CS 170 HW 12

Due **2019-11-20, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group.

2 Dominating Set

A dominating set of a graph $G = (V, E)$ is a subset D of V, such that every vertex not in D is a neighbor of at least one vertex in D. Let the Minimum Dominating Set problem be the task of determining whether there is a dominating set of size $\leq k$. Show that the Minimum Dominating Set problem is NP-hard. You may assume that G is connected.

Solution:
We will reduce the Minimum Set Cover problem to the Minimum Dominating Set problem. Suppose (S, U) is an instance of set cover where U denotes the set of all distinct elements and S is a set of subsets S_i of U. We will construct a graph $G = (V, E)$ as follows. For each element u in U construct a vertex; we will call these "element vertices". For each possible S_i construct a vertex; we will call these "set vertices". Connect each vertex S_i to all u in S_i.

Notice that if we were to run dominating set on the graph right now, we would be able to cover all the element vertices with any valid set cover, but we would have to pick every single set vertex in order to ensure that all set vertices are covered. To rectify this, connect every set vertex to every other set vertex, forming a clique. This ensures that we can cover all the set vertices by picking just one. This way we really only need to worry about covering the element vertices.

Suppose we have some minimum set cover of size k. This will correspond to a dominating set of size k as well. For each set in our Minimum set cover, pick the corresponding set vertex. It follows directly from the construction of the graph and the definition of a set cover that all set and element vertices are covered.

Suppose we have some dominating set D of size k. We can find a set cover of size $\leq k$. To do this, we will construct a new dominating set D' that contains only set vertices. Include every set vertex in D in D'. For each vertex in our dominating set that is an element vertex, pick any random neighboring set vertex and add it to D'. Observe that $|D'| \leq |D|$. Thus if there is a dominating set in G of size $\leq k$, there must be a set cover of size $\leq k$

3 More Reductions

Given an array $A = [a_1, a_2, \ldots, a_n]$ of nonnegative integers, consider the following problems:

1 **Partition**: Determine whether there is a subset $P \subseteq [n]$ ($[n] := \{1, 2, \cdots, n\}$) such that $\sum_{i \in P} a_i = \sum_{j \in [n] \setminus P} a_j$

2 **Subset Sum**: Given some integer k, determine whether there is a subset $P \subseteq [n]$ such that $\sum_{i \in P} a_i = k$

3 **Knapsack**: Given some set of items each with weight w_i and value v_i, and fixed numbers W and V, determine whether there is some subset $P \subseteq [n]$ such that $\sum_{i \in P} w_i \leq W$ and $\sum_{i \in P} v_i \geq V$

For each of the following clearly describe your reduction, justify runtime and correctness.

(a) Find a linear time reduction from Subset Sum to Partition.

(b) Find a linear time reduction from Subset Sum to Knapsack.

Solution:

(a) Suppose we are given some A with target sum t. Let s be the sum of all elements in A. If $s - 2t \geq 0$, generate a new set $A' = A \cup \{s - 2t\}$. If A' can be partitioned, then there is a subset of A that sums to t.

We know that the two sets in our partition must each sum to $s - t$ since the sum of all elements will be $2s - 2t$. One of these sets, must contain the element $s - 2t$. Thus the remaining elements in this set sum to t.

If $s - 2t \leq 0$, generate a new set $A' = A \cup \{2t - s\}$. If A' can be partitioned, then there is a subset of A that sums to t.

We know that the two sets in our partition must each sum to t since the sum of all elements will be $2t$. The set that does not contain $\{2t - s\}$ will be our solution to subset sum.

This reduction also clearly operates in $O(n)$, as we simply need to generate a new set with a single additional element (whose value is determined by summing all the elements of A).

(b) Suppose we are given some set A with target sum t. For each element k of the set, create an item with weight k and value k. Let $V = t$ and $W = t$. We know Knapsack will determine if there is a combination of items with sum of weights $\leq t$ and values $\geq t$. Because the weights and values are the same, we know (Sum of chosen weights) = (Sum of chosen values) $= t$. And since each weight/value pair is exactly the value of one of the original elements of A, we know that there will be a solution to our Knapsack problem iff there is one for our subset sum problem. This solution runs in linear time, as we need constant work for each element of A to generate our new input.

4 Randomization for Approximation

Oftentimes, extremely simple randomized algorithms can achieve reasonably good approximation factors.

(a) Consider Max 3-SAT (given a set of 3-clauses, find the assignment that satisfies as many of them as possible). Come up with a simple randomized algorithm that will achieve an approximation factor of $\frac{7}{8}$ in expectation. That is, if the optimal solution satisfies k clauses, your algorithm should produce an assignment that satisfies at least $\frac{7}{8} * k$ clauses in expectation. You may assume that every clause contains exactly 3 distinct variables.

(b) Given an instance of Max 3-SAT with n clauses, what is the maximum number of clauses that are guaranteed to be solved in at least one assignment of variables?

(c) Give an example of a Max 3-SAT instance where the optimal solution matches the number in (b).

Solution:

(a) Consider randomly assigning each variable a value. Let X_i be a random variable that is 1 if clause i is satisfied and 0 otherwise. We can see that the expectation of X_i is $\frac{7}{8}$. Note that $\sum X_i$ is the total number of satisfied clauses. By linearity of expectation, the expected number of clauses satisfied is $\frac{7}{8}$ times the total number of clauses. Since the optimal number of satisfied clauses is at most the total number of satisfied clauses, a random assignment will in expectation have value at least $\frac{7}{8} * k$.

(b) If in expectation at least $\frac{7}{8}$ of all clauses are satisfied, there must necessarily *always* exist an assignment where at least $\frac{7}{8}$ of all clauses are satisfied, as a random variable is at least sometimes greater than or equal to its mean.

Optional Note: It may seem unsatisfying that we have found an algorithm that only works in expectation. How many times would we have to run it before we actually find a solution that achives the approximation factor? Of course, it could be a lot. It turns out, however, that there is a way to effectively de-randomize this scheme in order to always produce an assignment with at least $\frac{7}{8}$ of all clauses satisfied.

Fix variable assignments one by one. When picking a value for variable x_k, consider the conditional expectation on the number of satisfied clauses given that x_0, ..., x_{k-1} hold whatever value you already set them to and x_k is assigned to true. Then consider the same conditional expectation but with x_k assigned to false. Assign x_k to whichever value maximizes the conditional expectation and then move on. At the very first step, the expectation of the two conditional expectations will be $\frac{7}{8}$ of the number of clauses. Thus the larger of the two must also be at least $\frac{7}{8}$ of the number of clauses. At the next step, one of our two choices will have conditional expectation at least as large as the expectation when conditioned only on the first value and so on.

Notice that this new scheme is fully deterministic, despite relying on probablistic concepts.

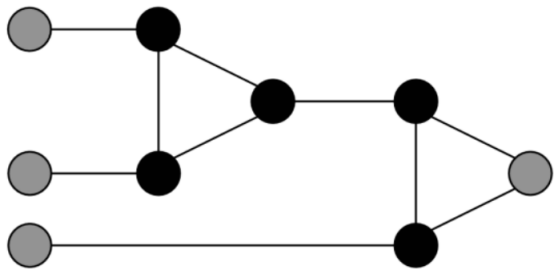
(c) Consider an instance with 3 variables and 8 clauses corresponding to all the possibilities. Then the optimal solution satisfies 7 clauses.

5 Reduction to 3-Coloring

Given a graph $G = (V, E)$, a valid 3-coloring assigns each vertex in the graph a color from $\{0, 1, 2\}$ such that for any edge (u, v), u and v have different colors. In the 3-coloring problem, our goal is to find a valid 3-coloring if one exists. In this problem, we will give a reduction from 3-SAT to the 3-coloring problem, showing that 3-coloring is NP-hard.
In our reduction, the graph will start with three special vertices, labelled "True", "False", and "Base", and the edges (True, False), (True, Base), and (False, Base).

(a) For each variable x_i in a 3-SAT formula, we will create a pair of vertices labeled x_i and $\neg x_i$. How should we add edges to the graph such that in any valid 3-coloring, one of $x_i, \neg x_i$ is assigned the same color as True and the other is assigned the same color as False?

(b) Consider the following graph, which we will call a "gadget":



Show that in any valid 3-coloring of this graph which does not assign the color 2 to any of the gray vertices, the gray vertex on the right is assigned the color 1 only if one of the gray vertices on the left is assigned the color 1.

(c) We observe the following about the graph we are creating in the reduction:

(i) For any vertex, if we have the edges (v, False) and (v, Base) in the graph, then in any valid 3-coloring v will be assigned the same color as True.

(ii) Through brute force one can also show that in the gadget, for any assignment of colors to gray vertices such that:

(1) All gray vertices are assigned the color 0 or 1
(2) The gray vertex on the right is assigned the color 1
(3) At least one gray vertex on the left is assigned the color 1

Then there is a valid coloring for the black vertices in the gadget.

Using these observations and your answers to the previous parts, give a reduction from 3-SAT to 3-coloring. Prove that your reduction is correct.

Solution:

(a) We add the edges (x_i, $\neg x_i$), (x_i, Base) and ($\neg x_i$, Base). Since $x_i, \neg x_i$ are both adjacent to Base they must be assigned a different color than Base, i.e. they both are assigned either the color of True or the color of False. Since we added an edge between x_i and $\neg x_i$, they can't be assigned the same color, i.e. one is assigned the same color as True and one the same color as False.

(b) It is easier to show the equivalent statement that if all the gray vertices on the left are assigned the color 0, then the gray vertex on the right must be assigned the color 0 as well. Consider the triangle on the left. Since all the gray vertices are assigned 0, the two left points must be assigned the colors 1 and 2, and so the right point in this triangle must be assigned 0 in any valid coloring. We can repeat this logic with the triangle on the right, to conclude that the gray vertex on the right must be assigned 0 in any valid coloring.

(c) Given a 3-SAT instance, we create the three special vertices and edges described in the problem statement. As in part a, we create vertices x_i and $\neg x_i$ for each variable x_i, and add the edges we gave in the answer to part a. For clause j, we add a vertex C_j and edges (C_j, False), (C_j, Base). Lastly, for clause j we add vertices and edges to create a gadget where the three gray vertices on the left of the gadget are the vertices of three literals in the clause, and the gray vertex on the right is the vertex C_j (All black vertices in the gadget are only used in this clause's gadget).

If there is a satisfying 3-SAT assignment, then there is a valid 3-coloring in this graph as follows. Assign False the color 0, True the color 1, and Base the color 2; assign x_i the color 1 if x_i is True and 0 if x_i is False (vice-versa for $\neg x_i$). Assign each C_j the color 1. Lastly, fix any gadget. Since the 3-SAT assignment is satisfying, in each gadget at least one of the gray vertices on the left is assigned 1, so by the observation (ii) in the problem statement the gadget can be colored.

If there is a valid 3-coloring, then there is a satisfying 3-SAT assignment. By symmetry, we can assume False is colored 0, True is colored 1, and Base is colored 2. Then for each literal where x_i is color 1, that literal is true in the satisfying assignment. By part a, we know that exactly one of $x_i, \neg x_i$ is colored 1, so this produces a valid assignment. By observation (i), we also know every node C_j must be colored 1. All literal nodes are colored 0 or 1, so by part b, this implies that for every clause, one of the gray literal nodes in the clause gadget is colored 1, i.e. the clause will be satisfied in the 3-SAT assignment.

6 Project: Drive the TAs Home

1. Logistics: Read the project spec to answer the following questions.
   (a) When are the Phase 1 and Phase 2 due dates?
   (b) What are the two deliverables in Phase 1?
   (c) What are the three deliverables in Phase 2?
   (d) Under what circumstances are you allowed to make changes to your group after 11/13?

(e) How many days are you allowed to submit after the deadline?

Solution:

(a) 11/22 at 11:59pm, 12/6 at 11:59pm

(b) Inputs, Design Doc

(c) Outputs, Final Report, Code

(d) None

(e) Zero

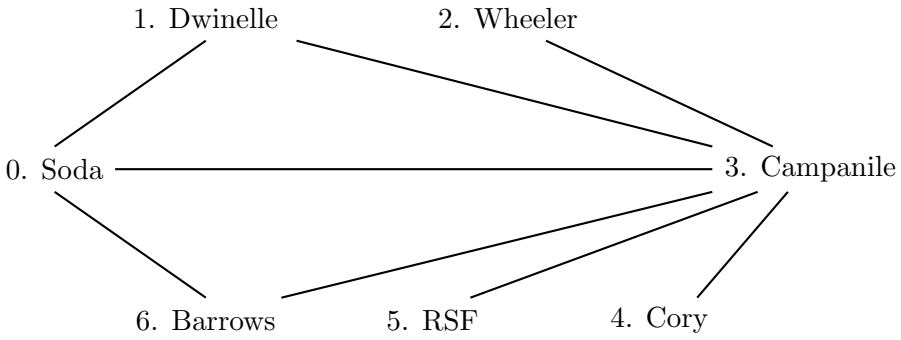
2. Cost Function

(a) Assume Rao and 2 TAs are in the car, and the car moves a distance of 5 (no TAs get dropped off in this process). How much cost does this incur?

Solution:

$$\frac{2}{3} \cdot 5 = \frac{10}{3}$$

(b) What is the cost of the example output in the spec? Graph of input is given below. All edge weights are 1. **Solution:**



$$\frac{2}{3}(1+1+1+1) + (2+2+2+0) = \frac{26}{3}$$

(c) Give the output with the optimal cost.

Solution:

```
Soda Campanile Soda
1
Campanile Campanile Wheeler Cory RSF
```

The cost of this solution is:

$$\frac{2}{3}(1+1) + (0+1+1+1) = \frac{13}{3}$$

3. In the metric TSP problem, we are given a graph $G = (V, E)$ with non-negative edge weights that satisfies the triangle inequality, and we want to find a minimum weight tour visiting each vertex at least once.

   (a) Show that if the driver takes $\frac{1}{2}$ the energy of a TA to travel the same unit of distance as walking, it would always be optimal to drop every TA off at their own home.

   Solution: We see that if the driver takes $\frac{1}{2}$ the energy of the TA, we have a new scoring function for cost. As denoted in the spec, let the path of the car be $u_0 \dots u_{n-1}$ where $u_0 = u_{n-1} = s$, let b_j be the location you drop off the j-th TA, and let h_j be that TA's home. Let ℓ_{ij} be the length of the road between locations i and j (which must be adjacent), and d_{ij} be the shortest path distance between locations i and j. We have cost:

   $$\frac{1}{2} \cdot \sum_{i=1}^{n-1} \ell_{u_{i-1}u_i} + \sum_{j=0}^{|H|-1} d_{b_j h_j}$$

   Suppose at some point in our optimal path, we have a drop off location that is not at the home of the ta. If this is the case, there must exists an u_i where the TA at a different location gets off, h_j. The student dropped off at u_i and will walk to h_j along some path $u_i v_1 v_2 \dots v_n h_j$. We see that

   $$d_{u_i h_j} = \ell_{u_1 v_1} + \ell_{v_1 v_2} + \cdots + \ell_{v_n h_j}.$$

   We observe that if instead of dropping the student off at u_i, we dropped that student off directly at their home, we incur of cost of

   $$\frac{1}{2}\left(\ell_{u_1 v_1} + \ell_{v_1 v_2} + \cdots + \ell_{v_n h_j} + \ell_{h_j v_n} + \cdots + \ell_{v_2 v_1} + \ell_{v_1 u_1}\right) = \frac{1}{2}(d_{u_i h_j} + d_{h_j u_i}) = d_{u_i h_j}$$

   which is the cost of the driver going to the home and coming back. Thus, for any optimal route $u_0 \dots u_{n-1}$, with dropoffs $b_1 \dots b_m$, we can in place of any dropoff location b_i, we can equivalently dropoff each student directly at their house and receive the same optimal cost.

   (b) Give a reduction from metric TSP to Drive the TAs Home where it takes the driver $\frac{1}{2}$ the energy of a TA to travel the same unit of distance. Assume here that metric TSP can visit each vertex more than once.

   Solution: Given a TSP instance $G(V, E)$, we convert it to an instance of Drive the TAs home where the car takes $\frac{1}{2}$ the energy of the TA by setting $H = V$. From this, the above, we see the driver is incentivized to drop off every student directly at home. We see also that since $H = V$, the car will visit every single vertex in the graph. Now since Drive the TAs Home outputs the car path of the least energy (corresponding to the least sum of costs of the edges on the path), the car path outputted by Drive the TAs Home will be a tour with the least cost and thus a solution to TSP.

   Going the other direction, given a solution to TSP, we know that this is a solution

to Drive the TAs Home where $H = V$. This is again because of the property above that the best cost solution must drop off every student at home, and since the homes are all the vertices in the graph, the car path must be a tour of the graph (allowing visiting vertices more than once). The tour that expends minimum energy is the tour of the least total distance or cost, which means that a solution to TSP will be a solution to Drive the TAs Home.

Finally, the reduction is linear time since we just pass G into Drive the TAs Home with $H = V$. Given an output of Drive the TAs Home, we simply return the car path as the TSP output.