CS 170 DIS 11

Released on 2018-04-11

1 Reduction Review

The dominating set of a graph $G = (V, E)$ is a subset D of V, such that every vertex not in D is a neighbor of at least one vertex in D.
Let the Minimal Dominating Set problem be the task of determining whether there is a dominating set of size $\leq k$.
Show that the Minimal Dominating Set problem is NP-Hard. You may assume for this question that all graphs are connected.

Solution:

We will reduce the Minimal Set Cover problem to the Minimal Dominating Set problem. Suppose (S, U) is an instance of set cover where U denotes the universe of distinct elements and S is a set of S_i, where each S_i is some subset of U.
We will construct a graph $G = (V, E)$ as follows.
For each element u in U construct a vertex. For each possible S_i construct a vertex. Connect each vertex S_i to all u in S_i.
Let's call "element vertices" vertices that correspond to elements in U, and "set vertices" vertices that correspond to some i.
Notice that if we were to run dominating set on the graph right now, we'd be able to cover all the element vertices with any valid set cover, but we'd have to pick every single set vertex in order to ensure that all set vertices are covered. To rectify this, connect every set vertex to every other set vertex, forming a clique. This ensures that we can cover all the set vertices by picking just one. This way we really only need to worry about covering the element vertices.
Suppose we have some minimal set cover of size k. This will correspond to a dominating set of size k as well. For each set in our minimal set cover, pick the corresponding set vertex. It follow directly from the construction of the graph and the definition of a set cover that all set and element vertices are covered.
Suppose we have some dominating set D of size k. We can find a set cover of size $\leq k$. To do this, we will construct a new dominating set D' that contains only set vertices. Include every set vertex in D in D'. For each vertex in our dominating set that is an element vertex, pick any random neighboring set vertex and add it to D'. Observe that $|D'| \leq |D|$.
Thus if there is a dominating set in G of size $\leq k$, there must be a set cover of size $\leq k$

2 Randomization for Approximation

Often times, extremely simple randomized algorithms can achieve reasonably good appromxiation factors.

a Consider Max 3-SAT (given a set of 3-clauses find the assignment that satisfies as many of the as possible). Come up with a simple randomized algorithm that will achieve an approximation factor of $\frac{7}{8}$ in expectation. That is, if the optimal solution satisfies k clauses, your algorithm should come up with an assignment that satisfies at least $\frac{7}{8} * k$ clauses in expectation. You may assume that every clause contains exactly 3 distinct variables in it.

b What can this tell us about any instance of Max 3-SAT?

Solution:

a Consider randomly assigning each variable a value. Let X_i be a random variable that is 1 if clause i is satisfied and 0 otherwise. We can see that the expectation of X_i is $\frac{7}{8}$. Note that $\sum X_i$ is the total number of satisfied clauses. By linearity of expectation, the expected number of clauses satisfied is $\frac{7}{8}$ times the total number of clauses. Since the optimal number of satisfied clauses is at most the total number of satisfied clauses, a random assignment will in expectation have value at least $\frac{7}{8} * k$

b Notice that finding this expectation actually tells us something even stronger than just an approximation scheme. If in expectation at least $\frac{7}{8}$ of all clauses are satisfied, there must necessarily *always* exist an assignment where at least $\frac{7}{8}$ of all clauses are satisfied, as a random variable is at least sometimes greater than or equal to its mean.

Optional Note: It may seem unsatisfying that we've found an algorithm that only works in expectation. How many times would we have to run it before we actually find a solution that achives the approximation factor? Of course, it could be a lot. It turns out, however, there's a way to effectively de-randomize this scheme in order to always produced an assignment with at least $\frac{7}{8}$ of all clauses satisfied.

Fix variable assignments one by one. When picking a value for variable x_k, consider the conditional expectation on the number of satisfied clauses given that x_0, ..., x_{k-1} hold whatever value you already set them to and x_k is assigned to true. Then consider the same conditional expectation but with x_k assigned to false. Assign x_k to whichever value maximizes the conditional expectation and then move on. At the very first step, the expectation of the two conditional expectations will be $\frac{7}{8}$ of the number of clauses. Thus the larger of the two must also be at least $\frac{7}{8}$ of the number of clauses. At the next step,

one of our two choices will have conditional expectation at least as large as the expectation when conditioned only on the first value and so on.

Notice that this new scheme is fully deterministic, despite relying on probablistic concepts.

3 Fermat's Little Theorem as a Primality Test

Recall that Fermat's Little Theorem states the following:

"For a prime p and a coprime with p, $a^{p-1} \equiv 1 \pmod{p}$."

Assume for a general (not necessarily prime) p, we want to determine if p is prime. It may be tempting to try to use Fermat's Little Theorem as a test for primality. That is, pick some random a and compute $a^{p-1} \pmod{p}$. If this is equal to 1, return that p is prime, else return that it is composite. In this question we will investigate how effective this method actually is.

a Suppose we wanted to test if 15 was prime. What is a choice of a that would trick us into thinking it is prime? What is a choice of a that would lead us to the correct answer? For choices of a that trick us into believing p is prime, we often say that p is "Fermat pseudoprime" to base a.

b Suppose there exists a single a in $\pmod{p}$ such that $a^{p-1} \not\equiv 1 \pmod{p}$, where a is coprime with p. Show that p is not Fermat pseudoprime to least half the numbers in $\pmod{p}$. How might we use this to make our algorithm more effective?

c Given the improvement from the previous question, why might our algorithm still fail to be a good primality test?

Solution:

a A choice of a that would trick us into thinking 15 is prime is 4. There are a few other numbers we could have used here. A choice of a that would lead us to the correct answer is 7.

b Let's assume there is at least one number b such that $b^{p-1} \equiv 1 \pmod{p}$. $(a * b)^{p-1} \not\equiv 1 \pmod{p}$. Further more, for each possible choice of b, $a * b$ will be a unique number. This is the case since a necessarily has an inverse in mod p, making the function $f(x) = a * x \pmod{p}$ a bijection. For every b that p is Fermat pseudoprime to, we have a unique a that would have led us to the correct answer. Thus at least half the numbers $\pmod{p}$ would lead us to the correct answer.
We can improve our algorithm by checking multiple a rather than just 1. This doesn't increase our runtime substantially, but will sharply decrease the probability of a false positive.

c For prime p we will always arrive at the correct answer. For non-prime p, we know that when there exists an a coprime with p such that $a^{p-1} \not\equiv 1 \pmod{p}$, we will probably arrive at the correct answer. However, we are not guaranteed the existence of such an a in the first place. There are potentially numbers where no such a exists. These numbers are called Carmichael numbers.