CSE 632: Analysis of Algorithms II **Fall 2017**

Lecture 2 (09/01/2017): Approximation Algorithm for k-Center

Lecturer: Shi Li *Scribe: Shi Li*

2.1 2-Approximation Algorithm for k-Center

In this section, we consider the k-center problem. In the problem, we are given a finite (symmetric) metric space (X, d)[1], and an integer $k \geq 1$, the goal of the problem is to find a set $C \subseteq X$ of size at most k so as to minimize

$$\mathrm{cost}(C) := \max_{u \in X} \min_{c \in C} d(u, c).$$

That is, we select at most k centers C and connect every point $u \in X$ to its nearest center $c \in C$ and its connection cost the distance $d(u, c)$; the goal is to minimize the maximum connection cost over all points $u \in X$. One application of the problem is clustering: we are given a set X of points that come from k hidden clusters and the goal is to recover the k clusters. We can solve the k-center problem and the k centers as well as the connections of points to the k centers will give the k clusters. Another application is the placement of fire stations (or some other facilities) in a city. We are given a map of buildings in the city and we need to build k fire stations while minimizing the maximum distance between a building and its nearest fire station.

We now consider a simpler task: assume we are give an upper bound L on the cost of the optimal solution C^* (which is, of course, not known to the algorithm). Our goal is to find a solution $C \subseteq X$, $|C| \leq k$, such that $\mathrm{cost}(C) \leq 2L$. We can use the following greedy algorithm to solve this task:

Algorithm 1 check(L)

```
Let S ← X, C ← ∅
for i ← 1 to k do
    let u ← arbitrary vertex in S
    let C ← C ∪ {u}, S ← S \ {v ∈ S : d(u,v) ≤ 2L}
    if S = ∅ then return C
declare failure
```

Observation 2.1 *If* check(L) *returns a set C, then C is a valid solution with* $\mathrm{cost}(C) \leq 2L$.

This holds since every time we add a center u to C, we only remove points whose distance is at most $2L$ to u from S, and we added at most k centers to C. If S becomes empty, then we have $\mathrm{cost}(C) \leq 2L$.

The important lemma we need to prove is

Lemma 2.2 *If $L \geq \mathrm{cost}(C^*)$, then the algorithm will always return a set C.*

[1] Recall that in a metric space (X, d), X is a set of points, $d : X \times X \to \mathbb{R}_{\geq 0}$ is a function such that $d(u, u) = 0$ for every $u \in X$, $d(u, v) = d(v, u)$ for every $u, v \in X$ and $d(u, w) \leq d(u, v) + d(v, w)$ for every $u, v, w \in X$.

Proof: Let $C^* = \{c_1, c_2, \cdots, c_k\}$ and let $B_j = \{u \in X : d(u, C_j) \leq L\}$ be the set of points whose distance to the center c_j is at most L. Thus, we have $\bigcup_{j=1}^k B_j = X$.

Intuitively, if we focus on the set of balls in $\{B_1, B_2, \cdots, B_k\}$ that are completely removed from S in check(L), then in each iteration the cardinality of the set will be increased by at least 1; i.e, one new ball will be completely removed from S. This is true since if $u \in B_j$, then $B_j \subseteq \{v \in X : d(u, v) \leq 2L\}$.

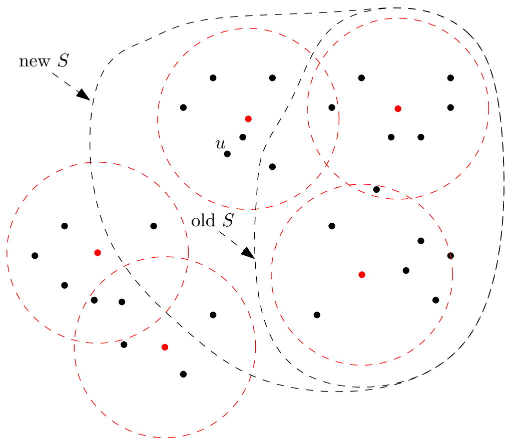

Figure 2.1: Analysis of check(L). The red dashed circles denote the balls $B_1, B_2, \cdots, B_k$. At the beginning of an iteration, S contains 2 full balls; at the end of the iteration, S contains 3 full balls.

Formally, we shall prove by induction that, after iteration i in the algorithm, there exists a set $I \subseteq [k]$ of size i, such that $\bigcup_{j \in I} B_j \subseteq X \setminus S$. This is clearly true for $i = 0$. Assume now the statement holds for $i = i' - 1$, for some $i' \in [k]$; that is there exists a set $I' \subseteq [k]$ of size $i' - 1$ such that $\bigcup_{j \in I'} B_j \subseteq X \setminus S$ at the end of iteration $i' - 1$. If the algorithm returns at the end of iteration $i' - 1$, there is nothing to prove. Otherwise, in iteration $i = i'$, we first choose a point $u \in S$. So, $u \notin \bigcup_{j \in I} B_j$. Since $B_j \subseteq \{v \in X : d(u, v) \leq 2L\}$, there must be some $j^* \notin I$ such that $u \in B_{j^*}$. For every point $v \in B_{j^*}$, we have

$$d(u, v) \leq d(u, c_{j^*}) + d(c_{j^*}, v) \leq L + L = 2L.$$

Thus, all points $v \in S \cap B_{j^*}$ shall be removed from S in iteration i. So, at the end of iteration i, we have $\bigcup_{j \in I' \cup \{j^*\}} B_j \setminus X \setminus S$. Thus the statement holds for iteration $i = i'$ with $I = I' \cup \{j^*\}$.

Thus, the algorithm either terminates before iteration k, or at the end of iteration k, S becomes $\emptyset$. In either case, the algorithm returns a solution C. ■

We can use the above procedure to obtain a 2-approximation for the k-center problem. There are at most $\binom{|X|}{2} + 1$ possible values for cost(C^*) since cost(C^*) must be the distance between two points in X. If we run check(L) for every L in the set, we shall obtain a 2-approximation for k-center.

Lemma 2.3 *Algorithm 2 returns a solution C to the k-center instance with* $\mathrm{cost}(C) \leq 2\mathrm{cost}(C^*)$.

Proof: Let $\mathrm{cost}(C^*) = d_{i^*}$. Then the above algorithm will return a set C at iteration $i \leq i^*$, since it is guaranteed that the algorithm will return a set C at iteration i^* (if it does not return before that), by Lemma 2.2. By Observation 2.1, we have $\mathrm{cost}(C) \leq 2d_i \leq 2d_{i^*} = 2\mathrm{cost}(C^*)$. ■

Algorithm 2 Solving k-center by enumeration

Let $\{d_1, d_2, \cdots, d_p\} \leftarrow \{d(u,v) : u, v \in X\}, d_1 < d_2 < \cdots < d_p$ be the set of all pairwise distances among points in X; thus $p \leq \binom{|X|}{2} + 1$
for $i \leftarrow 1$ to p **do**
  if check(d_i) returns a set C **then return** C

This establishes that Algorithm 2 is a 2-approximation for k-center. There is one big disadvantage for the algorithm: one has run check(L) possibly $\Theta(|X|^2)$ times, which is too much. A better way is to run use binary-search to find the right L.

Algorithm 3 Solving k-center by binary-search

Let $\{d_1, d_2, \cdots, d_p\} \leftarrow \{d(u,v) : u, v \in X\}, d_1 < d_2 < \cdots < d_p$ be the set of all pairwise distances among points in X; thus $p \leq \binom{|X|}{2} + 1$
$a \leftarrow 1, b \leftarrow p$
while $a < b$ **do**
  $i \leftarrow \left\lfloor \frac{a+b}{2} \right\rfloor$
  if check(d_i) returns a set C' **then**
    $b \leftarrow i, C \leftarrow C'$
  else
    $a \leftarrow i + 1$
return C

Assume cost$(C^*) = d_{i^*}$. Then at any time of the algorithm, it is guaranteed that $a \leq i^*$. Also, once C is set, we always have that C is a set returned by check(d_b). Also C will be set at some iteration. Thus, in the end we have $a = b \leq i^*$ and thus cost$(C) \leq 2d_b \leq 2d_{i^*} = 2$cost$(C^*)$. This way, we reduce the number of iterations to $O(\log |X|)$.

There is an even better algorithm that completely avoids enumerating L. Assume that in the algorithm check(L), we do not specify the exact value of L, but we require that the algorithm runs correctly for every L. To be more specific, we consider a game between the algorithm check, who does not known L, and a verifier who knows L. The verifier maintains a set S, and initially $S = X$; the algorithm does not know what S is. The game runs for k iterations. In each iteration, the algorithm adds some point u to the center set C. Then the verifier sees the point u the algorithm picked and removes $v \in S : d(u,v) \leq 2L$ from S; the algorithm can not see this operation. The algorithm runs incorrectly, if at some iteration, $S \neq \emptyset$ but the algorithm picks some $u \notin S$. Notice that it is OK if S becomes empty before iteration k but the algorithm keeps running after S becomes empty. Otherwise, the algorithm runs correctly.

Then the question becomes, how can we make sure that the algorithm always runs correctly, now matter what L is? This can be guaranteed if the algorithm always chooses the safest u in each iteration: the point u with the maximum distance to its nearest center in C. This vertex u is the safest, since if this u has been removed from S, then all points have been removed from S. This leads to the following algorithm:

It can be shown directly that the above algorithm gives a 2-approximation for k-center. We shall leave this as a homework exercise.

Algorithm 4 Algorithm for k-center without enumerating L

$C \leftarrow \{u\}$, where u is an arbitrary vertex in X
for $i \leftarrow 2$ to k **do**
    let u be the vertex in X with the largest $\min_{v \in C} d(u, v)$, breaking ties arbitrarily
    $C \leftarrow C \cup \{u\}$
return C