CSCI3160: Regular Exercise Set 13

Prepared by Yufei Tao

Problem 1 (Reduction from Hitting Set to Set Cover). Given an instance to the hitting set problem, explain how to convert it to a set cover problem.

Solution. In the hitting set problem, we are given a collection of sets $\mathcal{S}$, where each set $S \in \mathcal{S}$ is a subset of some universe U. We want to find a hitting set $H \subseteq U$ of the smallest size (recall that H is an hitting set if $H \cap S \neq \emptyset$ for every $S \in \mathcal{S}$).

Define a bipartite graph G where

- every left vertex of G corresponds to a set $S \in \mathcal{S}$;
- every right vertex of G corresponds to an element $e \in U$;
- G has an edge between a set vertex S and an element vertex e if and only if $e \in S$.

Solving the original hitting set problem is equivalent to finding a smallest set R of right vertices such that every left vertex is adjacent to at least one vertex in R.

For each $e \in U$, define N_e as the set of neighbors of the element vertex e (i.e., a right vertex). Note that a set vertex S (i.e., a left vertex) is in N_e if and only if $e \in S$. The set collection $\{N_e \mid e \in U\}$ defines a set cover problem, whose universe is the set of left vertices and has a size of $|\mathcal{S}|$. Let $\mathcal{C}$ be an optimal set cover of this problem. Then $H = \{e \in U \mid N_e \in \mathcal{C}\}$ must be an optimal hitting set for the original problem.

Problem 2 (Reduction from Set Cover to Hitting Set). Given an instance to the set cover problem, explain how to convert it to a hitting set problem.

Solution. In the set cover problem, we are given a collection $\mathcal{S}$ of sets and a universe $U = \bigcup_{S \in \mathcal{S}} S$. We want to find a set cover $\mathcal{C} \subseteq \mathcal{S}$ of the smallest size (recall that $\mathcal{C}$ is a set cover if $\bigcup_{S \in \mathcal{C}} S = U$).

Define a bipartite graph G where

- every left vertex of G corresponds to a set $S \in \mathcal{S}$;
- every right vertex of G corresponds to an element $e \in U$;
- G has an edge between a set vertex S and an element vertex e if and only if $e \in S$.

Solving the original set cover problem is equivalent to finding a smallest set L of left vertices such that every right vertex is adjacent to at least one vertex in L.

For each $e \in U$, define N_e as the set of neighbors of the element vertex e (i.e., a right vertex). Note that a set vertex S (i.e., a left vertex) is in N_e if and only if $e \in S$. The set collection $\{N_e \mid e \in U\}$ defines a hitting set problem. Find an optimal hitting set H of this problem (note that H is a set of set vertices). Then, the collection $\{S \in \mathcal{S} \mid \text{the vertex of } S \text{ is in } H\}$ must be an optimal set cover for the original problem.

Problem 3. In the hitting set problem, we are given a collection of sets $\mathcal{S}$, where each set $S \in \mathcal{S}$ is a subset of some universe U. We want to find a hitting set $H \subseteq U$ of the smallest size (recall that H is an hitting set if $H \cap S \neq \emptyset$ for every $S \in \mathcal{S}$). Let OPT be the size of an optimal hitting set. Design a polynomial time algorithm that returns a hitting set of size at most $\text{OPT} \cdot (1 + \ln |\mathcal{S}|)$.

Solution. Use the solution to Problem 1 to convert this problem to a set cover problem whose universe has size $|\mathcal{S}|$. Run our greedy set-cover algorithm to obtain a set cover of size $\text{OPT} \cdot (1 + \ln |\mathcal{S}|)$. Then, return $H = \{e \in U \mid N_e \in \mathcal{C}\}$ the original problem.

Problem 4. Let $G = (V, E)$ be an undirected simple graph where each edge $e \in E$ is associated with a non-negative weight $w(e)$. For any vertices $u, v \in V$, define $spdist(u, v)$ as the shortest path distance between u and v. Given a subset $C \subseteq V$, define its *cost* as

$$cost(C) \quad = \quad \max_{u \in V} \min_{c \in C} spdist(c, u).$$

Fix an integer $k \in [1, |V|]$. Let OPT be the smallest cost of all subsets $C \subseteq V$ with $|C| = k$. Design an algorithm to find a size-k subset with cost at most $2 \cdot \text{OPT}$. Your algorithm must run in time polynomial to $|V|$.

Solution. First, calculate the shortest path distances between all pairs of vertices in V. This can be done in polynomial time by resorting to Dijkstra's algorithm. Then, run the k-center algorithm discussed the class on V. Specifically, initialize an empty set C and add to C an arbitrary vertex. Then, repeat the following step until $|C| = k$: add to C the vertex u maximizing $\min_{c \in C} spdist(c, u)$.

The proof regarding the approximation ratio 2 remains valid as long as the distance function satisfies the triangle inequality. It is clear that shortest path distances satisfy the triangle inequality.

Problem 5. Consider the k-center problem on a set P of n 2D points. Our lecture made the assumption that the Euclidean distance of any two points can be computed precisely in polynomial time. This is not a realistic assumption (because the computation requires calculating square roots). Modify our 2-approximate algorithm to make it run in polynomial time without the assumption.

Solution. You do not need to compute Euclidean distances! All we need is to *compare* two Euclidean distances to see which one is larger. More specifically, given four points a, b, c, and d, it suffices to compare $dist(a, b)$ and $dist(c, d)$, where $dist(., .)$ represents the Euclidean distance between two points. Let $a[x]$ and $a[y]$ be the x- and y-coordinates of a, respectively (and adopt similar notations for b, c, and d). It suffices to compare $(a[x] - b[x])^2 + (a[y] - b[y])^2$ to $(c[x] - d[x])^2 + (c[y] - d[y])^2$. It is clear that such comparison can be done in $O(1)$ time.

It should now be straightforward to modify the algorithm to run in polynomial time without the assumption.

Problem 6.** Let P be a set of n 2D points. Given a subset $C \subseteq P$, define:

- (for each point $p \in P$) $dist_C(p) = \min_{c \in C} dist(c, p)$, where $dist(c, p)$ represents the Euclidean distance between c and p;
- $cost(C) = \max_{p \in P} dist_C(p)$.

Fix a real value $r > 0$. Call a subset $C \subseteq P$ an *r-feasible subset* if $cost(C) \leq r$. Prove: unless P = NP, there does not exist an algorithm that can find an r-feasible subset with the smallest size in time polynomial to n. You can assume that the Euclidean distance of any two points can be computed in polynomial time.

(Hint: Show that the existence of such an algorithm implies a polynomial time algorithm for the k-center problem.)

Solution. Let us refer to the above problem as the *r-radius problem.* Suppose that we are given an algorithm $\mathcal{A}$ that can solve the problem in polynomial time for any r. Next, we will show how to solve the k-center problem discussed in the class in polynomial time.

First, compute the distance between each pair of points in P. This produces a set R of $\binom{n}{2}$ distances. Sort these distances in ascending order, and denote the i-th smallest distance as r_i for $i \in [1, \binom{n}{2}]$. For each i, use algorithm $\mathcal{A}$ to solve the r_i-radius problem and obtain its output C_i^*. The sizes of $|C_1^*|, |C_2^*|, ..., |C_{\binom{n}{2}}^*|$ must be in non-ascending order. Identify the smallest j with $|C_j^*| \le k$ and return C_j^* as the solution to the k-center problem. If $\mathcal{A}$ runs in polynomial time, then the whole algorithm runs in polynomial time.

Next, we will prove that the above algorithm correctly solves the k-center problem. Let C^* be an optimal solution to the k-center problem. We will prove $cost(C_j^*) = cost(C^*)$ (recall that $cost(C_j^*) = r_j$). Suppose that $cost(C_j^*) > cost(C^*)$. It is important to note that $cost(C^*)$ equals the distance of two points in P and, hence, $cost(C^*) = r_t$ for some $t \in [1, \binom{n}{2}]$. Hence, the condition $cost(C_j^*) > cost(C^*)$ tells us $r_j > r_t$. As the distances in R are sorted in ascending order, we must have $j > t$. By how j is chosen, we know that $|C_t^*| > k = |C^*|$.

However, as C^* is an r_t-feasible subset, it is a better solution to the r_t-radius problem than C_t^* (due to the fact $|C^*| < |C_t^*|$). This contradicts the fact that C_t^* is an optimal solution to the r_t-radius problem.