Bo Yang (Aaron) USCID: 7526922531

USC CSCI 570 Homework6 Date: 2021-04-16

1. Integer Programming (20 Points)

In Linear Programming, variables are allowed to be real numbers. Consider that you are restricting variables to be only integers, keeping everything else the same. This is called Integer Programming. Integer Programming is nothing but a Linear Programming with the added constraint that variables be integers. Prove that integer programming is $NP-Hard$ by reduction from SAT.

Solution:

>

> When we are asked to prove that a problem is $NP-hard$, what we usually do is that to find an NPC problem that has already been proven and to reduce this NPC problem to that problem (i.e. $NPC <=_p NP-hard$)
>
> 1. Since the $3-SAT$ problem is well know NPC problem. So I decided to prove ILP (integer linear programming) is $NP-hard$ and reduction from $3-SAT$.
> 2. Consider a $3-SAT$ expression with variable $x_1, x_2, \ldots, x_n$.
> More specifically, those clauses are shown like this: $(x_1 \vee \bar{x_2} \vee \bar{x_3}) \wedge (x_2 \vee \bar{x_4} \vee x_5)\ldots$
> Next, create an ILP instance with variable $z_1, z_2, \ldots, z_n$. And then convert each clause into an equation as follows:
> $$z_1 + (1 - z_2) + (1 - z_3) \leq 0$$
> $$z_2 + (1 - z_4) + z_5 \leq 0$$
> $$\ldots.$$
> $$z_i \in \{0, 1\}, [i = 1, 2, 3..., n]$$
> In this way, we can convert an $3-SAT$ problem into a ILP problem. And a feasible solution of this ILP problem means a satisfying truth assignment of the $3-SAT$.
>
> So, we can say, ILP (integer linear programming) is $NP-hard$ and can be reduce from $3-SAT$ problem.

2. HALF-SAT (20 points)

We know that the SAT problem is $NP-complete$. Consider another variant of the SAT problem: given a CNF formula F, does there exist a satisfying assignment in which exactly half the variables are true? Let us call this problem $HALF-SAT$. Prove that $HALF-SAT$ is $NP-complete$.

Solution:

>

> Consider assignment function f is an instance of a SAT problem with variables $x_1, \ldots, x_n$.
>
> Then we create a instance f' of $HALF-SAT$. Except those original variables $x_1, \ldots, x_n$, we continue add $y_1, \ldots, y_n$. And we denote each clause in f is called C, and in other arragement f', the new clause is callled C', where the C' is obtained from C by replacing literals x_i with $\neg y_i$. More specifically, if $C = (\neg x_1 \vee x_2)$, then $C' = (y_1 \vee \neg y_2)$.
>
> If we can prove that f has a satisfying assignment if and only if f' has a satisfying assignment in which exactly half the variables are $True$, then we will have reduced SAT to $HALF-SAT$, proving that the latter problem ($HALF-SAT$) is $NP-complete$.
>
> First suppose that f' has the required assignment. Then the same assignment, restricted to the variables $x_1, \ldots, x_n$ is a satisfying assignment for f (since every clause in f is also in f').

variables $x_1, \ldots, x_n$, is a satisfying assignment for f (since every clause in f is also in f').

Now suppose that f has a satisfying assignment. We extend this by letting y_i be $True$ if x_i is $False$ and vice versa. It is straightforward to check that if it has a satisfying assignment for f' and that exactly half the variables are $True$.

3. Taking courses (20 points)

There is a set of courses, each of them is represented by a set of disjoint time intervals with the starting and finishing times. For example, a course could require the time from 9am to 11am and 2pm to 3pm and 4pm to 5pm. You want to know, given a number K, if it's possible to take at least K courses. You cannot choose any two overlapping courses. Prove that the problem is $NP-complete$, which means that choosing courses is indeed a difficult thing in our life. Use a reduction from the Independent set problem.

Solution:

>

1. Showing this interval scheduling (IS) problem $\in NP$.
   A course arrangement will be a certificate. For each courses arrangement, a can be computed by a greedy algorithm,
   And then iterate the all arrangements and check the length of the assigned course set is bigger than K or not.
   Thus, the certifier could be run in polynomial time.

1. Showing this IS is a $NP-hard$ problem.
   We can show this with a reduction from Independent set problem.
   Consider an IS instance graph $G=(V,E)$. Given a set of intervals (course time slot), we construct a graph H with the given intervals as nodes, where two nodes are adjacent whenever the represented intervals are intersecting.
   We show that graph G contains an independent set of size at least K if and only if H has a possiblity to take at least K courses.
   To prove the first direction, suppose there is an independent set whose size is more than K in graph G. Since, independent set is a subset of vertices and no any two of them are joined by an edge. As we said before, we will connect two nodes when their interval are intersecting. Hence, when there are K independent nodes means there are K not overlapping courses, so the alogorithm will outputs 'Yes'.
   For the other direction, suppose in graph H, student can take at least K courses without violate the rules. That also means there will be at least K nodes won't connect each other directly, and these vertices could form an independent set.

Since builting a graph is done in polynoimal time, the reduction is polynoimal time as well.

4. Approximation 1 (20 points)

It is well-known that planar graphs are 4-colorable. However finding a vertex cover on planar graphs is $NP-hard$. Design an approximation algorithm to solve the vertex cover problem on planar graph. Prove your algorithm approximation ratio.

Solution:

>

Consider a planar graph $G=(V,E)$, and color it in 4 colors.
In this way, there will split all vertices and generate four color groups. lets say, C_1, C_2, C_3 and C_4.
And then, we can find the biggest group, for example, C_1. The size of C_1, S_1, must bigger than $\frac{|V|}{4}$.
If we remove the largest group, and then use rest of those vertices as a vertex cover.

Since there won't have any two adjacent nodes with the same color, the rest of vertices must be able to cover all the edges, which also means the algorithm's output is correct.

Since the size of rest vertices is at most $|V| - S_1 \leq |V| - \frac{|V|}{4} \leq \frac{3|V|}{4}$, the algorithm approximation ratio will be $\frac{4}{3}$.

5. Approximation 2 (20 points)

Consider the following heuristic to compute a minimum vertex cover of a connected graph G. Pick an arbitrary vertex as the root and perform depth first search. Output the set of non-leaf vertices in the resulting depth first search tree.

- 1. Show that the output is indeed a vertex cover for G.
- 1. How good is this approximation? That is, upper bound the ratio of the number of vertices in the output to the number of vertices in a minimum vertex

Solution:

>

In a given graph $G = (V, E)$, we say T is the resulting DFS tree, and A is a set which contains a bunch of leaf nodes which appeared in DFS tree T. And the result, $B = V - A$ represents the set of non-leaf vertices.

1. Lets assume there exists an edge $e = (u, v)$ whose vertices are not covered by B, implying that both u and v should and will exist in A. If that DFS firstly explored u, then DFS would have left u, and continue to explore a new vertex, thereby making u a non-leaf node. This is a contradiction with our assumption. So the result B is indeed a vertex cover for G.

1. Next, we can create a matching instance from the DFS tree T,

   And for every vertex in set B, pick one edge to its descendant in the tree. Consider all edges of vertices in B, some of them are in odd levels, so we called it M_{odd} matching. Also, for those nodes in set B and in even levels, we named it M_{even} matching.

   Picking the maximial matching from these two instance $M = max(M_{odd}, M_{even})$. Since the total number of non leaf vertices in the DFS tree T is $|B|$, so the M contains at least $\frac{|B|}{2}$ vertices. In order to cover every edge in M, the optimal vertex cover S has to contain at least $|B|$ vertices. Hence our solution is at worst a 2-approximation.