CMSC858F: Algorithmic Lower Bounds: Fun with Hardness Proofs
Fall 2014
A Crash Course in Complexity Theory

Instructor: Mohammad T. Hajiaghayi
Scribe: Ahmed Abdelkader

September 4, 2014

1 Overview

In a general sense, complexity theory studies the basic question: how hard is this problem? Naturally, this leads to an elaborate classification of computational problems into easy, intermediate, hard, harder or even undecidable. For further information, the reader is referred to any of the standard textbooks, e.g. [1, 2].

Typically, the hardness of a problem is measured by the amout of work needed to solve the worst-case, i.e. hardest, instances of that problem. We first need to specify a model of computation to formally describe the process of executing an algorithm to find a solution. Turing machines are one such convenient model of computation, and can actually serve as a representative to *almost* all other models. Then, the amount of work, i.e. the number of basic operations, is expressed as a function in the input size, where the input is a string encoding of the instance to be solved. Asymptotic notation, e.g. $\{O, \Theta, \Omega\}$, greatly simplifies the analysis by allowing one to overlook some constants and ignore lower order terms.

2 The Scale of Computational Difficulty

In order of increasing difficulty, we may classify problems into three nested classes as shown in Figure 1 and defined informally as:

- P = {problems solvable in polynomial time, i.e. n^c}.
- EXP = {problems solvable in exponential time, i.e. 2^{n^c}}.
- R = {problems solvable in finite time}. (R for recursive [3, 4]).

Scribe: Ahmed Abdelkader
Lecture 2 **Date:** 09/04/2014

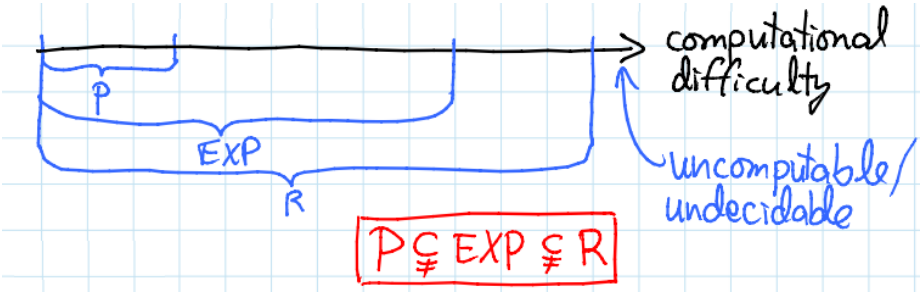

Figure 1: Scale of computational difficulty (Image courtesy of Erik Demaine).

2.1 Example Problems

- Negative-weight cycle detection $\in$ P.
  - Use the Bellman-Ford algorithm.
- $n \times n$ Chess $\in$ EXP but $\notin$ P.
  - Who wins starting from a given board configuration?
- TETRIS $\in$ EXP but unknown whether $\in$ P.
  - Survive an incoming sequence of pieces starting with a given board.
- Halting problem $\notin$ R.
  - Given an arbitrary pair of a computer program, or equivalently a Turing machine, and an input string, determine whether the program will terminate (halt) or run forever.
  - Being a decision problem that cannot be solved by a Turing machine (or any humanly conceivable computer), we say it is undecidable.

3 Nondeterminism and the Complexity Class NP

Unless stated otherwise, we consider decision problems, i.e. problems that can be stated as a YES/NO question.

- NP = {decision problems solvable in polytime via a *lucky algorithm*}.

Lucky means it can make lucky guesses, which turn out to be always *right*, without having to try all possible computation paths, i.e. no brute-force search. The sequence of guesses is also called a *certificate*. In addition, guesses may be thought of as binary decisions or 0/1 bits.

The abbreviation NP stands for nondeterministic polynomial-time. This nondeterministic model of computation means the algorithm makes guesses and

then outputs the decision. The guesses are guaranteed to lead to a `YES` outcome if possible, and `NO` otherwise.

The same class of problems can be defined equivalently as:

- NP = {decision problems with solutions that can be *verified* in polytime}.

This means that when the answer is `YES`, the algorithm can prove it by providing evidence in form of a certificate that can be checked by a polytime algorithm to verify it is actually correct. Figure 2 shows how the scale of difficulty looks like now.

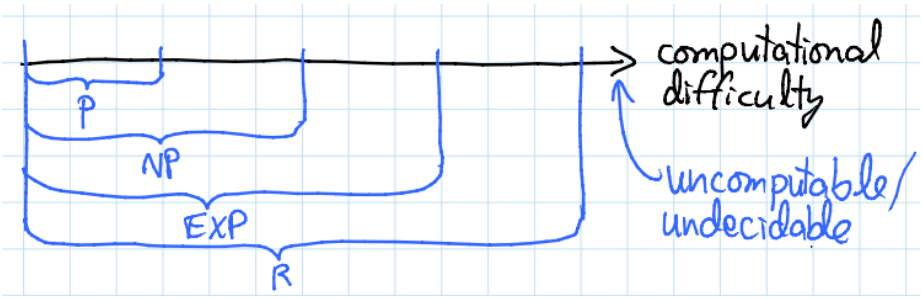

Figure 2: P vs NP (Image courtesy of Erik Demaine).

3.1 Example: TETRIS ∈ NP

- Below is an outline of a nondeterministic algorithm:
  - Guess a *lucky* sequence of moves.
  - Did you survive the input sequence?
- Proof of `YES`: the list of moves to make (rules of Tetris are easy).

In typical Tetris, $n \times m = 20 \times 10$, but for constant n the problem is in P and a dynamic programming approach can be used.

3.2 P vs NP

One of the Millennium Problems established by the Clay Mathematics Institute for a $1 Million prize.

- P ≠ NP is a big conjecture, widely believed to be true.
- Consequences?
  - Cannot engineer luck.
  - Generating (proofs of) solutions can be harder than checking them.
  - See [7] for a more elaborate exposition.

3.3 CoNP

Negated versions of the YES/NO questions in NP, loosely defined as:

- CoNP = {negations of problems ∈ NP}.
- CoNP = {decision problems with short verifiable proofs of NO answer}.

So far, everything in $\mathsf{NP} \cap \mathsf{CoNP} \in \mathsf{P}$.

4 X-hard and X-Complete

Here, X can be any complexity class, e.g. NP, EXP, etc. We give the informal definitions:

- $\mathsf{X} - \mathtt{hard}$ = "as hard as" every problem in X.
  - More on this below.
- $\mathsf{X} - \mathtt{complete}$ = X-hard ∩ X.
  - Sometimes X-easy is used as ∈ X.

4.1 Examples

- TETRIS is NP-complete [5].
  - If $\mathsf{P} \neq \mathsf{NP}$, then $\mathsf{TETRIS} \in \mathsf{NP} - \mathsf{P}$.
- CHESS is EXP-complete, which also implies $\notin \mathsf{P}$.
  - $\mathsf{CHESS} \in \mathsf{EXP} - \mathsf{NP}$ if $\mathsf{NP} \neq \mathsf{EXP}$ (which is also an open problem).

Figure 3 shows how complete problems lie at the boundaries between classes and how hard problems fill the gaps. It is always tempting to push some boundaries and bridge some gaps.

5 Space Complexity and the Class PSPACE

- PSPACE = {problems solvable in polynomial space}.
  - ⊆ EXP: a polynomially bounded space can only be in exponentially many states i.e. algorithm may not execute more than an exponential number of steps.
  - ⊇NP: simulate all paths of computation of polynomially bounded length (in search for the lucky path) and return YES if any succeeds.
  - open question whether either of the two inclusions above is strict.
- Example: Rush Hour is PSPACE-complete [8].

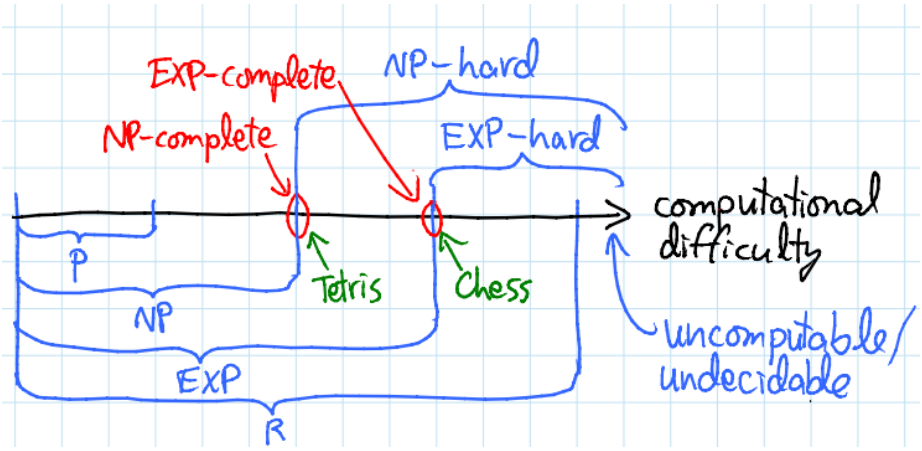

Figure 3: X-hard and $\mathsf{X}-\mathsf{complete}$ (Image courtesy of Erik Demaine).

– $\notin$ P if $\mathsf{P} \neq \mathsf{NP}$ (since $\mathsf{NP} \subseteq \mathsf{PSPACE}$) or $\mathsf{NP} \neq \mathsf{PSPACE}$ (since $\mathsf{P} \subseteq \mathsf{NP}$).

Figure 4 shows the scale of difficulty after introducing space complexity classes. Observe the inclusions and recall that $\mathsf{PSPACE} = \mathsf{NSPACE}$.

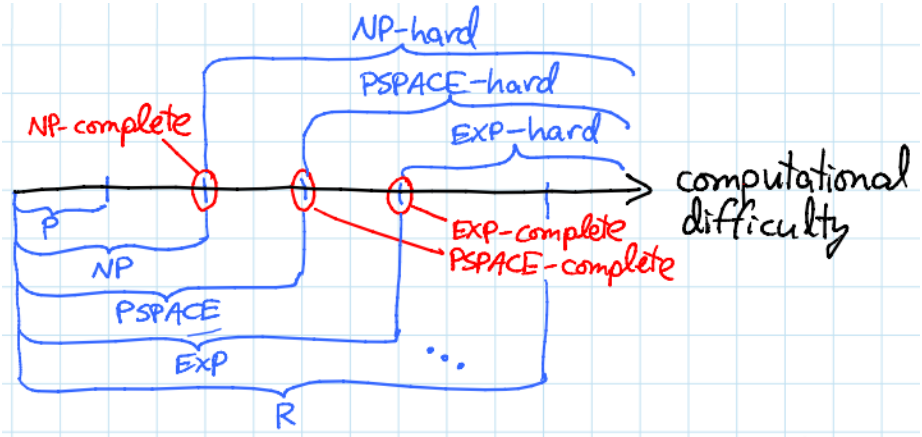

Figure 4: Space complexity classes (Image courtesy of Erik Demaine).

6 Beyond Exponential

We mention few more complexity classes, just for the sake of doing a more thourough review.

- $\mathsf{EXP}(\mathsf{TIME}) \subseteq \mathsf{EXPSPACE} \subseteq 2\mathsf{EXP}(\mathsf{TIME}) \subseteq 2\mathsf{EXPSPACE} \subseteq \ldots$
  - 2EXP: double exponential $2^{2^{n^c}}$, etc.
- $\mathsf{L} = \mathsf{LOGSPACE}$
  - Only $O(\log n)$ bits of space.
- Hierarchy theorems for both time and space classes, e.g.:
  - $\mathsf{EXP} \subsetneq 2\mathsf{EXP} \subsetneq \ldots$
  - $\mathsf{L} \subsetneq \mathsf{PSPACE} \subsetneq \mathsf{EXPSPACE} \subsetneq 2\mathsf{EXPSPACE} \subsetneq \ldots$
  - For more, refer to Chapters 3 and 4 in [1].
- Nondeterministic space
  - $\mathsf{NSPACE} = \mathsf{PSPACE}$ (Savitch's theorem) [9]
    - In general, space bound squares.
    - In a sence, if you can check in the validity of a solution in PSPACE then you can run over all solutions as well in PSPACE.
  - NEXP, N2EXP, ...: analogs of NP.
- Intermediate problems:
  - e.g. factorization and graph isomorphism.
  - Belong to the intersection of NP and CoNP but not known to be in P so far.
  - Showing $\mathsf{NP} \cap \mathsf{CoNP} = \mathsf{P}$ is equivalent to proving $\mathsf{P} \neq \mathsf{NP}$.

7 Defining "as hard as" via Reductions

Almost all hardness proofs are by reduction <u>from</u> known hard problem <u>to</u> your problem. Typically, a reduction is a polytime algorithm to convert an instance of A to an instance of B, such that the `YES`/`NO` solution to A yields a solution to B. The existence of the reduction has the following consequenes:

- If one can solve B, then one can also solve A. $\mathsf{B} \in \mathsf{P} \implies \mathsf{A} \in \mathsf{P}$, $\mathsf{B} \in \mathsf{NP} \implies \mathsf{A} \in \mathsf{NP}$, etc.
- Intuitively, we can say that B is "at least as hard as A".
- Equivalently, A may be regarded as a special case of B.
- This is a "one-call" reduction (Karp) [10].
- May also allow "multi-call" reduction (Turing) [11].
  - Solve A using an oracle that solves B.
  - Does not help much for problems we consider in this class.

7.1 Examples from Algorithms

- Unweighted shortest paths → weighted (hint: $w = 1$).
- Min-product path → min-sum path (hint: use logs).
- Longest path → shortest path (hint: negate).
- Min-weight k-step path → min weight path (hint: create k copies of the graph and link to adjacent nodes in next layer only).

8 Examples of Hardness Proofs

8.1 3SAT for NP-completeness

An instance of 3SAT is composed of n variables and m clauses in Conjunctive Normal Form (CNF), such that each clause hs a disjunction of 3 literals, where each literal is either a variable or a negation of a variable. An instance of 3SAT would look like this:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge \dots$$

3SAT asks to determine whether the boolean formula can be satisfied using an assignment of truth values to each of the n variables.

Naturally, many problems can be encoded as satisfiability problems. For example, a game like Super Mario Bros is NP-complete via a reduction from 3SAT [12].

8.2 Nondeterministic Constraint Logic PSPACE-completeness

Nondeterministic Constraint Logic (NCL) is a powerful technique for proving the hardness of games and puzzles e.g. showing Rush Hour is PSPACE-complete [8].

In particular, the following edge reversal problem [13] captures a nondeterministic model of computation which was used to prove the hardness of more games: Given a directed graph with edge weights $\in \{1, 2\}$ and regular degree 3, find a sequence of edge reversals to reverse a target edge, while at all times maintaining total in-weight ≥ 2 at each vertex.

Note that it is easy to to see why this problem is PSPACE-complete. Since PSPACE = NSPACE, by Savitch's theorem, we only need to show that we can verify a given solution in PSPACE. But, as we only maintain the current state, then this is actually feasible.

References

[1] Arora, Sanjeev, and Boaz Barak. Computational complexity: a modern approach. Cambridge University Press, 2009.

[2] Papadimitriou, Christos H. Computational complexity. John Wiley and Sons Ltd., 2003.

[3] Turing, Alan Mathison. "On computable numbers, with an application to the Entscheidungsproblem." J. of Math 58 (1936): 345-363.

[4] Church, Alonzo. "The calculi of lambda-conversion", Volume 6 of Annals of Mathematics Studies (1941).

[5] Breukelaar, R., Demaine, E. D., Hohenberger, S., Hoogeboom, H. J., Kosters, W. A., & Liben-Nowell, D. (2004). Tetris is hard, even to approximate. International Journal of Computational Geometry & Applications, 14(01n02), 41-68.

[6] Millennium Problems, Clay Mathematics Institute.

[7] Aaronson, Scott. "Why philosophers should care about computational complexity." In Computability: Gdel, Turing, Church, and beyond (eds. 2012).

[8] Flake, Gary William, and Eric B. Baum. "Rush Hour is PSPACE-complete, or Why you should generously tip parking lot attendants." Theoretical Computer Science 270.1 (2002): 895-911.

[9] Savitch, Walter J. "Relationships between nondeterministic and deterministic tape complexities." Journal of computer and system sciences 4.2 (1970): 177-192.

[10] Karp, Richard M. Reducibility among combinatorial problems. Springer US, 1972.

[11] Turing, Alan Mathison. "Systems of logic based on ordinals." Proceedings of the London Mathematical Society 2.1 (1939): 161-228.

[12] Aloupis, G., Demaine, E. D., Guo, A., & Viglietta, G. (2012). Classic Nintendo games are NP-hard. arXiv preprint arXiv:1203.1895.

[13] Demaine, Erik D., and Robert A. Hearn. "Constraint logic: A uniform framework for modeling computation as games." Computational Complexity, 2008. CCC'08. 23rd Annual IEEE Conference on. IEEE, 2008.