CS 170 Section 11

Approximation Algorithms

Owen Jow

April 11, 2018

University of California, Berkeley

Table of Contents

Reduction Review

Dominating Set

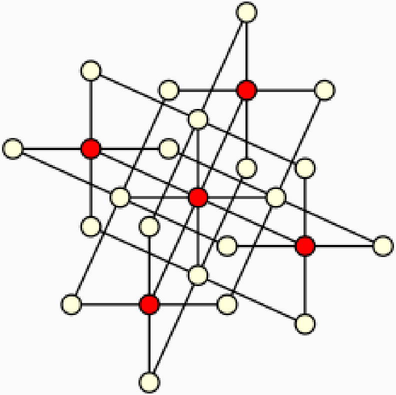


Figure 1: dominating set. A subset of vertices that either includes or touches (via an edge) every vertex in the graph.

Minimal Dominating Set

Minimal dominating set

A dominating set with $\leq k$ vertices.

Let $k = 2$:

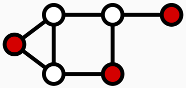
(a)

Not a minimal dominating set.

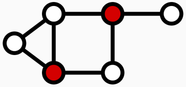
(b), (c)

Minimal dominating sets.

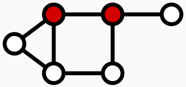
(a)



(b)



(c)



Proving NP-Hardness

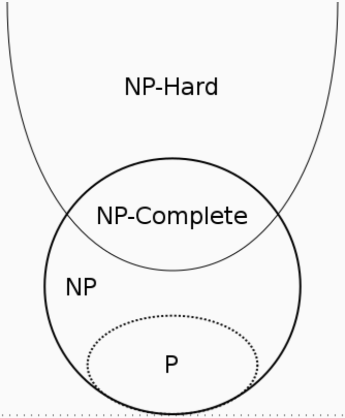
To prove that something is NP-hard, reduce a known NP-complete problem to it.

Recall:

- **NP-hard:** at least as hard as the NP-complete problems.
- Difficulty flows in the direction of the reduction. If we reduce A to B, then B is at least as hard as A.

Some NP-Complete Problems

Vertex cover	Subset sum
Set cover	Longest path
ZOE	Rudrata cycle
MAX-2SAT	Dominating set
SAT	Independent set
Battleship	3D matching
Knapsack	Balanced cut
Clique	Verbal arithmetic
TSP	Optimal Rubik's cube
ILP	Steiner tree (decision)



$\mathrm{P} \neq \mathrm{NP}$

Exercise 1

Argue that the *minimal dominating set* problem is NP-hard.

Exercise 1 Solution

Argue that the *minimal dominating set* problem is NP-hard.

We can reduce *minimal set cover* to minimal dominating set.

Randomization for Approximation

Exposition

NP-complete (and NP-hard) problems are everywhere. They're the shadows in the evening, the corruption in the government, the flyer people on Sproul.

What can be done? Assuming P $\neq$ NP, an optimal solution cannot be found in polynomial time.

So we must rely on alternatives. Intelligent exponential search is one of these. *Approximation algorithms* are another.

Approximation Algorithms

An **approximation algorithm** finds a solution with some guarantee of closeness to the optimum. Notably, it is *efficient* (of polynomial time).

There are many ways to approximate (think of all the efficient problem-solving strategies you've learned so far!). Greedy and randomized approaches are popular, as they tend to be easy to formulate.

Exercise 2a

Devise a randomized approximation algorithm for MAX-3SAT. It should achieve an approximation factor of $\frac{7}{8}$ in expectation.

Feel free to assume that each clause contains three distinct variables.

Exercise 2a Solution

Randomly assign each variable a value. Let X_i (for $i = 1, ..., n$) be a random variable that is 1 if clause i is satisfied and 0 otherwise. Then

$$\mathbb{E}[X_i] = (0)\left(\frac{1}{8}\right) + (1)\left(\frac{7}{8}\right) = \frac{7}{8}$$

Let $X = \sum_{i=1}^{n} X_i$ be the total number of clauses that are satisfied.

$$\mathbb{E}[X] = \mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbb{E}[X_i] = \sum_{i=1}^{n} \frac{7}{8} = \frac{7}{8}n$$

Let d^* be the optimal number of satisfied clauses. We have that $n \geq d^*$. Therefore, a random assignment is expected to satisfy $\frac{7}{8}n \geq \frac{7}{8}d^*$ clauses.

Exercise 2b

The fact that $\mathbb{E}[X] = \frac{7}{8}n$ tells us something about every instance of MAX-3SAT.

Namely...?

Exercise 2b Solution

The fact that $\mathbb{E}[X] = \frac{7}{8}n$ tells us something about every instance of MAX-3SAT.

Namely...?

There always exists an assignment for which at least $\frac{7}{8}$ of all clauses are satisfied. Otherwise the expectation could not be $\frac{7}{8}$ of all clauses.

Fermat's Little Theorem as a Primality Test

Fermat's Little Theorem

Fermat's little theorem:

If p is prime and a is coprime with p, then $a^{p-1} \equiv 1 \pmod{p}$.

a, b **coprime**
The GCD of a and b is 1.

Fermat's Little Theorem as a Primality Test

Say we want to determine whether n is prime. We might think to use FLT as a primality test, i.e.

- Pick an arbitrary $a \in [1, n-1]$ and compute $a^{n-1} \pmod{n}$.
- If this is equal to 1, declare n prime. Else declare n composite.

But does this really work? Spoilers: no.

Exercise 3a

(i) Find an a that will trick us into thinking that 15 is prime.

(ii) Find an a that will correctly identify 15 as composite.

Exercise 3a Solution

(i) Find an a that will trick us into thinking that 15 is prime.
4 will work for this. Note: when n is composite but $a^{n-1} \equiv 1 \pmod{n}$, we call n a *Fermat pseudoprime* to base a.

(ii) Find an a that will correctly identify 15 as composite. 7.

Exercise 3b

By FLT, primes will always be identified.

The problem is *false positives* – composite n that masquerade as primes. There's one silver lining, though: if $a^{n-1} \not\equiv 1 \pmod{n}$ for some a coprime to n, then **this must hold for at least half of the possible values of a**.

Exercise 3b

Suppose there exists some a in (mod n) s.t. $a^{n-1} \not\equiv 1 \pmod{n}$, where a is coprime with n. Show that n is **not** Fermat-pseudoprime to at least half of the numbers in (mod n).

How can we use this to make our algorithm more effective?

Exercise 3b Solution

For every b s.t. $b^{n-1} \equiv 1 \pmod{n}$,

$$(ab)^{n-1} = a^{n-1}b^{n-1} \not\equiv 1 \pmod{n}$$

Since a and n are coprime, a has an inverse modulo n.
Thus ab is unique for every unique choice of b ($ab_1 \not\equiv ab_2$ iff $b_1 \not\equiv b_2$).
By extension, for every b to which n **is** Fermat-pseudoprime, there is a unique ab to which n is **not** Fermat-pseudoprime.

We can make our algorithm more effective by checking a bunch of a (not just one). The chance of being wrong k times in a row is at most $\frac{1}{2^k}$.

Exercise 3c

Even with the improvement from (b), why might our algorithm still fail to be a good primality test?

Exercise 3c Solution

Even with the improvement from (b), why might our algorithm still fail to be a good primality test?

In order to correctly identify composite n, we need an a coprime with n s.t. $a^{n-1} \not\equiv 1 \pmod{n}$. But there is no guarantee that such an a exists!

(Such composite n, which pass the FLT primality test for all a, are called *Carmichael numbers*.)