dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

CSCI570 Homework 6

1. **Solution:**
   - We first show that ILP (Integer Linear Programming problem) belongs in NP.
   - This can be seen easily by the fact that given a solution for the ILP we can substitute the values of the variables in the inequalities given to us and verify our solution in polynomial time. Hence the ILP solution is polynomial time verifiable and in NP

   - We have SAT as a known NP-Hard problem. Now, we convert SAT problem into an instance of ILP problem i.e., $SAT \leq_p ILP$. We do this by saying that every clause in our SAT is an inequality and the literals are the variables in our ILP.
   - Another constraint we place on our literals is ($1 \leq x_i + x_i' \leq 1$) & ($0 \leq x_i \leq 1$) and literals are integers
   - Let us say that for example one of the clauses in SAT is ($x_1 \vee x_2 \vee x_4 \vee x_1'$) then the inequality associated with such a clause would be $x_1 + x_2 + x_4 + x_1' \geq 1$. The right-hand side and the sign are same for all the inequalities.

Claim: SAT instance with k clauses is satisfiable if and only if the ILP with k inequalities with the given constraints has a feasible solution.

Proof: We must show the implication in both directions

==> Assume that we have a truth assignment for the SAT. This means that every clause in the SAT is True. This means that at least one of the literals in each of the clauses is true. If we have this then we know that the inequalities laid out by us are satisfied. For example, if ($x_1 \vee x_2 \vee x_4 \vee x_1'$) is True then ($x_1 + x_2 + x_4 + x_1' \geq 1$) is also satisfied.

<==) Assume that we have satisfied all the inequalities in our ILP. This means that in every inequality we have at least one variable that is 1. This implies that every clause associated with the inequality is also satisfied as we need at least one truth assignment per clause.

- Since, solving a small instance of ILP is as hard as SAT, we can say that $SAT \leq_p ILP$. Hence, we have proved that Integer Linear Programming problem is $NP - Complete$

2. **Solution:**
   - We first show that the given problem belongs in NP
   - This can be easily seen by the fact that if we are given a solution for the problem, we can verify the solution in polynomial time by counting the number of cities and checking if they are equal to K and if they are connected and the total number of roads between these cities is greater than or equal to M.

   - For an undirected graph G= (E, V), where V is the set of vertices and E is the set of all edges, we take a compliment of this graph and name it G’
   - Now we take an independent set from graph G with K vertices from a subset of V such that no two vertices in K are adjacent.
   - The complement of G, G’ will have independent set vertices as the adjacent node.
   - Therefore, the total number of roads between the K cities is equal to the sum of the degrees of vertices outside the subset. This will be equal or larger than M. Hence, we will have roads M =K(K-1)/2

Claim: The Graph G contains an independent set of size k, if and only if our constructed graph contains a subset of cities of size k with at least M edges between them

Proof:

==> Assume that G contains independent set of size K. There is a set of K vertices without any edges between them. This corresponds to the k cities in our G’. This means that the number of roads between these cities is equal to the number of pairs of cities that are not connected. This is equal to $k(k-1)/2$. As we know that M $\leq K(K-1)/2$, this subset satisfies the condition of our given problem.

<==) Assume our constructed graph contains a subset of K cities with at least M roads between them. Now we can map these K cities back to K vertices in G' such that every edge in G' is covered by at least one of the K vertices. As each city corresponds to a vertex in G and each road corresponds to the absence of an edge in G, this subset of cities corresponds to an independent set of size K in G. If two cities in the subset are connected by a road, then the corresponding vertices in G cannot both be in the independent set, since that would violate the definition of an independent set.

Hence, the K vertices in G form an independent set of size K, which satisfies the conditions of the Independent Set problem.

- Since solving a small instance of the city problem is as hard as the independent set problem. We say that $independent\ set \leq city\ problem$.
- Hence city problem is NP- complete as it is both NP hard and in NP.

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

3. **Solution:**
   - We first show that SAT' problem belongs in NP
   - If we are given a solution for the SAT', i.e., given an assignment of literals it can be verified in polynomial time that the assignment is valid and m-2 clauses are in fact satisfied. This makes the SAT' problem an NP problem.

   - We have SAT as a known NP hard problem. We now convert any given SAT into an instance of SAT' problem by adding 4 very specific clauses after the given satisfied SAT.
   - In order to do this, we add clauses with 2 new literals which are not a part of given SAT. If SAT is (x_1 V x_2 V x_3) ^ (x_4 V x_5 V x_6).....(...V x_n) then SAT' for this particular SAT is s (x_1 V x_2 V x_3) ^ (x_4 V x_5 V x_6)^.....^(...V x_n) ^ (a)^(b)^(a')^(b') with a and b as the new literals.

Claim: SAT instance with clauses is satisfiable if and only if corresponding SAT' with m-2 clauses is satisfied.

Proof: We must show the implication in both directions

==> Assume, that we have a truth assignment for SAT which implies that all the clauses are satisfied and are true. Since SAT' has extra four clauses out of which 2 would always be false, SAT' is also satisfied with m-2 clauses being true. (if a=1, b=1 then a'=0, b'=0; if a=1, b=0 then a'=0, b'=1 ; if a=0, b=1 then a'=1, b'=0)

<==) Assume that for the SAT' m-2 clauses are satisfied. Since 2 clauses out of (a)^(b)^(a')^(b') are always going to be false, we can say that the other clauses are true in SAT' and hence all the clauses in SAT are satisfied.

- Since solving a small instance of SAT' is as hard as SAT problem, $SAT \leq_p SAT'$. As SAT is a NP hard problem, SAT' is also an NP hard problem along with being in NP.
- This means that the SAT' problem is NP complete. Hence, we proved that SAT' is NP complete.

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

4. **Solution:**

- We first need to show that Longest Path problem belongs to NP
- Given a valid solution of the vertices included in the longest path, we can check if they are connected and the path length is greater than or equal to the value k in polynomial time. Since the problem is polynomial time verifiable, we say that it is in NP.

- We have a known NP hard problem Hamiltonian path problem. We need to show that $Hamiltonian\ path\ problem\ \leq_p Longest\ Path\ problem.$ We now convert any Hamiltonian Path Problem into an instance of longest path problem by considering k = V-1.

Claim: Hamiltonian path problem is satisfied if and only if a Hamiltonian path exists in the longest path problem of size k

==> Assume that Hamiltonian path exists in the graph which means that every vertex is covered once in the graph. This implies that longest path of size k exists in the graph.

<==) Assume that a longest path of size k exists in the graph. Since the number of vertices in the graph is set to be k, this longest path is also the Hamiltonian path in the problem.

- Since solving a small instance of the given problem is as hard as solving Hamiltonian path problem, we can say that $Hamiltonian\ path\ problem\ \leq_p Longest\ Path\ problem.$
- This means that longest path problem is NP hard. Since it is NP and NP hard, we can conclude that the problem is NP complete

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

5. **SOLUTION:**
    - We have been given a polynomial time algorithm that decides if a directed graph contains a Hamiltonian cycle or not. We can call this function existHamCycle() which returns a 1 or a 0 depending on the whether Hamiltonian cycle exists or not in the given graph respectively.
    - Using this we can come up an algorithm that given a directed graph that contains a Hamiltonian cycle lists a sequence of vertices (in order) that form a Hamiltonian cycle by doing the following
        a. Select an edge that belongs to the given Graph (V, E) and remove it. Let us call this graph as G'.
        b. Run the polynomial time function existHamCycle() on G'. If the edge that was removed was a part of the Hamiltonian cycle in G, it would mean that there is no Hamiltonian cycle in G' and we would get a value of 0 through the function. If we get 1 as the value then the edge was a part of the Hamiltonian cycle in G.
        c. We repeat step a & b for all the edges in G and store the edges that are returning the value 1 through the function.
        d. By doing so we can find out the edges that are a part of Hamiltonian cycle in the directed graph. By traversing through these edges, we make a list of the sequence of vertices that form the Hamiltonian cycle.

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

6. **SOLUTION:**

a. **The algorithm must terminate:**

- We know that there are finite number vertices in the graph
- We are checking if moving the vertex from one side of the cut to the other side increases the number of edges in the cut only once. This means that we do not visit a vertex again in one iteration.
- The algorithm will therefore terminate after iterating through all the vertices.

b. **When the algorithm terminates, the size of the cut is at least half of the OPT**

- Here, let the total number of edges be |E|. Let e (u, v) =1 if there exists an edge between u and v and 0 if there is no edge.
- Let the greedy approximation algorithm cut the graph into 2 sets of vertices C and D = V\C arbitrarily and let it give us a cut of size S.
- For every vertex u in C, by construction we know that $\sum_{v\in C} e(u,v) \le \sum_{v\in D} e(u,v)$
- Where LHS are edges within C and RHS are edges going across C and D.
- For all vertices in C:

$$2\sum_{(u,v)\in C} e(u,v) \le \sum_{(u\in C,v\in D)} e(u,v) = A$$

- Similarly for all nodes in D:

$$2\sum_{(u,v)\in D} e(u,v) \le \sum_{(u\in D,v\in C)} e(u,v) = A$$

- Adding both inequalities given above:

$$2\sum_{(u,v)\in C} e(u,v) + 2\sum_{(u,v)\in D} e(u,v) \le 2A$$

$$\sum_{(u,v)\in C} e(u,v) + \sum_{(u,v)\in D} e(u,v) \le A$$

- Adding cut edges to both sides:

$$\sum_{(u,v)\in C} e(u,v) + \sum_{(u,v)\in D} e(u,v) + \sum_{(u\in D,v\in C\ \ )} e(u,v) = |E| \le 2A$$

- Here, LHS is the total number of edges |E|.
- Therefore, |E|≤ 2A.
- We know that the optimal solution OPT ≤ $|E|$. Therefore, OPT ≤ |E| ≤ 2A
- $So\ ,\ \frac{OPT}{A} \le 2$
- Therefore, when the algorithm stops, the size of the cut is at least half of the optimum.

dnabar@usc.edu Deven Parag Nabar
USC ID: 7229446568

7. **SOLUTION:**

- We have a planar graph G = (V, E). We now colour it in 4 colours.
- This will all the V in 4 groups depending on the colours they are associated with and none of the vertices would be connected. Let us call them X1, X2, X3, X4.
- We now use the vertices that are not a part of the most frequent colour group to construct our vertex cover.
- The LP for the vertex cover is as follows
  Min(z= $\sum_{v\ in\ V} x_v$)
  Subject to $x_u + x_v \geq 1\ for\ all\ (u,v) in\ E$
  $0 \leq x_v \leq 1 \quad for\ all\ v\ in\ V$
- In the solution to the LP , the values of x_u can be made to be {0, 0.5, 1} which means that every vertex of LP of Vertex Cover is half integral.
- Let the set of vertices with values {0,0.5,1} be denoted by V_0, $V_{0.5}$, V_1. In the usual vertex cover, we use the sets $V_{0.5}$, V_1.
- As we have a 4 color graph, we can discard the vertices belonging to the most frequent color vertices from $V_{0.5}$.
- Let the most frequent class be X_1. This means that vertex cover is $V_{0.5}$+ $V_{0.5}$' where $V_{0.5}' = V_{0.5} - X_1$

Approximation Ratio :

- After solving the LP, let optimal value of LP be z*, optimal vertex cover be $|VC^*| \geq z*$
- We know that $|V^1{}_{0.5}| \geq |V_{0.5}|/4$ as the first group is the most frequent color.
- $|V'_{0.5}| \leq 3\ |V_{0.5}|/4$
- Hence, we have $|VC| \leq |V_1| + 3/4 * |V_{0.5}| \leq\ |V_1| + 3/4 * |2(z^* - |V_1|) \leq 1.5|VC^*|$
- This gives an approximation ratio of 3/2 = 1.5

dnabar@usc.edu

Deven Parag Nabar
USC ID: 7229446568

8. SOLUTION:

- To prove that the given greedy algorithm for 0-1 Knapsack Problem is pretty bad which means it does not provide a constant approximation.
- We have been given a knapsack capacity of W
- If we have been given two items: item 1 of value 2 and weight 1
  & item 2 of value W and weight W
  … W>2
- The greedy algorithm ALG will always choose item 1 as the value/weight is 2. After this we cannot choose item 2 as the knapsack capacity has become W-1 and cannot accommodate item 2 in it.
- The Optimal solution is the one where item 2 is chosen first as it maximizes the value as W.
- Now since 0-1 Knapsack problem is a maximization problem,
  $\frac{ALG}{OPT} \geq \propto$

  Here $\propto \leq 2/W$
- As the value of W increases, ∝ becomes arbitrarily small and hence it doesn't provide a constant approximation.