Assignment 9

CSCI 270

Paul Kim | pbkim@usc.edu | 1723717002

Question 1

(1) [5+10=15 points]
To justify our focus on decision problems in studying the P vs. NP question, 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 SET COVER problem. Specifically, we will consider three closely related problems.

Decision Given a universe U of n elements, a collection of m subsets $S_1, \ldots, S_m$ of U (with $\bigcup_{i=1}^{m} S_i = U$), and an integer k, is there a collection of at most k of these subsets whose union is U?

Optimization Given a universe U of n elements, and a collection of m subsets $S_1, \ldots, S_m$ of U (with $\bigcup_{i=1}^{m} S_i = U$), what is the smallest k such that there is a collection of at most k subsets whose union is U?

Search Given a universe U of n elements, and a collection of m subsets $S_1, \ldots, S_m$ of U (with $\bigcup_{i=1}^{m} S_i = U$), find a smallest collection of subsets from $S_1, \ldots, S_m$ whose union is U.

(a) Show that the decision and optimization versions of SET COVER are polynomial-time equivalent. That is, either both are solvable in polynomial time or neither is.

(b) Show that the optimization and search versions of SET COVER are polynomial-time equivalent. That is, either both are solvable in polynomial time or neither is.
[Hint: The reduction from optimization to search is quite simple. The reduction from search to optimization is more involved, and involves *multiple* calls to a black box for solving the optimization problem, on different inputs. In this sense, it is different from the Karp reductions we are using for NP-hardness proofs. 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.]

Answer:

1. **Problem**: Show that the **Decision** and **Optimization** versions of **SET COVER** are poly-time equivalent.

   *Show that decision is at least as hard as optimization and vice versa.

   *A and B are polynomial time equivalent if: A <= (p) B and, B <= (p) A. This means that B is as hard A and vice versa.

Proof:

Main Intuition: If you can solve optimization in poly time, then you can do decision in poly time.

Prove: Decision Problem => Optimization Problem and Optimization Problem => Decision Problem.

In the decision version, we are given a K value to say True/False to say whether there exists a collection of subsets where at most K of these subsets unions is U. For this problem, we can just simply check in polynomial time whether there is a collection of at most K > 0 of these subsets whose union is U by using for loops and to determine if the sets chosen fulfills the condition.

The reduction we can apply to here, for the sake of the optimization problem, is that we can run the decision problem from K starting from 1 (K > 0), and keep on running it until we capture the smallest/first value of K from this order. Evidently, by our algorithm, it will say yes if K is a number such that there is a subset whose union is U.

What the optimization problem entails, is that we run a decision problem on every size until we capture the first ”yes”. This first yes, will be the smallest K value that satisfies the optimization problem.

Example: (Arbitrary Example)

When K=1... say no, when K=2... say no, when K=3, say yes. At this point K=3 must be the smallest K value, since it is of integer value, and the first one that satisfies the decision problem. (Basically we keep running decision, until we get a yes for the smallest K).

We take this solution, and if we arbitrarily choose a K that is smaller, then it is impossible (because we determined that K=3 is the smallest). (*Arbitrary value... just to show an example).

Therefore, by using what we can find using the Decision Problem by iterating K > 0, to find the smallest K in poly-time. We thereby also find the smallest K solution that satisfies the Optimization problem. If we decide to run K’ on the optimization problem where K’ is valid and K’ <= K, we know that K subsets will contain all the subsets in U. If K’ > K, then we return false since K’ is not the smallest value (as we determined there is another smaller K that satisfies the decision problem).

Runtime: We can see that both the decision problem and optimization problem are polynomial-time equivalent as we are just using loops (it is efficient).

2. **Problem**: Show that the **Optimization** and **Search** versions of SET COVER are polynomial-time equivalent.

   Main Intuition: When you have a result (a set), you just calculate how many elements are in the set, and the number of subsets in this set is the answer to the optimization. (Assume you call search and get the answer, then you get the smallest possible collection of sets, then return the size of that collection, which answers the optimization problem).

   Prove: Optimization Problem => Search Problem and Search Problem => Optimization Problem.

We can find the solution to the optimization decision problem on all available sets (at first which is all the subsets). From here, we can obtain the smallest number of collections K from what we have. From here we can run again on the available sets - S(i), to get K'. If K' > K, then removing S(i) will lower the optimal value, so that means we should keep the set.

If K'=K, then removing S(i) from the list of sets will not change the optimal value, so we are able to remove S(i) from the available set. What we are doing here, is we are attempting to remove all the irrelevant sets while keeping the solution optimal. Then in the end, we have the solution to the search problem, which is the smallest collection of subsets whose union is U.

*Basically attempt to remove from the set if still optimal.

Runtime: We can see that both the optimization problem and search problem are polynomial-time equivalent because again we are just using for loops to check conditions. (It is efficient).

Question 2

(2) [3+3=6 points]

(a) You are given a social network on n people. Each pair u, v of people either hate each other or are good friends — for this problem, there is no in-between, such as not knowing each other, or feeling neutral. You want to determine the largest set of people who either *all* hate each other, or are *all* friends with each other.

Phrase this problem as a decision problem (so as to capture the "essential" part of the problem), and prove that that decision problem is in NP.

[Note: you do not need to prove NP-hardness of this problem.]

(b) Prove that the following problem is in NP. Given two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, are they the same up to relabeling the vertices? That is, is there a function $f : V_1 \to V_2$ mapping vertices of V_1 to vertices of V_2 such that $f(u) \neq f(v)$ whenever $u \neq v$, and for all node pairs $u, v \in V_1$, the edge $(u, v) \in E_1$ if and only if the edge $(f(u), f(v)) \in E_2$?

[Note: again, you do not need to prove NP-hardness. In fact, if you do manage to prove that this problem is NP-hard, please let us know, since it would resolve one of the biggest open questions in CS — and it seems at this point more likely that the problem is polynomial-time solvable, or somewhere in the middle (not in P, but not NP-hard, either).]

Answer:

1. Problem A.)

   Decision Problem: Given pairs of people (u, v), determine how many possible pairs exist from N people, that are friends with each other, or hate each other.

   Proof:

   Certificate: The certificate is a proposed assignment of T/F to a pairing of U and V. The certifier easily looks at the two pairs of people and evaluates whether they are friends, or enemies.

   Runtime: This clearly runs in polynomial time where we can have a double for-loop that checks every combination of a pairing. (It is efficient).

2. Problem B.)

   Decision Problem: Given two graphs, is there a function that maps V1 to V2, mapping the vertices such that they are the same up to relabeling the vertices.

   Intuition: We can see 2 constraints on F. First: F(U) is not F(V), which means that there can't be two different nodes in V1 that maps to the same vertex in V2. If there's an edge in the original graph, then the mapping function must preserve given that U and V have an edge between then.

   Proof:

   Certificate: A proposed mapping F. The certifier verifies that F is a correct mapping by going through all pairs (U,V) in V1 to check the conditions above, returns true if F is a valid solution and false if not.

   Runtime: This clearly runs in polynomial time where we can have a double for-loop that checks for each U, every possible V (checks every combination of a pairing). (It is efficient)

Question 3

(3) [2+8=10 points]
Imagine that you are trying to build yourself a (digital) movie collection. Since you are on a budget, you want to buy several bundles[2] of movies that are being offered. For example, there may be offers of an "essential comedy" bundle {Office Space, Princess Bride, Hunt for the Wilderpeople, Dr. Strangelove, Modern Times}, another bundle of "Geek movies" {Office Space, The Matrix, Labyrinth, Travelling Salesman}, and another for "classic movies" {Dr. Strangelove, Modern Times, M, Casablanca}. Each of these bundles has a known price, and you have a known budget. For each of the movies, you know how much you would like it as part of your collection. Notice that getting multiple copies of a movie is no better than getting just one copy.

We model this as follows. There is a set M of all m movies. For each movie i, there is a value $v_i \geq 0$ of how much you would like that movie in your collection. There are n movie bundles $B_1, \ldots, B_n$. For each bundle j, you know the price $p_j \geq 0$ for getting that bundle. You have a budget of q to spend on movie bundles in total, and your goal is to get as much value as possible from the movies you get in this way. To turn it into a decision problem, in addition to your budget q, you now have a target total value V, so that if you select that set S of bundles, the total movie value you get, $\sum_{i \in \bigcup_{j \in S} B_j} v_i$, is at least V.

(a) Prove that the decision version of this problem is in NP.

(b) Prove that this decision problem is NP-hard. (You can use any of the problems we mentioned as NP-hard in the class; we recommend SET COVER (see above).)

Answer:

There is a set M of all m movies. For each movie i, there is a value V(i) >= 0 of how much you desire that movie. There are n movie bundles. For each bundle j, there is a price P(j) >= 0 for getting that bundle. Goal is to get as much value as possible from the movies without going over a budget Q. However, we now have a target total value V, so that if you select that set S of bundles, the total movie value you get is, (formula), is at least V.

1. Prove that the decision version of the problem is in NP.

Decision Problem: Given in the prompt and stated above.

Certificate: The set S of bundle you pick. The certifier loops through all the items in the bundle, adds the cost of all the bundles together to make sure the cost doesn't exceed Q (budget), and loops through all the movies and adding V(i), and as soon as you hit V you stop. If both conditions are met, you certify the certificate as true, and if not, false.

*Take a set of bundles and take it is within budget and the value is at least B(all).

Runtime: Clearly polynomial time as we are just using loops to sum values. (It is efficient)

2. Prove that the decision problem is NP-Hard.

Reduction from Set Cover to Movie Problem.

Prove: Set-Cover <=(P) Movie (*Show that the movie problem is at least as hard as Set-Cover. Model the problem in such a way.)

Set Cover:

Given a set U of elements, a collection (S1, S2,... Sm) subsets of U, and an integer K, does there exist a collection of at most K of these sets whose union is equal to U. -> **Set Cover is NP-Complete (Theorem)**

Proof:

-> We proved that the decision problem is in NP. (With the appropriate certification and certifiers).

-> We know that Set Cover is NP-Complete.

*Office Hours Notes

Reduction from Set Cover to Movie Problem:

Goal is to find a reduction F that takes the inputs of Set Cover (Collection of M sets, integer K, and set X) to output the inputs of the Movie Problem (Movie Bundle that maximizes the value (Price, Value, integer Q that denotes budget)).

Intuition: We can use Set-Cover (U,K) and its preprocessing to create a movie bundle that maximizes the value as desired in the prompt. Intuitively, if Movie(B,V,Q) satisfies the given condition, accept, otherwise reject.

Informal Algorithm:

For every element i in U: Convert i into a movie with value V(i) = 1.

For every set S(j) in U: Create a bundle B(j) with the same elements as S(j) with price P(j) equal to 1,

Set Budget Q to integer K.

*Make every element in U a movie, give each movie a value of 1, and then define the bundle.

Using the decision problem, we can denote a bundle as a set of movies where each individual movies have a value, and bundles that contain some amount of movies have a price. Therefore we can modify our input from U -> S. We can modify our new set as the set of bundles (B1, B2... Bm) as synonymous to the collection of M sets for Set Cover. And in each B1 through

Bm, a bundle carries a (B,V,Q) values where B is the price, V is the value, and Q is the capacity.

We set V (the value) to M where M is the number of elements in U from Set Cover (in this case, we have M total movies in the bundle). We assign V(i) = 1 for all movies. By doing this, we can set budget Q = K, and we make every movie to have a value of 1, and price of 1, and they must all be equal because we want them all to fulfill the condition. Since the budget is Q, we can cover all elements with K under Q, and where V is M.

Prove: Set Cover -> Movie Bundle

Intuition: If we have a collection of sets and we map the set to a bundle, it should be easy to check if it satisfies all conditions. (Movie bundle restrictions).

Suppose (U,K) is a Yes instance to Set Cover. Thus there exist a collection of subsets of U, and an integer K, such that at most K of these sets union is equal to U. If we assume that F is satisfied and we get a set of these subsets, then we can directly convert this into a set of bundles and check directly that the output meets all the conditions of the Movie Bundle decision problem.

If we assume that the Set Cover is correct, then it means that Movie Bundle is correct as well since we can directly check.

Prove: Movie Bundle - >Set Cover

Intuition: If there is a valid movie bundle, you have a valid set cover.

Assume we have a set of bundles that satisfy all criteria, and show that we have a set of subsets of size less than equal to K that covers U. The proof here is that if there exists this set of bundles that satisfies the decision problem that obtains the target value of at least V, and satisfies the budget, then that means that there is exist a set of subsets that satisfies Set Cover.

Runtime: F runs in polynomial time. This is obvious but for the sake of formality, we can just for loops to check if all the conditions are satisfied in both cases.