CMSC 451: Lecture 6
k-Center Clustering and Gonzalez's Algorithm

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 vector designating its *properties* or *features*. This is a numeric vector whose length may range from tens up 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 a partition of the point set into subsets, called *clusters*, denoted $\{C_1, \ldots, C_k\}$ (see Fig. 1(b)). In *center-based clustering*, the output is a set of cluster *center points*, $\mathcal{C} = \{c_1, \ldots, c_k\}$, and the clusters themselves are implicitly defined by the closest center (see Fig. 1(c)). In particular, a point lies in the ith cluster if its closest cluster center is c_i. Let $N(c_i)$ denote this *neighborhood* of nearest points. Given a set of k centers, we can assign points to their closest center in $O(nk)$ time.

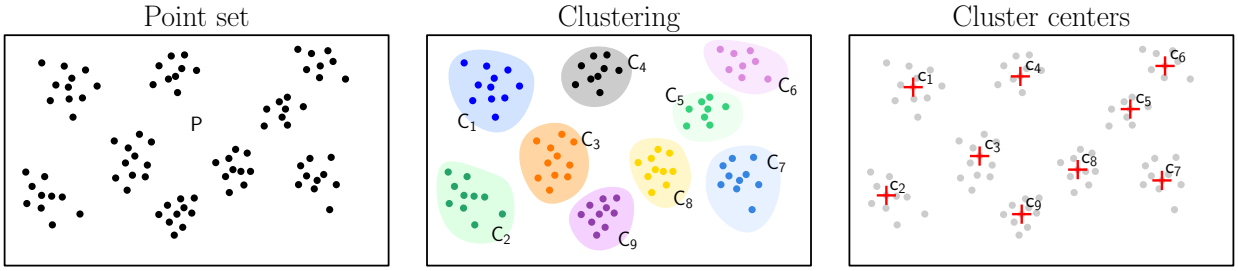

Fig. 1: Center-based geometry clustering.

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. (Points used in an optimization problem that are not drawn from the set are sometimes called *Steiner points*.)

Three Center-Based Clusterings: There are a number of ways to define center-based clusterings. Before getting into our main topic, called the k-center problem, we will contrast three alternatives. In all three cases, we are told the desired number of clusters. (Determining the desired number of clusters is a tricky question, which we will not address here.)

k-Median: Compute the k center points to minimize the *sum of Euclidean distances* to the nearest cluster center.

k-Means: Compute the k center points to minimize the *sum of squared Euclidean distances* to the nearest cluster center.

k-Center: Compute the k center points to minimize the *maximum Euclidean distance* to the nearest cluster center.

Which of these is best? It really depends on your application. In order to better understand the differences between these criteria, it is illustrative to consider the single-cluster version of all three.

1-Median: The 1-median point is the single point that minimizes the sum of distances. In 1-dimensional space, this is realized by the *median*[1] of the point set (see Fig. 1(a)).

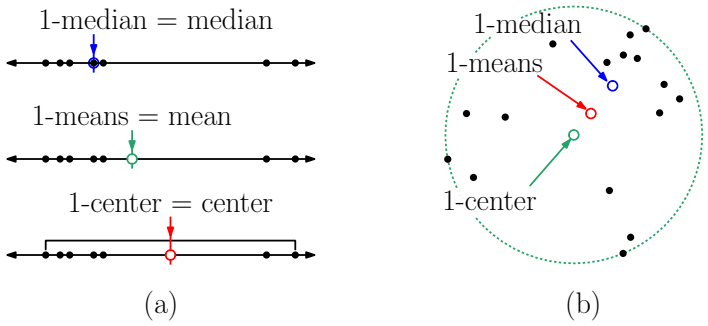

Fig. 2: 1-median, 1-means, and 1-center in 1- and 2-dimensional space.

In higher dimensions, there is no simple way to describe this point. The problem of computing the point in multi-dimensional space that minimizes the sum of Euclidean distances (see Fig. 1(b)). Computing this point is called the *Fermat-Weber problem* (named in honor of the famous 17th-century French mathematician, Pierre de Fermat, and the early 20th-century German economist Alfred Weber). There is no known exact algorithm, and best solutions only can approximate its coordinates.

1-Means: In 1-dimensional space, the 1-means problem is realized by the mean of the set. In higher dimensions, this is the *centroid* of the set. This can be easily computed by taking the mean coordinate along each of the coordinate axes. One of the reasons for the popularity of k-means is that the cluster centers are very easy to compute in $O(n)$ time.

[1]This can be proved by a simple variational argument. Suppose the point is not at the median. There are more points on one side, say the right, than on the other. By moving the point some distance Δx to the right, more distances on the right decrease by Δx to compensate for the distances on the left that increase by Δx.

There is a famous heuristic for the k-means problem, called *Lloyd's algorithm*. It starts with an initial set of centers, and then repeatedly moves each center to the centroid of its cluster, and then recomputes the clusters (based on changes in the nearest neighbors). It converges to a local minimum, but this may not be the global minimum.

1-Center: The solution to the 1-center problem is the center of the minimum radius ball that contains the points. In the 1-dimensional case, this is midway between the minimum and maximum. In higher dimensions, this is the center of the smallest enclosing ball for the point set. It is not obvious how to compute this ball. There is an amazingly simple $O(n)$-time randomized algorithm. The algorithm was developed by many researchers over time, but the simplest variant was presented and analyzed by the German computer-scientist Raimund Seidel.)

The k-Center Problem: For the rest of the lecture, we will focus on the k-center problem. Let P be a set of n points in some metric space. We are given a *distance function*, δ, which computes the distance between any two points $p, q \in P$. We assume that δ is a *metric*, meaning that it satisfies the following properties for any $p, q, r \in P$:

Positivity: $\delta(p, q) \geq 0$ and $\delta(p, q) = 0$ if and only if $p = q$.

Symmetry: $\delta(p, q) = \delta(q, p)$

Triangle inequality: $\delta(p, r) \leq \delta(p, q) + \delta(q, r)$

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 metric on P. Here is a formal problem statement.

k-center problem: Given a set P of n points in a metric 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.

We can state the problem in a more "mathy" form in terms of an objective function

$$\Delta_P(C) \;=\; \max_{p \in P} \; \min_{c \in C} \delta(p, c).$$

This just computes the maximum distance of any point p of P to its closest center in C. Then, the k-center problem is the optimization problem of computing the subset of k-centers from P that minimizes this function, that is,

$$\min_{\substack{C \subseteq P \\ |C| = k}} \; \Delta_P(C).$$

As mentioned above, we can view the k-center problem as finding the smallest radius r such that it is possible cover P using k balls of radius r, each centered at some point of P. The minimum radius r is just the optimum value of $\Delta_P(C)$ a covering problem by balls, and the points of C are the centers of these 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. 3(b)).

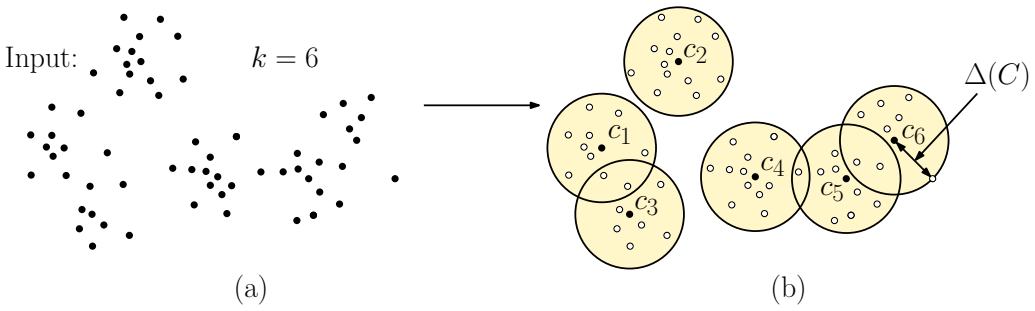

Fig. 3: The k-center problem ($k = 6$) in the Euclidean plane.

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.

Gonzalez's 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.) Instead, we will present simple greedy algorithm, due to Teofilo Gonzalez, an algorithms researcher from UCSB. It does not produce the optimum value of Δ, but the result is at most twice as large as the optimum value of Δ.

Let us start with a couple of useful definitions. Recall that P is our point set, and consider any set $C = \{c_1, \ldots, c_k\}$ of cluster centers. (Since P will be fixed, we'll omit explicit references to it in our notation.) For any $c_i \in C$, recall that $N(c_i)$ are the points of its cluster. Define the *bottleneck distance* of c_i to be the distance to its farthest point in $N(c_i)$, that is,

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

and the overall bottleneck distance is

$$\Delta(C) \;=\; \max_{1 \leq i \leq k} \Delta(c_i).$$

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

We will build up the greedy centers, denoted $G = \{g_1, g_2, \ldots\}$ as follows. The greedy algorithm begins by selecting any point of P to be the initial center g_1. (There may be better ways to select the point in practice, but it won't affect the worst-case analysis.) We then repeat the following process until we have k centers. For $0 \leq i < k$, let $G_i = \{g_1, \ldots, g_i\}$ denote the current set of greedy centers. Recall that $\Delta(G_i)$ is the maximum distance of any point of P from its nearest center. Let p be the point achieving this distance. Intuitively, p is the *most dissatisfied customer*, since he/she has to drive the farthest to get to the nearest Starbucks. The greediest way to satisfy p is to put the next center directly at p. (Thus plopping the next Starbucks right on top of p's house. Are you satisfied now?) In other words, set

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

The pseudocode is presented in the code block below. The value $d[p]$ denotes the distance from p 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.)

Gonzalez's Algorithm for k-center

```
gonzalez(P, k) {                        // Gonzalez's algorithm for k-center on P
    G = empty-set
    for each (p in P)                   // initialize distances
        d[p] = +infinity

    for (i = 1 to k) {
        p = point of P that maximizes d[p]
        add p to G                      // p is the next cluster center
        for each (q in P) {             // update distances to nearest center
            d[q] = min(d[q], dist(p, q))
        }
    }
    return (G, Delta)                   // final centers and max distance
}
```

It is easy to see that the algorithm's running time is $O(kn)$. One step of the algorithm is illustrated in Fig. 4. 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 two from the optimum. Given a point set P, let $G = \{g_1, \ldots, g_k\}$ denote the set of centers computed by the greedy algorithm, and let $\Delta_P(G)$ 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 $\Delta_P(O)$ is the smallest possible. Since P is fixed throughout, we'll drop the subscript.

Theorem: For a point set P, let G denote the output of Gonzalez's algorithm and O denote the optimum k-center solution. Then $\Delta(G) \leq 2\Delta(O)$.

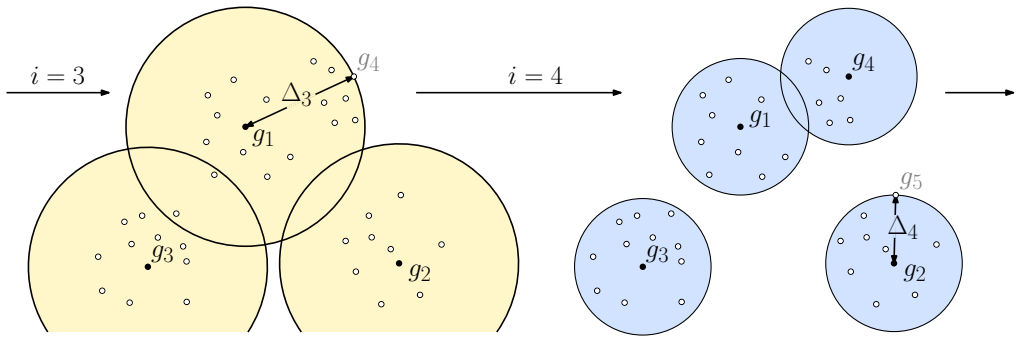

Fig. 4: Greedy approximation to k-center (from stage 3 to 4). (Not necessarily accurate to the algorithm's behavior.)

How can we hope to do this, since (assuming you cannot solve NP-hard problems), you cannot know what $\Delta(O)$ even is! Our approach will be to define estimate, denoted $\Delta_{\min}$ (see Claim 3 below). We will show that this estimate is a lower bound on the optimum, that is, $\Delta_{\min} \leq \Delta(O)$. We will also show that $2\Delta_{\min}$ provides an upper bound on greedy, that is, $\Delta(G) \leq 2\Delta_{\min}$. Combining these yields

$$\Delta(G) \;\leq\; 2\Delta_{\min} \;\leq\; 2\Delta(O),$$

which will imply that greedy is a 2-factor approximation to the optimum. Let's carry out this plan.

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). Thus, $\Delta(G) = \Delta_k$.

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 $\Delta(G)$. 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. 4 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.

Since $\Delta(G) = \Delta_k$, we have:

Corollary: Every pair of greedy centers in G_{k+1} is separated by a distance of at least $\Delta(G)$.

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 $\Delta(C)$,every point of P lies within distance $\Delta(C)$ of some point of C (see Fig. 5(a)). Since $G_{k+1} \subseteq P$, this is true for G_{k+1} as well. Since $|G_{k+1}| = k+1$, and C has only k clusters, 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 $c \in C$ (see Fig. 5(b) and (c)).

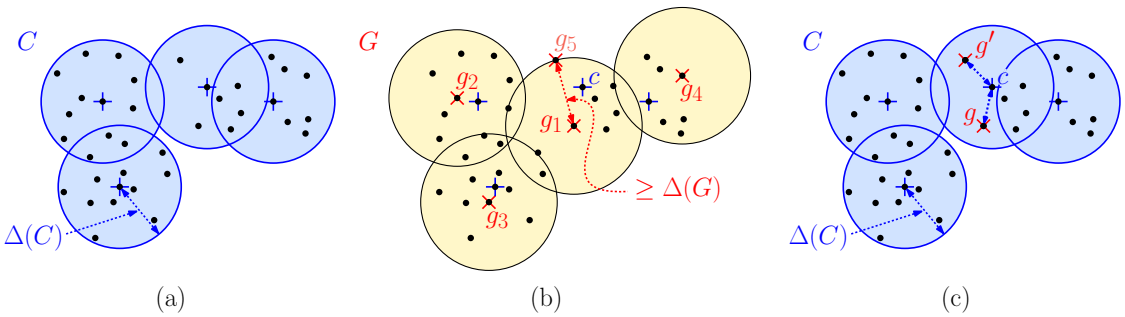

Fig. 5: Proof of Claim 3. (Not necessarily accurate to the algorithm's behavior.)

This implies that both $\delta(g, c)$ and $\delta(g', c)$ are less than or equal to $\Delta(c)$, which is less than or equal to $\Delta(C)$. Since $g, g' \in G_{k+1}$, by the corollary to Claim 2, $\delta(g, g') \geq \Delta(G)$. Combining these observations with basic properties of metric spaces, we have

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

Rewriting this, we have $\Delta(C) \geq \Delta(G)/2 = \Delta_{\min}$, as desired.

Since Claim 3 applies to any set of k clusters, it applies to the optimum, O. We conclude that $\Delta(O) \geq \Delta_{\min}$. 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.

Example: As an example, let's consider a set of points uniformly distributed along a line segment on the x-axis along the interval $[0, 12]$ and $k = 3$ (see Fig. 6).

The optimal center placement is at $x = \{2, 6, 10\}$, which yields a radius of 1. In contrast, if we start Gonzalez with the leftmost point, $x = 0$, (since the starting point is arbitrary), it will select the second point at $x = 12$, and the final point at $x = 6$. This yields a covering radius 1.5. Thus, in this case Gonzalez is suboptimal by a factor of $1.5/1 = 1.5$. (Can you come up with an example where the ratio is exactly 2?)

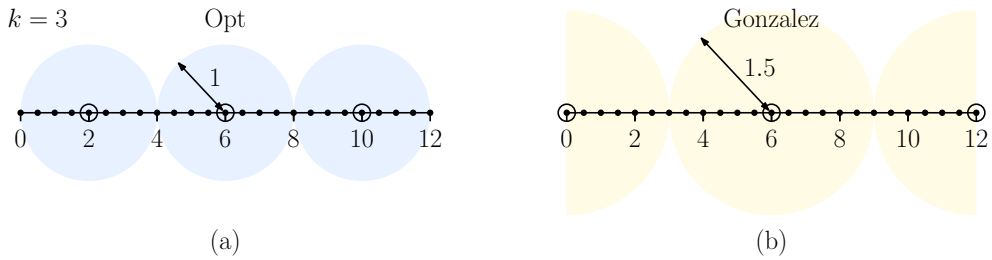

Fig. 6: Example of Gonzalez for $k = 3$. Approximation ratio is

Summary: We introduced a number of common center-based clustering methods (k-means, k-median, and k-center). All of these problems are NP-hard. We presented a simple greedy algorithm, Gonzalez's algorithm, for the k-center problem, and we showed that it produces a result that is at most twice that of the optimal, for any set of points (in any metric space) and any value of k.