CMSC 451: Lecture 8
Greedy Approximation Algorithms: The k-Center Problem

Tuesday, Sep 26, 2017

Reading: A variant of this problem is dicussed in Chapt 11 in KT and Section 9.2.2 in DPV.

Greedy Approximation for NP-Hard Problems: One of the common applications of greedy algorithms is for producing approximation solutions to NP-hard problems.

As we shall see later this semester, NP-hard optimization problems represent very challenging computational problems in the sense that there is no known exact solution that has worst-case polynomial-time running time. Given an NP-hard problem, there are no ideal algorithmic solutions. One has to compromise between optimality or running time. Nonetheless, there are are number of examples of NP-hard problems where simple greedy heuristics produce solutions that are not far from optimal.

Clustering and Center-Based Clustering: Clustering is a widely studied problem with applications in statistics, pattern recognition, and machine learning. In a nutshell, it involves partition a large set of objects, called *points*, into a small number of subsets whose members are all mutually close to one another. In machine learning, clustering is often performed in high-dimensional spaces. Each data object is associated with a *property vector*, a numeric vector whose length may range from tens to thousands that describes the salient properties of this object. Then clustering is performed to group similar objects together.

In this geometric view of clustering, we think of the data set as a set P of n points in a multi-dimensional space (see Fig. 1(a)). The output of the clustering algorithm is the set $C = \{c_1, \ldots, c_k\}$ of points called *cluster centers* or simply *centers* (see Fig. 1(b)). Depending on the method being employed, it may be required that the center points are drawn from P itself, or they may just be arbitrary points in space, sometimes called *Steiner points*. In our example, the cluster centers have been chosen from P.

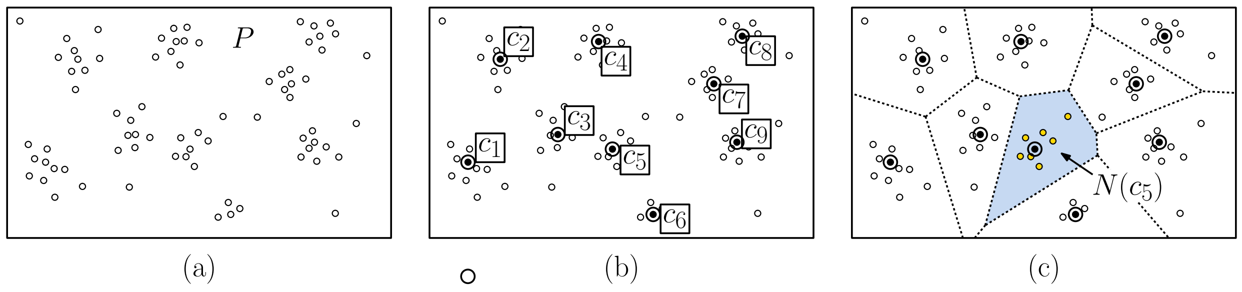

Fig. 1: Center-based geometry clustering.

Given a center point c_i, the associated cluster are the points of P that are closer to c_i center than to any other cluster center. We refer to this as c_i's *neighborhood*, denoted $N(c_i)$. In Fig. 1(c), we show a subdivision of space into nine regions according to which cluster center is closest, and highlight the neighborhood $N(c_5)$ in particular. (By the way, this subdivision is a famous structure in geometry, called the *Voronoi diagram* of the cluster centers. Computing

Voronoi diagrams is covered in a course on Computational Geometry, but it is not necessary for our purposes. In $O(nk)$ time we can compute the distance between each of the n points and each of the k centers, and assign each point to its closest center.)

The k-Center Problem: There are many ways in which to define clustering as an optimization problem. We will consider one of the simplest. We are given a set P of n points in space. For each pair of points $u, v \in P$, let $\delta(u, v)$ denote the distance between u and v. We will assume that the distance satisfies the typical properties of any natural distance function:

Positivity: $\delta(u, v) \geq 0$ and $\delta(u, v) = 0$ if and only if $u = v$.

Symmetry: $\delta(u, v) = \delta(v, u)$

Triangle inequality: $\delta(u, w) \leq \delta(u, v) + \delta(v, w)$

For the rest of the lecture, we can think of δ as the Euclidean distance, but the algorithm that we will present can be applied to any function satisfying the above properties.

k-center problem: Given a set P of n points in space and an integer $k \leq n$, find a set $C \subseteq P$ of k points in order to minimize the maximum distance of any point of P to its closest center in C.

Why maximum distance and not, say, sum of distances? There are many formulations of clustering, and this is the one that we have chosen for now.

We can view the k-center problem as a covering problem by balls. Given a point x in space and radius r, define the *ball* $B(x, r)$ to be the (closed) ball of radius r centered at x. Given any solution C to the k-center problem, let $\Delta(C)$ denote the maximum distance from any point of P to its closest center. If we now place balls of radius $\Delta(C)$ about each point in C, it is easy to see that every point of P lies within the union of these balls. By definition of $\Delta(C)$, one of the points of P will lie on the boundary of one of these balls (for otherwise we could make $\Delta(C)$ smaller). The neighborhood of each cluster will lie within its associated ball (see Fig. 2(b)).

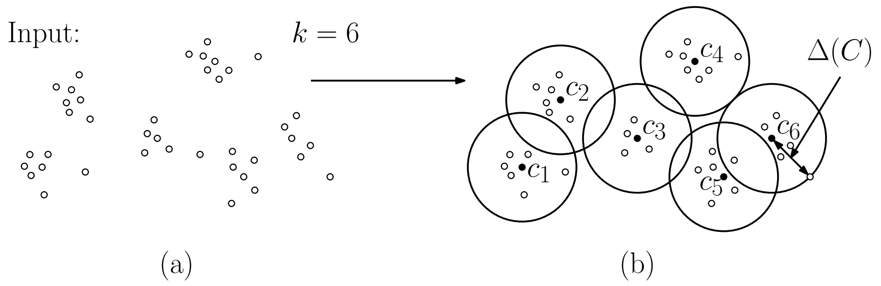

Fig. 2: The k-center problem in the Euclidean plane.

Given this perspective, we can see that the k-center problem is equivalent to the following problem:

k-center problem (equivalent form): Given a set P of n points in space and an integer $k \leq n$, find the minimum radius Δ and a set of balls of radius Δ centered at k points of P such that P lies within the union of these balls.

Greedy Approximation Algorithm: Like many clustering problems, the k-center problem is known to be NP-hard, and so we will not be able to solve it exactly. (We will show this later this semester for a graph-based variant of the k-center problem.) Today, we will present a simple greedy algorithm that does not produce the optimum value of Δ, but only an approximation to its value. Our algorithm will produce an estimate that is at most twice as large as the optimum value of Δ.

Let us start with a couple of useful definitions. Given a set $C = \{c_1, \ldots, c_k\}$ of centers and for any $c_i \in C$, define the *bottleneck distance* of c_i to be the distance to its farthest point in $N(c_i)$, that is,

$$\Delta(c_i) = \max_{v \in N(c_i)} \delta(v, c_i).$$

Clearly, the maximum *bottleneck distance* over all the centers is just $\Delta(C)$.

Intuitively, if we think of the cluster centers as the locations of Starbucks (or your favorite retailer), then each customer (point in P) is associated with the closest Starbucks. The set $N(c_i)$ are the customers that go to the ith Starbucks location, and $\Delta(c_i)$ is the maximum distance any of these customers needs to travel to get to c_i. $\Delta(C)$ is the maximum distance that *anyone* needs to travel to their nearest Starbucks.

The greedy algorithm begins by selecting any point of P to be the initial center g_1. We then repeat the following process until we have k centers. Let $G_i = \{g_1, \ldots, g_i\}$ denote the current set of centers. Recall that $\Delta(G_i)$ is the maximum distance of any point of P from its nearest center. Let u be the point achieving this distance. Intuitively, u is the *most dissatisfied customer*, since he/she has to drive the farthest to get to the nearest Starbucks. The greediest way to satisfy u is to put the next center directly at u. (Thus plopping the next Starbucks right on top of u's house. Are you satisfied now?) In other words, set

$$g_{i+1} \leftarrow u \qquad \text{and} \qquad G_{i+1} \leftarrow G_i \cup \{g_{i+1}\}.$$

The pseudocode is presented in the code block below. The value $d[u]$ denotes the distance from u to its closest center. (We make one simplification, by starting with G being empty. When we select the first center, all the points of P have infinite distances, so the initial choice is arbitrary.)

It is easy to see that the algorithm's running time is $O(kn)$. In general $k \leq n$, so this is $O(n^2)$ in the worst case. One step of the algorithm is illustrated in Fig. 3. Assuming that we have three centers $G = \{g_1, g_2, g_3\}$, let g_4 be the point that is farthest from its nearest center (g_1 in this case). In each step we create a center at g_4, so now $G = \{g_1, \ldots, g_4\}$. In anticipation of the next step, we find the point that maximizes the distance to its closest center (g_5 in this case), and if the algorithm continues, it will be the location of the next center.

Approximation Bound: Now, let us show that this algorithm is at most a factor of 2 from the optimum. Let $G = \{g_1, \ldots, g_k\}$ denote the set of centers computed by the greedy algorithm, and let $\Delta(G)$ denote its bottleneck distance. Let $O = \{o_1, \ldots, o_k\}$ denote the optimum set of centers, that is the set of k centers such that $\Delta(O)$ is the smallest possible. We will show that greedy is at most twice as bad as optimal, that is,

$$\Delta(G) \;\leq\; 2\Delta(O).$$

Greedy Approximation for k-center

```
GreedyKCenter(P, k) {
    G = empty
    for each u in P do                // initialize distances
        d[u] = INFINITY

    for (i = 1 to k) {
        Let u be the point of P such that d[u] is maximum
        Add u to G                    // u is the next cluster center
        for (each v in P) {           // update distance to nearest center
            d[v] = min(d[v], distance(v,u))
        }
        Delta = max_{v in P} d[v]     // update the bottleneck distance
    }
    return (G, Delta)                 // final centers and max distance
}
```

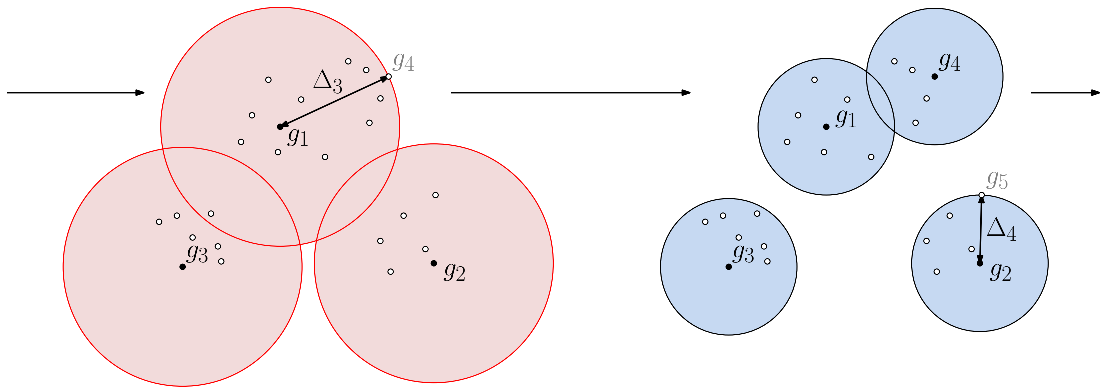

Fig. 3: Greedy approximation to k-center (from stage 3 to 4).

Of course, we don't know what O is or even what $\Delta(O)$ is. Our approach will be to determine a lower bound $\Delta_{\min}$ on the optimum bottleneck distance ($\Delta_{\min} \leq \Delta(O)$). We will then show that our algorithm produces a value that is at most twice this lower bound value ($\Delta(G) \leq 2\Delta_{\min}$). It will follow therefore that $\Delta(G) \leq 2\Delta(O)$.

The analysis is based on the following three claims, each of which is quite straightforward to prove. Define G_i to be the set of greedy centers after the ith execution of the algorithm, and let $\Delta_i = \Delta(G_i)$ denote its overall bottleneck distance (the fartest any point is from its closest center in G_i. The greedy algorithm stops with g_k, but for the sake of the analysis it is convenient to consider the next center to be added if we ran it for one more iteration. That is, define g_{k+1} to be the point of P that is maximizes the distance to its closest center in G_k. (This distance is $\Delta(G_k)$.) Also, define $G_{k+1} = \{g_1, \ldots, g_{k+1}\}$.

Claim 1: For $1 \leq i \leq k+1$, $\Delta_{i+1} \leq \Delta_i$. That is, the sequence of bottleneck distances is monotonically nonincreasing. (In Fig. 3 this is represented by the fact that the radii of the covering disks decrease with each stage.)

Proof: Whenever we add a new center, the distance to each point's closest center will either be the same or will decrease. Therefore, the maximum of such a set can never increase.

Claim 2: For $1 \leq i \leq k+1$, every pair of greedy centers in G_i is separated by a distance of at least Δ_{i-1}.

Proof: Consider the ith stage. By the induction hypothesis, the first $i-1$ centers are separated from each other by distance $\Delta_{i-2} \geq \Delta_{i-1}$. The ith center is, by definition, at distance Δ_{i-1} from its closest center, and therefore it is at distance at least Δ_{i-1} from all the other centers.

Claim 3: Let $\Delta_{\min} = \Delta(G)/2$. Then for any set C of k cluster centers, $\Delta(C) \geq \Delta_{\min}$.

Proof: By definition of $\Delta(C)$ we know that every point of P lies within distance $\Delta(C)$ of some point of C, and since $G \subseteq P$, this is true for G as well. Since $|G_{k+1}| = k+1$, by the pigeonhole principle, there exists at least two centers $g, g' \in G_{k+1}$ that are in the same neighborhood of some center $c \in C$, that is, $\max(\delta(g,c), \delta(g',c)) \leq \Delta(c)$. Since $g, g' \in G_{k+1}$, by Claim 2, $\delta(g, g') \geq \Delta_k = \Delta(G)$. By applying the triangle inequality to the triple (g, c, g') and symmetry of the distance function, we have

$$
\begin{aligned}
\Delta(G) &\leq \delta(g,g') \leq \delta(g,c) + \delta(c,g') \quad \text{(by triangle inequality)}\\
&\leq \delta(g,c) + \delta(g',c) \quad \text{(by distance symmetry)}\\
&\leq \Delta(c) + \Delta(c) \leq \Delta(C) + \Delta(C) = 2\Delta(C).
\end{aligned}
$$

Therefore, $\Delta(C) \geq \Delta(G)/2 = \Delta_{\min}$, as desired.

Now, by applying Claim 3 to O, we have $\Delta(O) \geq \Delta_{\min}$. By definition, $\Delta_{\min} = \Delta(G)/2$, and so $\Delta(G) \leq 2\Delta(O)$, completing the analysis.

You might wonder whether this bound is tight. We will leave it as an exercise to prove that there is a input such that (if you are unlucky about how you choose the first point) the greedy algorithm returns a value that is arbitrarily close to twice that of the optimum.