Lecture 12: Approximation Algorithms

Yury Makarychev
TTIC and the University of Chicago

Approximation Algorithms

- There are NP-hard optimization problems.
- They cannot be solved in polynomial time if $P \neq NP$.
- But many of them have numerous applications…

Solution: Design approximation algorithms for them.

$\mathcal{A}$ is an approximation algorithm for problem X with approximation factor $\alpha \geq 1$ if it finds a solution with value/cost within a factor of α of the optimum:

(maximization) $ALG \geq OPT/\alpha$

(minimization) $ALG \leq \alpha OPT$

Approximation Algorithms

$\mathcal{A}$ is an approximation algorithm for problem X with approximation factor $\alpha \geq 1$ if it finds a solution with value/cost within a factor of α of the optimum:

(maximization) $ALG \geq OPT/\alpha$

(minimization) $ALG \leq \alpha OPT$

Notation: For brevity, we say "an α-approximation algorithm".

For maximization problems, we sometimes say a $\beta \leq 1$ approximation.

$$\beta\text{-approximation} \equiv (1/\beta)\text{-approximation}$$

E.g., 2-approximation algorithm $\equiv$ 0.5-approximation algorithm.

We will consider only polynomial-time approximation algorithms.

Knapsack Problem

Consider the Knapsack problem with arbitrary item weights and values.

Recall:

- there are n items
- item i has weight $w_i > 0$ and value $v_i > 0$
- the knapsack capacity is a given parameter W

Goal: find a subset S of items with total weight $\sum_{i \in S} w_i \leq W$ so as to maximize

$$\sum_{i \in S} v_i$$

Knapsack Problem

- If all weights are integral, the problem can be solved in time $O(nW)$.
- If all values are integral, the problem can be solved in time $O(nV)$ where $V = \sum v_i$.
- In general, the problem is NP-hard.

We will show that for every $\varepsilon > 0$, there is a $(1+\varepsilon)$-approximation algorithm for the problem. Thus, the problem can be solved with an arbitrary precision.

Knapsack Problem: Special Case

Assume first that

- all values v_i are multiples of some a
- $v_i \leq Na$

By diving by all values by a (rescaling), we get an equivalent problem with integer values v_i.

This problem can be solved in time $O(n^2N)$ using dynamic programming.

Knapsack Problem: Special Case

DP Table: $T[i, v]$ with $i \in \{0, \dots, n\}$ and $v \in \{1, \dots, \sum v_i\}$.

> $T[i, v]$ equals the minimum weight of a set of items $S \subset \{1, \dots, i\}$ whose value is $v \cdot a$.

Initialization: $T[0, v] = 0$ for all v

Recurrence:

$$T[i, v] = \max(T[i-1, v], T[i-1, v - v_i/a]) \quad \text{if } v \geq v_i/a$$

$$T[i, v] = T[i-1, v] \quad \text{otherwise}$$

Output: $\max(\{v \cdot a : T[n, v] \leq W\})$

Knapsack Problem: General Case

Preprocessing: remove all items of weight $w_i > W$; no feasible solution can have them.

Let $M = \max_i v_i$ be the value of the most valuable item. Let $a = \frac{\varepsilon M}{n}$.

Round every v_i to a multiple of a:

$$v_i' = \left\lceil {}^{v_i}/_{a} \right\rceil \cdot a$$

Knapsack Problem: General Case

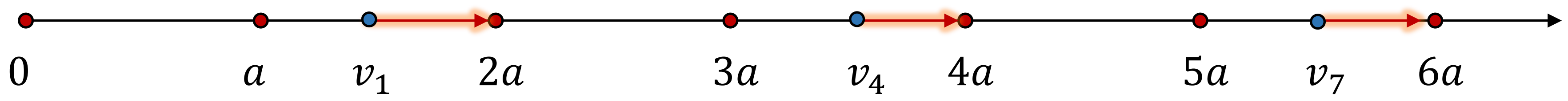
We get an instance in which all values v_i' are multiples of a.

$$\frac{v_i'}{a} = \left\lceil \frac{v_i}{a} \right\rceil \le \left\lceil M/(\frac{\varepsilon M}{n}) \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

Solve the new instance in time $O\left(n^2 \cdot \frac{n}{\varepsilon}\right) = poly\left(n, \frac{1}{\varepsilon}\right)$.

Obtain a solution S.

Note that S is a feasible solution for the original instance: $\sum_{i \in S} w_i \le W$.



$$a = \frac{\varepsilon M}{n}$$

Let S^* be an optimal solution for the original instance,

S be an optimal solution for the new instance, which we found.

$$v(S) = \sum_{i \in S} v_i \geq \sum_{i \in S} (v_i' - a) \geq \left(\sum_i v_i' \right) - na = v'(S) - na$$

(why?) $\geq v'(S^*) - na \geq v(S^*) - na = v(S^*) - \varepsilon M \geq (1 - \varepsilon) \cdot v(S^*)$

why?

We get a $(1 - \varepsilon)$-approximation algorithm.

PTAS: Polynomial-time Approximation Scheme

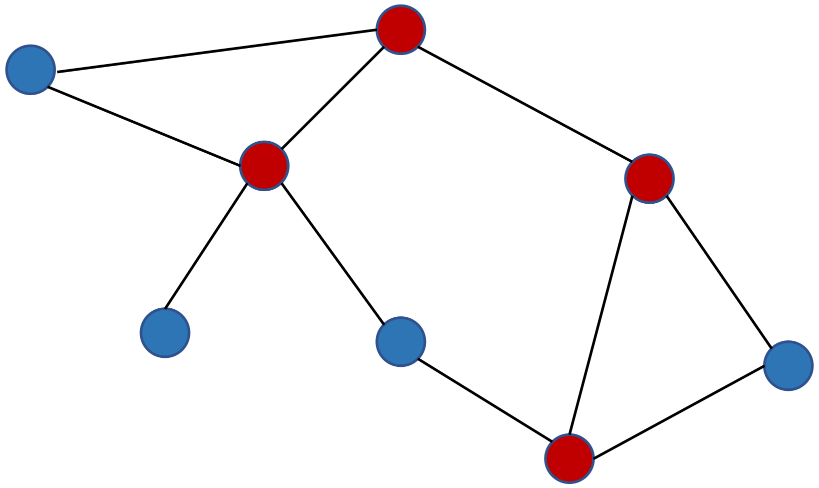
If for every $\varepsilon > 0$, a problem admits a $(1+\varepsilon)$-approximation, we say that there is a polynomial-time approximation scheme for it.

We showed that there is a PTAS for Knapsack.

However, there are no PTAS for many other problems if $P \neq NP$.

Minimum Vertex Cover

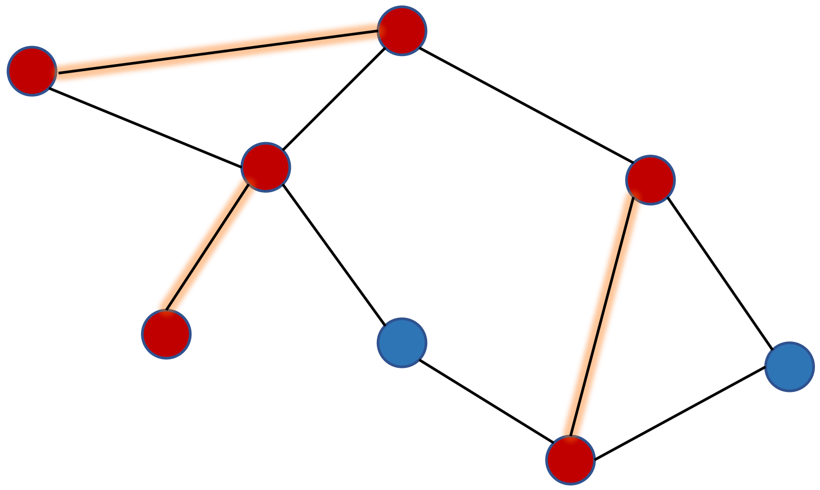
Given a graph $G = (V, E)$, find a vertex cover A of minimum size.



Minimum Vertex Cover

Given a graph $G = (V, E)$, find a vertex cover A of minimum size.

- Find a maximal matching M
- Let A be the set of vertices matched by M.



Claim: This is a 2-approximation algorithm.

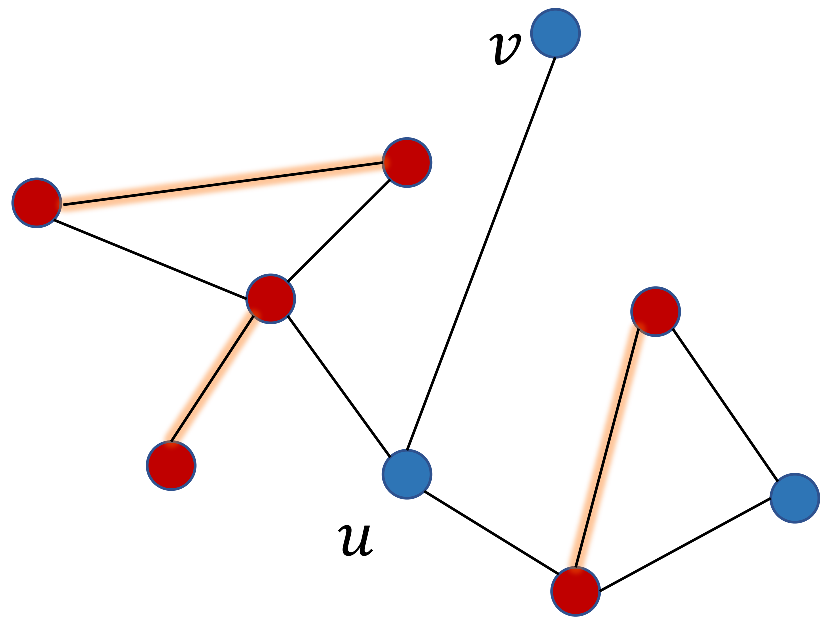
Minimum Vertex Cover

Proof:

a) Verify that A is a vertex cover. Assume that there is an edge (u, v) not covered by A.

Then u and v are not matched by M.
$M \cup \{(u, v)\}$ is a larger matching than M.

Contradiction!



Minimum Vertex Cover

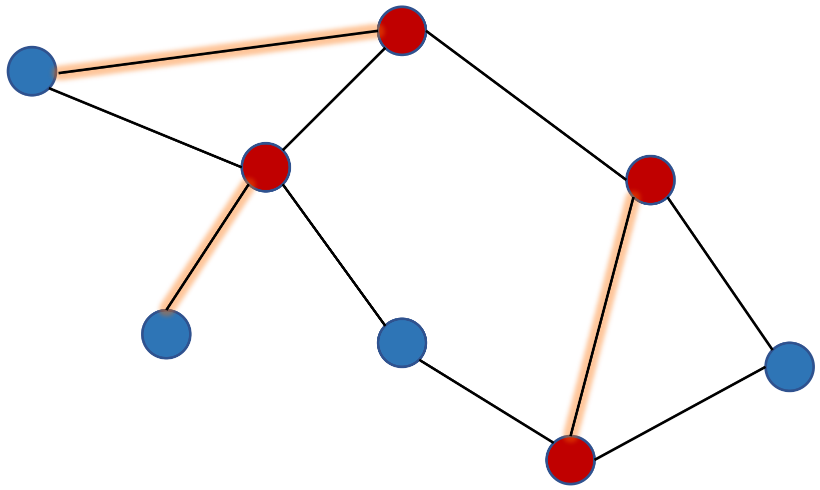
b) Prove that $|A| \leq 2|A^*|$.

At least one of the endpoints of every edge in G is in A^*.

Thus, at least one of the endpoints of every edge in M is in A^*.

Distinct edges in M don't share endpoints.

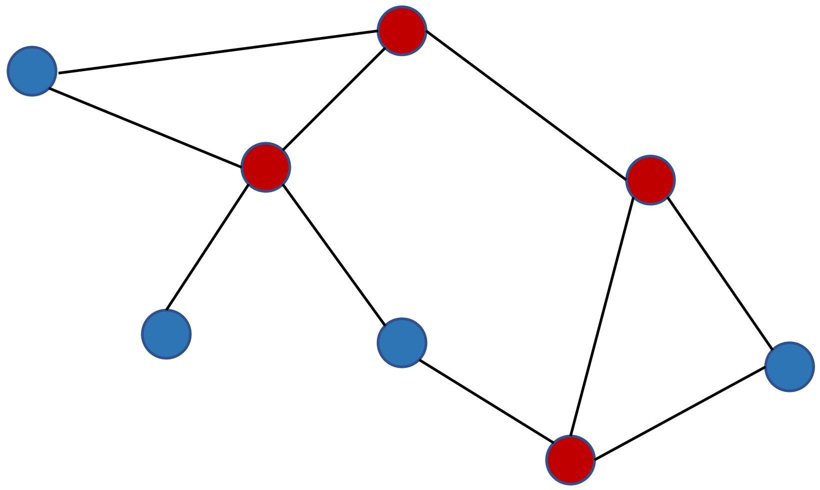
$\Rightarrow |A^*| \geq |M|$
$\Rightarrow |A| = 2|M| \leq 2|A^*|$



Minimum Weight Vertex Cover

Assume that each vertex v in G has weight $w_v > 0$.

Goal: Find a vertex cover A of minimum weight.



Use Linear Programming!

variables: x_u

min $\sum_{u \in V} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \in \{0,1\}$ for every $u \in V$

Minimum Weight Vertex Cover

Use Linear Programming!

variables: x_u

min $\sum_{u \in V} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \in \{0,1\}$ for every $u \in V$

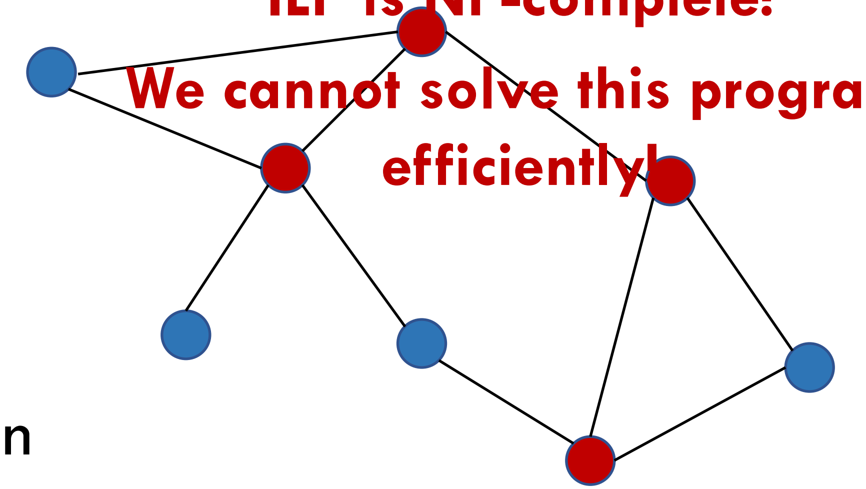
There is a one-to-one correspondence between vertex covers and ILP solutions:

$A = \{u : x_u = 1\}$

$\Rightarrow ILP = OPT$

ILP is NP-complete!

We cannot solve this program efficiently!



LP Relaxation

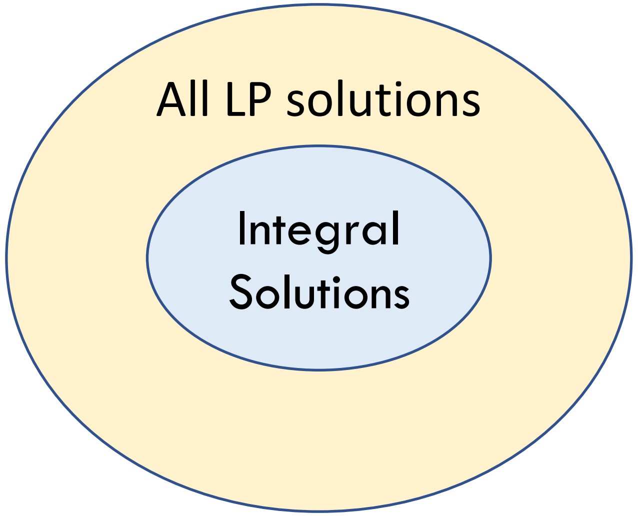
Use Linear Programming!

variables: x_u

min $\sum_{u \in \mathrm{V}} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \geq 0$ for every $u \in V$



Now: For every vertex cover M there is a corresponding solution x_u.

An LP solution x_u might not correspond to any vertex cover.

LP Relaxation

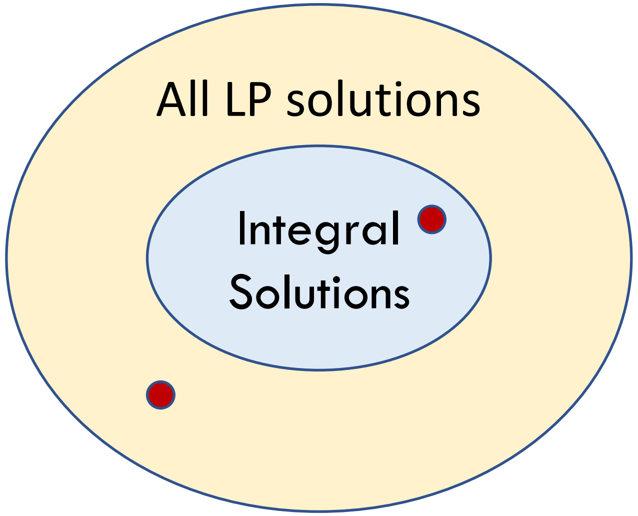
Use Linear Programming!

variables: x_u

min $\sum_{u \in \mathrm{V}} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \geq 0$ for every $u \in V$



Claim: $LP \leq OPT$ Why?

This is an LP relaxation.

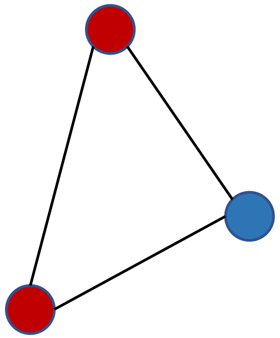
LP Relaxation

variables: x_u

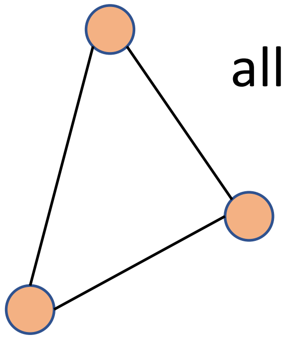
min $\sum_{u \in V} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \geq 0$ for every $u \in V$



$OPT = 2$



all $x_u = 1/2$

$LP = 3/2$

The integrality gap between LP and OPT is $\frac{OPT}{LP} = \frac{4}{3}$ for this instance.

LP Relaxation

variables: x_u

min $\sum_{u \in \mathrm{V}} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \geq 0$ for every $u \in V$

1. Solve the LP relaxation
2. ("Rounding step") $S = \left\{u: x_u \geq \frac{1}{2}\right\}$
3. return S

LP Relaxation

variables: x_u

min $\sum_{u \in V} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \geq 0$ for every $u \in V$

1. Solve the LP relaxation
2. ("Threshold rounding") $S = \left\{u: x_u \geq \frac{1}{2}\right\}$
3. return S

Claim: S is a vertex cover.

Proof: Consider edge (u, v). Then $x_u + x_v \geq 1$.

Therefore, either $x_u \geq 1/2$ or $x_v \geq 1/2$.

Thus, $u \in S$ or $v \in S$, as required.

LP Relaxation

variables: x_u

min $\sum_{u \in V} w_u x_u$

$x_u + x_v \geq 1$ for every $(u, v) \in E$

$x_u \geq 0$ for every $u \in V$

1. Solve the LP relaxation
2. ("Threshold rounding") $S = \left\{u: x_u \geq \frac{1}{2}\right\}$
3. return S

Claim: $w(S) \leq 2OPT$

Proof:

$$w(S) = \sum_{u \in S} w_u \leq \sum_{u \in S} (2x_u) \cdot w_u = 2\sum_{u \in S} w_u x_u \leq 2\sum_{u \in V} w_u x_u = 2LP$$

Minimum Vertex Cover: Better Approximation?

The state-of-the-art hardness of approximation result is based on a plausible complexity assumption

Unique Games Conjecture.

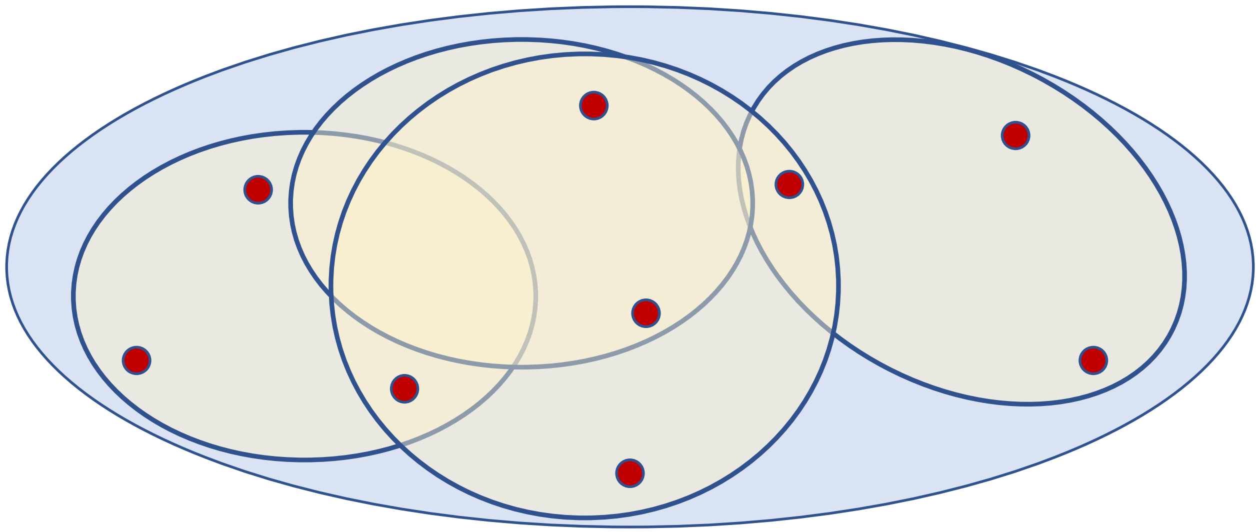
If Unique Games Conjecture is true, then

there is no $(2-\varepsilon)$-approximation for Minimum Vertex Cover.

Set Cover

Given: a set X of n elements and m subsets $S_1, \dots, S_m \subseteq X$.

Find: a collection of subsets S_i of smallest cardinality that covers entire X.



Set Cover

Given: a set X of n elements and m subsets $S_1, \dots, S_m \subseteq X$.

Find: a collection of subsets S_i of smallest cardinality that covers entire X.

Greedy algorithm: Suggestions?

Set Cover

Given: a set X of n elements and m subsets $S_1, \ldots, S_m \subseteq X$.

Find: a collection of subsets S_i of smallest cardinality that covers entire X.

Greedy algorithm:

- Let A_1 be the subset that covers most of elements in X.
- Let $A_2 = ?$

Set Cover

Given: a set X of n elements and m subsets $S_1, \dots, S_m \subseteq X$.

Find: a collection of subsets S_i of smallest cardinality that covers entire X.

Greedy algorithm:

- Let A_1 be the subset that covers most of elements in X.
- Let A_2 be the subset that covers most elements in $X \setminus A_1$.

…

- Let A_i be the subset that covers most elements in $X \setminus (A_1 \cup A_2 \cup \cdots \cup A_{i-1})$

Set Cover

We assume that every point $x \in X$ is covered by at least one set S_i.

The algorithm will find a feasible solution in at most $\min(m, n)$ iterations.

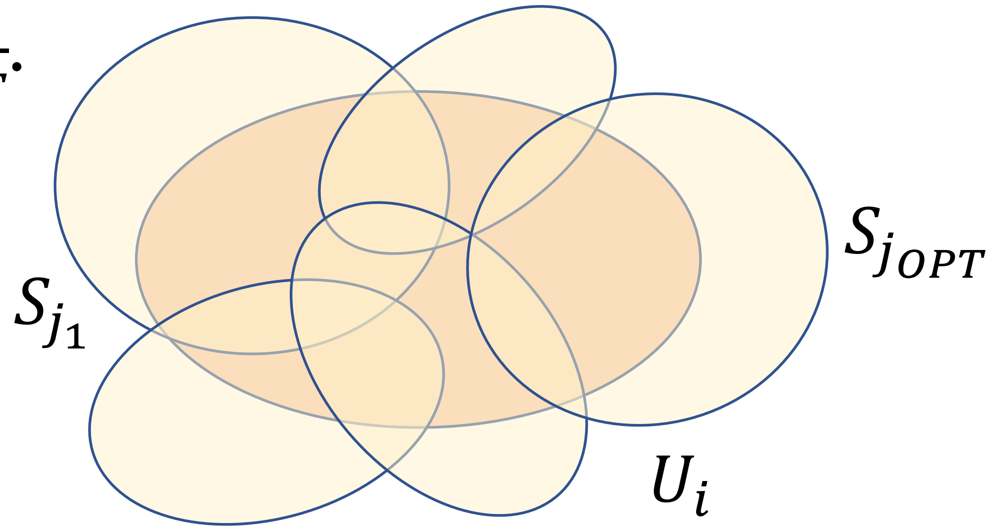
Let ℓ be the number of sets in the solution we found.

Set Cover

Claim: $\ell \leq H(n) \cdot OPT$, where $H(n) = \sum_i \frac{1}{i} = \ln n + O(1)$.

Proof: Consider iteration i.

- Let U_i be the set of uncovered elements (when iteration i starts).
- Some S_j covers at least $|U_i|/OPT$ elements in U_i. Why?
- Hence, $|A_i \cap U_i| \geq \frac{|U_i|}{OPT}$.



Basic algebra

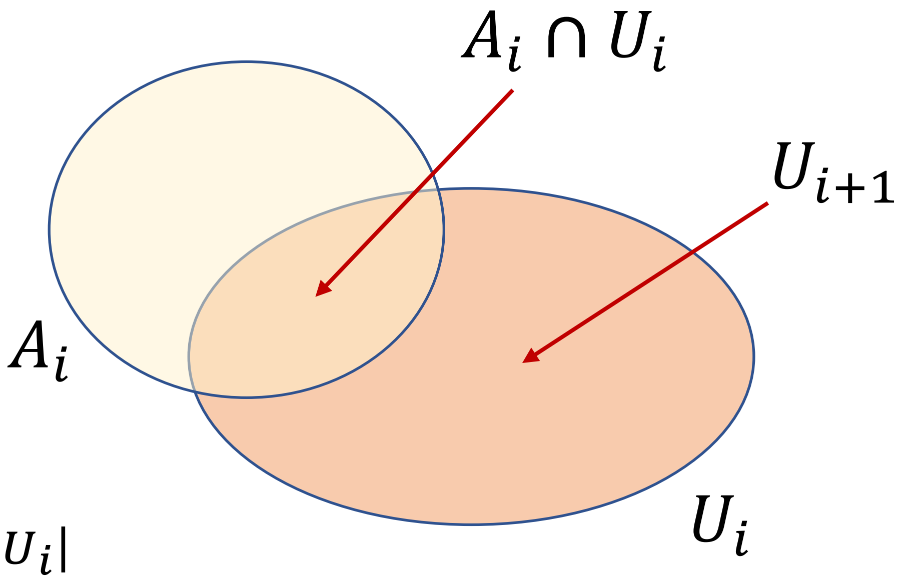
Let x and y be integers: $x > y \geq 0.$

$$\frac{x-y}{x} = \underbrace{\frac{1}{x} + \cdots + \frac{1}{x}}_{\color{blue}{x-y \text{ summands}}} \leq \frac{1}{y+1} + \frac{1}{y+2} + \cdots + \frac{1}{x-1} + \frac{1}{x}$$

$$= \left(1 + \cdots + \frac{1}{x}\right) - \left(1 + \cdots + \frac{1}{y}\right) = H_x - H_y$$

Conclusion: $\frac{x-y}{x} \leq H_x - H_y$

Set Cover



- $|U_i| - |U_{i+1}| = |A_i \cap U_i| \geq \frac{|U_i|}{OPT}$
- $\frac{|U_i| - |U_{i+1}|}{|U_i|} \leq H_{|U_i|} - H_{|U_{i+1}|}$

Conclusion: $H_{|U_i|} - H_{|U_{i+1}|} \geq \frac{1}{OPT}$

Set Cover

Keep track of $H_{|U_i|}$:

- Initially, $|U_1| = n$ and thus $H_{|U_1|} = H_n$.
- At each iteration, $H_{|U_i|}$ decreases by at least:

$$H_{|U_i|} - H_{|U_{i+1}|} \geq \frac{1}{OPT}$$

- After the algorithm terminates, $|U_{\ell+1}| = |\emptyset| = 0$ and thus $H_{|U_{\ell+1}|} = 0$.

Thus, $\ell \leq H_n \cdot OPT$ (why?).

Set Cover: Randomized Rounding

variables: $x_1, \dots, x_m$

intended solution: if S_i is chosen, $x_i = 1$; otherwise, $x_i = 0$

min $\sum_{i=1}^{m} x_i$

$\sum_{i:u \in S_i} x_i \geq 1$ for every $u \in X$

$x_i \geq 0$ for all i

Exercise: prove that this is a relaxation.

Randomized Rounding

variables: $x_1, \dots, x_m$

intended solution: if S_i is chosen, $x_i = 1$; otherwise, $x_i = 0$

min $\sum_{i=1}^{m} x_i$

$\sum_{i:u \in S_i} x_i \geq 1$ for every $u \in X$

$x_i \geq 0$ for all i

Attempt #1

- For every i, choose set S_i with probability x_i.
- Make all choices independently.

Q: What is the expected number of chosen sets?

$$LP$$

Q: What is the probability that a given element u is covered by one of the chosen sets?

$$\Pr(u \text{ is } \text{not} \text{ covered}) = \prod_{i:u \in S_i} (1 - x_i) \leq \prod_{i:u \in S_i} e^{-x_i} = e^{-\sum_{i:u \in S_i} x_i} \leq \frac{1}{e}$$

Randomized Rounding

Attempt #2

- For every i, choose set S_i with probability x_i.
- Make all choices independently.
- Repeat $2 \ln n$ times.

Q: What is the expected number of chosen sets?

$$(2 \ln n) LP$$

Q: What is the probability that a given element u is covered by one of the chosen sets?

$$\Pr(u \text{ is not covered}) \leq \frac{1}{e^{2 \ln n}} = \frac{1}{n^2}$$

Randomized Rounding

Union Bound: the probability that that there is an uncovered element is at most $n \cdot \frac{1}{n^2} = \frac{1}{n}$.

Success: With probability at least $1 - 1/n$, we cover all the elements of X.

$$\mathrm{E}[\#\text{chosen elements} \mid \text{success}] = \frac{\mathrm{E}[(\#\text{chosen elements}) \cdot 1_{\text{success}}]}{\Pr(\text{success})}$$

$$\leq \frac{E[\#\text{chosen elements}]}{\Pr(\text{success})} \leq \frac{2 \ln n}{1 - 1/n}$$

Set Cover: Better Approximation?

If $P \neq NP$, we cannot get an $O((1-\varepsilon)\log_e n)$ approximation for any $\varepsilon > 0$.

Metric TSP: Travel Salesman Problem

Given: a graph $G = (V, E)$ with edge lengths ℓ_{uv}.

A tour in G is a cycle that may visit the same vertex and edge multiple times.

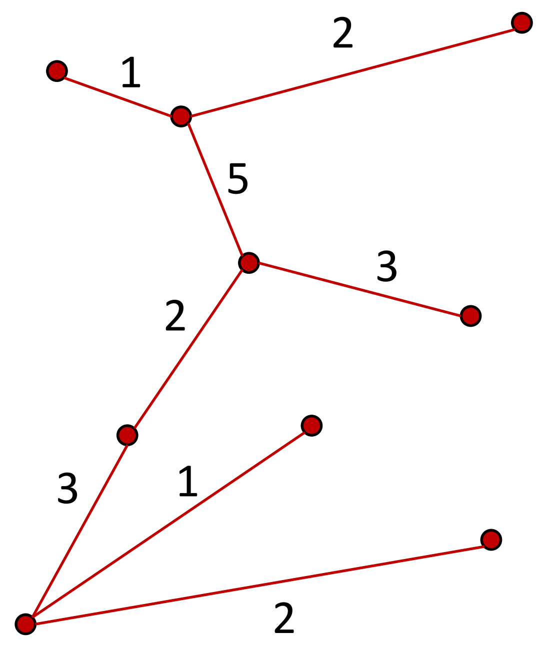
Goal: Find a tour of minimum length that visits all the vertices in the graph.

The problem is NP-hard.

For brevity, a tour will refer to a tour that visits all the vertices in the graph below.

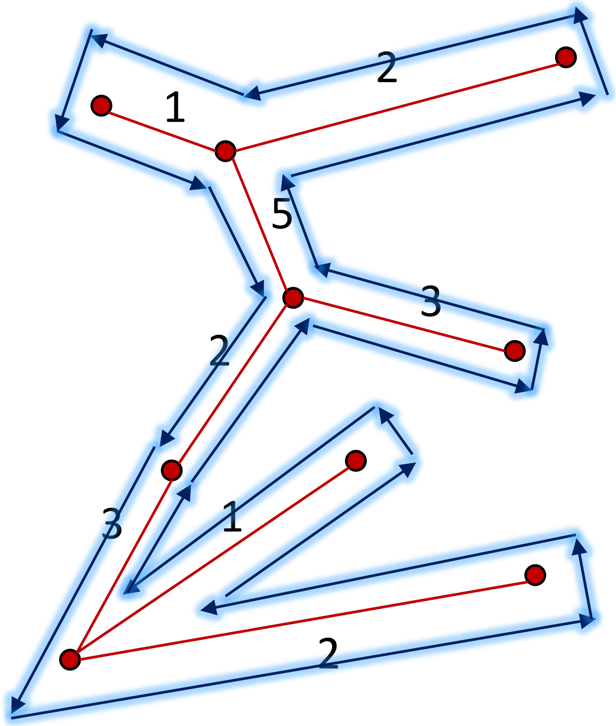
MST: Walking around a tree

- Assume that G is a tree
- Every tour must visit each edge at least 2: once cross it in one direction, and once in the opposite one.



MST: Walking around a tree

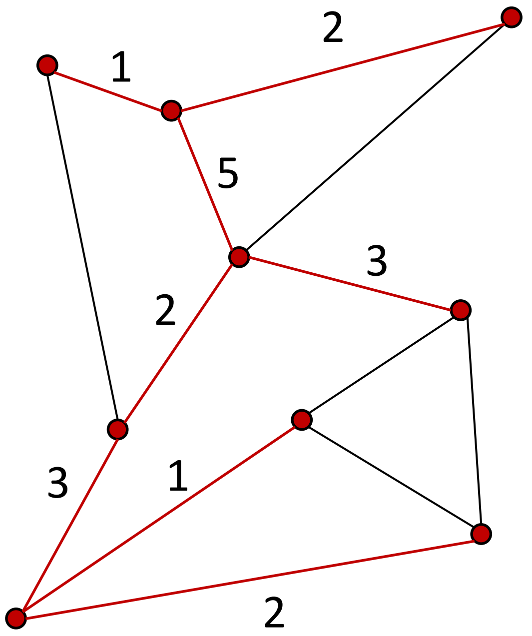
- Assume that G is a tree
- Every tour must visit each edge at least 2: once cross it in one direction, and once in the opposite one.
- There is tour that visits every edge exactly twice.



Minimum Spanning Tree

- Now G is arbitrary.
- Algorithm

> • Find an MST T is G
>
> • Let C be the tour in T of length $2\ MST$, where MST is the cost of the spanning tree T.



MST vs OPT

Claim: $MST \leq length(C')$ for every tour C'

Proof: Consider C'.

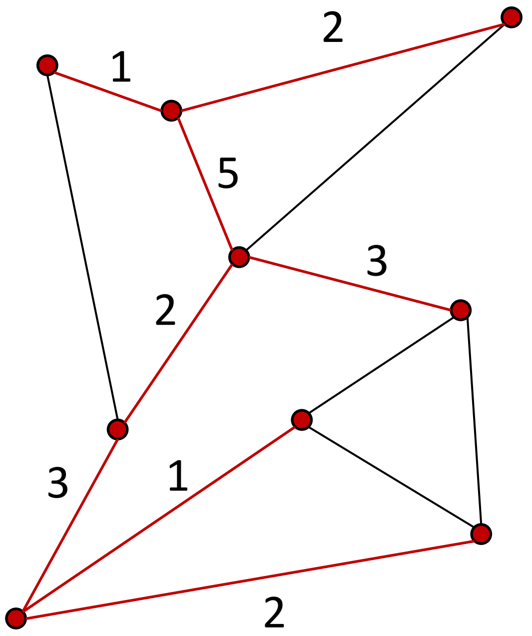
Let H be the subgraph of G on V formed by the edges of C'.

- H is a spanning subgraph.

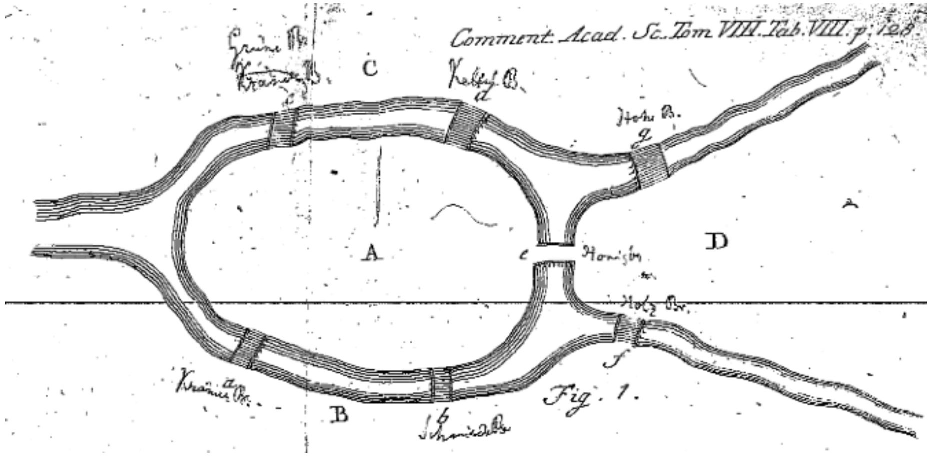
Let T' be a minimum spanning tree in H. Then

$$MST = length(T) \leq length(T') \leq \sum_{e \in H} \ell(e) \leq length(C')$$

Exercise: prove that $MST \leq \frac{n-1}{n}\, length(C')$



Eulerian Graphs



Corollary: $length(C) \leq 2MST \leq 2OPT$

Our algorithms gives a 2-approximation.

Eulerian Cycles and Graphs

Consider a graph $U = (V_U, E_U)$. U may have parallel edges.

- A cycle C in U is Eulerian if it visits every edge exactly once.
- U is an Eulerian graph if it has an Eulerian cycle.

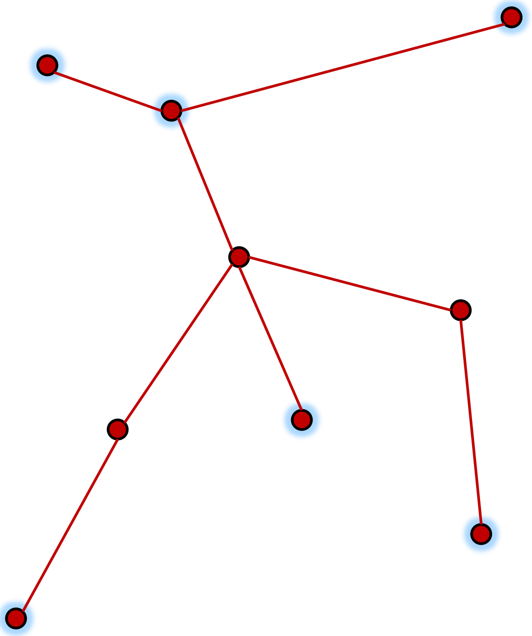
Theorem (Euler): Assume U is connected. Then U is Eulerian if and only if every vertex in U has an even degree.

TSP: Christofides algorithm

Consider MST T in G.

Let A be the set of odd degree vertices in T.

A: all leaves of T and odd degree branching points.



Claim: $|A|$ is even.

Proof: $\sum_u \deg_T u = 2|E|$. Hence, $\sum_u \deg_T u$ is even.

- $\sum_{u \notin A} \deg_T u$ is even (why?)
- Therefore, $\sum_{u \in A} \deg_T u$ is even,
- and $|A|$ is even.

TSP: Christofides algorithm

Construct an auxiliary graph H on A:

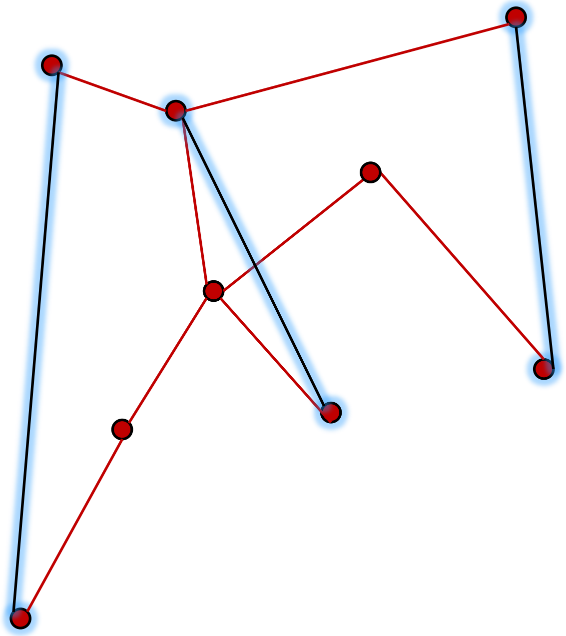
Connect every two vertices u and v with an edge (u, v) of length $d_G(u, v)$.

$|A|$ is even $\Rightarrow$ There is a perfect matching in H.

Let M be the minimum cost perfect matching in H.

(It's possible to find it using linear programming.)

Add edges from M to T: get an Eulerian graph (why?)

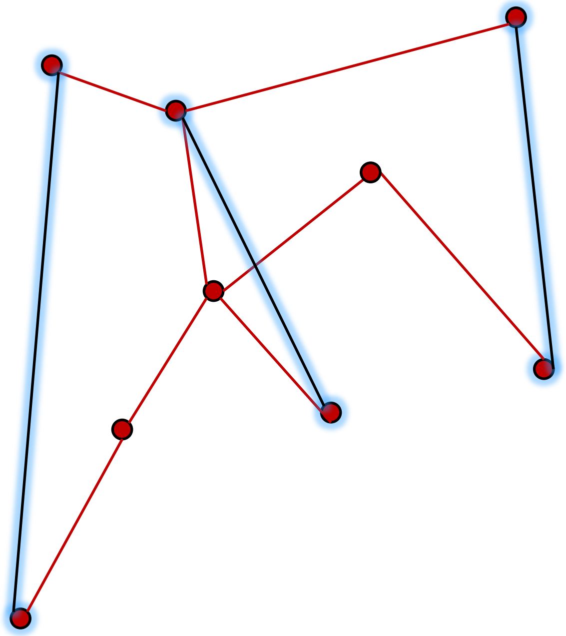


TSP: Christofides algorithm

$T + M$ is an Eulerian graph. Let C be an Eulerian graph in $T + M$.

$$\begin{aligned} length(C) &= length(T) + length(M) \\ &\leq OPT + \mathrm{cost}(M) \end{aligned}$$

need to upper bound this term

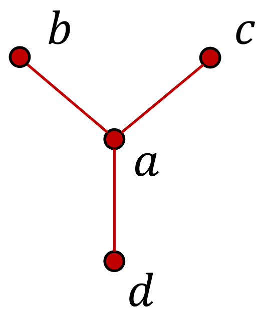


TSP: Christofides algorithm

Consider an optimal tour C^* in G: $C^* = c_1 \to c_2 \to \cdots \to c_T \to c_1$.

Note that some vertices may appear multiple times: e.g., $c_1 = c_5 = c_7 = c_{19}$.

Consider sequence $c_1, c_2, \cdots, c_T$. For every $u \in V$, keep only the first occurrence of u and remove all other occurrences of u from the sequence.



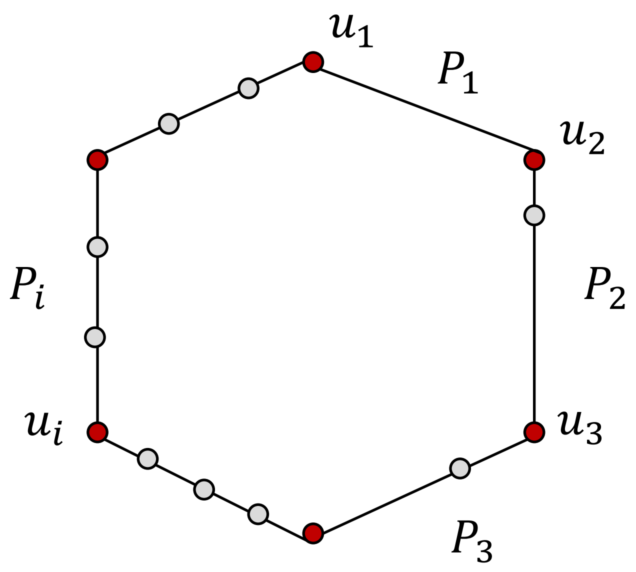
Example: $C^* = a \to b \to a \to c \to a \to d \to a$

Get sequence: a, b, c, d.

Denote the sequence by $u_1, u_2, \ldots, u_n$.

TSP: Christofides algorithm

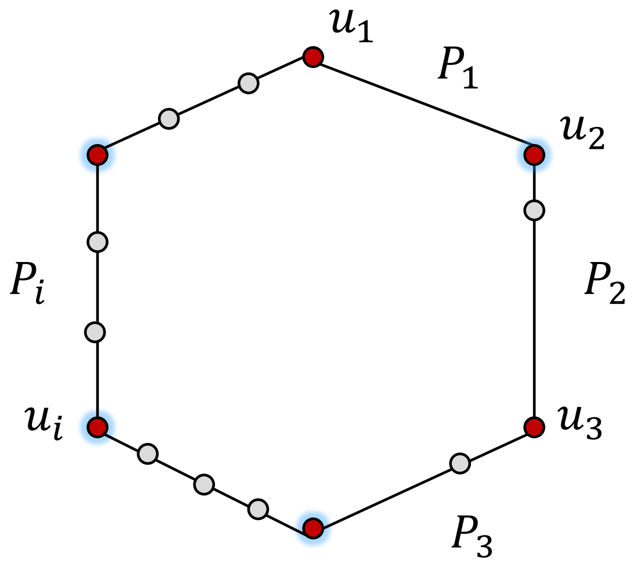
Denote the segment of C^* between u_i and u_{i+1} by P_i (where $u_{n+1} \equiv u_i$).



TSP: Christofides algorithm

Consider vertices from A (odd degree vertices in T).

Vertices from A split the tour into $|A|$ arcs/paths.

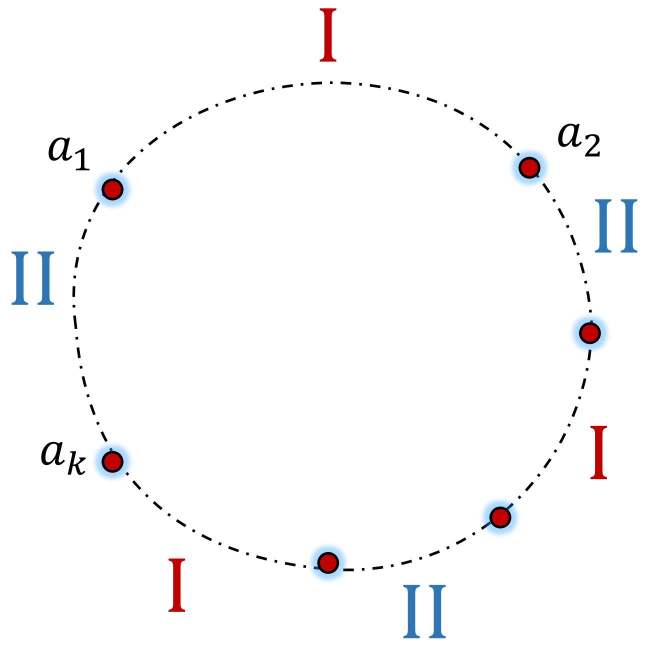


Vertices from A split the tour into $|A|$ arcs/paths.

Let I be the union of odd numbered arcs and II be the union of even numbered arcs.

Then $length(I) + length(II) = length(C^*) = OPT$.

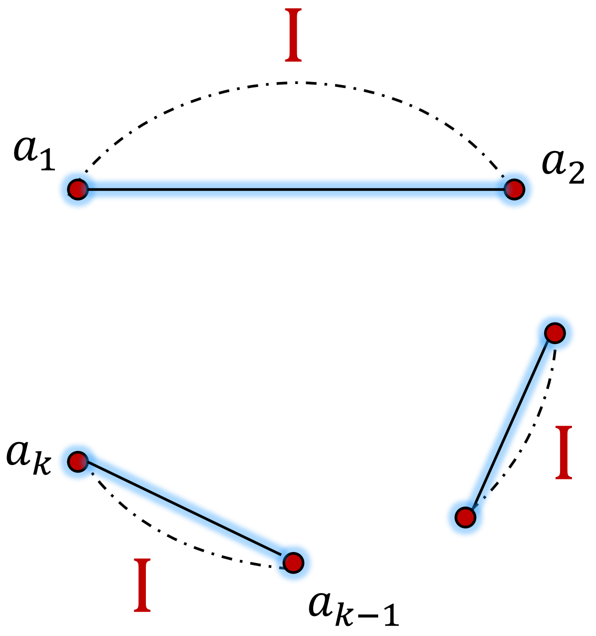
WLOG: $length(I) \leq OPT/2$



$length(I) \leq OPT/2$

I defines a matching M' on A:
2 vertices are matched if they are connected by an arc in I.

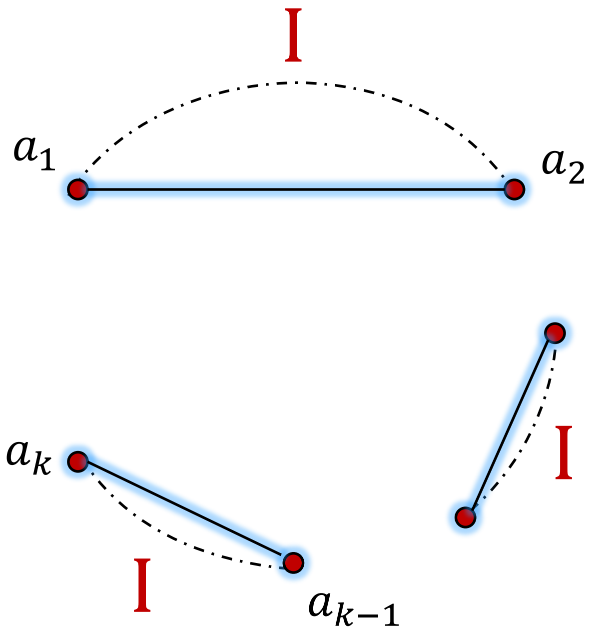
$$(a_1, a_2), (a_3, a_4), \ldots, (a_{k-1}, a_k)$$



$length(I) \leq OPT/2$

$$\mathrm{cost}(M') = d(a_1, a_2) + d(a_3, a_4) + \cdots + d(a_{k-1}, a_k) \leq length(I) \leq OPT/2$$

$$\Rightarrow \mathrm{cost}(M) \leq \mathrm{cost}(M') \leq OPT/2$$



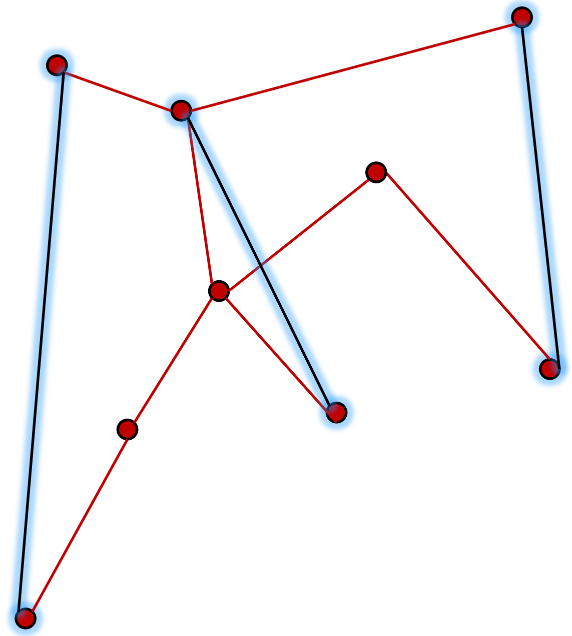
TSP: Christofides algorithm

$T + M$ is an Eulerian graph. Let C be an Eulerian graph in $T + M$.

$$\begin{aligned} length(C) &= length(T) + length(M) \\ &\leq OPT + cost(M) \\ &\leq {}^3/_2\ OPT \end{aligned}$$

We get a cycle C in $G + M$ of length $\frac{3}{2} OPT$.

Are we done?



TSP: Latest News

[1976] Christofides Algorithm

3/2-approximation

[2021] Karlin, Klein, and Oveis Gharan

$^3/_2 - \varepsilon$ approximation, where

$\varepsilon \sim 10^{-36}$

A (Slightly) Improved Approximation Algorithm for Metric TSP

Anna R. Karlin*, Nathan Klein†, and Shayan Oveis Gharan‡

University of Washington

May 11, 2021

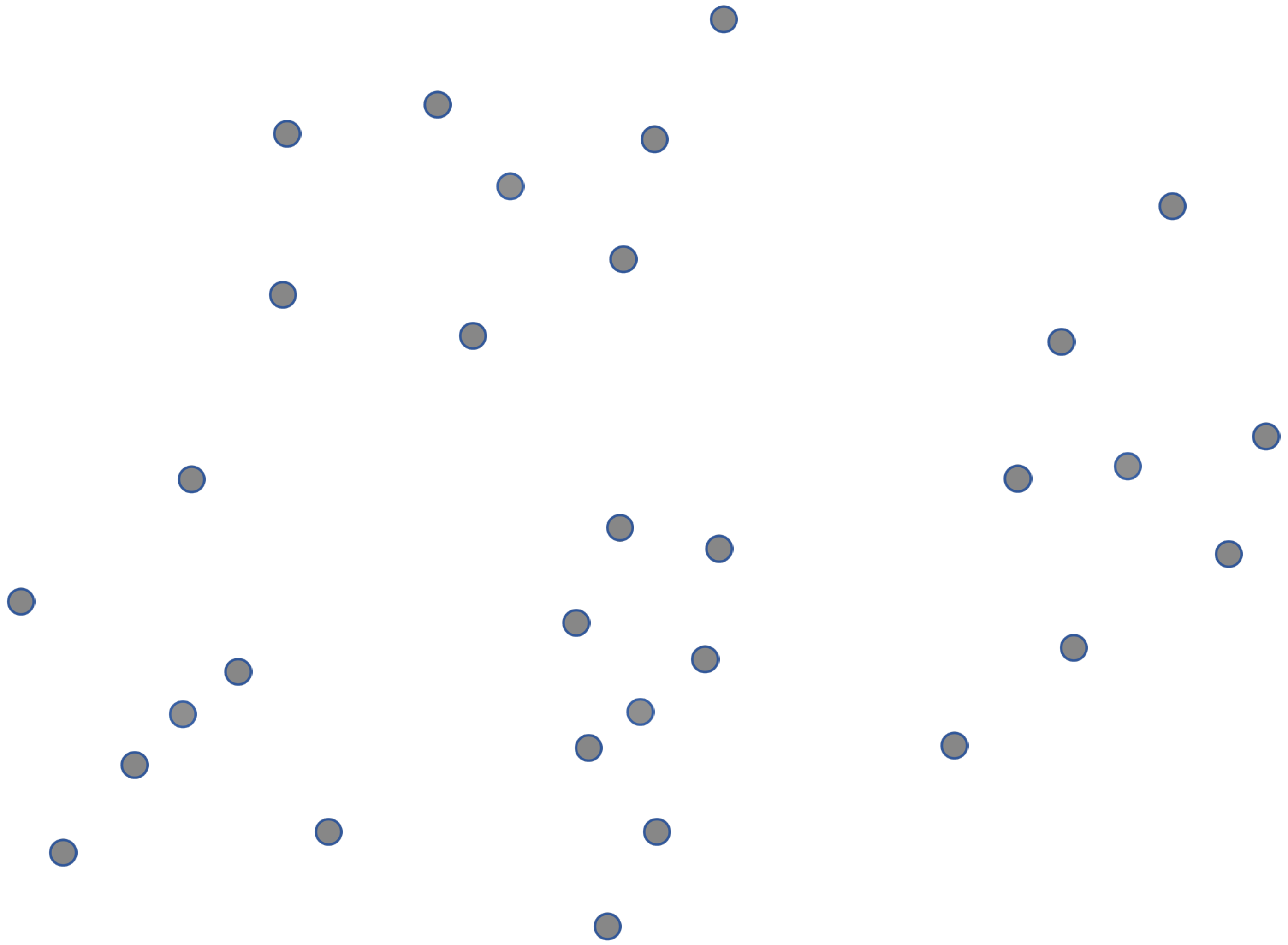
Abstract

For some $\epsilon > 10^{-36}$ we give a randomized $3/2 - \epsilon$ approximation algorithm for metric TSP.

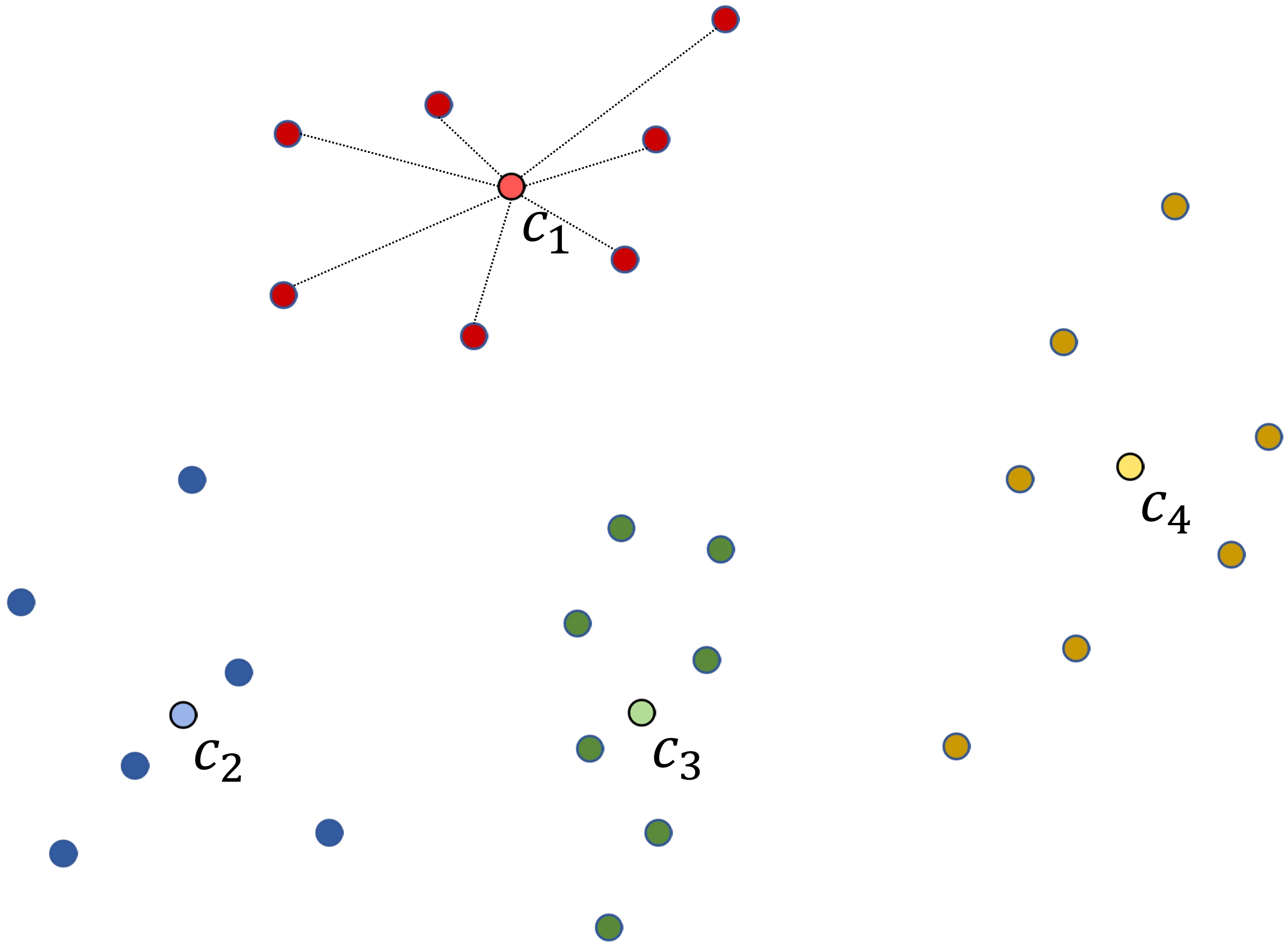
Clustering Problems

k-center, k-means, k-median

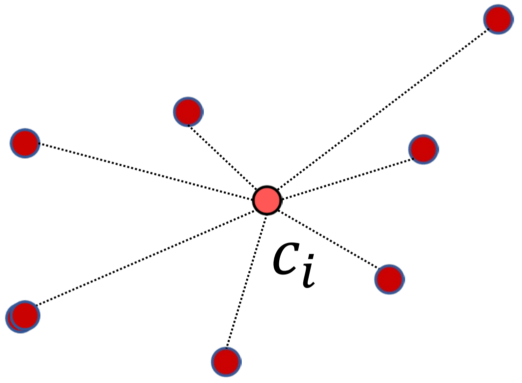
Clustering



Clustering



k-center, k-median, k-means



Given: a dataset X in $\mathbb{R}^m$ or an abstract metric space.

Goal: Find k-centers $c_1, \dots, c_k$ and assign each point $x \in X$ to the closest center $c(x)$. Get a clustering $C_1, \dots, C_k$.

Want to minimize

(k-center) $\max_{x \in X} d(x, c(x))$

(k-median) $\sum_{x \in X} d\big(x, c(x)\big)$

(k-means) $\sum_{x \in X} d\big(x, c(x)\big)^2$

Metric k-center

Given: a dataset X and a similarity measure/metric d on X

Goal: Find k-centers $c_1, \dots, c_k \in X$.

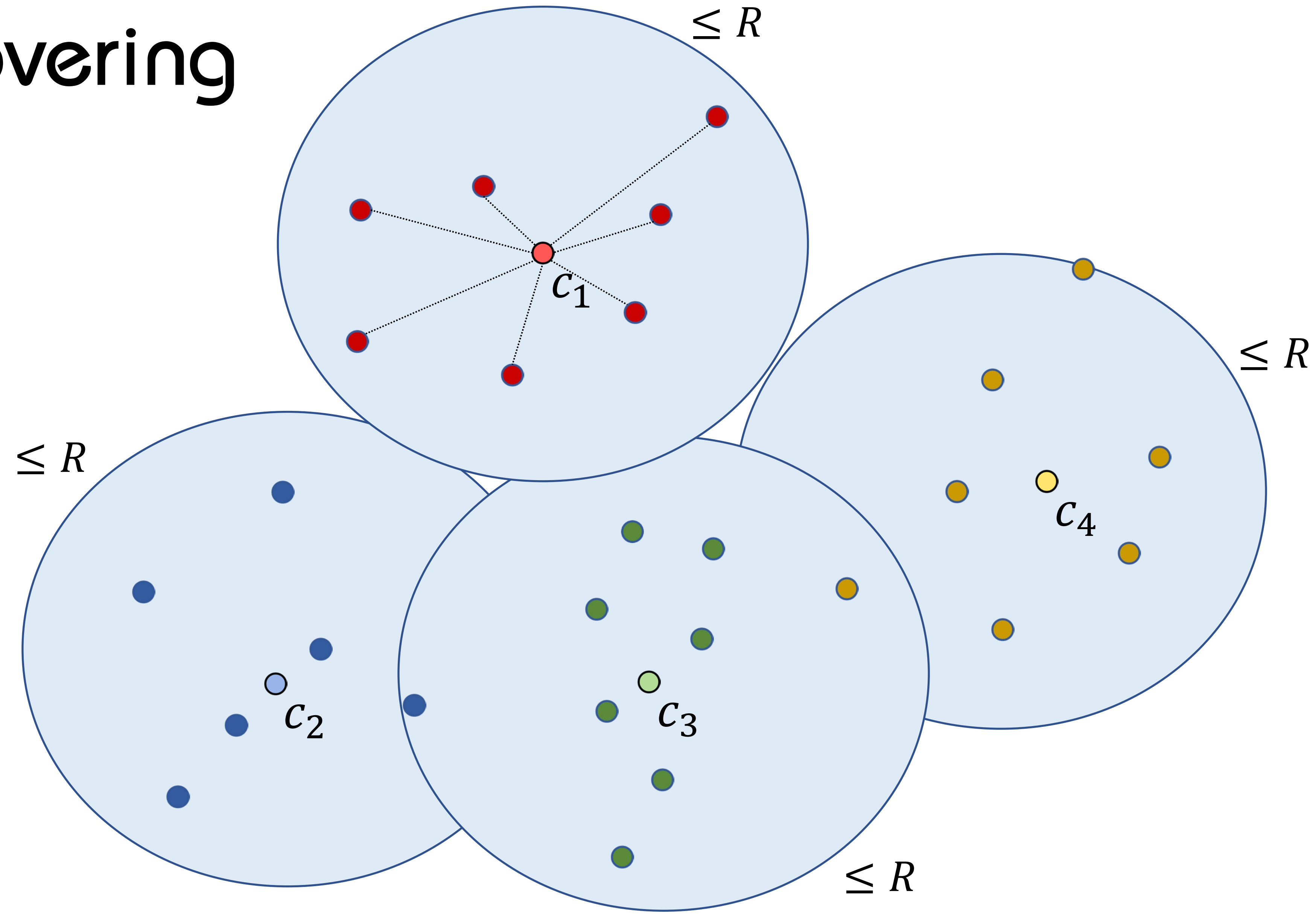
Assign each point $x \in X$ to the closest center $c(x)$.

Minimize $\max_{x \in X} d(x, c(x))$

Get a clustering $C_1, \dots, C_k$: $C_i = \{x \in X : c(x) = c_i\}$.

Assume $d(x,z) \le d(x,y) + d(y,z)$.

Ball Covering



Metric k-center

Metric k-center is NP-hard.

It's possible to get a 2-approximation.

There is no $(2-\varepsilon)$-approximation algorithm.

First, consider a decision version of the problem.

Given an instance and a parameter R.

- If there is a solution of cost $\leq R$, find a solution of cost $\leq 2R$.
- Otherwise, either report that there is no solution or find a solution of cost $\leq 2R$.

Metric k-center: Greedy Algorithm

Input: instance (X, d), parameters k and R

label all vertices as non-clustered

let $i = 0$

while there are unclustered points

    $i = i + 1$

    choose an arbitrary unclustered point c_i

    label all points at distance $\leq 2R$ from c_i as clustered

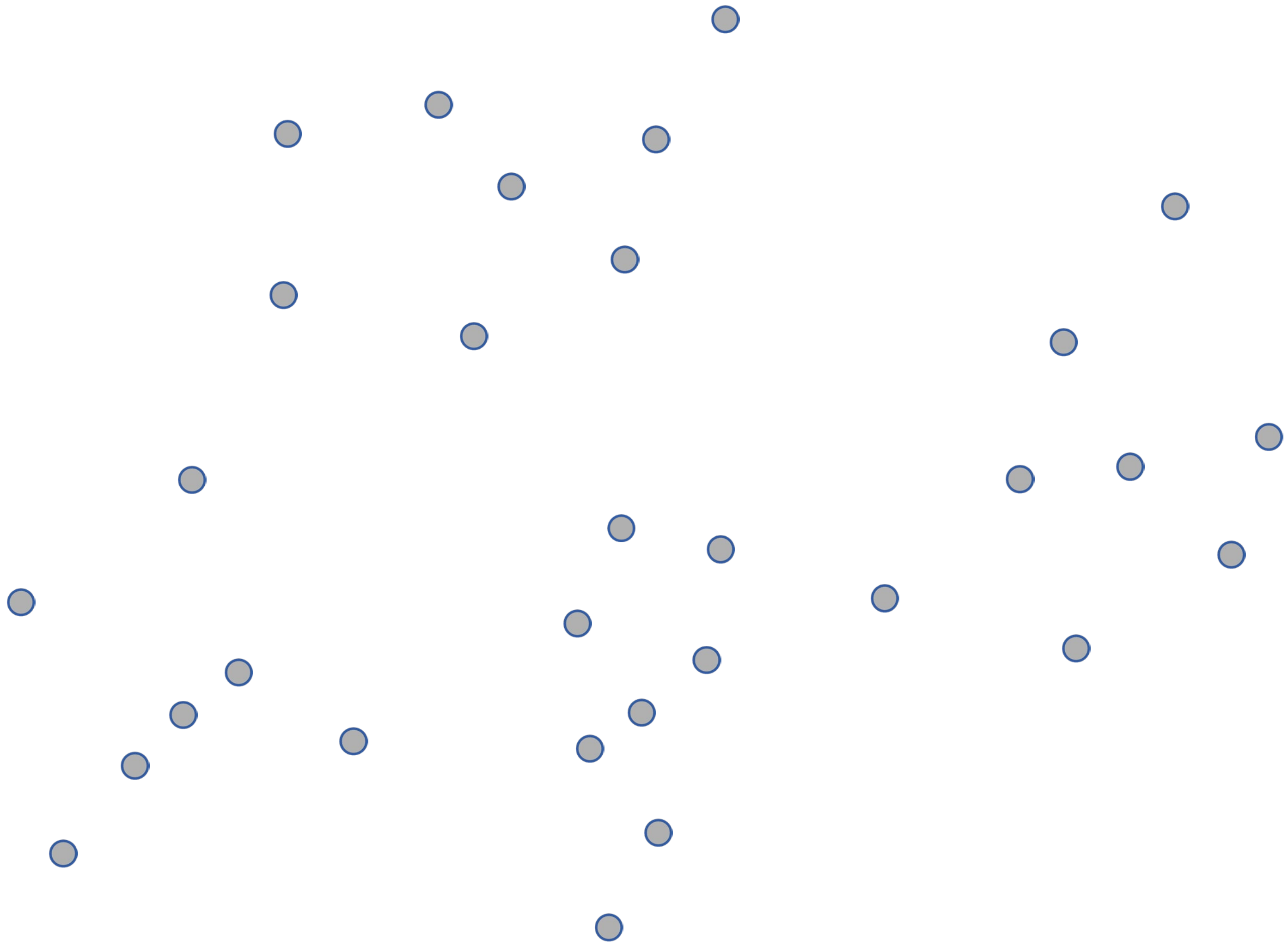
if $i \leq k + 1$

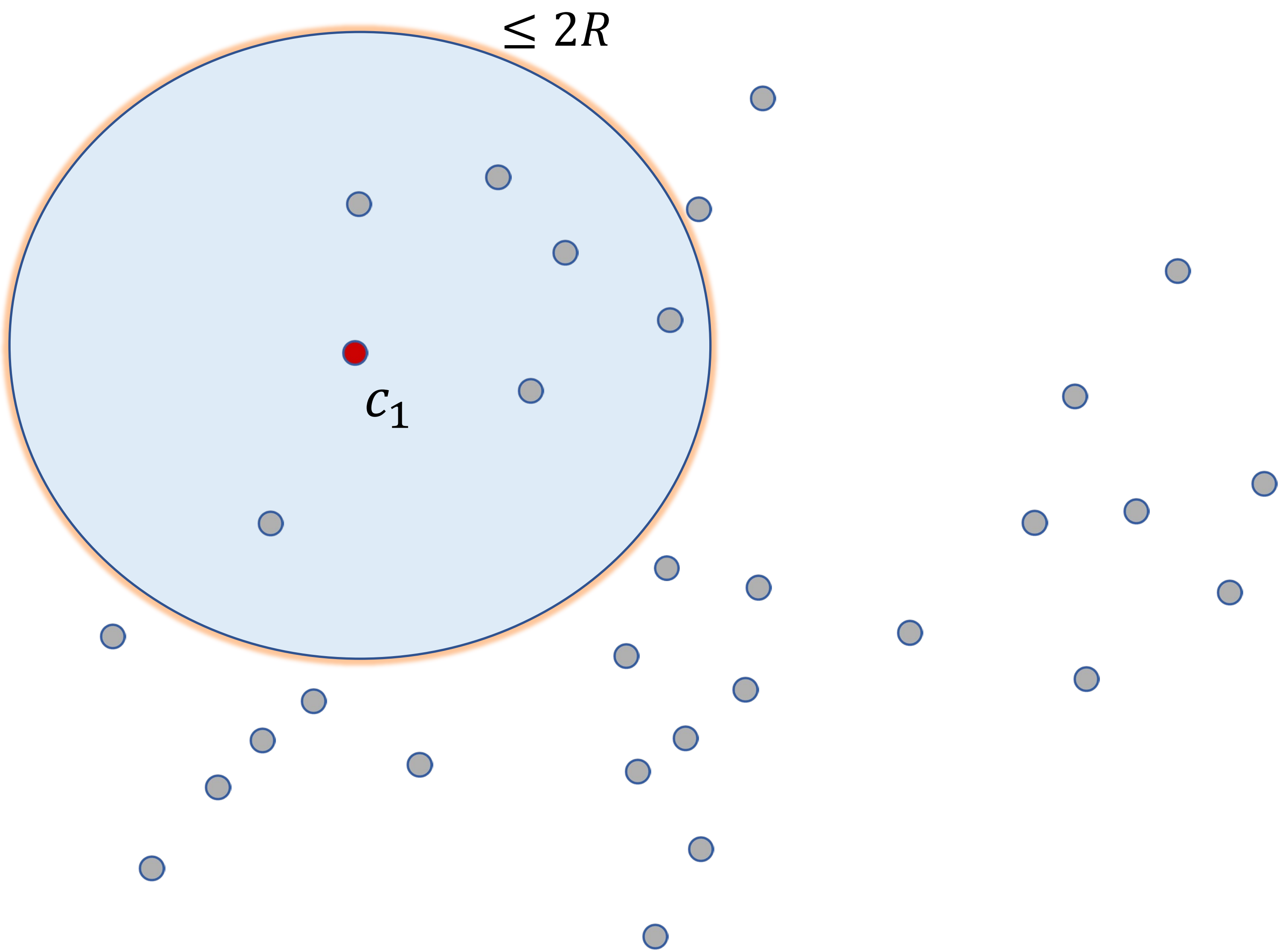
    output centers $c_1, \dots, c_i$

else

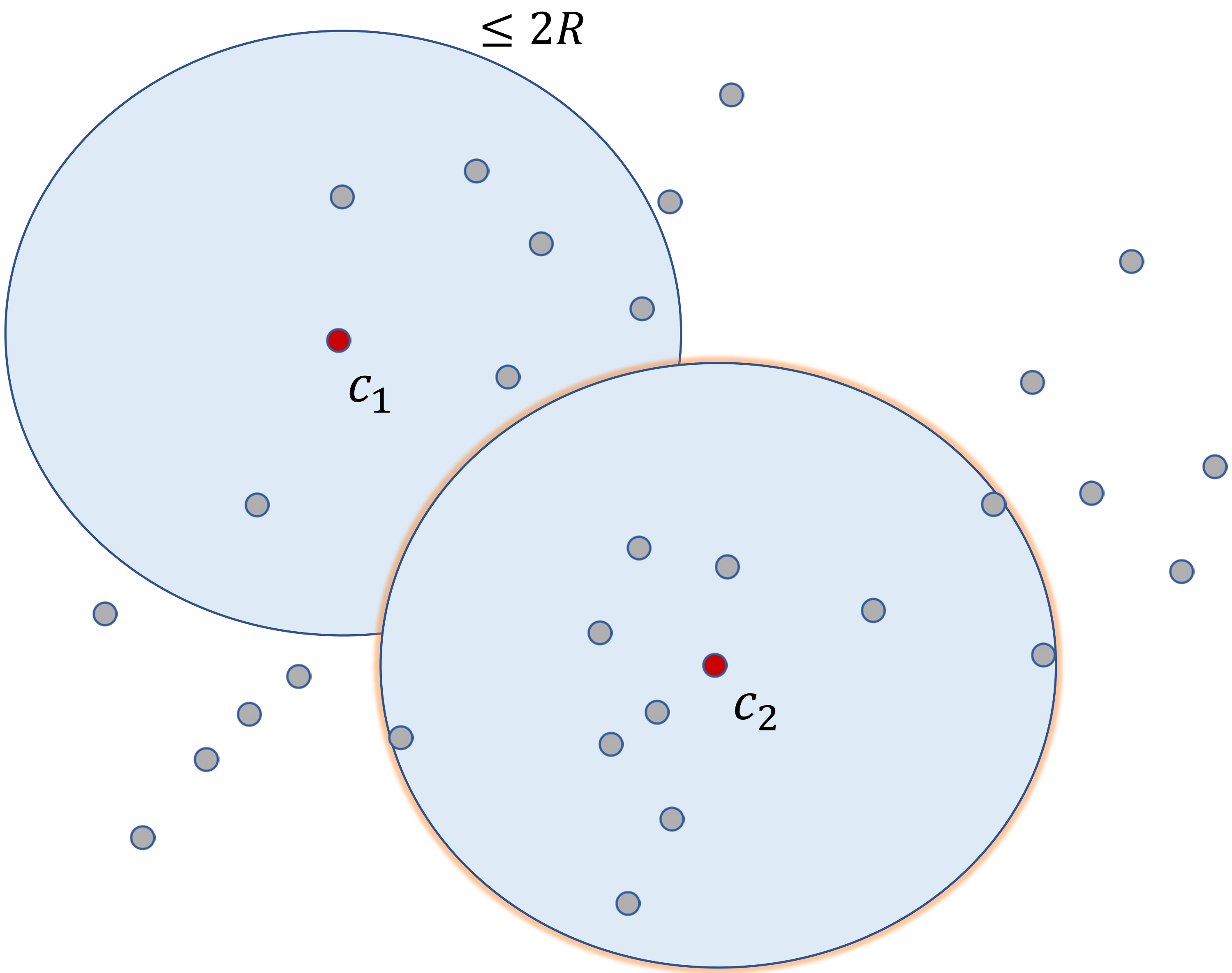
    report that there is no solution of cost $\leq R$

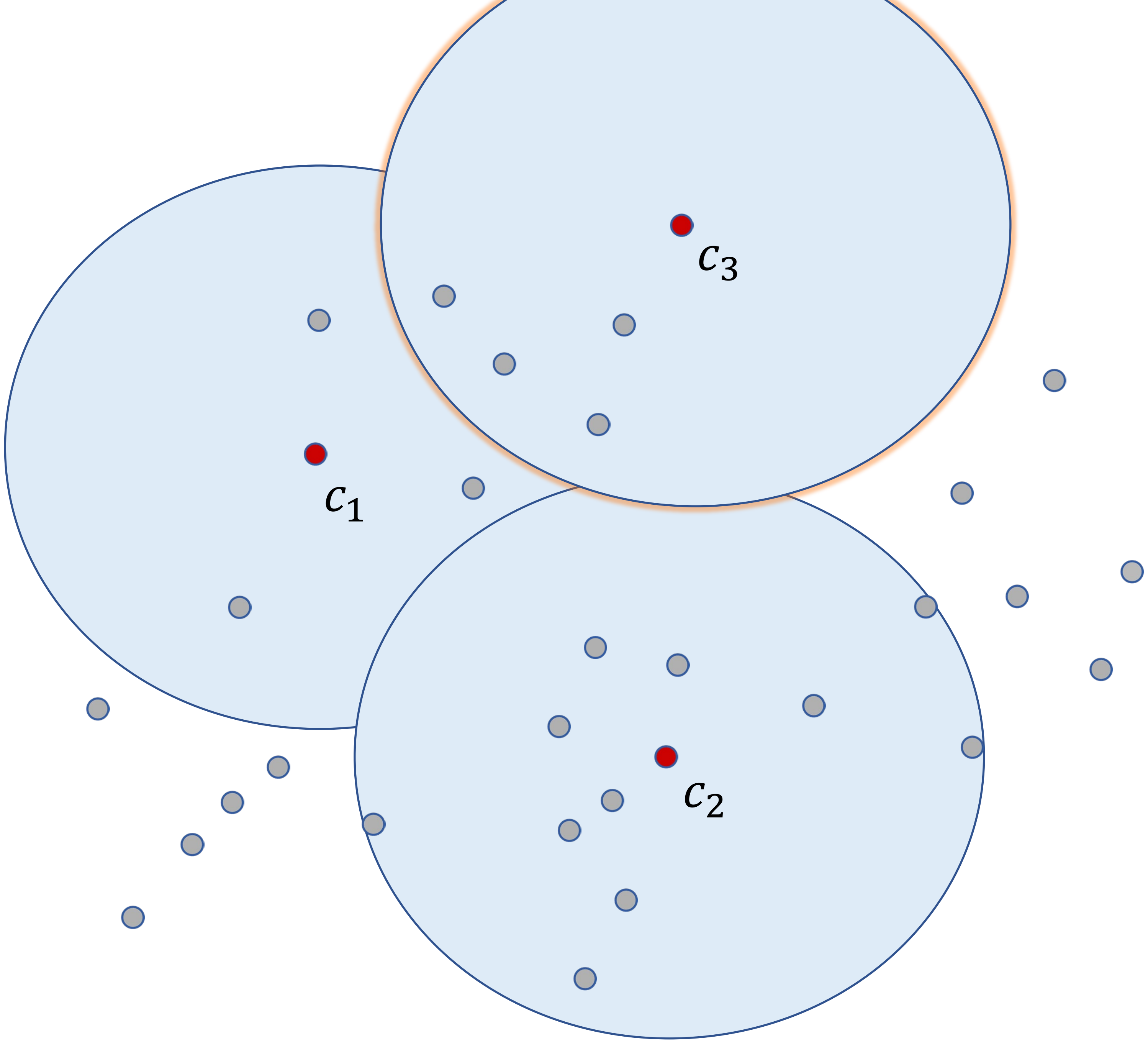
Algorithm

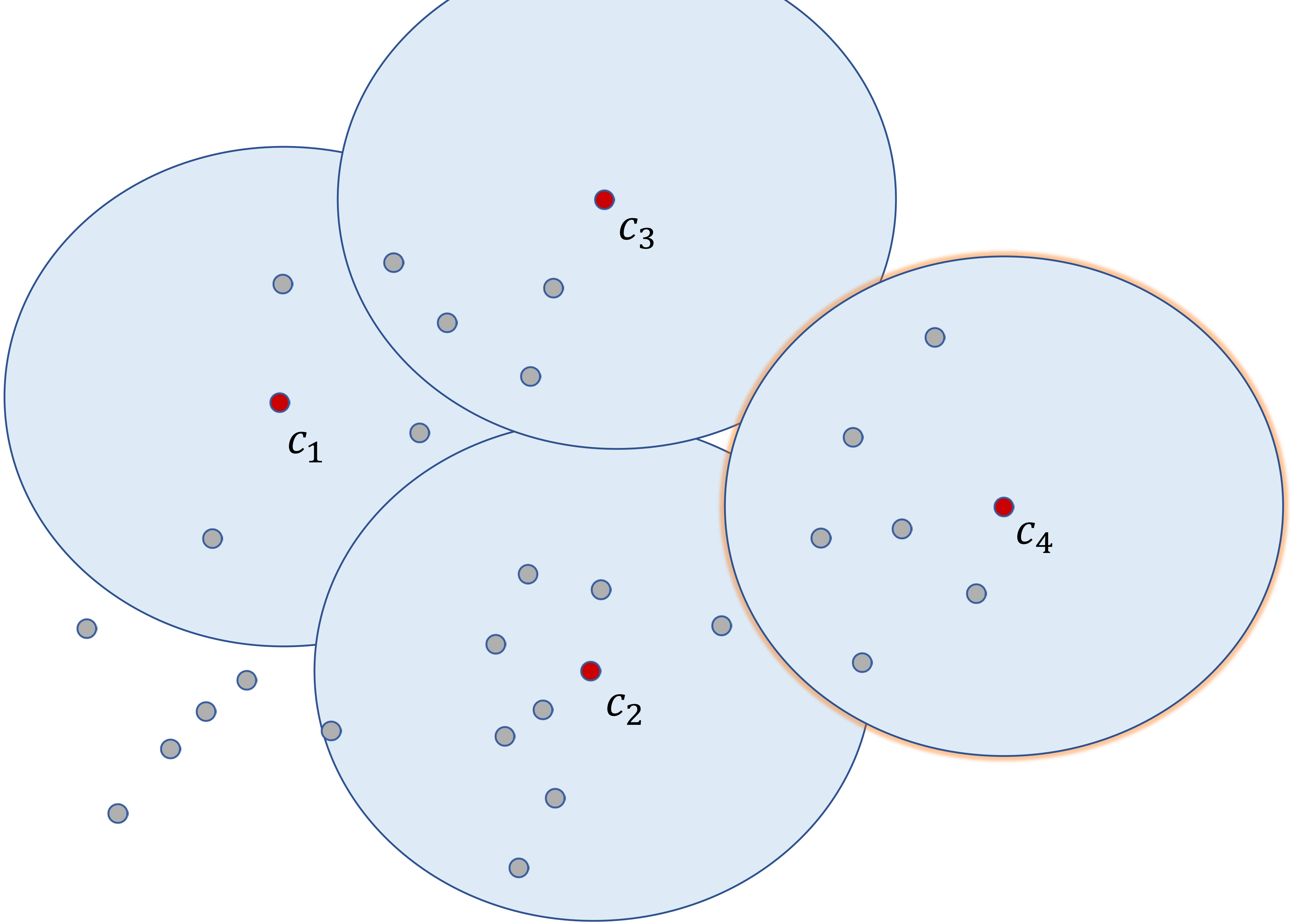




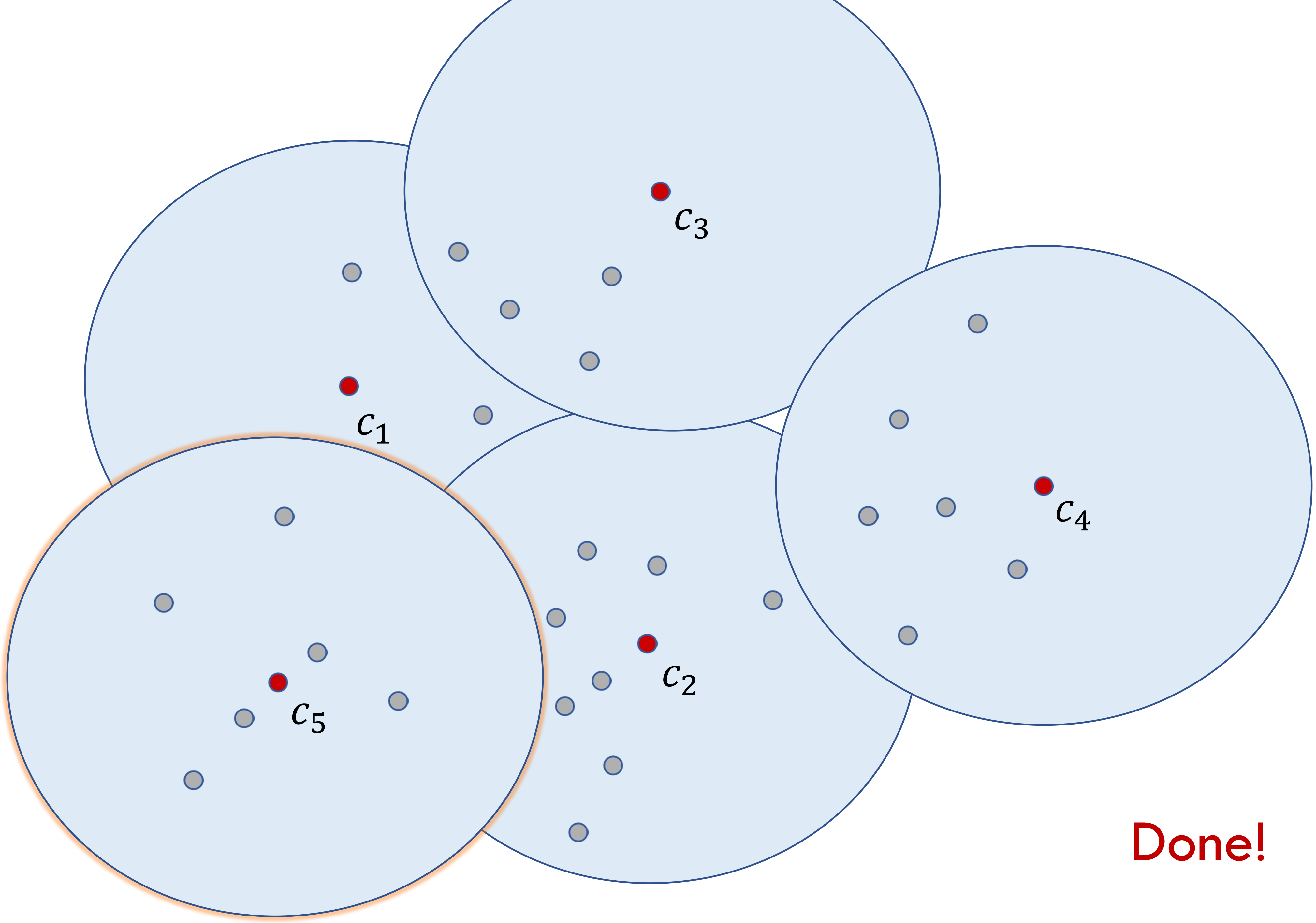
$\leq 2R$
c_1


$\leq 2R$
c_1
c_2


c_3
c_1
c_2



c_3
c_1
c_4
c_2



Done!

Metric k-center: Greedy Algorithm

If our algorithm outputs k or fewer centers, then all points $x \in X$ are clustered.

That is, for every $x \in X$, there is some iteration j when it gets clustered:

$$\Rightarrow \min_i d(x, c_i) \leq d(x, c_j) \leq 2R, \text{ as required.}$$

It remains to show that if the optimal clustering has cost $\leq R$,
then our algorithm finds at most k centers.

Metric k-center: Greedy Algorithm

If our algorithm outputs k or fewer centers, then all points $x \in X$ are clustered.

That is, for every $x \in X$, there is some iteration j when it gets clustered:

$$\Rightarrow \min_i d(x, c_i) \le d(x, c_j) \le 2R, \text{ as required.}$$

It remains to show that if the optimal clustering has cost $\le R$,
then our algorithm finds at most k centers.

Observation: $d(c_i, c_j) > 2R$. Why?

Decision Problem ⇒ Optimization Problem

If we know the cost of the optimal solution R, we can find a solution of cost at most $2R$.

But if we don't…

Suggestions?