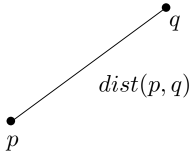
Approximation Algorithms 4: k-Center

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

Given 2D points p and q, we use *dist*(p, q) to represent their Euclidean distance.



In this lecture, we will make the assumption that $dist(p, q)$ can be computed in polynomial time.

P = a set of n points in 2D space.

Given a point $p \in P$, define its distance to a subset $C \subseteq P$ as

$$dist_C(p) = \min_{c \in C} dist(p, c).$$

The **penality** of C is

$$pen(C) = \max_{p \in P} dist_C(p).$$

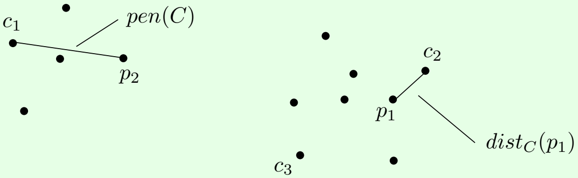
The k-Center Problem: Find a subset $C \subseteq P$ with size $|C| = k$ that has the smallest penalty.

Example:

$P =$ the set of black points

$k = 3$

$C = \{c_1, c_2, c_3\}$



The problem is NP-hard.

- No one has found an algorithm solving the problem in time polynomial in n and k.
- Such algorithms cannot exist if $\mathcal{P} \neq \mathcal{NP}$.

$\mathcal{A}$ = an algorithm that, given any legal input P, returns a subset of P with size k.

Denote by OPT_P the smallest penalty of all subsets $C \subseteq P$ satisfying $|C| = k$.

> $\mathcal{A}$ is a ρ-**approximate algorithm** for the k-center problem if, for any legal input P, $\mathcal{A}$ can return a set C with penalty at most $\rho \cdot OPT_P$.

The value ρ is the **approximation ratio**.
We say that $\mathcal{A}$ achieves an approximation ratio of ρ.

Consider the following algorithm:

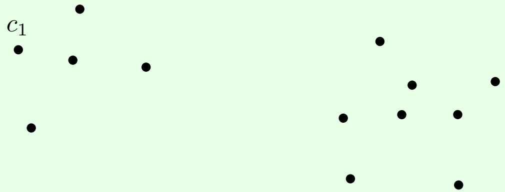
Input: P

1. $C \leftarrow \emptyset$
2. add to C an arbitrary point in P
3. **for** $i = 2$ to k **do**
4. $\quad$ $p \leftarrow$ a point in P with the maximum $dist_C(p)$
5. $\quad$ add p to C
6. return C

The algorithm can be easily implemented in polynomial time.

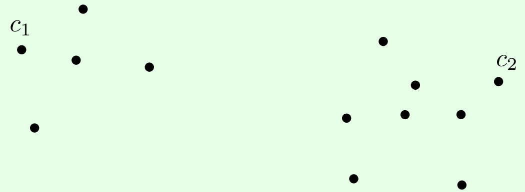
Later, we will prove that the algorithm is 2-approximate.

Example: $k = 3$



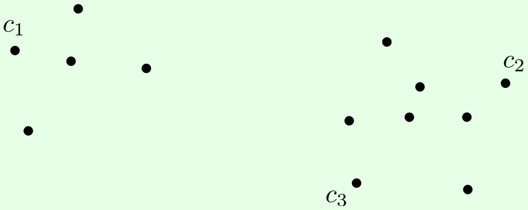
Initially, $C = \{c_1\}$

Example: $k = 3$



After a round, $C = \{c_1, c_2\}$

Example: $k = 3$



After another round, $C = \{c_1, c_2, c_3\}$

Theorem: The algorithm returns a set C with $pen(C) \leq 2 \cdot OPT_P$.

Proof: Let $C^* = \{c_1^*, c_2^*, ..., c_k^*\}$ be an optimal solution, i.e., $pen(C^*) = OPT_P$.

For each $i \in [1, k]$, define P_i^* as the set of points $p \in P$ satisfying

$$dist(p, c_i^*) \le dist(p, c_j^*)$$

for any $j \neq i$.

Observation:
For any point $p \in P_i^*$, $dist(p, c_i^*) = dist_{C^*}(p) \le pen(C^*)$.

Let $C_{ours} = \{c_1, c_2, ..., c_k\}$ be the output of our algorithm, where c_i ($i \in [1, k]$) is the i-th point added to C_{ours}.

Case 1: C_{ours} has a point in each of $P_1^*, P_2^*, ..., P_k^*$.

Consider any point $p \in P$. Suppose that $o \in P_i^*$ for some $i \in [1, k]$. Let c be a point in $C \cap P_i^*$. It holds that:

$$\begin{aligned} dist_{C_{ours}}(p) &\leq dist(c, p) \\ &\leq dist(c, c^*) + dist(c^*, p) \\ &\leq 2 \cdot pen(C^*). \end{aligned}$$

Therefore:

$$pen(C_{ours}) = \max_{p \in P} dist_{C_{ours}}(p) \leq 2 \cdot pen(C^*).$$

Case 2: C_{ours} has no point in at least one of $P_1^*, ..., P_k^*$. Hence, one of $P_1^*, ..., P_k^*$ must cover at least two points — say c_1 and c_2 — of C_{ours}. It thus follows that

$$dist(c_1, c_2) \le dist(c_1, c_i^*) + dist(c_2, c_i^*) \le 2 \cdot pen(C^*).$$

Next, we prove:

> **Lemma:** For any point $p \in P$, $dist_{C_{ours}}(p) \le dist(c_1, c_2)$.

The claim implies $pen(C_{ours}) \le 2 \cdot pen(C^*)$.

Proof of the Lemma:

W.l.o.g., assume that c_2 was picked after c_1 by our algorithm. Consider the moment right before c_2 was picked. At that moment, the set C maintained by our algorithm was a proper subset of C_{ours}.

From the fact that c_2 was the next point picked, we know $dist_C(p) \leq dist_C(c_2)$.

Because $c_1 \in C$, it holds that $dist_C(c_2) \leq dist(c_1, c_2)$.

The lemma then follows because

$$dist_{C_{ours}}(p) \leq dist_C(p) \leq dist_C(c_2) \leq dist(c_1, c_2).$$

□