Week 11 - Monday

COMP 4500

Last time

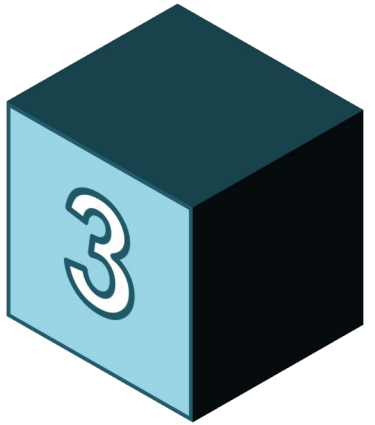
- What did we talk about last time?
- Polynomial-time reductions

Questions?

Assignment 6

Logical warmup

- A businesswoman has two cubes on her desk
- Every day she arranges both cubes so that the front faces show the current day of the month
- What numbers do you need on the faces of the cubes to allow this?
- Note: Both cubes must be used for every day



Three-sentence Summary of Reductions via Gadgets and Efficient Certification

Reductions via Gadgets

SAT and 3-SAT

- Consider a set of ***n*** Boolean variables, $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, …, $\boldsymbol{x_n}$
- Each value is 0 or 1
- A **term** is either a variable x_i or its negation $\overline{x_i}$
- A **clause** is a disjunction (set of logical ORs) of terms, like:

$$x_1 \vee \overline{x_6} \vee \overline{x_5} \vee x_2 \vee \overline{x_3} \vee x_4$$

- A clause has length ***l*** if it has ***l*** terms
- A truth assignment is an assignment of 0 or 1 to every x_i

Satisfiability

- A clause is **satisfied** if a truth assignment evaluates it to true
- A collection of clauses is satisfied if a truth assignment satisfies each clause
- Another way to view satisfiability is that, given clauses C_1, C_2, …, C_k, the following statement evaluates to true with some truth assignment:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

Satisfiability example

- Consider the following three clauses:

$$(x_1 \vee \overline{x_2}), (\overline{x_1} \vee \overline{x_3}), (x_2 \vee \overline{x_3})$$

- Can you find an assignment of 0 and 1 to each of the three variables such that all three clauses evaluate to true (1)?

Satisfiability

- The satisfiability problem (SAT):
  - Given a set of clauses $C_1, C_2, \ldots, C_k$ over a set of variables $\{x_1, x_2, \ldots, x_n\}$, is there a satisfying truth assignment?
- The 3-satisfiability problem (3-SAT) is a special case of SAT in which all clauses have exactly three terms:
  - Given a set of clauses $C_1, C_2, \ldots, C_k$, each of length 3, over a set of variables $\{x_1, x_2, \ldots, x_n\}$, is there a satisfying truth assignment?

Reducing 3-SAT to independent set

- Can we reduce 3-SAT to independent set?
- The problems seem completely different:
  - Assigning true or false to Boolean variables
  - Picking vertices that are not neighboring
- We need some glue that can tie together these two (seemingly) different problems
- Enter: gadgets
- In this case, we're going to build Boolean constraints into the nodes and edges of a graph

3-SAT $\leq_P$ independent set

- **Proof**:
  - We have a black box for independent set and want to solve an instance of 3-SAT consisting of variables $\{x_1, x_2, \ldots, x_n\}$ and clauses $C_1, C_2, \ldots, C_k$.
  - Because terms are ORed together in clauses, we only need to pick one term per clause to be true
  - We need to pick one that doesn't have a **conflict** with a term in another clause.
  - A term has a conflict if it is the negation of the term we pick.

Proof continued

- Construct a graph with 3k nodes grouped into k triangles such that each triangle contains nodes corresponding to each term in a clause, each connected by edges.
- In other words, for $i = 1, 2, \ldots, k$, we construct vertices v_{i1}, v_{i2}, v_{i3}, joined by edges.
- Each vertex is labeled v_{ij}, meaning term j from clause C_i.
- No vertices in an independent set are joined to each other; thus, no two vertices in each clause could be in the independent set.

Proof continued

- Although we have guaranteed that no two terms in a clause will be part of an independent set, we have not prevented conflicts.
- For each two terms that conflict, we add an edge between them as well.
- We claim that the original 3-SAT instance is satisfiable if and only if the graph we constructed has an independent set of size at least ***k***.

Proof continued

- If the 3-SAT instance is satisfiable, each triangle in our graph contains at least one node whose label evaluates to 1. Let ***S*** be a set consisting of one such node from each triangle.
- ***S*** is independent, because if there were an edge between two nodes ***u***, ***v*** ∈ S, the labels of ***u*** and ***v*** would conflict, but that can't happen, since they both evaluate to 1.

Proof continued

- In the other direction, suppose our graph has an independent set ***S*** of size at least ***k***.
- First, it must be exactly ***k***, because it can't be more without including more than one per triangle.
- Thus, it must include exactly one per triangle.
- There is a truth assignment that satisfies all clauses, specifically:
  - If neither label x_i nor $\overline{x_i}$ is in ***S***, arbitrarily set x_i to 1
  - Otherwise, one of them is in ***S***
    - If label x_i is in ***S***, we set it to 1, otherwise we set it to 0

■

Observations about reductions

- Every NP-complete problem has had other NP-complete problems reduced to it
- Coming up with clever gadgets that allow the reduction is hard
  - Countless papers proposing gadgets have been published
- Reductions are transitive
  - If $\boldsymbol{Z} \leq_P \boldsymbol{Y}$ and $\boldsymbol{Y} \leq_P \boldsymbol{X}$, then $\boldsymbol{Z} \leq_P \boldsymbol{X}$

Efficient Certification

What makes a problem NP?

- Is there something that sets apart problems that are NP-complete from other problems that (probably) take exponential time?
- Yes!
- It's easy to prove that you have an answer for one
- In other words, they're easy to **check**

Checking

- Does this graph have an independent set of size at least ***k***?
  - If you show me a set of vertices, claiming they're independent, it's easy to check that they are
- Can this 3-SAT be satisfied?
  - If you give me a truth assignment to variables, it's easy to check that every clause is true
- Can you put a set of objects with at least value ***v*** into this knapsack?
  - If you show me the objects, it's easy to check that they have the given value and fit in the knapsack
- It's hard to **find** these solutions, but they're easy to **check**!

Problems and algorithms

- Input to a problem will be encoded as a finite (binary) string ***s***
- The length of ***s*** is $|s|$
- For a decision problem, an algorithm ***A*** receives an input string and returns "yes" or "no"
  - This output is $A(s)$
- A decision problem ***X*** is the set of strings for which the answer is "yes"
- ***A* solves** the problem ***X*** if for all strings ***s***, $A(s)$ = "yes" if and only if $s \in X$

The class of problems P

- Formally, an algorithm ***A*** has polynomial running time if
  - There is a polynomial function ***p***(***x***)
  - Such that, for every input string ***s***, the algorithm ***A*** terminates on ***s*** in at most O(***p***(|***s***|)) steps
- Thus, **P** is the set of all decision problems ***X*** for which there is an algorithm ***A*** with polynomial running time that solves ***X***

Efficient certification

- ***B*** is an **efficient certifier** for a problem ***X*** if:
  - ***B*** is a polynomial-time algorithm that takes two input arguments ***s*** and ***t***
  - There is a polynomial function ***p***(***x***) such that, for every string ***s***, we have ***s*** ∈ ***X*** if and only if there exists a string ***t*** such that |***t***| ≤ ***p***(|***s***|) and ***B***(***s***,***t***) = "yes"
- ***B*** can evaluate a "proof" ***t*** for input ***s***
- You could use ***B*** as part of a brute force approach, trying lots of strings ***t*** to see if they work for ***s***

The class of problems NP

- **NP** is the set of all problems for which there exists an efficient certifier
- Note that **P** $\subseteq$ **NP**
  - Why?
  - We can make an efficient certifier by simply using an efficient solver
  - Such a certifier could even ignore string ***t*** and check ***s*** on its own
- **NP** is an abbreviation for "nondeterministic polynomial" because, for a machine that can nondeterministically explore all paths at the same time, checking a solution and finding a solution take the same time

P = NP?

- Is there a problem in **NP** that is not in **P**?
  - Many computer scientists believe that this is true
  - All **NP-complete** problems are believed to be too hard to solve in polynomial time
  - Other problems like factoring might not be as hard as **NP-complete** but might not be in **P** either
- People have tried really hard to solve **NP-complete** problems in polynomial time…and failed

Thoughts about NP

If P=NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett. It's possible to put the point in Darwinian terms: if this is the sort of universe we inhabited, why wouldn't we already have evolved to take advantage of it?

Scott Aaronson

Upcoming

Next time…

- Proving problems NP-complete
- Review

Reminders

- Work on Assignment 6
- Read Section 8.4