CSCI270 Homework 9

Jacob Ma

November 24, 2022

Problem 1

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.[a] 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.

[a] 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 of $1.a$. In order to prove the decision and optimization versions are equivalent in polynomial runtime, we need to prove they are equal on both direction. We need to prove two lemmas.

Lemma 0.0.1

$$\text{DECISION} \leqslant_p \text{OPTIMIZATION}$$

Proof of Lemma. We are trying to prove here that the Optimization version could be reduced to Decision version in polynomial time. This is trivially true, since we could run the Optimization version first given $G = (V, E)$ with integer edge capacities c_e and directly find the smallest capacity of any $s-t$ cut, denote it C'. Comparing C' with C, if $C' \leqslant C$, meaning the answer to the decision version is "Yes", otherwise "No". These reduction takes $\mathcal{O}(1)$ time, which is polynomial. Thus, the lemma holds. □

Lemma 0.0.2

$$\text{OPTIMIZATION} \leqslant_p \text{DECISION}$$

Algorithm 1: Decision Reduced to Optimization

```
Find smallest possible value min of s – t cut by adding all negative capacities ;
Find biggest possible value max of s – t cut by adding all positive capacities ;
L = min, R = max ;
while R > L do
    M = (L + R) /2 ;
    if DECISION(M) then
        R = M;
    else
        L = M + 1 ;
return R
```

Proof of Lemma. The above algorithm gives a rough sense of how to reduce Decision version could be reduced to Optimization version. The high level idea is that we try to find the largest k for which the answer in Decision version is "Yes". Since there is a upper bound a and a lower bound (both got in linear time) on k we need to search, which is

$$\sum_{\text{All } e \text{ with negative capacities}} c_e \leqslant k \leqslant \sum_{\text{All } e \text{ with positive capacities}} c_e$$

Thus, we could use binary search and solve decision version for $\mathcal{O}(\log n)$ different values of k, which is polynomial. Thus, the reduction holds from this direction. □

Because of both lemma, the decision and optimization versions of MINIMUM CUT WITH NEGATIVE CAPACITIES are polynomial-time equivalent. □

Proof of 1.*b.* In order to prove OPTIMIZATION and SEARCH version are equivalent in polynomial runtime, we prove they in both directions.

Lemma 0.0.3

$$\text{OPTIMIZATION} \leqslant_p \text{SEARCH}$$

Proof of Lemma. This is trivial. We could run the SEARCH version and outputs the smallest the minimum $s-t$ cut if exists, and compute the capacity of this $s-t$ cut, which takes at most $\mathcal{O}(m+n)$ time, which is polynomial. If such $s-t$ cut does not exist, OPTIMIZATION version will also does not exist a smallest value of $s-t$ cut. Thus, OPTIMIZATION could be reduced to SEARCH version, the lemma is true. □

Lemma 0.0.4

$$\text{SEARCH} \leqslant_p \text{OPTIMIZATION}$$

Let's consider the following algorithm.

Algorithm 2: Search Reduced to Optimization

Data: Graph $G = (V, E)$, source s, sink t, function OPTIMIZATION

```
Initialize S = {s}, S̄ = {t}, min = OPTIMIZATION(G) ;
for all vertex v ∈ V except v and t do
    if Does not exist edge e = (s, v) then
        Construct edge e = (s, v) ;
    Set capacity of e = (s, v) to ∞ ;
    if min == OPTIMIZATION(G = (V, E)) then
        Add v to S ;
    else
        Add v to S̄ ;
    Revert changes made to E, but leave changes to S;
return (S, S̄)
```

Firstly, we have the following lemma

Lemma 0.0.5

Edge v is added to S if and only if $v \in S$ in *some* minimum $s-t$ cut $(S, \overline{S})$.

Proof of Lemma. This needs to be proved in two directions:

(1)

$$v \text{ added to } S \implies v \in S \text{ in } \textit{some} \text{ minimum } s-t \text{ cut } (S, \overline{S})$$

This could be proved by proving the contrapositive:

$$v \notin S \text{ for } \textit{all} \text{ minimum } s-t \text{ cut } (S, \overline{S}) \implies v \text{ not added to } S$$

If v is not in S for any minimum $s-t$ cut, it means that $v \in \overline{S}$ in all cases while $s \in S$. Thus, any minimum cut will cut the edge $e = (s,t)$, and the capacity of minimum cut will count c_e. Since the c_e have been changed to ∞ during the construction, the value of the minimum $s-t$ cut calculated by OPTIMIZATION will also be changed from min to ∞. Since min! = OPTIMIZATION, v will be added to $\overline{S}$, thus $v \notin S$.

(2)

$$v \in S \text{ for } \textit{some} \text{ minimum } s-t \text{ cut } (S, \overline{S}) \implies v \text{ added to } S$$

Since $v \in S$ for some minimum $s-t$ cut, the changed to $e = (s,v)$ will not affect the value of the minimum $s-t$ cut. This is because both $s, v \in S$ for this minimum $s-t$ cut $(S, \overline{S})$, and the cut will not cut this edge, thus the OPTIMIZATION will remain same to min, thus v will be added to S.

Thus, the lemma holds in both direction. □

From the lemma above, we know that S is a union of nodes $v \in S$ in all minimum cuts, in other words, S is a union of all minimum $s-t$ cut. What left to show is that S is a cut. It suffices to show that the union of 2 minimum $s-t$ cuts is still a minimum $s-t$ cut, which is the following lemma.

Lemma 0.0.6

The union of 2 minimum cuts is still a minimum cut.

Proof of Lemma. Lets say we have two existing minimum cuts, which are $(S_1, \overline{S_1})$ and $(S_2, \overline{S_2})$. And we define the following sets:

$$\begin{cases} A = S_1 \cap S_2 \\ B = S_1 \cap \overline{S_2} \\ C = \overline{S_1} \cap S_2 \\ D = \overline{S_1} \cap \overline{S_2} \end{cases}$$

The purpose of this proof is trying to prove that $(S_1 \cup S_2, \overline{S_1 \cup S_2})$, or $(\overline{D}, D)$ is also a minimum cut.
Note that we have the following truth:

$$\begin{cases} A \cup B \cup C \cup D &= U \qquad U \text{ is Universe} \\ S_1 &= A \cup B \\ S_2 &= A \cup C \end{cases}$$

Thus, we can represent $c(S_1, \overline{S_1})$ and $c(S_2, \overline{S_2})$, which are the cost of minimum costs, as the following:

$$c(S_1, \overline{S_1}) = c(A \cup B, C \cup D) = c(A, C) + c(A, D) + c(B, C) + c(B, D)$$
$$c(S_2, \overline{S_2}) = c(A \cup C, B \cup D) = c(A, B) + c(A, D) + c(C, B) + c(C, D)$$

In order to prove $(\overline{D}, D)$ is a minimum cut, consider the following equation:

$$\begin{aligned} c(\overline{D}, D) + c(A, \overline{A}) &= c(A \cup B \cup C, D) + c(A, B \cup C \cup D) \\ &= (c(A, D) + c(B, D) + c(C, D)) + (c(A, B) + c(A, C) + c(A, D)) \\ &= 2c(A, D) + c(B, D) + c(C, D) + c(A, B) + c(A, C) + \underbrace{(c(B, C) + c(C, B))}_{=0} \\ &= (c(A, C) + c(A, D) + c(B, C) + c(B, D)) + (c(A, B) + c(A, D) + c(C, B) + c(C, D)) \\ &= c(S_1, \overline{S_1}) + c(S_2, \overline{S_2}) \end{aligned}$$

Since $(S_1, \overline{S_1})$ and $(S_2, \overline{S_2})$ are both minimum cuts and $(\overline{D}, D), (A, \overline{A})$ are valid cuts, to maintain the minimality of $c(S_1, \overline{S_1}) + c(S_2, \overline{S_2})$, both $(A, \overline{A})$ and $(\overline{D}, D)$ are minimum cuts.
The lemma follows the proof. □

Also, the reduction algorithm has a for loop, with $|V|$ iterations, which takes polynomial time in each iteration, thus the reduction algorithm also takes polynomial time.
The correctness follows the cases and lemmas above. □

Problem 2

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.

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.

Solution: 2 (*a*)

DECISION: Given n students, k teams where $k \mid n$, students' skill level $\{s_i\}_{i=1}^n$ and a given D, is there an assignment of students into k groups of size $\frac{n}{k}$ where the difference Δ between the most skilled group and least skilled group is at most D?

Certificate: One assignment of n students evenly into k groups with skill level $\{s_i\}_{i=1}^n$.
Certifier: Calculate the team skill level of all teams. This will iterate all n students, which takes a constant operation, addition, to each students, which takes $\mathcal{O}(n)$ time in total. Then keep track of two values min and max which leaves the lowest and highest team skill level, and iterate through k teams. This takes k iterations and constant assignment in each iterations, which takes $\mathcal{O}(k) = \mathcal{O}(n)$ in total. At last, calculate $\Delta = max - min$, which is constant time. Compare this Δ with D, if $\Delta \leqslant D$, returns "Yes"; "No" otherwise.
Runtime: As discussed in the certifier section, this takes $\mathcal{O}(n)$ time to verify the answer which is constant. Thus DECISION is NP.

Solution: 2 (*b*)

DECISION: Given a set V of nodes, k partitions $\{(S_i, \overline{S}_i)\}_{i=1}^k$, a cut capacity bound $C > 0$, and a number M, does there exist a set E of undirected edges where $|E| \geqslant M$ 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.

Certificate: A set E of undirected edges.
Certifier: If $|E| < M$, directly returns "No". Otherwise, count the maximum number of edges cut by $(S_i, \overline{S_i})$.

To count the maximum number of edges cut by partitions, for each partition $\left(S_i, \overline{S}_i\right)$, we iterate all edges in E, and check if it is being cut in constant time. Thus, for each portion, it takes $|E|$ iterations and $\mathcal{O}(1)$ time in each iterations, which is $\mathcal{O}(|E|)$ in total. And since there are k partitions, each takes $\mathcal{O}(|E|)$ time, the total runtime is $\mathcal{O}(k|E|)$, which is polynomial. After counting the maximum number of edges cut by $\{S_i, \overline{S_i}\}_{i=1}^{k}$, let's call it max, if $max \leqslant C$, answer "Yes"; "No" otherwise.
Runtime: As discussed in certifier section, the runtime is $\mathcal{O}(k|E|)$, which is polynomial.
Thus, DECISION is NP.

Problem 3

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.[a] SOKOBAN is a fun[b] 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.
2. What would be a natural certificate for SOKOBAN instances?
3. What would be a natural certifier for SOKOBAN?
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?

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

Solution

(1) DECISION: Given a 2-dimensional grid with squares, walls, and boxes, does there exist a sequence of movements with at most k steps such that moves the boxes to a given destination while obeying the moving rules in SOKOBAN?

(2) **Certifier**: A sequence of movements.

(3) **Certifier**:

(1) If all the movements are legal in SOKOBAN, i.e not pushing multiple boxes, crossing walls, or move boxes in other directions.

(2) If the boxes have all being put to the destination.

(3) If the sequence of movements have at most k steps.

(4) Because we haven't prove that the certifier can be rounded in polynomial time, i.e, haven't proved that we have an efficient certifier.

Problem 4

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 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.
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.)

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.

Proof of $4\,(a)$. **Certificate**: A quadruple $\left(P, \hat{x}, \hat{t}, \hat{z}\right)$ and a binary string y.
Certifier: If $|y| > |\hat{z}|$, directly return "No". Otherwise, run $P\left(\hat{x}, y\right)$ for $|\hat{t}|$ steps, if it does not terminate or returns "No", returns "No". Otherwise ($P\left(\hat{x}, y\right)$ returns "Yes" in $|\hat{t}|$ steps), returns "Yes".
Runtime: The certifier will run for at most $|\hat{t}|$ times, and in each iteration is constant operation, thus the whole process has $\mathcal{O}\left(\hat{t}\right)$ runtime, which is polynomial.
Thus, CERTIFICATION is in NP. □

Proof of $4\,(b)$. The main idea of this proof is that we are trying to make CERTIFICATION as the certifier for every single problem $X \in$NP.
By the definition of NP, for every $X \in NP$, X has a certified C runs in polynomial time, which takes at most $p\left(n\right)$ where p is a polynomial function and n is the input size. Also, there exists another polynomial function $r\left(n\right)$ casts a boundary on the size of the input, makes size of certificate $|y| \leqslant r\left(|x|\right)$, where x is the size of input. Thus, we could directly reduce all X into the problem CERTIFICATION$(C, x, r\left(|x|\right), p\left(|x|\right))$. All of the input are valid and it takes constant time to calculate $r\left(|x|\right)$ and $p\left(|x|\right)$. x will be certificated as "Yes" if and only if CERTIFICATION$(C, x, r\left(|x|\right), p\left(|x|\right))$ returns "Yes".
Thus, $X \leqslant_p$ CERTIFICATION. □