

Cell Probe Lower Bounds For Succinct Data Structures

Alexander Golynski

Google Inc., New York City *

agolynski@google.com

Abstract

In this paper, we consider several static data structure problems in the deterministic cell probe model. We develop a new technique for proving lower bounds for succinct data structures, where the redundancy in the storage can be small compared to the information-theoretic minimum. In fact, we succeed in matching (up to constant factors) the lower order terms of the existing data structures with the lower order terms provided by our lower bound. Using this technique, we obtain (i) the first lower bound for the problem of searching and retrieval of a substring in text; (ii) a cell probe lower bound for the problem of representing permutation π with queries $\pi(i)$ and $\pi^{-1}(i)$ that matches the lower order term of the existing data structures, and (iii) a lower bound for representing binary matrices that is also matches upper bounds for some set of parameters. The nature of all these problems is that we are to implement two operations that are in a reciprocal relation to each other (search and retrieval, computing forward and inverse element, operations on rows and columns of a matrix). As far as we know, this paper is the first to provide an insight into such problems.

1 Introduction

The current state of research in data structures reveals a gap between the number of techniques developed for upper and lower bounds. For example, consider the problem of text searching and retrieval: we are to preprocess a given text T in order to perform searching queries “does a given pattern X occurs in the text T ?” and reporting queries “find the location of the j -th occurrence of X in T ” efficiently. The upper bound research has been very prolific. The data structures developed for this problem include (to name a few): suffix trees [37, 25, 36, 14, 7, 20], suffix arrays [24, 22, 18, 19, 33, 34], FM-indices [9, 10, 11], wavelet trees [17], and their numerous modifications and improvements, e.g. we refer the reader to the extensive survey by

Navarro and Makinen [30]. The lower bound story is not that impressive: it includes the results of Demaine and López-Ortiz [5] and its improvements by Golynski [15, Chapter 4] and Gal and Miltersen [12]. Both of these results can only be applied in a restricted model, where we require that the text *must* be stored in the *raw form* together with a small index to allow an efficient implementation of the searching or reporting queries. However, most of the data structures mentioned above do not possess this property, one of the few results that store the text in the raw form is [3] by Barbay et al.

One of the most natural models for studying data structures was proposed by Minsky and Papert [28] and also by Yao [38]. In his influential paper “Should Tables be Sorted?” called the *cell probe model* which is now widely accepted as a framework for proving lower bounds. In this model, we have an array S of cells, each cell consists of w bits. An algorithm in this model can be viewed as a decision tree, the nodes of the tree are labeled with “ $S[i] = ?$ ”, where i is an index into S , i.e. a value from 1 to $|S|$. The outgoing edges are labeled with $[2^w]$. The *time cost* of the data structure is defined as the depth of this tree, and the *space cost* is defined as the size of the storage, $|S|$. This means that the cell probe model is only concerned with the number of cells accessed and the number of cells stored, while the computation is free, and no restrictions made on the way the data is represented. Thus, the lower bounds obtained in this model will also apply to any other reasonable model of computation, e.g. the RAM model. This model provides the first information-theoretic insight into a problem ignoring the computation issues, it also grasps the word parallelism exploited by modern data structures. Still, proving meaningful lower bounds in this model is hard even for simple data structural problems. We are aware of few techniques for proving lower bounds in this model: (i) communication complexity based (richness, and round elimination) [27] that led to a sequence of papers for the predecessor problem [1, 26, 27, 4, 35], and nearest neighbor search problem; and (ii) based on a certain error correction property of queries, that was used in the result of Gál and

*This work is done when the author was a student at University of Waterloo, and partly published in his PhD dissertation.

Miltersen for the polynomial evaluation problem [12]; (iii) Pătraşcu and Thorup [32] also improved communication complexity techniques and made it possible to distinguish between linear and quadratic space for example, but have not addressed the problem of distinguishing between $n \lg^c(n)$ spaces for different constants c [31]. This also implies that sub-linear redundancies cannot be analyzed with their methods. In fact, in [12], they stated “We don’t know how to prove similar lower bounds for natural storage and retrieval problems such as Substring Search. However, we get a natural restriction of the cell probe model by looking at the case of systematic or index structures.” We are not aware of reciprocal property being defined in earlier literature or any lower bounds for deterministic static succinct data structures except for the mentioned above (the results of this paper are extended abstract of [15, Chapter 2, Chapter 5]). We are aware of the work of Farzan and Munro [8] who use the techniques that are similar to [15] and this extended abstract.

To tackle with difficulties of the cell probe model, one can add a restriction that the data structure under investigation has to be an indexing data structure. In this more restricted model (called indexing model), the data (e.g. text) has to be stored in the raw form together with a small index of size r . The algorithm is allowed to access the index for free (imagine, that the index is stored in fast memory), and also allowed to perform unlimited computation. The algorithm is charged 1 unit of time for each access it performs to the raw data, and is charged r units of space for storing the index. Let us denote the worst case time cost by t (this model was also earlier introduced and used by Yao [39], and by Demaine and López-Ortiz [6]) In [6], they developed a technique (resembling the techniques in Yao [39] and in Gennaro and Trevisan [13]) to show $r = \Omega((L \lg L)/t)$ for substring report queries in the case of binary text of length L , patterns of length $p = \lg L - o(\lg L)$, and time costs $t = o(\lg L / \lg \lg L)$. Recently, Golynski [15] improved their techniques by relaxing the restriction $t = o(\lg L / \lg \lg L)$ to $t \leq \sqrt{\lg L / \lg L}$ and improving the constant factor in the dependence between r and t . Later, Gál and Miltersen [12] proved that $r = \Omega(L/(t \lg L))$ for the problem of implementing the substring search queries on binary texts of length L and patterns of length $p = \Theta(\lg L)$.

The variant of the substring report problem that we consider here is to represent a given text T of length L and perform queries **access** and **search** efficiently. The **access** $_T(i)$ query returns the substring of the text starting from position i of length p , where p is a given parameter; and **search** $_T(X, j)$ returns the position of j -th occurrence of a given pattern X of length p if it exists,

and -1 otherwise. We call this problem the **TEXTSEARCH** problem. The important special case of this problem is $p = 1$, then **access** $_T(i)$ returns the i -th character of T , and **search** $_T(c, j)$ returns the j -th occurrence of character c in T . This problem is called *access/select problem*.

We also consider the **PERMS** problem, and the **BINREL** problem. The permutations problem can be defined as follows. We are to represent a permutation π on n elements, and implement queries $\pi(i)$ and $\pi^{-1}(i)$ for $i \in [n]$. This problem was first considered by Hellman [21], he proposed a method of computing $\pi^{-1}(i)$ queries (inverting a permutation) using an array of n cells that store the values $\pi(i)$ together with an additional array of r cells, the time complexity of implementing $\pi^{-1}(i)$ is $t = n/r$ (obviously, the time complexity of $\pi(i)$ queries is $O(1)$ in this representation). Yao [39] stated that his bounds are optimal up to constant factors in the indexing model. Later, Munro et al. [29] rediscovered these results, and added another (non-indexing) data structure based on Benes networks that uses $O(n(\lg \lg n)^2 / (\lg n)^2)$ extra cells and implements $\pi(i)$ and $\pi^{-1}(i)$ queries in $O(\lg n / \lg \lg n)$ time. We show that both data structures of Hellman and Munro et al. are optimal up to constant factor in the cell probe model.

The **BINREL** problem is to represent an $m \times n$ binary matrix R with f 1-bits in it and implement queries **RSe1** and **CSe1**, where **RSe1** $_R(i, x)$ (respectively, **CSe1** $_R(x, j)$) returns the position of the x -th 1-bit in the i -th row (respectively, j -th column) of R , it returns -1 if the i -th row (respectively, the j -th column) contains less than x 1-bits. This problem was considered in Barbay et al. [2], their data structure is based on the work of Golynski et al. [16] representation of strings on large alphabets using the **PERMS** problem. They proposed two encodings called *Label* and *Object* that in addition to **RSe1** and **CSe1** implement two queries called **RRank** and **CRank**, where **RRank** $_R(i, j)$ (respectively, **CRank** $_R(i, j)$) return the number of 1-bits in the i -th row (respectively, j -th column) up to and including the position j (respectively, i). Golynski [15][Chapter 2] improved the data structure of Barbay et al. [2] and showed that there exist three representations called *Row*, *Column* and *Benes*. Figure 1 shows the comparison between Barbay et al. [2] and new results: This encoding achieved the succinct space bound $\Upsilon + o(\Upsilon)$ for all values of n , m , and f , while the encoding of Barbay et al. [3] only achieves this bound in the case where $f = \max\{n, m\}^{o(1)}$. Here we show a lower bound for this problem in the cell probe model that is tight for the case where n , m and f are polynomially related, i.e. $n = m^\alpha$ and $f = m^\beta$ for constants α and β such that $1 < \alpha < \beta < 1 + \alpha$.

Name	Row	Column	Benes	Label	Object
RRank	$\lg \lg \rho$	$\lg \lg \rho$	$\lg \lg \rho \left(\frac{\lg \rho}{\lg w} + 1 \right)$	$\lg \lg \xi$	$\lg \lg \xi \lg \lg \lg \xi$
RSel	1	$\frac{\lg \lg \rho}{\lg \lg \lg \rho}$	$\frac{\lg \rho}{\lg w}$	1	$\lg \lg \xi$
CRank	$\lg \lg \rho$	$\lg \lg \rho$	$\lg \lg \rho \left(\frac{\lg \rho}{\lg w} + 1 \right)$	$\lg \lg \xi \lg \lg \lg \xi$	$\lg \lg \xi$
CSel	$\frac{\lg \lg \rho}{\lg \lg \lg \rho}$	1	$\frac{\lg \rho}{\lg w}$	$\lg \lg \xi$	1
Space	$\Upsilon + O\left(\frac{\Upsilon \lg \lg \lg \rho}{\lg \lg \rho}\right)$	$\Upsilon + O\left(\frac{\Upsilon \lg \lg \lg \rho}{\lg \lg \rho}\right)$	$\Upsilon + O(f)$	$f(\lg \xi + o(\lg \xi))$	$f(\lg \xi + o(\lg \xi))$

where $\Upsilon = f \lg(nm/f) - O(f)$ is the information-theoretic minimum space to encode an $m \times n$ matrix with f 1-bits in it, $\rho = nm/f$ is the *inverse density* of R , and $\xi = \min\{m, n\}$.

Figure 1: Bounds for the BINREL problem (all running times are asymptotic).

The main contributions of this paper are: (i) a new technique for proving cell probe lower bounds for problems that possess the reciprocal property (we will define it in the next section); (ii) the first cell probe lower bound for the TEXTSEARCH problem, in particular, for $w = \lg L$ and patterns of length $p = \lg L / \lg \sigma$, where σ is the alphabet size, we show that the redundancy must be linear in terms of the information-theoretic lower bound to store the string, i.e. $L \lg \sigma / \lg L$ cells; (iii) a cell probe lower bound for the PERMS problem that matches both Hellman [21] and Munro et al. [29] representation up to constant factors (currently, these are the only known data structures for this problem), in particular, for cell size $w = \lg n$, we show that the extra space should be at least $\Omega(n/tt')$ cells, where t and t' are the cell probe complexities of π and π^{-1} operations respectively; (iv) a cell probe lower bound for the access/select problem; (v) a cell probe lower bound for the BINREL problem that matches the bounds mentioned above for some choices of parameters n , m , and f .

2 Preliminaries

In this section, we will define the notion of the reciprocal property. From now on, to avoid confusion, we will use the notation " $\pi(i)$ " to denote the query $\pi(i)$, and $\pi(i)$ will denote the value where i is mapped by π , we will use similar notation for other queries as well.

Let \mathcal{H} be the set of combinatorial objects. Let $\Upsilon = \left\lceil \frac{\log |\mathcal{H}|}{w} \right\rceil$ be the information-theoretic minimum space to represent an object from \mathcal{H} . For example, in this chapter, we consider several types of combinatorial objects: permutations π on n elements ($|\mathcal{H}| = n!$), texts T of length L over an alphabet of size σ ($|\mathcal{H}| = \sigma^L$), and binary matrices R of size $m \times n$ with f 1-bits ($|\mathcal{H}| = \binom{mn}{f}$). Let the size of the storage $|S| = \Upsilon + r$ cells, where r is called the *redundancy*. A *storage scheme*

is an injective mapping Rep from objects \mathcal{H} to arrays S (we only consider the deterministic storage schemes Rep here). Let $\mathcal{Q} = \mathcal{F} \cup \mathcal{I}$ be the set of queries that are to be implemented, where \mathcal{F} called *forward queries*, and \mathcal{I} called *inverse queries*.

For simplicity, we only consider the sets of queries \mathcal{Q} so that the object $B \in \mathcal{H}$ can be reconstructed only using the answers to \mathcal{Q} . This is not the case, for example, for the decision version of the substring searching problem, where we are to search for patterns of given length (e.g., patterns of length 1). This is not the essential constraint, however, and our techniques can be potentially applied to such problems as well (we omit the discussion until the journal version).

For example, for the problems that we consider in this paper, forward and inverse queries are defined in Figure 2.

Let us fix a combinatorial object B . Depending on B , we fix two sets of queries: a subset of the forward queries $\mathcal{F}_B \subset \mathcal{Q}$ and a subset of the inverse queries $\mathcal{I}_B \subset \mathcal{Q}$ so that $|\mathcal{F}_B| = |\mathcal{I}_B|$. We also fix a bijection η_B between these sets. The sets \mathcal{F}_B and \mathcal{I}_B and the bijection η_B will be chosen later depending on the problem in question. We start by giving some intuition about them. The first property (i) is that the object B is uniquely determined by either of the two: the set \mathcal{F}_B (namely, an encoding of the parameters of every query in \mathcal{F}_B) and (an encoding of) the values returned by all the queries in \mathcal{F}_B , or the set \mathcal{I}_B and the values returned by all the queries in \mathcal{I}_B . The correspondence η_B between forward and inverse queries will be chosen such that a forward query $q \in \mathcal{F}_B$ and its counterpart $q' = \eta(q) \in \mathcal{I}_B$ are "responsible for the same part" of the object B . Such a pair q, q' is called a *reciprocal pair*, and q and q' are called *reciprocal* (with respect to the object B) to each other. The second property (ii) is that the object B is uniquely determined by the sets $\mathcal{F}_B, \mathcal{I}_B$ and the mapping η_B .

Problem name	Forward queries	Inverse queries
PERMS	" $\pi(i)$ "	" $\pi^{-1}(j)$ "
TEXTSEARCH	" $\text{access}_T(i)$ "	" $\text{search}_T(X, j)$ "
BINREL	" $\text{R Sel}_R(i, x)$ "	" $\text{C Sel}_R(x', j)$ "

Problem	\mathcal{F}_B	\mathcal{I}_B	Reciprocity condition
PERMS	all	all	$\pi(i) = j$
TEXTSEARCH	$i \equiv 1 \pmod p$	$\text{search}_T(X, j) \equiv 1 \pmod p$	$\text{search}_T(X, j) = i$
BINREL	$\text{R Sel}_R(i, x) \neq -1$	$\text{C Sel}_R(x', j) \neq -1$	$\text{R Sel}_R(i, x) = j$ and $\text{C Sel}_R(x', j) = i$

Figure 2: Reciprocal queries for PERMS, TEXTSEARCH and BINREL.

For our problems, the sets \mathcal{F}_B and \mathcal{I}_B are chosen as follows, where the fourth column gives the condition under which the queries $q \in \mathcal{F}_B$ and $q' \in \mathcal{I}_B$ are reciprocal, see Figure 2. That is, all the queries " $\pi(i)$ " are in \mathcal{F}_π ; all the queries " $\pi^{-1}(j)$ " are in \mathcal{I}_π ; and " $\pi(i)$ " is reciprocal to " $\pi^{-1}(j)$ " if $j = \pi(i)$. For the TEXTSEARCH problem, \mathcal{F}_T consists of the queries " $\text{access}_T(ip+1)$ " for integer i ; only the queries " $\text{search}_T(X, j)$ " that return a value which is 1 modulo p are in \mathcal{I}_T ; and " $\text{access}_T(ip+1)$ " is reciprocal to " $\text{search}_T(X, j)$ " if the j -th occurrence of X is at position $ip+1$ (i.e. they are "responsible" for the substring of T from position $ip+1$ to position $(i+1)p$). For the BINREL problem, all the queries that return a positive value are in \mathcal{F}_R and \mathcal{I}_R , and " $\text{R Sel}_R(i, x)$ " is reciprocal to " $\text{C Sel}_R(x', j)$ " if they select the same 1-bit in R .

DEFINITION 2.1. Consider a problem of representing objects \mathcal{H} . We say that the problem possesses the reciprocal property if for every object $B \in \mathcal{H}$, we can find subsets $\mathcal{F}_B \subseteq \mathcal{F}$, $\mathcal{I}_B \subseteq \mathcal{I}$, and a bijection η_B between them, such that for any subsets $\mathcal{F}_B^* \subseteq \mathcal{F}_B$ and $\mathcal{I}_B^* \subseteq \mathcal{I}_B$, the object B is uniquely identified by

- sets \mathcal{F}_B , \mathcal{I}_B , \mathcal{F}_B^* , and \mathcal{I}_B^* ,
- the answers to all the queries in \mathcal{F}_B^* and in \mathcal{I}_B^* , and
- bijection $\eta_B|_{\mathcal{F}_B^*}$ restricted to the set $\mathcal{F}_B^* = \mathcal{F}_B \setminus \mathcal{F}_B^* \setminus \eta^{-1}(\mathcal{I}_B^*)$ that is reciprocal to the set $\mathcal{I}_B^* = \eta_B(\mathcal{F}_B^*) = \mathcal{I}_B \setminus \mathcal{I}_B^* \setminus \eta(\mathcal{F}_B^*)$.

First notice that, the property (i) corresponds to the case $\mathcal{F}_B^* = \mathcal{F}_B$ and $\mathcal{I}_B^* = \mathcal{I}_B$; the property (ii) corresponds to the case $\mathcal{F}_B^* = \mathcal{I}_B^* = \emptyset$. \mathcal{F}_B^* (respectively, \mathcal{I}_B^*) are the sets of forward (respectively, inverse) queries q such that "we do not know" the answer for q and its reciprocal q' ; this definition gives us the mapping between such queries. Also observe that, in our three examples, the situation is somewhat simpler (we made

the definition a little more general than necessary for our purposes): if we know that q and q' are reciprocal to each other, then we also "know" the answers to both q and q' . For the PERMS problem, if $q = \pi(i)$ and $q' = \pi^{-1}(j)$ then $\pi(i) = j$ and $\pi^{-1}(j) = i$. For the TEXTSEARCH problem, if $q = \text{access}_T(ip+1)$ and $q' = \text{search}_T(X, j)$, then $\text{access}_T(ip+1) = X$ and $\text{search}_T(X, j) = ip+1$. Also, for the BINREL problem, if $q = \text{query R Sel}(i, x)$ and $q' = \text{query C Sel}(x', j)$, then $\text{R Sel}(i, x) = j$ and $\text{C Sel}(x', j) = i$. Thus, for our three problems, this definition gives us that for every reciprocal pair (q, q') , we either "know" the answer to q or to q' (or to both). Given this information, it is not hard to reconstruct the underlying object B . In particular, for the PERMS problem, we either know the value of $\pi(i)$ or we know some j , such that $\pi^{-1}(j) = i$, and thus π can be reconstructed. In the TEXTSEARCH problem, we either know the value of $\text{access}(ip+1)$ or we know the query $q' = \text{search}(X, j)$ such that the answer to q is $ip+1$; in both cases we can reconstruct the substring of T from position $ip+1$ to position $(i+1)p$. For the binary relation problem, we start with a matrix R initialized with 0 entries. For each query q that we know an answer to, we write a 1-bit to the corresponding entry in R , e.g. if the answer to the query $\text{R Sel}(i, x)$ is j , then a 1-bit is written to location (i, j) of R . Since for every pair of reciprocal queries we know the answer to at least one, all the f of 1-bits will be written to R , and therefore R is reconstructed correctly. We conclude with the following

THEOREM 2.1. The PERMS problem, the TEXTSEARCH problem, and the binary relation problem possess the reciprocal property.

DEFINITION 2.2. We call a reciprocal problem of representing objects \mathcal{H} a $(\Upsilon, r, t, t', \gamma, w)$ -problem, if the cell size is w bits, the storage size is $\Upsilon + r$ cells, where $\Upsilon = \left\lceil \frac{\log |\mathcal{H}|}{w} \right\rceil$ for $r \geq 0$, the cell probe complexity of the

forward (respectively, inverse) queries is t (respectively, t'), and $|\mathcal{F}_B| = |\mathcal{I}_B| = \gamma$ for all $B \in \mathcal{H}$.

3 Compression Lemma

In this section, we derive the main tool of this paper. To simplify the presentation in this extended abstract, we will not try to optimize the results for the case where one of the running times t , and t' is much bigger than the other.

LEMMA 3.1. Consider a $(\Upsilon, r, t, t', \gamma, w)$ -problem with $r = O(\Upsilon)$. If the sets \mathcal{F}_B and \mathcal{I}_B can be encoded using at most

$$(3.1) \quad \frac{1}{64} \frac{\Upsilon^2 w}{\gamma t t'}$$

bits for any $B \in \mathcal{H}$, and if

$$(3.2) \quad \max\{t, t'\} < \min \left\{ 2^{w/4}, \frac{1}{16} \frac{\Upsilon w}{\gamma \lg w} \right\},$$

then

$$(3.3) \quad r \geq \frac{1}{32} \frac{\Upsilon^2}{\gamma t t'}$$

cells of memory.

Proof. Definition 2.1 will be used in the following fashion: for a given object B we encode the set of forward \mathcal{F} and inverse \mathcal{I} queries that are of interest to us (and from which B can be recovered from) in at most (3.1) space. For example, for the PERMS problem, the encoding of \mathcal{F} and \mathcal{I} are trivial, but for the TEXTSEARCH problem, we need to describe the set of all pairs (pattern X , index j) for which we want to perform $\text{search}_T(X, j)$. The queries \mathcal{F}^* and \mathcal{I}^* and answers to them would not depend on our choices, they will be determined by the algorithms A and A' and the representation Rep . In this proof, we will also describe how to encode the bijection $\eta_B|_{\mathcal{F}'_B}$.

Fix a representation Rep and the retrieval algorithms A and A' that implement \mathcal{F} and \mathcal{I} using t and t' cell probes respectively.

$$\begin{aligned} \text{Rep} : \mathcal{B} &\mapsto \{0, 1\}^{w(\Upsilon+r)} \\ A : \{0, 1\}^{w(\Upsilon+r)} \times \mathcal{F} &\mapsto \mathcal{Y}_{\mathcal{F}} \\ A' : \{0, 1\}^{w(\Upsilon+r)} \times \mathcal{I} &\mapsto \mathcal{Y}_{\mathcal{I}} \end{aligned}$$

Without loss of generality, assume that $t < t'$. Fix an object $B \in \mathcal{H}$ that is incompressible in the sense of Kolmogorov [23], i.e., $K(B) = \lg \Upsilon - O(1)$. The idea of the proof is to compress B using Rep , A and A' violating the condition of its incompressibility and obtaining a contradiction.

We compress the representation $S \in \{0, 1\}^{w(\Upsilon+r)}$ of B using an iterative procedure. At each step, we remove a cell from S and add some extra information that allows us to recover (perhaps, not efficiently) the original object B . We need to make sure that the amount of this new information is smaller than w bits, enough steps can be performed so that the resulting representation is smaller than $K(B)$ bits. We now give a bird eye description of the iterative procedure, and later provide the details and analyze it.

At the k -th step, we pick a cell d_k that is used by the least number of our queries \mathcal{F} and \mathcal{I} . We delete d_k (so we call it a *deleted cell*), and protect a set of some other cells $P(d_k)$, so that they cannot be deleted in the future steps (these cells are called *protected*). The cells that are not deleted or protected after a given step are called *remaining* and denoted by \mathcal{C}_k , $\mathcal{C}_0 = \{1, 2, \dots, \Upsilon + r\}$. The procedure is performed z times such that there are at least $\Upsilon/2$ remaining cells at the final step. At the end of the procedure, we encode the sequence of the values stored in the remaining cells (from left to right) in \mathcal{R} and the sequence of positions of deleted cells in \mathcal{D} . In other words, using \mathcal{R} and \mathcal{D} , we can recover S except for positions where the cell was deleted.

We say that a cell at location l is *used* by the query q (respectively, q') if A (respectively, A') probes that location. Let $R(q)$ denote the set of remaining cells used by $q \in \mathcal{F}_B \cup \mathcal{I}_B$; and $F(l)$ (respectively, $I(l)$) be the set of queries $q \in \mathcal{F}_B$ (respectively, $q \in \mathcal{I}_B$) that use a given cell $l \in \mathcal{C}_k$. Then,

$$\sum_{l \in \mathcal{C}_k} |F(l)| = \sum_{q \in \mathcal{F}_B} |R(q)| \leq t |\mathcal{F}_B| \leq t \gamma,$$

and

$$\sum_{l \in \mathcal{C}_k} |I(l)| = \sum_{q \in \mathcal{I}_B} |R(q)| \leq t' |\mathcal{I}_B| \leq t' \gamma.$$

Therefore, there are at most $|\mathcal{C}_k|/2$ cells l that $|F(l)| \geq 2t\gamma/|\mathcal{C}_k|$, and at most $|\mathcal{C}_k|/2$ cells l that $|I(l)| \geq 2t'\gamma/|\mathcal{C}_k|$. Since $|\mathcal{C}_k| \geq |\mathcal{C}_z| \geq \Upsilon/2$, so we can find $d_k \in \mathcal{C}_k$ that is used by at most $\beta = 4t\gamma/\Upsilon$ forward and $\beta' = 4t'\gamma/\Upsilon$ inverse queries. We delete d_k on the k -th step.

Let $F'(d_k) = F(d_k) \cap \eta_B^{-1}(I(d_k))$ denote the set of forward queries q that use the cell d_k and the reciprocal $q' = \mu_B(q)$ also uses d_k , analogously define $I'(d_k)$. Reciprocal mapping η_B defines a bijection between $F'(d_k)$ and $I'(d_k)$. To encode this bijection, we enumerate the elements of $F(d_k)$ using numbers from $[\beta]$ and the elements of $I(d_k)$ using numbers from $[\beta']$. We store an array $M(d_k)$ of β elements, each is of size $\lg \beta'$ bits. Define $M(d_k)[i] = j$, if the i -th element of

$F(d_k)$ is present in $F'(d_k)$ and its reciprocal is the j -th element of $I(d_k)$, otherwise $M[i] = 0$.

Let $Q(d_k) = \eta_B(F(d_k)) \cup \eta_B^{-1}(I(d_k))$ be the set of all reciprocal queries to the queries that use d_k , and $P(d_k) = \cup_{q \in Q(d_k)} R(q) \setminus \{d_k\}$ be the set of cells that they use excluding d_k . Protecting cells in P_{d_k} guarantees: (i) if a query q uses d_k and its reciprocal q' does not, then we can retrieve the answer to q' , (ii) if both q and q' use d_k then d_k is the only deleted cell that is needed to compute q and q' . The latter property is useful as we can encode correspondence between such queries “locally” at the cell d_k . The number of protected cells is at most

$$|P(d_k)| \leq t|I(d_k)| + t'|F(d_k)| \leq t\beta' + t'\beta \leq \frac{8\gamma tt'}{\Upsilon},$$

since $Q(d_k)$ contains $|I(d_k)|$ forward and $|F(d_k)|$ inverse queries. The number of remaining cells on the next step is

$$\begin{aligned} \mathcal{C}_{k+1} &\geq |\mathcal{C}_0| - \sum_{i=1}^k |P(d_i)| - k \\ &\geq \Upsilon + r - kt'\beta - kt\beta' - k \geq \Upsilon - k(2t'\beta + 1), \end{aligned}$$

since $t'\beta = t\beta'$. It remains to make sure that $\mathcal{C}_z \geq \Upsilon/2$ by setting $z = \Upsilon/(4t'\beta + 2)$.

After z iterations, the encoding procedure stores the following information:

\mathcal{D} - the locations of all the deleted cells $\mathcal{D} = \cup_k \{d_k\}$ using at most $\lg \binom{\Upsilon+r}{z} = \lg \binom{O(\Upsilon)}{z} = z \lg(O(\Upsilon)/z) = z \lg(t'\beta) + O(z)$ bits.

\mathcal{R} - the contents of all the cells (left to right) that are not deleted using $\Upsilon + r - z$ cells;

\mathcal{M} - the arrays $M(d_k)$ for all deleted cells d_k using $z\beta \lg \beta'$ bits.

The decoding procedure starts with an uninitialized array S of size $\Upsilon + r$. It writes the values from \mathcal{R} (in left to right order) to all the locations that are not encoded in \mathcal{D} . Then, it simulates all the queries from \mathcal{F}_B and \mathcal{I}_B (the order of the simulations does not matter). Note that some of the queries will *fail* due to the fact that the contents of z cells are missing in S . The queries that do not fail form the sets \mathcal{F}_B^* or \mathcal{I}_B^* from Definition 2.1. We call a query q *recoverable* if it fails, but its reciprocal q' does not fail.

For each query q that fails, we find the first deleted cell d that the algorithm (A or A') needs to probe in order to execute q . For each cell d , we list all the forward (inverse) queries F'_d (respectively, I'_d) in lexicographical order for which d is the first failing cell. By the construction, $M(d)$ gives the reciprocal bijection

between non-recoverable forward $F'_d \subseteq F'_d$ and inverse $I'_d \subseteq I'_d$ queries. Hence all the three components in Definition 2.1 can be recovered using \mathcal{R} , \mathcal{D} , \mathcal{M} , the sequence of M_d for all $d \in \mathcal{D}$. Therefore, the object B can be recovered as well.

It remains to account for the space our new representation uses. By the assumption, the encodings of \mathcal{F}_B and \mathcal{I}_B occupy at most $1/64(\Upsilon^2 w)/(\gamma tt') = zw/4$ bits, since $z = \frac{\Upsilon}{4t'\beta} = \frac{\Upsilon^2}{16\gamma tt'}$. It follows from (3.2) that $\beta' < w/(4 \lg w)$, and hence $z\beta \lg \beta' \leq z\beta' \lg \beta' < zw/4$, so that the storage \mathcal{M} occupies at most $zw/4$ bits. It also follows from (3.2) that $\lg t' < w/4$, so that the storage \mathcal{D} occupies at most $z \lg(t'\beta) + O(z) < zw/4 + z \lg w + O(z) = zw/4 + o(zw)$. The total storage for \mathcal{D} , \mathcal{R} , and \mathcal{M} together with the encodings of \mathcal{F}_B and \mathcal{I}_B is at most $\Upsilon + r - zw/4$. Since B is incompressible, it follows that $r \geq zw/4$ (we ignored $o()$ terms for simplicity).

4 Applications of the Compression Lemma

For the three problems that we considered earlier: the PERMS problem, the TEXTSEARCH problem, and for the binary relations problem, in this section, we show the lower bound trade-offs between r , t and t' as the consequences of the compression lemma.

4.1 The PERMS problem We start with the simplest of the three problems. In the case of permutations, we do not have to encode the sets \mathcal{F}_π and \mathcal{I}_π as they do not depend on the permutation π . The following theorem is a simple consequence of Lemma 3.1.

THEOREM 4.1. *For the PERMS problem, if*

$$\max\{t, t'\} < \min \left\{ 2^{w/4}, \frac{(w \lg n)(\lg w)}{16} \right\},$$

then

$$r \geq \frac{1}{32} \left(\frac{\lg n}{w} \right)^2 \frac{n}{tt'}$$

cells.

Proof. Substitute $\Upsilon = (\lg n!)/w = (n \lg n - \Theta(n))/w$, $\gamma = n$, and use Lemma 3.1.

In the interesting case where the cell size is $w = \lg n$, we have that $r = \Omega(n/(tt'))$ for $t = O((\lg n)^2/\lg \lg n)$. In particular, if we would like to find an algorithm that performs queries π and π^{-1} in constant time, then we need linear extra space. Also, if we require that our algorithm implements *either* π or π^{-1} in constant time, then we obtain a linear lower bound that matches (up to a constant factor) the upper bound from Munro et

al. [29] that uses “back pointers”. If we require that our algorithm implements both π and π^{-1} in $\Theta(\lg n / \lg \lg n)$ time, then we get a lower bound that matches (up to time and space constant factors) the upper bound from Munro et al. [29] that uses Benes networks. Namely, in [29, Theorem 7], they use $O(n(\lg \lg n)^2 / (\lg n))$ extra redundancy bits and cell size $w = \lg n$, while our lower bound requires at least $\Omega(n(\lg \lg n)^2 / (\lg n)^2)$ extra cells.

4.2 The TEXTSEARCH problem

THEOREM 4.2. *For the TEXTSEARCH problem on texts of length L , alphabets of size σ , and patterns of length p , if the following conditions are satisfied:*

$$(4.4) \quad p \lg \sigma \leq \lg L,$$

$$(4.5) \quad \max\{t, t'\} < \min\left\{2^{w/4}, \frac{1}{16} \frac{p \lg \sigma}{\lg w}\right\},$$

$$(4.6) \quad tt' < \frac{1}{128} \frac{(p \lg \sigma)^2}{w \lg p}$$

then

$$rtt' \geq \frac{1}{32} \frac{Lp(\lg \sigma)^2}{w^2}$$

Proof. Recall that in the TEXTSEARCH problem, we are given the text T of length L on an alphabet Σ of size σ . We are required to preprocess and store it such that we can efficiently search for the j -th occurrence of a given pattern X of length p in the text. For the purposes of proving a lower bound on the size of the storage, we only restrict ourselves to the queries of the form

$$\begin{aligned} \mathcal{F}_T &= \{\text{“query access}(ip+1) \mid 0 \leq i < L/p\} \\ \mathcal{I}_T &= \{\text{“query search}(X, j) \mid \\ &\quad \text{search}_T(X, j) = ip+1\}. \end{aligned}$$

While the encoding of forward queries is trivial, the encoding of inverse queries is bit involved. For all possible patterns X of length p on the alphabet Σ , we denote J_X to be the bit vector, such that $J_X[j] = 1$ if $\text{search}_T(X, j) \equiv 1 \pmod p$, and $J_X[j] = 0$ otherwise. So that what we need to encode is the set

$$\mathcal{J} = \{(X, J_X) \mid \text{all possible patterns } X \text{ of length } p\}$$

Let us order possible patterns X in lexicographic order, trim the trailing zeroes in J_X , and concatenate the trimmed bit vectors. The resulting bit vector J has exactly L/p 1-bits in it, since there are L/p positions of the form $i \equiv 1 \pmod p$. The length of J is $L-p+1$, since there are total $L-p+1$ positions that can be answers to the search queries. Thus, we can encode this bit vector using $\lg \binom{L}{L/p}$ bits. We also need to encode the sequence of cardinalities C_X (i.e. the number of 1-bits) of J_X for

all possible patterns X . It is a sequence of σ^p numbers the sum of which is L/p , and hence can be encoded using $\lg \binom{L/p+\sigma^p}{L/p}$ bits. The original bit vectors J_X can be restored by reading J from left to right and cutting it in a greedy fashion so that the resulting bit vectors have the cardinalities from the sequence of C_X . Thus, the encoding of \mathcal{J} is at most

$$(4.7) \quad \begin{aligned} \lg \binom{L}{L/p} + \lg \binom{L/p+\sigma^p}{L/p} &\leq 2 \lg \binom{L+L/p}{L/p} \\ &\leq (2L/p) \lg(2ep), \end{aligned}$$

We assumed that the total possible number of patterns X is not too large, that is

$$\sigma^p \leq L$$

Also, we used the fact that $\lg \binom{x}{y} \leq y \lg(ex/y)$. Now we are ready to apply Lemma 3.1 with parameters

$$\Upsilon = \left\lceil \frac{L \lg \sigma}{w} \right\rceil \text{ and } \gamma = \frac{L}{p}$$

The inequality

$$\max\{t, t'\} < \frac{1}{16} \frac{\Upsilon w}{\gamma \lg w}$$

of (3.2) transforms into

$$\max\{t, t'\} < \frac{\varepsilon}{16} \frac{p \lg \sigma}{\lg w},$$

and (3.3) into

$$rtt' \geq \frac{\varepsilon}{32} Lp \left(\frac{\lg \sigma}{w}\right)^2.$$

By the precondition of the Lemma 3.1, the size of the encoding of the set \mathcal{I}_T should be at most

$$(4.8) \quad \frac{\varepsilon}{64} \frac{\Upsilon^2 w}{\gamma tt'}.$$

The sufficient condition for (4.8) is

$$\frac{2L \lg(2ep)}{p} \leq \frac{\varepsilon}{64} \frac{Lp(\lg \sigma)^2}{wt t'},$$

which is satisfied if

$$(4.9) \quad tt' < \frac{\varepsilon}{128} \frac{(p \lg \sigma)^2}{w \lg(2ep)}.$$

Combining these conditions and $\max\{t, t'\} < 2^{w/4}$, we obtain the statement of the theorem.

For the interesting special case of patterns of length $p = \lg L / \lg \sigma$ and word size $w = \lg L$, we obtain that if $tt' < (1/128)(\lg L) / \lg \lg L$, then $r \geq (1/32)(L \lg \sigma) / \lg L = \Omega(\Upsilon)$. It follows that we need at least $\Omega(\Upsilon)$ cells (i.e., linear in the information-theoretic minimum) of extra space to support both **access** and **search** queries in constant time for patterns of size $p = \lg L / \lg \sigma$ in the cell probe model with word size $\lg L$. More tradeoffs and are given in [15, Chapter 5].

4.3 The Access/Select Problem

THEOREM 4.3. *Consider the Access/Select problem with $w = \lg L$, and alphabet of size σ such that $16 \lg \lg L \leq \lg \sigma \leq \alpha \lg L$ for some constant $\alpha < 1$. If $\max\{t, t'\} < \min\{2^{w/4}, (1/16)(\lg \sigma)(\lg \lg L)\}$, then $r \geq (1/32)((\lg \sigma) / \lg L)^2 L / (tt')$.*

For the case of $\sigma = L^\alpha$, $t = 1$, and $t' = \lg \lg \sigma / \lg \lg \lg \sigma$ the lower bound matches the upper bound of Golynski et al. [16] up to constant factors.

4.4 Binary Relations

THEOREM 4.4. *Consider the binary relation problem on $m \times n$ binary matrices R of cardinality f . If parameters t, t', n, m, f satisfy*

$$\max\{t, t'\} < \min\left\{2^{w/4}, \frac{1}{16} \frac{\lg(nm/f)}{\lg w}\right\},$$

and

$$tt' < \frac{1}{128} \frac{f(\lg(nm/f))^2}{wn \lg(ef/n)},$$

then

$$rtt' \geq \frac{1}{32} \frac{f(\lg(nm/f))^2}{w^2}.$$

Proof. We assume that $m \leq n$ without loss of generality (since we can interchange the roles of the rows and columns). Recall that the sets of forward and inverse queries are as follows

$$\begin{aligned} \mathcal{F}_R &= \{\text{“RSel}(i, x)\text{”} \mid 1 \leq i \leq m, 1 \leq x \leq \text{row_nb}(i)\} \\ \mathcal{I}_R &= \{\text{“CSel}(x', j)\text{”} \mid 1 \leq j \leq n, 1 \leq x' \leq \text{col_nb}(i)\}. \end{aligned}$$

so that $|\mathcal{F}| = |\mathcal{I}| = f$. Also recall that the queries $q = \text{“query RSel}(i, x)\text{”}$ and $q' = \text{“query CSel}(x', j)\text{”}$ are reciprocal if $\text{RSel}(i, x) = j$ and $\text{CSel}(x', j) = i$, i.e. if they are selecting the same 1-bit entry in R . We showed that this problem possess the reciprocal property, so that we can apply Lemma 3.1 with parameters

$$\gamma = f \text{ and } \Upsilon = \left\lceil \frac{\lg \binom{nm}{f}}{w} \right\rceil > \frac{f}{w} \lg \frac{nm}{f}$$

To encode the sets \mathcal{F}_R and \mathcal{I}_R we can store the values of $\text{row_nb}(i)$ and $\text{col_nb}(j)$ for each row i and column j . These values of $\text{row_nb}(i)$ form a sequence of m numbers, the sum of which is f , and hence can be stored using $\lg \binom{f+n}{n}$ bits. In a similar fashion, we can store the sequence $\text{col_nb}(j)$. So the size of encoding of \mathcal{F}_R and \mathcal{I}_R is at most

$$\lg \binom{f+n}{n} + \lg \binom{f+m}{m} \leq 2 \lg \binom{f+n}{n} < 2n \lg \frac{ef}{n}$$

bits by the assumption that $m \leq n$. The precondition of Lemma 3.1 requires that this encoding occupies at most

$$\frac{\varepsilon}{64} \frac{\Upsilon^2 w}{\gamma tt'} > \frac{\varepsilon}{64} \frac{f}{w tt'} \left(\lg \frac{nm}{f} \right)^2$$

bits, so it suffices to require

$$tt' < \frac{\varepsilon}{128} \frac{f(\lg(nm/f))^2}{wn \lg(ef/n)}.$$

In particular, consider the case of matrices of size $m \times m^\alpha$ with cardinalities m^β for constants α and β such that $1 < \alpha < \beta < 1 + \alpha$. Also assume that $w = \lg m$. Hence,

$$\Upsilon = \frac{(1 + \alpha - \beta)m^\beta \lg m + O(m^\beta)}{\lg m}.$$

We can derive the following corollary.

COROLLARY 4.1. *For the binary relation problem on $m \times m^\alpha$ matrices with $f = m^\beta$, where α and β are constants such that $1 < \alpha < \beta < 1 + \alpha$. If*

$$\max\{t, t'\} < \frac{1 + \alpha - \beta}{16} \frac{\lg m}{\lg \lg m},$$

then

$$rtt' \geq \frac{1 + \alpha - \beta}{32} m^\beta = \Omega(\Upsilon).$$

In particular, for the case where the constant time operations are needed, the extra space required to support them is at least linear in Υ , i.e. $\Omega(\Upsilon)$ extra cells, it also matches the bounds stated in Figure 1 for $t = 1$ and $t' = \lg \lg \rho / \lg \lg \lg \rho$.

5 Acknowledgments

Author is grateful to his supervisors Ian Munro and Prabhakar Ragde, and to his examining committee Faith Fich, Jeremy Barbay, Timothy Chan and Ashwin Nayak for the valuable comments on the version of these results that appeared as his PhD dissertation. Author is also thankful to the anonymous referees that greatly helped him to improve the presentation in this extended abstract.

References

- [1] Miklós Ajtai. A lower bound for finding predecessors in yao's cell probe model. *Combinatorica*, 8(3):235–247, 1988.
- [2] Jérémy Barbay, Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Adaptive searching in succinctly encoded binary relations and tree-structured documents. In *Combinatorial Pattern Matching*, pages 24–35, 2006.
- [3] Jérémy Barbay, Meng He, J. Ian Munro, and S. Srinivasa Rao. Succinct indexes for strings, binary relations and multi-labeled trees. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 680–689, 2007.
- [4] Paul Beame and Faith Ellen. Optimal bounds for the predecessor problem and related problems. *Journal of Computer and System Sciences*, 65(1):38–72, 2002.
- [5] Erik D. Demaine and Alejandro López-Ortiz. A linear lower bound on index size for text retrieval. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 289–294, 2001.
- [6] Erik D. Demaine and Alejandro López-Ortiz. A linear lower bound on index size for text retrieval. *Journal of Algorithms*, 48(1):2–15, 2003.
- [7] Martin Farach-Colton. Optimal suffix tree construction with large alphabets. In *IEEE Symposium on Foundations of Computer Science*, pages 137–143, 1997.
- [8] Arash Farzan and Ian Munro. Succinct representations of arbitrary graphs (accepted). In *ESA*, 2008.
- [9] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 390–398, 2000.
- [10] Paolo Ferragina and Giovanni Manzini. An experimental study of an opportunistic index. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 269–278, 2001.
- [11] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. An alphabet-friendly FM-index. In *Proceedings of the 11th Symposium on String Processing and Information Retrieval*, pages 150–160, 2004.
- [12] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In *International Colloquium on Automata, Languages and Programming*, pages 332–344, 2003.
- [13] Rosario Gennaro and Luca Trevisan. Lower bounds on the efficiency of generic cryptographic constructions. In *IEEE Symposium on Foundations of Computer Science*, pages 305–313, 2000.
- [14] Robert Giegerich and Stefan Kurtz. From ukkonen to mcreight and weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997.
- [15] Alexander Golynski. *Upper and Lower Bounds for Text Indexing Data Structures*. PhD thesis, University of Waterloo, <http://www.cs.uwaterloo.ca/~imunro/Golynski.pdf>, 2007.
- [16] Alexander Golynski, J. Ian Munro, and S. Srinivasa Rao. Rank/select operations on large alphabets: a tool for text indexing. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 368–373, 2006.
- [17] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 841–850, 2003.
- [18] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *ACM Symposium on Theory of Computing*, pages 397–406, 2000.
- [19] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal of Computation*, 35(2):378–407, 2005.
- [20] Dan Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. USA: Cambridge University Press, 1997.
- [21] M.E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26:401–405, 1980.
- [22] Juha Kärkkäinen and Peter Sanders. Simple linear work suffix array construction. In *International Colloquium on Automata, Languages and Programming*, pages 943–955, 2003.
- [23] P. Vitányi M. Li. *An Introduction to Kolmogorov Complexity and its Applications, 2nd edition*. Springer-Verlag, New York, 1997.
- [24] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.
- [25] Edward M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.
- [26] Peter Bro Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Symposium on Theory of Computing*, pages 625–634, 1994.
- [27] Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.
- [28] M.L. Minsky and S.A. Papert. *Perceptrons*. Cambridge, MA: MIT Press, 1969.
- [29] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations. In *International Colloquium on Automata, Languages and Programming*, pages 345–356, 2003.
- [30] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1):article 2, 2007.
- [31] Mihai Pătraşcu. Personal communication.
- [32] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Symposium on Theory of Computing*, pages 232–240, 2006.
- [33] Kunihiko Sadakane. Compressed text databases with efficient query algorithms based on the compressed

- suffix array. In *International Conference on Algorithms and Computation*, pages 410–421, 2000.
- [34] Kunihiko Sadakane. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 225–232, 2002.
- [35] Pranab Sen. Lower bounds for predecessor searching in the cell probe model. In *IEEE Conference on Computational Complexity*, pages 73–83, 2003.
- [36] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.
- [37] P. Weiner. Linear pattern matching algorithms. In *IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.
- [38] Andrew Chi-Chih Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.
- [39] Andrew Chi-Chih Yao. Coherent functions and program checkers (extended abstract). In *ACM Symposium on Theory of Computing*, pages 84–94, 1990.