Technische Universität Dresden

International Center for Computational Logic

COMPLEXITY THEORY

Lecture 27: Constant-Factor Approximations

Sergei Obiedkov

Knowledge-Based Systems

TU Dresden, 26 Jan 2026

More recent versions of this slide deck might be available.
For the most current version of this course, see
https://iccl.inf.tu-dresden.de/web/Complexity_Theory/en

How to Solve NP-hard Problems?

- An exponentially large search space.

How to Solve NP-hard Problems?

- An exponentially large search space.
- No known polynomial-time algorithm to explore it.

How to Solve NP-hard Problems?

- An exponentially large search space.
- No known polynomial-time algorithm to explore it.
- This doesn't always mean that you have to use exhaustive search.

How to Solve NP-hard Problems?

- An exponentially large search space.
- No known polynomial-time algorithm to explore it.
- This doesn't always mean that you have to use exhaustive search.
- Much more efficient albeit exponential-time algorithms may be possible.

How to Solve NP-hard Problems?

- An exponentially large search space.
- No known polynomial-time algorithm to explore it.
- This doesn't always mean that you have to use exhaustive search.
- Much more efficient albeit exponential-time algorithms may be possible.
- Polynomial-time algorithms for important special cases may exist.

How to Solve NP-hard Problems?

- An exponentially large search space.
- No known polynomial-time algorithm to explore it.
- This doesn't always mean that you have to use exhaustive search.
- Much more efficient albeit exponential-time algorithms may be possible.
- Polynomial-time algorithms for important special cases may exist.
- An exact solution is not always needed: an approximate solution may be sufficient.

Traveling Salesman Problem

MINIMAL TSP

Input: A complete undirected graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_+$

Problem: Find a minimum-weight Hamiltonian cycle in G

Traveling Salesman Problem

Minimal TSP

Input: A complete undirected graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_+$

Problem: Find a minimum-weight Hamiltonian cycle in G

Relatively Light TSP

Input: A complete undirected graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_+$

Problem: Find a relatively light Hamiltonian cycle in G

- w.r.t. the minimum weight

Approximation Algorithms

Approximation algorithms are used if

- A suboptimal solution is OK
- It is possible to quickly get a good approximation of an optimal solution
- There is no known way to efficiently find an optimal solution

Approximation Factor

Definition 27.1: Constant-factor approximation $c > 1$

In minimization problems: the solution cost $\leq c \cdot$ the optimal solution cost

In maximization problems: the solution cost $\geq \frac{1}{c} \cdot$ the optimal solution cost

Vertex Cover

Input: A graph $G = (V, E)$

Problem: Find a minimum vertex cover of G

Example 27.2: A 2-approximate algorithm found a vertex cover of size 50

$$? \leq \text{the minimum vertex cover size} \leq ?$$

Approximation Factor

Definition 27.1: Constant-factor approximation $c > 1$

In minimization problems: the solution cost $\leq c \cdot$ the optimal solution cost

In maximization problems: the solution cost $\geq \frac{1}{c} \cdot$ the optimal solution cost

Vertex Cover

Input: A graph $G = (V, E)$

Problem: Find a minimum vertex cover of G

Example 27.2: A 2-approximate algorithm found a vertex cover of size 50

$$25 \leq \text{the minimum vertex cover size} \leq 50$$

Approximation Factor

Definition 27.3: Constant-factor approximation $c > 1$

In minimization problems: the solution cost $\leq c \cdot$ the optimal solution cost

In maximization problems: the solution cost $\geq \frac{1}{c} \cdot$ the optimal solution cost

CLIQUE

Input: A graph $G = (V, E)$

Problem: Find a maximum clique in G

Example 27.4: A 2-approximate algorithm found a clique of size 50

$$? \leq \text{the maximum clique size} \leq ?$$

Approximation Factor

Definition 27.3: Constant-factor approximation $c > 1$

In minimization problems: the solution cost $\leq c \cdot$ the optimal solution cost

In maximization problems: the solution cost $\geq \frac{1}{c} \cdot$ the optimal solution cost

CLIQUE

Input: A graph $G = (V, E)$

Problem: Find a maximum clique in G

Example 27.4: A 2-approximate algorithm found a clique of size 50

$$50 \leq \text{the maximum clique size} \leq 100$$

Vertex Cover

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Second algorithm:

while there is an uncovered edge

add a vertex incident to as many uncovered edges as possible to the vertex cover

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Let M be a minimum vertex cover and A be a cover found by the algorithm.

- How many vertices of M does the algorithm put into A at every step?

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Let M be a minimum vertex cover and A be a cover found by the algorithm.

- How many vertices of M does the algorithm put into A at every step?
  - One or two, since at least one vertex of every edge is in M.

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Let M be a minimum vertex cover and A be a cover found by the algorithm.

- How many vertices of M does the algorithm put into A at every step?
  - One or two, since at least one vertex of every edge is in M.
- The number of iterations

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Let M be a minimum vertex cover and A be a cover found by the algorithm.

- How many vertices of M does the algorithm put into A at every step?
  - One or two, since at least one vertex of every edge is in M.
- The number of iterations $\leq |M| < |V|$.

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Let M be a minimum vertex cover and A be a cover found by the algorithm.

- How many vertices of M does the algorithm put into A at every step?
  - One or two, since at least one vertex of every edge is in M.
- The number of iterations $\leq |M| < |V|$.
- Can we bound $|A|$ in terms of $|M|$?

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Let M be a minimum vertex cover and A be a cover found by the algorithm.

- How many vertices of M does the algorithm put into A at every step?
  - One or two, since at least one vertex of every edge is in M.
- The number of iterations $\leq |M| < |V|$.
- Can we bound $|A|$ in terms of $|M|$?

$$|M| \leq |A| \leq 2|M|$$

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Let M be a minimum vertex cover and A be a cover found by the algorithm.

- How many vertices of M does the algorithm put into A at every step?
  - One or two, since at least one vertex of every edge is in M.
- The number of iterations $\leq |M| < |V|$.
- Can we bound $|A|$ in terms of $|M|$?

$$|M| \leq |A| \leq 2|M|$$

This is a 2-approximate algorithm

Vertex Cover

Second (greedy) algorithm:

while there is an uncovered edge

add a vertex incident to as many uncovered edges as possible to the vertex cover

Vertex Cover

Second (greedy) algorithm:

while there is an uncovered edge

add a vertex incident to as many uncovered edges as possible to the vertex cover

Not a constant-factor approximation algorithm.

Example 27.5: See the blackboard.

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Second algorithm:

while there is an uncovered edge

add a vertex incident to as many uncovered edges as possible to the vertex cover

- Which algorithm should one use?

Vertex Cover

First algorithm:

while there is an uncovered edge (u, v)

add u and v to the vertex cover

Second algorithm:

while there is an uncovered edge

add a vertex incident to as many uncovered edges as possible to the vertex cover

- Which algorithm should one use?
- Perhaps, use both and choose the better solution.

Weighted Vertex Cover

Weighted Vertex Cover

Input: A graph $G = (V, E)$ with weights $w\colon V \to \mathbb{R}_+$

Problem: Find a vertex cover of G with minimum weight

Weighted Vertex Cover

Weighted Vertex Cover

Input: A graph $G = (V, E)$ with weights $w\colon V \to \mathbb{R}_+$

Problem: Find a vertex cover of G with minimum weight

- Assign a price $p(e) \geq 0$ to each edge $e \in E$.
- Make sure that the following invariant holds:

$$\forall v \in V\colon \sum_{(v,u)\in E} p(v,u) \leq w(v)$$

Weighted Vertex Cover

Weighted Vertex Cover

Input: A graph $G = (V, E)$ with weights $w\colon V \to \mathbb{R}_+$

Problem: Find a vertex cover of G with minimum weight

- Assign a price $p(e) \geq 0$ to each edge $e \in E$.
- Make sure that the following invariant holds:

$$\forall v \in V\colon \sum_{(v,u)\in E} p(v,u) \leq w(v)$$

- Then for every vertex cover C:

$$\sum_{e\in E} p(e) \leq \sum_{v\in C} \sum_{(v,u)\in E} p(v,u) \leq \sum_{v\in C} w(v) = w(C)$$

Approximation Algorithm

A vertex v is tight if

$$\sum_{(v,u)\in E} p(v,u) = w(v).$$

- let all $p(e) = 0$

Approximation Algorithm

A vertex v is tight if

$$\sum_{(v,u)\in E} p(v,u) = w(v).$$

- let all $p(e) = 0$
- while neither u nor v is tight for some $(v,u) \in E$

Approximation Algorithm

A vertex v is tight if

$$\sum_{(v,u)\in E} p(v,u) = w(v).$$

- let all $p(e) = 0$
- while neither u nor v is tight for some $(v,u) \in E$
  - increase $p(v,u)$ as much as possible while maintaining the invariant

Approximation Algorithm

A vertex v is tight if

$$\sum_{(v,u)\in E} p(v,u) = w(v).$$

- let all $p(e) = 0$
- while neither u nor v is tight for some $(v,u) \in E$
  - increase $p(v,u)$ as much as possible while maintaining the invariant
- return the set of tight vertices

Approximation Factor

C is a minimum cover

D is the cover found by the algorithm

$$w(D) = \sum_{v \in D} w(v) = {}^{1} \sum_{v \in D} \sum_{(v,u) \in E} p(v,u) \leq$$

$$\leq \sum_{v \in V} \sum_{(v,u) \in E} p(v,u) = 2 \sum_{e \in E} p(e) \leq 2w(C)$$

[1] since all vertices in D are tight

Traveling Salesman Problem

Traveling Salesman Problem

Traveling Salesman Problem

Input: A complete undirected graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_+$

Problem: Find a minimum-weight Hamiltonian cycle in G

- The corresponding decision problem is NP-complete.
- A possible solution is a permutation of the vertices of V.
- Exhaustive search: $n!$ possible solutions ($n = |V|$).
- Can do faster: a dynamic-programming solution runs in time $O(n^2 2^n)$.

Metric Travelling Salesman Problem

Metric Traveling Salesman Problem

Input: A complete undirected graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_+$, where

$$\forall x, y \in V\colon w(x, y) + w(y, z) \geq w(x, z)$$

Problem: Find a minimum-weight Hamiltonian cycle in G

Metric Travelling Salesman Problem

METRIC TRAVELING SALESMAN PROBLEM

Input: A complete undirected graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_+$, where

$$\forall x, y \in V\colon w(x, y) + w(y, z) \geq w(x, z)$$

Problem: Find a minimum-weight Hamiltonian cycle in G

- The corresponding decision problem is still NP-complete.

Metric Travelling Salesman Problem

METRIC TRAVELING SALESMAN PROBLEM

Input: A complete undirected graph $G = (V, E)$ with weights $w\colon E \to \mathbb{R}_+$, where

$$\forall x, y \in V\colon w(x, y) + w(y, z) \geq w(x, z)$$

Problem: Find a minimum-weight Hamiltonian cycle in G

- The corresponding decision problem is still NP-complete.
- There is a polynomial-time 2-approximation algorithm.

Minimum Spanning Tree

- The weight of a minimum spanning tree $\leq$ the weight of a minimum Hamiltonian cycle

Minimum Spanning Tree

- The weight of a minimum spanning tree $\leq$ the weight of a minimum Hamiltonian cycle
  - A Hamiltonian cycle without one edge is a spanning tree

Minimum Spanning Tree

- The weight of a minimum spanning tree $\leq$ the weight of a minimum Hamiltonian cycle
  - A Hamiltonian cycle without one edge is a spanning tree

An algorithm for TSP

- Build a minimum spanning tree.

Minimum Spanning Tree

- The weight of a minimum spanning tree $\leq$ the weight of a minimum Hamiltonian cycle
  - A Hamiltonian cycle without one edge is a spanning tree

An algorithm for TSP

- Build a minimum spanning tree.
- Replace every undirected edge by two directed edges.

Minimum Spanning Tree

- The weight of a minimum spanning tree $\leq$ the weight of a minimum Hamiltonian cycle
  - A Hamiltonian cycle without one edge is a spanning tree

An algorithm for TSP

- Build a minimum spanning tree.
- Replace every undirected edge by two directed edges.
- Find an Eulerian cycle in the resulting “tree”.

Minimum Spanning Tree

- The weight of a minimum spanning tree $\leq$ the weight of a minimum Hamiltonian cycle
  - A Hamiltonian cycle without one edge is a spanning tree

An algorithm for TSP

- Build a minimum spanning tree.
- Replace every undirected edge by two directed edges.
- Find an Eulerian cycle in the resulting “tree”.
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

Approximation Quality

The weight of the resulting cycle is at most twice the weight of the minimum-weight Hamiltonian cycle.

Approximation Quality

The weight of the resulting cycle is at most twice the weight of the minimum-weight Hamiltonian cycle.

- T is a minimum spanning tree
- C is a minimum-weight Hamiltonian cycle
- D is the cycle found by the algorithm

Approximation Quality

The weight of the resulting cycle is at most twice the weight of the minimum-weight Hamiltonian cycle.

T is a minimum spanning tree

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$ since C is a spanning tree plus one edge

Approximation Quality

The weight of the resulting cycle is at most twice the weight of the minimum-weight Hamiltonian cycle.

T is a minimum spanning tree

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$ since C is a spanning tree plus one edge
- $w(D) \leq 2w(T)$ due to triangle inequality

Approximation Quality

The weight of the resulting cycle is at most twice the weight of the minimum-weight Hamiltonian cycle.

T is a minimum spanning tree

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$ since C is a spanning tree plus one edge
- $w(D) \leq 2w(T)$ due to triangle inequality

$\Rightarrow$ $w(D) \leq 2w(C)$ We have a 2-approximation.

Better Approximation

An algorithm for TSP

- Build a minimum spanning tree (MST).
- Replace every undirected edge by two directed edges.
- Find an Eulerian cycle in the resulting “tree”.
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

Better Approximation

An algorithm for TSP

- Build a minimum spanning tree (MST).
- Replace every undirected edge by two directed edges.
- Find an Eulerian cycle in the resulting “tree”.
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

Can we use a shorter Eulerian cycle?

Better Approximation

An algorithm for TSP

- Build a minimum spanning tree (MST).
- Replace every undirected edge by two directed edges.
- Find an Eulerian cycle in the resulting “tree”.
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

Can we use a shorter Eulerian cycle?

- A graph is Eulerian if and only if every its vertex has an even degree.
- To transform the MST into an Eulerian graph, it suffices to add one edge to each node of odd degree.
- There are an even number of odd-degree vertices (by Handshaking Lemma).

Better Approximation

An algorithm for TSP

- Build a minimum spanning tree (MST).
- Replace every undirected edge by two directed edges.
- Find an Eulerian cycle in the resulting “tree”.
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

Can we use a shorter Eulerian cycle?

- A graph is Eulerian if and only if every its vertex has an even degree.
- To transform the MST into an Eulerian graph, it suffices to add one edge to each node of odd degree.
- There are an even number of odd-degree vertices (by Handshaking Lemma).

$\Rightarrow$ Use a perfect matching in the subgraph induced by the odd-degree vertices.

Better Approximation

Christofides' Algorithm

- Build a minimum spanning tree T.
- Compute a minimum perfect matching M in the subgraph of G induced by vertices that have odd degree in T. can be done in polynomial time
- Find an Eulerian cycle in $T \cup M$. can be done in polynomial time
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

Better Approximation

Christofides' Algorithm

- Build a minimum spanning tree T.
- Compute a minimum perfect matching M in the subgraph of G induced by vertices that have odd degree in T. can be done in polynomial time
- Find an Eulerian cycle in $T \cup M$. can be done in polynomial time
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$

Better Approximation

Christofides' Algorithm

- Build a minimum spanning tree T.
- Compute a minimum perfect matching M in the subgraph of G induced by vertices that have odd degree in T. can be done in polynomial time
- Find an Eulerian cycle in $T \cup M$. can be done in polynomial time
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$
- $w(D) \leq w(T) + w(M)$

Better Approximation

Christofides' Algorithm

- Build a minimum spanning tree T.
- Compute a minimum perfect matching M in the subgraph of G induced by vertices that have odd degree in T. can be done in polynomial time
- Find an Eulerian cycle in $T \cup M$. can be done in polynomial time
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$
- $w(D) \leq w(T) + w(M)$
- Remove even-degree vertices from C to obtain a cycle C' on odd-degree vertices.

Better Approximation

Christofides' Algorithm

- Build a minimum spanning tree T.
- Compute a minimum perfect matching M in the subgraph of G induced by vertices that have odd degree in T. can be done in polynomial time
- Find an Eulerian cycle in $T \cup M$. can be done in polynomial time
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$
- $w(D) \leq w(T) + w(M)$
- Remove even-degree vertices from C to obtain a cycle C' on odd-degree vertices.
- C' consists of two disjoint perfect matchings for odd-degree vertices.

Better Approximation

Christofides' Algorithm

- Build a minimum spanning tree T.
- Compute a minimum perfect matching M in the subgraph of G induced by vertices that have odd degree in T. can be done in polynomial time
- Find an Eulerian cycle in $T \cup M$. can be done in polynomial time
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$
- $w(D) \leq w(T) + w(M)$
- Remove even-degree vertices from C to obtain a cycle C' on odd-degree vertices.
- C' consists of two disjoint perfect matchings for odd-degree vertices.

$\Rightarrow \; w(M) \leq \frac{1}{2} w(C') \leq \frac{1}{2} w(C)$

Better Approximation

Christofides' Algorithm

- Build a minimum spanning tree T.
- Compute a minimum perfect matching M in the subgraph of G induced by vertices that have odd degree in T. can be done in polynomial time
- Find an Eulerian cycle in $T \cup M$. can be done in polynomial time
- Delete repeated vertices to obtain a Hamiltonian cycle in the original graph.

C is a minimum-weight Hamiltonian cycle

D is the cycle found by the algorithm

- $w(T) \leq w(C)$
- $w(D) \leq w(T) + w(M)$
- Remove even-degree vertices from C to obtain a cycle C' on odd-degree vertices.
- C' consists of two disjoint perfect matchings for odd-degree vertices.

$\Rightarrow\ w(M) \leq \frac{1}{2}w(C') \leq \frac{1}{2}w(C) \quad \Rightarrow \quad w(D) \leq \frac{3}{2}w(C)$ We have a 3/2-approximation.

Reduction and Approximation

Reduction and Approximation

C is a vertex cover in graph (V, E)

$\Updownarrow$

$V \setminus C$ is an independent set in graph (V, E)

$\Updownarrow$

$V \setminus C$ is a clique in graph $(V, V^2 \setminus E)$

Reduction and Approximation

C is a vertex cover in graph (V, E)

$\Updownarrow$

$V \setminus C$ is an independent set in graph (V, E)

$\Updownarrow$

$V \setminus C$ is a clique in graph $(V, V^2 \setminus E)$

$\Downarrow ?$

An approximate algorithm for **Vertex Cover**

$\Updownarrow$

An approximate algorithm for **Independent Set**

$\Updownarrow$

An approximate algorithm for **Clique**

Reduction and Approximation

- Suppose that the size of a minimum vertex cover = k

$\Rightarrow$ The size of a maximum independent set =

Reduction and Approximation

- Suppose that the size of a minimum vertex cover = k

$\Rightarrow$ The size of a maximum independent set = $n - k$

- The size a “2-approximate” vertex cover $\leq 2k$

$\Rightarrow$ The size of the corresponding independent set $\geq n - 2k$

Reduction and Approximation

- Suppose that the size of a minimum vertex cover = k

$\Rightarrow$ The size of a maximum independent set = $n - k$

- The size a "2-approximate" vertex cover $\leq 2k$

$\Rightarrow$ The size of the corresponding independent set $\geq n - 2k$

- The approximation factor (for $k < n/2$):

$$\leq \frac{n-k}{n-2k} = 1 + \frac{k}{n-2k}$$

Reduction and Approximation

- Suppose that the size of a minimum vertex cover = k

$\Rightarrow$ The size of a maximum independent set = $n - k$

- The size a "2-approximate" vertex cover $\leq 2k$

$\Rightarrow$ The size of the corresponding independent set $\geq n - 2k$

- The approximation factor (for $k < n/2$):

$$\leq \frac{n-k}{n-2k} = 1 + \frac{k}{n-2k}$$

Not very useful, since we don't know k (completely useless if $k \geq n/2$).

Reduction and Approximation

- Suppose that the size of a minimum vertex cover = k

$\Rightarrow$ The size of a maximum independent set = $n - k$

- The size a "2-approximate" vertex cover $\leq 2k$

$\Rightarrow$ The size of the corresponding independent set $\geq n - 2k$

- The approximation factor (for $k < n/2$):

$$\leq \frac{n-k}{n-2k} = 1 + \frac{k}{n-2k}$$

Not very useful, since we don't know k (completely useless if $k \geq n/2$).

The approximation factor is not preserved under such reductions.

Reduction and Approximation

If P ≠ NP,

- there are no polynomial-time constant-factor approximation algorithms for **Clique** and **Independent Set**;
- there is no polynomial-time $(\sqrt{2} - \varepsilon)$-approximate algorithm for **Vertex Cover** for any $\varepsilon > 0$.

Clustering: Complexity and Algorithms

Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ satisfying some properties

Clustering

CLUSTERING

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ satisfying some properties

- $dist(x, y) \geq 0$

Clustering

CLUSTERING

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ satisfying some properties

- $dist(x, y) \geq 0$
- $dist(x, x) = 0$

Clustering

CLUSTERING

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ satisfying some properties

- $dist(x, y) \geq 0$
- $dist(x, x) = 0$
- $dist(x, y) = dist(y, x)$

Clustering

CLUSTERING

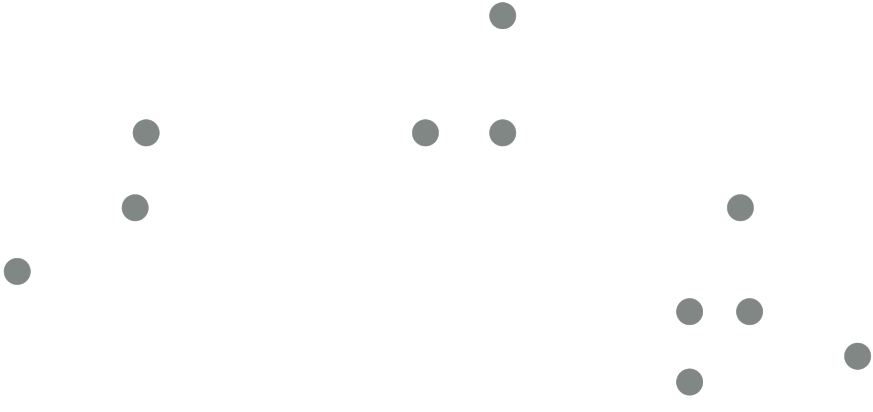
Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ satisfying some properties

- $dist(x, y) \geq 0$
- $dist(x, x) = 0$
- $dist(x, y) = dist(y, x)$
- $dist(x, y) \leq dist(x, z) + dist(z, y)$

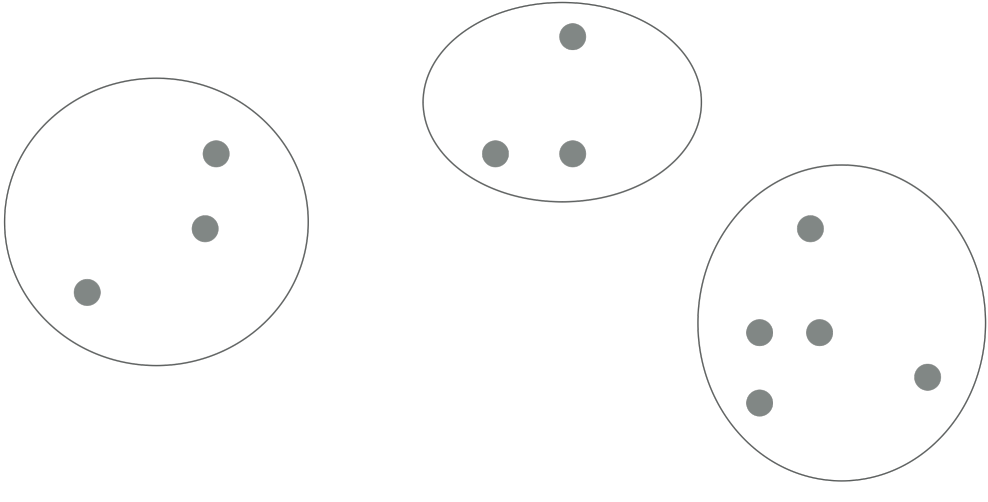
Clustering

$k = 3$



Clustering

$k = 3$



Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

- $dist(x, y) \geq 0$
- $dist(x, x) = 0$
- $dist(x, y) = dist(y, x)$
- $dist(x, y) \leq dist(x, z) + dist(z, y)$

Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

- $dist(x, y) \geq 0$
- $dist(x, x) = 0$
- $dist(x, y) = dist(y, x)$
- $dist(x, y) \leq dist(x, z) + dist(z, y)$

Diameter of C_i

$d(C_i) = \max_{x \in C_i, y \in C_i} dist(x, y)$

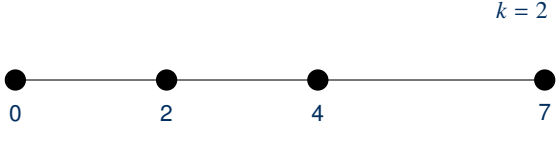
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



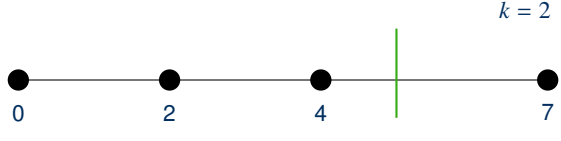
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



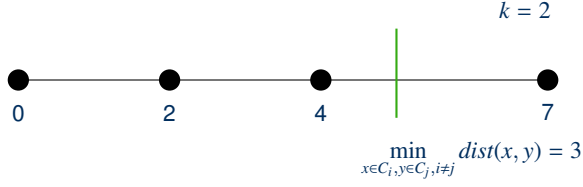
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min\limits_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max\limits_{x \in C_i, y \in C_i} dist(x, y)$



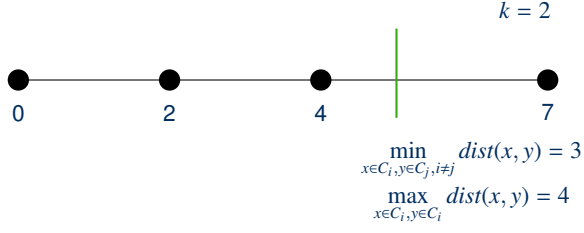
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min\limits_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max\limits_{x \in C_i, y \in C_i} dist(x, y)$



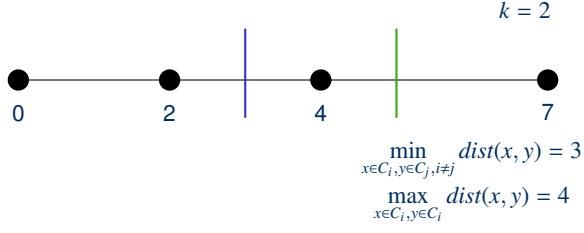
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min\limits_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max\limits_{x \in C_i, y \in C_i} dist(x, y)$



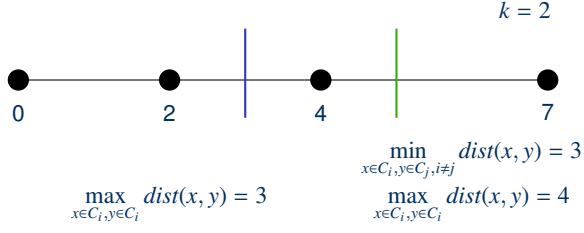
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min\limits_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max\limits_{x \in C_i, y \in C_i} dist(x, y)$



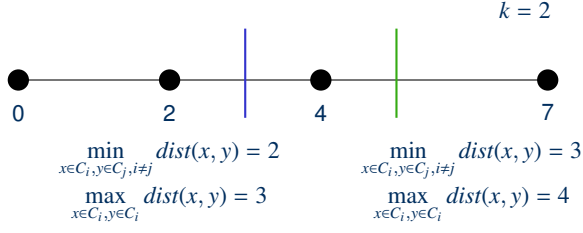
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



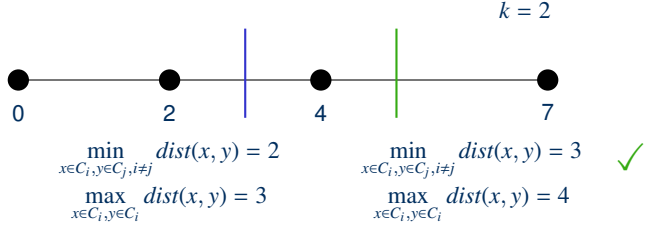
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max_{x, y \in C_i} dist(x, y)$



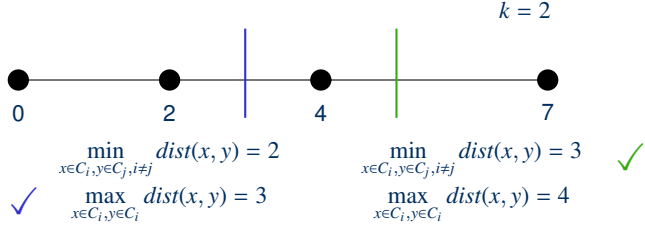
Clustering

CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that
maximizes $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$ or minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



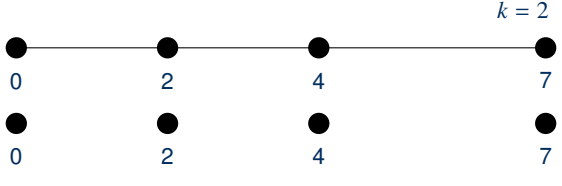
Maximizing inter-cluster distance

MAX-SPACING CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that maximises $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$



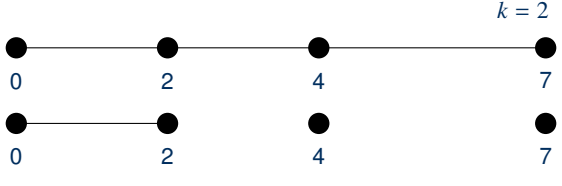
Maximizing inter-cluster distance

MAX-SPACING CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that maximises $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$



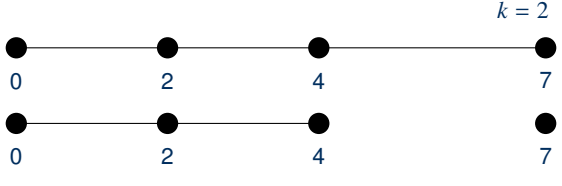
Maximizing inter-cluster distance

MAX-SPACING CLUSTERING

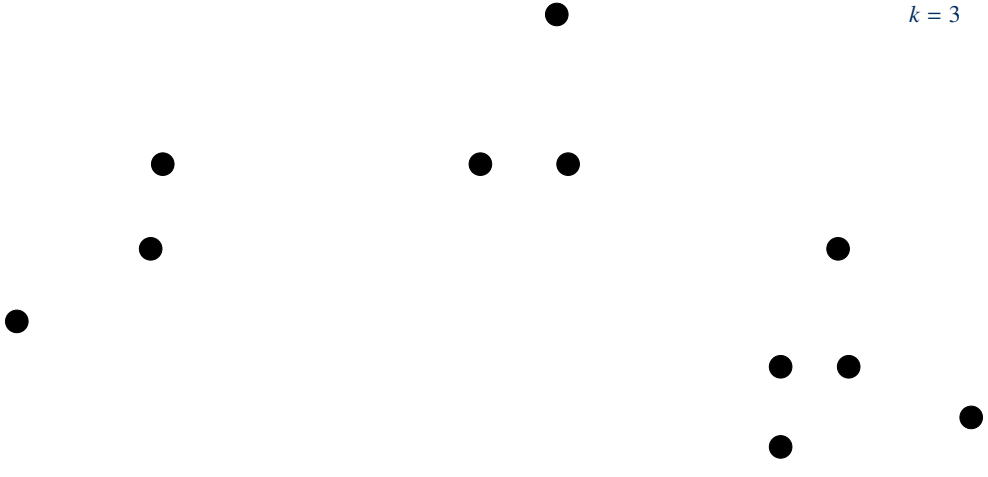
Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

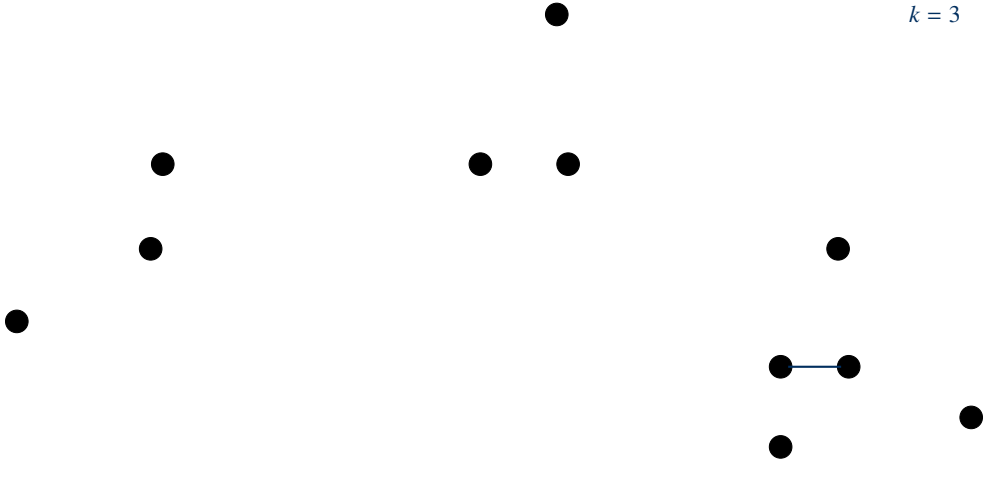
Problem: Find a partition of P into $C_1, \ldots, C_k$ that maximises $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$



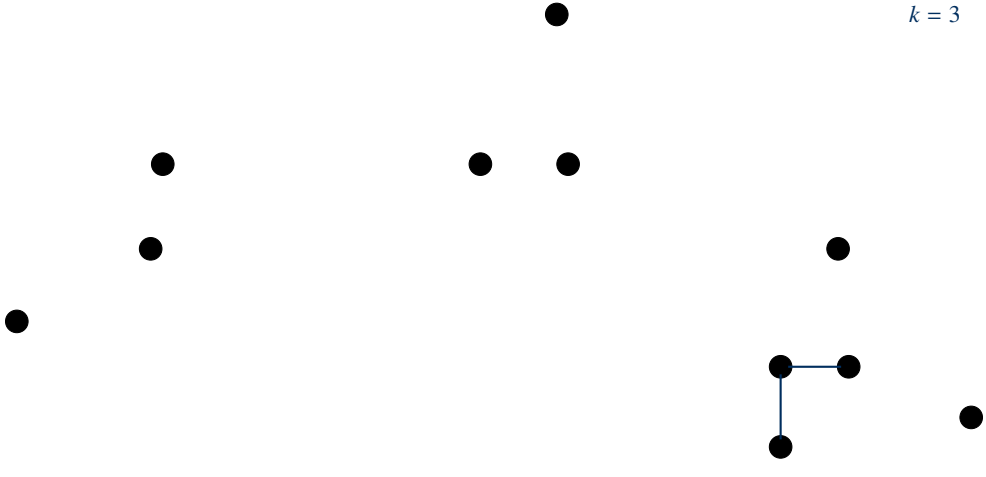
Maximizing inter-cluster distance



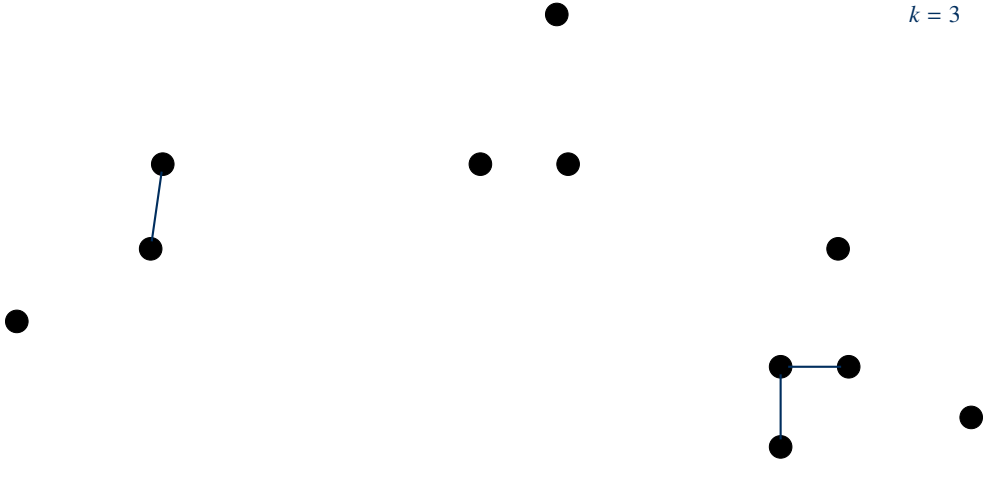
Maximizing inter-cluster distance



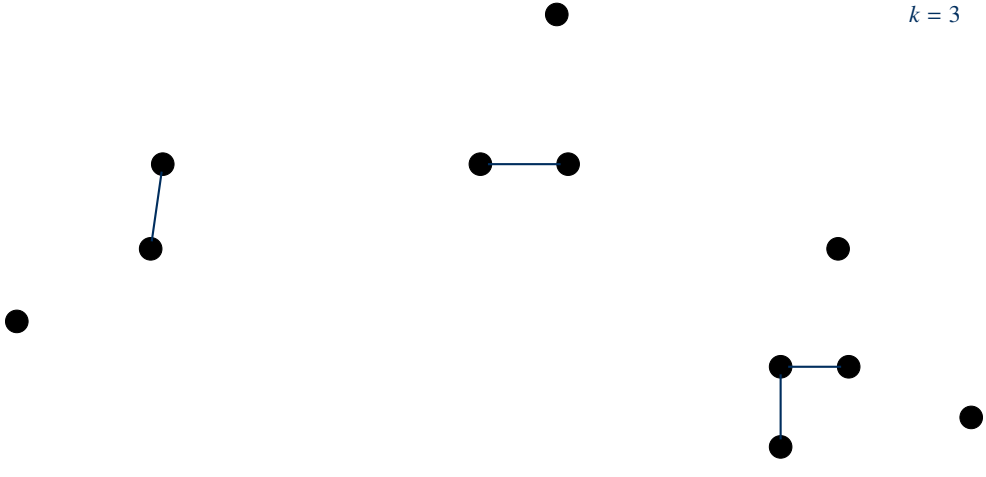
Maximizing inter-cluster distance



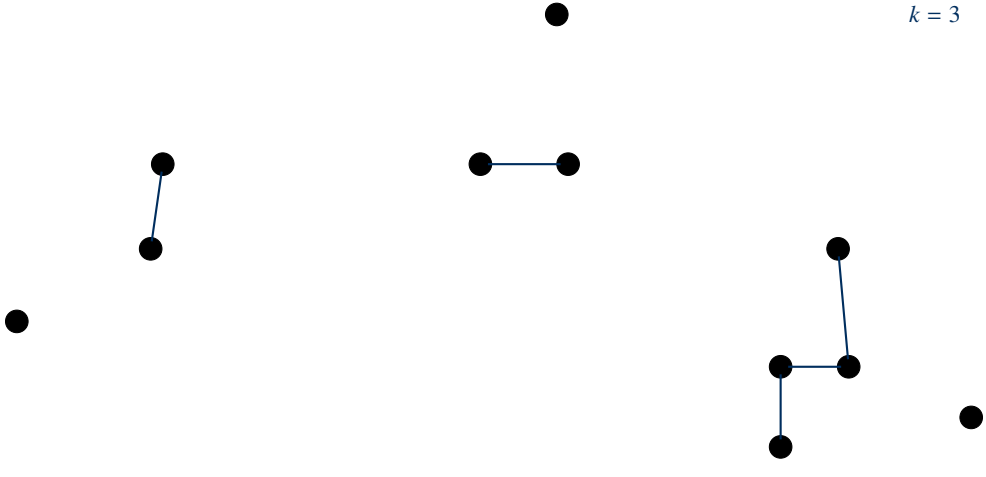
Maximizing inter-cluster distance



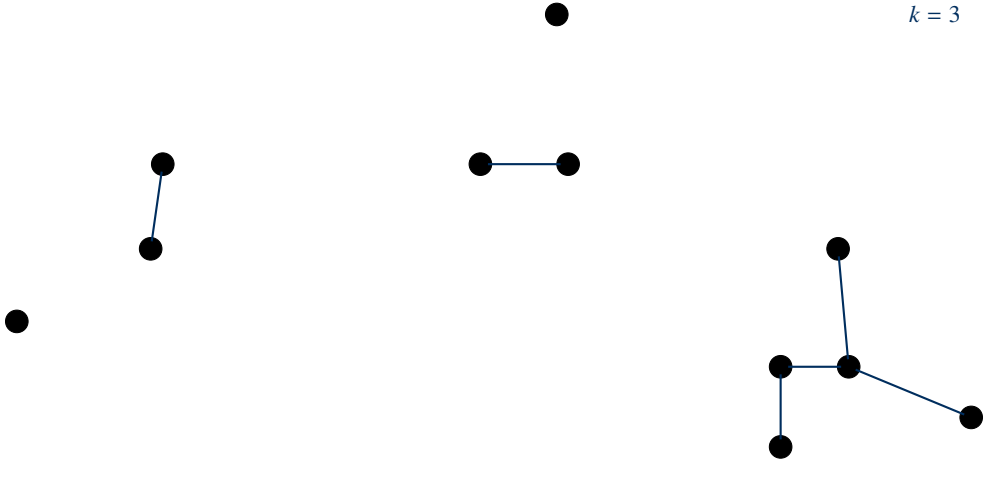
Maximizing inter-cluster distance



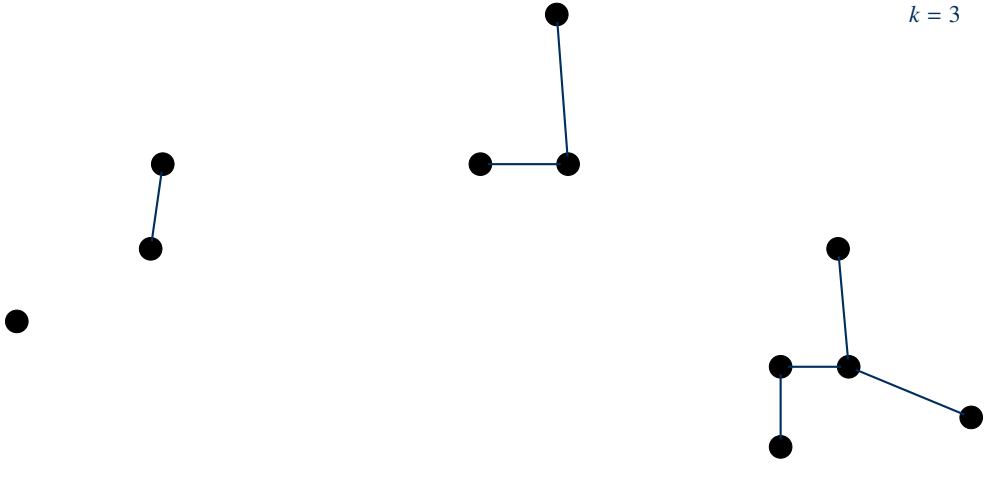
Maximizing inter-cluster distance



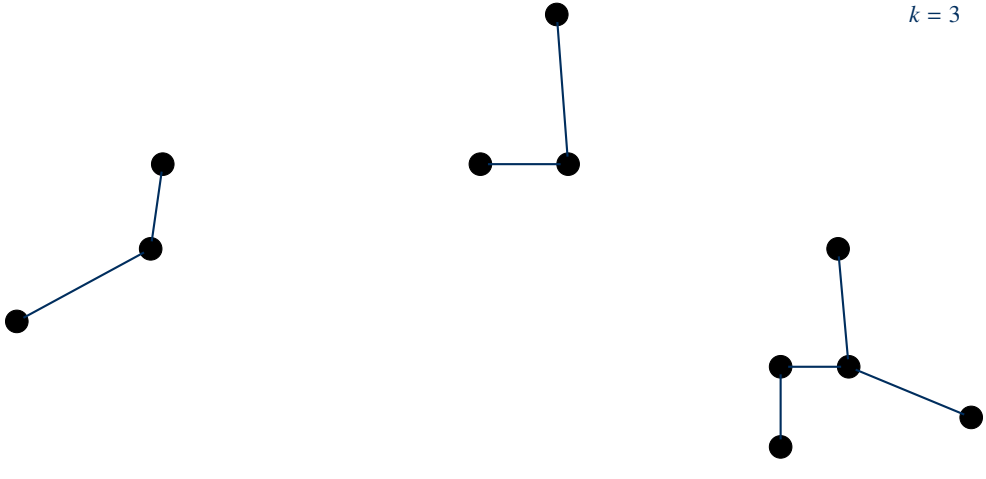
Maximizing inter-cluster distance



Maximizing inter-cluster distance



Maximizing inter-cluster distance



Maximizing inter-cluster distance

MAX-SPACING CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that maximises $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$

Maximizing inter-cluster distance

MAX-SPACING CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that maximises $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$

Solution: Kruskal's algorithm with early stopping (as soon as there are k trees)

Maximizing inter-cluster distance

MAX-SPACING CLUSTERING

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that maximises $\min_{x \in C_i, y \in C_j, i \neq j} dist(x, y)$

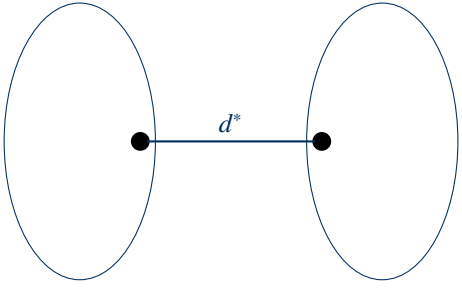
Solution: Kruskal's algorithm with early stopping (as soon as there are k trees)

Polynomial time!

Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.

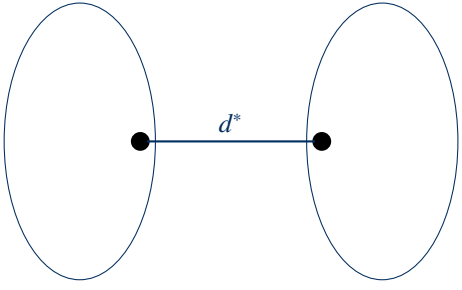


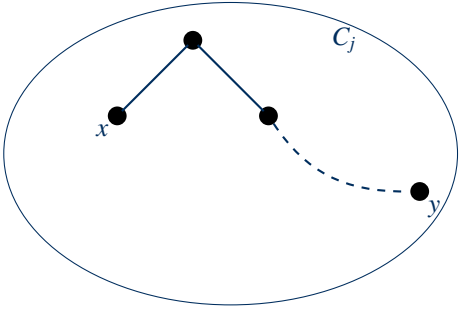
Let $C'_1, \ldots, C'_k$ be another clustering.

Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.



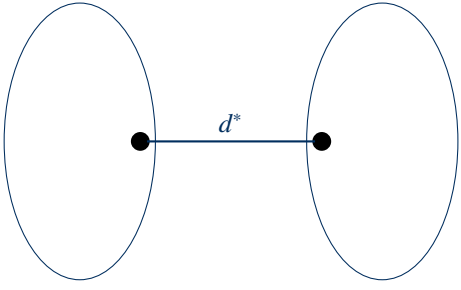


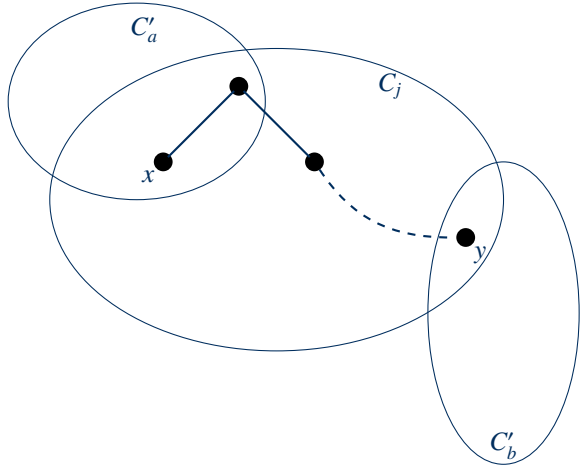
Let $C'_1, \ldots, C'_k$ be another clustering.

Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.



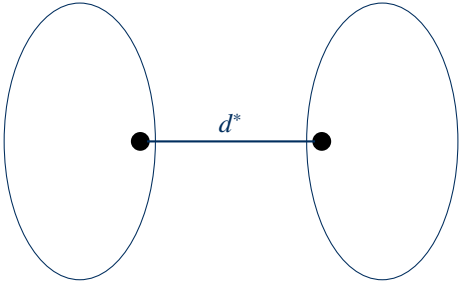


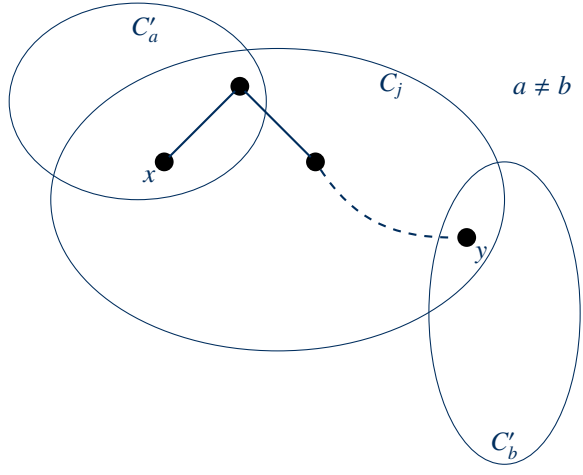
Let $C'_1, \ldots, C'_k$ be another clustering.

Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.



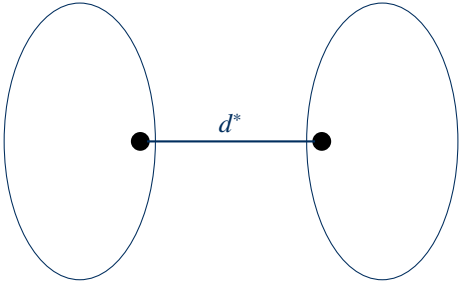


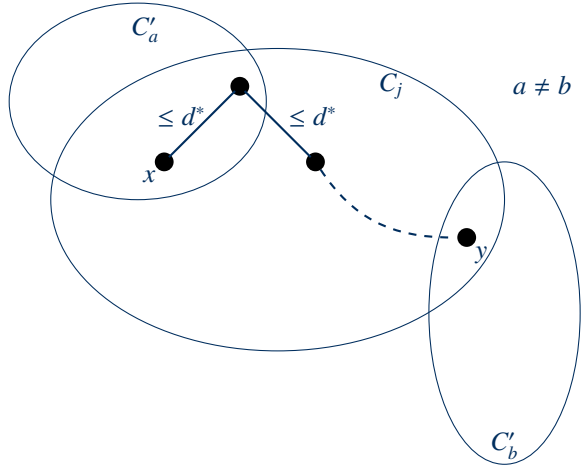
Let $C'_1, \ldots, C'_k$ be another clustering.

Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.



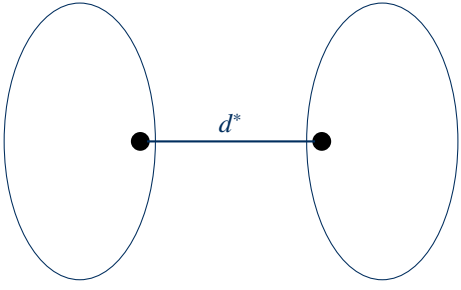


Let $C'_1, \ldots, C'_k$ be another clustering.

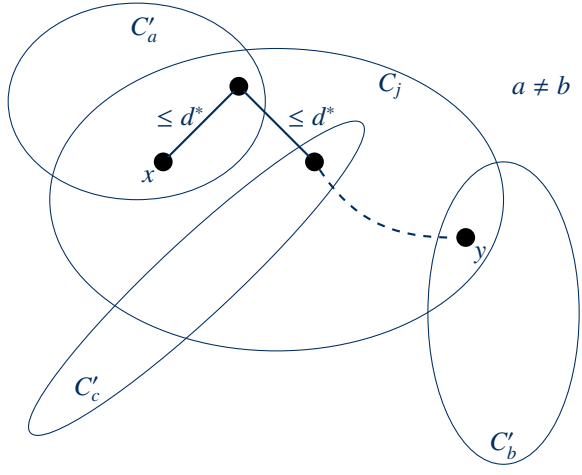
Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.



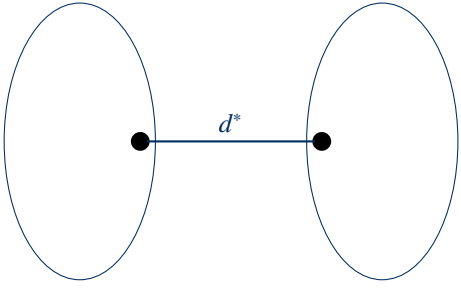
Let $C'_1, \ldots, C'_k$ be another clustering.

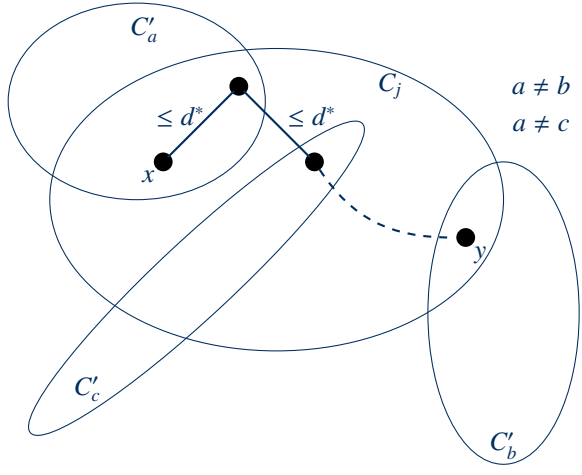


Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.



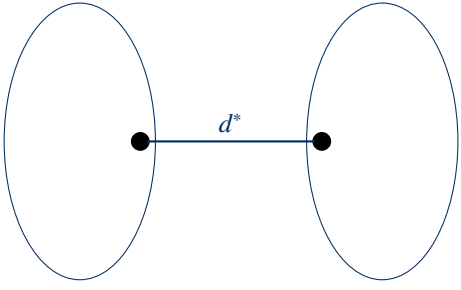


Let $C'_1, \ldots, C'_k$ be another clustering.

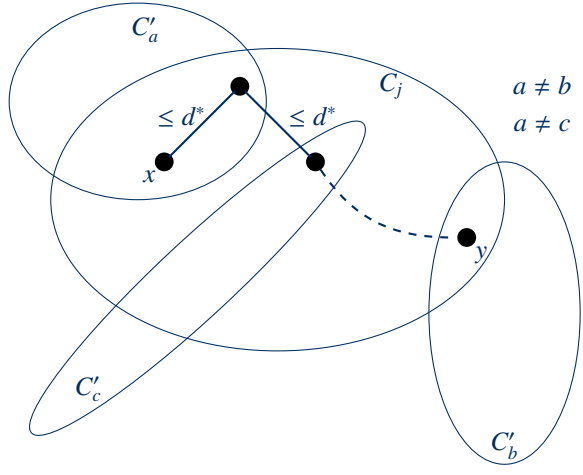
Maximizing Inter-Cluster Distance

Correctness

Let $C_1, \ldots, C_k$ be Kruskal's clustering with min distance d^* between clusters.



Let $C'_1, \ldots, C'_k$ be another clustering.



Distance between C'_a and $C'_c \leq d^*$

Maximizing inter-cluster distance

Which of the properties are important for our algorithm?

- $dist(x, y) \geq 0$
- $dist(x, x) = 0$
- $dist(x, y) = dist(y, x)$
- $dist(x, y) \leq dist(x, z) + dist(z, y)$

Maximizing inter-cluster distance

Which of the properties are important for our algorithm?

- $dist(x, y) \geq 0$
- $dist(x, x) = 0$
- $dist(x, y) = dist(y, x)$
- $dist(x, y) \leq dist(x, z) + dist(z, y)$

Minimizing intra-cluster distance

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

Minimizing intra-cluster distance

LOW-DIAMETER CLUSTERING

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

An NP-hard problem!!

Minimizing intra-cluster distance

Low-Diameter Clustering (decision version)

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters
- A maximum diameter m

Problem: Is there a partition of P into $C_1, \ldots, C_k$ such that

$$\forall i\colon D(C_i) \leq m$$

Minimizing intra-cluster distance

k-colorability $\leq_p$ **Low-Diameter Clustering**

Low-Diameter Clustering (decision version)

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters
- A maximum diameter m

Problem: Is there a partition of P into $C_1, \ldots, C_k$ such that

$$\forall i\colon D(C_i) \leq m$$

Minimizing intra-cluster distance

Low-Diameter Clustering (decision version)

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters
- A maximum diameter m

Problem: Is there a partition of P into $C_1, \ldots, C_k$ such that

$$\forall i\colon D(C_i) \leq m$$

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\qquad$ $P := V$

Minimizing intra-cluster distance

Low-Diameter Clustering (decision version)

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters
- A maximum diameter m

Problem: Is there a partition of P into $C_1, \ldots, C_k$ such that

$$\forall i\colon D(C_i) \leq m$$

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$	$P := V$
k	k

Minimizing intra-cluster distance

Low-Diameter Clustering (decision version)

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters
- A maximum diameter m

Problem: Is there a partition of P into $C_1, \ldots, C_k$ such that

$$\forall i\colon D(C_i) \leq m$$

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\quad$ $P := V$

k $\quad$ k

$$dist(u, v) = \begin{cases} 0, & u = v \\ 1, & u \neq v, (u, v) \notin E \\ 2, & u \neq v, (u, v) \in E \end{cases}$$

Minimizing intra-cluster distance

Low-Diameter Clustering (decision version)

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters
- A maximum diameter m

Problem: Is there a partition of P into $C_1, \ldots, C_k$ such that

$$\forall i\colon D(C_i) \leq m$$

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\quad$ $P := V$

k $\quad$ k

$$dist(u,v) = \begin{cases} 0, & u = v \\ 1, & u \neq v, (u,v) \notin E \\ 2, & u \neq v, (u,v) \in E \end{cases}$$

$\exists k$-coloring

$\Updownarrow$

$\exists k$-clustering with diameter 1

Minimizing intra-cluster distance

Low-Diameter Clustering (decision version)

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters
- A maximum diameter m

Problem: Is there a partition of P into $C_1, \ldots, C_k$ such that

$$\forall i\colon D(C_i) \le m$$

k-colorability $\le_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\quad$ $P := V$

k $\quad$ k

$$dist(u,v) = \begin{cases} 0, & u = v \\ 1, & u \neq v, (u,v) \notin E \\ 2, & u \neq v, (u,v) \in E \end{cases}$$

$\exists k$-coloring

$\Updownarrow$

$\exists k$-clustering with diameter 1

Proof: A valid coloring $\iff$
no single-colored $(u,v) \in E$ $\iff$
no $(u,v) \in E$ with u and v in the same cluster

Minimizing intra-cluster distance

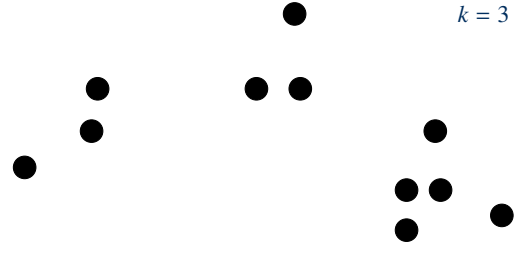
An approximate solution

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



Minimizing intra-cluster distance

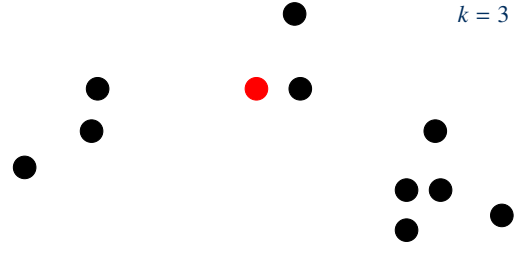
An approximate solution

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



Minimizing intra-cluster distance

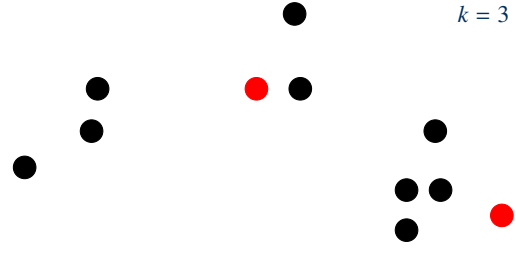
An approximate solution

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



Minimizing intra-cluster distance

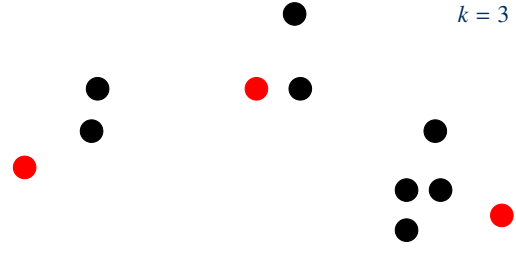
An approximate solution

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



Minimizing intra-cluster distance

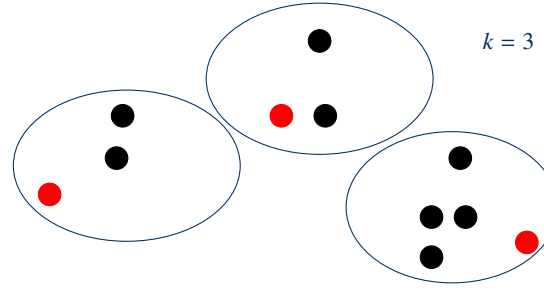
An approximate solution

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$



Minimizing intra-cluster distance

An approximate solution

$dist(p, C) = \min_{x \in C}(p, x)$

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:
- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$

$d(x, c_i) < d(x, c_j)$ or

Minimizing intra-cluster distance

An approximate solution

Low-Diameter Clustering

Input:

- A set P of points
- $dist\colon P^2 \to \mathbb{R}$
- A desired number k of clusters

Problem: Find a partition of P into $C_1, \ldots, C_k$ that minimizes $\max_{x \in C_i, y \in C_i} dist(x, y)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

Minimizing intra-cluster distance

An approximate solution

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

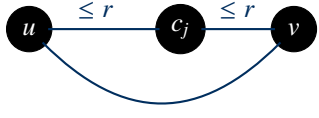
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

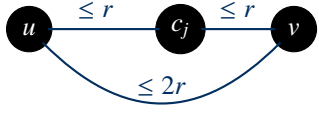
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

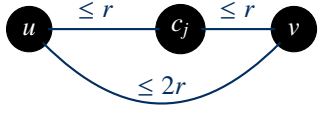
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



$\forall j\colon d(C_j) \leq 2r$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

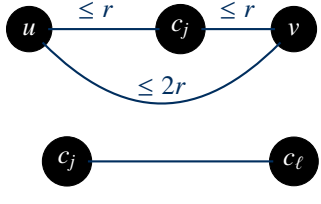
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

Let $u \in C_j$ and $v \in C_j$

$\forall j\colon d(C_j) \leq 2r$



$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

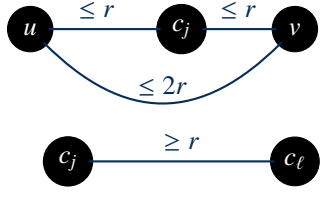
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

Let $u \in C_j$ and $v \in C_j$

$\forall j\colon d(C_j) \leq 2r$



$dist(p, C) = \min_{x \in C}(p, x)$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

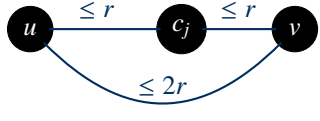
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

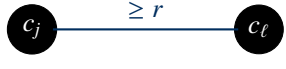
$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



$\forall j\colon d(C_j) \le 2r$

$\forall j \le \ell\colon dist(c_j, c_\ell) \ge r$



choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \ne i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

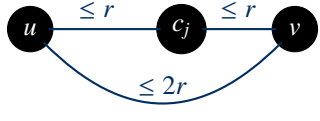
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

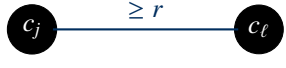
$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



$\forall j \colon d(C_j) \leq 2r$

$\forall j \leq \ell \colon dist(c_j, c_\ell) \geq r$



Let $C'_1, \ldots, C'_k$ be another clustering.

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

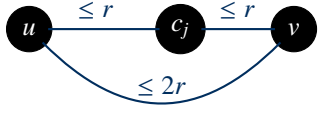
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

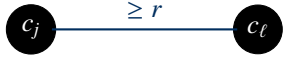
$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



$\forall j \colon d(C_j) \leq 2r$

$\forall j \leq \ell \colon dist(c_j, c_\ell) \geq r$



Let $C'_1, \ldots, C'_k$ be another clustering.
Among $c_1, \ldots, c_k, p$, two points are in the same cluster, C'_j.

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \neq i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

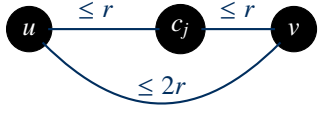
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

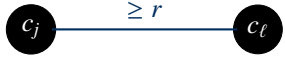
$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



$\forall j \colon d(C_j) \le 2r$

$\forall j \le \ell \colon dist(c_j, c_\ell) \ge r$



Let $C'_1, \ldots, C'_k$ be another clustering.
Among $c_1, \ldots, c_k, p$, two points are in the same cluster, C'_j.

$$\Rightarrow d(C'_j) \ge r$$

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \ne i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

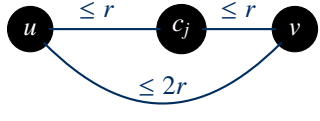
Minimizing intra-cluster distance

An approximate solution

$p = \arg\max_u dist(u, C) \quad p \in C_i \quad r := dist(p, c_i)$

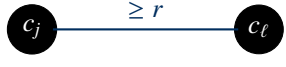
$dist(p, C) = \min_{x \in C}(p, x)$

Let $u \in C_j$ and $v \in C_j$



$\forall j\colon d(C_j) \le 2r$

$\forall j \le \ell\colon dist(c_j, c_\ell) \ge r$



Let $C'_1, \ldots, C'_k$ be another clustering.
Among $c_1, \ldots, c_k, p$, two points are in the same cluster, C'_j.

$$\Rightarrow d(C'_j) \ge r$$

A 2-approximate polynomial-time algorithm!

choose $c_1 \in P$

$C := \{c_1\}$

while $|C| < k$

$c_{|C|+1} := \arg\max dist(p, C)$

$C := C \cup \{c_{|C|+1}\}$

for $i := 1$ to k

$C_i = \{x \in P \mid \forall j \ne i :$
$d(x, c_i) < d(x, c_j)$ or
$(d(x, c_i) = d(x, c_j), i < j)\}$

Better-Quality Approximation?

Suppose that there is an algorithm that computes a $(2 - \varepsilon)$-approximation for $\varepsilon > 0$.

We would have solved k-colorability.

Better-Quality Approximation?

Suppose that there is an algorithm that computes a $(2-\varepsilon)$-approximation for $\varepsilon > 0$.

We would have solved k-colorability.

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\quad$ $P := V$

k $\quad$ k

$$dist(u,v) = \begin{cases} 0, & u = v \\ 1, & u \neq v, (u,v) \notin E \\ 2, & u \neq v, (u,v) \in E \end{cases}$$

$\exists k$-coloring

$\Updownarrow$

$\exists k$-clustering with diameter 1

Proof: A valid coloring $\iff$
no single-colored $(u,v) \in E$ $\iff$
no $(u,v) \in E$ with u and v in the same cluster

Better-Quality Approximation?

Suppose that there is an algorithm that computes a $(2-\varepsilon)$-approximation for $\varepsilon > 0$.

optimal diameter ≤ 1

$\Updownarrow$

approximately optimal diameter $\leq 2-\varepsilon$

We would have solved k-colorability.

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\quad$ k

$P := V$ $\quad$ k

$$dist(u,v) = \begin{cases} 0, & u = v \\ 1, & u \neq v, (u,v) \notin E \\ 2, & u \neq v, (u,v) \in E \end{cases}$$

$\exists k$-coloring

$\Updownarrow$

$\exists k$-clustering with diameter 1

Proof: A valid coloring $\iff$
no single-colored $(u,v) \in E$ $\iff$
no $(u,v) \in E$ with u and v in the same cluster

Better-Quality Approximation?

Suppose that there is an algorithm that computes a $(2-\varepsilon)$-approximation for $\varepsilon > 0$.

optimal diameter ≤ 1

$\Updownarrow$

approximately optimal diameter $\leq 2-\varepsilon$

$\Updownarrow$

approximately optimal diameter ≤ 1

We would have solved k-colorability.

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\qquad$ $P := V$

k $\qquad$ k

$$dist(u,v) = \begin{cases} 0, & u = v \\ 1, & u \neq v, (u,v) \notin E \\ 2, & u \neq v, (u,v) \in E \end{cases}$$

$\exists k$-coloring

$\Updownarrow$

$\exists k$-clustering with diameter 1

Proof: A valid coloring $\iff$
no single-colored $(u,v) \in E$ $\iff$
no $(u,v) \in E$ with u and v in the same cluster

Better-Quality Approximation?

Suppose that there is an algorithm that computes a $(2-\varepsilon)$-approximation for $\varepsilon > 0$.

optimal diameter ≤ 1

$\Updownarrow$

approximately optimal diameter $\leq 2-\varepsilon$

$\Updownarrow$

approximately optimal diameter ≤ 1

We would have solved k-colorability.

No $(2-\varepsilon)$-approximation unless P = NP.

k-colorability $\leq_p$ **Low-Diameter Clustering**

$G = (V, E)$ $\quad$ $P := V$

k $\quad$ k

$$dist(u,v) = \begin{cases} 0, & u = v \\ 1, & u \neq v, (u,v) \notin E \\ 2, & u \neq v, (u,v) \in E \end{cases}$$

$\exists k$-coloring

$\Updownarrow$

$\exists k$-clustering with diameter 1

Proof: A valid coloring $\iff$
no single-colored $(u,v) \in E$ $\iff$
no $(u,v) \in E$ with u and v in the same cluster

Summary and Outlook

Some NP-hard problems admit polynomial-time constant-factor approximation algorithms.

Others do not.

Approximations do not generally carry over reductions.

It may be possible to prove that a problem does not have a constant-factor approximation algorithm (at all or for a particular constant) by exploring the reduction in its NP-hardness proof.

What's next?

- Polynomial-time approximation schemes
- Parameterized complexity
- Examinations