CSCI 570 – Homework 6
Due: April 29th, 2021

Name: Rohan Mahendra Chaudhari
USC ID: 6675-5653-85
Email ID: rmchaudh@usc.edu

1. 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:

   a. Prove IP is NP:

      Certificate: Given a solution S with values for all variables of the ILP.

      Certifier: Given the solution S we can run the certifier to verify if the inequalities are satisfied for the given values of the variables.

      Thus the IP problem is NP and the certifier runs in polynomial time.

   b. Prove IP is NP Hard:

      Proof by reduction from SAT.

      A SAT has Boolean formula made of clauses and literals. Our Integer Programming problem will contain twice the variables/literals that SAT contains (the literal itself and its negation).

      Construction:
      Inequalities:
      for each clause $C = \{a_1, \sim a_2, \ldots, a_i\}$: $a_1 + \sim a_2 + \ldots + a_i >= 1$
      $0 <= a_i <= 1$ and $0 <= \sim a_i <= 1$ (Integer valued a_i and $\sim a_i$)
      $a_i + \sim a_i = 1$

      Claim: A SAT is satisfiable if and only if the IP has an integer solution.

=>) Any SAT problem has a solution in IP.

A True literal in a SAT solution corresponds to 1 in the Integer Programming problem. Thus, if the corresponding SAT is satisfied then at least one literal per clause must be True therefore making the sum of the inequality >= 1.

<=) Any IP solution gives a SAT solution.

Given an integer IP solution, set the literals to 1 for True and 0 for False. Following this format, no literal and its complement will have the same value and so it is a legal assignment which must also satisfy the clauses.

Since Satisfiability is NP-Complete, Integer Programming is NP-Hard/NP-Complete and the reduction is polynomial.

2. 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:

   a. To show HALF-SAT is NP:

      Certificate: A solution S with truth values for all variables.
      Certifier: Traverse the HALF-SAT by inserting the corresponding values and verify if the HALF-SAT is satisfiable for the given input and if half of the input variables hold a value of True.
      Thus the HALF-SAT problem is NP and the certifier runs in Polynomial time.

   b. To show HALF-SAT is NP-Hard:

      Reduction from SAT to HALF-SAT

      Construction:
      Let ‘T’ be a Turing machine that reduces SAT to HALF-SAT.
      T takes I as input.
      Let I have variables $x_1,\ldots,x_n$

T(I) = Construct a new CNF formula I' such that I' will have variables x_1…..x_n from I and new variables y_1….y_n such that y_i = ~x_i.

If a clause is of the form (x_1 v x_2), convert it to (x_1 v x_2) ^ (y_1 v y_2)

The output would be I'

Example:

Given I = (x_1 v x_2 v ~x_3) and x_1 = T, x_2 = F and x_3 = T
I = T v F v F = T
Introduce 3 new variables y_1,y_2,y_3 such that,
y_1 = ~x_1
y_2 = ~x_2
y_3 = ~x_3

I' = (x_1 v x_2 v ~x_3) ^ (y_1 v y_2 v y_3)
x_1 = T y_1 = F
x_2 = F y_2 = T
x_3 = T y_3 = F

Assigning values to I', I' = (T v F v F) ^ (T v F v T) = True

Here 3 variables are assigned True and other 3 are assigned False. Therefore the expression is in HALF-SAT.
Hence I' evaluates to True if and only if I evaluates to True.

Let x_1 = F, x_2 = F, x_3 = T
I = F v F v F = F
I' = (F v F v F) ^ (T v T v T) = F
Hence I' evaluates to False if and only if I evaluates to False.

Claim: A satisfiable SAT exists if and only if the HALF-SAT is satisfiable and exactly half of the variables are True.

=>) Given a SAT and a truth assignment for which the SAT is satisfiable, we add the disjunction of complement of all the literals present in I to form I'. Thus half of the literals have a value True and also the truth assignment that satisfies SAT will satisfy HALF-SAT as well.

<=) Given a HALF SAT, we can obtain a satisfiable SAT by removing the clause that contains the disjunction of the complements, i.e. removing (y_1 v y_2 v y_3) = (~x_1 v ~x_2 v x_3). This will help us obtain the original SAT which will be satisfiable since the HALF-SAT will also be satisfiable.

Hence HALF-SAT is NP-Complete and the reduction is polynomial.

3. 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:

To prove the course scheduling problem is NP complete:

a. To show the course selection problem is NP
Certificate: A non-conflicting schedule of at least k courses.
Certifier: To verify the above certificate, we first have to check the number of courses in the schedule. After this, we check if at most one course is running in each interval. Finally, we check if all courses in the schedule get all the intervals they require. This verification can be implemented in O(mn) time, where 'n' is the number of courses and 'm' is the number of intervals.

Thus the course scheduling problem is NP and the certifier runs in polynomial time.

b. To show the course selection problem is NP-Hard
This is done by reduction from Independent set (IS).

Construction:
Given an instance C of IS consisting of a graph G(V,E) and a number k, we construct an instance H of the course selection problem as follows:
For each vertex v_i belong to V, we construct a course c_i. For each edge (v_i,v_j) that belongs to E, we construct an interval t_{ij} and let c_i and c_j require the interval t_{ij}. Finally the number k is copied from C

to H. It is clear that this reduction can be performed in polynomial time.

Claim: Graph G has an independent set of size k if and only if there is a non-conflicting schedule that completes k courses.

Proof:
=>) Assume that G contains an independent set $\{v_{i1},v_{i2},\ldots,v_{ik}\}$. Then, we can schedule the corresponding k courses $c_{i1},c_{i2},\ldots,c_{ik}$, since no two of them require a common interval.

<=) Assume that there is a non-conflicting schedule of k courses $c_{i1},c_{i2},\ldots,c_{ik}$. Then the set $\{ v_{i1},v_{i2},\ldots,v_{ik}\}$ is an independent set of G; for otherwise, there must be some h and l such that (v_{ih},v_{il}) belongs to E, which means both c_{ih} and c_{il} require the interval t_{ihil}, contradicting the fact that c_{ih} and c_{il} are not conflicting.

Thus, the course scheduling problem is NP-complete and the reduction is polynomial.

4. 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:

Algorithm:
The algorithm starts by 4-coloring the graph. Next we choose a particular colour which has the most fractional vertices, let this colour be green. Then we assign a value of 0, to all the fractional vertices that have been assigned the colour green and 1 to the rest of the fractional vertices. Note that, no edge can have 2 green vertices incident on it by the property of 4-coloring a graph. Thus, all edges would still be covered. At most ¾ of the fractional vertices are not green, so we go from half-using each vertex to using ¾ of them, therefore multiplying the optimal value by at most 3/2.
Thus the approximation ratio is 3/2.

Proof:
It takes polynomial time to 4-color any planar graph. Thus we can use the fact that there is a polynomial-time algorithm to 4-color any planar graph and also the fact that linear program solvers return extreme points in polynomial time.

In general, the idea of linear program-based approximation algorithms is that you want to start with the solution to the vertex cover linear program which assigns an element of the set {0, ½ ,1} to every vertex, and turn it into an integer solution without changing the objective value too much.

Since the optimal value of the linear relaxation is at most the optimal value of the actual vertex cover problem, if you don't deviate from that optimal value by more than a factor of 3/2, you have a 3/2 approximation algorithm.

In this case, the linear program solution assigns some vertices a value of 0 or 1 which do not require any change and other vertices value ½ which may require some changes. In the worst case, we have n vertices with a value ½ , for objective value (½)n.

For a 3/2 approximation, we are allowed to go up to objective value (¾)n : giving at most ¾ of the fractional vertices value 1, and the rest value 0.

5. 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.
    a. Show that the output is indeed a vertex cover for G.
    b. 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:

Let G = (V,E) denote the graph, T = (V,E') denote the depth first search tree, L the set of leaf vertices and NL = V - L the set of non-leaf vertices.

a. Assume an edge e = (x,y) exists in E that NL does not cover and thus we can say that both x and y are leaf nodes and will belong to set L. On running DFS, assume we explore ‘x’ first but since we already have ‘e’, the DFS run would have left ‘x’ and explored a new vertex making ‘x’ a non-leaf, this would make our assumption incorrect and thus N does indeed cover every edge in E.

b. The next step is to create a matching M in G from the structure of T. For every vertex ‘x’ in N, we pick one edge that connects ‘x’ to one of its descendants in T. We call this edge e_x. Let the set of even level non-leaf vertices be called ‘even’ and the set of odd level non-leaf vertices be called ‘odd’. If the set ‘even’ is bigger than set ‘odd’, we set ‘upper bound’ = ‘even’ else set ‘upper bound’ = ‘odd’. Since the total number of non-leaf vertices in T is |N|, the set ‘upper bound’ has a size of at least |N|/2. In this case, the edge set M = {e_x|x belong to ‘upper bound’} is a matching of size at least |N|/2. Since M is a matching, to cover each edge in M the optimal vertex cover VC has to contain at least |M| vertices. Thus, VC contains at least |N|/2 vertices while our solution has |N| vertices.

   Hence our solution is at worst a 2-approximation.
   Example:
   E = {(x,y), (y,z)} with DFS rooted at x.