Lecture Notes CS:5350 Design & Analysis of Algorithm
Lecture 23: November 12
Masuma Akter Rumi

1 Hardness of Approximating k-Center

We have been talking about greedy algorithms for approximation and we showed a 2-approximation for k-Center problem in the previous class. We are going to show that it is hard to get a better than 2-approximation for k-Center.

Theorem 1 *For $\rho < 2$, if there is a ρ-approximation algorithm for k-*Center *then $P = NP$.*

Proof: We will prove this by reduction from *Minimum Dominating Set (MDS).*

Now, recall that a dominating set $S \subseteq V$ of a graph $G = (V, E)$ is such that for every $v \in V$, either $v \in S$ or a neighbor of v is in S. An example is given below:

Example:

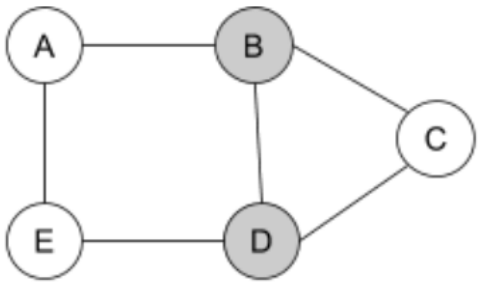

Figure 1: The *Minimum Dominating Set (MDS)* of the example graph is, S={B,D}

Now, we know that *MDS* is an optimization problem where we want to minimize the number of vertices that we pick for the dominating set. But, we also know that the theory of NP completeness, in general the theory of hardness pertains to decision problems. So, we are interested in the decision version of *MDS*. When we talk about decision problems, we ask YES/NO questions. We are not trying to produce an output that is minimizing or maximizing an objective function. Instead we are asking YES/NO questions and that is why these are called decision problem. The decision version of *MDS* is given below:

***MDS-decision version*:**
Input: A graph $G = (V, E)$ and a positive integer k.
Output: "YES" if there is a dominating set of G of size $\leq k$; otherwise "NO".

We know that *MDS-decision version* is *NP-complete.*

Polynomial time reduction:
Now we are going to talk about the polynomial time reduction from *MDS-decision version* to k-Center. Polynomial time reduction means that we will be given input for *MDS-decision version* and we will feed this input to an algorithm running in polynomial time. This algorithm will produce input to k-Center. A simple pictorial representation is given in Figure 2. A key property of the reduction is that solving the k-Center problem on the input produced by the reduction will lead to a solution to *MDS-decision version* input we started with.

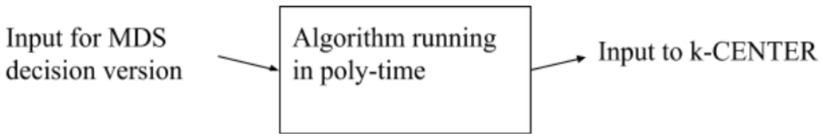
Input for MDS decision version
Algorithm running in poly-time
Input to k-CENTER

Figure 2: Pictorial representation

Now if we assume such algorithm exists, then if we are given the input of *MDS-decision version*, we would run the algorithm with this input and produce the input to k-Center. Then we would solve the k-Center problem assuming that there is a ρ-approximation better than 2. Finally, from that solution, we would extract solution (YES/NO answer) to *MDS-decision problem*. However, this will be a contradiction, as we know that *MDS-decision problem* is NP-complete, so there should not be any polynomial time algorithm for it. Now by adding the polynomial time reduction with the existence of better than ρ-approximation we find a contradiction. This is why, such ρ-approximation cannot exist.

Input Construction:
We construct an input to k-Center from the given input $(G = (V, E), k)$ to *MDS-decision version* as follows:

- $D(v, v) = 0$ for all vertices
- $D(u, v) = 1$ for all $\{u, v\} \in E$
- $D(u, v) = 2$ for all $\{u, v\} \notin E$
- Set k in k-Center to k in *MDS-decision version* input.

It is easy to check that D is a metric.

Example:
For figure 3,

- $D(A, A) = 0$
- $D(A, B) = 1$ as there is an edge between A and B
- $D(A, C) = 2$ as there is no direct edge between A and C

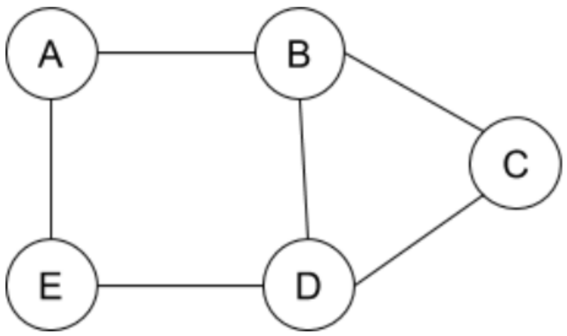

Figure 3: Example graph

- $D(A, D) = 2$ as there is no direct edge between A and D

Time complexity:
To construct an input to k-CENTER, we have to consider each pair of vertices which takes quadratic time in the number of vertices. Therefore, it is easy to check that this reduction can be done in polynomial time in the size of (G, k).

To continue the proof, now we suppose there exists a ρ-approximation algorithm A for k-CENTER for some $\rho < 2$.

Now, we can show the scheme together with the following figure:

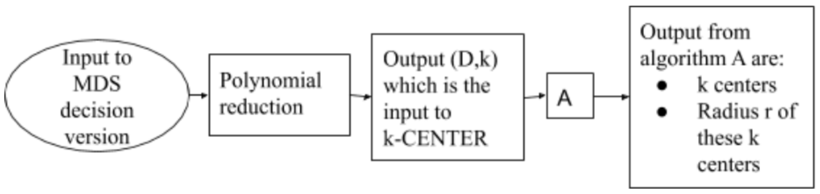
Input to MDS decision version
Polynomial reduction
Output (D,k) which is the input to k-CENTER
A
Output from algorithm A are:
• k centers
• Radius r of these k centers

Figure 4: Pictorial representation of the scheme

Now the question is how can we take this output from algorithm A and get an answer (YES/NO) to *MDS*?

To answer this question let us first understand what it means for a dominating set of size k to exist. It means that every vertex of the graph is within one hop of dominating set vertices. Which also means that every vertex has a distance of at most 1 from a dominating set. What happened is that all of the vertices are in distance at most 1 from a center. Which implies that optimal solution to k-CENTER has radius ≤ 1. Let us call this case as (Case 1) where the answer to the *MDS-decision version* is "YES". Let us now formally describe this Case 1.

Case 1: Suppose (G, k) is a "YES" instance of *MDS-decision version*, i.e. there exists a domi-

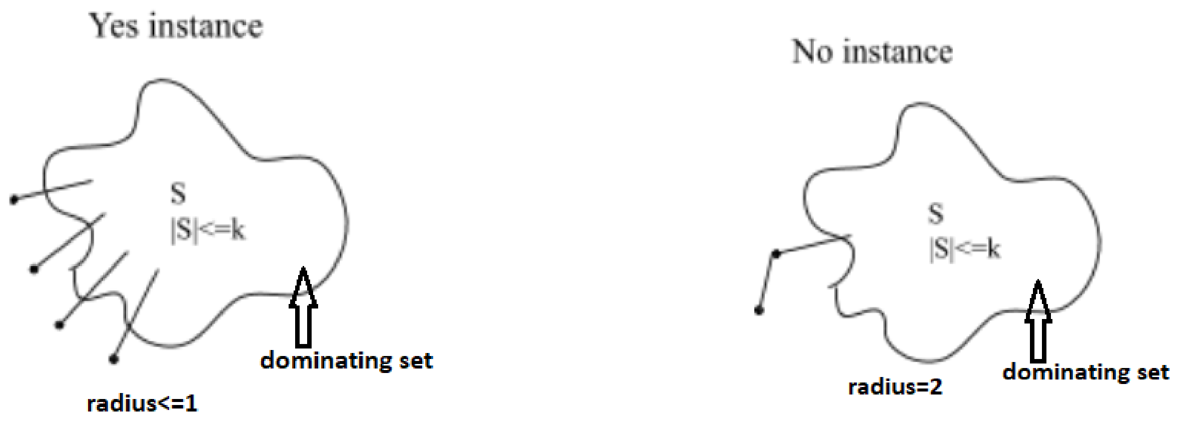

Figure 5: Illustrating showing “YES” and “NO” instances of *MDS-decision version*

nating set $S \subseteq V$ of G of size $\leq k$. This implies that if we choose vertices in S to be the k centers for the input (D, k), we get a solution with radius ≤ 1. Therefore, There is an optimal solution to k-Center on (D, k) with radius ≤ 1. So, algorithm A produces a solution with radius $\leq \rho < 2$

Now, we will consider the “NO” instance which states there does not exist any dominating set $\leq k$. It means that no matter which k vertices we pick, there exists at least one vertex which is 2 or more hops away from the set of vertices we are considering. Formal description of case 2 is given below:

Case 2: Suppose (G, k) is a “NO” instance of *MDS-decision version.* For any $S \subseteq V$, $|S| \leq k$, there is a vertex outside of S that has no neighbor in S. Therefore, $D(v, S) = 2$. So, the radius of any solution to k-Center is $= 2$. Therefore, algorithm A produces a solution with radius $= 2$

Therefore, by checking if radius of output is < 2 or $= 2$ we can distinguish between “YES” instance and “NO” instance of *MDS-decision version* in polynomial time. $\square$

2 SetCover

SetCover is another optimization problem for which a greedy algorithm gives the best approximation. Here, we are given a ground set X where $|X| = m$. We are also given n subsets $S_1, S_2, \ldots, S_n$ of X. Our goal is to find the minimum number of subsets that cover all the elements of X. The input and output of SetCover problem is given:

Input: Ground set X, $|X| = m$ and subsets $S_1, S_2, S_3, \ldots, S_n \subseteq X$.
Output: $C \subseteq \{1, 2, \ldots, n\}$ such that, $\bigcup_{i \in C} S_i = X$ and $|C|$ is minimum.

Example:
We can see in Figure 7, there are 5 sets S_1, S_2, S_3, S_4, S_5 that covers the given ground set with 12

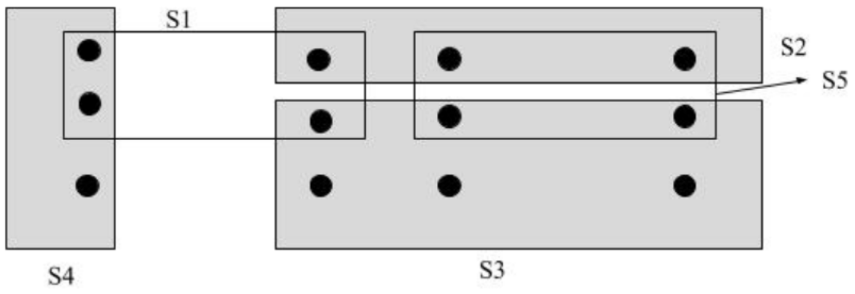

Figure 6: Example of SetCover where solution is (S_2, S_3, S_4)

elements. Our goal is to find minimum number of subsets that cover all the elements. From the figure we see that subsets S_2, S_3, S_4 cover all the vertices and also it is the minimum number of sets.

Observation: *Minimum Vertex Cover (MVC)* and *Minimum Dominating Set (MDS)* are special cases of SetCover.

For instance, in *MVC* problem, we try to cover all the edges of the graph. So the edges will be the ground set. And the subsets we are using to cover the edges are the set of vertices.

Algorithm 1 Greedy Algorithm for SetCover

```
1: $U \leftarrow X$ {U is to be covered elements}
2: $C \leftarrow \emptyset$ {C is the solution}
3: while $u \neq \emptyset$ do
4:     Pick $S_i$ such that $|S_i \bigcap u|$ is maximum
5:     Add $i$ to $C$
6:     $U \leftarrow U \setminus S_i$
7: end while
8: Output $C$
```

Greedy does not produce optimal solution. In fact we constructed a "bad" example for *MVC* for which the greedy algorithm returns a solution whose size is $\Omega(\log n)$ times optimal solution.

There is a hardness of approximation result known for *MDS*.

Theorem 2 *For any $\varepsilon > 0$, if there is a $(1-\varepsilon)\ln n$-approximation algorithm for* SetCover *then,* $NP \subseteq DTIME(n^{O(\log\log n)})$

We will continue this discussion with more details in the next class.