \{0,1\}^n$ , weight vector $w \in \{a_1, \ldots, a_p\}^n$ , and function $f : \mathbb{Z} \to \mathbb{R}$ presented by a comparison oracle. Define the *image* of $S$ under $w$ to be the set of values $wx$ taken by elements of $S$ ,

$$w \cdot S \quad := \quad \left\{ wx = \textstyle\sum_{j=1}^n w_j x_j \ : \ x \in S \right\} \quad \subseteq \quad \mathbb{Z}_+ \ .$$

As explained in the introduction, for a nonnegative integer $r$ , we say that $x^* \in S$ is an $r$-best solution if there are at most $r$ better objective values attained by feasible solutions. Formally, $x^* \in S$ is an *$r$-best solution* if

$$|\{f(wx) \ : \ f(wx) < f(wx^*) \ , \ x \in S\}| \ \leq \ r \ .$$

We point out the following simple observation.

**Proposition 1.** *If $f$ is given by a comparison oracle, then a necessary condition for any algorithm to find an $r$-best solution to the problem $\min\{f(wx) \ : \ x \in S\}$ is that it computes all but at most $r$ values of the image $w \cdot S$ of $S$ under $w$ .*

Note that this necessary condition is also sufficient for computing the weight $wx^*$ of an $r$-best solution, but not for computing an actual $r$-best solution $x^* \in S$ , which may be harder.

Any point $\bar{x}$ attaining $\max\{wx \ : \ x \in S\}$ provides an approximation of the image given by

$$\{wx \ : \ x \leq \bar{x}\} \quad \subseteq \quad w \cdot S \quad \subseteq \quad \{0, 1, \ldots, w\bar{x}\} \ . \tag{1}$$

This suggests the following natural naïve strategy for finding an approximate solution to the optimization problem over an independence system $S$ that is presented by a linear-optimization oracle.

**Naïve Strategy**

**input** independence system $S \subseteq \{0,1\}^n$ presented by a linear-optimization oracle, $f : \mathbb{Z} \to \mathbb{R}$ presented by a comparison oracle, and $w \in \{a_1, \ldots, a_p\}^n$ ;
**obtain** $\bar{x}$ attaining $\max\{wx \ : \ x \in S\}$ using the linear-optimization oracle for $S$ ;
**output** $x^*$ as one attaining $\min\{f(wx) \ : \ x \leq \bar{x}\}$ using the algorithm of Lemma 1 below .

Unfortunately, as the next example shows, the number of values of the image that are missing from the approximating set on the left-hand side of equation (1) cannot generally be bounded by any constant. So by Proposition 1, this strategy cannot be used *as is* to obtain a provably good approximation.

*Example 1.* Let $a := (1, 2)$ , $n := 4m$ , $y := \sum_{i=1}^{2m} \mathbf{1}_i$ , $z := \sum_{i=2m+1}^{4m} \mathbf{1}_i$ , and $w := y + 2z$ , that is,

$$y \;=\; (1, \ldots, 1, 0, \ldots, 0)\,, \quad z \;=\; (0, \ldots, 0, 1, \ldots, 1)\,, \quad w \;=\; (1, \ldots, 1, 2, \ldots, 2)\,,$$

define $f$ on $\mathbb{Z}$ by

$$f(k) := \begin{cases} k\ , & k \text{ odd;} \\ 2m\ , & k \text{ even,} \end{cases}$$

and let $S$ be the independence system

$$S \;:=\; \{x \in \{0,1\}^n \;:\; x \le y\} \;\cup\; \{x \in \{0,1\}^n \;:\; x \le z\}\ .$$

Then the unique optimal solution of the linear-objective problem $\max\{wx \;:\; x \in S\}$ is $\bar{x} := z$ , with $w\bar{x} = 4m$ , and therefore

$$\begin{aligned} &\{wx \;:\; x \le \bar{x}\} \;=\; \{2i \;:\; i = 0, 1, \ldots, 2m\}\ , \text{ and} \\ &w \cdot S \;=\; \{i \;:\; i = 0, 1, \ldots, 2m\} \;\cup\; \{2i \;:\; i = 0, 1, \ldots, 2m\}\ . \end{aligned}$$

So all $m$ odd values (i.e., $1, 3, \ldots, 2m-1$) in the image $w \cdot S$ are missing from the approximating set $\{wx \;:\; x \le \bar{x}\}$ on the left-hand side of (1), and $x^*$ attaining $\min\{f(wx) \;:\; x \le \bar{x}\}$ output by the above strategy has objective value $f(wx^*) = 2m$ , while there are $m = \frac{n}{4}$ better objective values (i.e., $1, 3, \ldots, 2m-1$) attainable by feasible points (e.g., $\sum_{i=1}^{k} \mathbf{1}_i$ , for $k = 1, 3, \ldots, 2m-1$).

Nonetheless, a more sophisticated refinement of the naïve strategy, applied repeatedly to several suitably chosen subsets of $S$ rather than $S$ itself, will lead to a good approximation. In the next two sections, we develop the necessary ingredients that enable us to implement such a refinement of the naïve strategy and to prove a guarantee on the quality of the approximation it provides. Before proceeding to the next section, we note that the naïve strategy can be efficiently implemented as follows.

**Lemma 1.** *For every fixed $p$-tuple $a$ , there is a polynomial-time algorithm that, given univariate function $f : \mathbb{Z} \to \mathbb{R}$ presented by a comparison oracle, weight vector $w \in \{a_1, \ldots, a_p\}^n$ , and $\bar{x} \in \{0,1\}^n$ , solves*

$$\min\{f(wx) \;:\; x \le \bar{x}\}\ .$$

*Proof.* Consider the following algorithm:

**input** function $f : \mathbb{Z} \to \mathbb{R}$ presented by a comparison oracle,
$w \in \{a_1, \ldots, a_p\}^n$ and $\bar{x} \in \{0,1\}^n$ ;
**let** $N_i := \{j \ : \ w_j = a_i\}$ and $\tau_i := \lambda_i(\bar{x}) = |\mathrm{supp}(\bar{x}) \cap N_i|, \ i = 1, \ldots, p$ ;
**for** every choice of $\nu = (\nu_1, \ldots, \nu_p) \leq (\tau_1, \ldots, \tau_p) = \tau$ **do**
  **determine** some $x_\nu \leq \bar{x}$ with $\lambda_i(x_\nu) = |\mathrm{supp}(x_\nu) \cap N_i| = \nu_i$,
  $i = 1, \ldots, p$ ;
**end**
**output** $x^*$ as one minimizing $f(wx)$ among the $x_\nu$ by using the
comparison oracle of $f$ .

Since the value $wx$ depends only on the cardinalities $|\mathrm{supp}(x) \cap N_i|, \ i = 1, \ldots, p$ , it is clear that

$$\{wx \ : \ x \leq \bar{x}\} \quad = \quad \{wx_\nu \ : \ \nu \leq \tau\} \ .$$

Clearly, for each choice $\nu \leq \tau$ it is easy to determine some $x_\nu \leq \bar{x}$ by zeroing out suitable entries of $\bar{x}$ . The number of choices $\nu \leq \tau$ and hence of loop iterations and comparison-oracle queries of $f$ to determine $x^*$ is

$$\prod_{i=1}^{p} (\tau_i + 1) \ \leq \ (n+1)^p \ . \qquad \square$$

## 4 Partitions of Independence Systems

Define the *face of* $S \subseteq \{0,1\}^n$ *determined by two disjoint subsets* $L, U \subseteq N = \{1, \ldots, n\}$ to be

$$S_L^U \ := \ \{x \in S \ : \ x_j = 0 \text{ for } j \in L \, , \ x_j = 1 \text{ for } j \in U\} \ .$$

Our first simple lemma reduces linear optimization over faces of $S$ to linear optimization over $S$ . We omit the proof which can be found in our preprint [4].

**Lemma 2.** *Consider any nonempty set* $S \subseteq \{0,1\}^n$ *, weight vector* $w \in \mathbb{Z}^n$ *, and disjoint subsets* $L, U \subseteq N$ *. Let* $\alpha := 1 + 2n \max |w_j|$ *, let* $\mathbf{1}_L, \mathbf{1}_U \in \{0,1\}^n$ *be the indicators of* $L, U$ *respectively, and let*

$$\begin{aligned} v &:= \max \{(w + \alpha(\mathbf{1}_U - \mathbf{1}_L))x \ : \ x \in S\} \ - \ |U|\alpha \\ &= \max \left\{ wx - \alpha \left( \textstyle\sum_{j \in U} (1 - x_j) + \sum_{j \in L} x_j \right) \ : \ x \in S \right\} \ . \end{aligned} \tag{2}$$

*Then either* $v > -\frac{1}{2}\alpha$*, in which case* $\max\{wx \ : \ x \in S_L^U\} = v$ *and the set of maximizers of* $wx$ *over* $S_L^U$ *is equal to the set of maximizers of the program (2), or* $v < -\frac{1}{2}\alpha$ *, in which case* $S_L^U$ *is empty.*

Let $S \subseteq \{0,1\}^n$ and $w \in \{a_1, \ldots, a_p\}^n$ be arbitrary, and let $N_i := \{j \in N \ : \ w_j = a_i\}$ as usual. As usual, for $x \in S$ , let $\lambda_i(x) := |\mathrm{supp}(x) \cap N_i|$ for each $i$ . For $p$-tuples $\mu = (\mu_1, \ldots, \mu_p)$ and $\lambda = (\lambda_1, \ldots, \lambda_p)$ in $\mathbb{Z}_+^p$ with $\mu \leq \lambda$ , define

$$S_\mu^\lambda \;:=\; \left\{ x \in S \;:\; \begin{array}{l} \lambda_i(x) = \mu_i \text{ , if } \mu_i < \lambda_i \text{ ,} \\ \lambda_i(x) \geq \mu_i \text{ , if } \mu_i = \lambda_i \text{ .} \end{array} \right\} \;. \tag{3}$$

The proof of the following statement is easy and omitted.

**Proposition 2.** *Let $S \subseteq \{0,1\}^n$ be arbitrary. Then every $\lambda \in \mathbb{Z}_+^p$ induces a partition of $S$ given by*

$$S \;=\; \biguplus_{\mu \leq \lambda} S_\mu^\lambda \;.$$

**Lemma 3.** *For all fixed $p$-tuples $a$ and $\lambda \in \mathbb{Z}_+^p$ , there is a polynomial-time algorithm that, given any independence system $S$ presented by a linear-optimization oracle, $w \in \{a_1, \ldots, a_p\}^n$ , and $\mu \in \mathbb{Z}_+^p$ with $\mu \leq \lambda$ , solves*

$$\max \left\{ wx \;:\; x \in S_\mu^\lambda \right\} \;.$$

*Proof.* Consider the following algorithm:

**input** independence system $S \subseteq \{0,1\}^n$ presented by a linear-optimization oracle , $w \in \{a_1, \ldots, a_p\}^n$ , and $\mu \leq \lambda$ ;
**let** $I := \{i \;:\; \mu_i < \lambda_i\}$ and $N_i := \{j \in N \;:\; w_j = a_i\},\ i = 1, \ldots, p$ ;
**for** every $S_i \subseteq N_i$ with $|S_i| = \mu_i,\ i = 1, \ldots, p,$ if any, **do**
  **let** $L := \bigcup_{i \in I} (N_i \setminus S_i)$ and $U := \bigcup_{i=1}^p S_i$ ;
  **find** by the algorithm of Lemma 2 an $x(S_1, \ldots, S_p)$ attaining $\max\{wx \;:\; x \in S_L^U\}$ if any;
**end**
**output** $x^*$ as one maximizing $wx$ among all of the $x(S_1, \ldots, S_p)$ (if any) found in the loop above .

It is clear that $S_\mu^\lambda$ is the union of the $S_L^U$ over all choices $S_1, \ldots, S_p$ as above, and therefore $x^*$ is indeed a maximizer of $wx$ over $S_\mu^\lambda$ . The number of such choices and hence of loop iterations is

$$\prod_{i=1}^p \binom{|N_i|}{\mu_i} \;\leq\; \prod_{i=1}^p n^{\mu_i} \;\leq\; \prod_{i=1}^p n^{\lambda_i} \;,$$

which is polynomial because $\lambda$ is fixed. In each iteration, we find $x(S_1, \ldots, S_p)$ maximizing $wx$ over $S_L^U$ or detect $S_L^U = \emptyset$ by applying the algorithm of Lemma 2 using a single query of the linear-optimization oracle for $S$ . □

We will later show that, for a suitable choice of $\lambda$ , we can guarantee that, for every block $S_\mu^\lambda$ of the partition of $S$ induced by $\lambda$ , the naïve strategy applied to $S_\mu^\lambda$ does give a good solution, with only a constant number of better objective values obtainable by solutions within $S_\mu^\lambda$ . For this, we proceed next to take a closer look at the monoid generated by a $p$-tuple $a$ and at suitable restrictions of this monoid.

## 5 Monoids and Frobenius Numbers

Recall that a $p$-tuple $a = (a_1, \ldots, a_p)$ is *primitive* if the $a_i$ are distinct positive integers having greatest common divisor $\gcd(a) = \gcd(a_1, \ldots, a_p)$ is 1 . For $p = 1$ , the only primitive $a = (a_1)$ is the one with $a_1 = 1$ . The *monoid* of $a = (a_1, \ldots, a_p)$ is the set of nonnegative integer combinations of its entries,

$$M(a) \;=\; \left\{\mu a = \textstyle\sum_{i=1}^{p} \mu_i a_i \;:\; \mu \in \mathbb{Z}_+^p\right\} \;.$$

The *gap set* of $a$ is the set $G(a) := \mathbb{Z}_+ \setminus M(a)$ and is well known to be finite [3]. If all $a_i \geq 2$ , then $G(a)$ is nonempty, and its maximum element is known as the *Frobenius number* of $a$ , and will be denoted by $\mathrm{F}(a) := \max G(a)$ . If some $a_i = 1$ , then $G(a) = \emptyset$ , in which case we define $\mathrm{F}(a) := 0$ by convention. Also, we let $\mathrm{F}(a) := 0$ by convention for the empty $p$-tuple $a = ()$ with $p = 0$ .

*Example 2.* If $a = (3, 5)$ then the gap set is $G(a) = \{1, 2, 4, 7\}$ , and the Frobenius number is $\mathrm{F}(a) = 7$ .

Classical results of Schur and Sylvester, respectively, assert that for all $p \geq 2$ and all $a = (a_1, \ldots, a_p)$ with each $a_i \geq 2$ , the Frobenius number obeys the upper bound

$$\mathrm{F}(a) \;+\; 1 \;\leq\; \min\left\{(a_i - 1)(a_j - 1) \;:\; 1 \leq i < j \leq p\right\} \;, \tag{4}$$

with equality $\mathrm{F}(a)+1 = (a_1-1)(a_2-1)$ holding for $p = 2$ . See [3] and references therein for proofs.

Define the *restriction* of $M(a)$ by $\lambda \in \mathbb{Z}_+^p$ to be the following subset of $M(a)$ :

$$M(a, \lambda) \;:=\; \left\{\mu a \;:\; \mu \in \mathbb{Z}_+^p \,,\; \mu \leq \lambda\right\} \;.$$

We start with two simple facts, the proofs of which are omitted.

**Proposition 3.** *For every $\lambda \in \mathbb{Z}_+^p$ , $M(a, \lambda)$ is symmetric on $\{0, 1, \ldots, \lambda a\}$ , that is, we have that $g \in M(a, \lambda)$ if and only if $\lambda a - g \in M(a, \lambda)$ .*

Recall that for $z, s \in \mathbb{Z}$ and $Z \subseteq \mathbb{Z}$ , we let $z + sZ := \{z + sx \;:\; x \in Z\}$ .

**Proposition 4.** *For every $\lambda \in \mathbb{Z}_+^p$ , we have*

$$M(a, \lambda) \;\subseteq\; \{0, 1, \ldots, \lambda a\} \setminus \left(\, G(a) \cup (\lambda a - G(a)) \,\right) \;. \tag{5}$$

Call $\lambda \in \mathbb{Z}_+^p$ *saturated for $a$* if (5) holds for $\lambda$ with equality. In particular, if some $a_i = 1$ , then $\lambda$ saturated for $a$ implies $M(a, \lambda) = \{0, 1, \ldots, \lambda a\}$ .
*Example 2, continued.* For $a = (3, 5)$ and say $\lambda = (3, 4)$ , we have $\lambda a = 29$, and it can be easily checked that there are two values, namely $12 = 4 \cdot 3 + 0 \cdot 5$ and $17 = 4 \cdot 3 + 1 \cdot 5$ , that are not in $M(a, \lambda)$ but are in $\{0, 1, \ldots, \lambda a\} \setminus \left(\, G(a) \cup (\lambda a - G(a)) \,\right)$ . Hence, in this case $\lambda$ is not saturated for $a$ .

Let $\max(a) := \max\{a_1, \ldots, a_p\}$ . Call $a = (a_1, \ldots, a_p)$ *divisible* if $a_i$ divides $a_{i+1}$ for $i = 1, \ldots p - 1$ . The following theorem asserts that, for any fixed primitive $a$ , every (component-wise) sufficiently large $p$-tuple $\lambda$ is saturated.

**Theorem 3.** *Let $a = (a_1, \ldots, a_p)$ be any primitive $p$-tuple. Then the following statements hold:*

1. *Every $\lambda = (\lambda_1, \ldots, \lambda_p)$ satisfying $\lambda_i \geq \max(a)$ for $i = 1, \ldots, p$ is saturated for $a$ .*
2. *For divisible $a$ , every $\lambda = (\lambda_1, \ldots, \lambda_p)$ satisfying $\lambda_i \geq \frac{a_{i+1}}{a_i} - 1$ for $i = 1, \ldots, p-1$ is saturated for $a$ .*

*Proof.* We provide only the proof of Part 2. The proof of Part 1 which is somewhat lengthy and tedious is omitted and can be found in our preprint [4].

To prove Part 2, we begin by using induction on $p$ . For $p = 1$ , we have $a_1 = 1$ , and every $\lambda = (\lambda_1)$ is saturated because every $0 \leq v \leq \lambda a = \lambda_1$ satisfies $v = \mu a = \mu_1$ for $\mu \leq \lambda$ given by $\mu = (\mu_1)$ with $\mu_1 = v$ .

Next consider $p > 1$ . We use induction on $\lambda_p$ . Suppose first that $\lambda_p = 0$ . Let $a' := (a_1, \ldots, a_{p-1})$ and $\lambda' := (\lambda_1, \ldots, \lambda_{p-1})$ . Consider any value $0 \leq v \leq \lambda a = \lambda' a'$ . Since $\lambda'$ is saturated by induction on $p$ , there exists $\mu' \leq \lambda'$ with $v = \mu' a'$ . Then, $\mu := (\mu', 0) \leq \lambda$ and $v = \mu a$ . So $\lambda$ is also saturated. Next, consider $\lambda_p > 0$ . Let $\tau := (\lambda_1, \ldots, \lambda_{p-1}, \lambda_p - 1)$ . Consider any value $0 \leq v \leq \tau a = \lambda a - a_p$ . Since $\tau$ is saturated by induction on $\lambda_p$ , there is a $\mu \leq \tau < \lambda$ with $v = \mu a$ , and so $v \in M(a, \tau) \subseteq M(a, \lambda)$ . Moreover, $v + a_p = \hat{\mu} a$ with $\hat{\mu} := (\mu_1, \ldots, \mu_{p-1}, \mu_p + 1) \leq \lambda$ , so $v + a_p \in M(a, \lambda)$ as well. Therefore

$$\{0, 1, \ldots, \tau a\} \ \cup \ \{a_p, a_p + 1, \ldots, \lambda a\} \ \subseteq \ M(a, \lambda) \ . \tag{6}$$

Now,

$$\tau a \ = \ \sum_{i=1}^{p} \tau_i a_i \ \geq \ \sum_{i=1}^{p-1} \lambda_i a_i \ \geq \ \sum_{i=1}^{p-1} \left( \frac{a_{i+1}}{a_i} - 1 \right) a_i \ = \ \sum_{i=1}^{p-1} (a_{i+1} - a_i) \ = \ a_p - 1 \ ,$$

implying the left-hand side of (6) is in fact equal to $\{0, 1, \ldots, \lambda a\}$ . So $\lambda$ is indeed saturated. This completes the double induction and the proof of Part 2. □

## 6 Obtaining an $r$-Best Solution

We can now combine all the ingredients developed in the previous sections and provide our algorithm. Let $a = (a_1, \ldots, a_p)$ be a fixed primitive $p$-tuple. Define $\lambda = (\lambda_1, \ldots, \lambda_p)$ by $\lambda_i := \max(a)$ for every $i$ . For $\mu \leq \lambda$ define

$$I_\mu^\lambda \ := \ \{i \ : \ \mu_i = \lambda_i\} \quad \text{and} \quad a_\mu^\lambda \ := \ \left( \frac{a_i}{\gcd(a_i \ : \ i \in I_\mu^\lambda)} \ : \ i \in I_\mu^\lambda \right) \ .$$

Finally, define

$$r(a) \quad := \quad \sum_{\mu \leq \lambda} \mathrm{F}(a_\mu^\lambda) \ . \tag{7}$$

The next corollary gives some estimates on $r(a)$ , including a general bound implied by Theorem 3.

**Corollary 1.** *Let $a = (a_1, \ldots, a_p)$ be any primitive $p$-tuple. Then the following hold:*

1. *An upper bound on $r(a)$ is given by $r(a) \leq (2\max(a))^p$ .*
2. *For divisible $a$ , we have $r(a) = 0$ .*
3. *For $p = 2$ , that is, for $a = (a_1, a_2)$ , we have $r(a) = \mathrm{F}(a)$ .*

*Proof.* Define $\lambda = (\lambda_1, \ldots, \lambda_p)$ by $\lambda_i := \max(a)$ for every $i$ . First note that if $I^\lambda_\mu$ is empty or a singleton then $a^\lambda_\mu$ is empty or $a^\lambda_\mu = 1$ , and hence $\mathrm{F}(a^\lambda_\mu) = 0$ .

Part 1: As noted, $\mathrm{F}(a^\lambda_\mu) = 0$ for each $\mu \leq \lambda$ with $|I^\lambda_\mu| \leq 1$ . There are at most $2^p(\max(a))^{p-2}$ $p$-tuples $\mu \leq \lambda$ with $|I^\lambda_\mu| \geq 2$ and for each, the bound of equation (4) implies $\mathrm{F}(a^\lambda_\mu) \leq (\max(a))^2$ . Hence

$$r(a) \;\leq\; 2^p(\max(a))^{p-2}(\max(a))^2 \;\leq\; (2\max(a))^p \;.$$

Part 2: If $a$ is divisible, then the least entry of every nonempty $a^\lambda_\mu$ is 1 , and hence $\mathrm{F}(a^\lambda_\mu) = 0$ for every $\mu \leq \lambda$ . Therefore $r(a) = 0$ .

Part 3: As noted, $\mathrm{F}(a^\lambda_\mu) = 0$ for each $\mu \leq \lambda$ with $|I^\lambda_\mu| \leq 1$ . For $p = 2$ , the only $\mu \leq \lambda$ with $|I^\lambda_\mu| = 2$ is $\mu = \lambda$ . Because $a^\lambda_\lambda = a$ , we obtain $r(a) = \mathrm{F}(a)$ . □

We are now in position to prove the following refined version of our main theorem (Theorem 1).

**Theorem 4.** *For every primitive $p$-tuple $a = (a_1, \ldots, a_p)$ , with $r(a)$ as in (7) above, there is an algorithm that, given any independence system $S \subseteq \{0,1\}^n$ presented by a linear-optimization oracle, weight vector $w \in \{a_1, \ldots, a_p\}^n$ , and function $f : \mathbb{Z} \to \mathbb{R}$ presented by a comparison oracle, provides an $r(a)$-best solution to the nonlinear problem $\min\{f(wx) \;:\; x \in S\}$ , in time polynomial in $n$ . Moreover:*

1. *If $a_i$ divides $a_{i+1}$ for $i = 1, \ldots, p-1$ , then the algorithm provides an optimal solution.*
2. *For $p = 2$ , that is, for $a = (a_1, a_2)$ , the algorithm provide an $\mathrm{F}(a)$-best solution.*

*Proof.* Consider the following algorithm:

**input** independence system $S \subseteq \{0,1\}^n$ presented by a linear-optimization oracle, $f : \mathbb{Z} \to \mathbb{R}$ presented by a comparison oracle, and $w \in \{a_1, \ldots, a_p\}^n$ ;
**define** $\lambda = (\lambda_1, \ldots, \lambda_p)$ by $\lambda_i := \max(a)$ for every $i$ ;
**for** every choice of $p$-tuple $\mu \in \mathbb{Z}^p_+$ , $\mu \leq \lambda$ **do**
  **find** by the algorithm of Lemma 3 an $x_\mu$ attaining $\max\{wx \;:\; x \in S^\lambda_\mu\}$ if any;
  **if** $S^\lambda_\mu \neq \emptyset$ **then** find by the algorithm of Lemma 1 an $x^*_\mu$ attaining $\min\{f(wx) \;:\; x \in \{0,1\}^n \,,\; x \leq x_\mu\}$ ;
**end**
**output** $x^*$ as one minimizing $f(wx)$ among the $x^*_\mu$ .

First note that the number of $p$-tuples $\mu \leq \lambda$ and hence of loop iterations and applications of the polynomial-time algorithms of Lemma 1 and Lemma 3 is $\prod_{i=1}^{p}(\lambda_i + 1) = (1 + \max(a))^p$ which is constant since $a$ is fixed. Therefore the entire running time of the algorithm is polynomial.

Consider any $p$-tuple $\mu \leq \lambda$ with $S_\mu^\lambda \neq \emptyset$ , and let $x_\mu$ be an optimal solution of $\max\{wx \ : \ x \in S_\mu^\lambda\}$ determined by the algorithm. Let $I := I_\mu^\lambda = \{i \ : \ \mu_i = \lambda_i\}$ , let $g := \gcd(a_i \ : \ i \in I)$ , let $\bar{a} := a_\mu^\lambda = \frac{1}{g}(a_i \ : \ i \in I)$ , and let $h := \sum\{\mu_i a_i \ : \ i \notin I\}$ . For each point $x \in \{0,1\}^n$ and for each $i = 1, \ldots, p$ , let as usual $\lambda_i(x) := |\mathrm{supp}(x) \cap N_i|$ , where $N_i = \{j \ : \ w_j = a_i\}$ , and let $\bar{\lambda}(x) := (\lambda_i(x) \ : \ i \in I)$ . By the definition of $S_\mu^\lambda$ in equation (3) and of $I$ above, for each $x \in S_\mu^\lambda$ we have

$$wx \ = \ \sum_{i \notin I} \lambda_i(x) a_i + \sum_{i \in I} \lambda_i(x) a_i \ = \ \sum_{i \notin I} \mu_i a_i + g \sum_{i \in I} \lambda_i(x) \frac{1}{g} a_i \ = \ h + g\bar{\lambda}(x)\bar{a} \ .$$

In particular, for every $x \in S_\mu^\lambda$ we have $wx \in h + gM(\bar{a})$ and $wx \leq wx_\mu = h + g\bar{\lambda}(x_\mu)\bar{a}$ , and therefore

$$w \cdot S_\mu^\lambda \ \subseteq \ h \ + \ g\left(M(\bar{a}) \cap \{0, 1 \ldots, \bar{\lambda}(x_\mu)\bar{a}\}\right) \ .$$

Let $T := \{x \ : \ x \leq x_\mu\}$ . Clearly, for any $\bar{\nu} \leq \bar{\lambda}(x_\mu)$ there is an $x \in T$ obtained by zeroing out suitable entries of $x_\mu$ such that $\bar{\lambda}(x) = \bar{\nu}$ and $\lambda_i(x) = \lambda_i(x_\mu) = \mu_i$ for $i \notin I$ , and hence $wx = h + g\bar{\nu}\bar{a}$ . Therefore

$$h \ + \ gM\left(\bar{a}, \bar{\lambda}(x_\mu)\right) \ \subseteq \ w \cdot T \ .$$

Since $x_\mu \in S_\mu^\lambda$ , by the definition of $S_\mu^\lambda$ and $I$ , for each $i \in I$ we have

$$\lambda_i(x_\mu) \ = \ |\mathrm{supp}(x) \cap N_i| \ \geq \ \mu_i \ = \ \lambda_i \ = \ \max(a) \ \geq \max(\bar{a}) \ .$$

Therefore, by Theorem 3, we conclude that $\bar{\lambda}(x_\mu) = (\lambda_i(x_\mu) \ : \ i \in I)$ is saturated for $\bar{a}$ and hence

$$M\left(\bar{a}, \bar{\lambda}(x_\mu)\right) \ = \ \left(M(\bar{a}) \cap \{0, 1 \ldots, \bar{\lambda}(x_\mu)\bar{a}\}\right) \ \setminus \ \left(\bar{\lambda}(x_\mu)\bar{a} - G(\bar{a})\right) \ .$$

This implies that

$$w \cdot S_\mu^\lambda \ \setminus \ w \cdot T \ \subseteq \ h \ + \ g\left(\bar{\lambda}(x_\mu)\bar{a} - G(\bar{a})\right) \ ,$$

and hence

$$|w \cdot S_\mu^\lambda \ \setminus \ w \cdot T| \ \leq \ |G(\bar{a})| \ = \ \mathrm{F}(\bar{a}) \ .$$

Therefore, as compared to the objective value of the optimal solution $x_\mu^*$ of

$$\min\{f(wx) \ : \ x \in T\} = \min\{f(wx) \ : \ x \leq x_\mu\}$$

determined by the algorithm, at most $\mathrm{F}(\bar{a})$ better objective values are attained by points in $S_\mu^\lambda$ .

Since $S = \biguplus_{\mu \leq \lambda} S_\mu^\lambda$ by Proposition 2, the independence system $S$ has altogether at most

$$\sum_{\mu \leq \lambda} \mathrm{F}(a_\mu^\lambda) \ = \ r(a)$$

better objective values $f(wx)$ attainable than that of the solution $x^*$ output by the algorithm. Therefore $x^*$ is indeed an $r(a)$-best solution to the nonlinear optimization problem over the (singly) weighted independence system. □

In fact, as the above proof of Theorem 4 shows, our algorithm provides a better, $g(a)$-best, solution, where $g(a)$ is defined as follows in terms of the cardinalities of the gap sets of the subtuples $a_\mu^\lambda$ with $\lambda$ defined again by $\lambda_i := 2\max(a)$ for all $i$ (in particular, $g(a) = |G(a)|$ for $p = 2$),

$$g(a) \quad := \quad \sum_{\mu \leq \lambda} |G(a_\mu^\lambda)| \; . \tag{8}$$

## 7 Finding an Optimal Solution Requires Exponential Time

We now demonstrate that our results are best possible in the following sense. Consider $a := (2, 3)$. Because $F(2, 3) = 1$, Theorem 1 (Part 2) assures that our algorithm produces a 1-best solution in polynomial time. We next establish a refined version of Theorem 2, showing that a 0-best (i.e., optimal) solution *cannot* be found in polynomial time.

**Theorem 5.** *There is no polynomial time algorithm for computing a* 0*-best (i.e., optimal) solution of the nonlinear optimization problem* $\min\{f(wx) : x \in S\}$ *over an independence system presented by a linear optimization oracle with* $f$ *presented by a comparison oracle and weight vector* $w \in \{2, 3\}^n$*. In fact, to solve the nonlinear optimization problem over every independence system* $S$ *with a ground set of* $n = 4m$ *elements with* $m \geq 2$*, at least* $\binom{2m}{m+1} \geq 2^m$ *queries of the oracle presenting* $S$ *are needed.*

*Proof.* Let $n := 4m$ with $m \geq 2$, $I := \{1, \ldots, 2m\}$, $J := \{2m + 1, \ldots, 4m\}$, and let $w := 2 \cdot \mathbf{1}_I + 3 \cdot \mathbf{1}_J$ . For $E \subseteq \{1, \ldots, n\}$ and any nonnegative integer $k$ , let $\binom{E}{k}$ be the set of all $k$-element subsets of $E$. For $i = 0, 1, 2$ , let

$$T_i \; := \; \left\{ x = \mathbf{1}_A + \mathbf{1}_B \; : \; A \in \binom{I}{m+i}, \; B \in \binom{J}{m-i} \right\} \; \subset \; \{0,1\}^n \; .$$

Let $S$ be the independence system generated by $T_0 \cup T_2$, that is,

$$S \; := \; \{z \in \{0,1\}^n \; : \; z \leq x \text{ , for some } x \in T_0 \cup T_2\} \; .$$

Note that the $w$-image of $S$ is

$$w \cdot S = \{0, \ldots, 5m\} \setminus \{1, 5m - 1\} \; .$$

For every $y \in T_1$ , let $S_y := S \cup \{y\}$ . Note that each $S_y$ is an independence system as well, but with $w$-image

$$w \cdot S_y = \{0, \ldots, 5m\} \setminus \{1\} \; ;$$

that is, the $w$-image of each $S_y$ is precisely the $w$-image of $S$ augmented by the value $5m - 1$ .

Finally, for each vector $c \in \mathbb{Z}^n$ , let

$$Y(c) \;:=\; \{y \in T_1 \,:\, cy > \max\{cx : x \in S\}\} \;.$$

*Claim:* $|Y(c)| \leq \binom{2m}{m-1}$ for every $c \in \mathbb{Z}^n$ .
*Proof of Claim:* Consider two elements (if any) $y, z \in Y(c)$ . Then $y = \mathbf{1}_A + \mathbf{1}_B$ and $z = \mathbf{1}_U + \mathbf{1}_V$ for some $A, U \in \binom{I}{m+1}$ and $B, V \in \binom{J}{m-1}$. Suppose, indirectly, that $A \neq U$ and $B \neq V$. Pick $a \in A \setminus U$ and $v \in V \setminus B$ . Consider the following vectors,

$$x^0 := y - \mathbf{1}_a + \mathbf{1}_v \;\in\; T_0 \;,$$
$$x^2 := z + \mathbf{1}_a - \mathbf{1}_v \;\in\; T_2 \;.$$

Now $y, z \in Y(c)$ and $x^0, x^2 \in S$ imply the contradiction

$$c_a - c_v = cy - cx^0 > 0 \;,$$
$$c_v - c_a = cz - cx^2 > 0 \;.$$

This implies that all vectors in $Y(c)$ are of the form $\mathbf{1}_A + \mathbf{1}_B$ with either $A \in \binom{I}{m+1}$ fixed, in which case $|Y(c)| \leq \binom{2m}{m-1}$, or $B \in \binom{J}{m-1}$ fixed, in which case $|Y(c)| \leq \binom{2m}{m+1} = \binom{2m}{m-1}$ , as claimed.

Continuing with the proof of our theorem, consider any algorithm, and let $c^1, \ldots, c^p \in \mathbb{Z}^n$ be the sequence of oracle queries made by the algorithm. Suppose that $p < \binom{2m}{m+1}$ . Then

$$\left|\bigcup_{i=1}^{p} Y(c^i)\right| \;\leq\; \sum_{i=1}^{p} |Y(c^i)| \;\leq\; p\binom{2m}{m-1} \;<\; \binom{2m}{m+1}\binom{2m}{m-1} \;=\; |T_1| \;.$$

This implies that there exists some $y \in T_1$ that is an element of none of the $Y(c^i)$ , that is, satisfies $c^i y \leq \max\{c^i x : x \in S\}$ for each $i = 1, \ldots, p$ . Therefore, whether the linear optimization oracle presents $S$ or $S_y$ , on each query $c^i$ it can reply with some $x^i \in S$ attaining

$$c^i x^i \;=\; \max\{c^i x : x \in S\} \;=\; \max\{c^i x : x \in S_y\} \;.$$

Therefore, the algorithm cannot tell whether the oracle presents $S$ or $S_y$ and hence can neither compute the $w$-image of the independence system nor solve the nonlinear optimization problem correctly. □

## 8 Discussion

We view this article as a first step in understanding the complexity of the general nonlinear optimization problem over an independence system presented by an oracle. Our work raises many intriguing questions including the following. Can the saturated $\lambda$ for $a$ be better understood or even characterized? Can a saturated $\lambda$ smaller than that with $\lambda_i = \max(a)$ be determined for every $a$ and

be used to obtain better running-time guarantee for the algorithm of Theorem 1 and better approximation quality $r(a)$ ? Can tighter bounds on $r(a)$ in equation (7) and $g(a)$ in equation (8) and possibly formulas for $r(a)$ and $g(a)$ for small values of $p$, in particular $p = 3$, be derived? For which primitive $p$-tuples $a$ can an exact solution to the nonlinear optimization problem over a (singly) weighted independence system be obtained in polynomial time, at least for small $p$, in particular $p = 2$ ? For $p = 2$ we know that we can when $a_1$ divides $a_2$ , and we cannot when $a := (2, 3)$ , but we do not have a complete characterization. How about $d = 2$ ? While this includes the notorious exact matching problem [5] as a special case, it may still be that a polynomial-time solution is possible. And how about larger, but fixed, $d$ ?

In another direction, it can be interesting to consider the problem for functions $f$ with some structure that helps to localize minima. For instance, if $f : \mathbb{R} \to \mathbb{R}$ is concave or even more generally quasiconcave (that is, its "upper level sets" $\{z \in \mathbb{R} \ : \ f(z) \geq \tilde{f}\}$ are convex subsets of $\mathbb{R}$ , for all $\tilde{f} \in \mathbb{R}$ ), then the optimal value $\min\{f(wx) \ : \ x \in S\}$ is always attained on the boundary of $\text{conv}(w \cdot S)$ , i.e., if $x^*$ is a minimizer, then either $wx^* = 0$ or $wx^*$ attains $\max\{wx \ : \ x \in S\}$ , so the problem is easily solvable by a single query to the linear-optimization oracle presenting $S$ and a single query to the comparison oracle of $f$ . Also, if $f$ is convex or even more generally quasiconvex (that is, its "lower level sets" $\{z \in \mathbb{R} \ : \ f(z) \leq \tilde{f}\}$ are convex subsets of $\mathbb{R}$ , for all $\tilde{f} \in \mathbb{R}$), then a much simplified version of the algorithm (from the proof of Theorem 4) gives an $r$-best solution as well, as follows (see our preprint [4] for a proof).

**Proposition 5.** *For every primitive $p$-tuple $a = (a_1, \ldots, a_p)$ , there is an algorithm that, given independence system $S \subseteq \{0, 1\}^n$ presented by a linear-optimization oracle, weight vector $w \in \{a_1, \ldots, a_p\}^n$ , and quasiconvex function $f : \mathbb{R} \to \mathbb{R}$ presented by a comparison oracle, provides a $(\max(a) - 1)$-best solution to the nonlinear problem $\min\{f(wx) : x \in S\}$, in time polynomial in $n$.*

## References

1. Berstein, Y., Lee, J., Maruri-Aguilar, H., Onn, S., Riccomagno, E., Weismantel, R., Wynn, H.: Nonlinear matroid optimization and experimental design. SIAM Journal on Discrete Mathematics 22, 901–919 (2008)
2. Berstein, Y., Onn, S.: Nonlinear bipartite matching. Discrete Optimization 5, 53–65 (2008)
3. Brauer, A.: On a problem of partitions. American Journal of Mathematics 64, 299–312 (1942)
4. Lee, J., Onn, S., Weismantel, R.: Nonlinear optimization over a weighted independence system. Oberwolfach Preprint Series OWP 2008 – 10, `http://www.mfo.de/publications/owp/2008/OWP2008_10.pdf`
5. Mulmuley, K., Vazirani, U.V., Vazirani, V.V.: Matching is as easy as matrix inversion. Combinatorica 7, 105–113 (1987)

# Improved Online Algorithms for Multiplexing Weighted Packets in Bounded Buffers*

Fei Li

Department of Computer Science, George Mason University, Fairfax, VA 22030
lifei@cs.gmu.edu

**Abstract.** Motivated by providing differentiated services in the Internet, we consider online buffer management algorithms for quality-of-service network switches. We study a *multi-buffer model*. Packets have values and deadlines; they arrive at a switch over time. The switch consists of multiple buffers whose sizes are bounded. In each time step, only one pending packet can be sent. Our objective is to maximize the total value of the packets sent by their deadlines. We employ competitive analysis to measure an online algorithm's performance. In this paper, we first show that the lower bound of competitive ratio of a broad family of online algorithms is 2. Then we propose a $(3+\sqrt{3} \approx 4.723)$-competitive deterministic algorithm, which is improved from the previously best-known result 9.82 (Azar and Levy. SWAT 2006).

## 1 Introduction

Motivated by providing quality-of-service for the next-generation networks, we study a model called a *multi-buffer model*. Time is discretized into time steps. Packets arrive at a switch over time and each packet $p$ is associated with an integer arriving time (release time) $r_p \in \mathbb{R}^+$, a non-negative weight $w_p \in \mathbb{R}^+$, an integer deadline $d_p \in \mathbb{Z}^+$ $(d_p \geq r_p, \forall p)$, and a target buffer (of the switch) that it can reside in. In the context of this paper, we use "value" and "weight" interchangeably. The deadline $d_p$ specifies the time by which $p$ should be sent. This model is preemptive: Packets already existing in the buffers can be dropped any time before they are transmitted. A dropped packet cannot be delivered any more. The switch consists of $m$ size-bounded buffers: $B_1, B_2, \ldots, B_m$. At any time, each buffer $B_i$ queues at most $b_i \in \mathbb{Z}^+$ packets, $\forall i = 1, 2, \ldots, m$. (If $m = 1$, we call this variant a *single-buffer model*.) A packet has only one destined buffer to stay in before it is either sent or preempted/dropped. In each time step, at most one pending packet can be sent. Our objective is to maximize *weighted throughput*, defined as the total value of the transmitted packets by their respective deadlines.

Scheduling packets with deadlines is essentially an online decision problem. In order to evaluate the worst-case performance of an online algorithm lacking

* Research is partially supported by the Seed Grant from the Office of the Vice President for Research and Economic Development at George Mason University.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 265–278, 2009.

of future input information, we compare it with a *clairvoyant* algorithm which is empowered to know the whole input sequence in advance to make its decision. In contrast to stochastic algorithms that provide statistical guarantees under some mild assumptions on input sequences, *competitive online algorithms* guarantee the *worst-case performance*. Moreover, competitive analysis is of fundamental importance, if a reasonable approximation of the input probability distribution is not available or reliable, or if analytical worst-case performance guarantee is what we are seeking for.

**Definition 1. Competitive ratio** *[4]. A deterministic online algorithm* ON *is called* $k$-competitive *if its weighted throughput on* any *instance is at least* $1/k$ *of the weighted throughput of an optimal offline algorithm on the same instance:*

$$k = \max_{\mathcal{I}} \frac{\mathtt{OPT}(\mathcal{I}) - \gamma}{\mathtt{ON}(\mathcal{I})}$$

*where* $\gamma$ *is a constant,* $\mathtt{OPT}(\mathcal{I})$ *is the optimal solution of an input* $\mathcal{I}$*. The parameter* $k$ *is known as* ON*'s* competitive ratio*. The optimal offline algorithm* OPT *is also called* adversary*.*

The *upper bounds* of competitive ratio are achieved by some known online algorithms. A competitive ratio less than the *lower bound* is not reachable by *any* online algorithm. If the additive constant $\gamma$ is no larger than 0, the online algorithm ON is called *strictly* $k$*-competitive*. In this paper, we design and analyze better online algorithms, in terms of competitive ratio, for scheduling weighted packets with deadlines in bounded buffers.

### 1.1 Related Work

The first quality-of-service buffer management model is introduced in [1]. Since then, quite a few researchers have studied this model as well as its variants [5], [9], [10], [11], [13], [14]. Many previous studies address the single buffer case with an infinitely-large buffer. Such model is called a *bounded-delay model*. The only work addressing scheduling packets in size-bounded buffer(s) is the *multi-buffer model* proposed by Azar and Levy in [3]. A deterministic 9.82-competitive algorithm is presented. This is the model we study in this paper. Note that if the buffer sizes are unlimited, the multi-buffer model is the same as the bounded-delay buffer model. Thus, the lower bound of competitive ratio $\phi := (1 + \sqrt{5})/2 \approx 1.618$ [2], [5] for the bounded-delay model directly applies to the multi-buffer model. For the size-bounded *single-buffer model* (that is, when the number of buffers $m = 1$), a 3-competitive deterministic algorithm and a ($\phi^2 \approx 2.618$)-competitive randomized algorithm have been proposed in [12].

### 1.2 Our Contributions

Our main contributions in this paper include a $(3 + \sqrt{3} \approx 4.732)$-competitive deterministic algorithm and a lower bound 2 of competitive ratio for a broad family of algorithms. Both lower bounds and upper bounds of competitive ratio of the multi-buffer model are improved.

## 2 Algorithms and Analysis

At first, we introduce a few concepts. Then we prove the lower bound 2 of competitive ratio for a broad family of online algorithms. At last, we present an online algorithm and its analysis.

**Definition 2. Provisional schedule** *[6], [9]. At any time $t$, a* provisional schedule $\mathbf{S}_t$ *is a schedule for the pending packets at time $t$ (assuming no new arriving packets). This schedule specifies the set of packets to be transmitted, and for each it specifies the delivery time.*

**Definition 3. Optimal provisional schedule** *[6], [9]. Given a set of pending packets, an* optimal provisional schedule *is the one achieving the maximum total value of packets among all provisional schedules.*

We use $\mathbf{S}_t$ to denote both the provisional schedule for time steps $[t,\ +\infty)$ and the set of packets specified by the schedule. All known online algorithms for the bounded-delay model [5], [7], [9], [10], [11], [13], [14] calculate the optimal provisional schedules at the beginning of each time step. These algorithms differ only by the packets they select to send. The online algorithms in such a broad family are defined as the *best-effort admission algorithms.*

**Definition 4. Best-effort admission algorithm**. *Consider an online algorithm* `ON` *and a set of pending packets $\mathbf{P}_t$ at any time $t$. If* `ON` *calculates the optimal provisional schedule $\mathbf{S}_t$ on $\mathbf{P}_t$ and selects one packet from $\mathbf{S}_t$ to send in step $t$, we call* `ON` *a* best-effort admission algorithm.

### 2.1 The Lower Bound of Competitive Ratio

In this section, we create an instance to prove that the lower bound of competitive ratio for all best-effort admission algorithms is 2. Note that for the bounded-delay model in which the single buffer size $b = +\infty$, the lower bound of competitive ratio is $\phi \approx 1.618$ [5], [1]. This lower bound holds for best-effort admission algorithms as well. Here, we improve the lower bound from 1.618 to 2 for the model in which buffer sizes are restricted.

**Theorem 1.** *Consider the single-buffer model in which the number of buffers $m = 1$. The lower bound of competitive ratio for best-effort admission algorithms is 2. This lower bound holds for the multi-buffer model in which $m > 1$ as well.*

*Proof.* In the following instance, we show: *If the buffer size is bounded, the packets that the optimal offline algorithm chooses to send may not be from the optimal provisional schedule calculated by the online algorithm, even if both algorithms have the same set of pending packets.* This property does not hold for the bounded-delay model; and it leads that any best-effort admission algorithm cannot achieve a competitive ratio better than 2.

Assume the buffer size is $b$. Let a best-effort admission online algorithm be `ON`. We use $(w_p,\ d_p)$ to represent a packet $p$ with a value $w_p$ and a deadline $d_p$.

Initially, the buffer is empty. A set of packets are released: $(1,\ b+1)$, $(1,\ b+2),\ \ldots,\ (1,\ b+b)$. Notice that all packets released have deadlines larger than the buffer size $b$. The optimal offline algorithm will accept $b-1$ packets: It drops $(1,\ b+1)$, and keeps $(1,\ b+2),\ \ldots,\ (1,\ b+b)$ in its buffer. On the other hand, based on its definition, upon these packets' arrival, ON will select all of them to put into its buffer.

In the same time step, $b$ packets $(1+\epsilon,\ 1),\ (1+\epsilon,\ 2),\ \ldots,\ (1+\epsilon,\ b)$ are released afterwards. Then there are no more new packets arriving in this step. The optimal offline algorithm only accepts $(1+\epsilon,\ 1)$. Instead, ON calculates the optimal provisional schedule in step 1 which includes all these newly arriving packets with value $1+\epsilon$. All such packets will be accepted by ON, but the packets $(1,\ b+i)$, $\forall i = 1,\ 2,\ \ldots,\ b$, will be dropped due to the buffer size constraint. After processing arrivals in step 1, the optimal offline algorithm sends the packet $(1+\epsilon,\ 1)$. Since ON's buffer is full of packets with value $1+\epsilon$, ON sends one of these packets in the first step.

At the beginning of each step $i = 2,\ 3,\ \ldots,\ b$, only one packet $(1+\epsilon,\ i)$ is released. Since the end of step $b$, no new packets will be released. Note that since the time after the first step, all packets available to ON have their deadlines $\leq b$ and values $\leq (1+\epsilon)$. Thus, in the time steps $2,\ 3,\ \ldots,\ b$, ON cannot schedule packets with a total value $> (1+\epsilon)\cdot(b-1)$.

Since there is one empty buffer slot at the beginning of each time step $i = 2,\ 3,\ \ldots,\ b$, the optimal offline algorithm can accept and send all newly released packets $(1+\epsilon,\ i)$ in steps $i = 2,\ 3,\ \ldots,\ b$. At the end of step $b$, the packets $(1,\ b+2),\ (1,\ b+3),\ \ldots,\ (1,\ b+b)$ are still remained in the optimal offline algorithm's buffer (they are not in ON's buffer though). Since there is no future arrivals, these $b-1$ packets with value 1 will be transmitted eventually by the optimal algorithm in the following $b-1$ steps. The total value of ON achieves is $(1+\epsilon)\cdot b$ while the optimal offline algorithm gets a total value $(1+\epsilon)\cdot b+1\cdot(b-1)$. The competitive ratio for this instance is

$$c = \frac{(1+\epsilon)\cdot b + 1\cdot(b-1)}{(1+\epsilon)\cdot b} = 2 - \frac{1+b\cdot\epsilon}{b+b\cdot\epsilon} \geq 2 - \frac{2}{b}, \quad \text{if } \epsilon\cdot b = 1 \text{ and } b \geq 2.$$

If $b$ is large, ON cannot perform asymptotically better than 2-competitive. This lose is due to ON calculating an optimal provisional schedule to find out the packets to send in each time step. Theorem 1 is proved. ■

### 2.2 A Deterministic Online Algorithm DON

Our idea straightforwardly follows the logic of the modified earliest-deadline-first policy: $\texttt{EDF}_\alpha$. In each time step, we calculate a provisional schedule. Then, we identify the *earliest-deadline packet* $e$ and the *maximum-value packet* $h$ (ties broken in favor of the earliest-deadline one). If $e$ is with a sufficiently large value compared with $h$ (say, $w_e \geq w_h/\alpha$ for some $\alpha \geq 1$), then $e$ is sent. Otherwise, $h$ is sent. (If $\alpha = 1$, $\texttt{EDF}_\alpha$ is the greedy algorithm.) However, because the buffers are size-bounded, there is still one question (also, the most important question in designing online algorithms for the multi-buffer model) unanswered yet:

*Problem 1.* Given a set of packets, how do we identify a subset of them to put into the size-bounded buffers as pending packets?

Unlike the bounded-delay model with a single buffer, in the multi-buffer model, there are two constraints to locate packets in the buffers in addressing Problem 1: ($C_1$) Not all unexpired-yet packets can be put into the buffers due to the buffer size constraints; and ($C_2$) the order of the buffers that the optimal offline algorithm chooses packets from to send is unknown to the online algorithms. The first constraint has been expressed in the lower bound construction (see Theorem 1). On the second constraint, we here show that if we consider each buffer separately (that is, we maximize the value gained by the buffer $B_i$, assuming the packets selected to send are from this buffer only since a time step $t$), we will conclude that any best-effort admission algorithm cannot perform better than $m$-competitive, where $m$ is the number of buffers. Consider the following instance:

*Example 1.* A packet $p$ is represented by a triple $(w_p,\ d_p,\ b_p)$, where $b_p$ is the buffer that $p$ should be in. Without loss of generality, we assume all buffers have the same size $b$.

Initially, we have $2 \cdot b$ packets arriving at the buffer $B_i$, $\forall i = 1,\ 2,\ \ldots,\ m$: $(1+\epsilon,\ j,\ B_i)$ and $(1,\ (i-1)\cdot b + j,\ B_i)$, $\forall j = 1,\ 2,\ \ldots,\ b$. There will be no future packet arrivals.

Consider a best-effort admission algorithm `ON`. If `ON` considers each buffer separately, all packets $(1+\epsilon,\ j,\ B_i)$, $\forall j = 1,\ 2,\ \ldots,\ b$, will be stored in the buffer $B_i$. Note that all stored packets have deadlines $\leq b$, thus, `ON` achieves a total value $\leq (1+\epsilon)\cdot b$. On the other hand, the optimal offline algorithm picks the packets $(1,\ (i-1)\cdot b + j,\ B_i)$, $\forall j = 1,\ 2,\ \ldots,\ b$, to store into the buffer $B_i$. All such packets will be successfully delivered from the buffers $B_1,\ B_2,\ \ldots, B_m$ consequently. Hence, the optimal offline algorithm achieves a total value $1 \cdot b \cdot m$, asymptotically $m$ times of what `ON` gains. For this instance, `ON`'s competitive ratio is $m/(1+\epsilon)$. ■

Example 1 motivates us to identify the provisional schedule with the considerations of *the packets already in the buffer, new packet arrivals, and their respectively buffers* for a given time step $t$. In our algorithm `DON`, we apply two actions on the pending packets $\mathbf{P}$:

- Identify a provisional schedule $\mathbf{S}$, $\mathbf{S} \subseteq \mathbf{P}$. All packets in $\mathbf{S}$ are put into the buffers; and they are feasibly sent if there are no future arrivals.
- For those packets $\in (\mathbf{P} \setminus \mathbf{S})$ (if any), we simply drop them.

Motivated by Theorem 1 (on the constraint $C_1$ above) and Example 1 (on the constraint $C_2$ above), we will detail a procedure `PS`($\mathbf{P}$, $t$) in identifying a provisional schedule $\mathbf{S}$ for a set of packets $\mathbf{P}$ at a time $t$; this procedure is different from the one described in [3]. We note that $\mathbf{S}$ may not be optimal but 2-approximation (see Lemma 2).

**How do we calculate a provisional schedule?** Let us consider a set of pending packets $\mathbf{P}$. Notice that for any algorithm, it can send at most $L$ ($L := \sum_{l=1}^{m} b_l$) packets from $\mathbf{P}$, given the assumption of no future arrivals. Thus, we identify at most $L$ packets from $\mathbf{P}$, in a greedy manner, to fill in a "super-queue" $Q$ with size $L$ in calculating a provisional schedule starting at time $t$. Note that the "super-queue" $Q$ is a supplementary data structure only but not a real buffer to store packets.

Assume the buffer slots in $Q$ are indexed as 1, 2, ..., $L$. Without loss of generality, we assume the current time $t = 1$. We sort all pending packets $\mathbf{P}$ in non-increasing weight order and fill each packet $p \in \mathbf{P}$ in the "super-queue" $Q$ as *later* as it could be — an empty buffer slot $i$ with a larger index satisfying $d_p \geq i$ is favored to store $p$. Then, if a packet $p$ can be filled in an empty buffer slot of the "super-queue", we try to put $p$ into its own target buffer (which has been specified when $p$ is released) *as long as* there is an empty buffer slot there as well. If either $p$'s buffer is full or $p$ cannot find an empty buffer slot in the "super-queue" $Q$, $p$ is discarded. We extract and examine each packet in $\mathbf{P}$ in order till $\mathbf{P}$ becomes empty. The procedure $\texttt{PS}(\mathbf{P}, t)$ of locating $\mathbf{S}$ from the pending packets $\mathbf{P}$ in the buffers at time $t$ is described in Algorithm 1.

**Algorithm 1.** $\texttt{PS}(\mathbf{P}, t)$

```
1: Sort all packets in P in decreasing weight order, with ties broken in favor of the
   larger deadlines. S is initialized as an empty set.
2: while P ≠ ∅ do
3:    Pick up a packet p from P.
4:    for each buffer slot i indexed from min{t_p − t, Σ_{l=1}^{m} b_l} down to 1 in the super-
      queue Q do
5:       if the super-queue's buffer slot indexed as i is empty and there is a buffer slot
         in p's target buffer then
6:          Put p into the super-queue's buffer slot i and its target buffer. S ← S ∪ p.
7:          Break.
8:       end if
9:    end for
10: end while
11: Sort the selected packets in each buffer in increasing deadline order with ties broken
    in favor of the larger-value ones.
```

Immediately from Algorithm 1, we have:

*Property 1.* The maximum-value packet (with ties broken in favor of the earliest-deadline one) $h \in \mathbf{P}$ is selected in $\mathbf{S}$.

**Lemma 1.** *If the number of buffers $m = 1$, $\texttt{PS}(\mathbf{P}, t)$ calculates an optimal provisional schedule.*

*Proof.* This is a standard greedy algorithm, whose correctness follows from the fact that feasible schedules form a matroid [8]. ■

Note that for multiple bounded buffers ($m > 1$), we have not developed an efficient algorithm in identifying the optimal provisional schedule $\mathbf{S}^*$ nor proved its NP-completeness. Specifically, we have

**Lemma 2.** *Given a set of pending packets* $\mathbf{P}$*, the set of packets* $\mathbf{S}$ *selected by* $\mathtt{PS}(\mathbf{P},\ t)$ *is with a total value at least* $1/2$ *of that of an optimal provisional schedule* $\mathbf{S}^*$ *out of* $\mathbf{P}$*. For each packet* $p^* \in (\mathbf{S}^* \setminus \mathbf{S})$*, there is a uniquely corresponding packet* $p \in \mathbf{S}$ *with* $w_p \geq w_{p^*}$*.*

*Proof.* Let the subset of packets chosen by an optimal provisional algorithm (respectively, $\mathtt{PS}(\mathbf{P},\ t)$) be $\mathbf{S}^*$ (respectively, $\mathbf{S}$). We show that for any packet $p \in (\mathbf{S}^* \setminus \mathbf{S})$, if $p$ is rejected, one of the following cases must happen:

- We can uniquely locate a packet $q$ in $Q$ corresponding to $p$ with $w_q \geq w_p$; or
- We can uniquely locate a packet $q$ in the same target buffer as $p$ is in with $w_q \geq w_p$.

This packet $q$ has been already chosen by PS (that is, $q \in \mathbf{S}$) at the time when $p$ is being evaluated by PS. Thus, we can always find such a unique corresponding packet $q \in \mathbf{S}$ with $w_q \geq w_p$ for $p$. Note

$$\sum_{p \in \mathbf{S}^*} w_p = \sum_{p \in (\mathbf{S}^* \cap \mathbf{S})} w_p + \sum_{p \in (\mathbf{S}^* \setminus \mathbf{S})} w_p \leq \sum_{p \in \mathbf{S}} w_p + \sum_{q \in \mathbf{S}} w_q \leq 2 \cdot \sum_{p \in \mathbf{S}} w_p.$$

Thus, Lemma 2 is proved. ■

The following example shows that for PS, the approximation ratio 2 is tight. This infers that the calculated provisional schedule $\mathbf{S}$ $(= \mathtt{PS}(\mathbf{P},\ t))$ may not be optimal for $\mathbf{P}$ in the interval $[t,\ +\infty)$.

*Example 2.* Consider two buffers $B_1$ and $B_2$ with the same buffer size $b$. We use a triple $(w_p,\ d_p,\ b_p)$ to denote a packet $p$ with a weight $w_p$, a deadline $d_p$, and its target buffer $b_p$.

For the buffer $B_1$, $2 \cdot b$ packets are available: $(1+\epsilon,\ i,\ B_1)$ and $(1,\ b+i,\ B_1)$, $\forall i = 1,\ 2,\ \ldots,\ b$. For the buffer $B_2$, $b$ packets are available: $(1,\ i,\ B_2)$, $\forall i = 1,\ 2,\ \ldots,\ b$. There are no other packets.

PS will store packets $(1+\epsilon,\ i,\ B_1)$, $\forall i = 1,\ 2,\ \ldots,\ b$, in the buffer $B_1$ and no packets will be stored in the buffer $B_2$. On the other hand, an optimal provisional schedule will store packets $(1,\ b+i,\ B_1)$ for the buffer $B_1$ and $(1,\ i,\ B_2)$ for the buffer $B_2$, $\forall i = 1,\ 2,\ \ldots,\ b$. It is easy to see that the optimal provisional schedule achieves a total value $2 \cdot b$ of packets, which is asymptotically 2 times as what PS stores $(1+\epsilon) \cdot b$. ■

Note that given a set of pending packets without future arrivals, any algorithm has a total value no more than two times of what our provisional schedule (calculated by PS) has.

**How does the online algorithm DON work?** Now, we are ready to introduce the deterministic online algorithm DON. In each time step, given a provisional schedule, either the earliest-deadline packet (if it has a sufficiently large value) or the maximum-value packet in the buffers is sent.

Before we choose a packet to send, we employ another technique in admitting packets. In calculating the provisional schedule, we associate "*virtual deadlines*" rather than the deadlines specified in the input sequence with packets. A motivating example of introducing virtual deadlines has been presented in [12], and to save space, we skip repeating it. A packet's virtual deadline is initialized as its deadline specified at its arrival. Then, the virtual deadline will be updated to be the tentative time step in which the provisional schedule sends it. Since we use packets' virtual deadlines in calculating a provisional schedule, hence we may update a packet's virtual deadline along the course of our algorithm. To avoid confusion, we still use $d_p$ to denote a packet $p$'s virtual deadline in DON's buffer.

Assume the set of pending packets at time $t$ be $\mathbf{P}$. In each time step, our algorithm works as in Algorithm 2. The delivery part of the algorithm (that is, line 3 to line 7 of Algorithm 2) is the same as the one described in [3].

**Algorithm 2.** DON($\mathbf{P}$, $t$)

1: $\mathbf{S} = \mathtt{PS}(\mathbf{P},\ t)$. Drop the packets $\in (\mathbf{P} \setminus \mathbf{S})$.
   {$\mathbf{S}$ is the provisional schedule.}
2: Update a packet $p \in \mathbf{S}$'s virtual deadline $d_p$, if necessary, to be the time step $\mathbf{S}$ supposes to send $p$.
   {All packets in $\mathbf{S}$ have distinct virtual deadlines: $t,\ t+1,\ t+2,\ \ldots$.}
3: Let $e$ be the earliest-(virtual)-deadline packet in $\mathbf{S}$ and $h$ be the maximum-value packet in $\mathbf{S}$, with ties broken in favor of the earliest virtual deadline one.
   {Note $d_e = t$.}
4: **if** $w_e \geq w_h/\alpha$ **then**
5:  Send $e$.
6: **else**
7:  Send $h$ (and $e$ expires).
8: **end if**

### 2.3 Analysis of DON

Our main result in this section is

**Theorem 2.** DON *is* $(3 + \sqrt{3} \approx 4.732)$*-competitive in maximizing weighted throughput for the multi-buffer model, where* $\alpha = 1 + \sqrt{3} \approx 2.732$.

*Proof.* Let $\Gamma := \tau_1\tau_2\ldots$ be a series of events: arrival events and delivery events. Those arrival events construct the packet input sequence $\mathcal{I}$. We use OPT to denote the optimal offline algorithm. Let the set of packets sent by OPT be $\mathcal{O} := \{p_1,\ p_2,\ \ldots,\ p_{|\mathcal{O}|}\}$, the packet $p_i$ is sent in the time step $i$. If there is nothing to send in a time step $i$, we let $p_i$ be a null packet with value $w_{p_i} = 0$

and deadline $d_{p_i} = i$. More specifically, we let $\mathcal{O}_t$ denote the set of packets that OPT will deliver in steps $[t, +\infty)$. Clearly, $\mathcal{O} = \mathcal{O}_1$. Without loss of generality, we enforce OPT to be the one such that

*Remark 1.* OPT only accepts the packets that it will send.

*Remark 2.* When OPT chooses a packet from the buffer $B_i$ to send, it always picks up the packet with the earliest deadline.

The following observation will be used in our analysis:

*Remark 3.* Given a packet $p_i \in \mathcal{O}$, which is supposed to be sent by OPT in the step $i$, we can always modify the deadline $d_{p_i}$ to $d'_{p_i}$ for $p_i$ in OPT's buffers, as long as $i \leq d'_{p_i} \leq d_{p_i}$.

In our analysis, we feed DON and OPT the same original input $\mathcal{I}$. However, we may modify the deadlines of packets in OPT's buffers (as in Remark 3) to simplify the invariants (see below). One aspect of modifying packets in OPT's buffer is to make sure: Given a packet $p$ in DON's buffer $B_i$ and a packet $q$ in OPT's same buffer $B_i$, if $w_p = w_q$ and $d_p = d'_q$, we can regard $p$ and $q$ the same packet. Also, we use $d'_q$ to represent the deadline of a packet $q$ in OPT's buffers. For each new packet $q \in \mathcal{O}$, $d'_q$ is initialized as $d_q$.

We prove DON's competitiveness using the analysis approach proposed in [13], [14]. Recall that the potential function approach is a commonly used method in analyzing online algorithms [4]. However, we do not use the potential function argument explicitly. Instead, our analysis relies on modifying DON's buffers (using virtual deadlines instead of real ones) and OPT's buffers (such as changing the deadlines from $d_{p_i}$ to $d'_{p_i}$) as well as the packet sending sequence $\mathcal{O}_t$ judiciously in each step $t$. We need to assign an appropriate credit to OPT to account for these modifications but keep a set of invariants always hold for each time step. Then, we bound the competitive ratio by comparing the modified OPT's gain to that of DON's in each step.

We use $Q_t^{\texttt{DON}}$ (respectively, $Q_t^{\texttt{OPT}}$) to denote the buffers of DON (respectively, OPT) at time $t$. We use $\Phi(Q_t^{\texttt{DON}})$ (respectively, $\Phi(Q_t^{\texttt{OPT}})$) to denote the potential of the buffers of $Q_t^{\texttt{DON}}$ (respectively, $Q_t^{\texttt{OPT}}$). In a step $t$, we use $\mathbf{X}_t^{\texttt{DON}}$ and $\mathbf{X}_t^{\texttt{OPT}}$ to denote the set of packets sent by DON and charged (by us) to OPT respectively. Our goal is to prove that at the end of each event, the main Inequality 1 holds.

$$c \cdot \sum_{j \in \mathbf{X}_t^{\texttt{DON}}} w_j + \Phi(Q_t^{\texttt{DON}}) \geq \sum_{k \in \mathbf{X}_t^{\texttt{OPT}}} w_k + \Phi(Q_t^{\texttt{OPT}}). \tag{1}$$

where $c = 3 + \sqrt{3} \approx 4.732$. As a consequence, this yields Theorem 2.

In our analysis, we will prove that the competitive ratio $c$ of DON is ($\alpha \geq 1$)

$$c \leq \max\{2 + \alpha,\ 4 + 2/\alpha\}.$$

$c$ is optimized at $3 + \sqrt{3} \approx 4.732$ when we set $\alpha = 1 + \sqrt{3} \approx 2.732$. In order to prove Inequality 1, we first present a set of invariants. We then prove them hold at the end of each event.

$I_1$. $c \cdot \sum_{j \in \mathbf{X}_t^{\texttt{DON}}} w_j \geq \sum_{k \in \mathbf{X}_t^{\texttt{OPT}}} w_k$.

$I_2$. For each packet $p \in Q_t^{\texttt{DON}}$, $p$ maps to at most one packet $p' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ with $w_p \geq w_{p'}$ and $p$ and $p'$ have the same target buffer.
For each packet $p \in (Q_t^{\texttt{DON}} \setminus Q_t^{\texttt{OPT}})$, $p$ maps to at most one packet $q \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ with $w_p \geq w_q$, $d_p \geq d'_q$, and $p$ and $q$ *do not have to* have the same target buffer.
(Hence a packet $p \in (Q_t^{\texttt{DON}} \setminus Q_t^{\texttt{OPT}})$ *may* map two distinct packets, a packet $p' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ in same target buffer and a packet $q \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$, which may or may not be in the same target buffer as $p$.)

$I_3$. For each packet $q' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$, $q'$ must be mapped uniquely by a packet $p \in Q_t^{\texttt{DON}}$.
(Such $q'$ must be either the packet $p'$ or the packet $q$ in the invariant $I_2$.)

$I_4$. Assume a packet $p \in Q_t^{\texttt{DON}}$ maps a packet $p' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$. For any packet $i \in Q_t^{\texttt{DON}}$ with the same target buffer as $p'$, either $d_i \geq d'_{p'}$ or $w_i \geq w_{p'}$.

We prove that the set of invariants hold separately for the events of both packet arrivals and packet deliveries. In the following, we case study to prove the existence of these invariants.

**Packet deliveries.** DON either delivers the earliest-deadline packet $e$ or the maximum-value packet $h$ in the current buffers. We assume OPT sends $j$. From Invariant $I_3$ and Property 1, we claim that $w_j \leq w_h$, no matter whether $j$ is in the provisional schedule ($j$ is in DON's buffer; $j \in (Q_t^{\texttt{DON}} \cap Q_t^{\texttt{OPT}})$) or not ($j \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$). For the earliest-virtual-deadline packet $e$ in the provisional schedule, we have $w_e \leq w_h$ and $d_e = t$.

We summarize all the possible consequences into the following 4 cases, based on the packet DON sends and the packet OPT sends in each step. We use $W_t$ and $V_t$ to denote the value we charge to (the modified) OPT and DON, respectively. To prove Invariant $I_1$, we need to show that $W_t/V_t \leq \max\{2+\alpha,\ 4+2/\alpha\}$.

Note that if a considered packet $\in Q_t^{\texttt{DON}}$ is not in a mapping, we assume its "mapped" packets be null packets with value 0. This facilitates us that we can assume every packet in $Q_t^{\texttt{DON}}$ has mapped packets in $Q_t^{\texttt{OPT}}$. Also, remember that every packet in DON's buffers are in the provisional schedule.

1. Assume DON sends $e$ and $j \in (Q_t^{\texttt{DON}} \cap Q_t^{\texttt{OPT}})$.
   $w_j \leq w_h$. Since DON sends $e$, $w_e \geq w_h/\alpha \geq w_j/\alpha$.
   - If $j = e$, we charge DON a value $w_e$, and we charge OPT a value $w_j + w_{j'}$ (since $j$ may map another $\mathcal{O}$-packet $j' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$). We remove $j$ and $j'$ out of OPT's buffers. Then
   $$\frac{W_t}{V_t} = \frac{w_j + w_{j'}}{w_e} \leq \frac{w_j + w_j}{w_e} = \frac{w_e + w_e}{w_e} = 2.$$
   - If $j \neq e$, then $e \notin Q_t^{\texttt{OPT}}$ (otherwise, OPT will send $e$ in this step to avoid losing its value; see Remark 2).
     We charge DON a value $w_e$, and we charge OPT a value $w_{e'} + w_{e''} + w_j$. Note $e \notin Q_t^{\texttt{OPT}}$ may map an $\mathcal{O}$-packet $e' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ in the same target

buffer as $e$ is in, and another $\mathcal{O}$-packet $e'' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ in a possibly different target buffer from which $e$ is in. We remove $e'$, $e''$, and $j$ out of OPT's buffers. Based on Invariants $I_2$ and $I_3$, $w_e \geq \max\{w_{e'},\ w_{e''}\}$. Then

$$\frac{W_t}{V_t} = \frac{w_{e'} + w_{e''} + w_j}{w_e} \leq \frac{w_e + w_e + w_h}{w_e} \leq \frac{w_e + w_e + \alpha \cdot w_e}{w_e} = 2 + \alpha.$$

2. Assume DON sends $e$ and $j \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$.
$w_j \leq w_h$. Since DON sends $e$, $w_e \geq w_h/\alpha \geq w_j/\alpha$.
Since $j \neq e$, then $e \notin Q_t^{\texttt{OPT}}$ (otherwise, OPT will send $e$ in this step to avoid losing its value; see Remark 2).
We charge DON a value $w_e$, and we charge OPT a value $w_{e'} + w_{e''} + w_j$. Note $e \notin Q_t^{\texttt{OPT}}$ may map an $\mathcal{O}$-packet $e' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ in the same target buffer as $e$ is in, and another $\mathcal{O}$-packet $e'' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ in a possibly different target buffer from which $e$ is in. We remove $e'$, $e''$, and $j$ out of OPT's buffers. Based on Invariants $I_2$ and $I_3$, $w_e \geq \max\{w_{e'},\ w_{e''}\}$, Then

$$\frac{W_t}{V_t} = \frac{w_{e'} + w_{e''} + w_j}{w_e} \leq \frac{w_e + w_e + w_h}{w_e} \leq \frac{w_e + w_e + \alpha \cdot w_e}{w_e} = 2 + \alpha.$$

3. Assume DON sends $h$ and $j \in (Q_t^{\texttt{DON}} \cap Q_t^{\texttt{OPT}})$.
$w_j \leq w_h$. Since DON sends $h$, $w_e < w_h/\alpha$.
We charge DON a value $w_h$. Assume $e$ maps $e' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ (and $e$ maps $e'' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ if $e \notin \mathcal{O}$). From Invariants $I_2$ and $I_3$, $w_e \geq \max\{w_{e'},\ w_{e''}\}$. Assume $j$ maps $j'$, from Invariant $I_2$, $w_j \geq w_{j'}$. Assume $h$ maps $h' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$. $w_h \geq \max\{w_{h'},\ w_j,\ w_{j'}\}$.
   - If $j = e$, we charge OPT a value $w_j + w_{j'} + w_h + w_{h'}$, and we remove $j$, $j'$, $h$ and $h'$ out of OPT's buffer.
   (If $h \notin Q_t^{\texttt{OPT}}$, we replace $h$ with the possibly mapped packet $h'' \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ in the following $W_t$'s calculation.)

$$\frac{W_t}{V_t} = \frac{w_j + w_{j'} + w_h + w_{h'}}{w_h} \leq \frac{w_e + w_e + w_h + w_{h'}}{w_h}$$
$$\leq \frac{w_h/\alpha + w_h/\alpha + 2 \cdot w_h}{w_h} \leq \frac{(2 + 2/\alpha) \cdot w_h}{w_h} = 2 + 2/\alpha.$$

   - If $j \neq e$, then $e \notin Q_t^{\texttt{OPT}}$ (otherwise, OPT will send $e$ in this step to avoid losing its value; see Remark 2).
     - If $h \in \mathcal{O}$, we charge OPT a value $w_j + w_{j'} + w_{e'} + w_{e''} + w_h + w_{h'}$, and we remove $e'$, $e''$, $j$, $j''$, $h$ and $h'$ out of OPT's buffer.

$$\frac{W_t}{V_t} = \frac{w_j + w_{j'} + w_{e'} + w_{e''} + w_h + w_{h'}}{w_h}$$
$$\leq \frac{w_h + w_h + w_h/\alpha + w_h/\alpha + 2 \cdot w_h}{w_h} \leq \frac{(4 + 2/\alpha) \cdot w_h}{w_h} = 4 + 2/\alpha.$$

- If $h \notin \mathcal{O}$, we charge OPT a value $w_j + w_{j'} + w_{e'} + w_{e''} + w_{h'} + w_{h''}$, and we remove $e'$, $e^{''}$, $j$, $j'$, $h'$ and $h^{''}$ out of OPT's buffer.

$$\frac{W_t}{V_t} = \frac{w_j + w_{j'} + w_{e'} + w_{e''} + w_{h'} + w_{h''}}{w_h} \leq \frac{w_h + w_h + w_h/\alpha + w_h/\alpha + 2 \cdot w_h}{w_h} \leq \frac{(4 + 2/\alpha) \cdot w_h}{w_h} = 4 + 2/\alpha.$$

4. Assume DON sends $h$ and $j \in (Q_t^{\mathtt{OPT}} \setminus Q_t^{\mathtt{DON}})$.
$w_j \leq w_h$. Since DON sends $h$, $w_e < w_h/\alpha$.
We charge DON a value $w_h$. Assume $e$ maps $e' \in (Q_t^{\mathtt{OPT}} \setminus Q_t^{\mathtt{DON}})$ (and $e$ maps $e^{''} \in (Q_t^{\mathtt{OPT}} \setminus Q_t^{\mathtt{DON}})$ if $e \notin \mathcal{O}$). From Invariants $I_2$ and $I_3$, $w_e \geq \max\{w_{e'},\ w_{e''}\}$. Assume $h$ maps $h' \in (Q_t^{\mathtt{OPT}} \setminus Q_t^{\mathtt{DON}})$. $w_h \geq \max\{w_{h'},\ w_j\}$.
   - If $h \in \mathcal{O}$, we charge OPT a value $w_j + w_{e'} + w_{e''} + w_h + w_{h'}$, and we remove $e'$, $e^{''}$, $j$, $h$ and $h'$ out of OPT's buffer.

$$\frac{W_t}{V_t} = \frac{w_j + w_{e'} + w_{e''} + w_h + w_{h'}}{w_h} \leq \frac{w_h + w_h/\alpha + w_h/\alpha + 2 \cdot w_h}{w_h} \leq \frac{(3 + 2/\alpha) \cdot w_h}{w_h} = 3 + 2/\alpha.$$

   - If $h \notin \mathcal{O}$, we charge OPT a value $w_j + w_{e'} + w_{e''} + w_{h'} + w_{h''}$, and we remove $e'$, $e^{''}$, $w_j$, $h'$ and $h^{''}$ out of OPT's buffer.

$$\frac{W_t}{V_t} = \frac{w_j + w_{e'} + w_{e''} + w_{h'} + w_{h''}}{w_h} \leq \frac{w_h + w_h/\alpha + w_h/\alpha + 2 \cdot w_h}{w_h} \leq \frac{(3 + 2/\alpha) \cdot w_h}{w_h} = 3 + 2/\alpha.$$

**Packet arrivals.** Upon new packets arriving, we consider the possible changes of mappings, from the packets in the provisional schedule in DON's buffers $Q_t^{\mathtt{DON}}$ to those packets only in OPT's buffers $(Q_t^{\mathtt{OPT}} \setminus Q_t^{\mathtt{DON}})$, such that we can ensure that Invariants $I_2$ to $I_4$ still hold.

Let $(w_p,\ d_p)$ denote a packet $p$ with value $w_p$ and deadline $d_p$. Besides Remark 3, we also have the following remark to help us modify OPT's buffers appropriately (if necessary) and facilitate us in the mapping reconstruction.

*Remark 4.* Consider any two packets $p = (w_p,\ d_p)$ and $q = (w_q,\ d_q)$ in $Q_t^{\mathtt{OPT}}$. We are allowed to swap these packets' values such that we use $p' = (w_q,\ d_p)$ and $q' = (w_p,\ d_q)$ to replace $p$ and $q$ respectively. We extend this claim such that given any $n$ packets in OPT's buffers with values $\{w_1,\ w_2,\ \ldots,\ w_n\}$, we can always permute their values as $\{w'_1,\ w'_2,\ \ldots,\ w'_n\}$ for them, as long as $\{w'_1,\ w'_2,\ \ldots,\ w'_n\} = \{w_1,\ w_2,\ \ldots,\ w_n\}$.

Consider a continuous part of the provisional schedule (packets) $\mathbf{S}_1$ in $Q_t^{\mathtt{DON}}$ (a packet $j$ is represented by $(w_j,\ d_j)$, where $d_j$ is the virtual deadline)

$$\mathbf{S}_1 :\ i = (w_i,\ t),\ p_1 = (w_{p_1},\ t+1),\ p_2 = (w_{p_2},\ t+2),\ \ldots,\ p_k = (w_{p_k}, t+k).$$

Since accepting $(w_p,\ d_p)$ with $d_p \geq t+k$ enforces $i$ to leave DON's buffers, from Algorithm 1, we have

$$\min\{w_{p_1},\ w_{p_2},\ \ldots,\ w_{p_k},\ w_p\} \geq w_i. \tag{2}$$

Then, the provisional schedule $\mathbf{S}_1$ is updated as follows with packets' new virtual deadlines:

$$\mathbf{S}_2:\ p'_1 = (w_{p_1},\ t),\ p'_2 = (w_{p_2},\ t+1),\ \ldots,\ p'_k = (w_{p_k},\ t+k-1),\ p = (w_p,\ t+k).$$

Now, based on Remark 3, Remark 4 and Inequality 2, we are going to claim the set of packets $\in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ mapped by packets in $\mathbf{S}_1$ can be mapped by $\mathbf{S}_2$ without violating Invariants $I_2$ to $I_4$.

Here, we consider the case when a packet $i$ in mappings is moved out of the provisional schedule after accepting a packet $p$. The case in which an $\mathcal{O}$-packet is accepted by OPT but rejected by DON can be analyzed in a similar way.

*Value constraints in the mappings.* Let us consider the packets $\in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ that are mapped by packets in $\mathbf{S}_1$. Note the set $\{w_{p_1},\ w_{p_2},\ \ldots,\ w_{p_k},\ w_p\}$ dominates the set $\{w_i,\ w_{p_1},\ w_{p_2},\ \ldots,\ w_{p_k}\}$ in values (see Inequality 2). Thus, based on Remark 4, we can always permutate the values of the mapped packets $\in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ such that the value-relationship in Invariants $I_2$ to $I_4$ holds.

*Deadline constraints in the mappings.* We address two issues here: (1) $i \notin Q_t^{\texttt{OPT}}$ is with possible two mapped packets $\in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ but $p \in (Q_t^{\texttt{DON}} \cap Q_t^{\texttt{OPT}})$ is allowed to map at most one packet; and (2) packets $\in \mathbf{S}_2$ have their virtual deadlines updated and earlier than the time step OPT sends them.

First, we claim that we can always find a packet $q \in (\mathbf{S}_1 \setminus Q_t^{\texttt{OPT}})$ which has at most one mapped $\mathcal{O}$-packet $\in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$ before $p$'s arrival; since otherwise, OPT cannot deliver all mapped packets successfully (see the deadline constraints specified in Invariant $I_2$). Note $w_q \geq w_i$ (from Inequality 2), $q$ can replace $i$ to map those mapped packets by $i \in \mathbf{S}_1$.

Second, we take $p$ as an example. Assume $p$, sent in a step $t'$ by OPT, has its virtual deadline updated to $d_p < t'$ in $\mathbf{S}_2$. Then, $p$ cannot map another $\mathcal{O}$-packet with deadline $\leq d_p$; otherwise, OPT cannot deliver all such packets by their deadlines. We regard $p$ as a non-$\mathcal{O}$-packet $p'$ and we get a mapping in Invariant $I_2$: $p' \in Q_t^{\texttt{DON}}$ maps to $p \in (Q_t^{\texttt{OPT}} \setminus Q_t^{\texttt{DON}})$, $p'$ and $p$ have the same target buffer.

Based on our case study at packet arrival and delivery events discussed above, Theorem 2 is proved. ■

## 3 Conclusion

In this paper, we study a multi-buffer model for buffer management in quality-of-service network switches. We first show that the lower bound of competitive ratio of best-effort admission algorithms is 2. Then we propose a $(3+\sqrt{3} \approx 4.723)$-competitive deterministic algorithm, which is better than the previously best-known result 9.82 (Azar and Levy. SWAT 2006).

It is unknown that for the single-buffer model, whether a non-best-effort admission online algorithm can achieve an upper bound better than 2. It is an open problem to close or shrink the gap [1.618, 4.732] between the lower bound and the upper bound of competitive ratio for deterministic online algorithms for the multi-buffer model. We are also interested to know whether randomness can help improve competitive ratios in these problem settings.

## References

1. Aiello, W., Mansour, Y., Rajagopolan, S., Rosen, A.: Competitive queue policies for differentiated services. In: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 431–440 (2000)
2. Andelman, N., Mansour, Y., Zhu, A.: Competitive queuing polices for QoS switches. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 761–770 (2003)
3. Azar, Y., Levy, N.: Multiplexing packets with arbitrary deadlines in bounded buffers. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 5–16. Springer, Heidelberg (2006)
4. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
5. Chin, F.Y.L., Fung, S.P.Y.: Online scheduling with partial job values: Does timesharing or randomization help? Algorithmica 37(3), 149–164 (2003)
6. Chrobak, M., Jawor, W., Sgall, J., Tichy, T.: Online scheduling of equal-length jobs: Randomization and restart help? SIAM Journal on Computing (SICOMP) 36(6), 1709–1728 (2007)
7. Chrobak, M., Jawor, W., Sgall, J., Tichy, T.: Improved online algorithms for buffer management in QoS switches. ACM Transactions on Algorithms, 3(4), Article number 50 (2007)
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press and McGraw-Hill (2001)
9. Englert, M., Westermann, M.: Considering suppressed packets improves buffer management in QoS switches. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 209–218 (2007)
10. Hajek, B.: On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In: Proceedings of the 35th Annual Conference on Information Science and Systems (CISS), pp. 434–438 (2001)
11. Kesselman, A., Lotker, Z., Mansour, Y., Patt-Shamir, B., Schieber, B., Sviridenko, M.: Buffer overflow management in QoS switches. SIAM Journal of Computing (SICOMP) 33(3), 563–583 (2004)
12. Li, F.: Competitive scheduling of packets with hard deadlines in a finite capacity queue. In: Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM) (2009)
13. Li, F., Sethuraman, J., Stein, C.: An optimal online algorithm for packet scheduling with agreeable deadlines. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 801–802 (2005)
14. Li, F., Sethuraman, J., Stein, C.: Better online buffer management. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 199–208 (2007)

# Latency Constrained Aggregation in Chain Networks Admits a PTAS

Tim Nonner* and Alexander Souza

Department of Computer Science, Albert-Ludwigs-University of Freiburg,
Freiburg im Breisgau, Germany
tim.nonner@informatik.uni-freiburg.de

**Abstract.** This paper studies the aggregation of messages in networks that consist of a chain of nodes, and each message is time-constrained such that it needs to be aggregated during a given time interval, called its due interval. The objective is to minimize the maximum cost incurred at any node, which is for example a concern in wireless sensor networks, where it is crucial to distribute the energy consumption as equally as possible. First, we settle the complexity of this problem by proving its NP-hardness, even for the case of unit length due intervals. Second, we give a QPTAS, which we extend to a PTAS for the special case that the lengths of the due intervals are constants. This is in particular interesting, since we prove that this problem becomes APX-hard if we consider tree networks instead of chain networks, even for the case of unit length due intervals. Specifically, we show that it cannot be approximated within $4/3 - \epsilon$ for any $\epsilon > 0$, unless P=NP.

## 1 Introduction

The aggregation of distributed information to a central powerful node, called *sink*, is a basic task in many networks. This is for example a concern in wireless sensor networks (WSN) [1,13,18], whose nodes are small battery-powered devices which sense the environment. Whenever an event occurs at some node (e.g. seismic activity, temperature, wind-speed, etc.), then this event needs to reported to the sink. To save energy, messages are delivered to the sink via a precomputed tree topology. If a node runs out of energy, then the network becomes disconnected. Hence, to maximize the lifetime of the network, we need to minimize the maximum energy consumption of any node. If there are no time constraints, then the optimal strategy is that each node waits until it has received all messages from its successors in the tree, and then passes a single combined message, that aggregates all collected ones, to its predecessor. However, in most applications timing issues are a concern, and it is often required that each message is delivered to the sink before a specified individual deadline expires. Finding an optimal data aggregation schedule results in a combinatorial optimization

* Corresponding author. Supported by DFG research program No 1103 *Embedded Microsystems.*

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 279–291, 2009.

problem, called *Latency Constrained Data Aggregation Problem* (DA), treated before by Becchetti et al. [2]. In this paper, we mostly consider networks where the underlying tree structure is a chain, called *chain networks*. We refer to DA in this special case as DAC. Chain networks are useful when the WSN is installed along highways, runways, shopping miles etc. We will see in the following that this results in a natural generalization of the Interval Stabbing Problem that, given a set of horizontal intervals, asks to find a minimal set of vertical lines such that each interval intersects with at least one of them. We investigate an off-line perspective. However, all results in this paper can be also applied to the case where the measurements are periodically repeated over time.

**Formal problem definition.** An instance $A$ of DA consists of an intree $T = (V, E)$ rooted in a single sink that describes the network topology, a set of $n$ messages $M$, and a sending cost function $c : V \to \mathbb{R}^+$. Each message in $M$ is described by a tuple $(v, r, d)$, where $v \in V$ is the *release node*, $r \in \mathbb{R}^+$ is the *release time*, and $d \in \mathbb{R}^+$ is the *due time* (we do not distinguish between an instance and its message set, e.g., we also write $(v, r, d) \in A$ if $(v, r, d) \in M$). A schedule $S$ for such an instance assigns a sequence of sending times to each node. Let $S_v$ be the number of sending times of a node $v \in V$. Whenever $v$ sends, all messages currently stored at this node are passed to the node at the other end of the single outgoing edge in one combined message, which implies sending cost $c(v)$. We say that such a schedule is *feasible* if each message is *aggregated* to the sink before its due time, i.e., if for each message $(v, r, d) \in A$, the nodes on the simple path from $v$ to the sink have sending times such that $(v, r, d)$ is delivered to the sink during its *due interval* $[r, d]$ in a sequence of combined messages. As Becchetti et al. [2], we distinguish two different objective functions:

$$\begin{array}{ll} \text{(MinMax)} & \text{(MinSum)} \\ \max_v c(v)S_v & \sum_v c(v)S_v \end{array}$$

We denote the cost of a schedule $S$ with respect to both objectives by $\text{cost}(S)$, since the used objective is always clear from the context. Moreover, let $\text{OPT}(A)$ be the cost of an optimal schedule. This leads to two variants, DA and DA-Sum: Given an instance $A$, find a feasible schedule $S$ for $A$ that minimizes the MinMax- and MinSum-objective, respectively. For the special case of chain networks, we denote the corresponding problems by DAC and DAC-Sum, respectively. As noted in [2], the MinMax-objective is more reasonable in a distributed environment, where due to a decentralized energy-supply via batteries, we have to equally distribute the energy consumption. Becchetti et al. [2] introduced this model with *transit times*, i.e., they assume that it takes some time to pass a combined message over an edge. However, we do not consider transit times in this paper, since the following lemma says that it is possible to reduce the case with transit times to the case without transit times (proof in the full version):

**Lemma 1.** *For any of the considered problems* DA, DAC, DA-SUM, *and* DAC-SUM, *an approximation algorithm for the case without transit times yields an approximation algorithm for the case with transit times with the same approximation ratio.*

**Previous work.** Becchetti et al. [2] proved the NP-hardness of DA and DA-SUM. Moreover, they showed that both problems are in APX by presenting a 2-approximation algorithm. Recall that an $\alpha$-approximation algorithm returns, in polynomial time, a solution whose cost is at most $\alpha$ times larger than the cost of an optimal solution. In contrast, they showed that the chain case DAC-SUM is polynomially solvable via dynamic programming. The same dynamic programming approach was independently presented by Gijswijt et al. [6], whereas they interpreted this problem as a batch processing problem. However, Bechetti et al. [2] left the complexity of DAC open.

A closely related problem is the Multicast Acknowledgment Problem [3,11,15], where we also want to aggregate messages in a tree topology, but we do not have fixed deadlines. Instead, the objective is to minimize the sum of sending costs plus the sum of delays, which is closely related to the well-known flow time objective from scheduling. Papadimitriou et al. [16] used a similar model for the communication structure of a company, whereas they assumed that messages are queued according to the Poisson process. However, since our main motivation are distributed sensor networks, it is reasonable to minimize the maximum energy consumption of any node instead of the energy consumption of the whole network. Furthermore, Korteweg et al. [12] discussed the multiobjective problem of trading sending costs for delay. They presented several distributed algorithms that balance these objectives.

**Contributions and outline.** We settle the complexity of DAC by proving its NP-hardness in Section 5, even if all due intervals have unit length ($d - r = 1$, for each messages $(v, r, d) \in A$), which solves an open problem from [2]. Since Becchetti et al. [2] showed that DA and hence DAC are constant factor approximable, we are mostly interested in approximation schemes. Recall that an *approximation scheme* is a family of algorithms which contains a $(1 + \epsilon)$-approximation algorithm for any $\epsilon > 0$. First, we show in Section 3 that there is a quasipolynomial time approximation scheme (QPTAS), that is, an approximation scheme that runs in time $O(n^{\mathrm{polylog(n)}})$. This implies that DAC is not APX-hard, unless NP $\subseteq$ DTIME($n^{\mathrm{polylog}(n)}$). In the following Section 4, we investigate the special case where the due interval lengths are constants, i.e., these lengths are not part of the input, which includes the case of unit length due intervals (recall that DAC is NP-hard, even for unit length due intervals). This is reasonable especially from a practical point of view, since it is likely to have a finite set of priority levels that indicate how fast a message needs to be aggregated, and each level corresponds to a specific due interval length. For this special case, we extend the QPTAS to a polynomial time approximation scheme (PTAS) which is based on an interesting iterative LP-rounding strategy. Finally, in contrast to DAC, we prove in Section 5 that the tree case DA is APX-hard, even for unit length due intervals, which solves another open problem from [2].

Specifically, we give a quite strong inapproximability results by showing that DA cannot be approximated within $4/3-\epsilon$ for any $\epsilon > 0$, unless P = NP (recall that there is a 2-approximation algorithm).

## 2 Preliminaries

**Restricting the search space.** In chain networks, we can label the non-sink nodes with $1, 2, \ldots, k$, where node $1 \leq v \leq k$ has hop-distance $k - v + 1$ to the sink (since the sink does not send any messages, we can omit it in the chain case for the sake of simplicity). The following simple observation allows us to restrict the search space:

**Observation 1 ([2]).** *For any instance, there is an optimal feasible schedule such that (1) all messages only wait at their respective release nodes, and (2) each sending time is the due time of a message.*

Hence, we only consider schedules with the properties listed in Observation 1. Using this, we can interpret a schedule $S$ as a set of *lines* such that each line is defined by a tuple $(v, t)$, where $1 \leq v \leq k$ is the *starting node* and $t \in \mathbb{R}^+$ is the *time* of this line, which is the due time of some message. Specifically, a node $u$ sends at time $t$ if and only if there is a line $(v, t) \in S$ with $v \leq u$, and hence $S_u = |\{(v, t) \in S \mid v \leq u\}|$. Thus, we say that a message $(u, r, d)$ is *covered* by a line $(v, t)$ if and only if $v \leq u$ and $r \leq t \leq d$, and a message is hence aggregated if and only if it is covered by at least one line. If there is more than one such line, then assume that the message is aggregated by the first one.

**Subinstances.** Given some instance $A$, observe that each open interval $(a, b) \subseteq \mathbb{R}^+ \cup \{\infty\}$ defines a *subinstance* $A(a, b)$ of $A$ that only contains the messages whose due intervals are contained in the open interval $(a, b)$, that is $\{(v, r, d) \in A \mid a < r \leq d < b\}$. Moreover, $A(a, b)$ inherits the sending cost function from $A$. Naturally, we say that two subinstances $A_1, A_2$ of $A$ are *disjoint* if $A_1 \cap A_2 = \emptyset$.

**Relations to interval stabbing.** Consider the instance $A$ depicted in Fig. 1 with $k = 4$. The four vertical levels in this figure correspond to the four non-sink nodes in $A$. Consequently, we depict the due intervals of the messages in $A$ at their respective levels. Specifically, nodes 1 and 2 both release two messages, and so on. As explained above, we have to cover these messages with vertical lines. The vertical dashed lines represent such a schedule $S$. Specifically, there are two lines in $S$ starting at node 1, and so on. Since each message is covered by some line, $S$ is clearly feasible.

Observe that if $k = 1$, then we obtain the *Interval Stabbing Problem* (IS), that is, we simply have to cover intervals with a minimal number of vertical lines. We conclude that we can interpret DAC and DAC-Sum as a generalization of IS. Therefore, except for finding application in the area of data aggregation, we think that these problems are of independent interest. IS is equivalent to Clique Cover in interval graphs, and can be solved optimally via a simple greedy

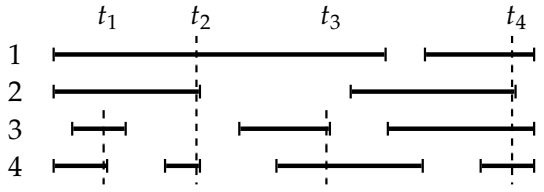



**Fig. 1.** An example instance $A$ with $k = 4$ and a feasible schedule $S$ which contains the lines $(1, t_2), (1, t_4), (3, t_1)$, and $(3, t_3)$

procedure in linear time [8]. Due to its fundamental nature, several extensions have been considered in the past. For instance, Even et al. [4] showed that even the capacitated version can be solved in polynomial time, that is, each possible line has a capacity contraint that indicates how many intervals it is allowed to cover. However, IS becomes NP-hard for two dimensions [9], i.e., if we want to cover rectangles with vertical and horizontal lines, but there is a 2-approximation algorithm [5] for this extension.

**Monotonicity of sending times.** Since our interpretation of a schedule as a set of vertical lines implies that the number of sending times can only increase, we obtain that $S_1 \leq S_2 \leq \ldots \leq S_k$. We also refer to the number of sending times $S_k$ of the last non-sink node $k$ as the *number of sending times* of $S$. This raises the question whether we can strengthen this monotonicity property. To this end, we need the following definition:

**Definition 1 (normal form).** Let $S$ be a feasible schedule. We say that a line $(v, t) \in S$ can be *pushed* if replacing this line with the line $(v + 1, t)$ does not affect the feasibility of $S$ (if $v = k$, then we remove $(v, t)$ instead of replacing it), i.e., there is no message that becomes uncovered. Using this, we say that a feasible schedule $S$ is in *normal form* if (1) no line can be pushed and (2) there is no pair of lines $(v, t), (u, s) \in S$ such that we can *replace* these lines with the lines $(u, t), (v, s)$ without affecting the feasibility of $S$, and then additionally push $(v, s)$.

In words, a feasible schedule given in normal form cannot be trivially improved by the simple local improvements of pushing, respectively by replacing and pushing. Note that the schedule in Fig. 1 is not given in normal form, since we can replace the first two lines $(3, t_1), (1, t_2)$ with the lines $(1, t_1), (3, t_2)$, and then additionally push $(3, t_2)$ without affecting the feasibility of $S$.

**Lemma 2.** *Any feasible schedule can be transformed into normal form in polynomial time.*

*Proof.* Observe that we can interpret the two normal form properties in Definition 1 as operations on a feasible schedule. Iteratively applying these operations as often as possible clearly yields a feasible schedule in normal form. Each time such an operation is successful, $S_v$ decreases by one for some node $v$, which gives

the polynomial running time, since initially $S_v \leq n$. On the other hand, note that $S_v$ does not increase for any node $v$ during this process. □

By Lemma 2, we can wlog assume in the following that every feasible schedule is given in normal form. Using this, the following definition gives a stronger monotonicity property:

**Definition 2 (exponential growth).** For two constants $\eta, \gamma > 1$, we say that an instance $A$ has $(\eta, \gamma)$-*exponential growth* if for any feasible schedule $S$ for $A$ (given in normal form) and any pair of nodes $v < u < k$, $S_u > \eta^{u-v}/\gamma \cdot S_v$.

In words, starting with any node $v$, the number of sending times grows exponentially with respect to $\eta$ and $\gamma$, whereas we have to exclude the single node $k$ for technical reasons. We need the assumption that any feasible schedule is given in normal form, since otherwise, we cannot find instances with exponential growth. To see this, observe that there is always the feasible schedule $\{(1, d) \mid (v, r, d) \in A\}$, which clearly does not fulfill the exponential growth property for $v = 1$. Using Definition 2, the following technical reduction lemma says that we can wlog assume that any given instance has exponential growth. We will use this lemma multiple times in the following (proof in full version):

**Lemma 3.** *If for any pair of constants $\eta, \gamma$, there is a (Q)PTAS for* DAC *restricted to instances with $(\eta, \gamma)$-exponential growth, then there is a (Q)PTAS for* DAC.

## 3 Dynamic Programming and a QPTAS

In this section, we present a dynamic programming approach which can be applied to DAC-Sum and DAC, whereas it runs only in polynomial time for DAC-Sum. Let $d_1 < d_2 < \ldots < d_n$ be an ordering of the due times in $A$, and let $d_0 := 0$ and $d_{n+1} := \infty$ (assume wlog that the due times are distinct). For any pair $0 \leq s < t \leq n + 1$, if $A(d_s, d_t) \neq \emptyset$, then let $v(s, t) := \min\{v \mid \exists (v, r, d) \in A(d_s, d_t)\}$ be the least numbered node that releases a message in $A(d_s, d_t)$ (recall that $A(d_s, d_t)$ is a subinstance of $A$). In this case, we choose an arbitrary message $(v(s, t), r, d) \in A(d_s, d_t)$. Clearly, in any optimal feasible schedule $S$ for $A(d_s, d_t)$, there must be a line $(v(s, t), d_i)$ with $s < i < t$ which covers this message. Observe that this line decomposes the instance $A(d_s, d_t)$ in two subinstances $A(d_s, d_i), A(d_i, d_t)$ such that there are two feasible schedules $S_1$ and $S_2$ for $A(d_s, d_i)$ and $A(d_i, d_t)$, respectively, with

$$S = S_1 \cup S_2 \cup \{(v(s, t), d_i)\}.$$

This immediately yields a polynomial time dynamic programming approach for DAC-Sum as follows: For each pair $0 \leq s < t \leq n + 1$, it suffices to store $\mathrm{OPT}(A(d_s, d_t))$ in a dynamic programming array, since we can inductively fill this array by using the following recurrence:

$$\mathrm{OPT}(A(d_s, d_t)) = \min_{s<i<t} (\mathrm{OPT}(A(d_s, d_i)) + \mathrm{OPT}(A(d_i, d_t)) + \sum_{v=v(s,t)}^{k} c(v),$$

where $k$ is the number of non-sink nodes as explained in Subsection 2. This approach was used in [2] and [6] to construct a polynomial time algorithm for DAC-Sum.

To adapt this approach to DAC, observe that we can think of DAC as a multi-objective variant of DAC-Sum, where the dimensions are the costs of the nodes. For an introduction to the notions of multi-objective optimization, we refer to [17]. We use the following two definitions from this area:

**Definition 3 (dominate).** We say that a schedule $S$ is *$\alpha$-dominated* by another schedule $S'$ if $S'_v \leq \alpha S_v$, for each node $v$. If $\alpha = 1$, then we simply say that $S'$ *dominates* $S$.

**Definition 4 (Pareto front).** We call a set of feasible schedules $D$ for an instance $A$ a *Pareto front* if for each feasible schedule $S$ for $A$, there is a schedule $S' \in D$ that dominates $S$. Moreover, we say that $D$ is *$\mu$-restricted* if this property only holds for each feasible schedule $S$ for $A$ with at most $\mu$ sending times.

Note that our definition of a Pareto front slightly differs from the usual definition of a Pareto front, since we do not require that there is no pair of schedules $S, S' \in D$ such that $S$ dominates $S'$.

Clearly, having a Pareto front $D$, we can find an optimal feasible schedule by computing $\text{cost}(S) = \max_v c(v)S_v$, for every schedule $S \in D$. But since the number of nodes is part of the input, we obtain a multi-objective optimization problem with an unbounded number of dimensions. Therefore, a Pareto front might have superpolynomial size. However, we can use the following lemma:

**Lemma 4.** *The dynamic programming approach for* DAC-Sum *can be adapted to* DAC *such that given some positive integer $\mu$, it computes a $\mu$-restricted Pareto front of size at most $(\mu+1)^k$ in time polynomial in $n$ and $\mu^k$.*

*Proof.* We use a larger dynamic programming array $\Pi$, which contains one entry $\Pi(s,t,f)$ for each pair $0 \leq s < t \leq n+1$ and each function $f : \{1,\dots,k\} \to \{0,1,\dots,\mu\}$ that indicates if there exists a feasible schedule $S$ for the subinstance $A(d_s, d_t)$ with $S_v = f(v)$, for each node $v$. The following recurrence allows us to inductively fill this array:

$$\Pi(s,t,f) = \text{true} \iff \exists i, f_1, f_2 : (\Pi(s,i,f_1) = \text{true}) \wedge (\Pi(i,t,f_2) = \text{true}) \wedge (s < i < t) \wedge f(v) = \begin{cases} 0 & 1 \leq v < v(s,t) \\ f_1(v) + f_2(v) + 1 & v(s,t) \leq v \leq k \end{cases}$$

We initialize $\Pi$ by setting $\Pi(s,t,f)$ to false if $A(d_s,d_t)$ does not contain a message. Finally, the set of schedules $D$ that realizes the entries $\Pi(0,n+1,f)$, $f \in \{0,1,\dots,\mu\}^{\{1,\dots,k\}}$, is returned. The claim clearly follows. □

**Theorem 1.** *There is a QPTAS for* DAC.

*Proof.* Consider a fixed pair $\eta, \gamma > 1$, and let $A$ be an instance with $(\eta, \gamma)$-exponential growth. Since any schedule $S$ for $A$ has at most $n$ sending times, i.e., $S_k \leq n$, we immediately conclude with the exponential growth property of $A$ that the number of nodes is logarithmically bounded in the number of messages, i.e., $k = O(\log(n))$. In this case, for the parameter choice $\mu = n$, the dynamic programming approach from Lemma 4 has quasipolynomial running time $O(n^{O(log(n))})$, and the returned Pareto front $D$ has quasipolynomial size. Note that $\mu = n$ implies that $D$ contains an optimal feasible schedule, which we can hence find in quasipolynomial time. The claim then follows from Lemma 3. □

## 4 A PTAS for Constant Length Due Intervals

In this section, we present a PTAS for DAC for the case that the due interval lengths are constants. For simplicity, we assume throughout this section that we have unit length due intervals. However, all arguments can be easily extended to due intervals with arbitrary but constant length.

Starting with the seminal work of Hochbaum and Maass [10], it has become a common approach in the design of approximation schemes for geometric optimization problems to exploit the geometric structure in order to decompose an instance in smaller subinstances. Such an approach has also been already used in the context of multiobjective optimization to obtain a PTAS for Multiobjective Disk Cover [7]. We apply a similar approach. Specifically, we decompose an instance in subinstances that have smaller Pareta fronts. However, there is one significant difference to the approach in [7]: They have a finite number of dimensions, and consequently, their Pareto front has polynomial size. Therefore, it is possible to combine the Pareto fronts of the subinstances to a Pareto front of the original instance in polynomial time. This does not hold in our case. Hence, we need to compute a schedule from the Pareto fronts of the subinstances without computing the Pareto front of the original instance. To this end, we use an LP-rounding approach. This rounding approach iteratively solves a linear program such that after each iteration we obtain some new integral variables. We think that this approach is likely to find application in other multi-objective scenarios with an unbounded number of dimensions. The decomposition used in this section is defined as follows:

**Definition 5 ($\epsilon$-conserving decomposition).** Given some $\epsilon > 0$ and an instance $A$, we call a sequence of pairwise disjoint subinstances $A_1, A_2, \ldots, A_n$ of $A$ an *$\epsilon$-conserving decomposition* if there is some (possibly non-feasible) schedule $S^0$ for $A$ with $\text{cost}(S^0) \leq \epsilon \text{OPT}(A)$, and for any sequence of feasible schedules $S^1, S^2, \ldots, S^n$ for the subinstances $A^1, A^2, \ldots, A^n$, respectively, the union $\cup_{i=0}^{n} S^i$ is a feasible schedule for $A$.

The following lemma is proven in the full version of this paper:

**Lemma 5.** *For any sufficiently small $\epsilon > 0$ and any instance $A$ with unit length due intervals, there is an $\epsilon$-conserving decomposition $A_1, A_2, \ldots, A_n$ such that any feasible schedule for one of these subinstances has at most $\kappa_\epsilon$ sending times for some constant $\kappa_\epsilon$.*

**Extending a combination of schedules.** Assume now that given an instance $A$ and a sufficiently small $\epsilon > 0$, we have an $\epsilon$-conserving decomposition $A_1, A_2, \ldots, A_n$ with an associated schedule $S^0$ as described in Lemma 5. It follows from this lemma that in order to find an arbitrary good approximation of an optimal feasible schedule for $A$, it suffices to select an optimal combination of feasible schedules $S^1, S^2, \ldots, S^n$ for the subinstances $A_1, A_2, \ldots, A_n$, respectively. We exploit this in the following. Assume that we have a Pareto front $D_i$ for each subinstance $A_i$. Moreover, assume that we have already selected some feasible schedules $S^i$, $i \in \overline{Q}$, for a subset $Q \subseteq \{1, \ldots, n\}$, where we define $\overline{Q} := \{1, \ldots, n\} \backslash Q$. Clearly, finding some optimal feasible schedules $S^1, S^2, \ldots, S^n$ with respect to the already selected schedules corresponds then to finding some feasible schedules $S^i$, $i \in Q$, that minimize the following objective:

$$\max_v c(v) \left( \sum_{i \in Q} S^i_v + \sum_{i \in \overline{Q}} S^i_v \right).$$

We can formulate this problem as an integer program by introducing an integral variable $x_{i,S}$ for each $i \in Q$ and each schedule $S \in D_i$ that indicates whether this schedule is part of this selection. We need a more general integer program which allows us to restrict the nodes to a subset $C \subseteq \{1, \ldots, k\}$:

$$\begin{aligned}
&\text{minimize} && z && && \\
&\text{subject to} && z \geq c(v) \left( \sum_{i \in Q, S \in D_i} x_{i,S} S_v + \sum_{i \in \overline{Q}} S^i_v \right) && \text{for } v \in C && (1) \\
& && \sum_{S \in D_i} x_{i,S} \geq 1 && \text{for } i \in Q && (2) \\
& && x_{i,S} \in \{0, 1\} && \text{for } i \in Q, S \in D_i && (3)
\end{aligned}$$

Constraints (2) ensure that exactly one schedule is picked from each Pareto front $D_i$, and constraints (1) enforce that the objective is minimized. Observe that the number of variables in this integer program depends on the sizes of the sets $D_i$, $i \in Q$. We refer to the corresponding linear program in which we relax the integrality constraints (3) by $x_{i,S} \geq 0$ as $LP(Q, C)$. This linear program is used multiple times in the following procedure EXTEND which *extends* a given combination of feasible schedules $S^i$, $i \in \overline{Q}$, to some feasible schedules $S^1, S^2, \ldots, S^n$:

**EXTEND**$(Q)$

1. Set $v^* \leftarrow \min\{v \mid \exists i \in Q, S \in D_i : S_v > 0\}$.
2. **(inner loop)** While $Q \neq \emptyset$:
   (a) If $|Q| = 1$, then, for the single element $i \in Q$, set $S^i$ to an arbitrary schedule in $D_i$, and terminate this procedure.
   (b) Set $C \leftarrow \{v^*, \dots, \min\{k, v^* + |Q| - 2\}\}$.
   (c) Compute an optimum $x$ of the linear program $LP(Q, C)$.
   (d) Set $Q' \leftarrow \{i \in Q \mid \exists S \in D_i : x_{i,S} \notin \{0, 1\}\}$.
   (e) For each $i \in Q \backslash Q'$, set $S^i$ to the schedule in $D_i$ indicated by the corresponding integral variable in $x$.
   (f) Set $Q \leftarrow Q'$.

The node $v^*$ in step 1 is simply the smallest labeled node such that there exists a set $D_i$ with a schedule $S \in D_i$ that has at least one sending time at $v^*$. Moreover, we set $Q'$ in each step 2(d) to the 'non-integral part' of $Q$ with respect to $x$.

**Lemma 6.** *The inner loop has at most n iterations, and hence procedure EXTEND terminates in linear time.*

*Proof.* By the setting of $C$ in step 2(b), we have in each iteration that $|C| < |Q|$. We have to show that this yields that the optimum $x$ of the linear program $LP(Q, C)$ has an integral part, since then the cardinality of $Q$ decreases by at least one in each iteration, which yields the claim. To this end, we follow the arguments from Theorem 1 in [14]. Let $z$ be the cost of $x$. Moreover, let $w := \sum_{i \in Q} |D_i|$, $u := |Q|$ and $l := |C|$. Hence, $x$ is a point on the polyhedron in $\mathbb{R}^w$ defined by the following $l + u + w$ constraints:

$$z/c(v) - \sum_{i \in \overline{Q}} S_v^i \geq \sum_{i \in Q, S \in D_i} x_{i,S} S_v \quad \text{for } v \in C$$

$$\sum_{S \in D_i} x_{i,S} \geq 1 \quad \text{for } i \in Q \tag{4}$$

$$x_{i,S} \geq 0 \quad \text{for } i \in Q, S \in D_i \tag{5}$$

We can wlog assume that $x$ is a vertex on this polyhedron. In this case, since this polyhedron is $w$-dimensional, we have that $x$ satisfies at least $w$ of these constraints with equality. Consequently, at least $w - u - l$ of constraints (5) are satisfied with equality by $x$. Therefore, at least $w - u - l$ of the variables in $x$ are 0. Define $h_i := |\{S \in D_i \mid x_{i,S} \neq 0\}|$, for each $i \in Q$. By the arguments above, $\sum_{i \in Q} h_i \leq u + l$. Hence, since $l = |C| < |Q| = u$,

$$\frac{1}{|Q|} \sum_{i \in Q} h_i < 2. \tag{6}$$

This term is the average size of the integers $h_i$, $i \in Q$. But on the other hand, constraints (4) ensure that $h_i \geq 1$, for each $i \in Q$. Hence, by inequality (6), we obtain that there must be at least one $i \in Q$ with $h_i = 1$, and therefore, there is at least one $S \in D_i$ with $x_{i,S} = 1$. This proves the claim of the lemma. □

**The algorithm.** Using procedure EXTEND, we can now formulate the final algorithm, which takes an instance $A$, a sufficiently small $\epsilon > 0$, and a positive integer parameter $\beta$ as input. The parameter $\beta$ affects the running time of the algorithm, and we will show in the following that we can choose a constant $\beta$ such that the algorithm yields a PTAS:

**ALGORITHM**$(A, \epsilon, \beta)$

1. Compute an $\epsilon$-conserving decomposition $A_1, A_2, \ldots, A_n$ of $A$ according to Lemma 5 with an associated schedule $S^0$ for $A$.
2. For $i = 1, \ldots, n$: Compute a Pareto front $D_i$ for $A_i$ according to Lemma 4 with $\mu = \kappa_\epsilon$.
3. **(outer loop)** For each set $Q \subseteq \{1, \ldots, n\}$ of size $n - \beta$ and each combination of schedules $S^i \in D_i$, $i \in \overline{Q}$:
   (a) Use procedure EXTEND to extend the combination of schedules $S^i$, $i \in \overline{Q}$, to some schedules $S^1, S^2, \ldots, S^n$.
   (b) Set $S \leftarrow \cup_{i=0}^{n} S^i$.
4. Return the best schedule $S$ computed in the outer loop.

**Lemma 7.** *If the input instance $A$ has $(\eta, \gamma)$-exponential growth for some pair $\eta, \gamma > 1$, then the algorithm has polynomial running time.*

*Proof.* If $A$ has exponential growth, then we have that $k = O(\log(n))$, since the number of sending times $S_k$ is at most $n$. Therefore, by the parameter choice $\mu = \kappa_\epsilon$, we conclude with Lemma 4 that the dynamic programming approach yields the Pareto fronts $D_1, D_2, \ldots, D_n$ in polynomial time $O(\kappa_\epsilon^{O(\log(n))})$, and these sets have also polynomial size. In each iteration of the inner loop, we have to solve a linear program, where the size of this linear program is $O(\sum_{i=1}^{n} |D_i|)$, and hence polynomial. Consequently, we can solve this linear program with the Ellipsoid method in polynomial time. Now we count the number of times we have to solve such a linear program. First, again since the Pareto fronts $D_1, D_2, \ldots, D_n$ have polynomial size and $\beta$ is constant, we see that the outer loop has polynomially many iterations. Second, Lemma 6 shows that each inner loop has at most $n$ iterations. The claim follows by combining all these facts. □

The following lemma is proven in the full version of this paper:

**Lemma 8.** *If the input instance $A$ has $(\eta, \gamma)$-exponential growth for some pair $\eta, \gamma > 1$, then, for any $\epsilon > 0$, we can choose a constant $\beta$ such that the algorithm returns a feasible schedule $S$ with cost$(S) \leq (1 + 2\epsilon)OPT(A)$.*

**Theorem 2.** *There is a PTAS for* DAC *with unit length due intervals.*

*Proof.* Combine Lemmas 3, 7, and 8. □

## 5 Hardness Results

The following theorems are proven in the full version of this paper:

**Theorem 3.** DAC *is strongly NP-hard, even for unit length due intervals.*

**Theorem 4.** *For any* $\epsilon > 0$, DA *cannot be approximated within* $4/3 - \epsilon$ *in polynomial time, even for unit length due intervals, unless P=NP.*

## References

1. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Comput. Netw. 38(4), 393–422 (2002)
2. Becchetti, L., Korteweg, P., Marchetti-Spaccamela, A., Skutella, M., Stougie, L., Vitaletti, A.: Latency constrained aggregation in sensor networks. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 88–99. Springer, Heidelberg (2006)
3. Brito, C., Koutsoupias, E., Vaya, S.: Competitive analysis of organization networks or multicast acknowledgement: how much to wait? In: Proc. of the 15th annual ACM-SIAM symposium on Discrete algorithms (SODA), pp. 627–635 (2004)
4. Even, G., Levi, R., Rawitz, D., Schieber, B., Shahar, S., Sviridenko, M.: Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. ACM Trans. Alg. 4(3) (2008)
5. Gaur, D.R., Ibaraki, T., Krishnamurti, R.: Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. J. Algorithms 43(1), 138–152 (2002)
6. Gijswijt, D., Jost, V., Queyranne, M.: Clique partitioning of interval graphs with submodular costs on the cliques. RAIRO - Operations Research 41(3), 275–287 (2007)
7. Glasser, C., Schmitz, H., Reitwiessner, C.: Multiobjective disk cover admits a PTAS. In: Proc. of the 19th International Symposium on Algorithms and Computation (ISSAC) (2008)
8. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Discrete Mathematics, vol. 57. North-Holland Publishing Co, Amsterdam (2004)
9. Hassin, R., Megiddo, N.: Approximation algorithms for hitting objects with straight lines. Discrete Applied Mathematics 30(1), 29–42 (1991)
10. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM 32(1), 130–136 (1985)
11. Khanna, S., Naor, J(S.), Raz, D.: Control message aggregation in group communication protocols. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 135–146. Springer, Heidelberg (2002)
12. Korteweg, P., Marchetti-Spaccamela, A., Stougie, L., Vitaletti, A.: Data aggregation in sensor networks: Balancing communication and delay costs. In: Prencipe, G., Zaks, S. (eds.) SIROCCO 2007. LNCS, vol. 4474, pp. 139–150. Springer, Heidelberg (2007)

13. Krishnamachari, B., Estrin, D., Wicker, S.B.: The impact of data aggregation in wireless sensor networks. In: Proc. of the 29th International Conference on Distributed Computing Systems (ICDCS), pp. 575–578 (2002)
14. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Math. Program. 46(3), 259–271 (1990)
15. Oswald, Y.A., Schmid, S., Wattenhofer, R.: Tight bounds for delay-sensitive aggregation. In: Proc. of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 195–202 (2008)
16. Papadimitriou, C.H.: Computational aspects of organization theory (extended abstract). In: Díaz, J. (ed.) ESA 1996. LNCS, vol. 1136, pp. 559–564. Springer, Heidelberg (1996)
17. Papadimitriou, C.H., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proc. of the 41st Annual Symposium on Foundations of Computer Science (FOCS), pp. 86–92 (2000)
18. Yu, Y., Krishnamachari, B., Prasanna, V.K.: Energy-latency tradeoffs for data gathering in wireless sensor networks. In: Proc. of the 23rd Conference of the IEEE Communications Society (INFOCOM) (2004)

# Cutting a Cake for Five People

Amin Saberi and Ying Wang

Stanford University, Stanford CA 94305, USA
{saberi,yw1984}@stanford.edu

**Abstract.** Given a heterogeneous cake and 5 players, we give the first bounded algorithm for computing an envy-free division of the cake, such that each person thinks he gets the largest piece. The case with 4 players was solved in a famous paper by Brams et al. in 1997. Our algorithm can be discretized to obtain an $\epsilon$ envy-free division in $O(\text{polylog}\,(1/\epsilon))$ time. The algorithm is based on augmenting the irrevocable advantage graph in a new way.

We also look at the open problem of finding discrete procedures for computing envy-free division among 4 players. We present a simple algorithm that finds an envy-free division of a portion of the cake, such that each player gets at least 1/4 of the whole cake (in his valuation).

## 1 Introduction

Since the beautiful works of Steinhaus and Banach and Knaster [1,2], cake cutting problems have been a favorite among mathematicians. They have found various applications in the fields of economics, political science, operations research, and computer science.

The main objective in the cake cutting problem is fairness. The two most commonly used notions of fairness are "proportionality" and "envy-freeness". In proportional cake cutting, each person gets a piece which he considers as at least $\frac{1}{n}$ of the whole cake. "Envy-free" is a stronger condition, saying that each person must get the piece he thinks is largest (or at least tied with the largest).

Despite the simplicity of the model, the envy-free cake cutting problem has been eluding researchers for many decades. In 1976, Selfridge and Conway gave a simple discrete procedure for computing an envy-free allocation for 3 players. Later in 1980, the existence of envy-free divisions with $n-1$ cuts was proved by Stromquist [3]. Not much progress was made until 1995, when Brams and Taylor [4] gave the first finite discrete algorithm for envy-free division for general $n$. However, the number of cuts needed in their algorithm depends on the actual utility function of the players, and it may be unbounded. Finding a bounded algorithm for general $n$ is known to be one of the most important open problems in the field. Even for $n = 4$ no such algorithm is known.

In 1997, Brams et al. gave a moving-knife procedure for an envy-free division for 4 people [5]. In the paper they pointed out that a solution for $n = 5$ exists if a certain extension of Austin's procedure [6] exists. The existence of that extension seems to be unlikely.

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 292–300, 2009.

In this paper we solve the case for $n = 5$ using a different approach. In particular, we give the first moving-knife procedure for $n = 5$. Our algorithm is based on the idea of *irrevocable advantage (IA)* used in [4]. Assume the cake has been allocated partially, we say $A$ has IA over $B$ if $A$ will not envy $B$ even if $B$ is given the rest of the cake. Our algorithm first computes an envy-free division over a portion of the cake in a way such that some IA relationships can be obtained when dividing the leftover. We then recurse on the leftover until a certain structure exists in the graph of IA relationships. At that point we can divide the remaining part directly, without any trimming. The idea is similar to the Selfridge-Conway procedure. However, new ideas are used for obtaining a partial allocation and augmenting the IA graph, and these ideas may be useful for bigger $n$.

A desirable property of our procedure is that it is "almost discrete", in the sense that all operations used are discrete except for the Austin's procedure. We show how the Austin's procedure can be discretized, yielding an $\epsilon$-approximate envy-free division in $O(\text{polylog}\left(\frac{1}{\epsilon}\right))$ time, using only a constant number of cuts.

Based on the same idea, we also give a discrete procedure for $n = 4$. The procedure produces an envy-free division such that each person gets at least $\frac{1}{4}$ of the cake (in his own valuation), but it does not always allocate the whole cake.

### 1.1 Related Work

A more popular line of research in theoretical computer science is computing proportional (or approximately proportional) divisions. In [7], Even and Paz gave a deterministic divide-and-conqueror algorithm that computes a proportional division, and it uses only $O(n \log n)$ queries. In the last few years, there has been several papers on the lower bound of query complexity of computing proportional division. Sgall and Woeginger [8] showed that every proportional protocol (either deterministic or randomized) has query complexity $\Omega(n \log n)$, under the constraint that each player must receive a consecutive piece. Edmonds and Pruhs [9] showed the same lower bound for any deterministic approximate-fair protocol, and later they gave randomized algorithms for computing approximate proportional divisions, using a linear number of cuts with high probability [10,11].

For envy-free divisions, Su [12] gave a constructive proof of the existence of an envy-free division with $n-1$ cuts. The proof can be turned into an approximate algorithm. However, the running time of the algorithm is exponential. Deng et al. [13] use that proof to give a $O(\text{polylog}\left(\frac{1}{\epsilon}\right))$ algorithm for the case $n = 3$. No such algorithm is known for bigger $n$. The major difference between their work and this paper is that in this paper we may use more than $n-1$ cuts.

## 2 Notations and Assumptions

The cake cutting problem involves a cake, denoted by the $[0, 1]$ interval, and $n$ players $P_1, P_2, \ldots, P_n$. Each player has a utility function $u_i : 2^{[0,1]} \to R^+$

which is a measure over subsets of $[0,1]$, i.e., $u_i(\emptyset) = 0$, $u_i([0,1]) = 1$, and $u(\cup_{i=1}^{\infty} S_i) = \sum_{i=1}^{\infty} u(S_i)$ if $S_i$'s are disjoint.

A division of the cake is a partition of the interval $[0,1] = C_1 \cup C_2 \cdots \cup C_n$, where $C_i$ is assigned to the $i$th player. A division is *fair* (or *proportional*) if $u_i(C_i) \geq \frac{1}{n}$ for all $i$. It is *envy-free* if $u_i(C_i) \geq u_i(C_j)$ for all $i$ and $j$, and $\epsilon$ *envy-free* if $u_i(C_i) \geq u_i(C_j) - \epsilon$. In this paper, we consider only the case where each $C_i$ is the union of finitely many intervals.

In a *discrete* procedure, only two types or queries are allowed.

1. Value query $Q_1(i, S)$: The query takes a player $i$ and a set $S$, and it returns $u_i(S)$.
2. Cut query $Q_2(i, [a,b], p)$: The query takes a player $i$, an interval $[a,b]$, and a real number $0 < p < 1$, and it outputs the minimum number $c$ such that $u_i([a,c]) = p \times u_i([a,b])$.

The value query is natural, and the cut query allows a player to divide an interval into any integral number of equal pieces.

When considering approximation algorithms for the problem we only consider value queries, and we assume utility functions are upper bounded by $M$, i.e., $u_i(I) \leq M|I|$ for all players $i$ and intervals $I$. This assumption is necessary because otherwise the utility can concentrate on a very small interval and it may take the algorithm arbitrarily long to find that interval using only value queries.

## 3 5-Person Envy-Free Division

First we define the concept of irrevocable advantage formally:

**Definition 1.** *Let $S$ be the part of cake that has been allocated and assume the piece assigned to player $i$ is $C_i$. Let $T$ be part of the leftover, we say player $i$ has IA over player $j$ when dividing $T$ if $u_i(C_i) \geq u_i(C_j \cup T)$. In other words, $i$ does not envy $j$ even if $j$ is given the whole piece $T$.*

Before we state the algorithm, we need to introduce the Austin's procedure:

**Lemma 1.** *Assume there are 2 players and the utility functions $u_i([0,x])$ are continuous in $x$, then there is a moving-knife procedure that cuts the cake into $k$ pieces, such that both players think all pieces have equal values.*

*Proof.* Suppose $k = 2$. Denote the cake by the interval $[0,1]$. Initially put one knife at 0, and another knife at $x$ such that $u_1([0,x]) = 1/2$. Now move both knives towards right such that the value of the cake in between remains $1/2$ for $P_1$. The claim is that at some point the middle part also values $1/2$ for $P_2$.

Suppose $u_2([0,x]) \leq 1/2$. When the left knife moves to $x$, the right knife is at 1. Therefore for $P_2$, initially the middle part values $u_2([0,x]) \leq 1/2$ and at the end it values $u_2([x,1]) = 1 - u_2([0,x]) \geq 1/2$. By continuity, there must be a moment where the middle part values $1/2$ for $P_2$ as well. The above procedure can be easily generalized to the case $k > 2$. □

Using Austin's procedure as a black box, we are now ready to state the main procedure.

**Lemma 2.** *Given a cake and* 5 *players, there exists a procedure that produces an envy-free division for part of the cake, using a constant number of cuts. Furthermore, the leftover can be divided into two parts* $L_1$ *and* $L_2$*, such that when dividing* $L_1$*, both* $P_1$ *and* $P_2$ *have IA over some player in* $P_3, P_4, P_5$*, and when dividing* $L_2$*,* $P_1$ *and* $P_2$ *have IA over another player.*

*Proof.* First, use Austin's procedure on $P_1$ and $P_2$ to produce 5 equal pieces for both players. Depending on the favorite pieces of the rest of the players, there are 3 cases.

- The favorite pieces of $P_3, P_4, P_5$ are all different.
  In this case we can let $P_3, P_4$ and $P_5$ take their favorite pieces and let $P_1, P_2$ take whatever is left. Since $P_1$ and $P_2$ are indifferent about the five pieces, we get an envy-free division.
- All of $P_3, P_4, P_5$ like the same piece.
  Without loss of generality assume all of $P_3, P_4, P_5$ prefer the first piece. Suppose we shrink the first piece gradually, then at some point one of $P_3, P_4, P_5$ will think the first piece is as large as another piece. Again without loss of generality, assume $P_3$ is now tied between the first piece and the second.
  Now we shrink the first piece and the second piece simultaneously such that $P_3$ is always indifferent between them. Two cases can happen here:

  (i) At some point, one of $P_4$ and $P_5$ thinks his favorite piece (either first or second) is tied with one of the rest (either the third, the fourth, or the fifth). Without loss of generality assume $P_4$'s favorite pieces become the first and the third. Now we have obtained an envy-free division: Let $P_5$ takes his favorite piece, and $P_4$ takes the third piece, and $P_3$ takes whatever is left in the first two pieces, and $P_1$ and $P_2$ take the fourth and the fifth pieces.

  (ii) $P_3$ is tied with another piece, say the third piece. In this case, if $P_4$ and $P_5$ prefer different pieces then we are done; otherwise, suppose they both like the first piece. Shrink the first piece until one of them is tied with another piece. It is easy to check that no matter which piece it is, we get a perfect matching (and hence an envy-free division).

- Two of $P_3, P_4, P_5$ like the same piece.
  Assume $P_3$ and $P_4$ prefer the first piece and $P_5$ prefers the second. Trim the first piece until one of $P_3$ and $P_4$ is tied between the first piece and another piece. If that piece is not the second, then we have a perfect matching; If it is the second piece, the problem reduces to the previous case, where we shrink the first piece and the second piece simultaneously. □

Denote the previous procedure by "Procedure 1". The proof of the following theorem shows how the leftover can be handled.

**Theorem 1.** *Given a cake and* 5 *players, there exists a moving-knife procedure which produces an envy-free division, using a constant number of cuts.*

*Proof.* Use Procedure 1 to obtain an envy-free division for part of the cake, and assume the leftover is $T_1 \cup T_2$, such that when allocating $T_1$ both $P_1$ and $P_2$ have IA over $P_3$, and for $T_2$ both $P_1$ and $P_2$ have IA over $P_4$. We show how $T_1$ can be divided, and $T_2$ can be handled similarly.

On $T_1$, run Procedure 1 again with $P_1$ and $P_3$ performing the Austin's procedure. Part of $T_1$ might remain unallocated, which can be divided into $R_1 \cup R_2$ such that $P_1$ and $P_3$ have IA over another player when dividing $R_1$ and $R_2$ respectively. Suppose $P_1$ and $P_3$ have IA over $P_4$ on $R_1$, then $R_1$ can be divided in the following way: Let $P_2$ and $P_5$ divide $R_1$ into 5 equal pieces by performing the Austin's procedure, and let the players take the largest remaining piece in the following order: $P_4$, $P_3$, $P_1$, $P_2$, $P_5$. $P_4$ envies no one because he picks first; $P_3$ does not envy $P_4$ because of IA; $P_1$ envies neither $P_4$ nor $P_3$ because of IA; $P_2$ and $P_5$ do not envy anyone because they are indifferent about the five pieces. $R_2$ can be divided similarly. □

The proof of the theorem essentially gives the algorithm, which can be viewed as a recursive algorithm that always ends at depth 3. It should be noted that Procedure 1 is "almost discrete". The only part that uses moving knives is the Austin's procedure. To see this, it suffices to check all other operations can be done discretely. Essentially there are two types of operations we used.

1. We decrease a piece until some player is tied with this piece and another one. It can be done by letting each player mark a point on this piece which makes him tied if the part to the left of the point is removed. Then cut along the leftmost mark.
2. We decrease two pieces simultaneously such that some player is always tied between the two, and end when a player is tied between a piece within the two and a piece outside the two. This can be done similarly by making marks and cut along the leftmost one.

Moving-knife procedures are not desirable from an algorithmic perspective. Next we show Procedure 1 can be discretized to yield an approximate solution.

**Theorem 2.** *Procedure 1 can be discretized such that it computes an* $\epsilon$ *envy-free solution in time* $O(\log\left(\frac{M}{\epsilon}\right)^2)$*, using a constant number of cuts.*

*Proof.* We show how to discretize The Austin's procedure by performing binary search on the position of the left knife. The difficulty comes from the fact that a small perturbation of the left knife can result in a large move in the right knife (because the valuation of $P_1$ can be 0 on some interval), and also the exact location of the right knife can not be found using only value queries.

For simplicity, we only show the case of cutting the cake into $k = 2$ equal parts, while the algorithm for $k > 2$ is essentially the same. Let $x$ be a point satisfying $u_1([x, 1]) \geq \frac{1}{2}$. We define $L_x$ to be the largest number that is found by binary search such that $u_1(x, L_x) \leq \frac{1}{2}$, and similarly $R_x$ as the smallest number

(found by binary search) such that $u_1(x, R_x) \geq \frac{1}{2}$. Formally, $L_x$ and $R_x$ are computed by the following binary-search algorithm:

(1) Initialize $L_x = 0, R_x = 1$.
(2) Repeat the following process $\log\left(\frac{M}{\epsilon}\right)$ times: Let $m = \frac{L_x + R_x}{2}$. Set $L_x = m$ if $u_1([x, m]) \leq \frac{1}{2}$, otherwise set $R_x = m$.

The following facts hold for $L_x$ and $R_x$.

1. $L_x$ and $R_x$ are nondecreasing functions of $x$.
2. Every point $y$ that satisfies $u_1([x, y]) = \frac{1}{2}$ is contained in the interval $[L_x, R_x]$.
3. $\frac{1}{2} - \epsilon \leq u_1([x, L_x]) \leq \frac{1}{2} \leq u_1([x, R_x]) \leq \frac{1}{2} + \epsilon$,

We prove the first property briefly: Let $0 < x_1 < x_2 < 1$, assume we run one procedure for $x_1$ and one for $x_2$. The two procedures differ at a step only when $u(x_1, m) \geq \frac{1}{2}$ and $u(x_2, m) \leq \frac{1}{2}$, but then we have $L_{x_1}, R_{x_1} \leq m$ and $L_{x_2}, R_{x_2} \geq m$. If no such step exists, then both procedures produce the same $L_x$ and $R_x$. In either case, $L_{x_1} \leq L_{x_2}$ and $R_{x_1} \leq R_{x_2}$. The second property is trivial from the algorithm. Notice that the length of the interval $[L_x, R_x]$ halves in each iteration, so $R_x - L_x = \frac{\epsilon}{M}$ at the end of the algorithm. By boundedness of the utility function we have $u_1([L_x, R_x]) \leq \epsilon$, and together with the second property this implies the third property.

We are now ready to state the discretized Austin's procedure. Assume that $u_2([0, t]) < \frac{1}{2}$ where $t$ satisfies $u_1([0, t]) = \frac{1}{2}$.

(1) Initialize $l = 0, r = 1$.
(2) Set $x = \frac{l+r}{2}$. If $u_1([0, x]) > \frac{1}{2}$ then set $r = x$ and go to (4); Otherwise, compute $L_x, R_x$.
(3) If $u_2([x, L_x]) > \frac{1}{2} + \epsilon$ set $r = x$; Otherwise if $u_2([x, R_x]) < \frac{1}{2} - \epsilon$ set $l = x$; Otherwise, find a point $y \in [L_x, R_x]$ by binary search such that $u_2([x, y]) \in [\frac{1}{2} - 2\epsilon, \frac{1}{2} + 2\epsilon]$. Return the cut $(x, y)$ and terminate the algorithm.
(4) Repeat step (2) and (3) $\log\left(\frac{M}{\epsilon}\right)$ times.
(5) If the algorithm has not terminated, find a point $y$ in $[L_l, R_r]$ by binary search such that $u_2([l, y]) \in [\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon]$. Return the cut $(l, y)$.

*Claim.* The above algorithm returns a cut $(x, y)$ such that $\frac{1}{2} - 2\epsilon \leq u_i([x, y]) \leq \frac{1}{2} + 2\epsilon$ for $i = 1, 2$.

*Proof.* For any $x$ such that $u_1([0, x]) \leq \frac{1}{2}$, we use $H_x$ to denote the smallest number such that $u_1([x, H_x]) = \frac{1}{2}$. By previous arguments we know $L_x \leq H_x \leq R_x$. We claim that at each step, the algorithm either terminates or the following invariant holds: $u_2([l, H_l]) < \frac{1}{2}$ and $u_2([r', H_{r'}]) > \frac{1}{2}$, where $r' = \min(r, t)$. Initially this is true by assumption; At step (3), we set $r = x$ only when $u_2([x, L_x]) > \frac{1}{2} + \epsilon$, but then $u_2([r, H_r]) \geq u_2([r, L_r]) > \frac{1}{2}$. Similarly when we set $l = x$ the invariant still holds. Also notice that at each step, any point $y \in [L_x, R_x]$ yields a feasible cut $(x, y)$ for player 1 (an interval $I$ is said to be feasible if $\frac{1}{2} - \epsilon \leq u(I) \leq \frac{1}{2} + \epsilon$). If at step (3) the third case happens, i.e. $u_2([x, L_x]) \leq \frac{1}{2} + \epsilon$ and $u_2([x, R_x]) \geq \frac{1}{2} - \epsilon$,

then we know there also exists a feasible point for player 2 in the interval $[L_x, R_x]$. We may not find this point exactly but we lose at most an additive $\epsilon$ by performing binary search.

If the algorithm reaches step (5), we must have $r - l \leq \frac{\epsilon}{M}$. By the invariant just proved, $u_2([l, L_l]) < \frac{1}{2}$ and $u_2([l, R_r]) = u_2([l, r]) + u_2([r, R_r]) > \frac{1}{2}$, so we can find $y \in [L_l, R_r]$ such that $u_2(l, y) \in [\frac{1}{2} - \epsilon, \frac{1}{2} + \epsilon]$. We must have $\frac{1}{2} - \epsilon \leq u_1(l, L_l) \leq u_1(l, y) \leq u_1(l, r) + u_1(r, R_r) \leq \frac{1}{2} + 2\epsilon$. (note: it might happen that $r > t$ at step (5) and $R_r$ is not well defined. In that case just consider $r' = \min(r, t)$ as in proving the invariant.) □

In practice we can not find $t$ exactly so we do not know if $u_2([0, t])$ is less than $1/2$ or not. To overcome this, we can compute $L_0$ and $R_0$. If $u_2([0, L_0]) \leq 1/2 \leq u_2([0, R_0])$, then we directly perform a binary search and return; Otherwise, either $u_2([0, L_0]) > 1/2$ or $u_2([0, R_0]) < 1/2$ and correspondingly we have $u_2([0, t]) > 1/2$ or $u_2([0, t]) < 1/2$. □

## 4 4-Person Discrete Procedure

**Theorem 3.** *Given a cake and* 4 *players, there exists a discrete procedure which produces an envy-free division of part of the cake, such that* $P_1$ *gets at least* $\frac{1}{4}$ *of the whole cake. Furthermore, the leftover can be divided into two parts, such that* $P_1$ *has IA over another player when dividing each part.*

*Proof.* The procedure is essentially the same as Procedure 1. Instead of performing Austin's procedure in the first step, just let $P_1$ cut the cake into 4 equal pieces. Everything that follows is the same. □

**Theorem 4.** *Given a cake and* 4 *players, there exists a discrete procedure which produces an envy-free division of part of the cake, and each player gets at least* $\frac{1}{4}$ *of the whole cake (in his valuation).*

*Proof.* Apply the previous procedure with $P_1$ dividing first, and apply it again to the leftover with $P_2$ dividing, and do the same for $P_3$ and $P_4$. By Theorem 3, $P_1$ already gets at least $\frac{1}{4}$ of the whole cake in the first round. $P_2$ gets at least $\frac{1}{4}$ of the part that was divided in the first round, and he gets at least $\frac{1}{4}$ of the leftover in the second round. The same argument holds for $P_3$ and $P_4$. □

## 5 Conclusion

In this paper we discussed envy-free divisions among 4 and 5 players. We presented a moving-knife procedure that yields envy-free division for 5 players, and a discrete procedure that partially allocates the cake among 4 players and the allocating is both envy-free and proportional. Both procedures use a constant number of cuts. The moving-knife procedure for 5 people can be discretized to obtain an $\epsilon$ envy-free allocation. The running time is $O(\log\left(\frac{M}{\epsilon}\right)^2)$ and the number of cuts is still constant.

The moving-knife procedure for 5 players is based on two ideas: First, we can obtain a partial allocation, such that some IA relationships are created when dividing the remaining part. Second, when the graph of IA has a certain graph minor, we can allocate the whole piece without trimming. In our example, the minor can be found after two recursive calls.

An interesting question is whether this idea can be generalized to $n > 5$. Here the two challenging questions are how a partial allocation can be found, and what kind of IA graph is sufficient for a direct allocation.

An interesting aspect of the problem is its inherent connection to classic algorithmic concepts such as maximum weight matching or price equilibria. Suppose the cake-cutter is allowed to charge different amounts for each piece. Then it is easy to see that she can take an arbitrary cut of the cake and make it envy free by using the dual prices of the maximum weight matching between the players and the pieces. Our problem can be seen as finding the right placement of cuts so that the dual prices of the corresponding maximum weight matching is zero.

Finding a discrete procedure for 4 players is a more challenging open question. Our discrete algorithm finds a partial allocation, and it guarantees that each player has at least one out edge in the IA graph. However in the worst case we can have two cycles of length 2, and we were not able to design a procedure which can augment that particular IA graph.

## Acknowledgments

The authors thank Walter Stromquist for pointing to a number of useful references, and for simplifying the proof of Theorem 2, and the authors want to thank the anonymous reviewers for many helpful suggestions.

## References

1. Steinhaus, H.: The problem of fair division. Econometrica 16, 251–269 (1948)
2. Steinhaus, H.: Sur la division pragmatique. Econometrica (supplement) 17, 315–319 (1949)
3. Stromquist, W.: How to cut a cake fairly. American Mathematical Monthly 87(8), 640–644 (1980)
4. Brams, S.J., Taylor, A.D.: An envy-free cake division protocol. American Mathematical Monthly 102(1), 9–18 (1995)
5. Brams, S.J., Taylor, A.D., Zwicker, W.S.: A moving-knife solution to the four-person envy-free cake division. Problem. Proceedings of the American Mathematical Society 125, 547–554 (1997)
6. Austin, A.K.: Sharing a cake. Mathematical Gazette (6), 437 (1982)
7. Even, S., Paz, A.: A note on cake cutting. Discrete Applied Mathematics (7), 285–296 (1984)
8. Sgall, J., Woeginger, G.J.: A lower bound for cake cutting. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 459–469. Springer, Heidelberg (2003)

9. Edmonds, J., Pruhs, K.: Cake cutting really is not a piece of cake. In: SODA 2006: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm, pp. 271–278. ACM Press, New York (2006)
10. Edmonds, J., Pruhs, K.: Balanced allocations of cake. In: FOCS 2006: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, Washington, DC, USA, pp. 623–634. IEEE Computer Society Press, Los Alamitos (2006)
11. Jeff, E., Kirk, P., Jaisingh, S.: Confidently cutting a cake into approximately fair pieces. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 155–164. Springer, Heidelberg (2008)
12. Su, F.: Rental harmony: Sperner's lemma in fair division. Amer. Math. Monthly 106, 930–942 (1999)
13. Deng, X., Qi, Q., Saberi, A.: On complexity of envy-free cake-cutting (work in progress)

# PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications

Yi Wang[1], Hongjie Bai[1], Matt Stanton[2],
Wen-Yen Chen[1], and Edward Y. Chang[1]

[1] Google Beijing Research, Beijing, 100084, China
{wyi,hjbai,wenyen,edchang}@google.com
[2] Computer Science, CMU, USA
mlstanto@cs.cmu.edu

**Abstract.** This paper presents PLDA, our parallel implementation of Latent Dirichlet Allocation on MPI and MapReduce. PLDA smooths out storage and computation bottlenecks and provides fault recovery for lengthy distributed computations. We show that PLDA can be applied to large, real-world applications and achieves good scalability. We have released MPI-PLDA to open source at http://code.google.com/p/plda under the Apache License.

## 1 Introduction

Latent Dirichlet Allocation (LDA) was first proposed by Blei, Ng and Jordan to model documents [1]. Each document is modeled as a mixture of $K$ latent topics, where each topic, $k$, is a multinomial distribution $\boldsymbol{\phi}_k$ over a $V$-word vocabulary. For any document $d$, its topic mixture $\boldsymbol{\theta}_d$ is a probability distribution drawn from a Dirichlet prior with parameter $\alpha$. For each $i^{th}$ word $w_{d,i}$ in $d$, a topic $z_{d,i}$ is drawn from $\boldsymbol{\theta}_d$, and $w_{d,i}$ is drawn from $\boldsymbol{\phi}_{z_{d,i}}$.

Given an input corpus $W$, the LDA learning process consists of calculating $\Phi$, a maximum-likelihood estimate of model parameters. Given this model, we can infer topic distributions for arbitrary documents. The idea of describing documents in terms of their topic compositions has seen broad application in information-management applications. For example, in a query 'apple pie', LDA can infer from the presence of 'pie' that the meaning of 'apple' is closer to 'fruit' than 'computer'. Using this meaning information obtained by learning an LDA model, documents with the meaning 'fruit' can be effectively identified and returned to answer the 'apple pie' query.

In this paper, we first present LDA and related work in Section 2. In Section 3 we present parallel LDA (PLDA) and explain how it works via a simple example. We then present our two fault-tolerant PLDA implementations (the current core algorithm of PLDA is the AD-LDA algorithm [2]), one on MPI [3] and the other on MapReduce [4]. Section 4 uses two large-scale applications to demonstrate the scalability of PLDA. Finally, we discuss future research plans in Section 5.

## 2 Learning Algorithms for LDA

Blei, Ng and Jordan [1] proposed using a Variational Expectation Maximization (VEM) algorithm for obtaining maximum-likelihood estimate of $\Phi$ from $W$. This algorithm

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 301–314, 2009.

iteratively executes an E-step and an M-step, where the E-step infers the topic distribution of each training document, and the M-step updates model parameters using the inference result. Unfortunately, this inference is intractable, so variational Bayes is used in the E-step for approximate inference. Minka and Lafferty proposed a comparable algorithm [5], which uses another approximate inference method, Expectation Propagation (EP), in the E-step.

Griffiths and Steyvers [6] proposed using Gibbs sampling, a Markov-chain Monte Carlo method, to perform inference. By assuming a Dirichlet prior, $\beta$, on model parameters $\Phi = \{\phi_k\}$ (a set of topics), $\Phi$ can be integrated (hence removed from the equation) using the Dirichlet-multinomial conjugacy. The posterior distribution $P(Z|W)$ can then be estimated using a collapsed Gibbs sampling algorithm, which, in each iteration, updates each topic assignment $z_{d,i} \in Z$ by sampling the full conditional posterior distribution:

$$p(z_{d,i} = k \mid Z_{-(d,i)}, w_{d,i} = v, W_{-(d,i)}) \propto \left(C^{\text{doc}}_{d,k} + \alpha\right) \frac{C^{\text{word}}_{v,k} + \beta}{\sum_{v'} C^{\text{word}}_{v',k} + V\beta}, \quad (1)$$

where $k \in [1, K]$ is a topic, $v \in [1, V]$ is a word in the vocabulary, $w_{d,i}$ denotes the $i^{th}$ word in document $d$ and $z_{d,i}$ the topic assigned to $w_{d,i}$, $W_{-(d,i)}$ denotes the words in the training corpus with $w_{d,i}$ excluded, and $Z_{-(d,i)}$ the corresponding topic assignments of $W_{-(d,i)}$. In addition, $C^{\text{word}}_{v,k}$ denotes the number of times that word $v$ is assigned to topic $k$ not including the current instance $w_{d,i}$ and $z_{d,i}$, and $C^{\text{doc}}_{d,k}$ the number of times that topic $k$ has occurred in document $d$ not including $w_{d,i}$ and $z_{d,i}$. Whenever $z_{d,i}$ is assigned to a sample drawn from (1), matrices $C^{\text{word}}$ and $C^{\text{doc}}$ are updated. After enough sampling iterations to burn in the Markov chain, $\Theta = \{\boldsymbol{\theta}_d\}_{d=1}^{D}$ and $\Phi = \{\boldsymbol{\phi}_k\}_{k=1}^{K}$ can be estimated by

$$\theta_{d,k} = \frac{C^{\text{doc}}_{d,k} + \alpha}{\sum_{k'=1}^{K} C^{\text{doc}}_{d,k'} + K\alpha} \qquad \phi_{v,k} = \frac{C^{\text{word}}_{v,k} + \beta}{\sum_{v'=1}^{V} C^{\text{word}}_{v',k} + V\beta}. \quad (2)$$

Griffiths and Steyvers [6] conducted an empirical study of VEM, EP and Gibbs sampling. The comparison shows that Gibbs sampling converges to a known ground-truth model more rapidly than either VEM or EP.

### 2.1 LDA Performance Enhancement

The computation complexity of Gibbs sampling is $K$ multiplied by the total number of word occurrences in the training corpus. Prior work has explored multiple alternatives for speeding up LDA, including both parallelizing LDA across multiple machines and reducing the total amount of work required to build an LDA model. Relevant parallelization efforts include:

- Nallapati and et al. [7] reported distributed computing of the VEM algorithm for LDA [1].
- Newman and et al. [2] presented two synchronous methods, AD-LDA and HD-LDA, to perform distributed Gibbs sampling. AD-LDA is similar to distributed EM [8] from a data-flow perspective; HD-LDA is theoretically equivalent to learning a mixture of LDA models but suffers from high computation cost.

– Asuncion, Smyth and Welling [9] presented an asynchronous distributed Gibbs sampling algorithm.

In addition to these parallelization techniques, the following optimizations can reduce LDA model learning times by reducing the total computational cost:

– Gomes, Welling and Perona [10] presented an enhancement of the VEM algorithm using a bounded amount of memory.
– Porteous and et al. [11] proposed a method to accelerate the computation of (Eq.1). The acceleration is achieved by no approximations but using the property that the probability vectors, $\boldsymbol{\theta}_d$, are sparse in most cases.

## 3 PLDA

We consider two well-known distributed programming models, MPI [3] and MapReduce [4], to parallelize LDA learning. Before introducing PLDA, we briefly review the AD-LDA algorithm [2], and its dependency on the collective communication operation, *AllReduce*. We show how to express the AD-LDA algorithm [6] in both models of MPI and MapReduce.

### 3.1 Parallel Gibbs Sampling and AllReduce

The AD-LDA algorithm [2] distributes $D$ training documents over $P$ processors, with $D_p = D/P$ documents on each processor. AD-LDA partitions document content $W = \{\boldsymbol{w}_d\}_{d=1}^{D}$ into $\{W_{|1}, \ldots, W_{|P}\}$ and corresponding topic assignments $Z = \{\boldsymbol{z}_d\}_{d=1}^{D}$ into $\{Z_{|1}, \ldots, Z_{|P}\}$, where $W_{|p}$ and $Z_{|p}$ exist only on processor $p$. Document-specific counts, $C^{\text{doc}}$, are likewise distributed; however, each processor maintains its own copy of word-topic counts, $C^{\text{word}}$. We represent processor-specific counts as $C^{\text{doc}}_{|p}$. $C^{\text{word}}_{|p}$ is used to temporarily store word-topic counts accumulated from local documents' topic assignments on each processor.

In each Gibbs sampling iteration, each processor $p$ updates $Z_{|p}$ by sampling every $z_{d,i|p} \in Z_{|p}$ from the approximate posterior distribution:

$$p(z_{d,i|p} = k \mid Z_{-(d,i)}, w_{d,i|p} = v, W_{-(d,i)}) \propto \left(C^{\text{doc}}_{d,k|p} + \alpha\right) \frac{C^{\text{word}}_{v,k} + \beta}{\sum_{v'} C^{\text{word}}_{v',k} + V\beta}, \quad (3)$$

and updates $C^{\text{doc}}_{|p}$ and $C^{\text{word}}$ according to the new topic assignments. After each iteration, each processor recomputes word-topic counts of its local documents $C^{\text{word}}_{|p}$ and uses an AllReduce operation to reduce and broadcast the new $C^{\text{word}}$ to all processors.

### 3.2 Illustrative Example

We use a two-category, nine-document example, originally presented in [12] for explaining LSA, to illustrate how PLDA works. Table 1 shows nine documents separated into two categories, where symbol *h* stands for human computer interaction, and *m* for

**Table 1.** Nine-Document Example

| *d* | *p* | *Document Title* |
|---|---|---|
| *h1* | *p1* | *Human* machine *interface* for ABC *computer* applications |
| *h2* | *p2* | A *survey* of user opinion of *computer system response time* |
| *h3* | *p1* | The *EPS* user *interface* management *system* |
| *h4* | *p2* | *System* and *human system* engineering testing of *EPS* |
| *h5* | *p1* | Relation of *user* perceived *response time* to error measurement |
| *m1* | *p2* | The generation of random, binary, ordered *trees* |
| *m2* | *p1* | The intersection *graph* of paths in *trees* |
| *m3* | *p2* | *Graph minors* IV: Widths of *trees* and well-quasi-ordering |
| *m4* | *p1* | *Graph minors*: A *survey* |

**Table 2.** $C^{doc}$ Matrices on machines $p1$ and $p2$

| *p* | *d* | $C^{doc}_{d,t1}$ | $C^{doc}_{d,t2}$ | *Topic Assignment* |
|---|---|---|---|---|
| *p1* | *h1* | 2 | 1 | human=$t1$, interface=$t1$, computer=$t2$ |
| | *h3* | 2 | 2 | interface=$t1$, user=$t2$, system=$t1$, EPS=$t2$ |
| | *h5* | 2 | 1 | user=$t1$, response=$t2$, time=$t1$ |
| | *m2* | 2 | 0 | trees=$t1$, graph=$t1$ |
| | *m4* | 1 | 2 | survey=$t2$, graph=$t1$, minors=$t2$ |
| *p2* | *h2* | 4 | 2 | computer=$t1$, user=$t1$, system=$t2$, response=$t1$, time=$t2$, survey=$t1$ |
| | *h4* | 2 | 2 | human=$t2$, system=$t1$, system=$t2$, EPS=$t1$ |
| | *m1* | 1 | 0 | trees=$t1$ |
| | *m3* | 2 | 1 | trees=$t1$, graph=$t2$, minors=$t1$ |

mathematical graph theory. There are five document titles (with extracted terms italicized) in the *h* category, labeled from *h1* to *h5*, and four documents in the *m* category, from *m1* to *m4*.

Suppose we use two machines *p1* and *p2* and target for finding two latent topics *t1* and *t2*. Nine documents are assigned to *p1* or *p2* as depicted in the second column of the table. PLDA first initializes each word's topic from a uniform distribution $U(1, K = 2)$. Table 2 depicts the document-topic matrices on machines $p1$ and $p2$, or $C^{\text{doc}}_{|p1}$ $C^{\text{doc}}_{|p2}$, respectively. The first row shows that document $h1$ on machine $p1$ receives topic assignment $t1$ on words *human* and *interface*, and topic assignment $t2$ on word *computer*. Therefore, the $h1$ row of topic counts are 2 for topic $t1$ and 1 for $t2$. PLDA performs this counting process on all documents on machines $p1$ and $p2$, respectively. Notice that $C^{\text{doc}}_{|p1}$ and $C^{\text{doc}}_{|p2}$ reside on local machines, and no inter-machine communication is involved.

The other important data structure is the word-topic matrices depicted in Table 3. For instance, the first column under machine $p1$, $C^{\text{word}}_{w,t1|p1}$, records how many times topic $t1$ is assigned to each word on machine $p1$. The second column under machine $p2$, $C^{\text{word}}_{w,t2|p2}$, records topic $t2$ assignment on machine $p2$. Each machine also replicates a global topic assignment matrix $C^{\text{word}}$, which is updated at the end of each iteration through the AllReduce operation. This is where inter-machine communication takes place.

Next, PLDA performs a number of Gibbs sampling iterations. Rather than performing topic assignment randomly in the initialization step, Gibbs sampling performs topic assignment according to Equation (3). After each iteration, both Tables 2 and 3 are updated. At the end, the master machine outputs $C^{\text{word}}$, on which one can look up for the topic distribution of a word.

**Table 3.** $C^{word}$ Matrices

| | Machine $p1$ | | | | Machine $p2$ | | | |
|---|---|---|---|---|---|---|---|---|
| $w$ | $C^{word}_{w,t1\|p1}$ | $C^{word}_{w,t2\|p1}$ | $C^{word}_{w,t1}$ | $C^{word}_{w,t2}$ | $C^{word}_{w,t1\|p2}$ | $C^{word}_{w,t2\|p2}$ | $C^{word}_{w,t1}$ | $C^{word}_{w,t2}$ |
| *EPS* | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| *computer* | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| *graph* | 2 | 0 | 2 | 1 | 0 | 1 | 2 | 1 |
| *human* | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| *interface* | 2 | 0 | 2 | 0 | 0 | 0 | 2 | 0 |
| *minors* | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| *response* | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| *survey* | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| *system* | 1 | 0 | 2 | 2 | 1 | 2 | 2 | 2 |
| *time* | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| *trees* | 1 | 0 | 3 | 0 | 2 | 0 | 3 | 0 |
| *user* | 1 | 1 | 2 | 1 | 1 | 0 | 2 | 1 |

### 3.3 Parallel LDA Using MPI

The MPI model supports AllReduce via an API function:

```
int MPI_Allreduce(void *sendbuf, void *recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op);
```

When a *worker*, meaning a thread or a process that executes part of the parallel computing job, finishes sampling, it shares topic assignments and waits for AllReduce by invoking `MPI_Allreduce`, where `sendbuf` points to word-topic counts of its local documents: a vector of `count` elements with type `datatype`. The worker sleeps until the MPI implementation finishes AllReduce and the results are in each worker's buffer `recvbuf`. During the reduction process, word-topic counts vectors are aggregated element-wise by the addition operation `op` explained in Section 3.1.

Figure 1 presents the details of Procedure `MPI-PLDA`. The algorithm first attempts to load checkpoints $Z_{|p}$ if a machine failure took place and the computation is in the recovery mode. The procedure then performs initialization (lines 5 to 10), where for each word, its topic is sampled from a uniform distribution. Next, $C^{\text{doc}}_{|p}$ and $C^{\text{word}}_{|p}$ can be computed from the histogram of $Z_{|p}$ (line 12). To obtain $C^{\text{word}}$, Procedure `MPI-PLDA` invokes `MPI_Allreduce` (line 13). In the Gibbs sampling iterations, each word's topic is sampled from the approximate posterior distribution (Eq.3) and $C^{\text{doc}}_{|p}$ is updated accordingly (lines 15 to 19). At the end of each iteration, the procedure checkpoints $Z_{|p}$ (line 20) and recomputes $C^{\text{word}}_{|p}$ and $C^{\text{word}}$ (lines 21 to 22). After a sufficient number of iterations, the converged LDA model is outputted by the master(line 25).

*Performance and Fault Recovery.* Various MPI implementation systems use different AllReduce algorithms; the state-of-the-art is the recursive doubling and halving (RDH) algorithm presented in [3], which was used by many MPI implementations including the well known MPICH2. RDH includes two phases: *Reduce-scatter* and *All-gather*. Each phase runs a recursive algorithm, and in each recursion level, workers are grouped into pairs and exchange data in both directions. This algorithm is particularly efficient when the number of workers is a power of 2, because no worker would be idle during communication.

**Procedure** `MPI-PLDA(`*iteration-num*`)`

**if** *there is a checkpoint* **then**
  $t \leftarrow$ The number of iterations already done;
  Load $Z_{|p}$ from the checkpoint;
**else**
  $t \leftarrow 0$;
  Load documents on current worker $p$ into $W_{|p}$;
  **for** *each word* $w_{d,i|p} \in W_{|p}$ **do**
    Draw a sample $k$ from uniform distribution $U(1, K)$;
    $z_{d,i|p} \leftarrow k$,
  **end**
**end**
Compute $C^{\text{doc}}_{|p}$ and $C^{\text{word}}_{|p}$;
`MPI_Allreduce(`$C^{\text{word}}_{|p}$`,` $C^{\text{word}}$`,` $V \times K$`,` `''float-number'',` `''sum'');`
**for** *;* $t <$ *iteration-num;* $t \leftarrow t + 1$ **do**
  **for** *each word* $w_{d,i|p} \in W_{|p}$ **do**
    $C^{\text{doc}}_{d,z_{d,i}} \leftarrow C^{\text{doc}}_{d,z_{d,i}} - 1$, $C^{\text{word}}_{w_{d,i},z_{d,i}} \leftarrow C^{\text{word}}_{w_{d,i},z_{d,i}} - 1$;
    $z_{d,i} \leftarrow$ draw new sample from (3), given $C^{\text{word}}$ and $C^{\text{doc}}_{d|p}$;
    $C^{\text{doc}}_{d,z_{d,i}} \leftarrow C^{\text{doc}}_{d,z_{d,i}} + 1$, $C^{\text{word}}_{w_{d,i},z_{d,i}} \leftarrow C^{\text{word}}_{w_{d,i},z_{d,i}} + 1$;
  **end**
  Checkpoint $Z_{|p}$;
  Recompute $C^{\text{word}}_{|p}$;
  `MPI_Allreduce(`$C^{\text{word}}_{|p}$`,` $C^{\text{word}}$`,` $V \times K$`,` `''float-number'',` `''sum'');`
**end**
**if** *this is the master worker* **then**
  Output $C^{\text{word}}$;
**end**

**Fig. 1.** The MPI Procedure of PLDA

RDH provides no facilities for fault recovery. In order to provide fault-recovery capability in `MPI-PLDA`, we checkpoint the worker state before AllReduce. This ensures that when one or more processors fail in an iteration, we can roll back all workers to the end of the most recent succeeded iteration, and restart the failed iteration. The checkpointing code is executed immediately before the invocation of `MPI_Allreduce` in `MPI-PLDA`. In practice, we checkpoint only $Z_{|p}$, because $W_{|p}$ can be reloaded from training data, $C^{\text{doc}}_{|p}$ and $C^{\text{word}}$ can also be recovered from the histogram of $Z_{|p}$. The recovery code is at the beginning of `MPI-PLDA`: if there is a checkpoint on the disk, load it; otherwise perform the random initialization.

### 3.4 Parallel LDA Using MapReduce

MapReduce processes input and output in the form of key-value pairs known as *tuples*. A set of tuples is normally distributed across multiple processors, so each processor can

efficiently load and process the local tuple subset, known as a *shard*. The operation of dividing and distributing tuples is known as *sharding*.

A MapReduce job consists of three successive phases: *mapping*, *shuffling* and *reducing*. The mapping and reducing phases are *programmable*. To program the mapping phase for LDA, we define three procedures: `MapperStart`, `Map` and `MapperFlush`. To program the reducing phase, we define `ReducerStart`, `Reduce` and `ReducerFlush`.

For every input shard, the MapReduce implementation system creates a thread, known as a *map worker*, on the processor where the shard resides. Each map worker invokes `Map` to process each tuple in the shard. A map worker invokes `MapperStart` before the first invocation of `Map`, and invokes `MapperFlush` after the last invocation. These user-defined functions invoke an API function `MapperOutput` to output tuples known as *map-outputs*. Map-output tuples are collected and processed by the shuffling phase; values of map outputs that share the same key are aggregated into a new tuple known as *reduce-input*. The value of a reduce-input is a set of values of map-outputs. Reduce-inputs are grouped into reduce-input shards. For each reduce-input shard, the MapReduce implementation system creates a thread, *reduce worker*, which invokes `ReducerStart`, `Reduce` and `ReducerFlush` in turn to process each reduce-input in the local reduce-input shard. All map workers run in parallel, as do all reduce workers. Workers communicate only in the shuffling phase.

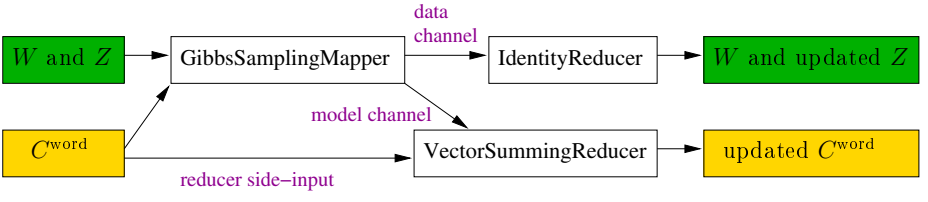



**Fig. 2.** The MapReduce job corresponding to one Gibbs sampling iteration of PLDA

We model each Gibbs sampling iteration of PLDA as a MapReduce job, as illustrated in Figure 2, where the map phase does Gibbs sampling and the reduce phase updates the model and topic assignments. Related procedures are depicted in Figure 3. We organize each fraction of $D_p = D/P$ documents, denoted by $W_{|p}$ in Section 3.1, in an input shard, which is then assigned to a map worker by the MapReduce implementation system. Each map worker loads a local copy of the model, $C^{\text{word}}$, when executing `PLDA-MapperStart`. Then it invokes `PLDA-Map` for each document $\boldsymbol{w}_d \in W_{|p}$ to updates the corresponding topic assignments, $\boldsymbol{z}_d$, and outputs $\boldsymbol{z}_d$ to the channel *data*. After Gibbs sampling on all documents in a shard are finished, the map worker invokes `PLDA-MapperFlush` to output the model update opinion matrix, $\Delta C^{\text{word}}$, to the channel model with each row of $\Delta C^{\text{word}}$ as a map-output tuple. The concept *channel* comes from an extension to the standard MapReduce model that allows us to use two reducers to output both the updated topic assignments, $Z$, and the model, $C^{\text{word}}$. This extension adds an additional parameter to `MapperOutput`, indicating a mapper output channel, where each channel connects to a reducer. Not all

**Procedure** `PLDA-MapperStart`

1 Load $C^{\text{word}}$ updated by previous iteration from GFS;
2 Initialize $\Delta C^{\text{word}}$ as a zero matrix with the same size as $C^{\text{word}}$;
3 Seed the random number generator in a shard-dependent way;

**Procedure** `PLDA-Map` (*key, value*)

1 $d \leftarrow$ parse `key`;
2 $\{\boldsymbol{w}_d, \boldsymbol{z}_d\} \leftarrow$ parse `value`;
3 $C_d^{\text{doc}} \leftarrow$ histogram unique topics in $\boldsymbol{z}_d$;
4 **for** *each* $w_{d,i} \in \boldsymbol{w}_d$ **do**
5 $\quad C^{\text{doc}}_{d,z_{d,i}} \leftarrow C^{\text{doc}}_{d,z_{d,i}} - 1$, $\Delta C^{\text{word}}_{d,z_{d,i}} \leftarrow \Delta C^{\text{word}}_{d,z_{d,i}} - 1$, $C^{\text{word}}_{w_{d,i},z_{d,i}} \leftarrow C^{\text{word}}_{w_{d,i},z_{d,i}} - 1$;
6 $\quad z_{d,i} \leftarrow$ draw new sample from (1), given $C^{\text{word}}$ and $C_d^{\text{doc}}$;
7 $\quad C^{\text{doc}}_{d,z_{d,i}} \leftarrow C^{\text{doc}}_{d,z_{d,i}} + 1$, $\Delta C^{\text{word}}_{d,z_{d,i}} \leftarrow \Delta C^{\text{word}}_{d,z_{d,i}} + 1$, $C^{\text{word}}_{w_{d,i},z_{d,i}} \leftarrow C^{\text{word}}_{w_{d,i},z_{d,i}} + 1$;
8 **end**
9 `Output(channel=data, key=`$d$`, value=`$\{\boldsymbol{w}_d, \boldsymbol{z}_d\}$`);`

**Procedure** `PLDA-MapperFlush`

1 **for** *each unique word* $v$ *in the vocabulary* **do**
2 $\quad$ `Output(channel=model, key=`$v$`, value=`$\Delta C_v^{\text{word}}$`);`
3 **end**

**Fig. 3.** Three MapReduce Procedures for PLDA

MapReduce implementations support this extension. However, we can implement the extension using the standard MapReduce model by appending the channel indicator to each map-output key, then defining `Reduce` to decompose the indicator by parsing from the reduce-input key and invoking different reduce algorithms according the indicator.

In the reduce phase, we use two standard reducers in Figure 2. For each $\boldsymbol{z}_d$, output by GibbsSamplingMapper, IdentityReducer copies it to GFS; for each word $v$ in the vocabulary, VectorSummingReducer aggregates and outputs $C_v^{\text{word}} \leftarrow C_v^{\text{word}} + \sum_{p=1}^{P} \Delta C_{v|p}^{\text{word}}$. Here we use another extension for VectorSummingReducer, the side-input of reducers, which can also be implemented using the standard MapReduce model by appending tuples in side-input, $C^{\text{word}}$, after the standard map-input, $\{W, Z\}$, and defining `Map` identically to output tuples in the side input.

From Figure 2 we see that the input and output of the PLDA MapReduce job are identical—both consist of document accompanied by topics assignments, $W_{|p}$ and $Z_{|p}$, as well the model, $C^{\text{word}}$. This allows us to chain up a series of PLDA MapReduce jobs to model the Gibbs sampling iterations.

*Performance and Fault Recovery.* MapReduce performs AllReduce in the shuffling and reducing phases after the mapping phase. In these phases map outputs are buffered on the local disk, creating a temporary checkpoint, and then aggregated and re-distributed by the shuffling phase. This implementation helps fault recovery. In order to guarantee correct fault recovery, the map workers must execute a *deterministic* map algorithm,

**Table 4.** Comparing MPI and MapReduce in supporting PLDA

| | Communication Efficiency | Inter-iteration fault-recovery | Intra-iteration fault-recovery |
|---|---|---|---|
| MPI | AllReduce through memory/network | By customized checkpointing | Not yet supported |
| MapReduce | *Shuffling* via GFS/Disk IO | Not necessary | Built in |

which ensures that repeating a map input shard gives the same result. This is necessary because when a map worker fails, the corresponding map input shard is repeated, and the previous map-output may already have been consumed by some reducers. We do not want these map-outputs be generated and reduced again when we repeat processing this input shard. For PLDA, over-reduction will over-accumulate some elements in $C^{\text{word}}$. MapReduce performs consistency checking by comparing the output checksums from a map shard. The checksum itself is commutative: if you generate the same set of outputs in a different ordering, the checksum remains the same. This checksum duplication detection avoids over-reduction. But it also requires that the duplication is detectable—the MapReduce program must generate the same map-outputs for an input shard in different runs. However, the Gibbs sampling algorithm of PLDA is *stochastic* instead of deterministic: the outputs of recovered map workers are different from and will be reduced together with those old outputs. To avoid over-reduction in `PLDA-MapperStart`, we seed the random number generator in a shard-dependent way to ensure that whenever a failed map worker is recovered, it generates the same map-output as in its previous run.

Table 4 compares the MPI and MapReduce implementations of PLDA. In the absence of machine failures, `MPI-PLDA` is more efficient because no disk IO is required between computational iterations. When the number of machine is large, and the mean-time to machine failures becomes a legitimate concern, the target application should either use `MapReduce-PLDA` or force checkpoints with `MPI-PLDA`.

## 4 Large-Scale Applications

LDA has been shown effective in many tasks (e.g.,[13,14,15]). In this section, we use two large-scale applications, *community recommendation* and *document summarization*, to demonstrate the scalability of PLDA.

### 4.1 Mining Social-Network User Latent Behavior

Users of social networking services (e.g., Orkut, Facebook, and MySpace) can connect to each other explicitly by adding friends, or implicitly by joining communities. When the number of communities grows over time, finding an interesting community to join can be time consuming. We use PLDA to model users' community membership [16]. On a matrix formed by users as rows and communities as columns, all values in user-community cells are initially unknown. When a user joins a community, the corresponding user-community cell is set to one. We apply PLDA on the matrix to assign

**Table 5.** Speedup Performance of `MPI-PLDA`

| *# Machines* | *Computation* | *Communication* | *Synch* | *Total Time* | *Speedup* |
|---|---|---|---|---|---|
| 1 | 28911s | 0s | 0s | 28911s | 1 |
| 2 | 14543s | 417s | 1s | 14961s | 1.93 |
| 4 | 7755s | 686s | 1s | 8442s | 3.42 |
| 8 | 4560s | 949s | 2s | 5511s | 5.25 |
| 16 | 2840s | 1040s | 1s | 3881s | 7.45 |
| 32 | 1553s | 1158s | 2s | 2713s | 10.66 |
| 64 | 1558s | 1209s | 2s | 2769s | 10.44 |

a probability value between zero and one to the unknown cells. When PLDA assigns a high probability to a cell, this can be interpreted as a prediction that that cell's user would be very interested in joining that cell's community.

The work of [16] conducted experiments on a large community data set of $492,104$ users and $118,002$ communities in a privacy-preserved way. The experimental results show that `MPI-PLDA` achieves effective performance for personalized community recommendation. Table 5 shows the speedup performance and overhead analysis. When we increased the number of machines, we could always reduce computation time in a near-linear fashion. Unfortunately, the communication time increased as number of machines increased. When $32$ machines were used, the communication time approached the computation on a single machine. When $64$ machines were used, the speedup was worse than using $32$ machines. The result was expected due to Amdahl's law: the speedup of a parallel algorithm is limited by the time needed for the overhead or sequential fraction of the algorithm. When accounting for communication and synchronization overheads (see the total time column), the speedup deteriorates as the number of machines increases. Between the two overheads, the synchronization overhead has very little impact on the speedup compared to the communication overhead (which increases with the number of machines). The good news is that when the data size increases (the results of two larger datasets are reported in the next section), we can add more machines to achieve better speedup, because the deterioration point is deferred.

### 4.2 Category-Sensitive Document Summarization

In recent years there is a surge of studies on keyword extraction and document summarization that use graph-based ranking algorithms like PageRank and HITS to rank text entities such as words and sentences. However, in many cases, documents have category labels, a factor ignored in most previous work. Consider e-business websites like `amazon.com`, which categorize products and support reviews by users. It is useful to summarize reviews for each product by extracting that product's most relevant properties. For example, properties such as size, weight, and stand-by time are relevant for mobile phones, whereas properties such as safety, exterior/interior design, and equipment packages are for automobiles.

We can realize this category-sensitive summarization using PLDA. By creating a training document from all sentences in product reviews. By taking each sentence from product reviews as a training document, we run the PLDA learning algorithm to cluster words into topics. This step estimates the conditional probability distribution of words given topics, $P(w|z)$. By normalizing each column of $C^{\text{word}}$, we can also obtain $P(z|w)$. Note that during the Gibbs sampling iterations, every word in the training corpus is assigned a most likely topic. Given the category labels of each sentence, we can estimate the conditional probability distributions of topics given categories ($P(z|c)$) and vice versa ($P(c|z)$) by counting the co-occurrences of topics and categories.

Using the learning result, we can rank all review sentences of a product, given the category of that product. Denote the input reviews by $\mathcal{I} = \{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_D\}$, where $\boldsymbol{w}_d$ represents a sentence. By running the Gibbs sampling inference algorithm, we estimate the topic assignment of every word in $\mathcal{I}$. By counting the co-occurrence of topic assignments and sentences, we can estimate $P(z|\boldsymbol{w}_d)$ and $P(\boldsymbol{w}_d|z)$. The inference result is useful to compute a category-sensitive characteristic measure $\text{char}(\boldsymbol{w}_d; c) = P(\boldsymbol{w}_d|c)P(c|\boldsymbol{w}_d)$, where $c$ denotes the product category of reviews $\mathcal{I}$. $\text{char}(\boldsymbol{w}_d; c)$ is a natural extension of the topic-sensitive characteristic measure, $\text{char}(\boldsymbol{w}_d; z) = P(\boldsymbol{w}_d|z)P(z|\boldsymbol{w}_d)$, proposed by [13]. Expanding $\text{char}(\boldsymbol{w}_d; c)$, we obtain:

$$\begin{aligned} \text{char}(\boldsymbol{w}_d; c) &= P(\boldsymbol{w}_d|c)P(c|\boldsymbol{w}_d) \\ &= \left[\sum_z P(\boldsymbol{w}_d|z)P(z|c)\right]\left[\sum_z P(z|\boldsymbol{w}_d)P(c|z)\right] \end{aligned} \tag{4}$$

where $P(z|c)$ and $P(c|z)$ come from the learning result and $P(\boldsymbol{w}_d|z)$ and $P(z|\boldsymbol{w}_d)$ come from the inference result.

We performed experiments on two datasets: a Wikipedia dataset and a forum dataset. The Wikipedia set consists of $2,122,618$ articles after removing those with less than $100$ words. The forum set consists of $2,450,379$ entries extracted from http://www.tianya.cn. While the effectiveness of PLDA on document summarization on the Wikipedia dataset is reported in [17], we report here our experimental results on scalability conducted upon both datasets.

We measured and compared the speedup of `MPI-PLDA` and `MapReduce-PLDA` using these two datasets. The dataset size and training parameters are shown in Figure 4. The experiments were conducted on up to $1,024$ machines at Google's distributed data centers. Not all machines are identically configured; however, each machine is configured with a CPU faster than 2GHz and memory larger than 4GBytes. We ran Wikipedia dataset on $16/32/64/128/256$ distributed machines. Because the data set is too large to be fit into a single machine's memory, we used $16$ machines as the baseline to measure the speedup of using more than $16$ machines. To quantify speedup, we made an assumption that the speedup of using $16$ machines is $16$ compared to using one machine. This assumption is reasonable for our experiments, since PLDA does enjoy approximately linear speedup when the number of machines is up to $32$. Similarly we ran the forum dataset on $64/128/256/512/1,024$ distributed machines and used $64$ machines as the

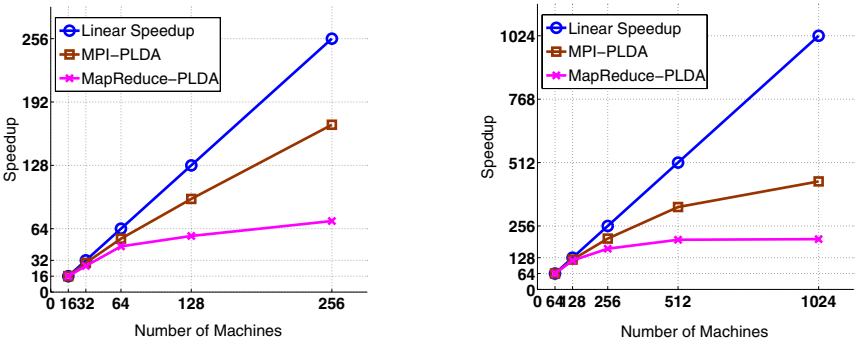



**Fig. 4.** The speedup of (a) Wikipedia: $K = 500$, $V = 20000$, $D = 2,122,618$, TotalWordOccurrences $= 447,004,756$, iterations $= 20$, $\alpha = 0.1$, $\beta = 0.1$. (b) Forum Dataset: $K = 500$, $V = 50000$, $D = 2,450,379$, TotalWordOccurrences $= 3,223,704,976$, iterations $= 10$, $\alpha = 0.1$, $\beta = 0.1$.

**Table 6.** Speedup Performance of MPI-PLDA and MapReduce-PLDA

**(a)** Widipedia dataset (Runtime of 20 iterations)

| *# Machines* | MPI-PLDA | | MapReduce-PLDA | |
|---|---|---|---|---|
| | *Running Time* | *Speedup* | *Running Time* | *Speedup* |
| 16 | 11940s | 16 | 12022s | 16 |
| 32 | 6468s | 30 | 7288s | 26 |
| 64 | 3546s | 54 | 4165s | 46 |
| 128 | 2030s | 94 | 3395s | 57 |
| 256 | 1130s | 169 | 2680s | 72 |

**(b)** Forum dataset (Runtime of 10 iterations)

| *# Machines* | MPI-PLDA | | MapReduce-PLDA | |
|---|---|---|---|---|
| | *Running Time* | *Speedup* | *Running Time* | *Speedup* |
| 64 | 9012s | 64 | 10612s | 64 |
| 128 | 4792s | 120 | 5817s | 117 |
| 256 | 2811s | 205 | 4132s | 164 |
| 512 | 1735s | 332 | 3390s | 200 |
| 1024 | 1323s | 436 | 3349s | 203 |

baseline. Since our aim was to measure speedup, not convergence, we ran 20 iterations on Wikipedia and 10 on the forum dataset[1].

[1] Since the time of running $N$ Gibbs sampling iterations is the same as $N$ times the time of running one iteration, we do not need to run PLDA to convergence in order to measure and compare speedup.

Figure 4 shows that PLDA can achieve linear speedup when the number of machines is below about 100. It can no longer achieve linear speedup when the number of machines continues to increase beyond a data-size dependent threshold. This is expected due to both the increase in the absolute time spending in communication between machines, and the increase in the fraction of the communication time in the entire execution time. When the fraction of the computation part dwindles, adding more machines (CPUs) cannot improve much speedup. Worse yet, when the communication time continues to increase, the computation time reduced by parallelization cannot compensate for the increase in the communication time, and speedup actually decreases. On the one hand, as we previously stated and also observed in [18], when the dataset size increases, and hence the computation time increases, we can add more machines to productively improve speedup. On the other hand, a job will eventually be dominated by the communication overhead, and adding more machines may be counter-productive. Therefore, the next logical step in performance enhancement is to consider communication time reduction [19] (discussed further in concluding remarks).

Comparing `MPI-PLDA` with `MapReduce-PLDA`, `MPI-PLDA` enjoys better speedup than `MapReduce-PLDA`. This is because `MPI-PLDA` uses highly efficient in-memory communication, whereas `MapReduce-PLDA` involves machine scheduling and disk IO between iterations. Indeed, Table 6 shows that when more machines are added, `MPI-PLDA` enjoys better scalability. For instance, the running time that `MPI-PLDA` takes on the Wikipedia dataset, using 256 machines, is $1,130$ seconds, which is less than a half of the running time that `MapReduce-PLDA` takes. While training the Wikipedia set on one machine for 20 iterations can take two days (if we configure that machine with sufficient memory), it takes just 20 minutes to complete on 256 machines.

When the data size becomes much larger and hence more machines (say, tens of thousands) are used, the chance that some machines may fail during a computation iteration becomes non-negligible. In such situation, we can either employ `MapReduce-PLDA` because of its support of intra-iteration fault recovery, or we can support intra-interaction recovery in `MPI-PLDA`[2].

## 5 Conclusion

In this paper, we presented two parallel implementations of PLDA, one based on MPI and the other on MapReduce. We have released the MPI version to open source at http://code.google.com/p/plda under the Apache License.

We plan to further our work in several directions. First, we plan to experiment with different probabilistic distributions or processes such as Pitman Yor and Chinese Restaurant Process. Second, we are investigating algorithms for further speeding up Gibbs sampling. Third, communication time increases as the number of machines increases, and this further reduces the computation fraction of an algorithm. As pointed by J. Demmel [19], since the improvement of CPU performance outpaces the improvement of

[2] When machine can fail frequently during one iteration, hardware redundancy may be necessary to ensure reliability.

IO/communication performance, communication cost increasingly dominates a parallel algorithm. We will look into strategies to reduce communication time.

## References


1. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet allocation. Journal of Machine Learning Research (2003)
2. Newman, D., Asuncion, A., Smyth, P., Welling, M.: Distributed inference for latent Dirichlet allocation. In: NIPS (2007)
3. Thakur, R., Rabenseifnery, R., Gropp, W.: Optimization of collective communication operations in mpich. Int'l Journal of High Performance Computing Applications 19(1), 49–66 (2005)
4. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: OSDI, pp. 137–150 (2004)
5. Minka, T., Lafferty, J.: Expectation-propagation for the generative aspect model. In: UAI (2002)
6. Griffiths, T.L., Steyvers, M.: Finding scientific topics. Proceedings of the National Academy of Sciences of U.S. 101, 5228–5235 (2004)
7. Nallapati, R., Cohen, W., Lafferty, J.: Parallelized variational em for latent dirichlet allocation: An experimental evaluation of speed and scalability. In: ICDM Workshops (2007)
8. Chu, C.-T., Kim, S.K., Lin, Y.-A., Yu, Y., Bradski, G., Ng, A.Y., Olukotun, K.: Mapreduce for machine learning on multicore. In: NIPS (2006)
9. Asuncion, A., Smyth, P., Welling, M.: Asynchronous distributed learning of topic models. In: NIPS (2008)
10. Gomes, R., Welling, M., Perona, P.: Memory bounded inference in topic models. In: ICML (2008)
11. Porteous, I., Newman, D., Ihler, A., Asuncion, A., Smyth, P., Welling, M.: Fast collapsed gibbs sampling for latent dirichlet allocation. In: KDD (2008)
12. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse Processes 25, 259–284 (1998)
13. Cohn, D., Chang, H.: Learning to probabilistically identify authoritative documents. In: ICML (2000)
14. Popescul, A., Ungar, L., Pennock, D., Lawrence, S.: Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In: UAI (2001)
15. Li, L.-J., Wang, G., Fei-Fei, L.: OPTIMOL: automatic online picture collection via incremental model learning. In: CVPR (2007)
16. Chen, W.-Y., Chu, J.-C., Luan, J., Bai, H., Wang, Y., Chang, E.Y.: Collaborative filtering for orkut communities: Discovery of user latent behavior. In: Proc. of the 18th International WWW Conference (2009)
17. Liu, Z., Wang, Y., Zhu, K., Sun, M.: Category-focused ranking for keyword extraction and document summarization. Google Technical Report (submitted for publication, 2009)
18. Chang, E.Y., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., Cui, H.: Psvm: Parallelizing support vector machines on distributed computers. In: NIPS (2007)
19. Demmel, J.: Avoiding communication in dense and sparse linear algebra (invited talk). In: MMDS (2008)

# On Job Scheduling with Preemption Penalties*

Feifeng Zheng[1,2], Yinfeng Xu[1,2], and Chung Keung Poon[3]

[1] School of Management, Xi'an JiaoTong University, Xi'an 710049, China
[2] The State Key Lab for Manufacturing Systems Engineering, Xi'an 710049, China
{zhengff,yfxu}@mail.xjtu.edu.cn
[3] City University of Hong Kong, Hong Kong, China
ckpoon@cs.cityu.edu.hk

**Abstract.** This paper studies the problem of online job scheduling in a model with preemption penalty introduced by Zheng et al. [11]. In such a model with preemption penalty parameter $\rho$, the scheduler has to pay a penalty of $\rho$ times the weight of each aborted job. We consider two cases according to the scheduler's knowledge of $\Delta$ (ratio of length between longest and shortest jobs). In the first case where the exact value of $\Delta$ is known at the beginning, we re-investigate the WAL algorithm of Zheng et al. and prove that it is $((1+\rho)\Delta + o(\Delta))$-competitive for sufficiently large $\Delta$. In particular, when $\rho = 1$, the previous competitive ratio of $3\Delta + o(\Delta)$ proved in [11] is improved to $2\Delta + o(\Delta)$. In the second case where the online strategy only knows beforehand that $\Delta \geq k^3(\rho+1)^3$ for some parameter $k > 1$, a $(\frac{k(1+\rho)}{k-1}\Delta + o(\Delta))$-competitive deterministic strategy is presented. For large $\Delta$, the competitive ratio approaches that of WAL as $k$ increases.

## 1 Introduction

Due to its many applications, online job scheduling is an important area in recent decades, see for example [7,8]. In a typical scenario in manufacturing, there is a manufacturer who may accept or decline jobs that arrive one by one over time. After arrival, each job will stay in the system waiting to be served until it expires, i.e., when its deadline can no longer be met even if it is started at once. The manufacturer will gain a profit for each completed job and his objective is to maximize the total profit. In the standard *preemption-restart model*, preemption is allowed but the aborted job has to be started again from the beginning in order to obtain its profit.

In some applications, starting a job represents a commitment to serve a client. Aborting the job will then likely cause certain degree of discontent in the affected client. Therefore, we are motivated to study the scheduling when there is penalty for preemption. Zheng et al. [11] were the first to study the scheduling with preemption penalties for the above online job scheduling problem (i.e., with the objective of maximizing the total profit). They introduced a model in which the

* The work is partially supported by NSF grants of China no. 70525004, 70702030 and 60736027, and Doctoral Fund of Ministry of Education of China (no. 20070698053).

A. Goldberg and Y. Zhou (Eds.): AAIM 2009, LNCS 5564, pp. 315–325, 2009.

net profit is equal to the total profit of completed jobs minus $\rho$ times the profit of each aborted job, where $\rho$ is called the *preemption penalty parameter*. They presented the WAL strategy which makes use of the knowledge of the ratio, $\Delta$, of length between the longest and shortest jobs and proved it to be $(3\Delta + o(\Delta))$-competitive for $\Delta > 9$ and $\rho = 1$. They also gave a $(1.366\Delta + 0.366)$ lower bound.

Fung [2] considered the general case for $\rho > 0$, and proved a lower bound of $((1 + 1/\rho)^{1/\lceil \Delta \rceil} - 1)^{-1} + 1$. When $\rho = 1$, the lower bound is approximately $\Delta / \ln 2 \approx 1.443\Delta$ for large $\Delta$, improving the previous bound of $(1.366\Delta + 0.366)$. Fung [2] also pointed out that WAL has in fact a competitive ratio of $(2 + \rho)\Delta + o(\Delta)$ for constant $\rho$ and sufficiently large $\Delta$. When $\rho = 1$, the ratio is the same as that in Zheng et al. [11].

In this paper, we consider the problem for general $\rho > 0$ and give a tighter analysis of the WAL strategy. We also discuss another case as well where the online strategy only knows of a lower bound on $\Delta$ initially. The case is motivated by the scenario where the manufacturer may foresee the information of some future jobs at the beginning via certain business technic and hence a lower bound on $\Delta$.

The problem is formally described as follows. There is a manufacturer who processes jobs arriving over time. Each job $J$ has four attributes, $a(J)$, $p(J)$, $w(J)$ and $d(J)$, representing its arrival time, processing time (i.e., job length), profit and deadline respectively. The existence of a job and its attributes are only known on its arrival, i.e., at time $a(J)$. The manufacturer gains a profit of $w(J)$ if he completes job $J$ by its deadline $d(J)$. On the other hand, there is a penalty of $\rho w(J)$ if he starts $J$ but aborts it before its completion. Here the preemption penalty parameter $\rho$ is a real number $\geq 0$. We assume that $1 \leq p(J) \leq \Delta$ where $\Delta$ is a natural number. We will investigate the case where the scheduler knows $\Delta$ *a priori* and the case where the scheduler only knows of a lower bound for $\Delta$. The goal is to maximize the total profit of the completed jobs minus the total penalties caused by abortions. When preemptions cause no penalties, the model reduces to the one that maximizes the total profit of the completed jobs.

To measure the performance of an on-line strategy $\mathcal{A}$, competitive ratio analysis (refer to Borodin and El-yaniv, 1998 [1]) is often used. Denote by $\Gamma_{\mathcal{A}}(I)$ and $\Gamma^*(I)$ the schedules produced by $\mathcal{A}$ and by an optimal offline strategy OPT on a job input set $I$ respectively, and by $|\Gamma_{\mathcal{A}}(I)|$ and $|\Gamma^*(I)|$ the total profit of completed jobs in $\Gamma_{\mathcal{A}}(I)$ and $\Gamma^*(I)$ respectively. Denote by $|P_{\Gamma_{\mathcal{A}}}(I)|$ the total preemption penalty received by $\mathcal{A}$ in $\Gamma_{\mathcal{A}}(I)$. For OPT, it is an offline optimal strategy and never aborts jobs, implying that there is no preemption penalties. Therefore, the competitive ratio of $\mathcal{A}$ is defined as $r_{\mathcal{A}} = \sup_I \frac{|\Gamma^*(I)|}{|\Gamma_{\mathcal{A}}(I)| - |P_{\Gamma}(I)|}$.

### 1.1 Related Work

A closely related body of research is the online scheduling without preemption penalties. Fung et al. [3] studied an online broadcast problem. Translated into our terminology, they proved a $(\Delta + 2\sqrt{\Delta} + 2)$-competitive strategy ACE (Another Completes Earlier). Zheng et al. [10] presented a lower bound of $\Omega(\Delta / \ln \Delta)$.

Ting [9] proved a matching upper bound of $O(\Delta/\log \Delta)$. In all these works, the online strategies make use of knowledge of $\Delta$. Kim et al. [5] presented a 5-competitive greedy strategy GD for the case of unit length of job, i.e., $\Delta = 1$. GD makes an abortion if a newly arrived job has profit $\alpha$ times that of the job being processed. Otherwise GD continues the current service. With simple reasoning, GD is $(4\Delta + 1)$-competitive with $\alpha = 2$ for $\Delta \geq 1$. We may treat GD as an online strategy that acts without the knowledge of $\Delta$ and thus performs poorly for the case where $\Delta > 1$.

Another related line of research considers the scenario of non-preemptive scheduling (i.e., infinite preemption penalties) and the length of job can only be selected from a finite set of real numbers instead of an arbitrary number within $[1, \Delta]$. Lipton and Tomkins [6] studied the scenario to maximize resource utilization. One of their results is a 2-competitive non-preemptive algorithm in the case where $p(J)$ is either 1 or $\Delta$, deadline is tight (i.e., $d(J) = a(J) + p(J)$) and the profit is proportional to the job length (i.e., $w(J) = p(J)$). Goldwasser [4] extended Lipton and Tomkins's work and investigated the case where each job has slack time equal to $k \geq 0$ times of job length, i.e., $d(J) - a(J) = (k+1)p(J)$. They proved a matching upper and lower bound of $(2 + \frac{\lceil \Delta \rceil - 1}{\Delta})$ when $\frac{1}{\Delta} \leq k < 1$, and a matching bound of $(1 + \frac{\lceil \Delta \rceil}{\Delta})$ when $1 \leq k < \Delta$.

### 1.2 Our Results

In this work, we study the preemption-restart model with preemption penalties, aiming at maximizing the net profit. Following the study of Fung [2], we consider the general case for $\rho > 0$. Two cases on the knowledge of $\Delta$ will be investigated. In the first case where the exact value of $\Delta$ is known beforehand, we will re-investigate the WAL strategy in [11] and prove that it is $((1 + \rho)\Delta + o(\Delta))$-competitive for large enough $\Delta$. In particular, WAL is $(2\Delta + o(\Delta))$-competitive when $\rho = 1$, improving the previous bound of $(3\Delta + o(\Delta))$. In the second case, we assume that the online strategy has only certain partial knowledge on $\Delta$. Specifically, it only knows of the minimum job length (assumed to be normalized to 1) and a lower bound on $\Delta$, i.e., $\Delta \geq k^3(1 + \rho)^3$ for some real number $k > 1$. We will prove a $(\frac{k(1+\rho)}{k-1}\Delta + o(\Delta))$-competitive strategy for large enough $\Delta$. As $k$ increases, the ratio approaches $((1 + \rho)\Delta + o(\Delta))$ from below.

The rest of the work is organized as follows. Section 2 discusses the case where the value of $\Delta$ is known beforehand. We reinvestigate the WAL strategy for the case and prove an improved result. In Section 3 we investigates the case where only a lower bound of $\Delta$ is known beforehand, and propose a deterministic online strategy. Section 4 concludes the work.

## 2 Online Scheduling with Knowledge of $\Delta$

In this section, we will give a tighter analysis of the algorithm WAL (Weight-and-Length) proposed in [11]. We first state the WAL Algorithm below.

**The WAL Strategy.** The strategy is triggered when either a job is completed or a new one arrives. When WAL completes a job, it will start to process the job with the largest profit among those that have arrived but not yet satisfied. If a job $R$ arrives while WAL is processing $J$, WAL will abort $J$ to start $R$ if and only if one of the following two conditions is satisfied:

**C1:** $w(R) \geq \beta w(J)$
**C2:** $\alpha w(J) \leq w(R) < \beta w(J)$ and $p(R) < p(J)/\sqrt{\Delta}$

where $\alpha$ is some constant (to be determined later) such that $1 < \alpha < \beta$ and $\beta = \Delta^{1/3}$. (Note that we could have chosen $\beta = \Delta^{\gamma}$ for any positive $\gamma < 1/2$. We fix $\beta$ at $\Delta^{1/3}$ to avoid introducing more symbols in our analysis.)

To analyze the competitive ratio of WAL, we define the notion of *preempting chain* (called *subschedule* in [11]) as follows. A *preempting chain* in the schedule produced by WAL is a sequence of jobs $\sigma = (J_1, \ldots, J_m)$ such that $J_1$ is preceded by an idle period or a completed job, $J_i$ is preempted by $J_{i+1}$ for all $1 \leq i \leq m-1$ and $J_m$ is a completed job.

Since OPT is an optimal offline algorithm, we can assume without loss of generality that it never aborts a job that it starts. By construction of WAL, every job scheduled by OPT must start between the start and completion of some preempting chain unless it is already finished by WAL earlier. (Otherwise, if there is some job $J$ started by OPT at time $t$ outside any preempting chain and $J$ has not been finished by WAL earlier, then $J$ is available at time $t$ while WAL is idle, a contradiction.) Thus, to prove an upper bound on the competitive ratio, it suffices to compute the maximum ratio, $r$, of the profit of OPT to that of WAL on an arbitrary preempting chain. Then the competitive ratio is at most $r + 1$.

The main idea of [11] is to continually modify a preempting chain until it possesses certain desirable properties. Moreover, the ratio of the optimal profit to that obtained by WAL can only increase by the sequence of changes. More precisely, suppose $J_{i+1}$ preempts $J_i$ by condition C2 in $\sigma$. Then we change the weight and processing time of $J_i$ and $J_{i+1}$ so that $J_i$ preempts $J_{i-1}$ by condition C2 and $J_{i+1}$ preempts $J_i$ by condition C1. We achieve this by decreasing $w(J_i)$ to $w(J_{i+1})/\beta$ and swapping $p(J_i)$ and $p(J_{i+1})$. We repeat this change until no more such change is possible. Using this approach, Zheng et al. proved that the competitive ratio is at most $3\Delta + o(\Delta)$.

Here, we use a different approach. To simplify our notations, denote by $a_i, w_i$ and $p_i$ the arrival time, profit and length of job $J_i$ $(1 \leq i \leq m)$ respectively in a preempting chain $\sigma = (J_1, \ldots, J_m)$. Let $|\sigma|$ be the net profit of WAL in processing $\sigma$. Thus $|\sigma| = w_m - \rho(w_1 + \cdots + w_{m-1})$. Let $|\sigma^*|$ denote the total weight of jobs started by OPT while WAL is processing $\sigma$. Let $O_i$ be the set of jobs that are started by OPT while WAL is processing $J_i$, and $|O_i|$ be the total profit of the jobs in $O_i$. Note that $O_i$ may contain multiple jobs. Thus, $|\sigma^*| = \sum_{i=1}^{m} |O_i|$.

We first give some intuition. Observe that $w_i$ and $|O_i|$ grow exponentially in $i$. Thus, the profits of both WAL and OPT in $\sigma$ are almost determined by the last several jobs in $\sigma$. We will prove that no two consecutive abortions are due to condition C2. This will allow us to derive tighter bounds on the profit of WAL

and OPT. For the remaining jobs, we can afford to be slightly more generous in upper bounding the profit of OPT and lower bounding that of WAL.

Below, we state two fundamental lemmas (proved in [11]).

**Lemma 1.** *Consider an arbitrary preempting chain $\sigma = (J_1, \ldots, J_m)$ produced by WAL. If $J_{i+1}$ preempts $J_i$ by condition C2, then $p_{i+1} < \sqrt{\Delta} < p_i$.*

*Proof.* This follows easily from $p_{i+1} < p_i/\sqrt{\Delta} \leq \sqrt{\Delta}$ and $p_i > \sqrt{\Delta} p_{i+1} \geq \sqrt{\Delta}$. □

Combining Lemma 1 and the second inequality in condition C2, no two consecutive abortions can be caused by condition C2.

**Lemma 2.** *Consider an arbitrary preempting chain $\sigma = (J_1, \ldots, J_m)$ produced by WAL.*

*(a) For $1 \leq i < m$, if $p_i \leq \sqrt{\Delta}$, then*

$$|O_i| < \beta(\sqrt{\Delta} + 1)w_i$$

*and if $p_i > \sqrt{\Delta}$, then*

$$|O_i| < \alpha \Delta w_i$$

*(b) If $p_m \leq \sqrt{\Delta}$, then*

$$|O_m| < \beta(\sqrt{\Delta} + 1)w_m$$

*and if $p_m > \sqrt{\Delta}$, then*

$$|O_m| < (\alpha(\Delta - 1) + \beta)w_m$$

*Proof.* The first part of (a) and (b) are easy. If $p_i \leq \sqrt{\Delta}$, OPT can start at most $\sqrt{\Delta} + 1$ jobs of unit length, each of profit less than $\beta w_i$.

To prove the second part of (a), suppose OPT starts $x$ jobs of length larger than $p_i/\sqrt{\Delta}$. (So, $0 \leq x \leq \sqrt{\Delta}$.) Each of these jobs must have profit less than $\beta w_i$. Otherwise, WAL would have aborted $J_i$ by condition C1. Since the total length of these jobs is at least $xp_i/\sqrt{\Delta}$, OPT can start at most $\lceil p_i - xp_i/\sqrt{\Delta} \rceil$ jobs of length $< p_i/\sqrt{\Delta}$ and each of these jobs must have profit at most $\alpha w_i$. Otherwise, WAL would have aborted $J_i$ by condition C2. Hence we have

$$\begin{aligned} |O_i| &\leq x\beta w_i + \lceil p_i - xp_i/\sqrt{\Delta} \rceil \alpha w_i \\ &\leq (p_i + 1)\alpha w_i + (\beta - p_i\alpha/\sqrt{\Delta})xw_i. \end{aligned}$$

If $\beta - \alpha p_i/\sqrt{\Delta} > 0$, then the right hand side of the previous line is maximized when $x = \sqrt{\Delta}$. Hence $|O_i| \leq (\alpha + \beta\sqrt{\Delta})w_i \leq \alpha \Delta w_i$ for large enough $\Delta$.

If $\beta - \alpha p_i/\sqrt{\Delta} \leq 0$, then the right hand side is maximized when $x = 0$. Therefore, $|O_i| \leq \lceil p_i \rceil \alpha w_i \leq \alpha \Delta w_i$.

The proof for the second part of (b) is similar. The only difference is that OPT can start, as the last one in $O_m$, a job of length larger than $p_m/\sqrt{\Delta}$ and profit less than $\beta w_m$. Hence $|O_m| < (\alpha(\Delta - 1) + \beta)w_m$. □

Note: On simplifying the second part of (b), we have $|O_m| \leq (\alpha\Delta + o(\Delta))w_m$. Also, the upper bound of $|O_m|$ does not apply to $|O_i|$ $(1 \leq i < m)$. Based on the above two lemmas, we have the following theorem.

**Theorem 1.** *Suppose the preemption penalty is $\rho$ times the profit of each preempted job (where $\rho$ is a constant $> 0$). Then WAL is $((1+\rho)\Delta + o(\Delta))$-competitive for large enough $\Delta$.*

*Proof.* As discussed before, it suffices to bound the ratio $|\sigma^*|/|\sigma|$ for an arbitrary preempting chain $\sigma = (J_1, \ldots, J_m)$.

We first consider the case when $m \geq 4$. The total weight of the jobs $J_i$ $(1 \leq i \leq m-3)$ is

$$\begin{aligned} & w_1 + \cdots + w_{m-3} \\ & \leq \Big(\frac{1}{\alpha^{m-4}} + \cdots + \frac{1}{\alpha^0}\Big) w_{m-3} \\ & < \frac{\alpha}{\alpha - 1} w_{m-3} \end{aligned}$$

and by Lemma 2,

$$|O_1| + \cdots + |O_{m-3}| < \frac{\alpha^2 \Delta}{\alpha - 1} w_{m-3}.$$

We now bound the ratio $|\sigma^*|/|\sigma|$ by the following case analysis.

*Case 1: $J_m$ preempts $J_{m-1}$ by condition C2.* Then $p_m < \sqrt{\Delta}$ by Lemma 1. Also, by construction of WAL, $w_{m-1} < w_m/\alpha$. Since no two consecutive abortions are due to condition C2, $J_{m-1}$ must abort $J_{m-2}$ due to condition C1. Hence $w_{m-2} \leq w_m/(\alpha\beta)$ and $w_{m-3} \leq w_m/(\alpha^2\beta)$.

Using Lemma 2, $|O_m| < \beta(\sqrt{\Delta}+1)w_m = o(\Delta)w_m$ and $|O_{m-1}| < (\alpha\Delta)w_{m-1}$ $\leq \Delta w_m$. Also, $|O_{m-2}| < \alpha\Delta w_{m-2}$ for large enough $\Delta$. Therefore, $|O_{m-2}| < \alpha\Delta w_m/(\alpha\beta) = o(\Delta)w_m$.

Combining all the parts, we have

$$\begin{aligned} & |O_1| + \cdots + |O_m| \\ & < \frac{\alpha^2\Delta}{\alpha - 1}\frac{w_m}{\alpha^2\beta} + o(\Delta)w_m + \Delta w_m + o(\Delta)w_m \\ & \leq (\Delta + o(\Delta))w_m. \end{aligned}$$

On the other hand, the net profit gained by WAL is at least

$$\begin{aligned} & w_m - \rho\Big(\frac{\alpha}{\alpha - 1} w_{m-3} + w_{m-2} + w_{m-1}\Big) \\ & \geq w_m - \rho\Big(\frac{\alpha}{\alpha - 1}\frac{1}{\alpha^2\beta} + \frac{1}{\alpha\beta} + \frac{1}{\alpha}\Big) w_m \\ & \geq \Big(1 - \frac{\rho}{\alpha}(1 + o(1))\Big) w_m. \end{aligned}$$

Hence the competitive ratio is at most $\alpha\Delta/(\alpha - \rho)$.

*Case 2: $J_m$ preempts $J_{m-1}$ by condition C1.* Then $p_m \le \Delta$ and $w_{m-1} \le w_m/\beta$. Since at least one of $J_{m-3}$ and $J_{m-2}$ is aborted by condition C1, $w_{m-2} \le w_m/(\alpha\beta)$ and $w_{m-3} \le w_m/(\alpha\beta^2)$.

Similar to case 1, we have $|O_m| < (\alpha\Delta + o(\Delta))w_m$, $|O_{m-1}| < \alpha\Delta w_m/\beta = o(\Delta)w_m$ and $|O_{m-2}| < \alpha\Delta w_m/(\alpha\beta) = o(\Delta)w_m$.

Combining all the parts, we have

$$\begin{aligned} &|O_1| + \cdots + |O_m| \\ &< \frac{\alpha^2\Delta}{\alpha - 1}\frac{w_m}{\alpha\beta^2} + o(\Delta)w_m + o(\Delta)w_m + (\alpha\Delta + o(\Delta))w_m \\ &\le (\alpha\Delta + o(\Delta))w_m. \end{aligned}$$

while the net gain by WAL is

$$\begin{aligned} |\sigma| &\ge w_m - \rho\Big(\frac{\alpha}{\alpha - 1}w_{m-3} + w_{m-2} + w_{m-1}\Big) \\ &\ge w_m - \rho\Big(\frac{\alpha}{\alpha - 1}\frac{1}{\alpha\beta^2} + \frac{1}{\alpha\beta} + \frac{1}{\beta}\Big)w_m \\ &\ge (1 - o(1))w_m. \end{aligned}$$

Hence the competitive ratio is at most $\alpha\Delta$.

Setting $\alpha = \rho + 1$, the competitive ratio is at most $\alpha\Delta + o(\Delta)$ in both cases.

Now, consider the case when $m \le 3$. When $m = 1$, Lemma 2 directly gives the bound $\alpha\Delta$. When $m = 2$ or 3, the bounds obtained above for the two cases still hold. This completes the proof of the theorem. □

## 3 Online Scheduling with Partial Knowledge of $\Delta$

In this section we consider the case when the online algorithm has only partial knowledge of $\Delta$. Specifically, it knows the minimum processing time and a lower bound, $\tilde{\Delta}$, on $\Delta$, i.e., $\Delta \ge \tilde{\Delta}$.

Without loss of generality, we assume that the minimum processing time is 1. Thus, the maximum job length ratio $\Delta$ is numerically equal to the maximum job length. We define $\Delta_t$ as the maximum between $\tilde{\Delta}$ and the maximum job length ratio among all the jobs arrived by time $t$. We now present our Dynamic-Preemption-Condition (abbr. DPC) strategy.

**DPC Strategy.** The strategy is triggered either when a job is completed or when a new one arrives. In the former case, DPC will start a job with the largest profit among those that have arrived but not yet satisfied. When a new job $R$ arrives at time $t$, DPC will first update the current $\Delta_t$ if necessary. Then if another job $J$ is being served, DPC will abort $J$ to start $R$ if and only if either conditions below is satisfied:

**C1:** $w(R) \ge \beta_t w(J)$

**C2:** $\alpha w(J) \le w(R) < \beta_t w(J)$ and $p(R) < p(J)/\sqrt{\Delta_t}$ and $J$ did not preempt its predecessor by condition C2

where $\alpha = \rho + 1$ as in previous section and $\beta_t = \Delta_t^{1/3}$. (Again, we could have chosen $\beta = \Delta_t^{\gamma}$ for any positive $\gamma < 1/2$.)

Consider an arbitrary preempting chain $\sigma = (J_1, \ldots, J_m)$. The symbols $a_i$, $w_i$ and $p_i$ denote respectively the arrival time, weight and processing time of job $J_i$ as usual. Furthermore, we let $\Delta_i$ and $\beta_i$ be respectively the value of $\Delta_t$ and $\beta_t$ when $t = a_i$.

**Lemma 3.** *Consider an arbitrary preempting chain $\sigma = (J_1, \ldots, J_m)$ produced by DPC. If $J_{i+1}$ preempts $J_i$ by condition C2, then $p_{i+1} < \sqrt{\Delta_{i+1}} < p_i$.*

*Proof.* By condition C2, $p_{i+1} < p_i/\sqrt{\Delta_{i+1}}$. Since $p_i \leq \Delta_i < \Delta_{i+1}$, we have $p_{i+1} < \sqrt{\Delta_{i+1}}$. Since $p_{i+1} \geq 1$, we have $p_i > \sqrt{\Delta_{i+1}}$. □

**Lemma 4.** *Consider an arbitrary preempting chain $\sigma = (J_1, \ldots, J_m)$ produced by WAL.*

*(a) For $1 \leq i < m$, if $p_i \leq \sqrt{\Delta_i}$, then*

$$|O_i| < \beta_{i+1}(\sqrt{\Delta_i} + 1)w_i$$

*and if $p_i > \sqrt{\Delta_i}$, then*

$$|O_i| < (\alpha\Delta_i + o(\Delta_{i+1}))w_i$$

*(b) If $p_m \leq \sqrt{\Delta_m}$, then*

$$|O_m| < \beta_{m+1}(\sqrt{\Delta_m} + 1)w_m$$

*and if $p_m > \sqrt{\Delta_m}$, then*

$$|O_m| < (\alpha(\Delta_m - 1) + \beta_{m+1} + o(\Delta))w_m$$

*where $\beta_{m+1} = \Delta^{1/3}$.*

*Proof.* The proof is similar to that in Lemma 2, except that $\Delta$ and $\beta$ are replaced by the appropriate $\Delta_j$'s and $\beta_j$'s.

For the first part of (a) and (b), note that OPT can start at most $p_i + 1 \leq \sqrt{\Delta_i} + 1$ unit-length jobs, each of weight less than $\beta_{i+1}w_i$ (where $1 \leq i \leq m$).

For the second part of (a), if OPT starts $\lceil p_i \rceil$ ($\leq \Delta_i$) jobs of unit length, each of weight less than $\alpha w_i$, it gains at most $|O_i| < \alpha\Delta_i w_i$. Alternatively, OPT can start as many long jobs as possible and fill the rest with unit-length jobs. Each long job has length at least $p_i/\sqrt{\Delta_{i+1}}$ and weight less than $\beta_{i+1}w_i$. Hence there are at most $\sqrt{\Delta_{i+1}}$ long jobs and $|O_i| < (\beta_{i+1}\sqrt{\Delta_{i+1}} + \alpha)w_i = o(\Delta_{i+1})w_i$. Combining the two alternatives, $|O_m| < \max\{\alpha\Delta_i, o(\Delta_{i+1})\}w_i \leq (\alpha\Delta_i + o(\Delta_{i+1}))w_i$.

The proof for the second part of (b) is similar. More precisely, OPT can start at most $\Delta_m - 1$ jobs of unit length and profit less than $\alpha w_m$ and then one long job (so that condition C2 will not hold) and profit less than $\beta_{m+1}w_m$. Then $|O_m| < (\alpha(\Delta_m - 1) + \beta_{m+1})w_m$. Alternatively, it can start as many long jobs as possible and fill the rest with unit-length jobs. Then $|O_m| < (\beta_{m+1}\sqrt{\Delta} + \alpha)w_m$ $= o(\Delta)w_m$. Hence $|O_m| \leq (\alpha(\Delta_m - 1) + \beta_{m+1} + o(\Delta))w_m$. □

**Theorem 2.** *Suppose the preemption penalty is $\rho$ times the profit of each preempted job (where $\rho$ is a constant $> 0$) and it is known that $\Delta > \tilde{\Delta} = k^3(\rho+1)^3$ for some $k > 1$. Then DPC is $(\frac{k(\rho+1)}{k-1}\Delta + o(\Delta))$-competitive for large enough $\Delta$.*

*Proof.* The proof idea is the same as that in Theorem 1. By construction of DPC, no two consecutive abortions are caused by condition C2 in an arbitrary preempting chain $\sigma = (J_1, \ldots, J_m)$.

Consider the jobs $J_i$ $(1 \le i \le m-3)$. We have that $w_1 + \cdots + w_{m-3} \le \frac{\alpha}{\alpha-1} w_{m-3}$. By Lemma 4(a), $|O_1| + \cdots + |O_{m-3}| < \frac{\alpha^2 \Delta_{m-3} + \alpha o(\Delta_{m-2})}{\alpha-1} w_{m-3}$.

We now bound the ratio $|\sigma^*|/|\sigma|$ by a case analysis.

*Case 1: $J_m$ preempts $J_{m-1}$ by condition C2.* Then $p_m < \sqrt{\Delta_m}$ by Lemma 3. Also, by construction of WAL, $w_{m-1} \le w_m/\alpha$. Since no two consecutive abortions are due to condition C2, $J_{m-1}$ must abort $J_{m-2}$ due to condition C1. Hence $w_{m-2} \le w_m/(\alpha\beta_{m-1})$ and $w_{m-3} \le w_m/(\alpha^2\beta_{m-1})$.

Using Lemma 4, $|O_m| < (\sqrt{\Delta_m} + 1)\beta_{m+1} w_m = o(\Delta) w_m$ and $|O_{m-1}| < (\alpha\Delta_{m-1} + o(\Delta_m)) w_{m-1} \le \Delta w_m$.

Regarding, $|O_{m-2}|$, if $J_{m-2}$ preempts $J_{m-3}$ by condition C2, then $p_{m-2} \le \sqrt{\Delta_{m-2}}$ and $|O_{m-2}| < \beta_{m-1}(\sqrt{\Delta_{m-2}} + 1) w_{m-2} \le o(\Delta) w_m$. If $J_{m-2}$ preempts $J_{m-3}$ by condition C1, then $|O_{m-2}| < (\alpha\Delta_{m-2} + o(\Delta_{m-1})) w_{m-2} \le (\frac{\Delta_{m-2}}{\beta_{m-1}} + \frac{o(\Delta_{m-1})}{\alpha\beta_{m-1}}) w_m = o(\Delta) w_m$. Thus, $|O_{m-2}| < o(\Delta) w_m$ in both cases.

By a similar argument, we can show that $|O_1| + \cdots + |O_{m-3}| < o(\Delta) w_m$.

Combining all the parts, we have

$$\begin{aligned} &|O_1| + \cdots + |O_m| \\ &< (\Delta + o(\Delta)) w_m. \end{aligned}$$

On the other hand, the net profit gained by WAL is at least

$$\begin{aligned} & w_m - \rho\Big(\frac{\alpha}{\alpha-1} w_{m-3} + w_{m-2} + w_{m-1}\Big) \\ &\ge w_m - \rho\Big(\frac{\alpha}{\alpha-1}\frac{1}{\alpha^2\beta_{m-1}} + \frac{1}{\alpha\beta_{m-1}} + \frac{1}{\alpha}\Big) w_m \\ &\ge \Big(1 - \frac{\rho}{\alpha}\big(\frac{1}{\rho k\alpha} + \frac{1}{k\alpha} + 1\big)\Big) w_m \\ &\ge \frac{k-1}{k\alpha} w_m. \end{aligned}$$

Hence the competitive ratio is at most $\frac{k(1+\rho)\Delta}{k-1} + o(\Delta)$.

*Case 2: $J_m$ preempts $J_{m-1}$ by condition C1.* Then $p_m \le \Delta$ and $w_{m-1} \le w_m/\beta_m$. Since at least one of $J_{m-3}$ and $J_{m-2}$ is aborted by condition C1, $w_{m-2} \le w_m/(\alpha\beta_m)$ and $w_{m-3} \le w_m/(\alpha\beta_m\beta_{m-2})$.

Similar to case 1, we have $|O_m| < (\alpha\Delta + o(\Delta))w_m$ and $|O_{m-1}| < (\alpha\Delta_{m-1} + o(\Delta_m))w_m/\beta_m = o(\Delta)w_m$.

If $J_{m-2}$ preempts $J_{m-3}$ by condition C2, then $p_{m-2} \leq \sqrt{\Delta_{m-2}}$ and $|O_{m-2}| < o(\Delta)w_m$. If $J_{m-2}$ preempts $J_{m-3}$ by condition C1, then $|O_{m-2}| < (\alpha\Delta_{m-2} + o(\Delta_{m-1}))w_m/(\alpha\beta_m) = o(\Delta)w_m$.

Combining all the parts, we have

$$\begin{aligned}
&|O_1| + \cdots + |O_m| \\
&< \frac{\alpha^2\Delta_{m-3} + \alpha o(\Delta_{m-2})}{\alpha - 1}\frac{w_m}{\alpha\beta_m\beta_{m-2}} + o(\Delta)w_m + o(\Delta)w_m + (\alpha\Delta + o(\Delta))w_m \\
&\leq (\alpha\Delta + o(\Delta))w_m.
\end{aligned}$$

while the net gain by WAL is

$$\begin{aligned}
|\sigma| &\geq w_m - \rho\Big(\frac{\alpha}{\alpha - 1}w_{m-3} + w_{m-2} + w_{m-1}\Big) \\
&\geq w_m - \rho\Big(\frac{\alpha}{\alpha - 1}\frac{1}{\alpha\beta_m\beta_{m-2}} + \frac{1}{\alpha\beta_m} + \frac{1}{\beta_m}\Big)w_m \\
&\geq \Big(1 - \frac{\alpha}{\alpha^3k^2} - \frac{\rho}{\alpha^2k} - \frac{\rho}{k\alpha}\Big)w_m \\
&\geq \frac{(k-1)(k\alpha^2+1)}{k^2\alpha^2}w_m \\
&\geq \frac{k-1}{k}w_m.
\end{aligned}$$

Hence the competitive ratio is at most $\frac{k(1+\rho)}{k-1}\Delta + o(\Delta)$. □

By Theorem 2, as $k$ increases, the competitive ratio of DPC approaches $((1+\rho)\Delta + o(\Delta))$, the competitive ratio of WAL in the case with the knowledge of $\Delta$.

## 4 Conclusion

The paper discussed two scenarios of online job scheduling with preemption penalties. For the first scenario with the knowledge of $\Delta$ beforehand, we proved that WAL strategy is $((1+\rho)\Delta + o(\Delta))$-competitive for large enough $\Delta$. For the second scenario where the online strategy has only the knowledge of the lower bound, $k^3(\rho+1)^3$, of $\Delta$, we put forward a $(\frac{k(\rho+1)}{k-1}\Delta + o(\Delta))$-competitive strategy for large enough $\Delta$. The ratio approaches $((1+\rho)\Delta + o(\Delta))$ as $k$ increases.

For the first case, there is still a gap around $0.557\Delta$ between upper and lower bounds for the special case where $\rho = 1$. An obvious open question is where the true competitive ratio lies within the range $[1.443\Delta, 2\Delta]$. Moreover, it is interesting to find out whether randomization helps to break the lower bound of $1.443\Delta$ for $\rho = 1$.

## References


1. Borodin, A., El-yaniv, R.: Online computation and competitive analysis. Cambridge University Press, Cambridge (1998)
2. Fung, S.P.Y.: Lower bounds on online deadline scheduling with preemption penalties. Information Processing Letters 108(4), 214–218 (2008)
3. Fung, S.P.Y., Chin, F.Y.L., Poon, C.K.: Laxity helps in broadcast scheduling. In: Coppo, M., Lodi, E., Pinna, G.M. (eds.) ICTCS 2005. LNCS, vol. 3701, pp. 251–264. Springer, Heidelberg (2005)
4. Goldwasser, M.H.: Patience is a virtue: the effect of slack on competitiveness for admission control. Journal of Scheduling 6(2), 183–211 (2003)
5. Kim, J.-H., Chwa, K.-Y.: Scheduling broadcasts with deadlines. In: Warnow, T.J., Zhu, B. (eds.) COCOON 2003. LNCS, vol. 2697, pp. 415–424. Springer, Heidelberg (2003)
6. Liption, R., Tomkins, A.: Online interval scheduling. In: Proc. 5th ACM-SIAM Symposium on Discrete Algorithms, pp. 302–311 (1994)
7. Pardalos, P.M., Resende, M.G.C. (eds.): Handbook of Applied Optimization. Oxford University Press, Oxford (2002)
8. Pruhs, K., Torng, E., Sgall, J.: Online scheduling. In: Leung, J.Y.T. (ed.) Handbook of scheduling: algorithms, models and performance analysis, vol. 15, pp. 15–1–15–41. CRC Press, Boca Raton (2004)
9. Ting, H.F.: A near optimal scheduler for on-demand data broadcasts. Theoretical Computer Science 410(1-3), 77–84 (2008)
10. Zheng, F.F., Fung, S.P.Y., Chan, W.T., Chin, F.Y.L., Poon, C.K.: Improved on-line broadcast scheduling with deadlines. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 320–329. Springer, Heidelberg (2006)
11. Zheng, F.F., Xu, Y.F., Zhang, E.: On-line production order scheduling with preemption penalties. Journal of Combinatorial Optimization 2(13), 189–204 (2007)

# Author Index