Homework 9

Winston (Hanting) Zhang

Friday, November 23, by 23:00

General Instructions. The following assignment is meant to be challenging, and we anticipate that it will take most of you at least 10–15 hours to complete, so please allow yourself plenty of time to work on it. We highly recommend reading the entire assignment right away — you never know when inspiration will strike. Please provide a formal mathematical proof for all your claims, and present runtime guarantees for your algorithms using asymptotic (big-$O/\Omega/\Theta$) notation, unless stated otherwise. You may assume that all basic arithmetic operations (multiplication, subtraction, division, comparison, etc.) take constant time. In problems involving graphs, we typically use n and m to denote the number of nodes and edges, respectively, unless otherwise stated.

Collaboration. Please carefully check the collaboration policy on the course website. When in doubt, ask an instructor.

Consulting outside sources. Please carefully check the policy regarding outside sources on the course website. Again, when in doubt, ask an instructor.

Submission. Homework submission will be through the Gradescope system. Instructions and links have been provided through the course website and Piazza. The only accepted format is PDF. Only typed solutions will be accepted, and we highly recommend producing your solutions using LaTeX (the text markup language we are also using for this assignment).

Recommended practice problems (do not hand in): KT Problems 8.3, 8.4, 8.5, 8.6, 8.12, 8.13, 8.15, 8.16, 8.19, 8.26, 8.27, 8.28, 8.31, 8.37

Problem 1. [5+10=15]

To justify our focus on decision problems in studying the P vs. NP question (and related questions later), we mentioned in class that most natural optimization problems are "equivalent" to a closely related decision problem, or that the decision problem "captures the essence of the difficulty" of the problem. Here, you will explore this for yourselves for a specific example, the MINIMUM CUT WITH NEGATIVE CAPACITIES problem.[1] Specifically, we will consider three closely related problems.

Decision Given a (directed) graph $G = (V, E)$ with integer edge costs/capacities c_e *which are allowed to be positive or negative*, a designated source s and a designated sink t, and an integer C, is there an s-t cut $(S, \overline{S})$ of capacity at most C?

Optimization Given a (directed) graph $G = (V, E)$ with integer edge costs/capacities c_e *which are allowed to be positive or negative*, a designated source s and a designated sink t, *find* the smallest capacity of any s-t cut, i.e., just the value.

Search Given a (directed) graph $G = (V, E)$ with integer edge costs/capacities c_e *which are allowed to be positive or negative*, a designated source s and a designated sink t, *find* a minimum s-t cut, i.e., the set S minimizing $c(S, \overline{S})$.

1. Show that the decision and optimization versions of MINIMUM CUT WITH NEGATIVE CAPACITIES are polynomial-time equivalent. That is, either both are solvable in polynomial time or neither is.
2. Show that the optimization and search versions of MINIMUM CUT WITH NEGATIVE CAPACITIES are polynomial-time equivalent. That is, either both are solvable in polynomial time or neither is.

Note that here, we are not specifically asking you for Karp reductions. In fact, given that Karp reductions are only defined from one decision problem to another, that wouldn't even make any sense here. Instead, the mindset is that someone else has a program that solves one of the three problems in polynomial time, and you are supposed to show how to use that program (possibly calling it multiple times on different inputs) in order to solve another problem.

The reductions between decision and optimization should be quite simple if you understood the definitions well. The reduction from optimization to search is also not so hard. The reduction from search to optimization is more involved, and will probably involve *multiple* calls to a black box for solving the optimization problem, on different inputs. Think about this direction as someone giving you code that solves the optimization version, and you want to use multiple calls to this code (using different inputs) in order to "reconstruct" a solution.

Proposition 1. *The decision and optimization versions of* MINIMUM CUT WITH NEGATIVE CAPACITIES *are polynomial-time equivalent.*

Proof. We prove both directions. Assume that $f(G, c_e, C)$ is a function that decides if there is a cut with capacity at most C. To find the findest capacity over all cuts, let $C_{\text{max}} = \sum_{e \text{ out of } s} c_e$, and binary search on the interval $[0, C_{\text{max}}]$. At each point, we query some value C. If $f(G, c_e, C)$ returns "yes", then we query the upper interval with $(C + C_{\text{max}})/2$. Otherwise if $f(G, c_e, C)$ returns "no", then we query the lower interval with $C/2$. When we find the minimum value, we terminate. This makes $\log C_{\text{max}}$ calls to f, so we can find the smallest capacity in $O(f \cdot \log C_{\text{max}})$. If $O(f)$ is polynomial, then $O(f \cdot \log C_{\text{max}})$ is polynomial as well.

For the other direction, assume that $g(G, c_e)$ is a function that finds the smallest value C_{min} over all cuts. To find if there exists a cut with capacity at most C, simply compare $C_{\text{min}} \leq C$. If true, then return "yes", otherwise return "no". This runs in $O(g)$, which is polynomial.

Thus f is polynomial iff g is polynomial, as desired. $\square$

Proposition 2. *The the optimization and search versions of* MINIMUM CUT WITH NEGATIVE CAPACITIES *are polynomial-time equivalent.*

[1]As you well know by now, when all edge weights are non-negative, you can find a minimum cut in polynomial time. We will not prove this in class, but when you allow negative edge weights, the problem becomes NP-hard.

Proof. We prove both directions. Assume that $g(G, c_e)$ is a function that finds the smallest value C_{min} over all cuts. To find the set S^* that minimizes the cut, we do the following procedure for each node $v \in V$:

1. Compute $C = g(G, c_e)$
2. Add an edge (s, v) with infinite capacity.
3. Compute $C' = g(G, c_e)$ with the new edge.
4. If $C = C'$, then add v to S^*.

Then return S^* in the end.

We claim that this finds a minimum cut. First, we show that if $C = C'$, then v is in some minimum cut. Indeed, suppose v was in some minimum cut S. Then adding the edge (s, v) can only increase the value of all cuts of G. Since $v \in S$, the capacity of S doesn't change when we add (s, v). Thus our check $C = C'$ correctly adds v to our answer when if v is in some minimum cut.

If v is not in any minimum cut, then adding the edge (s, v) will force v to be in the new minimum cut, or else (s, v) will be counted as an outgoing edge from s and our minimum capacity would be infinity, which is bad. If $C \neq C'$, then our check correctly does not add v to our answer. The tricky part is that is still may be possible that $C = C'$, since there may be multiple valid minimum cuts. But we claim that this cannot happen and leads to contradiction. Indeed, if S is a minimum cut of the same capacity C after adding (s, v), then S had the same capacity before adding (s, v), thus S was originally a minimum cut, contradiction! Thus all $v \in S^*$ is in some minimum cut.

The second tricky part is that we may have again different minimum cuts being represented in S^*, and the union is minimum cuts may not be minimal. However, we again claim that this isn't an issue. By the min-flow max-cut theorem, if we have two minimum cuts S and T, then all edges across S and T have full flow. The edges across $S \cup T$ is a subset of the edges of S and T and therefore also have full flow. This implies that $S \cup T$ is also a min cut. Thus S^* is actually a union of all min cuts, but it itself is also a min cut! This completes the first direction.

The other direction is easy: simply compute S and calculate the capacity of S. $\square$

Problem 2. [4+6=10]

For each of the following two problems, convert it into the "natural" decision problem, and prove that this decision problem is in NP. Notice that we are not asking you to prove that the decision problems are NP-hard, only that they are in NP.

1. An instructor has a classroom of n students, and wants to divide them into k teams of n/k students each. (You can assume that k divides n.) Each student has an integer skill level $s_i \geq 0$. The skill of a team is the sum of the skills of its team members. To be as fair to the students as possible, the instructor wants the skills to be even. The instructor will measure this as the difference Δ between the total skill of the most skilled team and the least skilled team. The instructor's goal is now to minimize the difference, over all possible legal assignments.

   Answer: Rephase the problem as a decision problem: Given n students, does there exist a division of students into k teams of n/k students such that the difference between the most skilled and least skilled team is at most some constant C?

   We claim that this decision problem is in NP. The certifier does the following: We compute the skill levels of each team and find the maximum and minimum teams. If the difference is at most C, return "Yes", otherwise return "No". This runs in polynomial time ($O(n)$) and the size of the certificate is polynomial in the input, so we have an efficient certifier.

2. You are given a set V of nodes, but no edges (yet). Instead, you are given k partitions $(S_i, \overline{S}_i), i = 1, \ldots, k$ of nodes, and a cut capacity bound $C \geq 0$. Your goal is to "stuff" as many undirected edges into the graph as possible, but making sure that for each of the given cuts, at most C edges are cut. More formally, your goal is to find a set E of undirected edges of maximum size such that for each of the cuts $(S_i, \overline{S}_i)$, the number of edges it cuts in the graph (V, E) is at most C.

As a motivation for this problem, imagine that you have seen n people form teams for k games, and you assume that they mostly stay on the same team with friends, so at most C friendships are broken for each team assignment. Now, you are trying to figure out from this information who might have been friends with whom. Of course, this would be super easy if you could just return that no one was friends with anyone, so you want to force the solution to include many edges.

Answer: Rephase the problem as a decision problem: Given nodes V, k partitions $(S_i, \bar{S_i})$, and a cut capacity of C, does there exist a set of edges E with size at least n, such that for each cut $(S_i, \bar{S_i})$, at most C edges are cut?

We claim that this decision problem is in NP. The certifier does the following: We compute the number of edges in each cut, c_i. If $c_i \leq C$ for all i and $|E|$ is at least n, return "Yes", otherwise return "No". This runs in polynomial time ($O(Vk)$ to find each cut plus some more linear stuff) and the size of the certificate is polynomial in the input (E is $O(V^2)$), so we have an efficient certifier.

Problem 3. [1+1+1+2=5]

To help you understand the concepts underlying NP and certifiers more, let's look at a problem that seems like it should be in NP, but probably is not.[2] SOKOBAN is a fun[3] single-player puzzle game. You play it on a 2-dimensional grid. There are squares you can walk on, walls (which you can never walk on or cross), and boxes. The boxes start in particular positions on the grid, and must end at given destination positions. (Boxes are interchangeable, so it doesn't matter which box ends in which destination.) Your character can walk up or down or left or right. It can also *push* single boxes, i.e., if it stands next to a box, and the square on the other side is open, it can push the box onto that square, while stepping on the square previously occupied by the box. The character cannot pull any boxes (or move them in any way besides pushing), and cannot push two or more boxes at once. The goal is to get all boxes to the destinations in as few moves as possible.

1. Phrase SOKOBAN as a decision problem.

   Given a SOKOBAN board state b and a move limit C, is there a sequence of at most C moves that solves b (i.e. get all the boxes to where they need to be)?

2. What would be a natural certificate for SOKOBAN instances?

   A sequence of valid moves is the natural certificate for SOKOBAN.

3. What would be a natural certifier for SOKOBAN?

   Given a SOKOBAN board b and a certificate y, the certifier $B(b, y)$ simply simulates the moves y and checks if all boxes are where they need to be.

4. In light of your answers to the previous questions, what is the missing piece? That is, why haven't you shown that SOKOBAN is in NP?

   We haven't shown that $B(b, y)$ is an efficient certifier, because we haven't shown that there is at least one y with size polynomial in b. For example, there may be board states of size n that only have exponential sized solutions.

Problem 4. [2+6=8]

Here is an NP-complete problem which you will actually be able to prove NP-complete *by hand* yourself. We will call the problem CERTIFICATION. The input is a quadruple $(P, \hat{x}, \hat{t}, \hat{z})$. P is the source code of a C++ program which takes two inputs; and $\hat{x}, \hat{t}, \hat{z}$ are binary strings. The question (decision problem) is whether there exists a binary string y of length at most $|y| \leq |\hat{z}|$ such that $P(\hat{x}, y)$ terminates in at most $|\hat{t}|$ steps and returns "Yes". (Notice that the contents

[2]SOKOBAN is PSPACE-complete, and while it is not known if PSPACE is different from NP (it is not even known if PSPACE is different from P!!!!), people suspect that PSPACE and NP are different.

[3]and quite addictive, so maybe don't start playing it until after the course final

of $\hat{t}, \hat{z}$ are never referenced. Only their lengths are used, as a bound on the number of computation steps and on the length of y.)

1. Prove that CERTIFICATION is in NP.

   Let $\alpha = (P, \hat{x}, \hat{t}, \hat{z})$ and $B(\alpha, y)$ be a certifier that does the following: If $|y| > |\hat{z}|$, then return "No". If $|y| \leq |\hat{z}|$, take y and run the program $P(\hat{x}, y)$; if $P(\hat{x}, y)$ terminates within $|\hat{t}|$ steps and returns "Yes", then $B(\alpha, y)$ returns "Yes", otherwise $B(\alpha, y)$ returns "No".

   We claim that B is an efficient certifier. Indeed, if the answer to $\alpha = (P, \hat{x}, \hat{t}, \hat{z})$ is "Yes", then there exists some y such that $|y| \leq |\hat{z}|$ such that $P(\hat{x}, y)$ answers "Yes" in at most $|\hat{t}|$ steps. In particular, $|y| \leq |\hat{z}| \leq |(P, \hat{x}, \hat{y}, \hat{z})| = |\alpha|$ is polynomial in α. By the definition of B, $B(\alpha, y)$ returns "Yes".

   If the answer to $\alpha = (P, \hat{x}, \hat{t}, \hat{z})$ is "No", then there is no binary string y with length $|y| \leq |\hat{z}|$ such that $P(\hat{x}, y)$ answers "Yes" in at most $|\hat{t}|$ steps. Now consider $B(\alpha, y)$ on arbitrary inputs y. Let's do casework on whether $|y|$ is at most or greater than $|\hat{z}|$. If it's the former, then we know that $P(\hat{x}, y)$ is always "No", so $B(\alpha, y)$ is "No". If it's the latter, then by definition $B(\alpha, y)$ is "No".

   Thus B is an efficient certifier.

2. For every problem $X \in$ NP, give a reduction $X \leq_p$ CERTIFICATION. (Note that this is infinitely many reductions, so please don't write them all out individually. Instead, give a generic reduction, using nothing about X except that X is in NP.)

   Since X is in NP, it has an efficient certifier B. Furthermore, we know that $B(x, y)$ runs in some polynomial time $b(|x|, |y|)$ and we are given some r to bound the size of y. We claim that the function $x \mapsto (B, x, b(|x|, r(|x|)), r(|x|))$ is a Karp reduction from CERTIFICATION to X. Indeed, each input $x \in X$ generates a valid output in CERTIFICATION.

   If the answer to x is "Yes", then there exists some y of length at most $|y| \leq r(|x|)$ such that $B(x, y)$ answers "Yes" (in at most $b(|x|, r(|x|))$ steps). By the definition of CERTIFICATION, we have the answer to $(B, x, b(|x|, r(|x|)), r(|x|))$ is "Yes".

   If the answer to x is "No", then there is no y such that $B(x, y)$ answers "Yes". By the definition of CERTIFICATION, the answer to $(B, x, b(|x|, r(|x|)), r(|x|))$ is "No".

   Thus this function is a valid Karp reduction and for all $X \in$ NP, we have $X \leq_p$ CERTIFICATION, as desired.

Note: This problem is quite easy if you fully understand what NP and certification mean. Until you do fully understand them, though, it may look very intimidating and complicated.

Problem 0.

Chocolate Problem: 1 chocolate bar

You are given a directed graph G (without edge weights) and a source s and sink t, as well as a bound b. You are to decide if there are two vertex-disjoint paths from s to t (i.e., they share no vertex besides s, t), each using at most b edges. Prove that this problem is NP-complete. Note that without the length bound of b, this problem can be solved in polynomial time using integer Max-Flow.

(Hint: 3SAT)