CMSC858F: Algorithmic Lower Bounds: Fun with Hardness Proofs
"Hardness made easy"
Fall 2014
Course Introduction

Instructor: Mohammad T. Hajiaghayi
Scribe: Ahmed Abdelkader

September 2, 2014

1 Overview

In this lecture, we review the course description and motivate the topics to be covered.

2 What is this class?

- A practical guide to formally prove the hardness of computational problems. We also consider the common techniques to prove hardness in different settings.
- Not a course in Complexity Theory, but we use complexity results and refer to needed materials.
- Another name for this class can be "Assumptions & Reductions". We learn about different hard problems, or at least believed to be very hard. Then, assuming the hardness of these problems to hold, we prove other problems would be equally hard by showing that a solution to the new problem can easily be used to find a solution to the hard problem.
- *Anti*-algorithmic perspective: Instead of learning how to design algorithms, we study how to prove one cannot design fast algorithms in a certain model of computation. This could mean you need to change your model or change the problem formulation. Still, some backgroud in Algorithms is needed. In a way, a reduction that maps one problem to another is effectively an algorithm.

- Lots of researchers design algorithms for different settings such as counting algorithms, approximation algorithms, fixed-parameter algorithms, streaming algorithms, parallel algorithms, algorithmic game theory, geometric problems, online algorithms, puzzle algorithms. However, sometimes we cannot design algorithms of certain efficiency in terms of time, space or approximation. In this course, we want to understand why there exists such limits.
- We believe every cs grad student, especially in theory, should have a basic knowledge about hardness of algorithms at the level of this course.
- In short, while in a typical Advanced Algorithms course we learn about the design of algorithms for different settings, in this course we learn why we cannot design algorithms with certain guarantees in different settings.

3 Why take this class?

- Know the limits in algorithm design.
- Master techniques for proving hardness, in particular:
  - Key problems and assumptions.
  - Key reductions: proof styles and gadgets.
- Cool connections between problems. Most problems are equivalent to each others and we want to learn how to prove it
- Fun problems to solve (even some puzzle problems) leading to potentially publishable papers.

4 Prerequisites

- Algorithms.
- Asymptotics.
- Combinatorics.
- Moderate knowledge of complexity.
- Passing 451 or an equivalent course, but not much background is assumed.
- Having passed a graduate level course in algorithms is a plus.

5 Requirements

- 3-4 problem sets.
- Scribe notes.
- Class participation (especially, attend lectures).
- One in-class exam.
- Research-based projects (survey and new theory).
- Class presentation.

6 Topics

- NP-completeness (3SAT, 3-partition, hamiltonicity, geometry, 3-coloring).
- PSPACE, EXPTIME, EXPSPACE.
- Inapproximability (PCP, APX, set-cover, label cover, UGC, indep. set, ...).
- Fixed-parameter Intractability (W, clique, ...)
- 3SUM (towards n^2) and all pairs shortest paths (towards n^3).
- Counting (#P) and uniqueness (ASP).
- Algorithmic game theory (PPAD).
- Existential theory of reals, undecidability (if time permits).
- Lower bounds for streaming algorithms (indexing, set disjointedness, ...).
- Lower bounds for online algorithms (matching, set cover, k-server, ...).

We often focus on graph-like problems in this course.

References

[1] Computers and Intractability A Guide to the Theory of NP-Completeness: book by Michael R. Garey and David S. Johnson.

[2] Johnson's followup NP-completeness Columns.

[3] Games, Puzzles, & Computation: book by Robert A. Hearn and Erik D. Demaine.

[4] Complexity Zoo.

[5] A compendium of NP optimization problems.