Class 1 Scribe Notes	Notetaker: Adam Hesterberg
6.890	
Instructor: Erik Demaine	*2014-09-04*

1 Overview

1.1 Administrivia

1. **Title:** Algorithmic Lower Bounds: Fun with Hardness Proofs
2. **Professor:** Erik Demaine
3. **TAs:** Sarah Eisenstadt and Jayson Lynch
4. **Website:** http://courses.csail.mit.edu/6.890/fall14/
5. **Class:** TR 3:35pm-4:55pm in 2-105.
6. **Open Problem Session:** Once/week, time and location TBA.
7. **Textbook:** None, but recommended are Garey and Johnson's *Computers and Intractability: A Guide tothe Theory of NP-Completeness* and Hearn and Demaine's *Games, Puzzles, and Computation*, available online at MIT.
8. **Requirements:**
   (a) Attend lectures.
   (b) Scribe notes (like these) for one or two lectures.
   (c) Do problem sets, due roughly every other week.
   (d) Final project and presentation—e.g. something theoretical, or reading, or implementation, or Wikipedia editing, or art.
9. **Background:** No background in complexity required, but some background in algorithms is.

1.2 Goals

1. Get experience in techniques for proving problems hard. This is not a complexity class; more practical.
2. Know what to avoid in designing algorithms—save time by not trying approaches that would equally well solve problems that are known to be hard.
3. Have fun with games like Mario and Tetris, and maybe get publishable papers out of it.

1.3 Topics

(These are all on http://courses.csail.mit.edu/6.890/fall14/.)

1. NP-completeness
   (a) 3SAT and all its variations
   (b) 3-partition
   (c) Hamiltonicity

   (d) 2D/3D geometric problems
2. PSPACE, EXPTIME, EXPSPACE
3. Games, Puzzles, and Computation
   (a) Tetris
   (b) Nintendo games: Super Mario Bros., The Legend of Zelda, Metroid, Pokmon, ...
   (c) Sliding blocks and Rush Hour
   (d) Constraint Logic
   (e) Sudoku and other Nikoli pencil-and-paper games
   (f) Chess, Go, Othello, and other board games
4. Inapproximability
   (a) PCP theorem and OPT-preserving reductions
   (b) APX-hardness (e.g., Vertex Cover)
   (c) Set-cover hardness
   (d) Group Steiner tree
   (e) k-dense subgraph
   (f) Label cover and Unique Games Conjecture (UGC)
   (g) Independent set
5. Fixed-parameter intractability
   (a) Parameter-preserving reductions
   (b) W hierarchy
   (c) Clique-hardness
6. 3SUM-hardness (n^2 and the Exponential Time Hypothesis)
7. Counting problems (#P)
8. Solution uniqueness (ASP)
9. Game theory (PPAD)
10. Existential theory of the reals
11. Undecidability

2 Computational Complexity Crash Course

Definition 2.1. P is the set of problems that can be solved in time polynomial in the size n of the problem instance, i.e. $n^{O(1)}$. (There may be subtleties in what counts as the "problem instance", to be discussed later.)

Definition 2.2. EXP is the set of problems that can be solved in time exponential in the size n of the problem instance, i.e. $2^{n^{O(1)}}$. For instance, $n \times n$ chess is in EXP, but not P. Also, Tetris with foresight (of all future pieces) is in EXP, but it's unknown whether it's in P.

Definition 2.3. R is the set of problems that can be solved in finite time. The halting problem (given an algorithm, does it terminate in finite time?) is not in R; in fact, "most" decision problems are not in R because there are countably many algorithms but uncountably many problems.

Definition 2.4. NP is the set of problems that can be solved in time polynomial in the size n of the problem instance, i.e. $n^{O(1)}$, via a "lucky" algorithm. Specifically, the model of nondeterminism (the N in NP) allows the algorithm to make a series of guesses and eventually say YES or NO, but the guesses are guaranteed to lead to a YES outcome if possible. Alternately, NP is the set of decision problems with solutions that can be "checked" in polynomial time.

Example: Tetris $\in$ NP: given a sequence of pieces, just say where they should each go, which can be quickly checked. Alternately, at every second, guess whether to press left, right, down, rotate, or wait.

Definition 2.5. CoNP is like NP, but flip YES and NO.

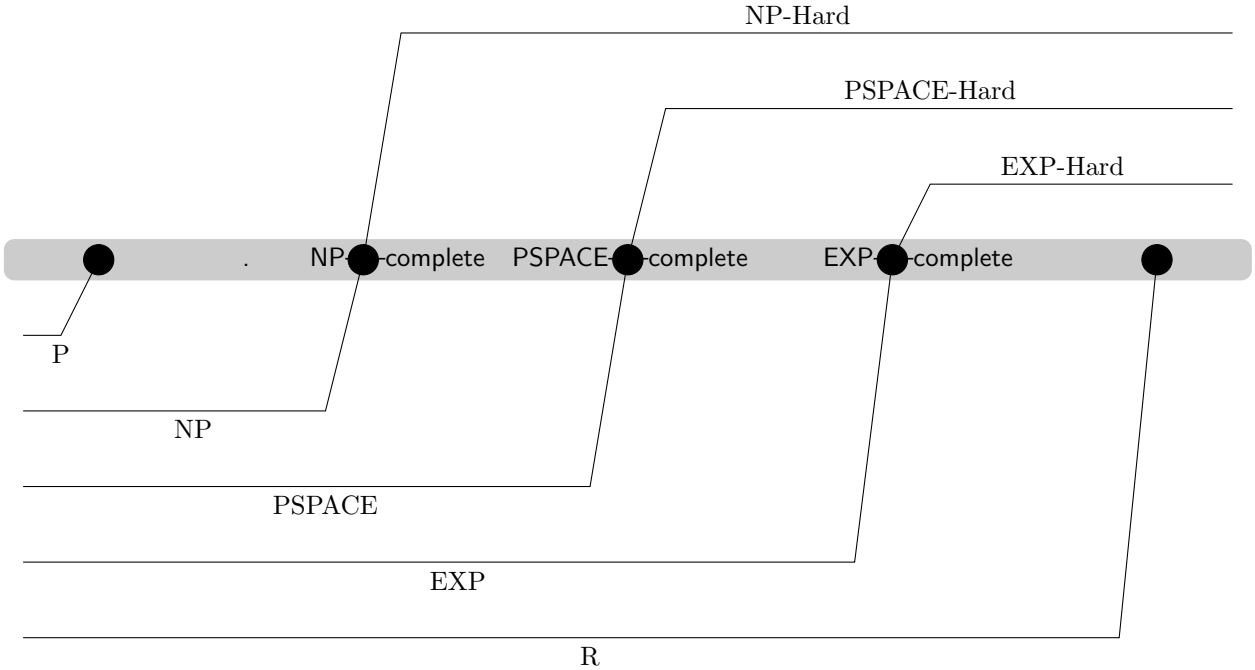
Definition 2.6. PSPACE is the set of problems solvable in polynomial space (memory).

In this class, we'll usually deal with P and EXP.

It's known that $P \subseteq NP \subseteq PSPACE \subseteq EXP \subseteq R$.

If X is a complexity class, we say a problem is **X-hard** iff it's at least "as hard as" every problem in X.

If X is a complexity class, we say a problem is **X-complete** iff it's X-hard and in X, where X is a complexity class, like EXP.



Example: Rush Hour (see section 4) is *PSPACE-complete*. It's in NPSPACE (which equals PSPACE by Savitch's Theorem[1]) because keeping track of the state of the board takes only polynomially much space, and an algorithm can just guess the correct moves.

A *reduction* from A to B is a polynomial-time algorithm to convert an instance of A to an instance of B that takes solutions of A to solutions of B. So, if we have an algorithm to solve B, we can solve A by converting to B and solving B, and we say B is *as hard as* A iff there's a reduction from A to B.[2]

To prove a problem B X-hard, we reduce from a problem A known to be X-hard to B.

[1] http://en.wikipedia.org/wiki/Savitch's_theorem

[2] This is called a "one-call" or "Karp" polynomial-time reduction. One could imagine reductions allowing something other than polynomial time, or allowing multiple calls to B, but in this class we'll usually not use them.

3 Mario

We'll prove that Super Mario Brothers 1[3] is NP-hard[4]. The problem we'll reduce from is 3-SAT ("3-SATisfiability"): given a list of *clauses* like

1. x_5 OR x_3 OR (NOT x_1)
2. x_7 OR (NOT x_2) OR (NOT x_5)

   ⋮

where each clause consists of three *literals*, where each literal is either a *variable* or its negation, is there an assignment of TRUE or FALSE to each variable that makes every clause TRUE? 3-SAT is known to be NP-complete.

Given an instance of 3-SAT, which has some variables, literals, and clauses, we'll make *gadgets* to represent them in Mario. A variables gadget is a place where you have to fall in one direction or the other, which we think of as corresponding to setting the variable TRUE or FALSE. A clause gadget is a set of fire bars to run through, just after three boxes in the floor containing invincibility stars that last long enough to let you run through the fire bars. Each box containing an invincibility star is breakable from the gadget for exactly one literal in the clause; that is, you can release an invincibility star for a clause gadget if and only if in at least one of that clause's variables' gadgets you made the satisfying choice.

The gadgets are arranged so that you first set each variable (and can pop out all the corresponding stars), then run through the clause gadgets in order, which is possible if and only if one literal from each of them was satisfied, that is, if and only if there's a satisfying assignment to the original 3-SAT problem.

The graph of paths between the variable gadgets and clause gadgets isn't planar, so there's also a crossover gadget for the paths, guaranteeing that when entered from the left it can only be exited from the right, and when entered from the bottom it can only be exited from the top. Note that the crossover gadget breaks if you go through it both ways, but that's ok because we only care about reachability.

4 Rush Hour

We'll prove that Rush Hour[5] is PSPACE-hard[6]. The problem we'll reduce from is *Constraint Logic*: You're given a graph with red edges (with weight 1) and blue edges (with weight 2), and the constraint that every vertex must have at least 2 weight worth of edges incoming. A legal move is to reverse an edge, preserving that constraint. One can construct things like AND and OR gates. For instance, a vertex with three blue edges, two of them thought of as inputs and one of them thought of as an output, is like an OR gate: the output can point out (which we think of as TRUE) iff the first input OR the second input points in (which we think of as TRUE). A problem instance gives a graph and asks whether you can reverse a given edge.

We can construct AND gadgets and protected-OR gadgets (in which we guarantee that not both of the inputs will be TRUE, which we have to guarantee because the gadget would fall apart otherwise) in Rush Hour, so Rush Hour is PSPACE-hard.

[3] http://en.wikipedia.org/wiki/Super_Mario_Bros
[4] Proof from http://erikdemaine.org/papers/Nintendo_FUN2014/.
[5] http://www.thinkfun.com/mathcounts/play-rush-hour
[6] Proof from http://erikdemaine.org/papers/NCL_TCS/