理论计算机科学导论

上海交通大学
陶表帅
2022年10月24日

前情提要

近似算法

- 许多优化问题是NP-hard的
- 在P ≠ NP的假设下，NP-hard优化问题无法在**多项式时间**内被解决
- 如何处理NP-hard优化问题？
- 近似算法（approximation algorithm）
  - 输出“**近似**”最优解的算法

近似算法定义

- 考虑一个**最小化问题**的算法$\mathcal{A}$。对于输入I，设$\mathcal{A}(I)$为算法输出值，$OPT(I)$为I的最优解（最小值）。$\mathcal{A}$是一个α-近似算法当

$$\forall I: \quad \frac{\mathcal{A}(I)}{OPT(I)} \leq \alpha$$

- 对于**最大化问题**，$\mathcal{A}$是一个α-近似算法当

$$\forall I: \quad \frac{\mathcal{A}(I)}{OPT(I)} \geq \alpha$$

当我们成功设计出一个α-近似算法后……

我们有两个思路去提升**近似比** α:

- 思路1：同样算法，更好的分析（better analysis?）
- 思路2：更好的算法（better algorithms?）

- 想要否定思路1，找一个例子说明该算法在该例子下只能做到α近似。这种例子被称作tight example。
- 想要否定思路2，证明不可近似性。

本节课预览

- 更多的近似算法
  - 基于贪心（greedy）思路的近似算法
- 基本的证明不可近似性之技巧
  - α-不可近似性：如果某问题具有**多项式时间复杂度**的α-近似算法，则P = NP

贪心算法

Greedy Algorithm

贪心算法（Greedy Algorithm）

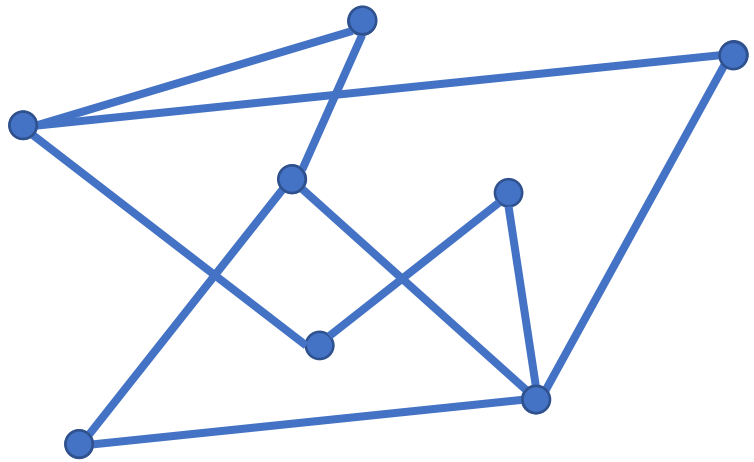
- 每一轮选“局部最优解”
- 每次只看下一步怎么走，不考虑更远的未来

Greedy Algorithm for Minimum Vertex Cover

1. Choose a vertex with maximum degree
2. Remove this vertex and all its incident edges
3. Choose a vertex with maximum degree in the remainder graph
4. Remove this vertex and all its incident edges
5. Keep doing 3 and 4
6. Until all edges are removed

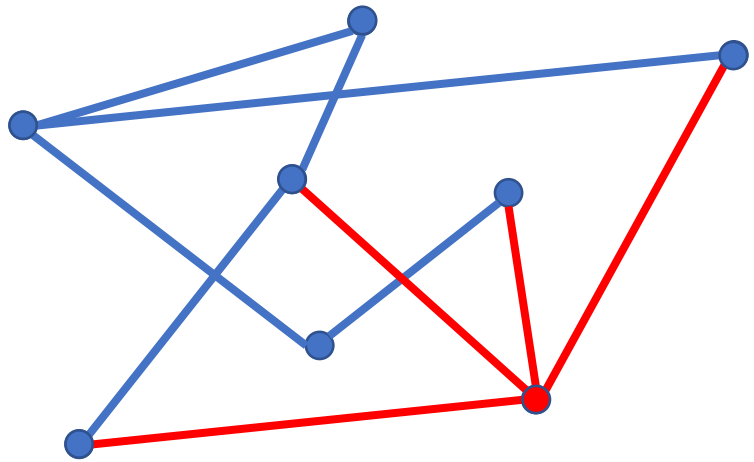
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



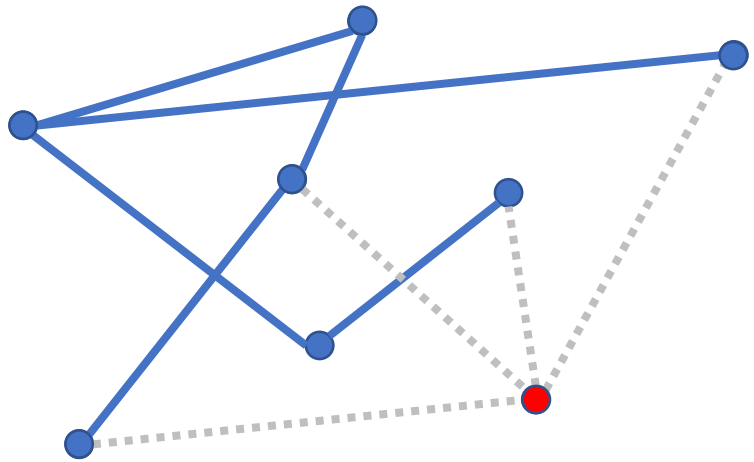
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



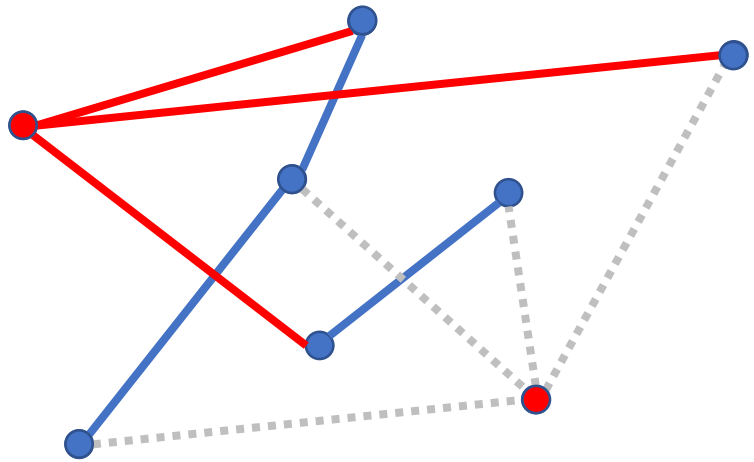
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



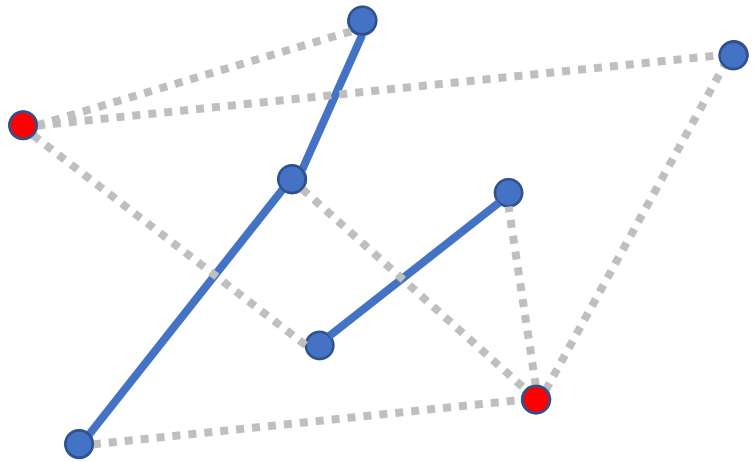
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



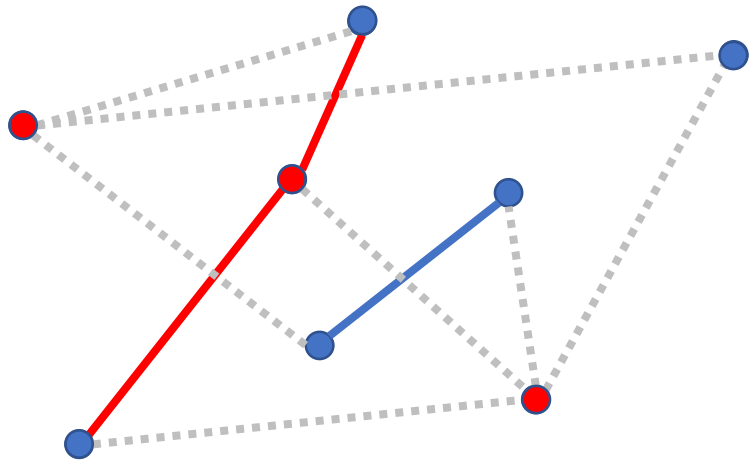
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



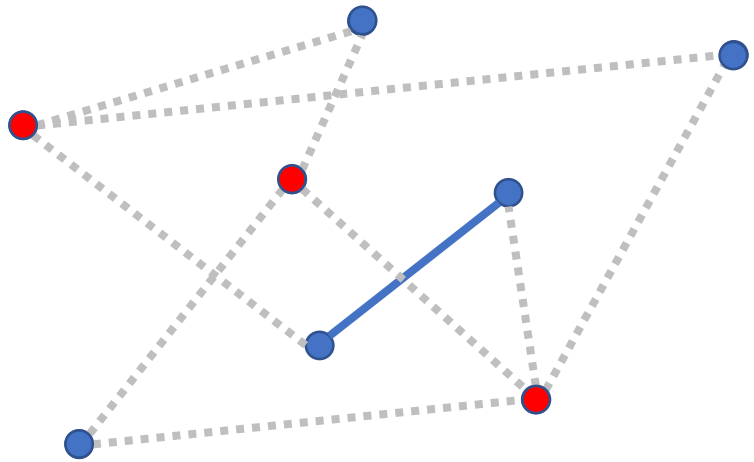
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



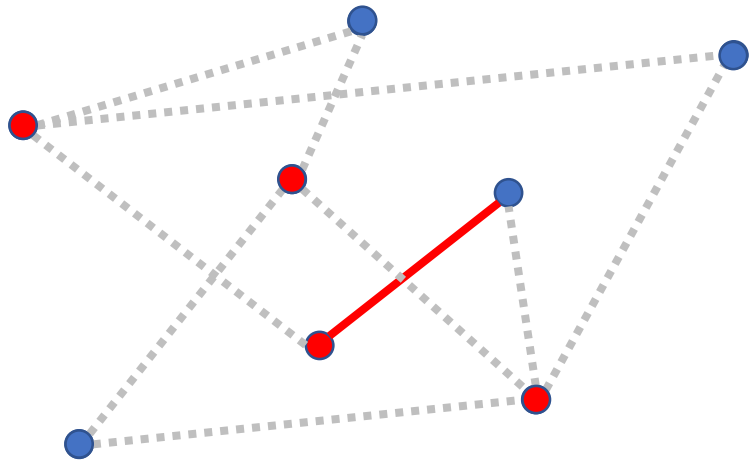
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



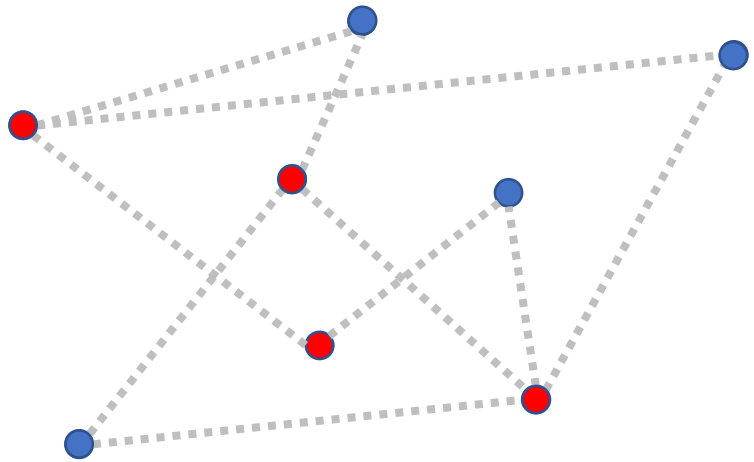
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



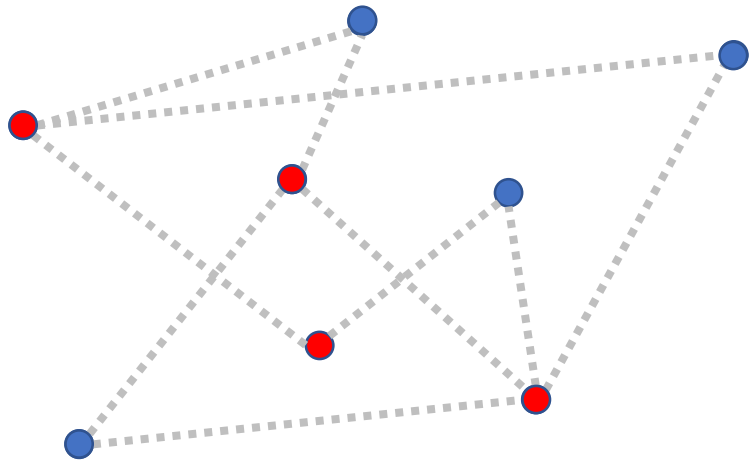
Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



Greedy Algorithm for Minimum Vertex Cover

- Iteratively choose a vertex with maximum degree and remove its incident edges.



DONE!

猜一猜

- 该贪心算法是几的近似算法?
- 你能给出一些例子使得该贪心算法之表现是糟糕的吗?
- 你之例子下，该贪心算法有多糟糕?

贪心算法的表现

贪心算法的表现取决于我们所解决的问题以及其设计：

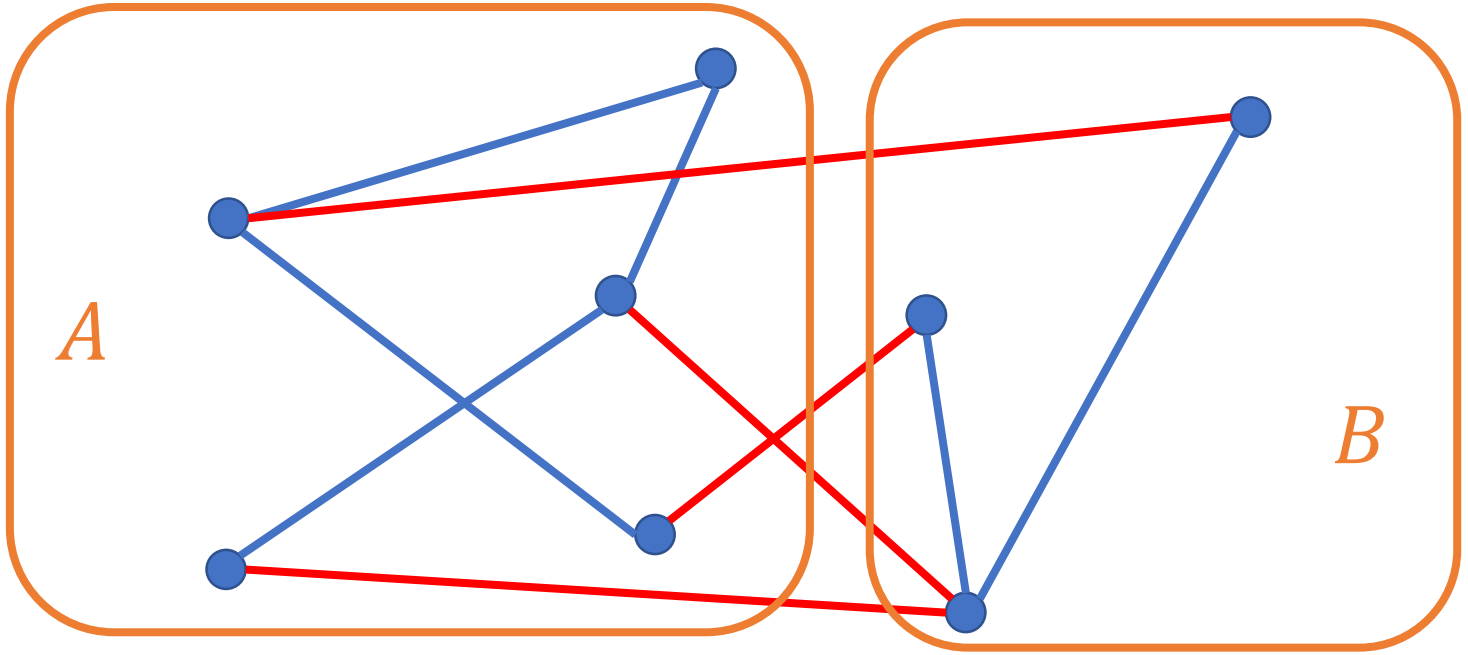
- 有些贪心算法能直接得到最优解
  - Dijkstra算法、Kruskal算法
- 有些贪心算法能提供一个不错的近似算法
  - 【这节课】Max-cut、K-centers
- 有些贪心算法甚至不是一个好的近似算法
  - 最小点集覆盖

Max-Cut问题

Cut

- Given an undirected graph $G = (V, E)$, a cut is a partition of vertices (A, B).
- The value of the cut (A, B), denoted by $c(A, B)$, is the number of the edges between A and B.

$c(A, B) = 4$



Max-Cut

- **[Max-Cut]** Given an undirected graph $G = (V, E)$, find a cut (A, B) with maximum value $c(A, B)$.

- [Karp, 1972] Max-Cut is NP-hard.

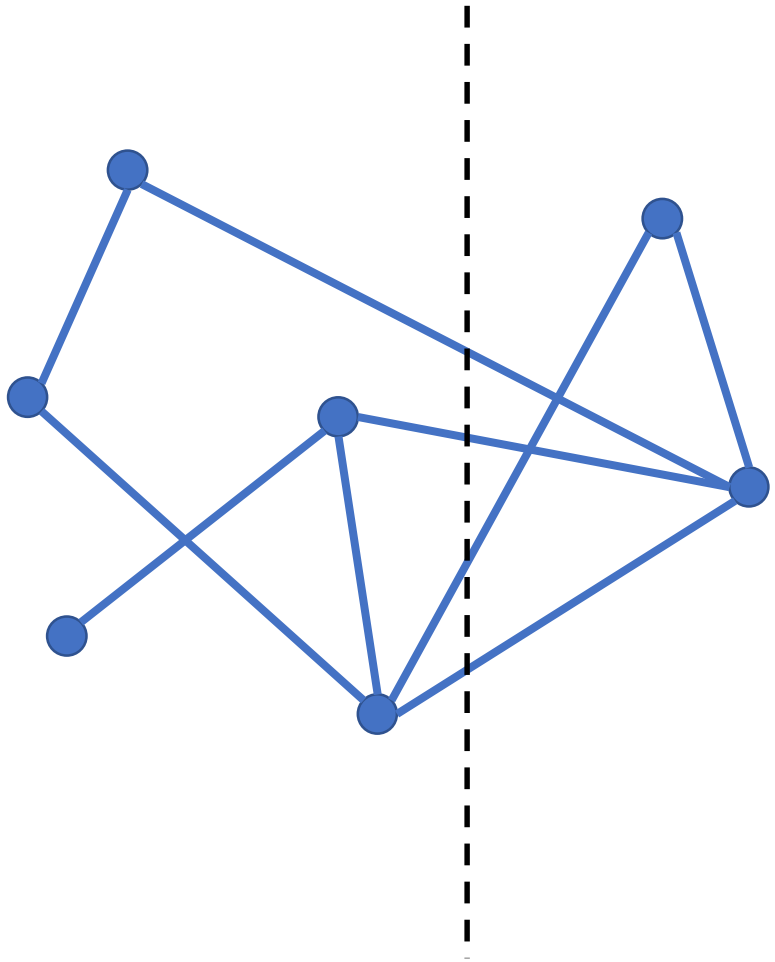
A Greedy Algorithm

1. Start with any partition (A, B).
2. If moving a vertex u from A to B or from B to A increases $c(A, B)$, move it.
3. Terminate until no such movement is possible.

Example

A

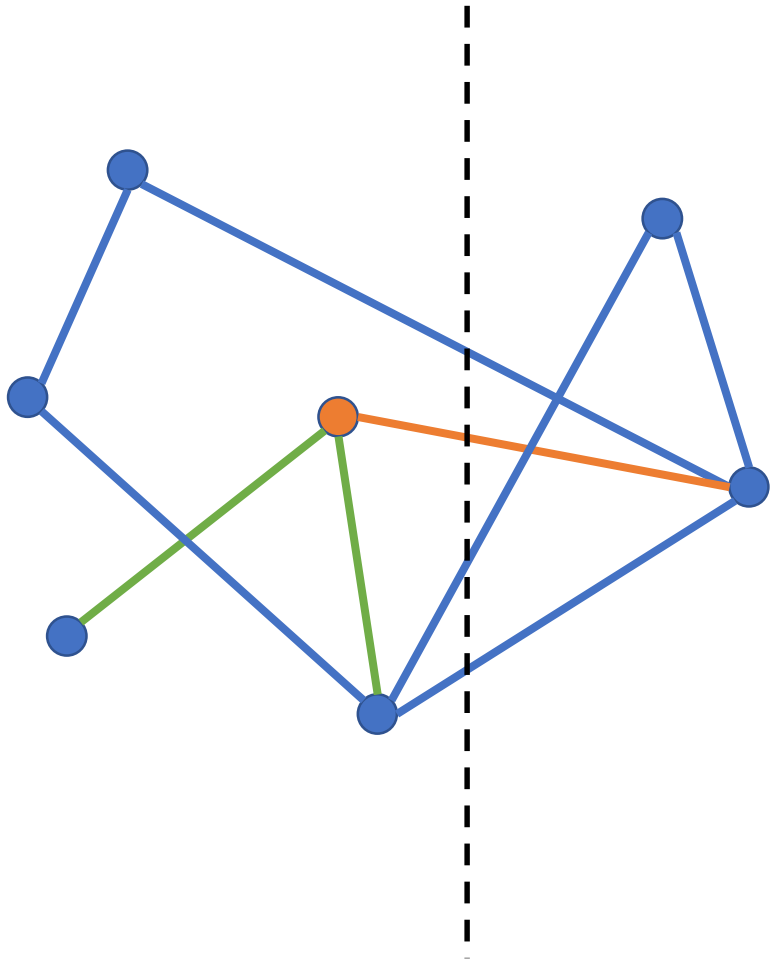
B



Example

A

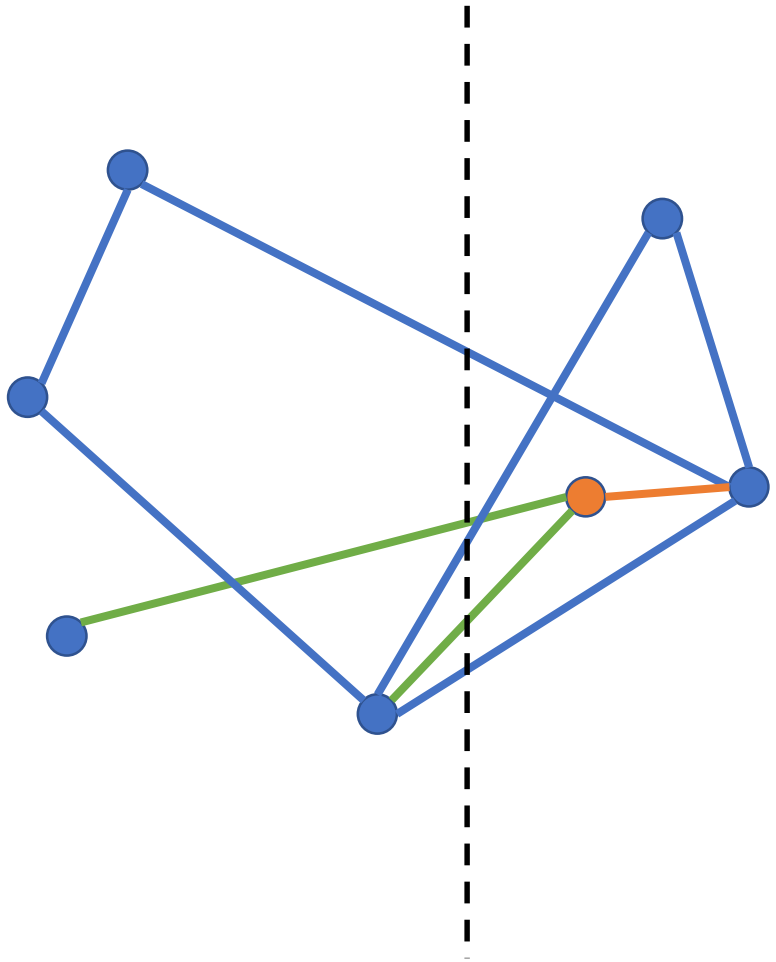
B



Example

A

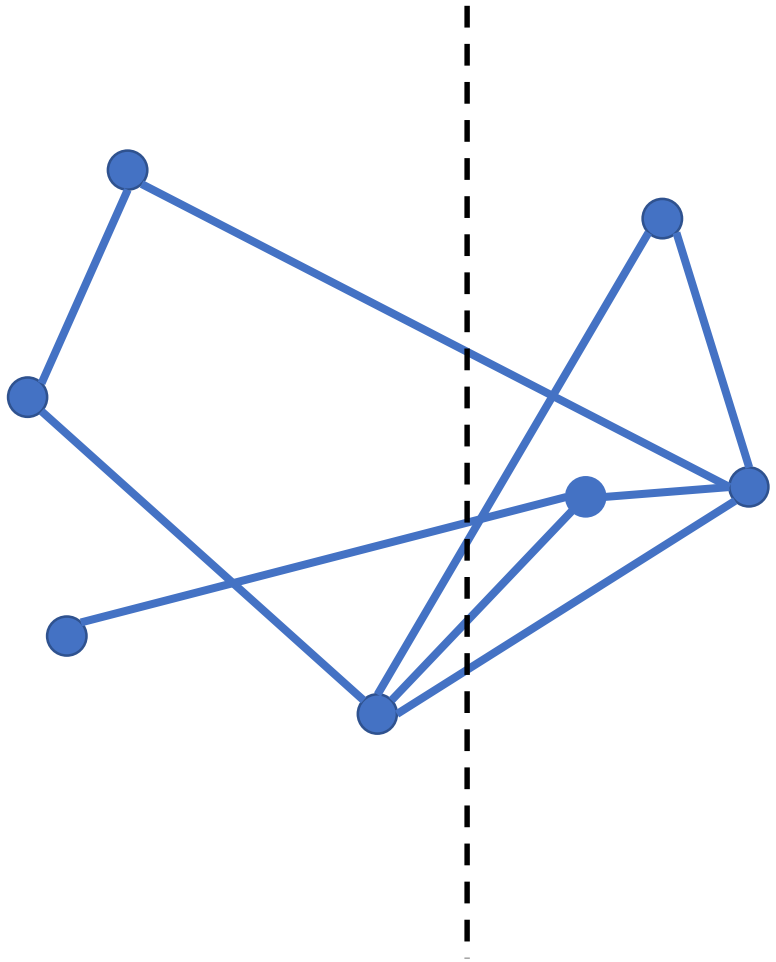
B



Example

A

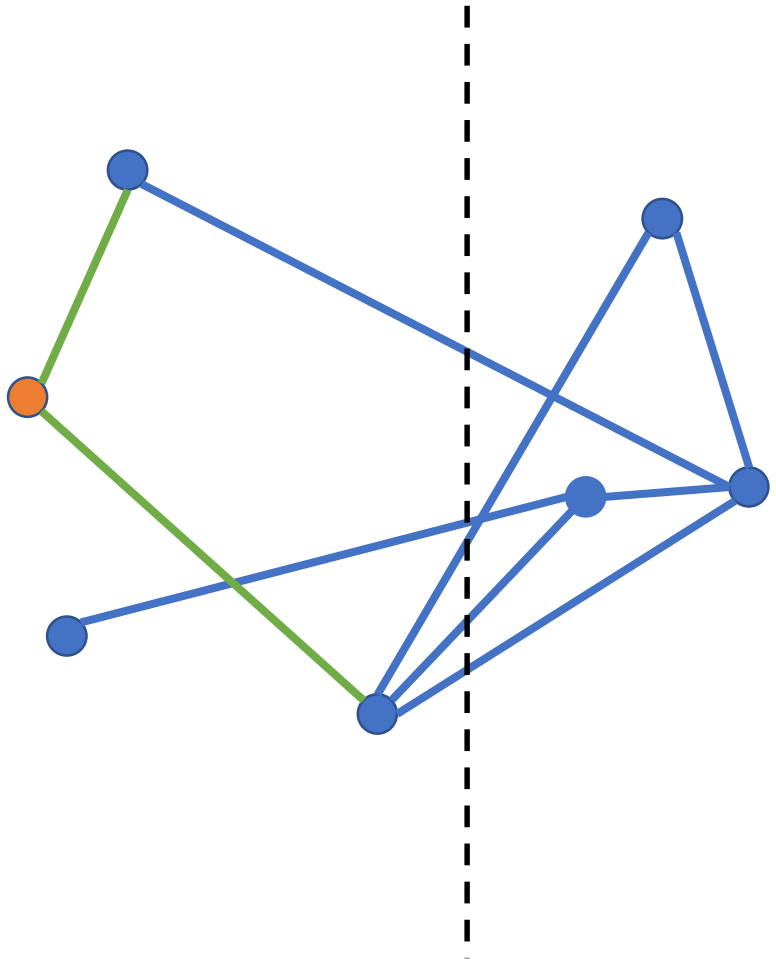
B



Example

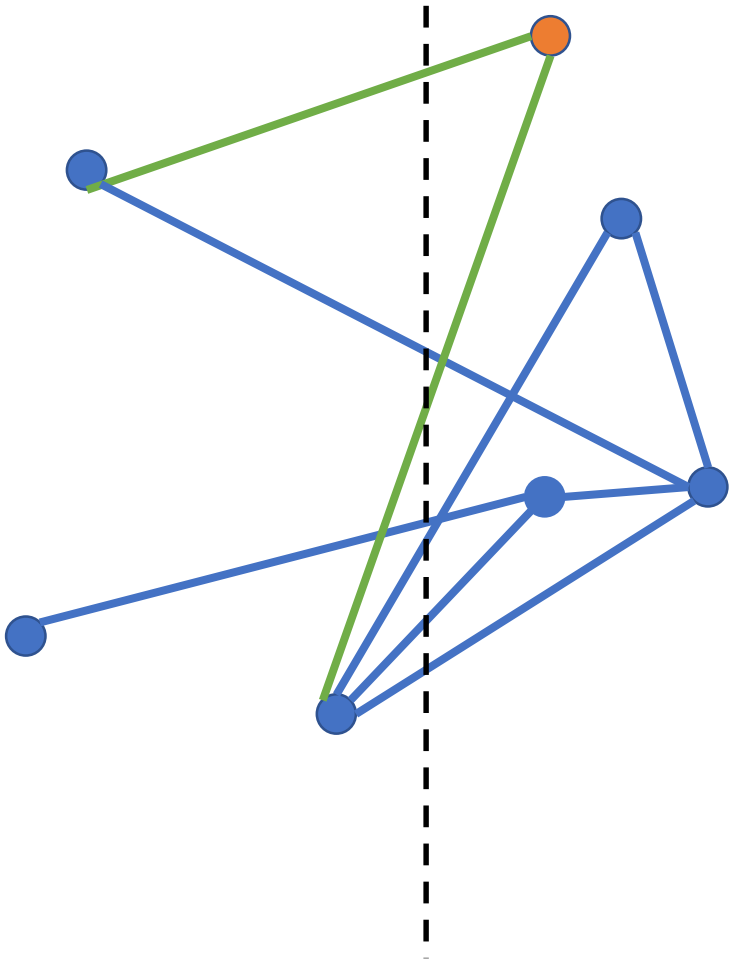
A

B



Example

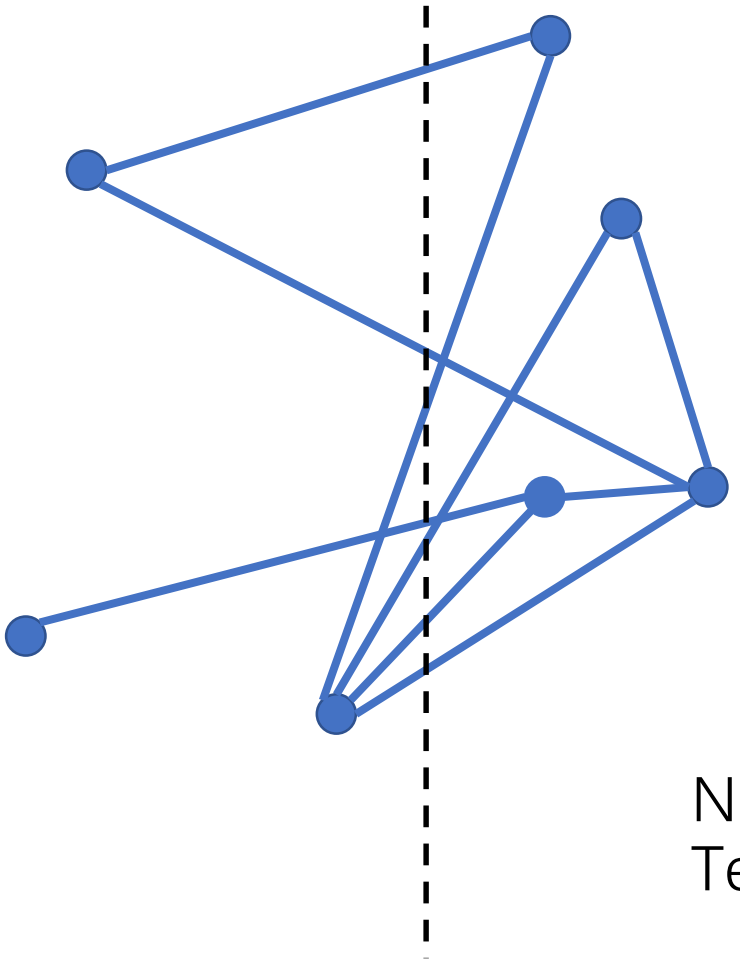
A



B

Example

A



B

No more update can improve.
Terminate…

Time Complexity?

- Each update searches for at most $O(|V|)$ vertices.
- For each vertex, decide if the update is beneficial takes at most $O(|E|)$ time.
- Total number of updates is at most $|E|$.
  - Each update increases the cut size by at least 1.
- Overall: $O(|V||E|^2)$ - polynomial time!

Approximation Guarantee?

- Each vertex u: at least $\frac{1}{2}\deg(u)$ incident edges in the cut.
- Thus,

$$c(A,B) \geq \frac{1}{2}\sum_{u\in V}\frac{1}{2}\deg(u) = \frac{1}{2}|E|.$$

- $|E|$ is an obvious upper bound to OPT.
- Therefore, the local search algorithm is a 0.5-approximation.

Can the algorithm do better than 0.5-approximation?

- No…
- Can you give a tight example?

Are there better approximation algorithms?

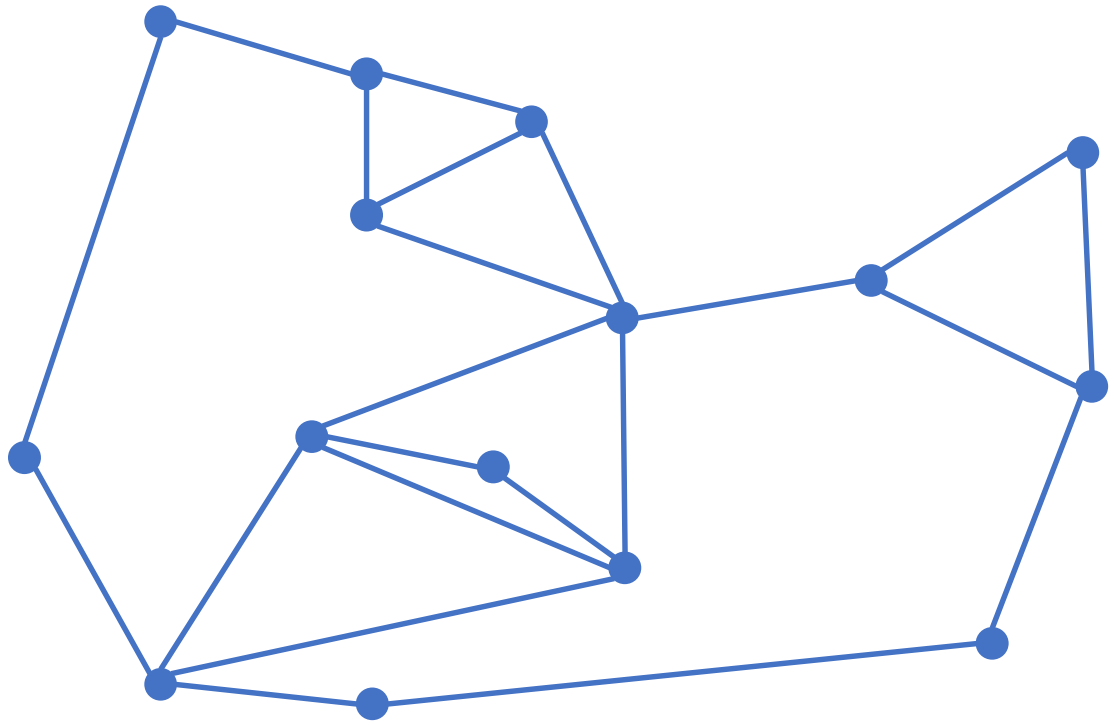
- Yes!
- [Goemans and Williamson] 0.878-approximation
- You may learn it in a future course after two or three years!

k-centers问题

2-近似算法

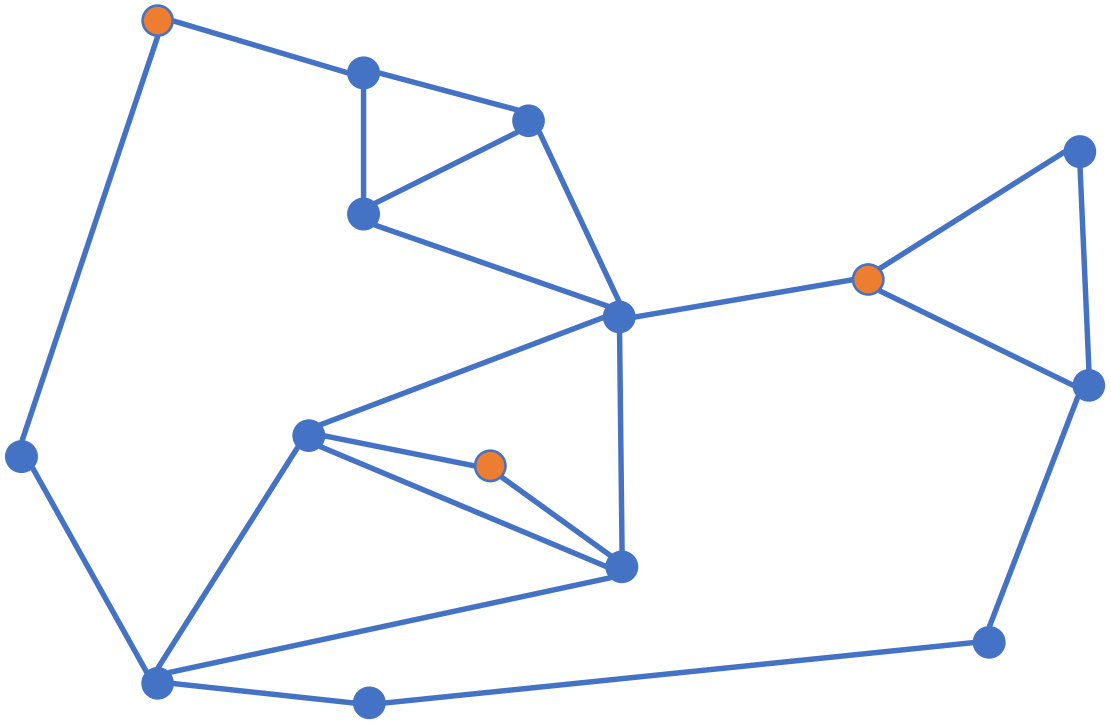
Facility Location

- Build k facilities



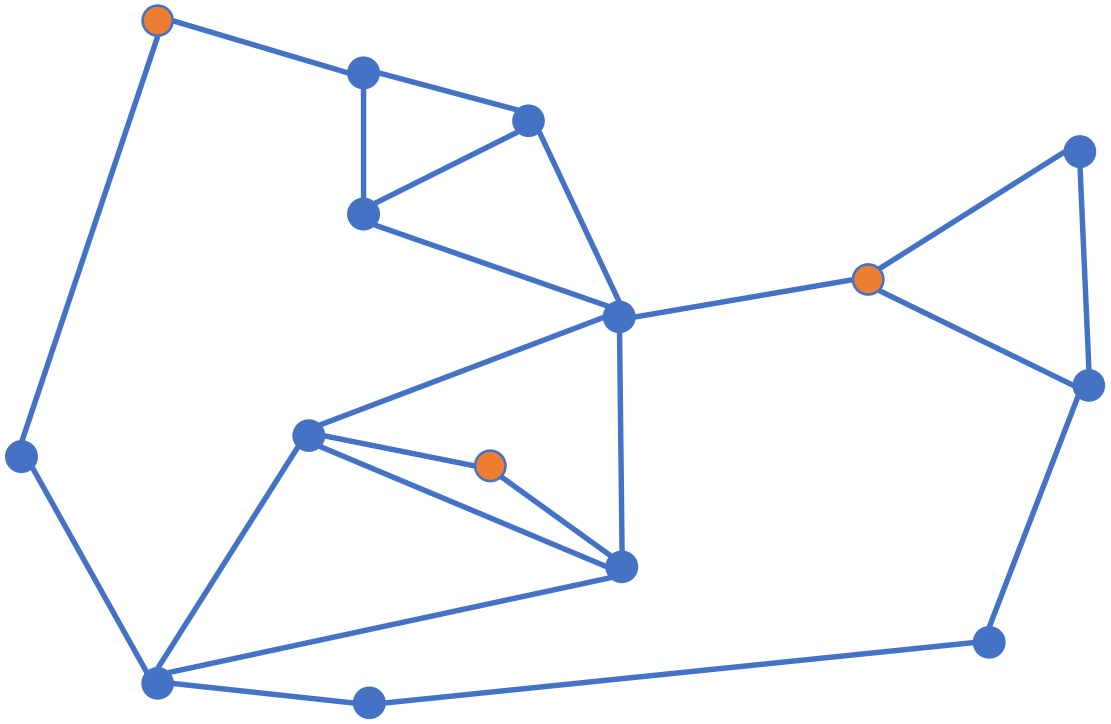
Facility Location

- Build k facilities



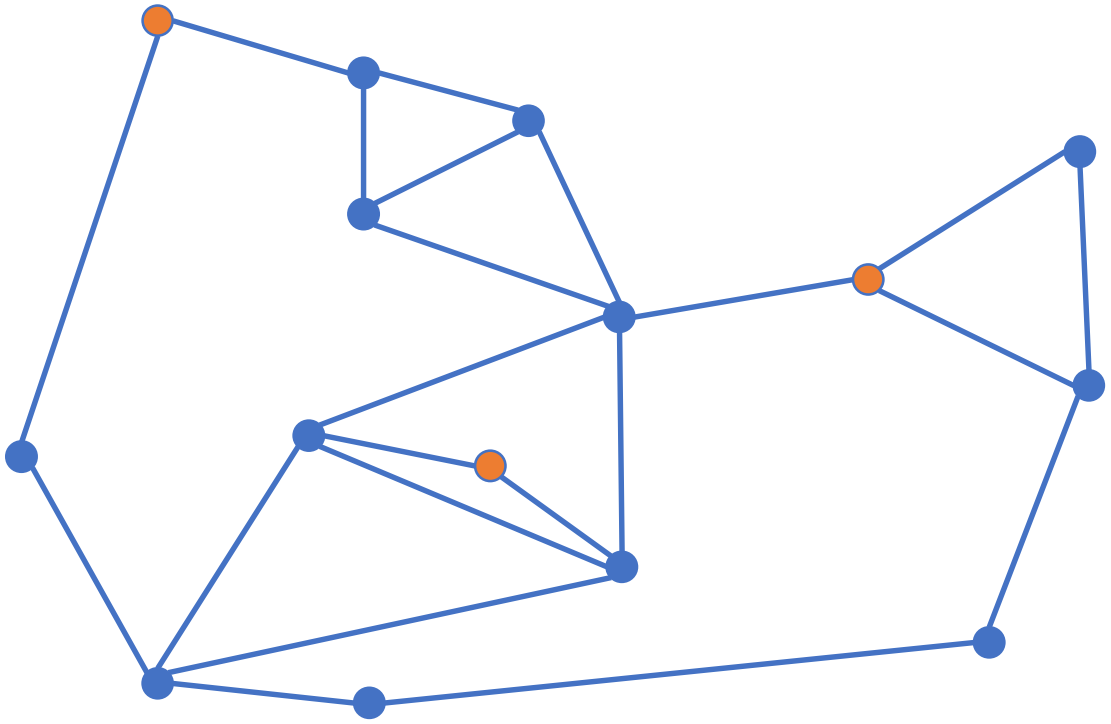
k-centers

- Build k facilities
- Objective: minimize the maximum distance between any vertex to its closest facility



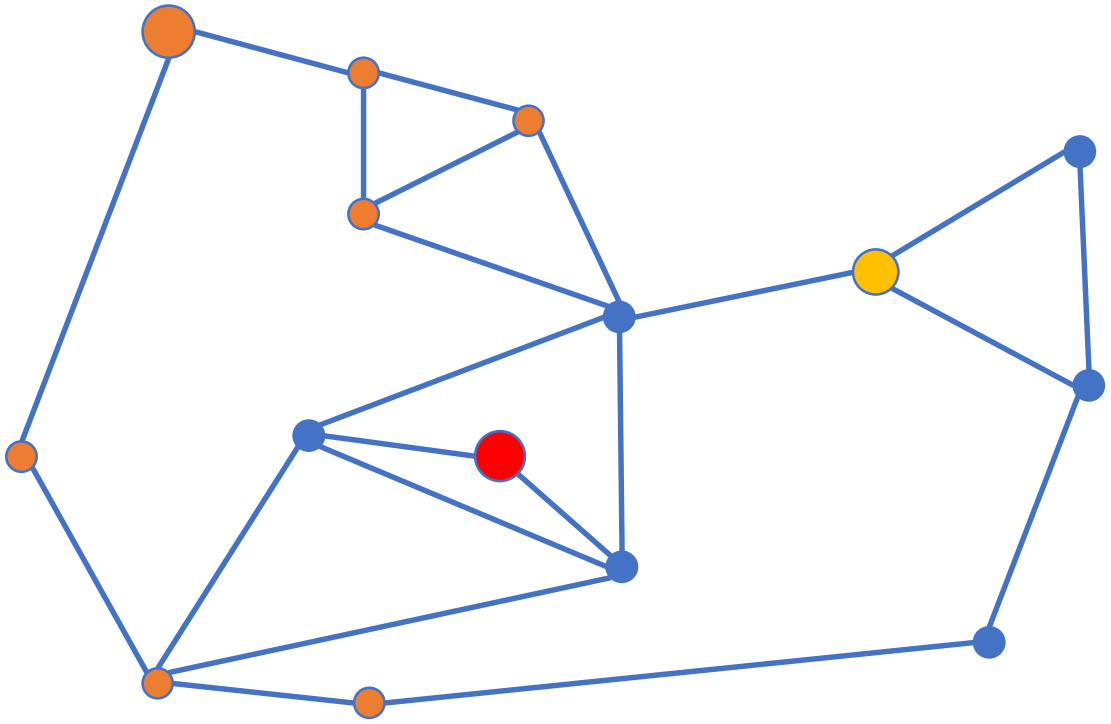
k-centers

- Build k facilities
- Objective: minimize the maximum distance between any vertex to its closest facility



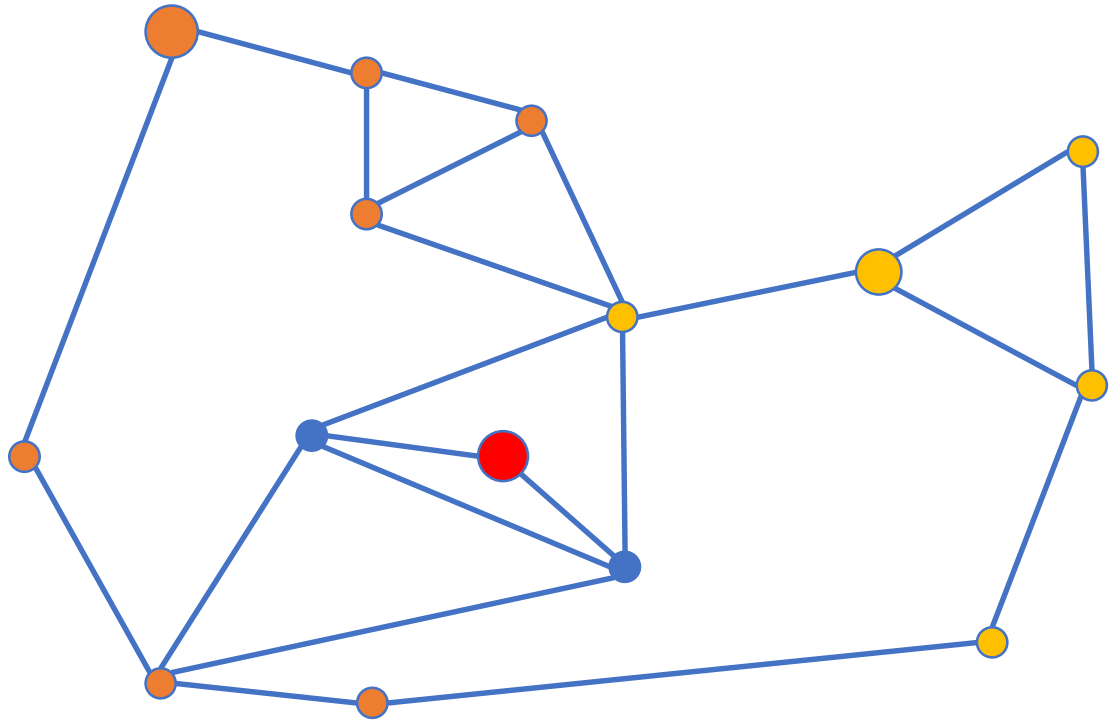
k-centers

- Brown vertices: vertices closest to the brown center. (break tie arbitrarily)



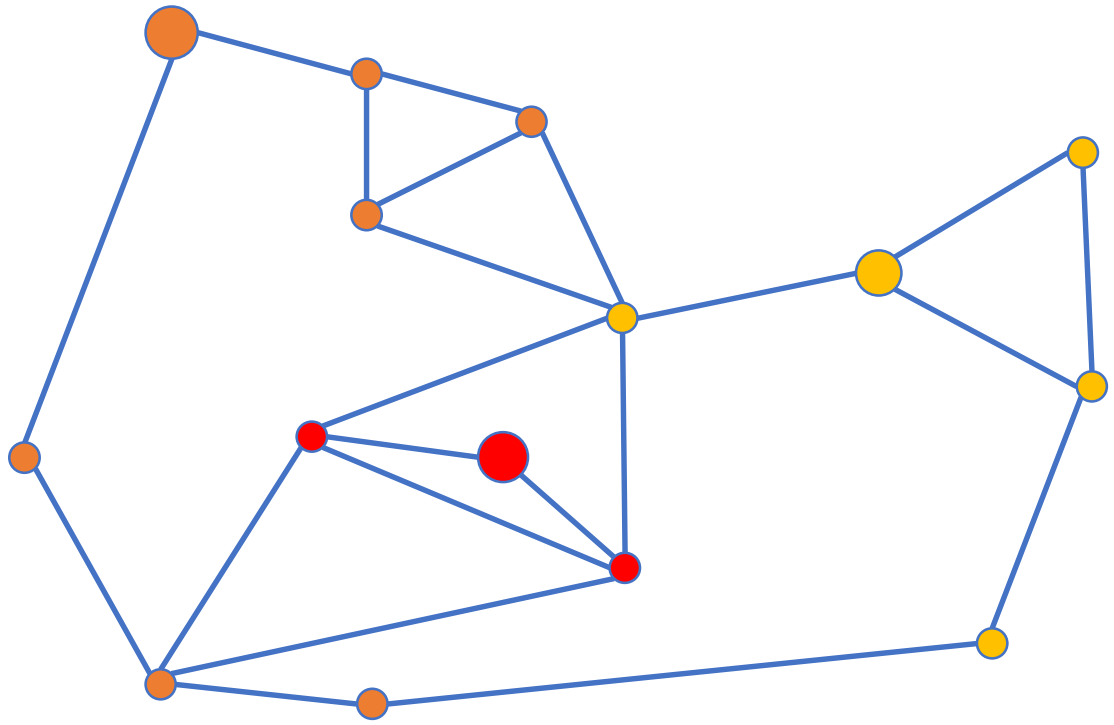
k-centers

- Yellow vertices: vertices closest to the yellow center. (break tie arbitrarily)



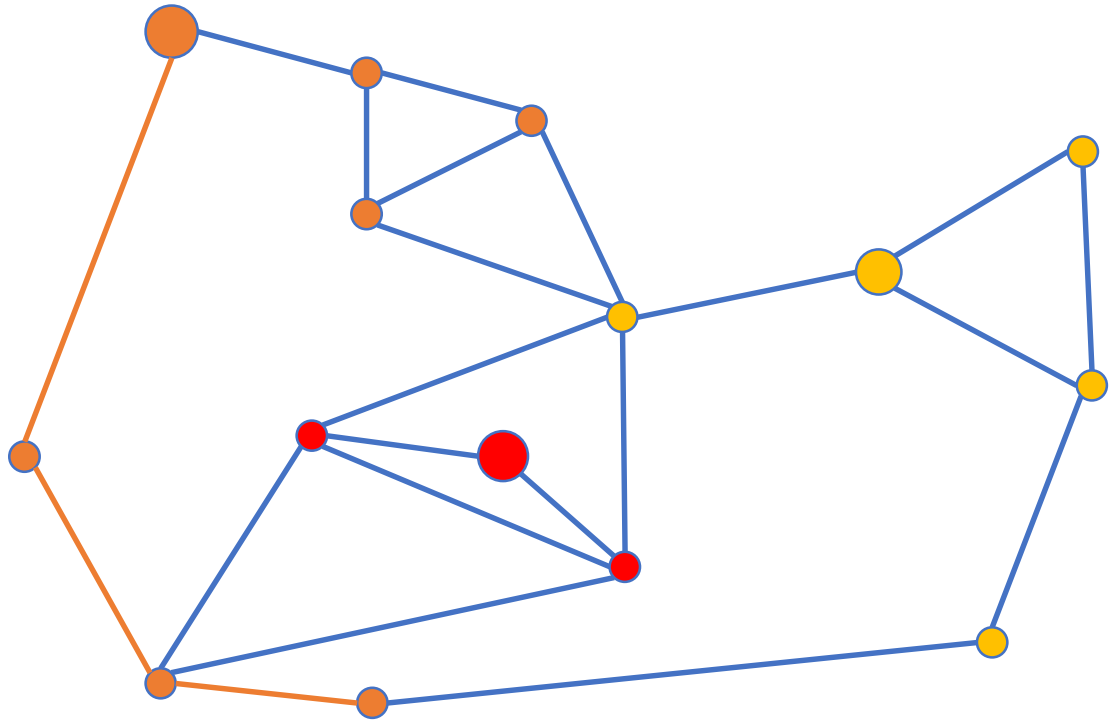
k-centers

- Red vertices: vertices closest to the red center. (break tie arbitrarily)



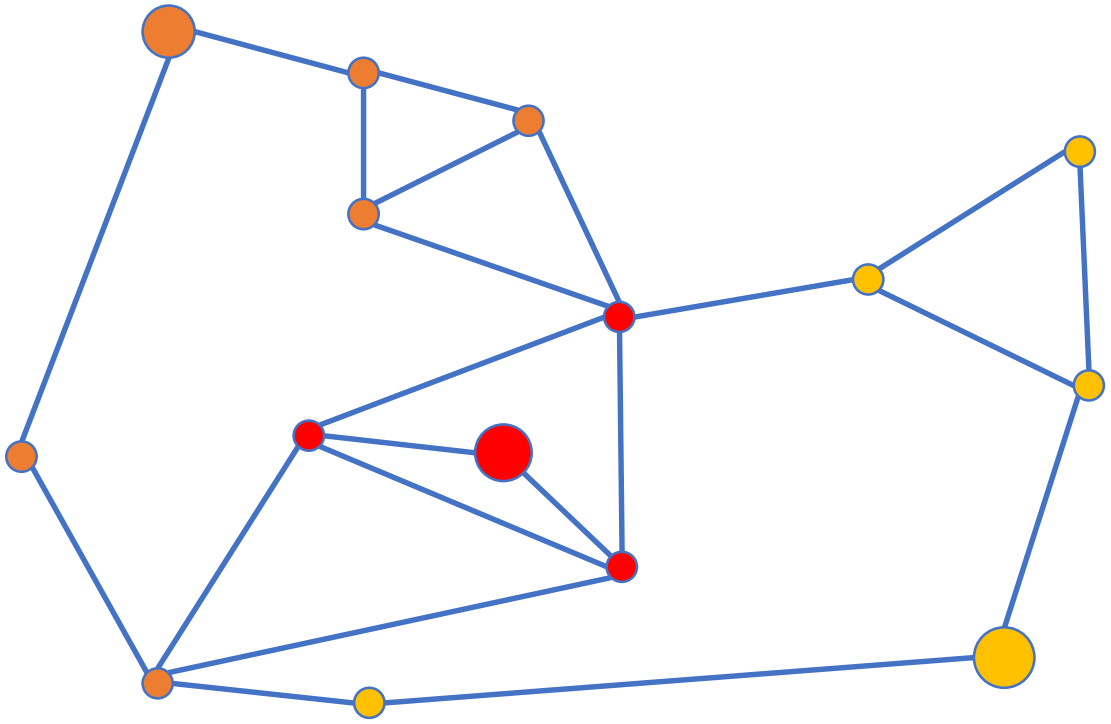
k-centers

- Maximum distance between a vertex to its closest center
- Cost = 3



k-centers

- A better solution with cost 2…



Notation

- $d(u, v)$: the distance from u to v
- $d(A, v) = \min_{s \in A} d(s, v)$: the distance from A to v

k-centers

Input:

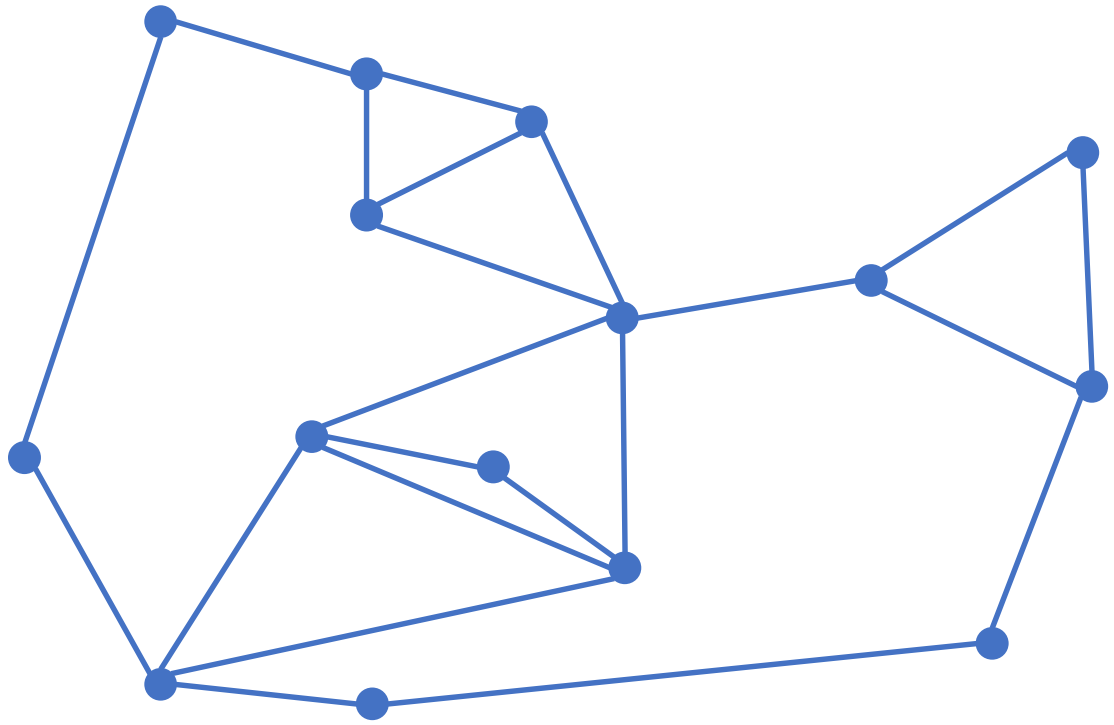
- $G = (V, E)$
- $k \in \mathbb{Z}^+$

Output:

- A set of k centers, $S \subseteq V$ with $|S| = k$, that minimizes
$$f(S) = \max_{v \in V} d(S, v)$$
- $f(S)$: the maximum distance of any vertex $v \in V$ to its closest center $s \in S$

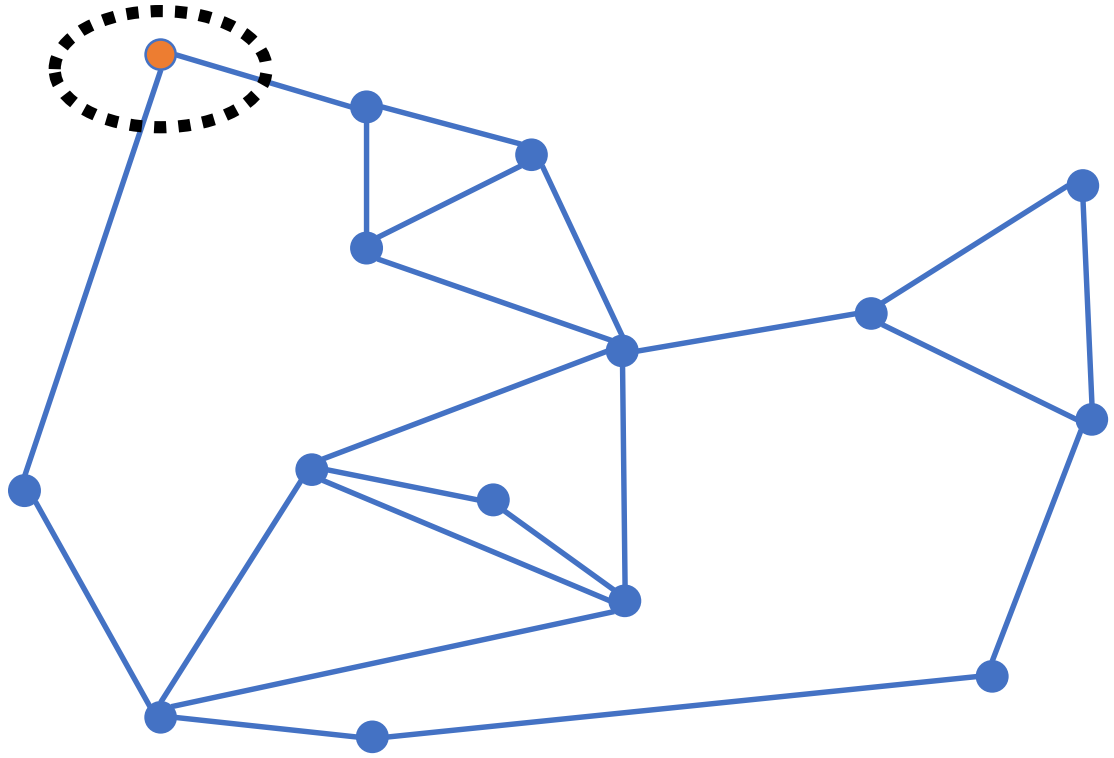
A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Idea of greedy: maximize the reduction in the cost at each iteration.



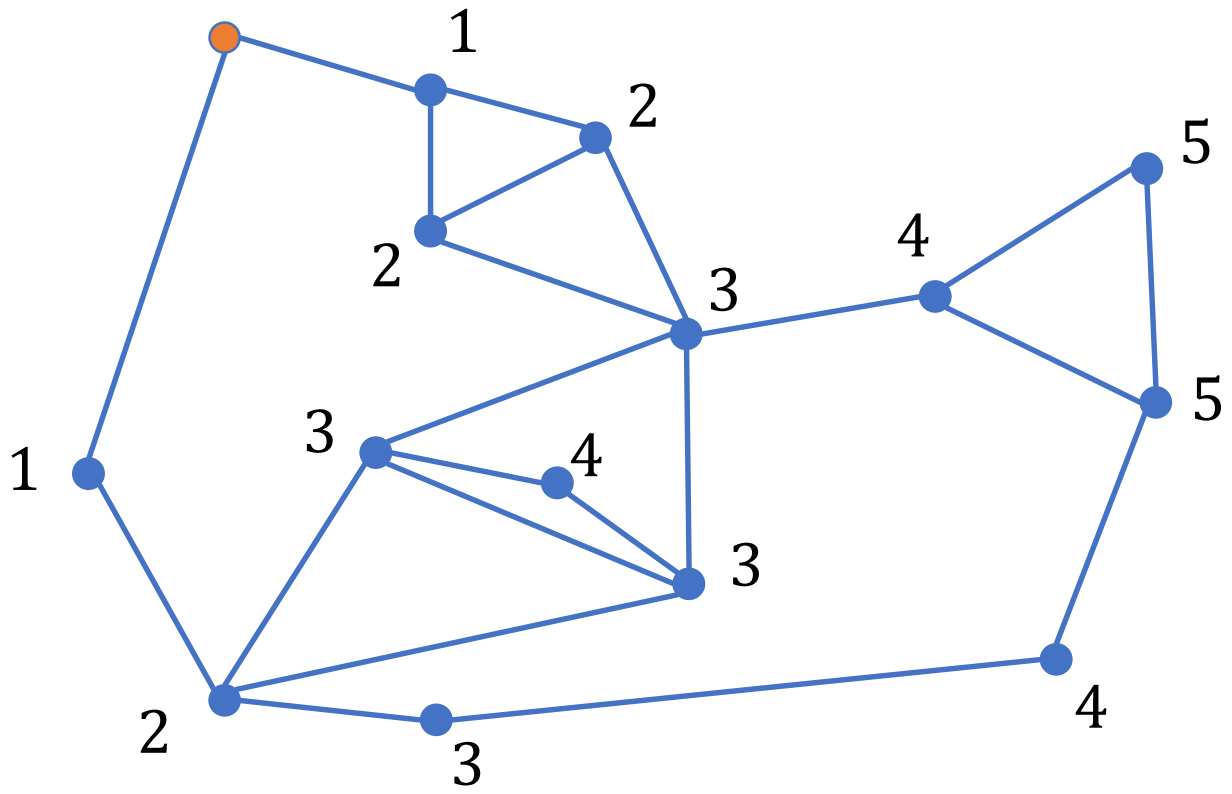
A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Pick the first center arbitrarily…



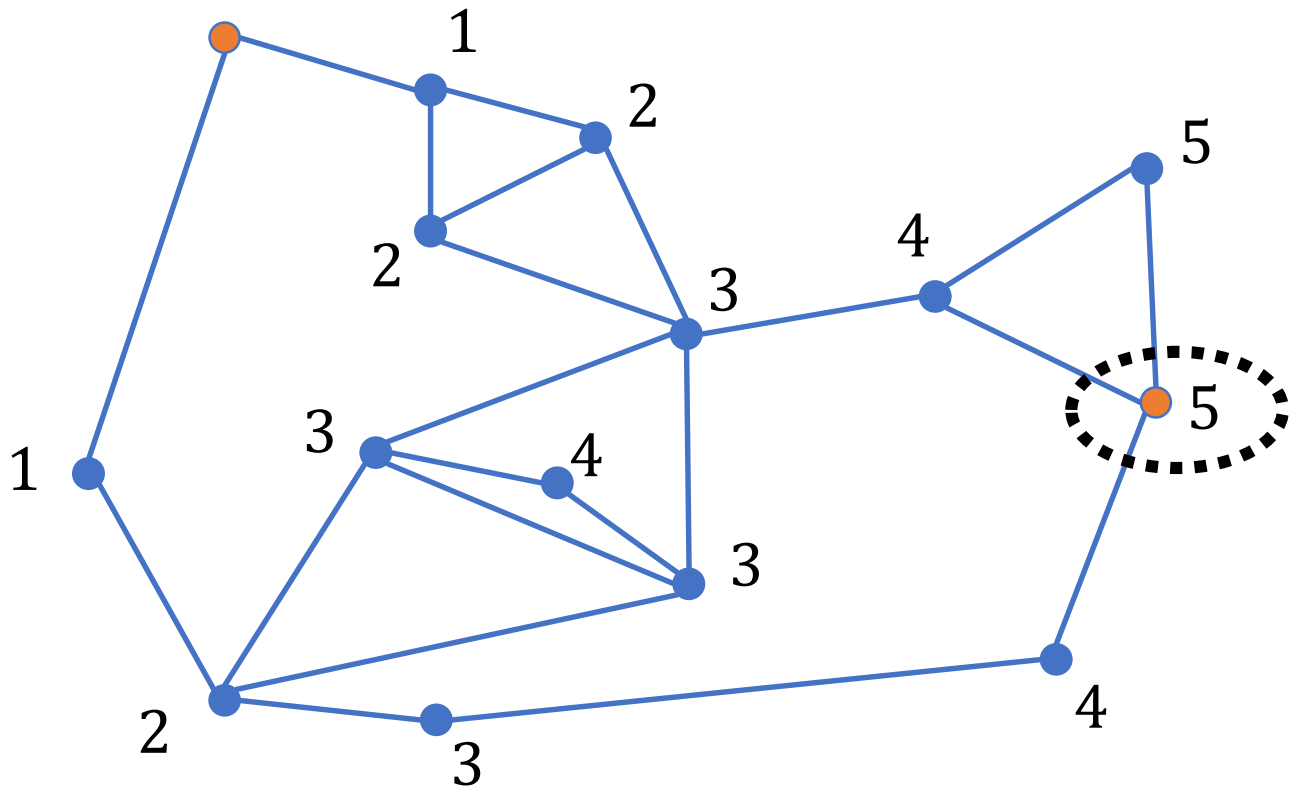
A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Get the distance of all vertices…



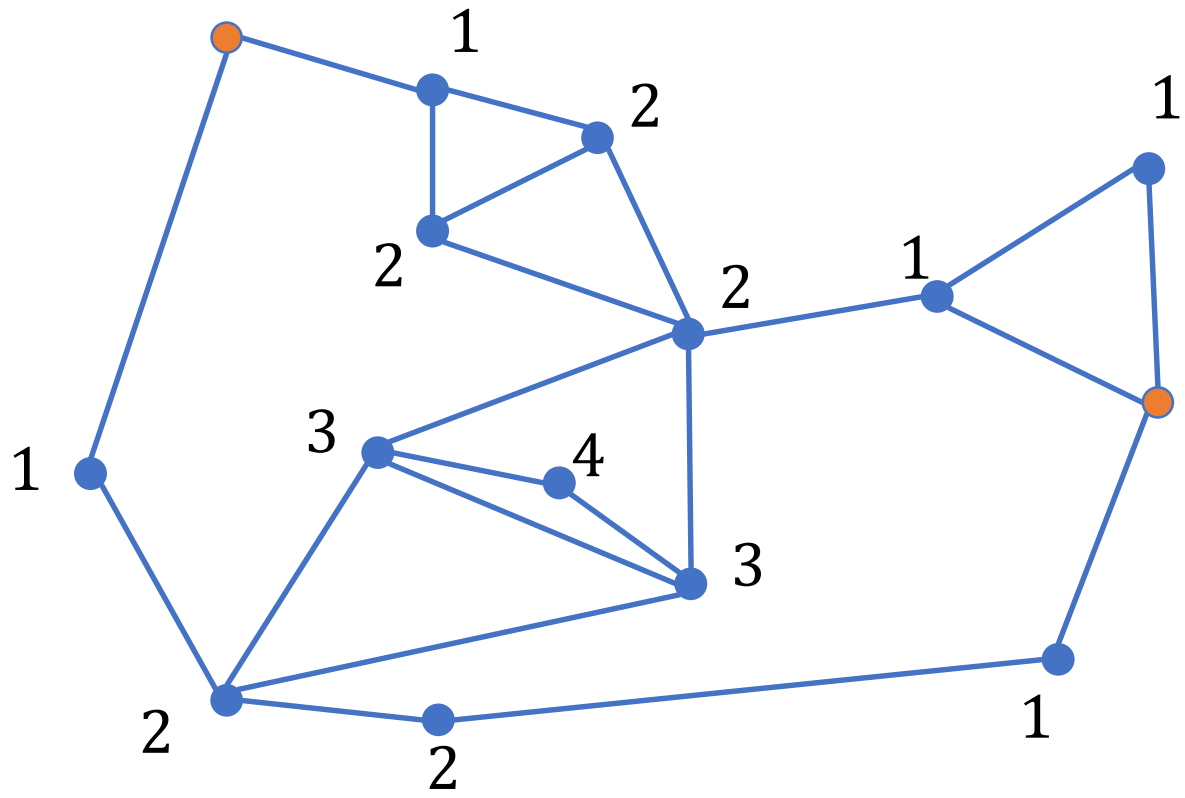
A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Choose the farthest vertex



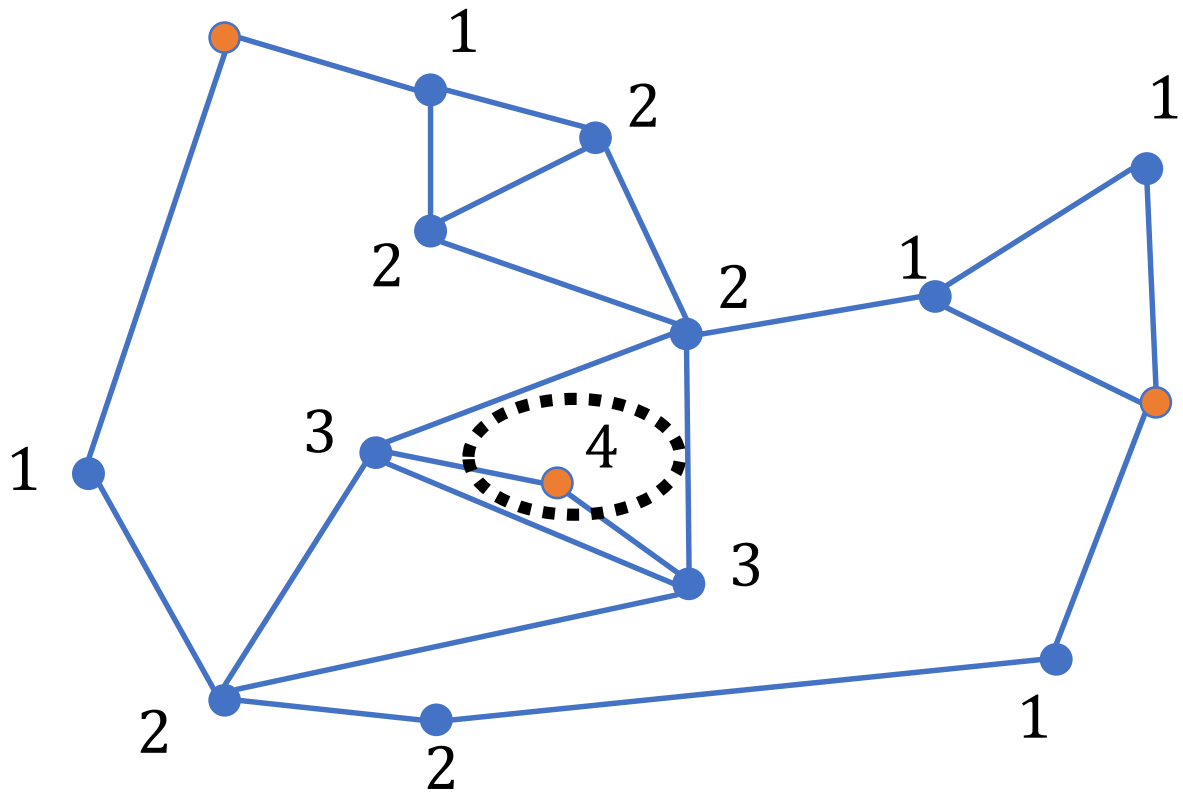
A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Update distances…



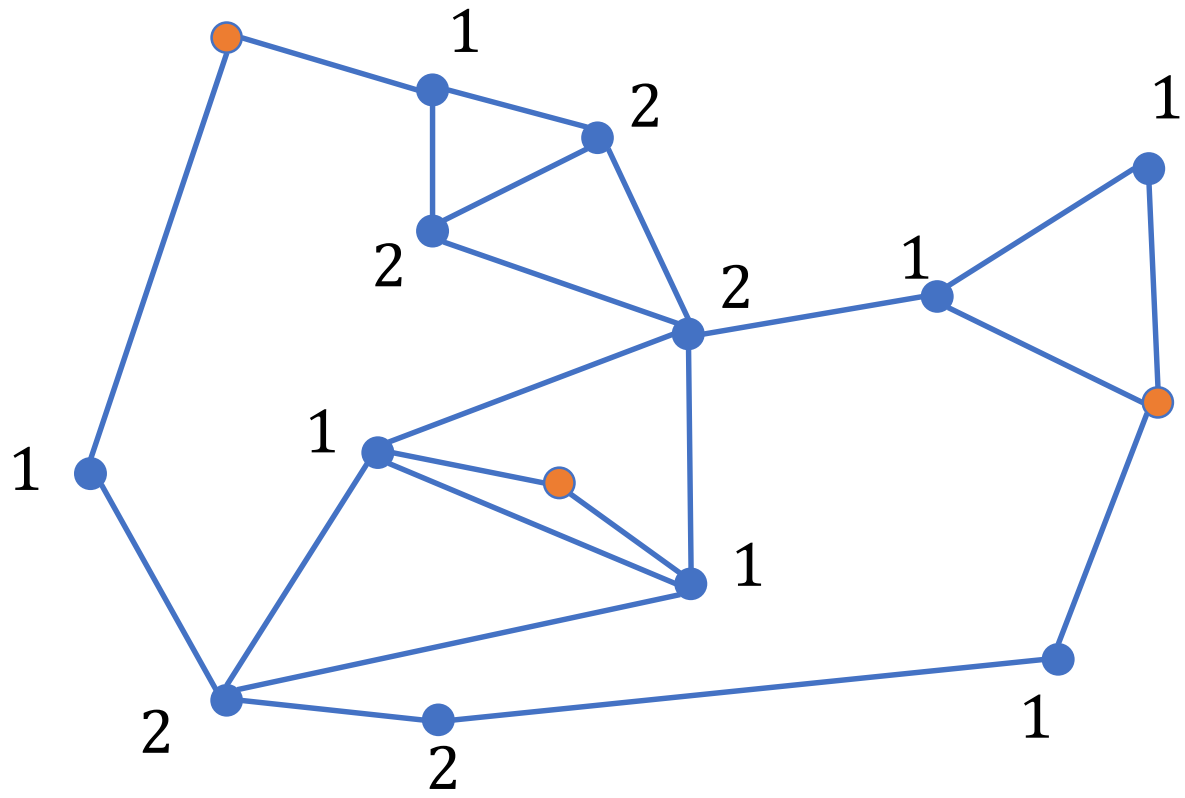
A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Choose the farthest vertex



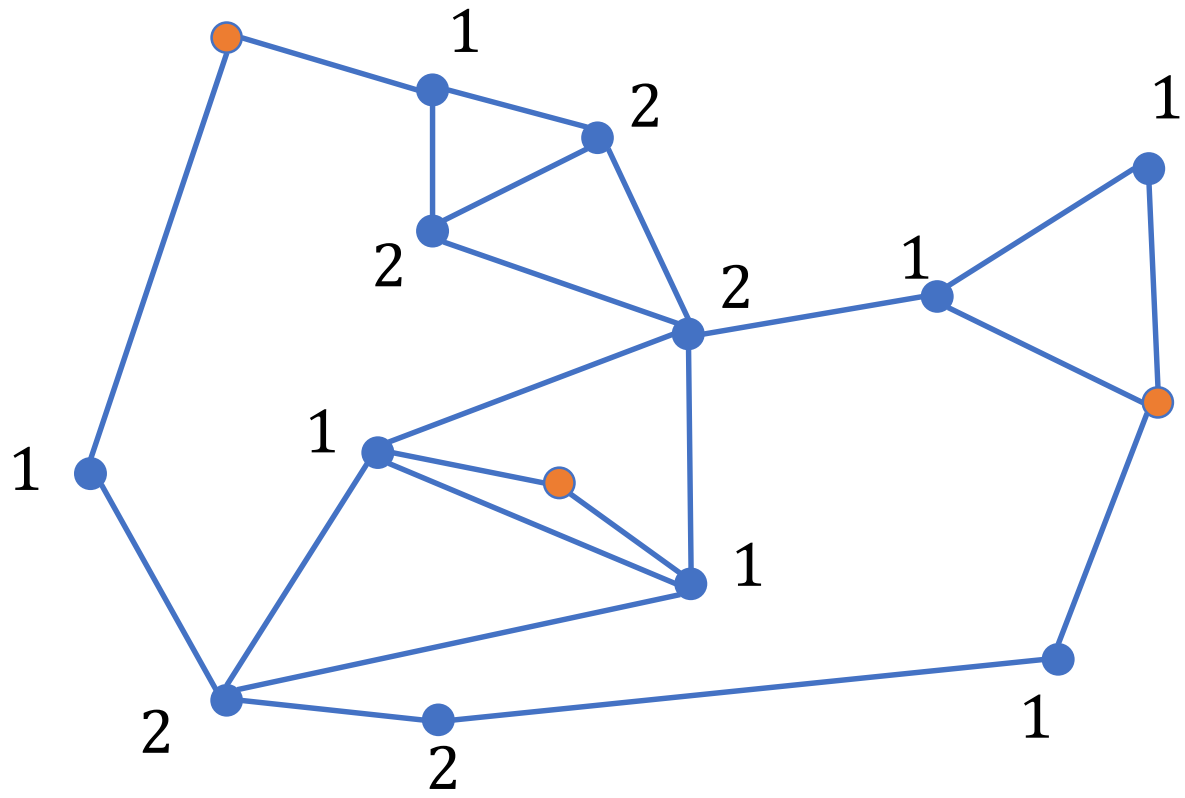
A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Update the distances…



A Natural Greedy Algorithm

- Iteratively pick the center farthest to the existing centers
- Do this for k iterations…

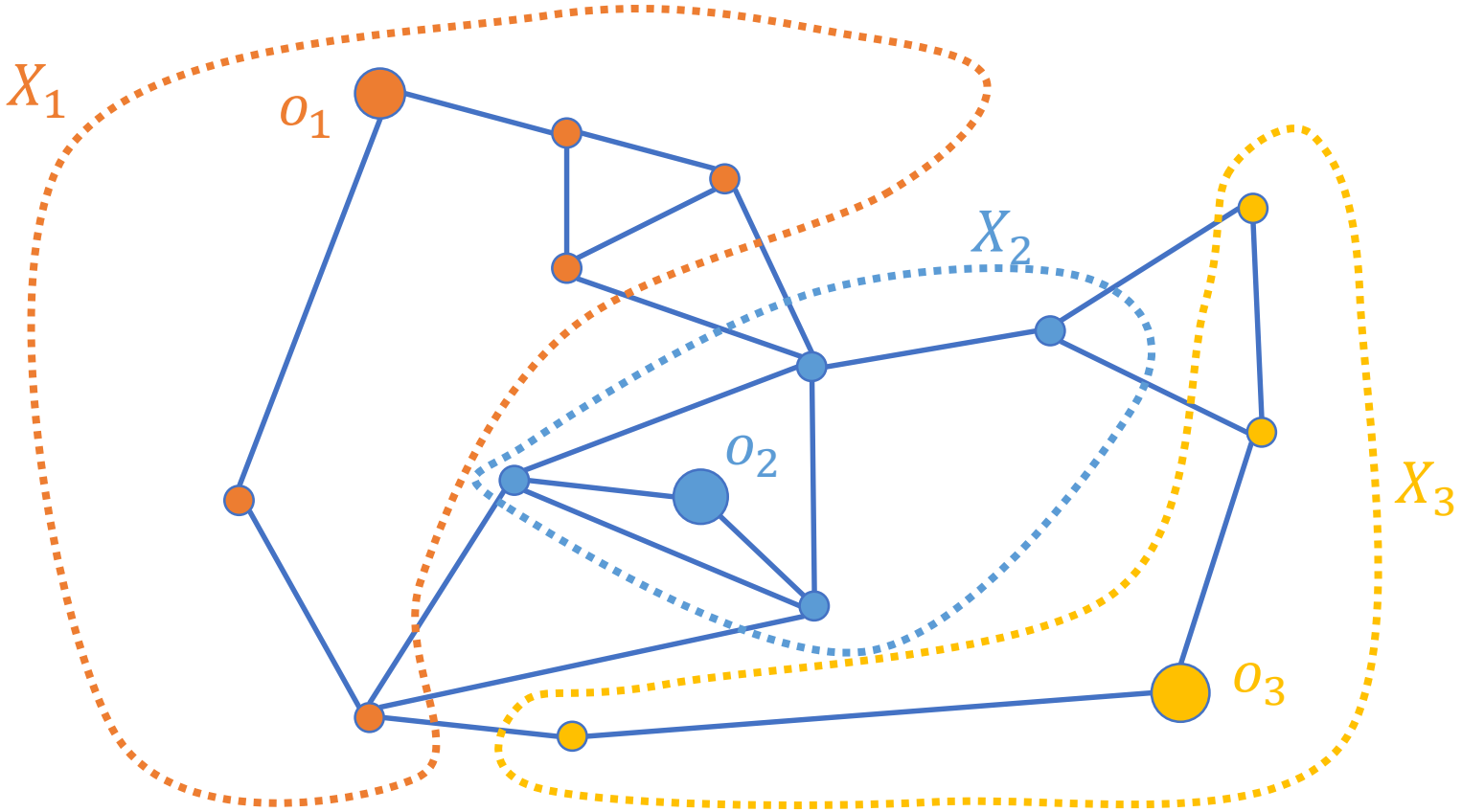


2-Approximation

- **Theorem**. The greedy algorithm is a 2-approximation algorithm.
- Let $O = \{o_1, \dots, o_k\}$ be the optimal solution with cost OPT
- Let $A = \{a_1, \dots, a_k\}$ be the algorithm's output with cost ALG
- The theorem says $\mathrm{ALG} \leq 2 \cdot \mathrm{OPT}$

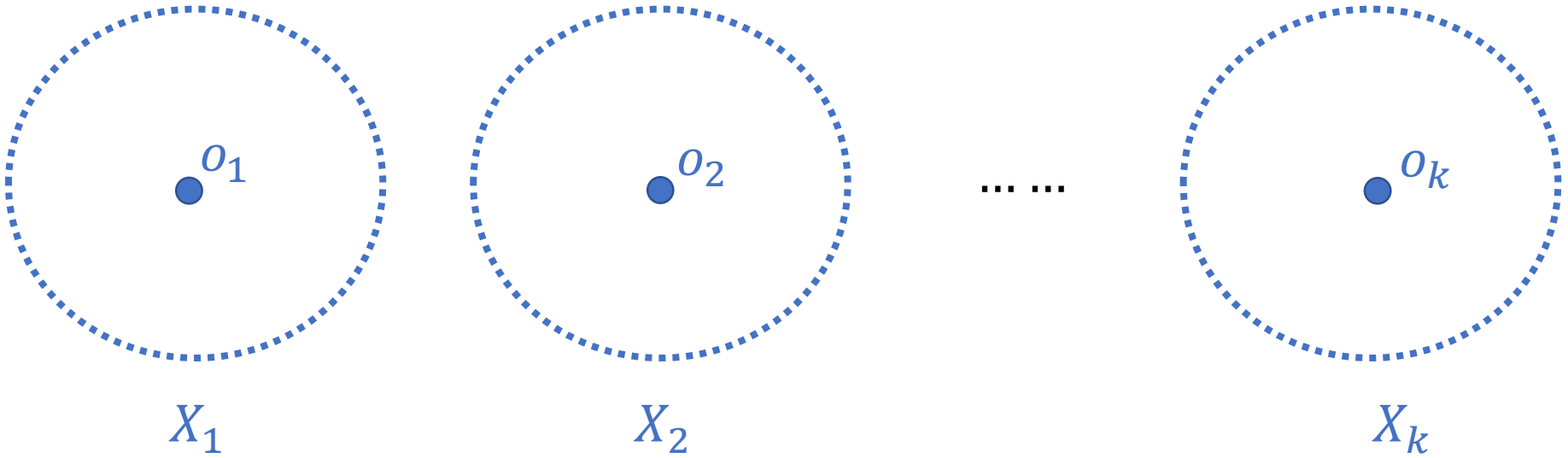
Notation

- For each o_i, let X_i be the set of vertices closest to o_i.



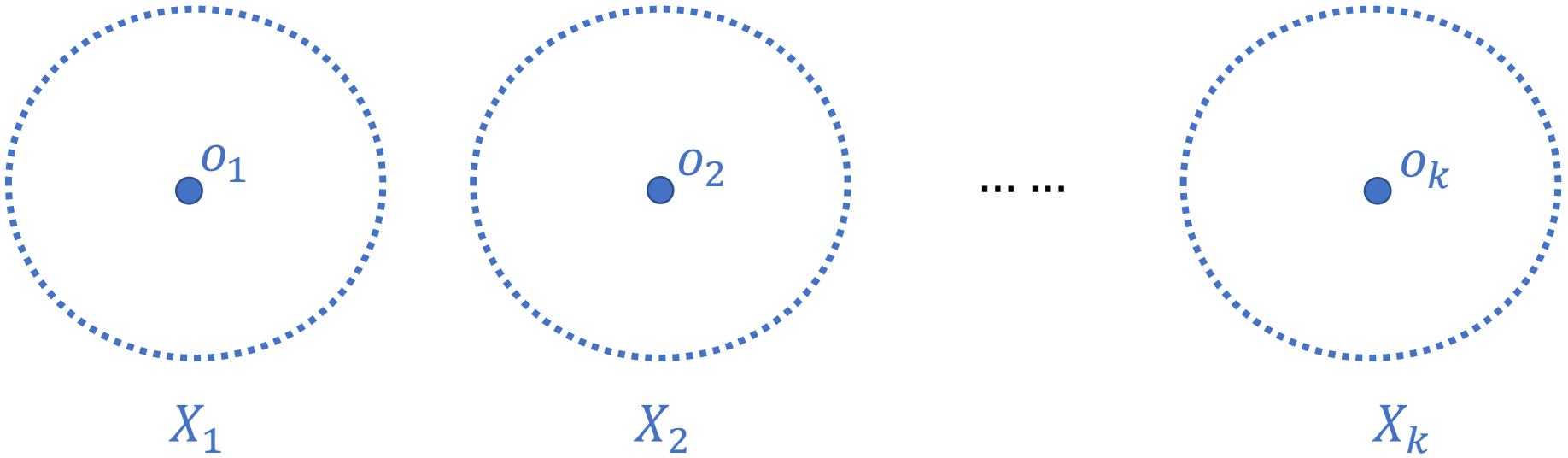
Notation

- For each o_i, let X_i be the set of vertices closest to o_i.
- $\{X_1, \ldots, X_k\}$ forms a partition of V



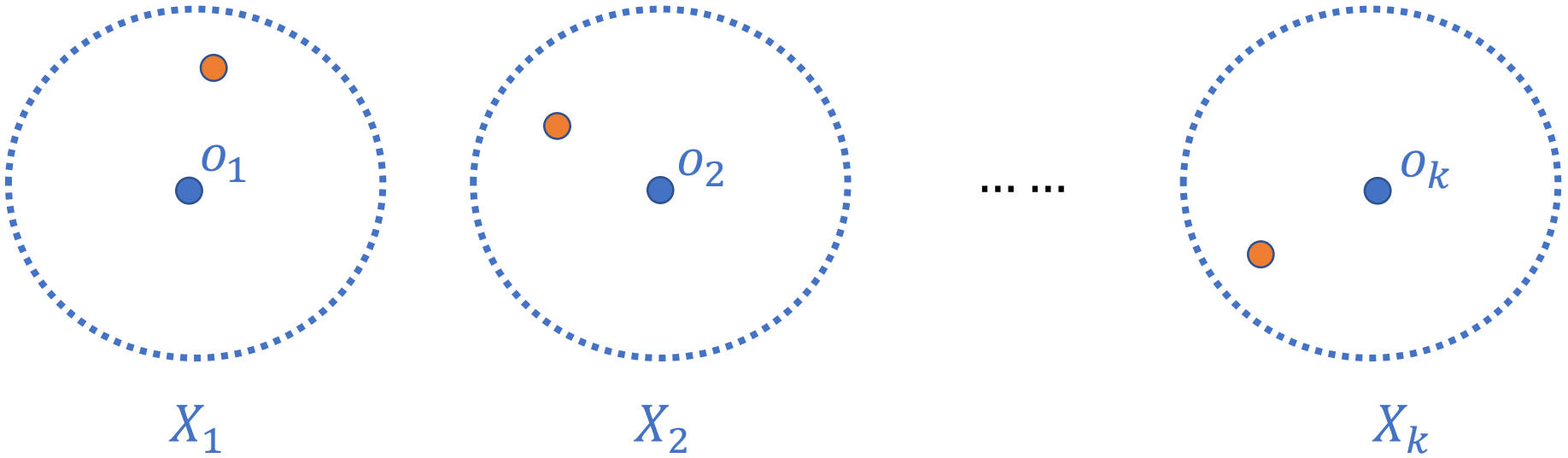
Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

- Suppose $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$



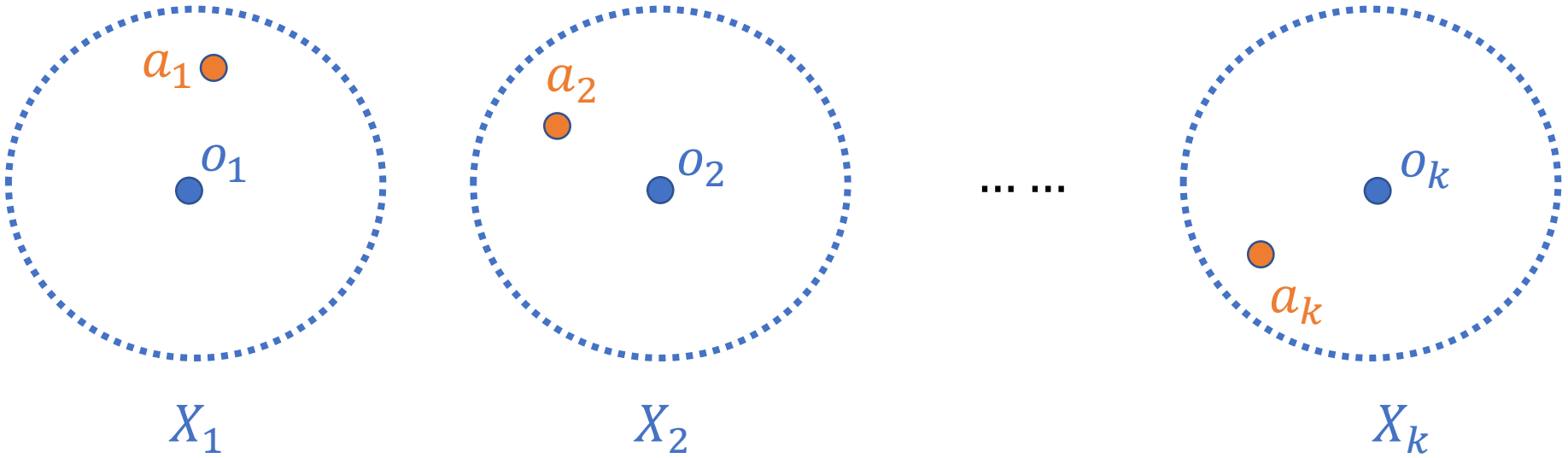
Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \dots, k$

- Suppose $A \cap X_i \neq \emptyset$ for each $i = 1, \dots, k$
- Then each X_i contains a center in $A = \{a_1, \dots, a_k\}$



Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

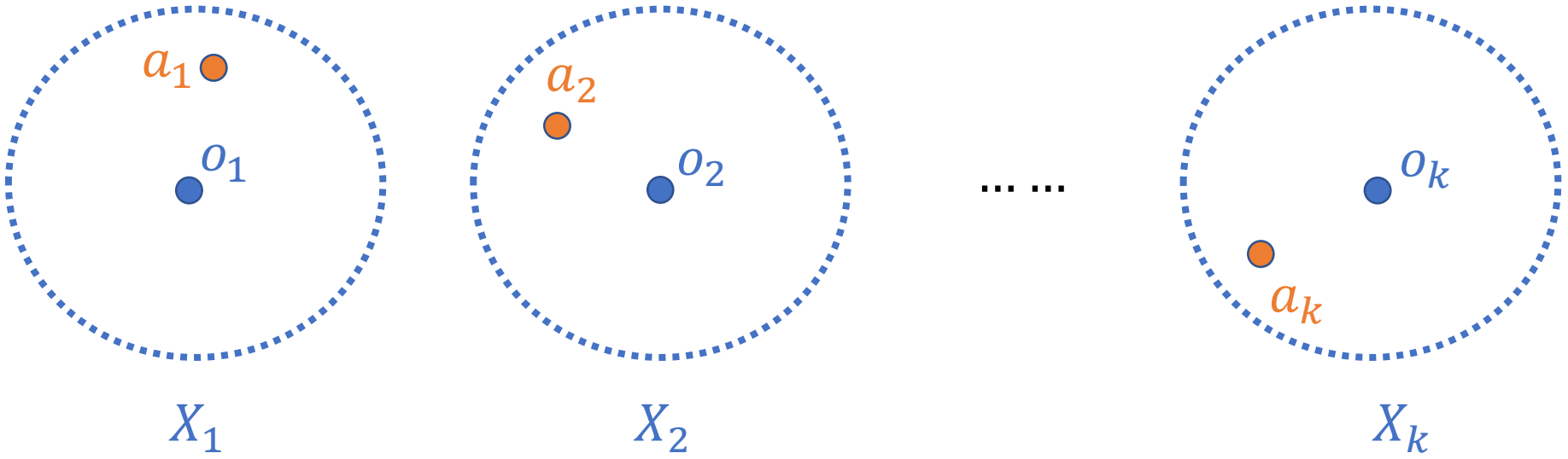
- Suppose $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$
- Then each X_i contains a center in $A = \{a_1, \ldots, a_k\}$
- Assume $a_i \in X_i$ WLOG



Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

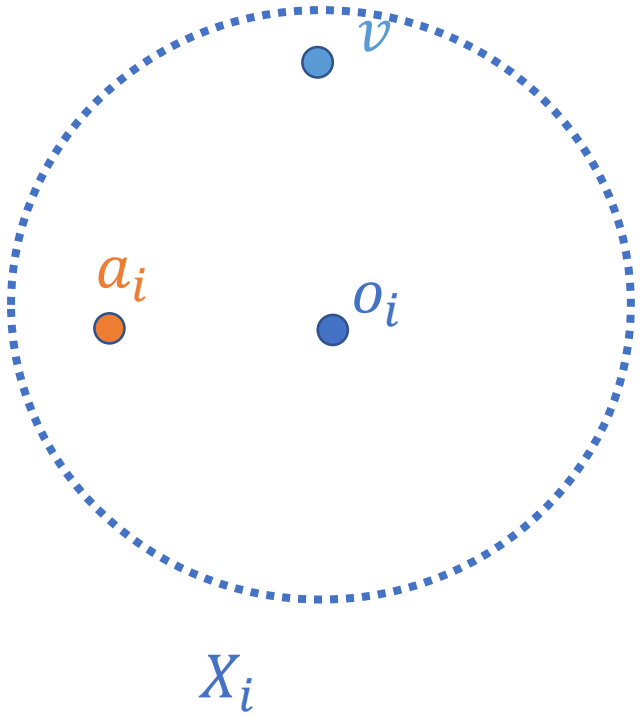
Intuition:

- For each v that is assigned to o_i in optimal solution, assigning it to a_i is "not too bad".



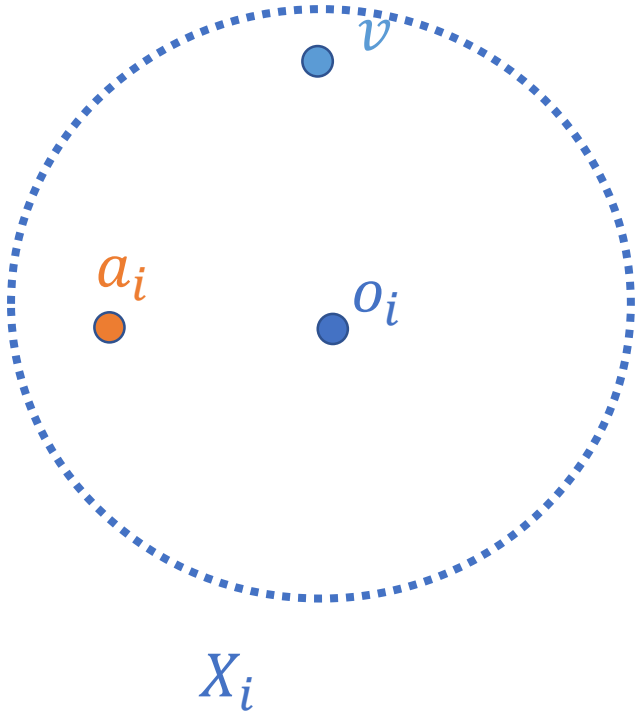
Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

- Consider any $v \in V$ and let X_i be the cluster containing v.



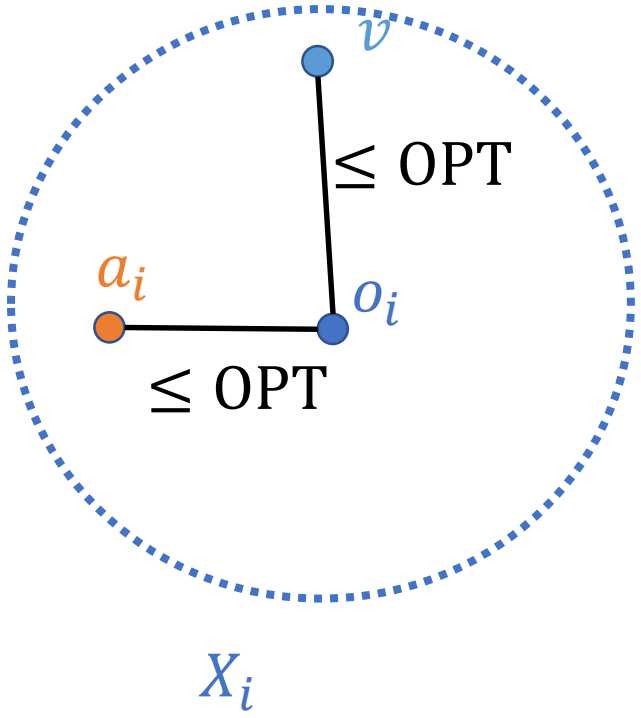
Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

- Consider any $v \in V$ and let X_i be the cluster containing v.
- Recall $\mathrm{OPT} = \max_{v \in V} d(O, v)$
- Each vertex u in X_i is closest to o_i; so $d(o_i, u) \leq \mathrm{OPT}$



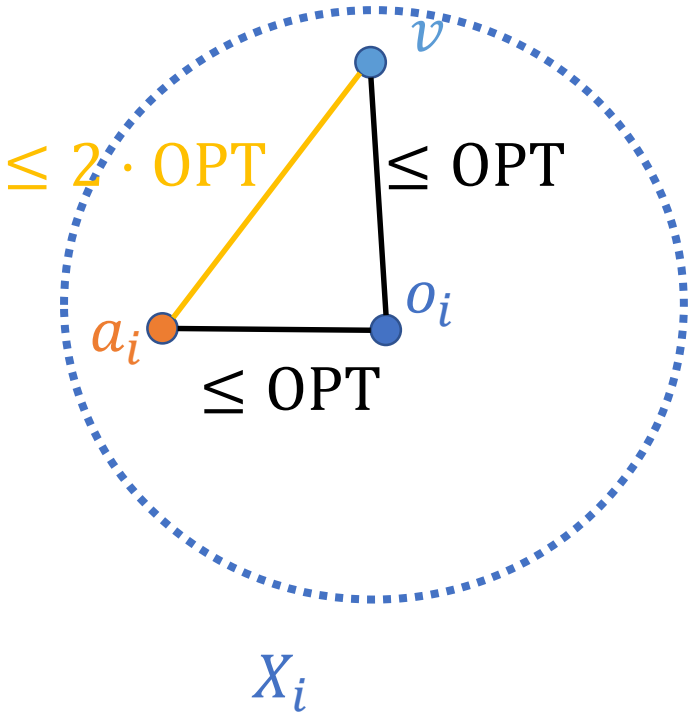
Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

- Consider any $v \in V$ and let X_i be the cluster containing v.
- Recall $\mathrm{OPT} = \max_{v \in V} d(O, v)$
- Each vertex u in X_i is closest to o_i; so $d(o_i, u) \leq \mathrm{OPT}$
- Thus, $d(o_i, v) \leq \mathrm{OPT}$ and $d(o_i, a_i) \leq \mathrm{OPT}$



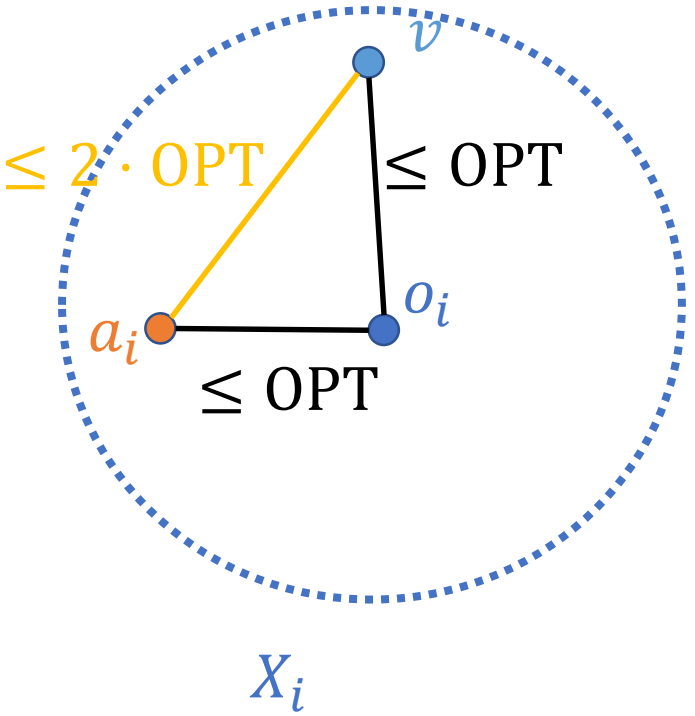
Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

- Consider any $v \in V$ and let X_i be the cluster containing v.
- Recall $\mathrm{OPT} = \max_{v \in V} d(O, v)$
- Each vertex u in X_i is closest to o_i; so $d(o_i, u) \leq \mathrm{OPT}$
- Thus, $d(o_i, v) \leq \mathrm{OPT}$ and $d(o_i, a_i) \leq \mathrm{OPT}$
- By triangle inequality, $d(a_i, v) \leq 2 \cdot \mathrm{OPT}$



Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

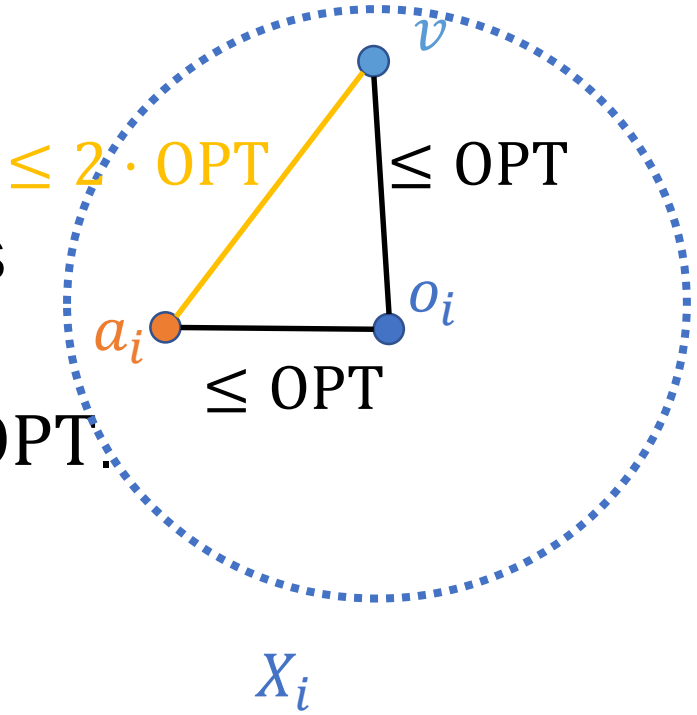
- Consider any $v \in V$ and let X_i be the cluster containing v.
- Recall $\mathrm{OPT} = \max_{v \in V} d(O, v)$
- Each vertex u in X_i is closest to o_i; so $d(o_i, u) \leq \mathrm{OPT}$
- Thus, $d(o_i, v) \leq \mathrm{OPT}$ and $d(o_i, a_i) \leq \mathrm{OPT}$
- By triangle inequality, $d(a_i, v) \leq 2 \cdot \mathrm{OPT}$
- Distance of v to its closest center in A satisfies
$$d(A, v) \leq d(a_i, v) \leq 2 \cdot \mathrm{OPT}$$



v is closest to a_i: $d(A, v) = d(a_i, v)$
Otherwise: $d(A, v) \leq d(a_i, v)$

Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$

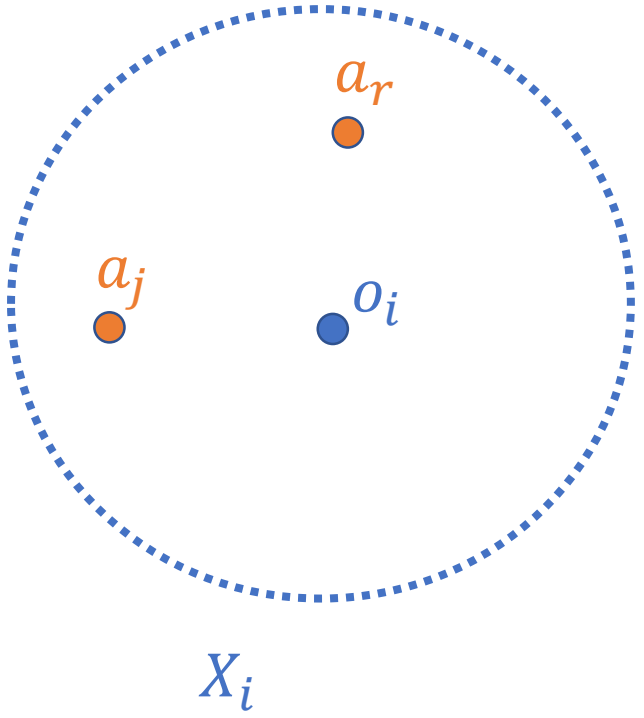
- Consider any $v \in V$ and let X_i be the cluster containing v.
- Recall $\mathrm{OPT} = \max_{v \in V} d(O, v)$
- Each vertex u in X_i is closest to o_i; so $d(o_i, u) \leq \mathrm{OPT}$
- Thus, $d(o_i, v) \leq \mathrm{OPT}$ and $d(o_i, a_i) \leq \mathrm{OPT}$
- By triangle inequality, $d(a_i, v) \leq 2 \cdot \mathrm{OPT}$
- Distance of v to its closest center in A satisfies
$$d(A, v) \leq d(a_i, v) \leq 2 \cdot \mathrm{OPT}$$
- Since v is arbitrary, $\mathrm{ALG} = \max_{v \in V} d(A, v) \leq 2 \cdot \mathrm{OPT}$.



Case 2: $A \cap X_i = \emptyset$ for some i

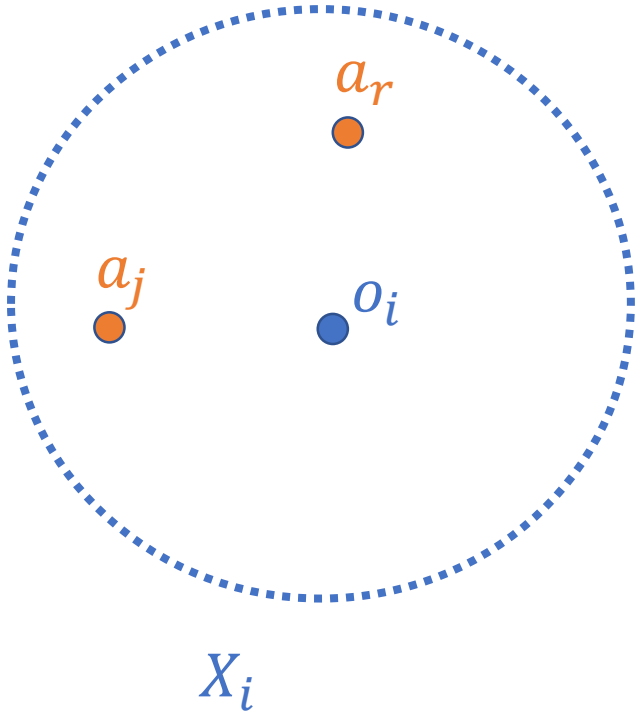
Case 2: $A \cap X_i = \emptyset$ for some i

- In this case, some X_i contains two centers in A.



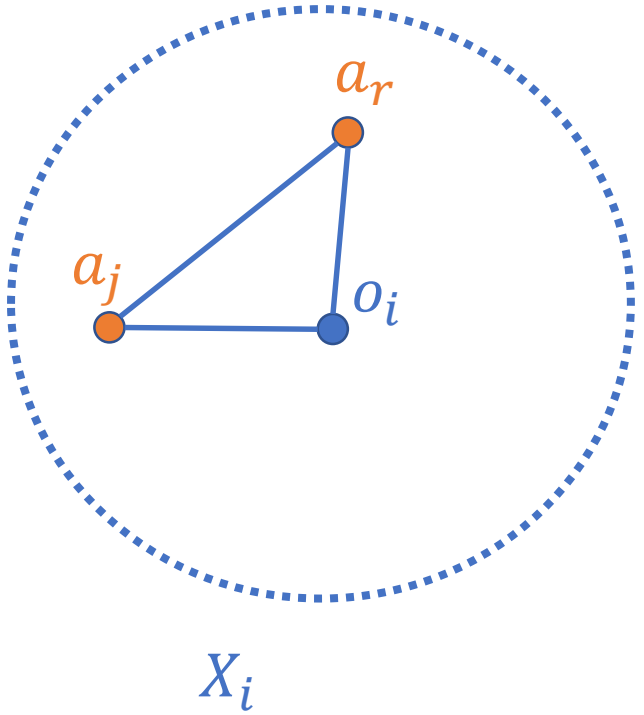
Case 2: $A \cap X_i = \emptyset$ for some i

- In this case, some X_i contains two centers in A.
- Suppose $a_j, a_r \in X_i$. Let a_r be the second center chosen in X_i.
- By our greedy choice, $\mathrm{ALG} = d(A, a_r) \leq d(a_j, a_r)$ before a_r was chosen.
- Thus, $\mathrm{ALG} \leq d(a_j, a_r)$ when algorithm ends.



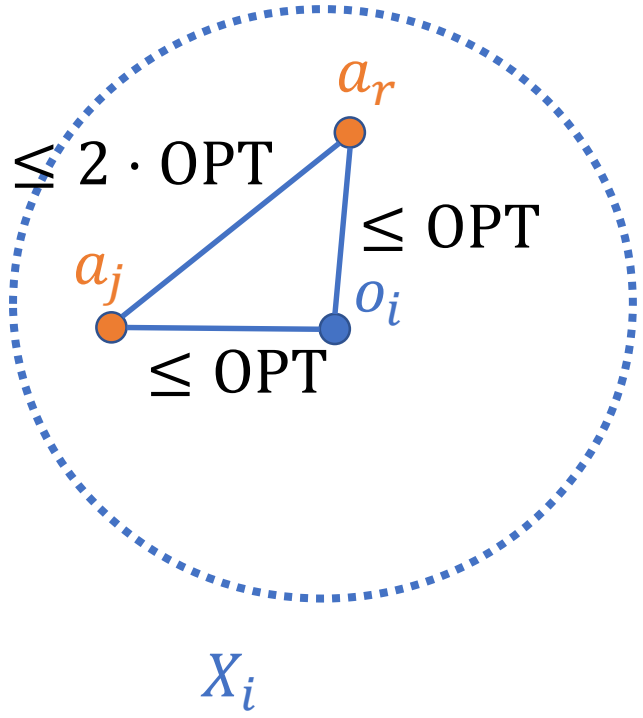
Case 2: $A \cap X_i = \emptyset$ for some i

- In this case, some X_i contains two centers in A.
- Suppose $a_j, a_r \in X_i$. Let a_r be the second center chosen in X_i.
- By our greedy choice, $\mathrm{ALG} = d(A, a_r) \leq d(a_j, a_r)$ before a_r was chosen.
- Thus, $\mathrm{ALG} \leq d(a_j, a_r)$ when algorithm ends.
- Additionally, $d(a_j, a_r) \leq d(a_j, o_i) + d(a_r, o_i)$



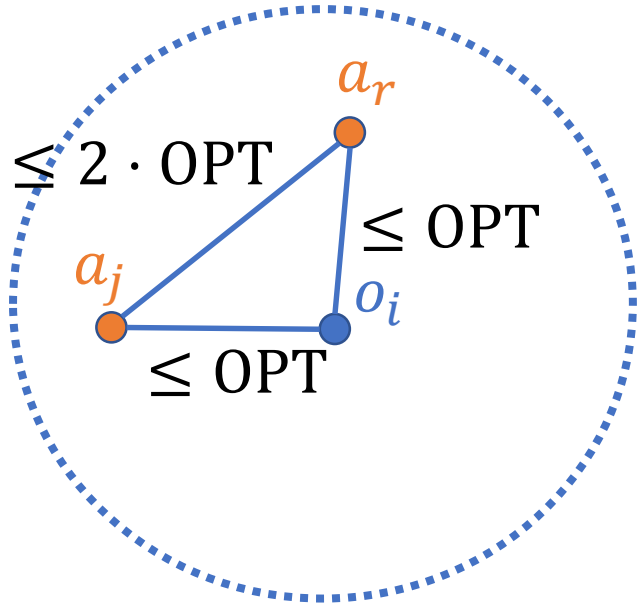
Case 2: $A \cap X_i = \emptyset$ for some i

- In this case, some X_i contains two centers in A.
- Suppose $a_j, a_r \in X_i$. Let a_r be the second center chosen in X_i.
- By our greedy choice, $\mathrm{ALG} = d(A, a_r) \leq d(a_j, a_r)$ before a_r was chosen.
- Thus, $\mathrm{ALG} \leq d(a_j, a_r)$ when algorithm ends.
- Additionally, $d(a_j, a_r) \leq d(a_j, o_i) + d(a_r, o_i)$
- $\leq \mathrm{OPT} + \mathrm{OPT}$



Case 2: $A \cap X_i = \emptyset$ for some i

- In this case, some X_i contains two centers in A.
- Suppose $a_j, a_r \in X_i$. Let a_r be the second center chosen in X_i.
- By our greedy choice, $\mathrm{ALG} = d(A, a_r) \leq d(a_j, a_r)$ before a_r was chosen.
- Thus, $\mathrm{ALG} \leq d(a_j, a_r)$ when algorithm ends.
- Additionally, $d(a_j, a_r) \leq d(a_j, o_i) + d(a_r, o_i)$
- $\leq \mathrm{OPT} + \mathrm{OPT}$
- Putting together: $\mathrm{ALG} \leq d(a_j, a_r) \leq 2 \cdot \mathrm{OPT}$



We are done!

- **Theorem**. The greedy algorithm is a 2-approximation algorithm.

We have proved $\mathrm{ALG} \leq 2 \cdot \mathrm{OPT}$ for both cases below:

- Case 1: $A \cap X_i \neq \emptyset$ for each $i = 1, \ldots, k$
- Case 2: $A \cap X_i = \emptyset$ for some i

This algorithm runs in polynomial time (not proved in this course).

Same Questions as Before

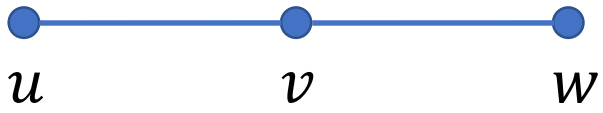
- **Theorem**. The greedy algorithm is a 2-approximation algorithm.

Question: Can we do better?

- Same algorithm with a more careful analysis?
- Another more clever algorithm?

Same algorithm with a better analysis?

- NO!
- A tight example:
  - $V = \{u, v, w\}$
  - $d(u, v) = d(v, w) = 1$, $d(u, w) = 2$
  - $k = 1$
- The first center is chosen arbitrarily, so u may be chosen!
- $\mathrm{ALG} = 2$ and $\mathrm{OPT} = 1$



Same algorithm with a better analysis?

- Well, this looks like cheating!
- How about this? We modify the algorithm such that the first center is chosen that minimizing the distance to the farthest vertex.

Same algorithm with a better analysis?

- Still doesn't work…
- Tight example: V form a "line" and $k = 2$

Same algorithm with a better analysis?

- Still doesn't work…
- Tight example: V form a "line" and $k = 2$
- Greedy: $\mathbf{ALG} = \frac{|V|}{2}$

Same algorithm with a better analysis?

- Still doesn't work…
- Tight example: V form a "line" and $k = 2$
- Greedy: $\mathrm{ALG} = \frac{|V|}{2}$

- Optimal solution: choose the one-fourth and three-fourth points
- $\mathrm{OPT} = \frac{|V|}{4}$

A more clever algorithm?

- Unlikely…
- A polynomial time $(2-\epsilon)$-approximation algorithm any $\epsilon > 0$ implies P = NP.

两类贪心算法

1. Max-Cut: 从一个任意解出发，然后“贪心地”调整
   - 该类算法又叫做local search
2. k-centers: “贪心地”去构造一个解

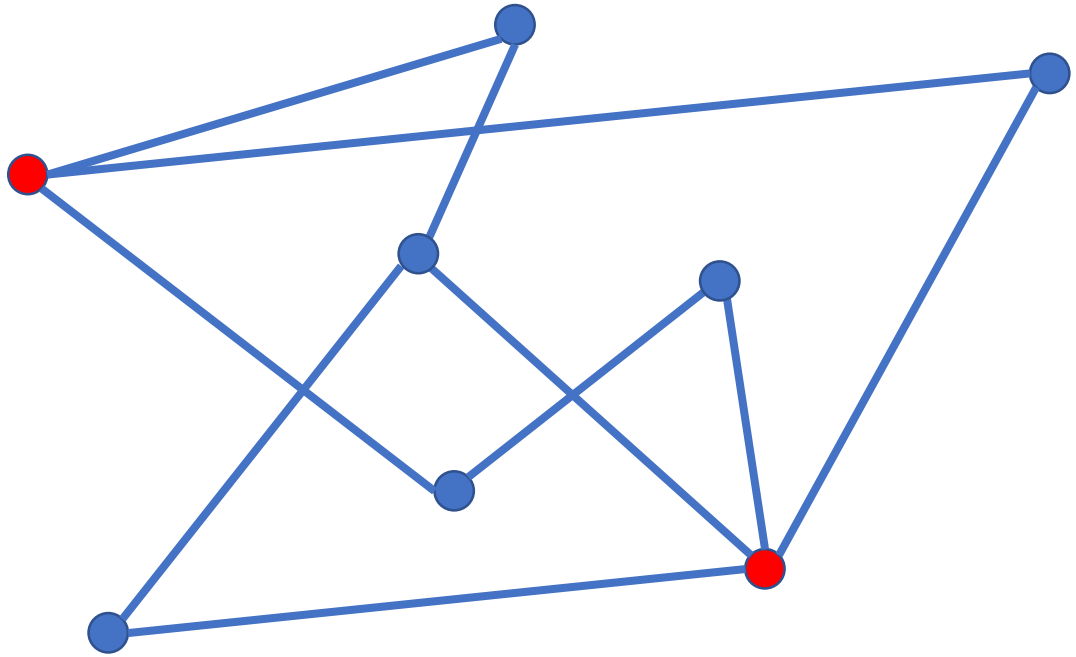
k-centers问题

$(2-\varepsilon)$-不可近似性

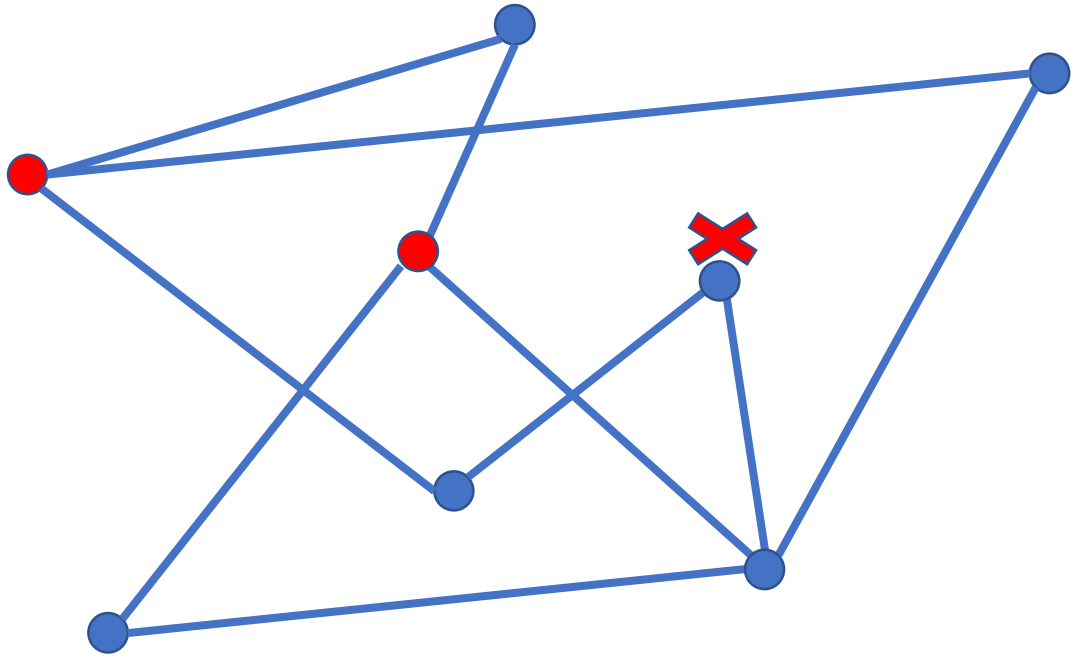
Let’s first see another NP-complete problem

Dominating Set

- Given an undirected graph $G = (V, E)$, a dominating set is a subset of vertices S such that, for any $v \in V \setminus S$, there is a vertex $u \in S$ that is adjacent to v.



a dominating set



not a dominating set

Dominating Set Problem

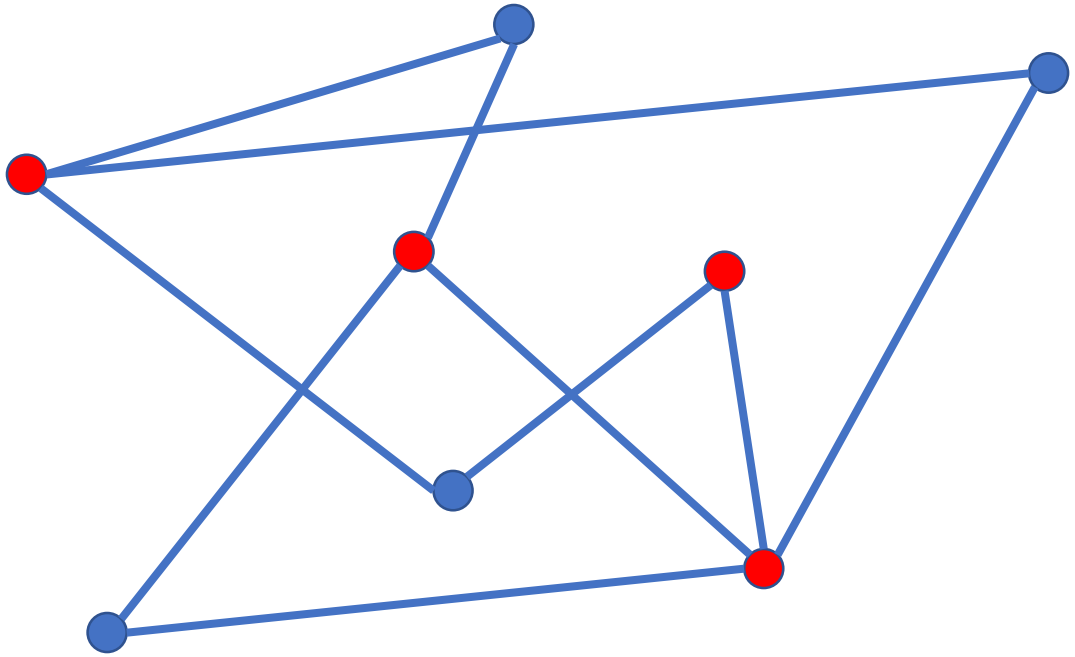
- [DominatingSet] Given an undirected graph $G = (V, E)$ and an integer $k \in \mathbb{Z}^+$, decide if G contains a dominating set with size k.
- **Theorem**. DominatingSet is NP-complete.

Reduction from VertexCover

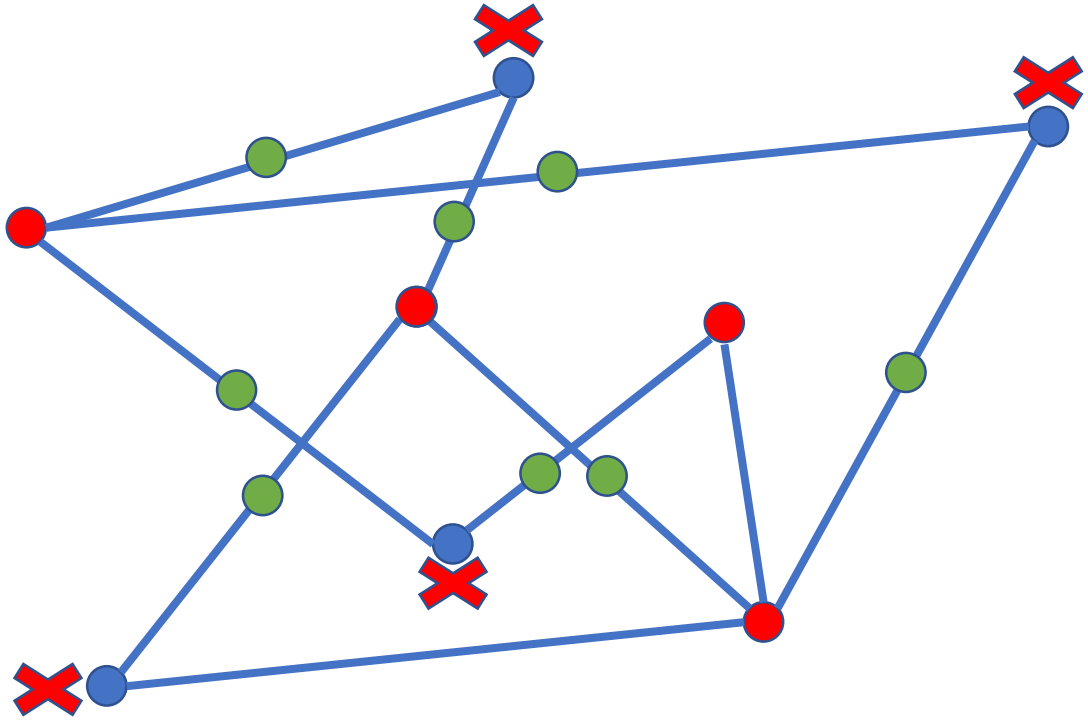
- A dominating set is similar to a vertex cover:
  - Vertex cover: S covers edges
  - Dominating set: S covers vertices
- An idea for reduction:
  - Introduce an intermediate vertex for each edge
  - cover the edge $\Longrightarrow$ cover the intermediate vertex

Does it work?

Does it work? NO!



a vertex cover

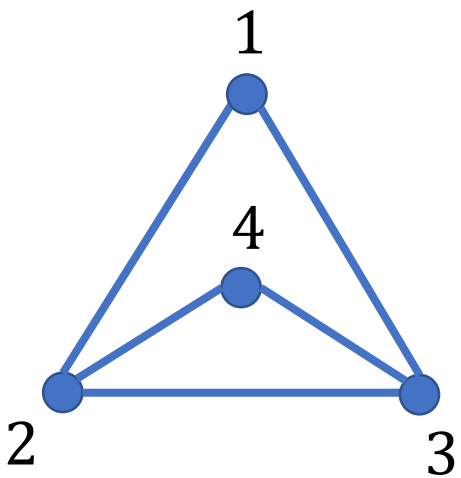


not a dominating set

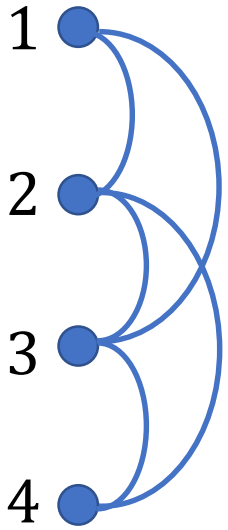
- New vertices are covered, but original vertices may not be covered!
- Can we fix it?

Fix the Reduction

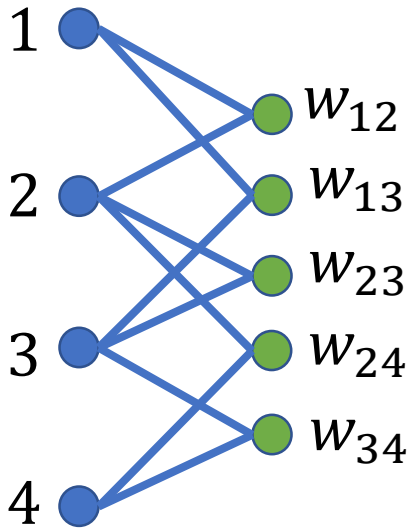
- All we have to do: make the original vertices a clique!
- Now, selecting a single vertex in the original vertex set covers all the original vertices.



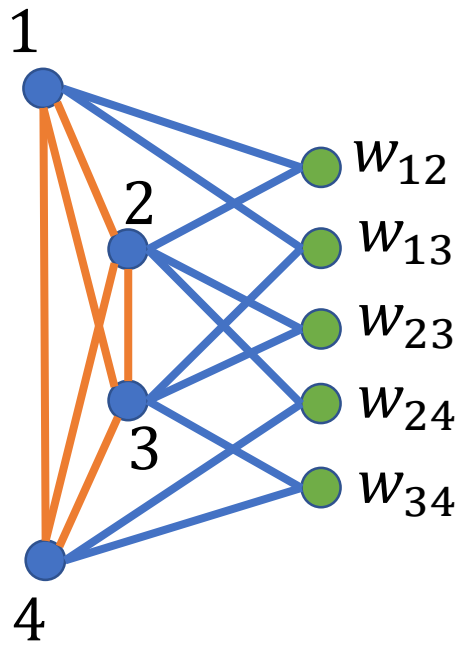
original graph



redraw it



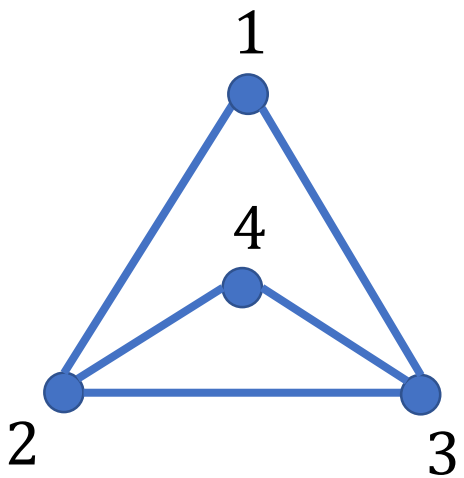
our idea that doesn't work



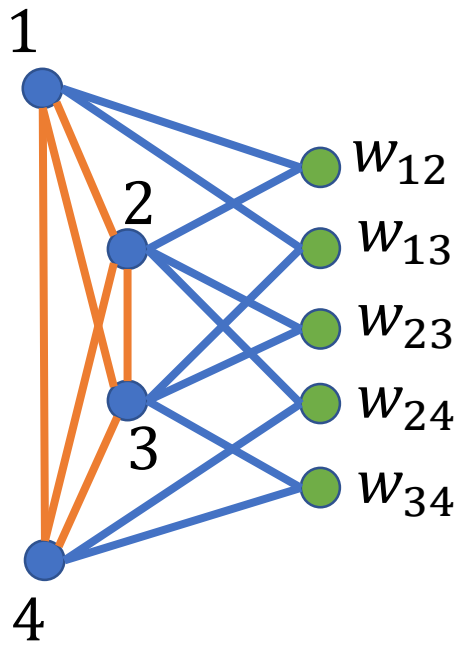
fixing it

Fix the Reduction

- **Lemma**. G contains a vertex cover of size k if and only if H contains a dominating set of size k.



G



H

VertexCover $\leq_T^P$ DominatingSet

- Given a polynomial time algorithm $\mathcal{A}$ for solving DominatingSet,
- We describe a polynomial time algorithm to solve VertexCover.
- Convert the VertexCover input (G, k) to (H, k) described earlier.
- Use $\mathcal{A}$ to decide if H has a dominating set of size k.
- If H has a dominating set of size k, output yes.
- Otherwise, output no.

Theorem. DominatingSet is NP-complete

- Proof. DominatingSet is in NP. (Can you prove it?)
- We have proved VertexCover $\leq_T^P$ DominatingSet
- VertexCover is NP-complete $\Longrightarrow$ DominatingSet is NP-hard
- Thus, DominatingSet is NP-complete.

Let’s go back to k-centers

DominatingSet vs k-centers

- If G has a dominating set of size k, what can we say about the optimal solution for k-centers?
- What if G does not have a dominating set of size k?

DominatingSet vs k-centers

- If G has a dominating set of size k, then the optimal solution for k-centers has cost 1.
- If G does not have a dominating set of size k, then the optimal solution for k-centers has cost at least 2.

Reversely:

- If the optimal solution for k-centers has cost 1, then G has a dominating set of size k.
- If the optimal solution for k-centers has cost at least 2, then G does not have a dominating set of size k.

Discussion: If we have a polynomial time 1.99-approximation algorithm for k-centers, can you design a polynomial time algorithm for solving DominatingSet?

Reduction

- Suppose we have a polynomial time 1.99-approximation algorithm $\mathcal{A}$ for k-centers.
- Consider an input (G, k) of the DominatingSet problem.
- Implement $\mathcal{A}$ on (G, k) to find a set S of k centers.
- Let $\mathbf{ALG}$ be the cost output by $\mathcal{A}$, and $\mathbf{OPT}$ be the optimal cost.
- If $\mathbf{ALG} = 1$, then $\mathbf{OPT} = 1$, and (G, k) is a yes DominatingSet input.
- If $\mathbf{ALG} \geq 2$, then $\mathbf{OPT} \geq \frac{\mathbf{ALG}}{1.99} \geq \frac{2}{1.99} > 1$.
- We have $\mathbf{OPT} \geq 2$ as $\mathbf{OPT}$ takes integer values.
- This implies (G, k) is a no DominatingSet input.

Hardness-of-approximation for k-centers

- **Theorem**. A polynomial time $(2-\varepsilon)$-approximation algorithm with any $\varepsilon > 0$ for the k-centers problem implies P = NP.

- A polynomial time $(2-\varepsilon)$-approximation algorithm can be used to solve the NP-complete problem DominatingSet in polynomial time!

更多不可近似性

PCP定理

- PCP: probabilistic checkable proof
- PCP定理：$\mathrm{NP} = \mathrm{PCP}\left[O\left(\log(n)\right), O(1)\right]$
- PCP定理之一个等价表述：
- **Theorem**. There exists a constant $c > 0$ such that a polynomial time $(1 - c)$-approximation algorithm for 3-SAT implies $\mathrm{P} = \mathrm{NP}$.

基于PCP定理的不可近似性结论

- **[Max-3SAT]** For any $\varepsilon > 0$, a polynomial time $\left(\frac{7}{8} + \varepsilon\right)$-approximation algorithm implies $\mathrm{P} = \mathrm{NP}$. [Håstad, 2001]
- **[VertexCover]** For any $\varepsilon > 0$, a polynomial time $\left(\sqrt{2} - \varepsilon\right)$-approximation algorithm implies $\mathrm{P} = \mathrm{NP}$. [Khot, Minzer& Safra, 2017]
- **[IndependentSet]** For any $\varepsilon > 0$, a polynomial time $\left(\frac{1}{n^{1-\varepsilon}}\right)$-approximation algorithm implies $\mathrm{P} = \mathrm{NP}$. [Håstad, 1999] [Khot, 2001] [Zuckerman, 2006]
  - Can you give an n-approximation algorithm?
  - an $O\left(\frac{n(\log\log n)^2}{(\log n)^3}\right)$-approximation algorithm [Feige, 2004]