CSC373 Assignment 4 Submission

Harrison Deng

April 11, 2024

Q1 [15 Points] Set Cover

Here is the *Set-Cover* problem. You are given a set $E = \{e_1, ..., e_n\}$, and m subsets $S_1, ..., S_m \subseteq E$. For each $j \in [m]$, we associate a weight $w_j \geq 0$ to the set S_j. The goal is to find a minimum-weight collection of subsets that covers all of E.

(a) [5 Points] Form the set-cover problem as an integer linear program, and then relax it to a linear program. Define your variables. [Hint: you might want to have a constraint like $\sum_{j:e_i \in S_j} x_j \geq 1$ for each element e_i.]

(b) [5 Points] Let x^* denote the optimal solution to the relaxed LP you defined in part (a). Let f be the maximum number of subsets in which any element appears. Here's the rounding algorithm: given x^*, we include S_j if and only if $x_j^* \geq 1/f$. Let $I = \{j : S_j \text{ is selected by the rounding algorithm}\}$. Prove that the collection of subsets S_j where $j \in I$ chosen by the rounding algorithm is a set cover.

(c) [5 Points] Let OPT be value of the optimal solution of the set-cover. Prove that the rounding algorithm in (b) gives an f-approximation.

Solution

(b) We wish to show that the collection of subsets S_j where $j \in I$ chosen by the rounding algorithm is a set cover. In order to do this, we must show that each element $e_i \in E$ is included by at least one subset in S_j.

Let f_i be the number of subsets in which e_i appears.
To begin, note that for all x_i^*, at least one x_i^* must be greater than $\frac{1}{f_i}$ due to constraints on the linear program (**claim 1**):

$$\forall i \in n, \sum_{j:e_i \in S_j} x_j \geq 1 \qquad \text{(LP constraints)} \tag{1}$$

$$\forall i \in n, \sum_{j:e_i \in S_j} x_j^* \geq 1 \qquad \text{(Optimal solution substitution)} \tag{2}$$

Then (**claim 2**),

$$f_i \leq f \tag{3}$$

$$\frac{1}{f_i} \geq \frac{1}{f} \tag{4}$$

since, for each element, there is at least one $x_i^* \geq \frac{1}{f_i}$ (claim 1), and $\frac{1}{f_i} \geq \frac{1}{f}$ (claim 2), then, (**claim 3**) $x_i^* \geq \frac{1}{f}$ ($x_i^* \geq \frac{1}{f_i} \geq \frac{1}{f}$).

Claim 3 thus allows us to state that for every element, there exists at least one $x_i^* \geq \frac{1}{f}$. Since the rounding algorithm selects all subsets S_i such that $x_i^* \geq \frac{1}{f}$ (as described), then, we can state that each element will appear at least once in at least one subset selected.

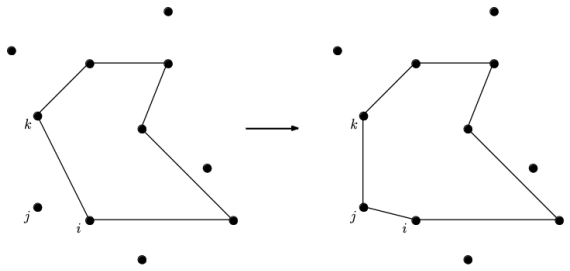
Since our selection selection criteria is such that $x_j^* \geq \frac{1}{f}$, Then, according to our first constraint in the relaxed LP, for any element e_i, they must exist we have that at least one subset S_j exists such that $x_{j*} \geq \frac{1}{f}$ We know that it must be at least 1. This means that the sum of all x_{j*} must evaluate to 1 as well, meaning that the collection of subsets, which indeed span all elements in E form a set cover.

□

Q2 [15 Points] Traveling Salesman

Here's the *metric traveling salesman* problem. You are given a complete graph $G = (V, E)$, where $V = \{1, ..., n\}$ represents the cities the salesman needs to visit. For each edge $(i, j) \in E$, we associate it with a cost c_{ij}. We call it "metric" because for every triplet of vertices $i, j, k \in V$, it respects the triangle inequality, i.e. $c_{ik} \leq c_{ij} + c_{jk}$. The goal is to have a tour of the cities (i.e. a Hamiltonian cycle of G) such that each city is visited exactly once (except for the starting city where you have to come back to), and the total cost is minimized.

Here is our approximation algorithm, which is also a greedy algorithm: Among all pairs of cities, find the two closest cities, say i and j, and start by building a tour on that pair of cities; the tour consists of going from i to j and then back to i again. This is the first iteration. In each subsequent iteration, we extend the tour on the current subset $S \subseteq V$ by including one additional city, until we include the full set of cities. Specifically in each iteration, we find a pair of cities $i \in S$ and $j \notin S$ for which the cost c_{ij} is minimum; let k be the city that follows i in the current tour on S. We add j to S, and replace the path $i \to k$ with $i \to j$ and $j \to k$. See the picture below for illustration:



Let OPT be the value of the optimal solution of the metric traveling salesman problem. Prove that the approximation algorithm above gives a 2-approximation.

Solution

Variables and Assumptions: To begin, we will define our variables and state our assumptions, let $G = (E, V)$ be the complete graph where V represents the spatial nodes to be visited and E be a series of edges that connect all vertices with each other. Each edge is assigned a weight c_{ij} for the corresponding vertices i and j. Furthermore, we will assume that the triangle inequality holds for all triangles formed by all edges. In other words, $\forall i, j, k \in V, c_{ik} \leq c_{ij} + c_{jk}$ essentially stating that for any given vertices i, j, k, the direct edge from i to k is never worse than the sequence of edges from i to j, to k. Let GRD be the given greedy algorithm and $S_g = E_g, V_g$ be the graph representation of the sequence of vertices and edges to take as the solution (traversal graph) produced by such an algorithm. We let function $c(S)$ be the sum of all edge weights for traversing all vertices of a graph S. Lastly, we will assume OPT is the cost of a optimal solution to traveling salesman problem (TSP) where such a traversal graph is $S_o = (E_o, V_o)$. Our objective is to show that, S_g is at worst, $c(S_g) \leq 2 \times$ OPT.

Prim's Minimum Spanning Tree Algorithm Review: Very briefly, the Prim's minimum spanning tree (MST) algorithm begins by arbitrarily selecting a vertex from a graph, and iteratively selecting the next vertex with the lowest edge weight connecting to the current set of selected vertices.

Claim 1: To begin, notice that $c(S_o)$ is a cycle that traverses all vertices minimally and cyclically where each node is traversed exactly once with the exception of the starting node. Then, see that $c(S_o)$ may be trivially converted into a tree graph by simply removing any edge in S_o and arbitrarily selecting a vertex to become the root of the tree. Such a tree is spanning (all vertices connected). Furthermore, see that the traversal of such a tree costs will not cost more than the traversal of the original cycle S_o. In other words, $\forall e \in E_o, c(S_o - \{e\}) \leq c(S_o)$.

Claim 2: See that the greedy traversal graph S_g will always result in requiring half of the number of edges to traverse all nodes via connected edges when compared to traversing a MST. To see this, we assert that `GRD` produces a traversal graph S_g that is no different from a graph produced by Prim's MST algorithm $S_p = (E_p, V_p)$ from G, after running a depth first search (DFS) on S_p, and removing the duplicates, connecting edges that traversed to the duplicate vertices directly to the subsequent vertex after the removed vertex.

This is because `GRD` is substantially different only in the step of adding the selected vertex to the current graph. Where in Prim's, the algorithm selects the vertex $s \in G$ associated with the lowest weighted edge that connects to a vertex i in the partial solution $S_{pp} = (E_{pp}, V_{pp})$ and proceeding to the next iteration, `GRD` selects the next vertex and edge in the same fashion, however, instead of moving to the next iteration, `GRD` connects the selected node, $s \in G$, to the next node i was linked to $k \in S_{pp}$. In other words, where Prim's may resolve to connect $i \to s$ such that $E_{pp} = \{\ldots, \{i, k\}, \{i, s\}, \ldots\}$, and a traversal by DFS results in a sequence $\ldots \to i \to k \to i \to s \to \ldots$. `GRD` resolves the newly selected vertex such that the partial traversal graph for `GRD` is $S_{gp} = (E_{gp}, V_{gp})$ where $E_{gp} = \{\ldots, \{i, s\}, \{s, k\}, \ldots\}$, effectively changing $i \to k$ to $i \to s \to k$ thus maintaining the chain form of the graph.

We can then see that Prim + DFS does create double the edges than `GRD`, since the DFS traversal $\ldots \to i \to k \to i \to s \to \ldots$ can be simplified into $\ldots \to i \to k \to s \to \ldots$ (removing the second appearance of i in the sequence) without worsening the total weight required for the traversal by the triangle inequality (TI) assumption. To prove this, we may focus on $k \to i \to s$, and see that $c_{ks} \leq c_{ki} + c_{is}$ (TI assumption).

From this, we can see that Prim's algorithm is known to generate a MST, and to traverse such a tree one vertex after another is double the cost of the cyclical traversal path provided by `GRD`. In other words, we have proven $2c(S_p) = (c(S_g))$ or $c(S_p) = \frac{1}{2}(c(S_g))$.

Proof Algorithm is 2-Approximation:

$$c(S_p) \leq c(S_o - \{e\}) \qquad \text{(Prim's is MST therefore, costs} \leq \text{other spanning trees.)} \tag{5}$$

$$\frac{1}{2}c(S_g) \leq c(S_o - \{e\}) \qquad \text{(Claim 2 substitution)} \tag{6}$$

$$\frac{1}{2}c(S_g) \leq c(S_o - \{e\}) \leq c(S_o) \qquad \text{(Claim 1)} \tag{7}$$

$$c(S_g) \leq 2c(S_o) \tag{8}$$

Where S_g is the solution produced by `GRD`, and S_o is the optimal solution. Hence, we've shown that the described `GRD` algorithm will always result in a solution no worse than twice the optimal solution, i.e., 2-approximation.

Q3 [20 Points] Randomized Algorithms

Let $G = (V, E)$ be an undirected graph. For any subset of vertices $U \subseteq V$, define

$$\mathsf{cut}(U) = \{(u, v) \in E : u \in U \text{ and } v \notin U\}.$$

The set $\mathsf{cut}(U)$ is called the *cut* determined by the vertex set U. The size of the cut is denoted by $|\mathsf{cut}(U)|$. The *Max-Cut* problem asks you to find the cut with maximum size, i.e., $\max_{U \subseteq V} |\mathsf{cut}(U)|$.

Here is a randomized algorithm for *Max-Cut*: Take a uniform random subset U of V, and choose $\mathsf{cut}(U)$ to be the cut. Let OPT be the size of the maximum cut in G. Prove that the randomized algorithm gives a cut of expected size at least half of the optimal solution, i.e., $\mathbb{E}[|\mathsf{cut}(U)|] \geq \frac{1}{2}\mathsf{OPT}$.

Q4 [5 Points] Extra Credit

"Here is the link for EC3, you should submit this with HW4 (not HW3)." — Harry

https://colab.research.google.com/drive/1Mo8S-asikkd4qBakMldCwlsDHcpyzEmo?usp=sharing

References

Please write down your references here, including any paper or online resources you consult.