Algorithms Notes

Simon Xiang

Lecture notes for the Spring 2022 section of Algorithms and Complexity (CS 331) at UT Austin, taught by Dr. Price. These notes were taken live in class (and so they may contain many errors). Source files: https://git.simonxiang.xyz/math_notes/files.html

Contents

1 Complexity theory

We have seen lots of algorithms to solve many problems. Can we classify problems by difficulty? What can't we solve? The answer to this question "what can't we solve" is almost everything. An informal statement: take a known solvable problem, and tweak it, the result is probably not solvable.

Example 1.1. We can solve shortest path. What about the *longest* simple path. Nope. We can solve minimum spanning tree, but what about **minimum Steiner tree**, just connecting some subset $S \subseteq V$. We don't know how to solve it in polynomial time.

We have talked about a min (s,t) cut, what about a max (s,t) cut? We don't know. In flows, we talked about max (s,t) flow. What about multi-commodity flow? Nope, even with just two commodities. We can solve interval packing, which has job packing as an application—what about preferences? We don't know.

We can say *something* however about these problems, which is the theory of NP completeness. We classify problems into "complexity classes". Let n be the size of the input in bits, and only consider **decision problems** (the answer is yes or no).

Example 1.2. Shortest path is an optimization problem. Let (G,s,t,k). To rephrase this as a decision problem, say "does there exists a (s,t) path in G of length $\leq k$". Then try different ks on some binary search to find the smallest one. The number of bits n wrt V,E,s,t,k,w (where w represents edge weights) is $V \log V$ for vertices, $E \log V$ for edges, $2 \log V$ for s,t, $E \cdot w$ for edge weights, $w + \log E$ (since $k \leq E \cdot 2^w$). Summing this up, we get $n = O(E \log V + EW)$.

Complexity theorists care about classes. The class P contains problems solvable in **polynomial time**, $n^{O(1)}$. This includes most problems we covered in class. The class NP contains problems that a solution can be *verified* in polynomial time. A prototypical example is the traveling salesman problem (cycle visiting all cities exactly once). Formally:

Definition 1.1 (NP)**.** There exists a polynomial time verifier A such that **completeness** holds: for all YES instances x, there exists a proof $y \in \{0,1\}^{\text{Poly}(n)}$ such that $A(x,y) = \text{YES}$. Another condition **soundness** must hoold: for all NO instances x, there must not exist a proof $y \in \{0,1\}^{\text{Poly}(n)}$ such that $A(x,y) = \text{YES}$.

We have $P \subseteq NP$. There are some problems not in NP, for example, *how many* longest paths are there? Or, does white win this chess position? The final problem is the **halting problem**, which is not even computable.

We can say many problems are **equally hard**: they are all in P or none in P (NP-completeness). We do this by **reductions**. To show that problem A is harder than B (say TSP), tragically my laptop ran out of battery at this point in time

2 Reductions

Dr. Price forgot his notes at home today, so the lecture will be slightly scatterbrained. Recall: P consists of problems solvable in polynomial, NP consists of verifiable problems in polynomial time, NP-hard problems once solved can solve all NP problems. Today we show how to show problems are NP-hard, hence implying they are probably not in P.

Cook's Theorem. *Circuit SAT is NP-complete, i.e., CSAT is both in NP and NP-hard.*

A description of CSAT is as follows: "Does there exists an input such that a given circuit outputs 1?"[1] We can see that CSAT is in NP because we just plug it in. Furthermore, CSAT reduces to SAT, which consists of satisfiable

[1] Here circuit refers to set of logic gates, a Boolean circuit.

boolean *formulas*. It is easy to formulate a SAT formula as a SAT circuit, but the other way is not as easy. The idea is that you make a new variable for each wire. The formula is "3-CNF" (3-conjunction normal form), the intersection of unions of three clauses at a time. So CSAT, SAT, and 3SAT are all NP-complete.

Consider the problem max independent set, where given a graph G, an *independent set* I is a subset of V such that no edge $e \in E$ has $|e \cap S| = 2$. The max independent set asks if there exists an independent set S of size $|S| \geq k$. The claim is that max independent set is NP complete, or that it in NP and is NP-hard.We do this by reducing from 3SAT. Recall that reducing from X (3SAT) to Y (max independent set) means that we can solve X (3SAT) from Y (max independent set), or Y (max independent set) is no harder than X (3SAT, hard). How do we do this?

(1) Construct a transformation from an *arbitrary* X instance x to a *special* Y instance y.

(2) If x is a YES instance, then y is a YES instance.

(3) If y is a YES instance, then x is a YES instance.

Given a 3SAT instance $(a \cup b \cup c) \cap (b \cup \overline{c} \cup \overline{a}) \cap (a \cup b \cup d) \cap (a \cup \overline{c} \cup d)$, this is YES because set a and b to be true. Make a graph with vertices as above, grouped into threes. Edges just form some strongly connected components (formally called a "clause gadget"). Now we need some variable gadgets, expressing the fact that each triple relates to each other. We do this by drawing an edge from each instance of a to each instance of $\overline{a}$, b to $\overline{b}$, etc.

The claim is that if 3SAT is satisfiable, then the corresponding graph has an independent set of size greather than or equal to the number of clauses. 3SAT being satisfiable means that there exists an assignment of x_i to $\{0, 1\}$ such that each clause has greater than or equal to 1 satisfied variable. For our specific instance, our assignment is $a = 1, b = 1, d = 1, c = 0$ (one for each clause). Take S to be one of those satisfied variable vertices per clause gadget, where $|S| = \#$ of clauses, then S is independent. So if 3SAT is satisfiable, there exists a large independent set.

Now we need to show the other direction, that is, if our independent set instance is a YES, then so is the 3SAT instance. First observe than in each clause, S has exactly one vertex for each clause gadget. Furthermore, S never has both $x_i$and $\overline{x}_i$ labeled vertices. Set

$$x_i = \begin{cases} 1 & \text{if any } x_i \text{ vertex} \in S \\ 0 & \text{if any } \overline{x}_i \text{ vertex} \in S \\ 0 & \text{otherwise} \end{cases}$$

This assignment is consistent and satisfies all clauses. So this is a complete NP hardness proof.

Example 2.1 (Mario)**.** Now for a "fun" reduction example, Super Mario Bros the original. This is NP hard. We can construct a giant level in Mario Maker, and we detail why solving a general Mario level is NP hard. We do this by reducing from 3SAT. Now our clause gadgets and variable gadgets, rather than being graphs, are pieces of a level. Mario often needs to make a choice if some thing x_1 is true or false (variable gadgets), some choice x_2, etc. Then he runs back through clause gadgets to make it to the final. Zelda is also NP hard (opening doors with swords).

???? Stars/mushrooms? So clauses are like stages. So someone could give you a password, to PSAT, to SAT, to 3SAT, to a certain Mario level. Zelda is even harder, it is PSPACE-hard. In order be in NP, we need to be able to check an answer; it takes exponential time to prove (play levels, due to backtracking).

Karp's 21 problems

3 ok