CHAPTER 2

Greedy algorithms and Local search

Introduction

- A **greedy** algorithm builds a solution step by step: Each time going for the option that gives the most profit or smallest cost at that step. It stops when a feasible solution is found.

- A **local search** algorithm starts with a feasible solution and changes the solution step by step: Each time selecting an improved solution in the neighborhood of the current solution. It stops when no local improvement is possible.

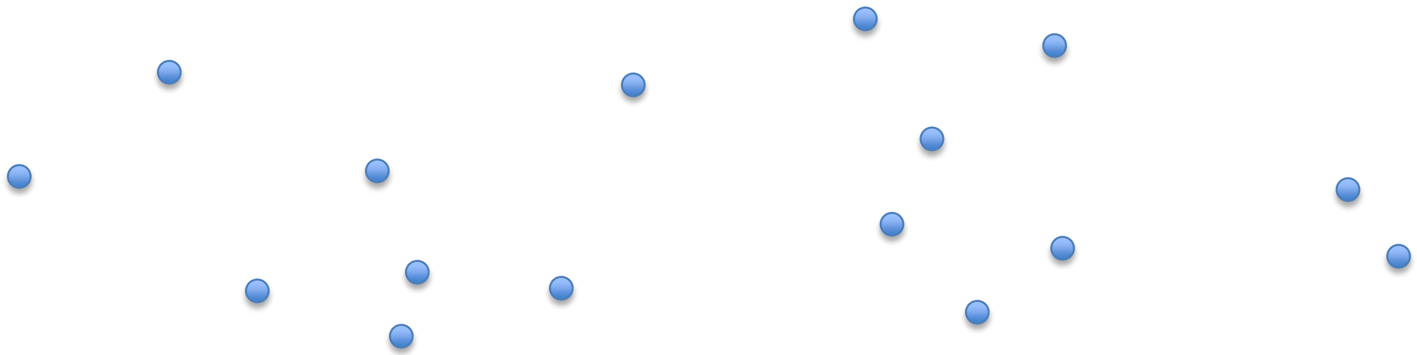
k-center

Instance: Set V of n points in a metric space. Integer k.

Solution: $S \subseteq V$ with $|S|=k$

Cost: $\max_{j\in V} \text{dist}(j,S)$ ($\text{dist}(j,S) := \min_{s\in S} \text{dist}(j,s)$)

Goal: Find a solution of minimum cost.



k=3

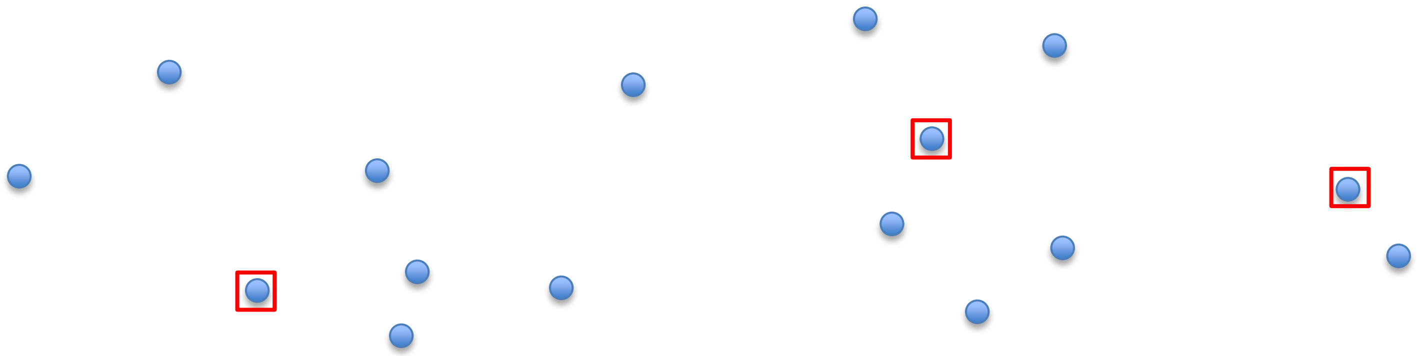
k-center

Instance: Set V of n points in a metric space. Integer k.

Solution: $S \subseteq V$ with $|S|=k$

Cost: $\max_{j\in V} \mathrm{dist}(j,S)$ ($\mathrm{dist}(j,S) := \min_{s\in S} \mathrm{dist}(j,s)$)

Goal: Find a solution of minimum cost.



k=3

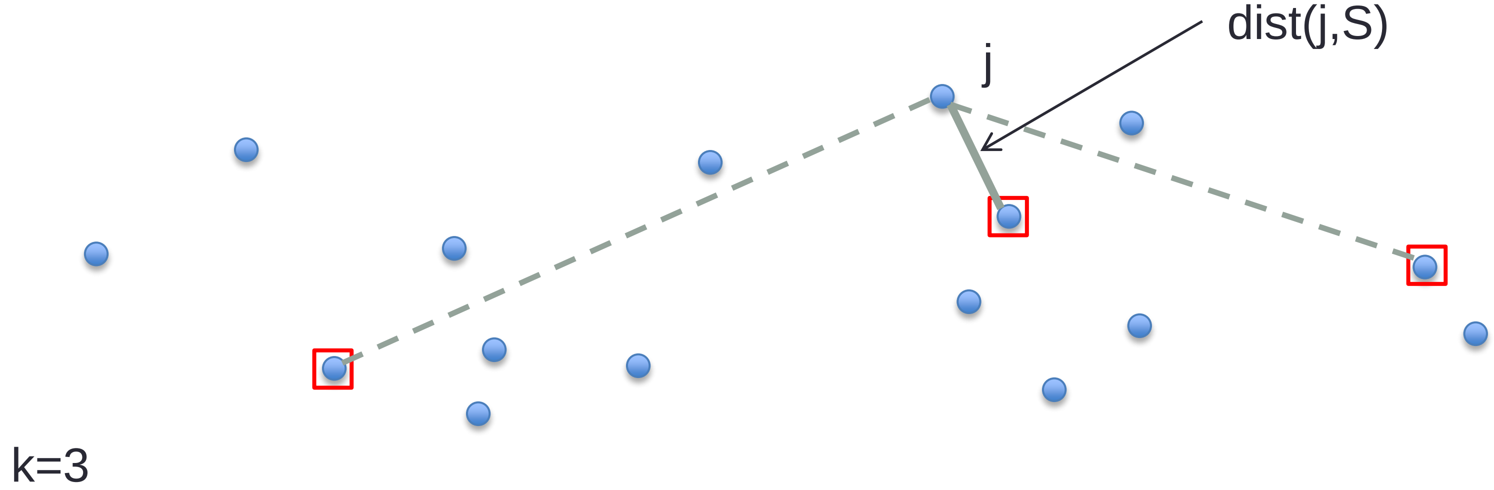
k-center

Instance: Set V of n points in a metric space. Integer k.

Solution: $S \subseteq V$ with $|S|=k$

Cost: $\max_{j\in V} \mathrm{dist}(j,S)$ ($\mathrm{dist}(j,S) := \min_{s\in S} \mathrm{dist}(j,s)$)

Goal: Find a solution of minimum cost.



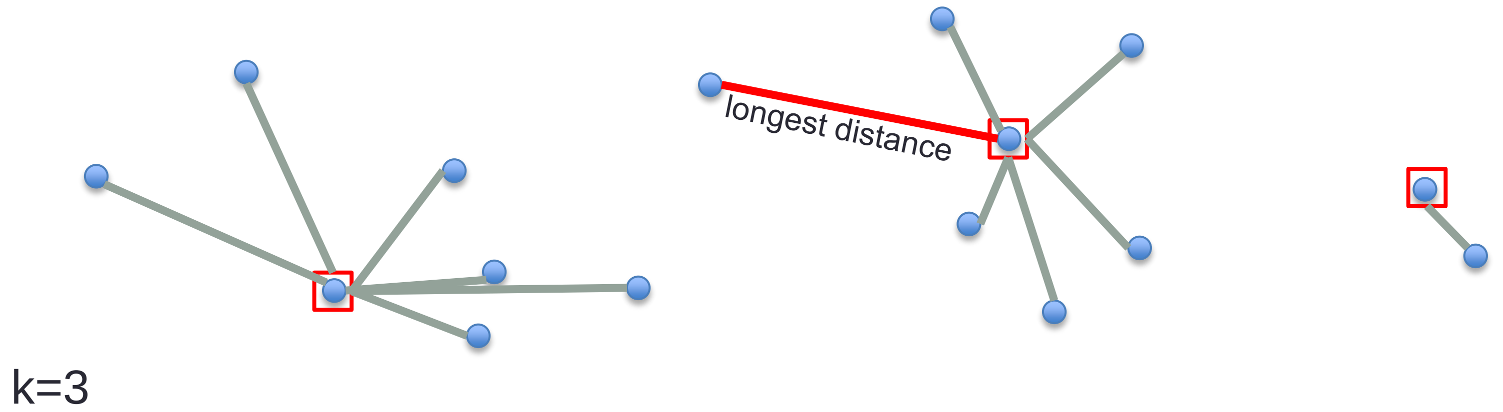
k-center

Instance: Set V of n points in a metric space. Integer k.

Solution: $S \subseteq V$ with $|S|=k$

Cost: $\max_{j\in V} \mathrm{dist}(j,S)$ ($\mathrm{dist}(j,S) := \min_{s\in S} \mathrm{dist}(j,s)$)

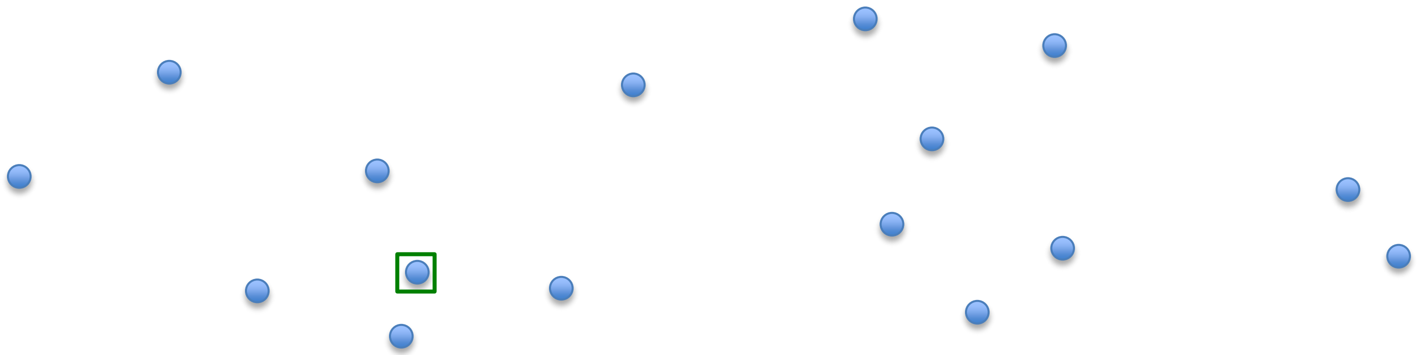
Goal: Find a solution of minimum cost.



Connect each to its nearest center

k-center

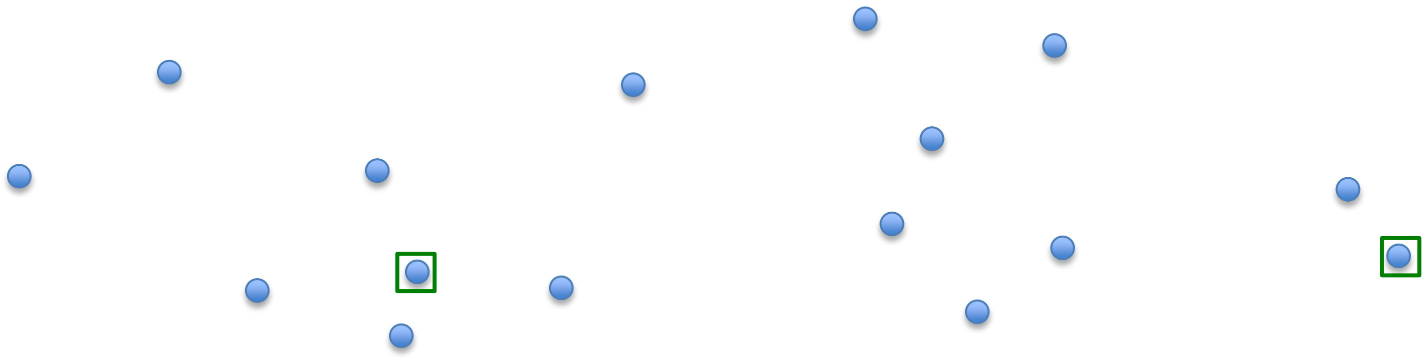
Greedy: Pick the first center arbitrarily. Next, always choose the point that is furthest away from the set of centers already chosen.



k=3

k-center

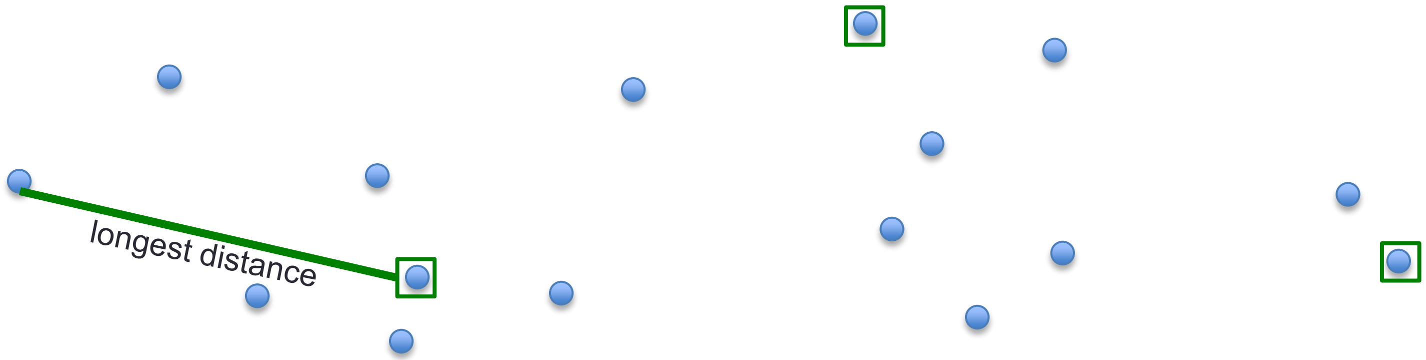
Greedy: Pick the first center arbitrarily. Next, always choose the point that is furthest away from the set of centers already chosen.



k=3

k-center

Greedy: Pick the first center arbitrarily. Next, always choose the point that is furthest away from the set of centers already chosen.



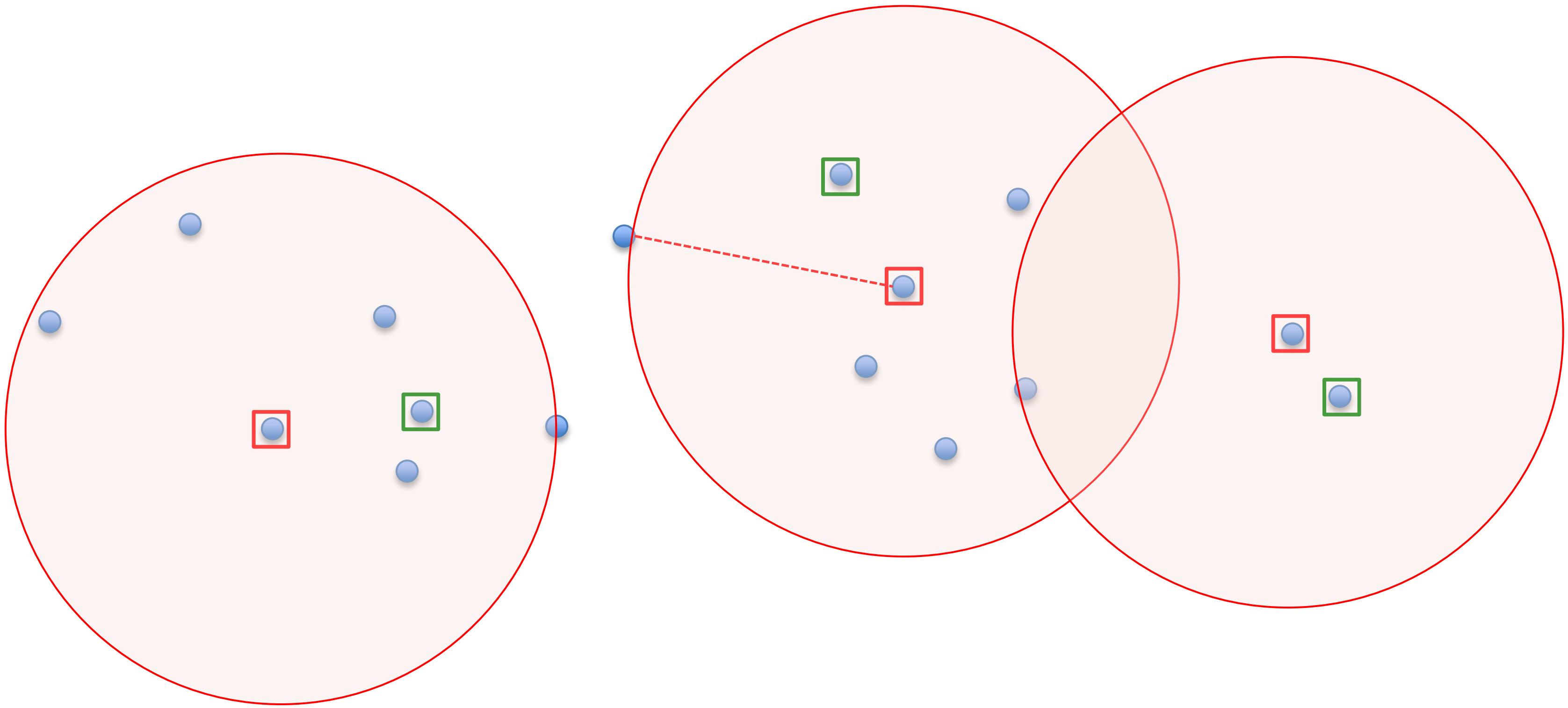
k=3

k-center

Theorem: Greedy is a 2-approximation algorithm.

Proof: Let S*, r* be, respectively, optimal solution and its value.

→ All points are within distance r* from S*



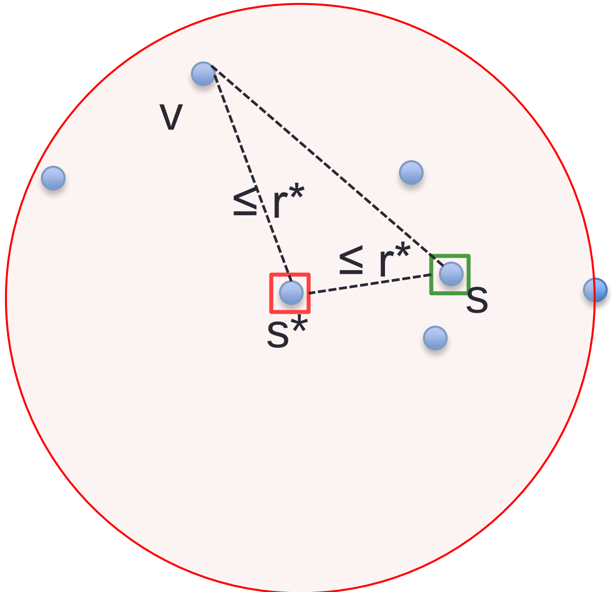
k-center

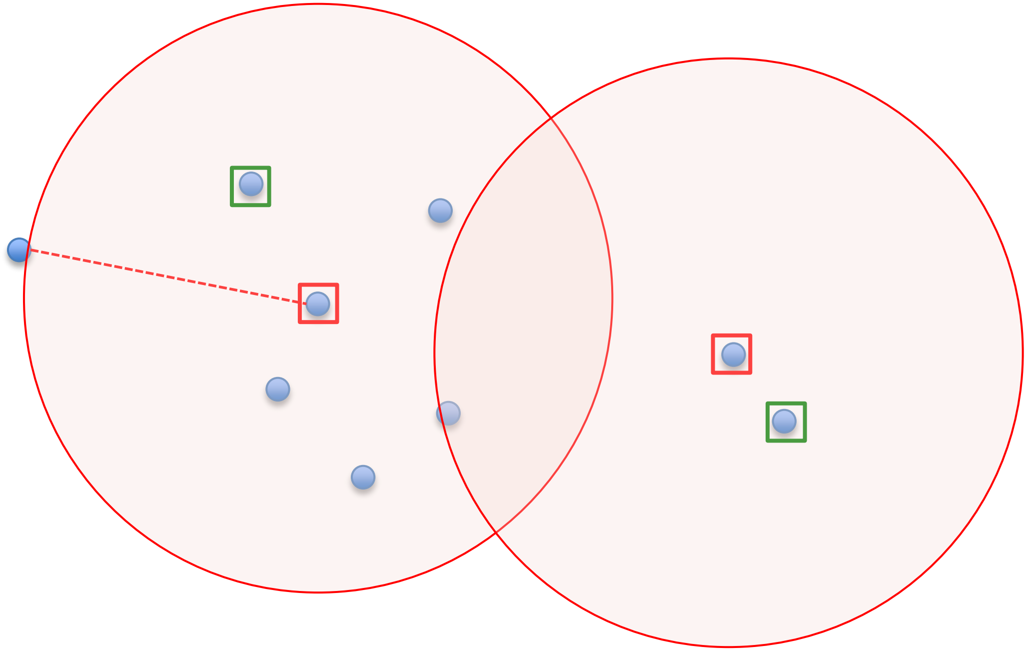
Proof (cont.):

Take any v∈ V. There must be an s*∈ S* with dist(v,s*) ≤ r*.

<u>If</u> there is an s∈ S with dist(s,s*) ≤ r*. Then OK by triangle inequality.

What if no such s?

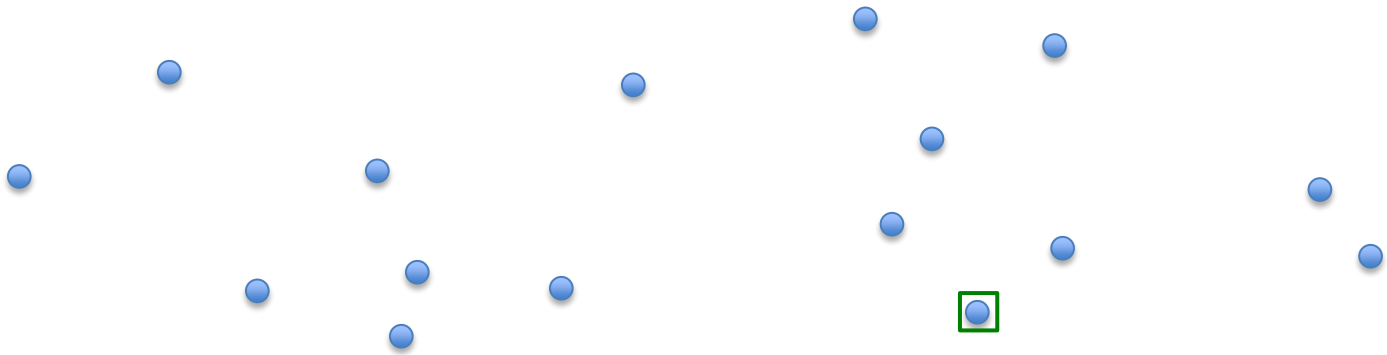




k-center

Proof (cont.):

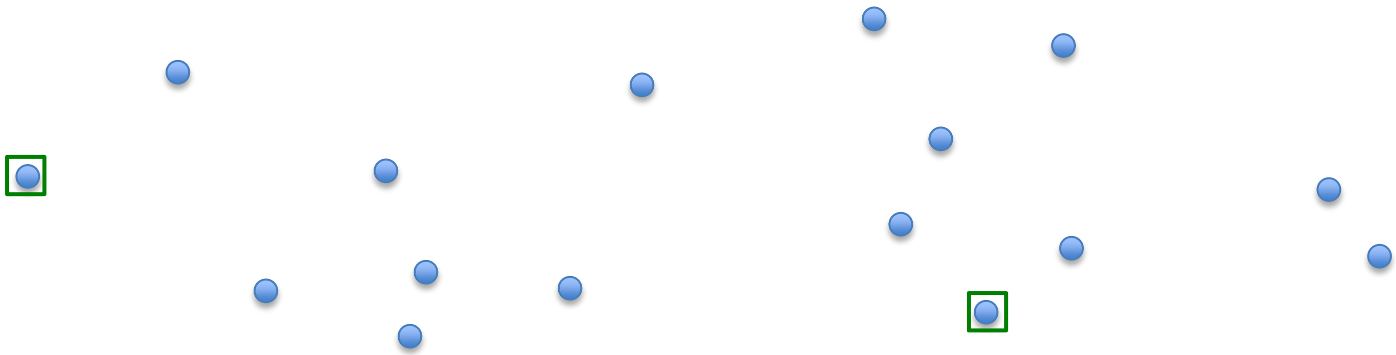
Another Greedy solution:



k-center

Proof (cont.):

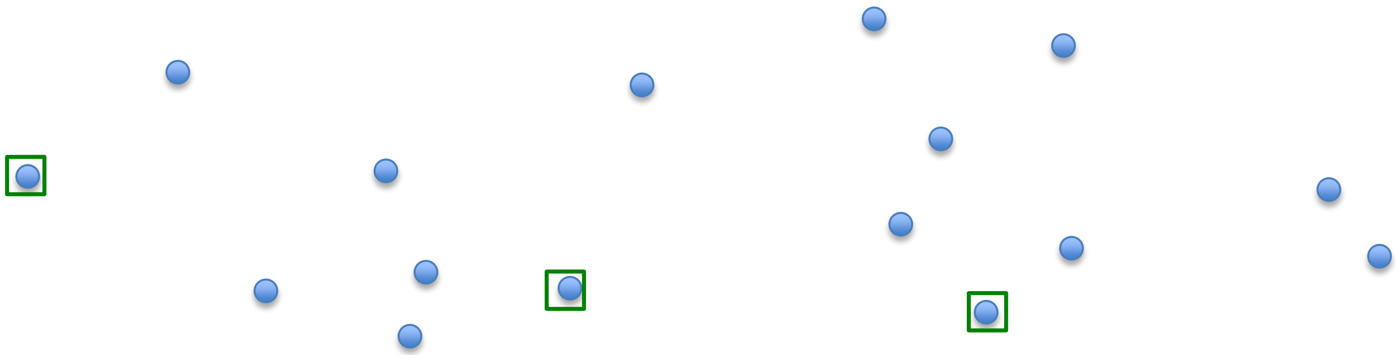
Another Greedy solution:



k-center

Proof (cont.):

Another Greedy solution:



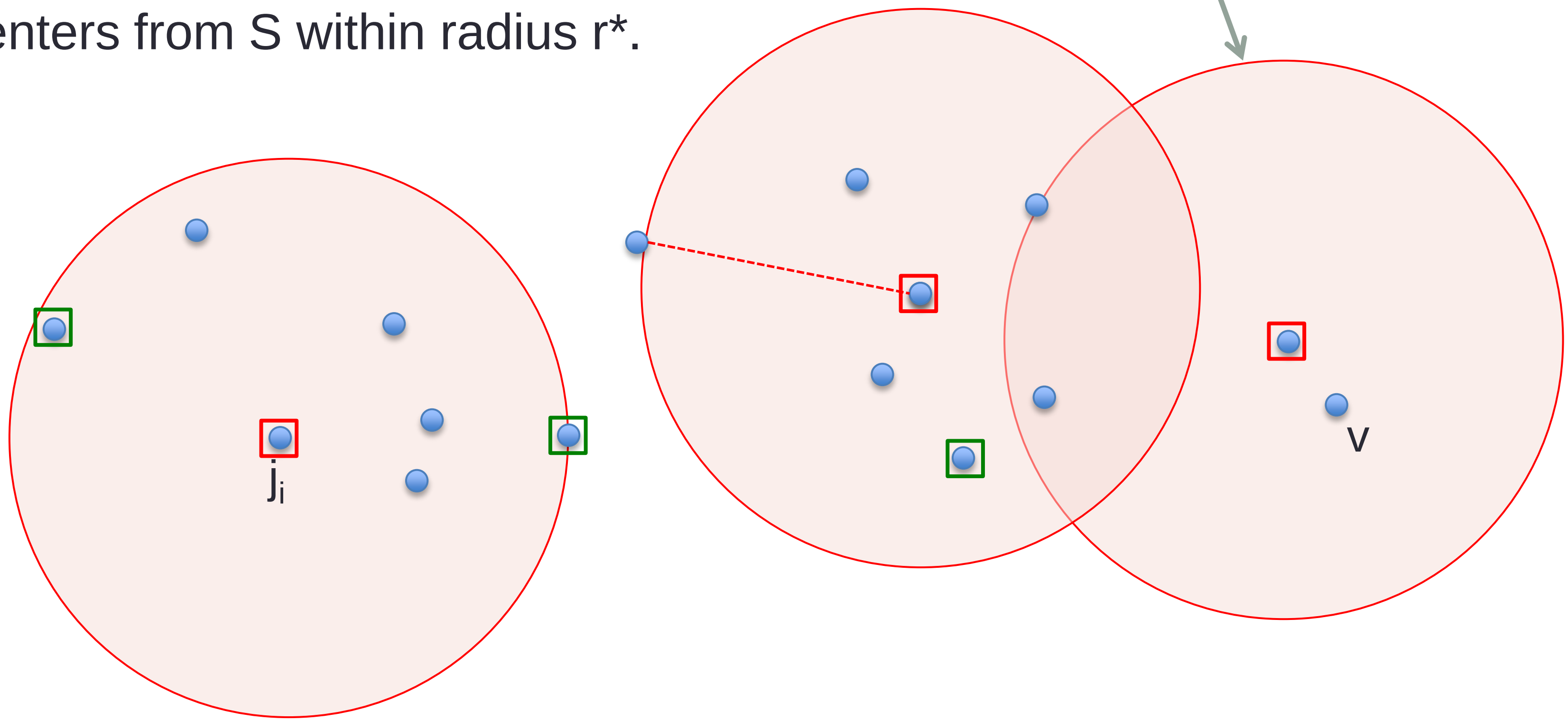
k-center

Proof (cont.):

Can not use the same argument here.

→There must be a center j_i with at least 2 centers from S within radius r*.

No Greedy center in here.

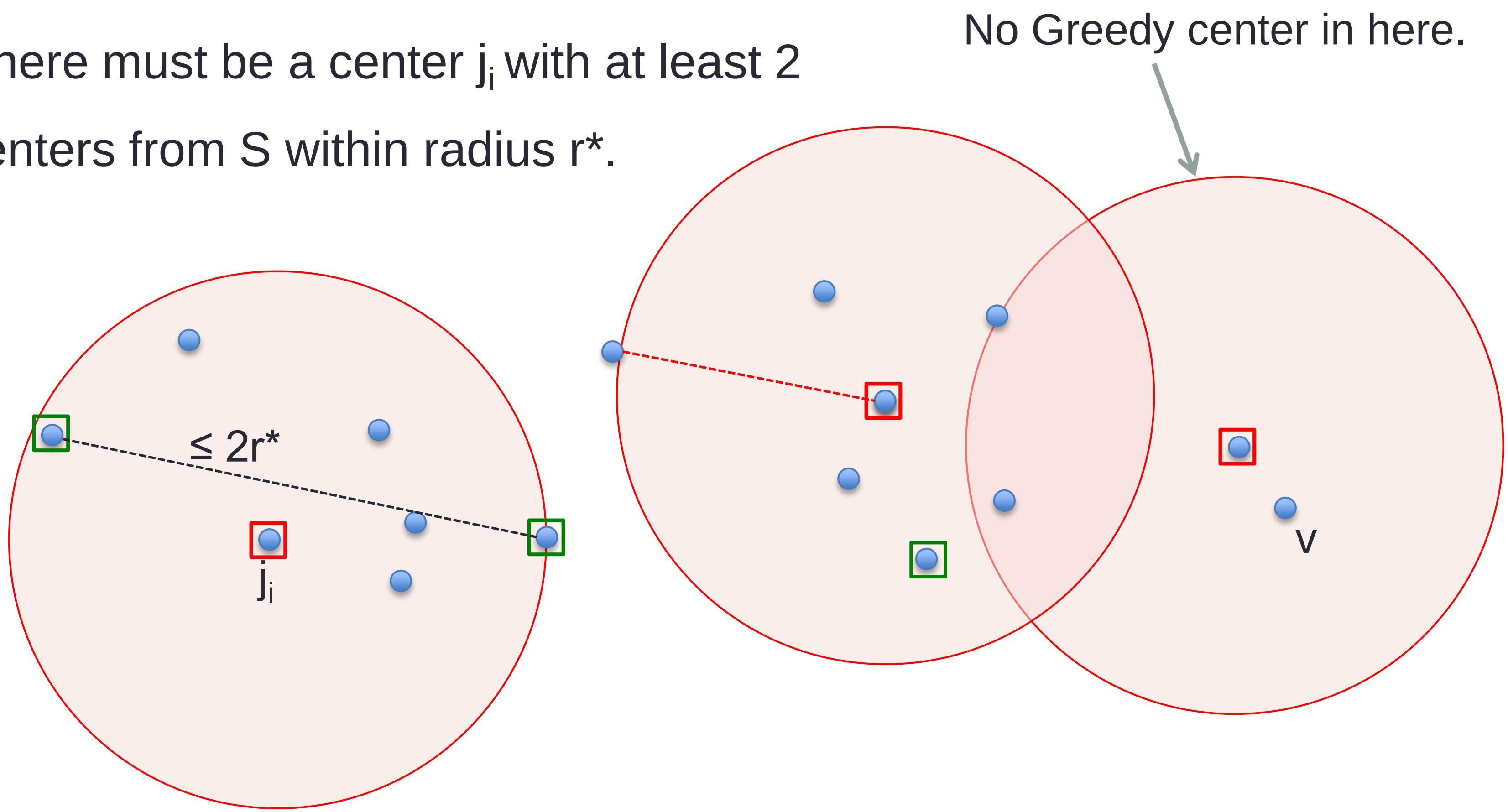


k-center

Proof (cont.):

Can not use the same argument here.

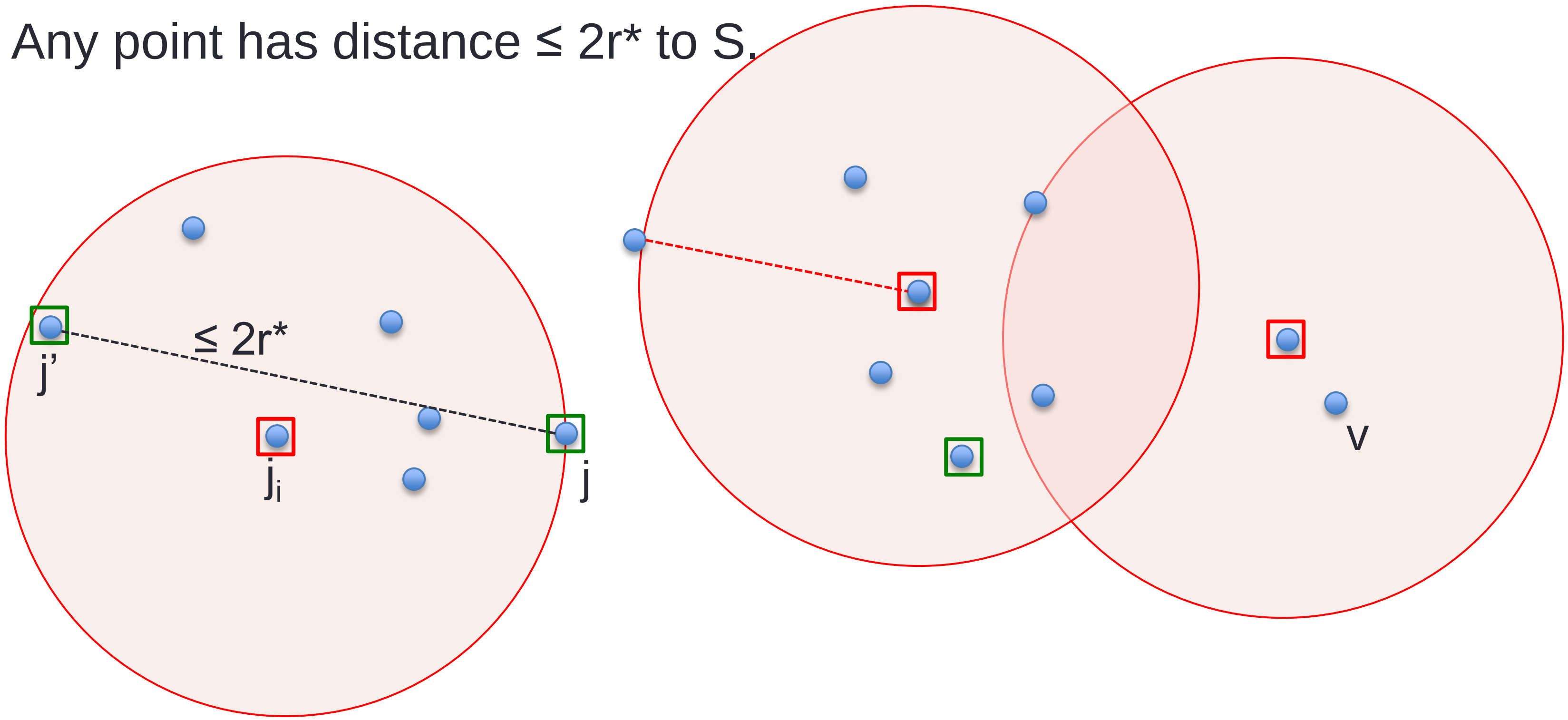
→There must be a center j_i with at least 2 centers from S within radius r*.



k-center

Proof (cont.):

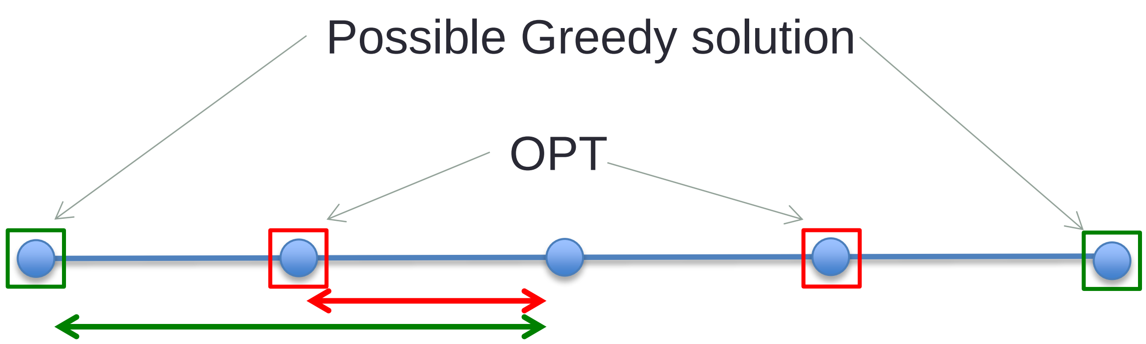
- j was picked after j'.
- j had maximum distance to the chosen centers when it was picked.
- → Any point has distance ≤ $2r^*$ to S.



k-center

Theorem Greedy is not better than a 2-approximation.

Proof See example.



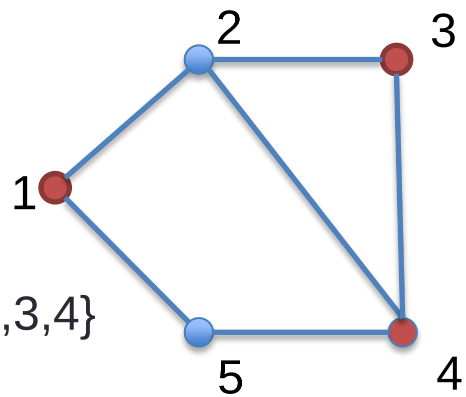
k-center

Theorem There is no α-approximation algorithm for any $\alpha < 2$, unless P=NP.

Proof By a reduction from the Dominating Set problem. (Next slides)

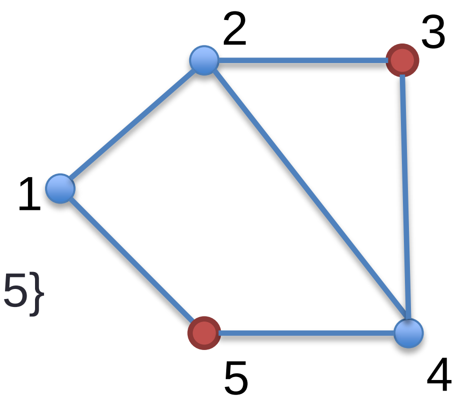
Dominating Set

Similar to Vertex Cover.



Vertex Cover:
Any edge has an endpoint in I

I={1,3,4}



Dominating Set:
Any vertex is in I or has a neighbour in I.

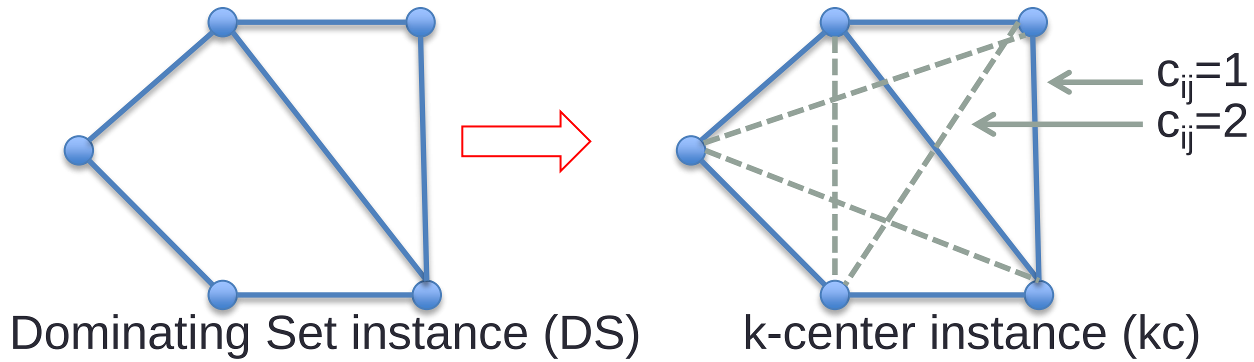
I={3,5}

Dominating Set

Fact: Dominating Set is NP-complete (Somebody once proved this).

Corollary: There is no α-approximation algorithm for any $\alpha < 2$, unless P=NP.

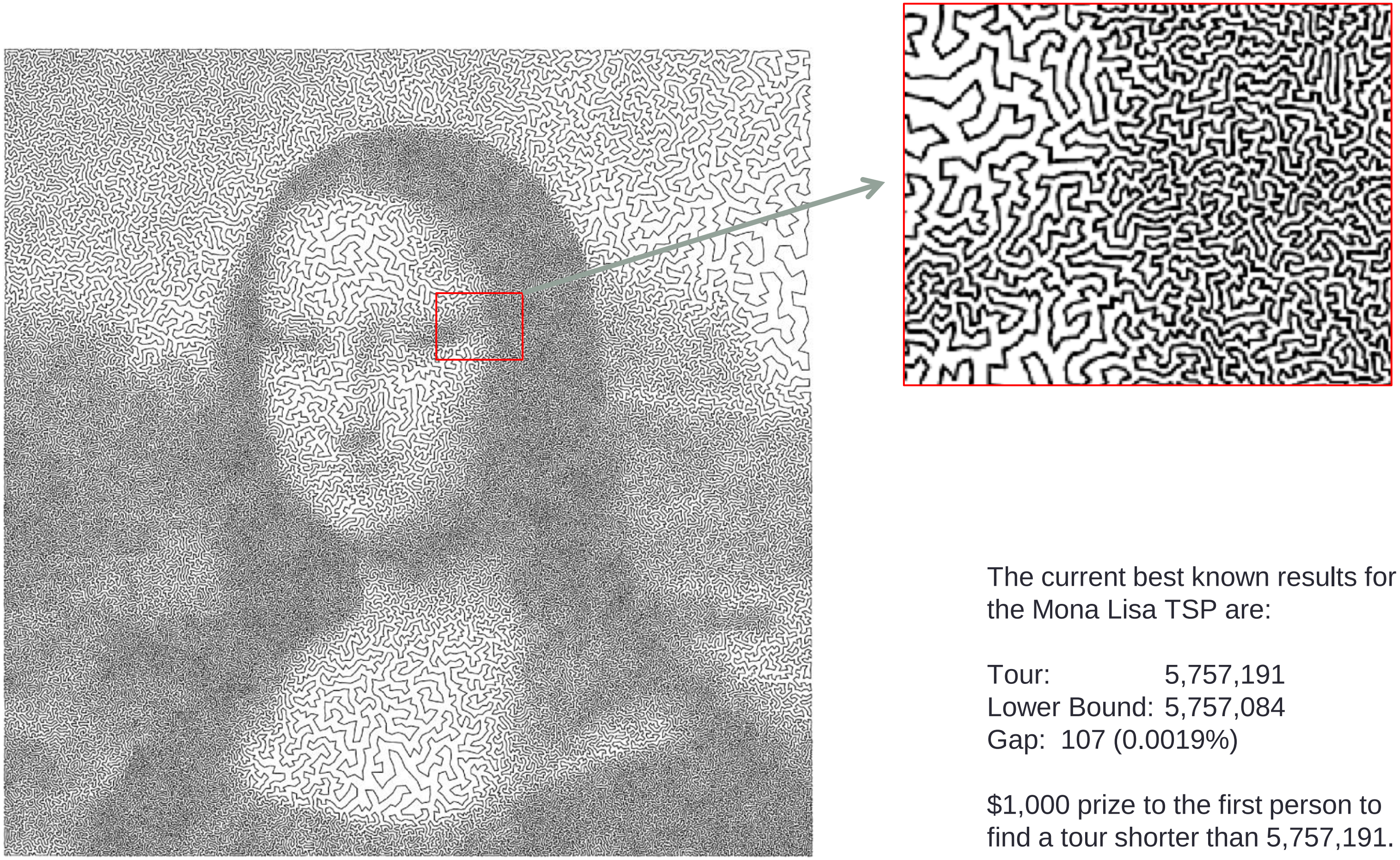
Proof: Let $G=(V,E)$ be an instance of DS. Is $OPT \leq k$? This can be answered if we would have an α-approximation algorithm ALG for k-center with $\alpha < 2$.



$OPT^{DS} \leq k \longrightarrow OPT^{kc}=1 \longrightarrow ALG \leq \alpha OPT^{kc} < 2 \longrightarrow ALG=1$

$OPT^{DS} > k \longrightarrow OPT^{kc}=2 \longrightarrow ALG=2$

Traveling Salesman Problem



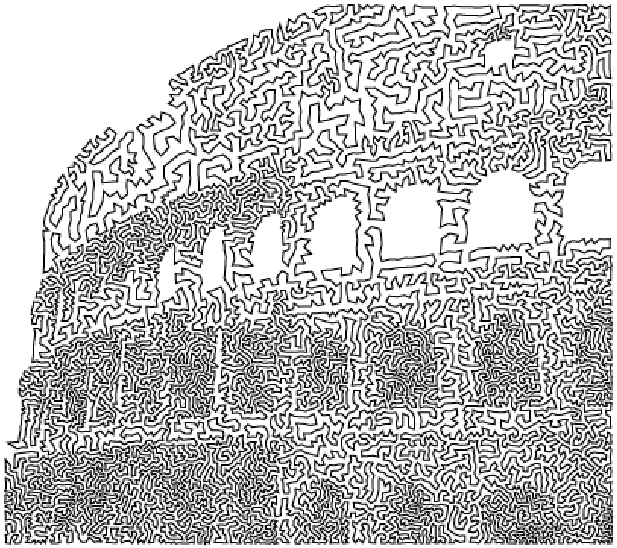
The current best known results for the Mona Lisa TSP are:

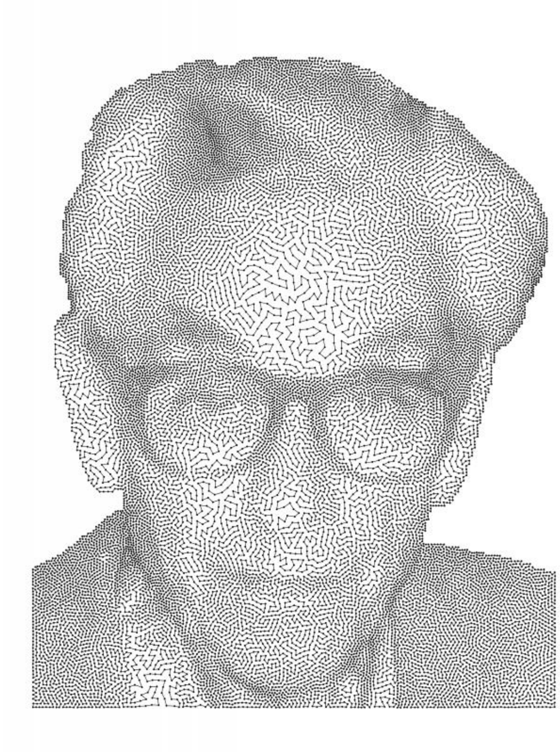
Tour: 5,757,191
Lower Bound: 5,757,084
Gap: 107 (0.0019%)

$1,000 prize to the first person to find a tour shorter than 5,757,191.

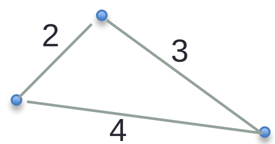
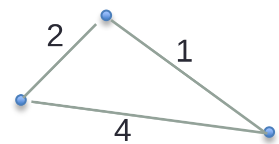
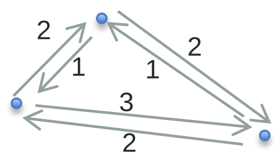
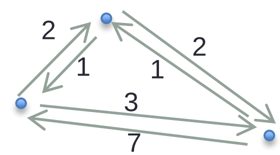
Source:
http://www.math.uwaterloo.ca/tsp/data/ml/monalisa.html

Traveling Salesman Problem





Traveling Salesman Problem

TSP	Metric (Δ-inequality)	Non-metric
Symmetric	 1.5-approx. Section 2.4	 No α-approx. Section 2.4
Asymmetric	 O(log n) -approx. Exercise 1.3	 No α-approx. Section 2.4

A special case of Symmetric metric TSP is *Euclidean TSP*
(Pictures on previous slides)

Traveling Salesman Problem

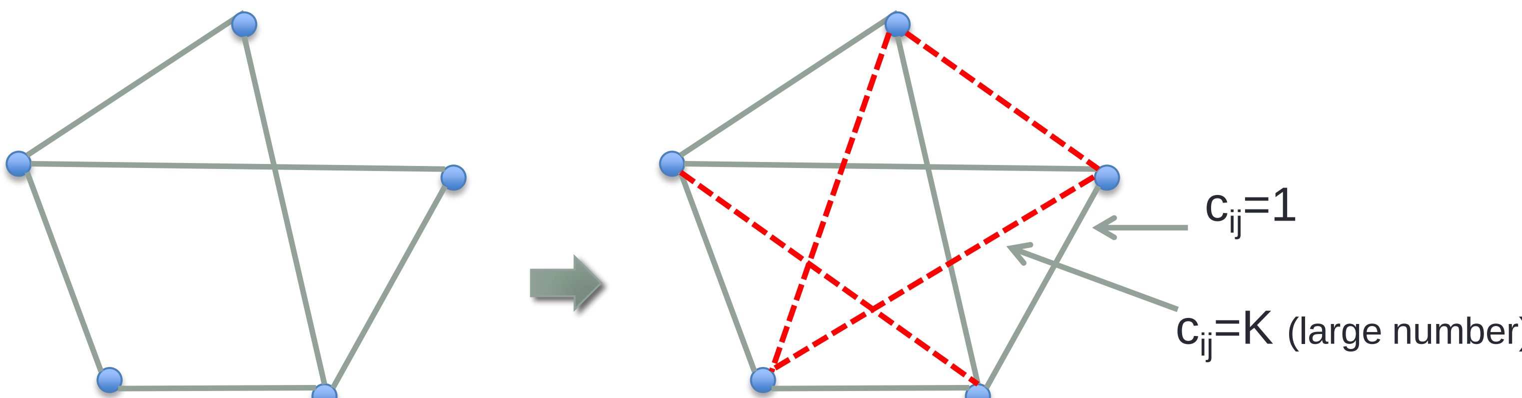
Theorem For TSP without the triangle inequality assumption, there does not exist an α-approximation algorithm for any $\alpha \geq 1$, provided P≠NP.

Proof (next slide) Follows from a reduction from Hamiltonian Cycle (HC). We show that, if there exists an α-approximation algorithm ALG with $\alpha \geq 1$, then the HC problem can be solved in polynomial time.

Traveling Salesman Problem

Fact The Hamiltonian Cycle problem is NP-complete
(Somebody once proved this)

Corrolary For TSP without the triangle inequality assumption, there does not exist an α-approximation algorithm for any $\alpha \geq 1$, provided P≠NP.



G=(V,E), |V|=n

TSP-instance.

HC? Yes. $\longrightarrow$ $OPT^{TSP}=n$ $\longrightarrow$ $ALG \leq \alpha n$

HC? No $\longrightarrow$ $OPT^{TSP} \geq K+n-1$ $\longrightarrow$ $ALG \geq K+n-1$ $\xrightarrow[\boxed{\text{Take } K \geq \alpha n}]{}$ $ALG \geq \alpha n+1$

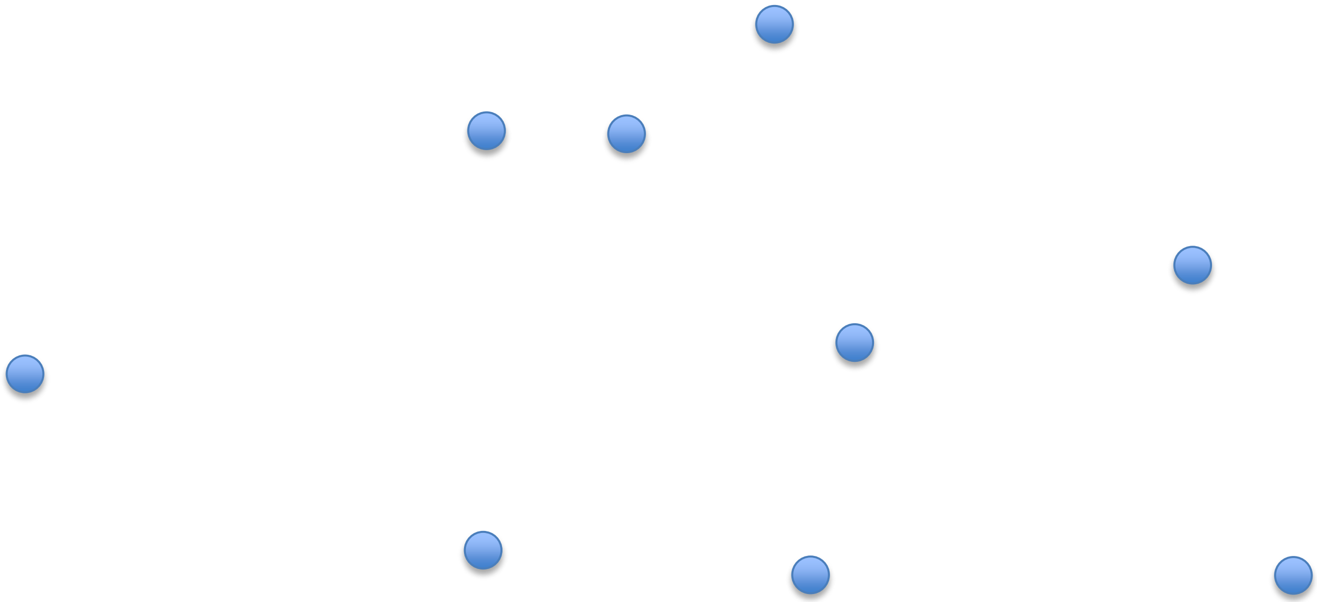
Traveling Salesman Problem

Three algorithms for metric symmetric TSP:

- Double tree
- Nearest addition
- Christofides' algorithm

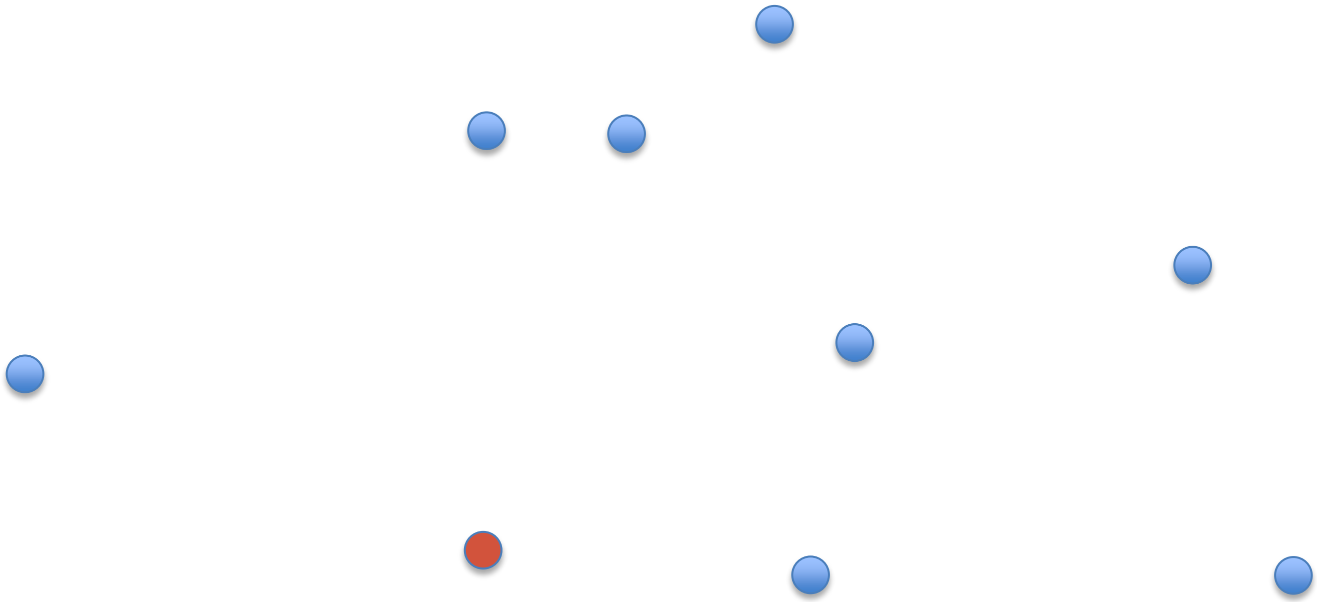
Prim’s Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



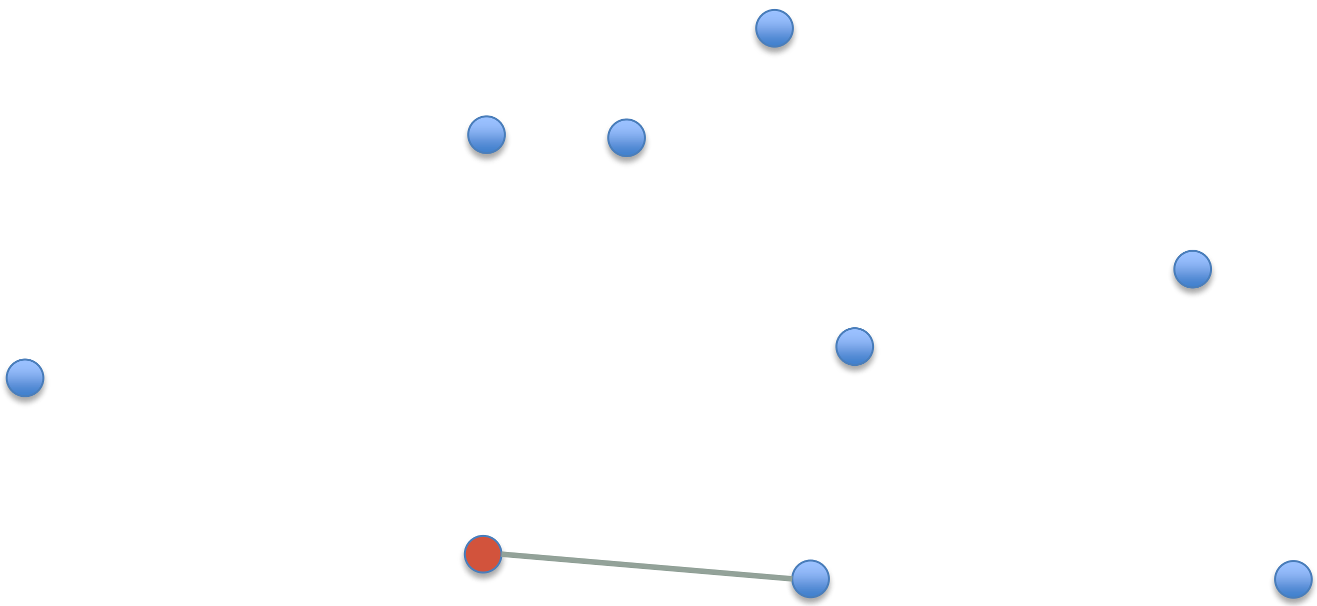
Prim’s Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



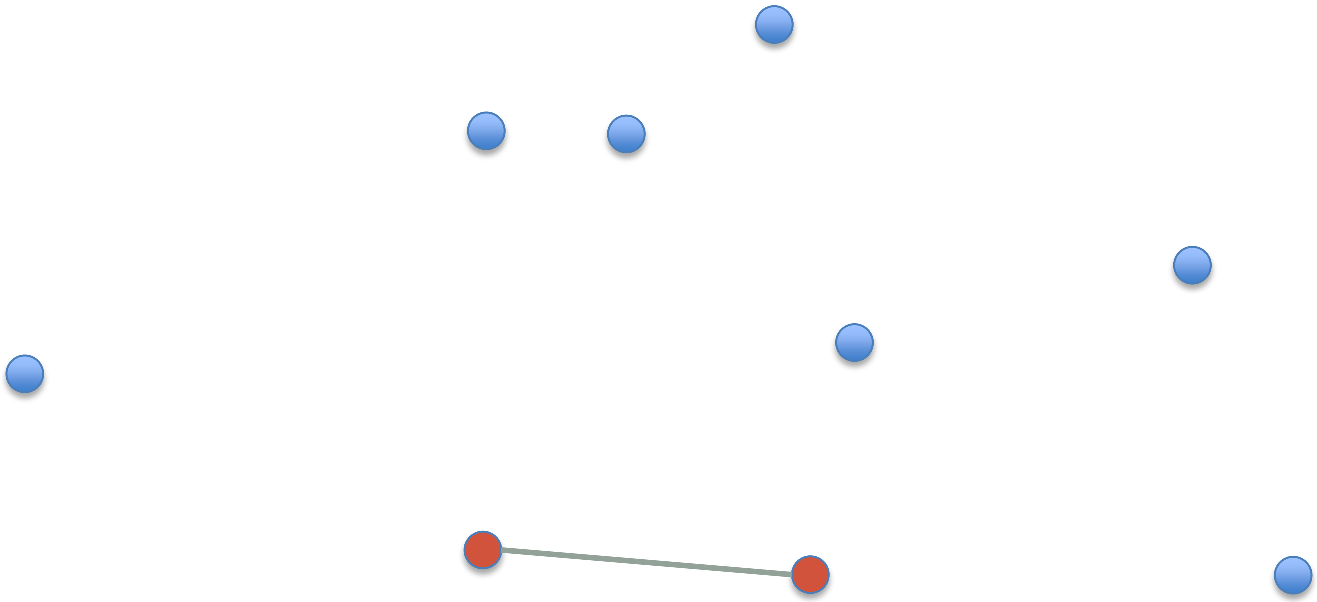
Prim’s Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



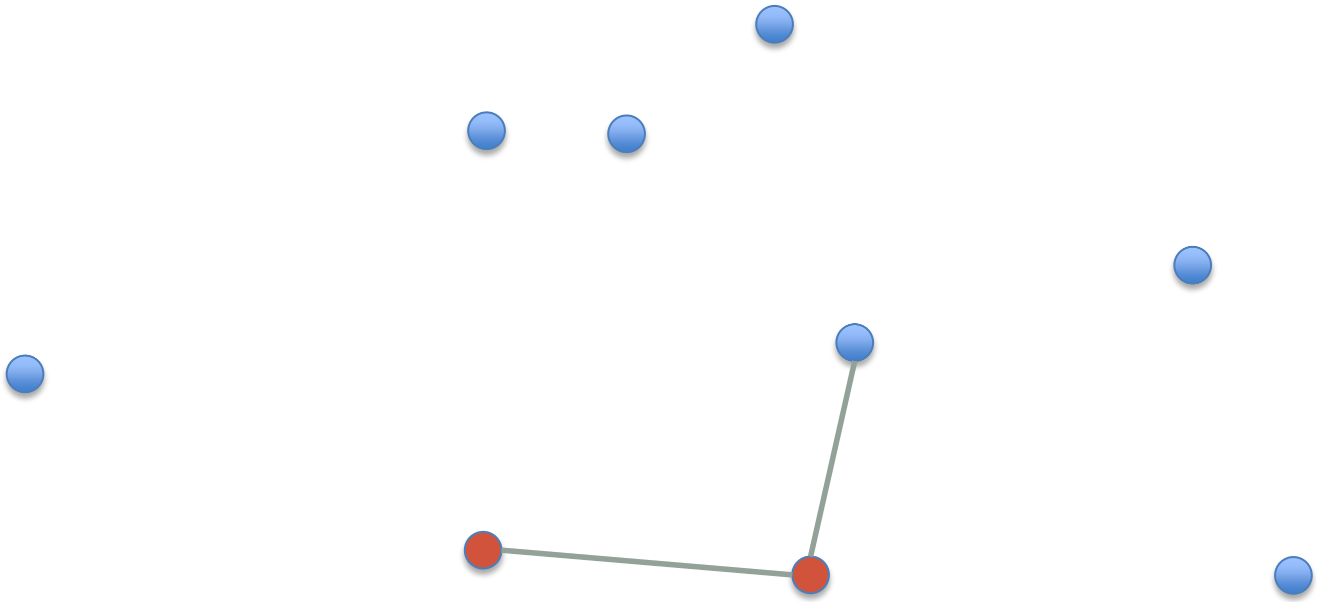
Prim's Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



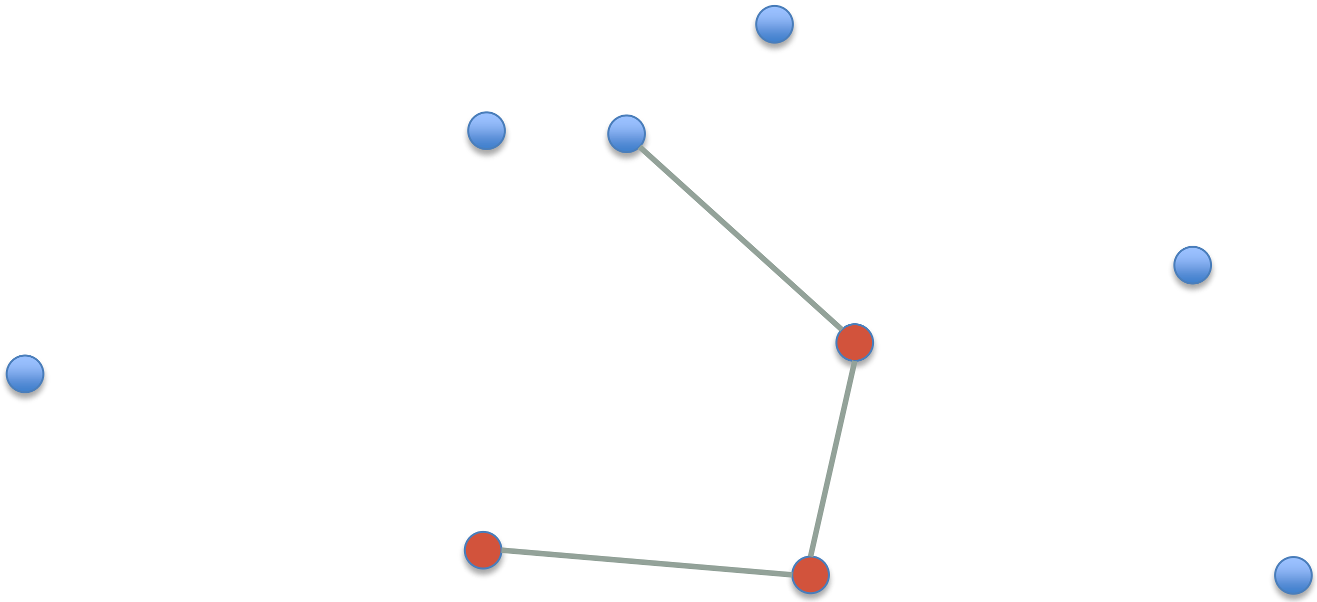
Prim’s Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



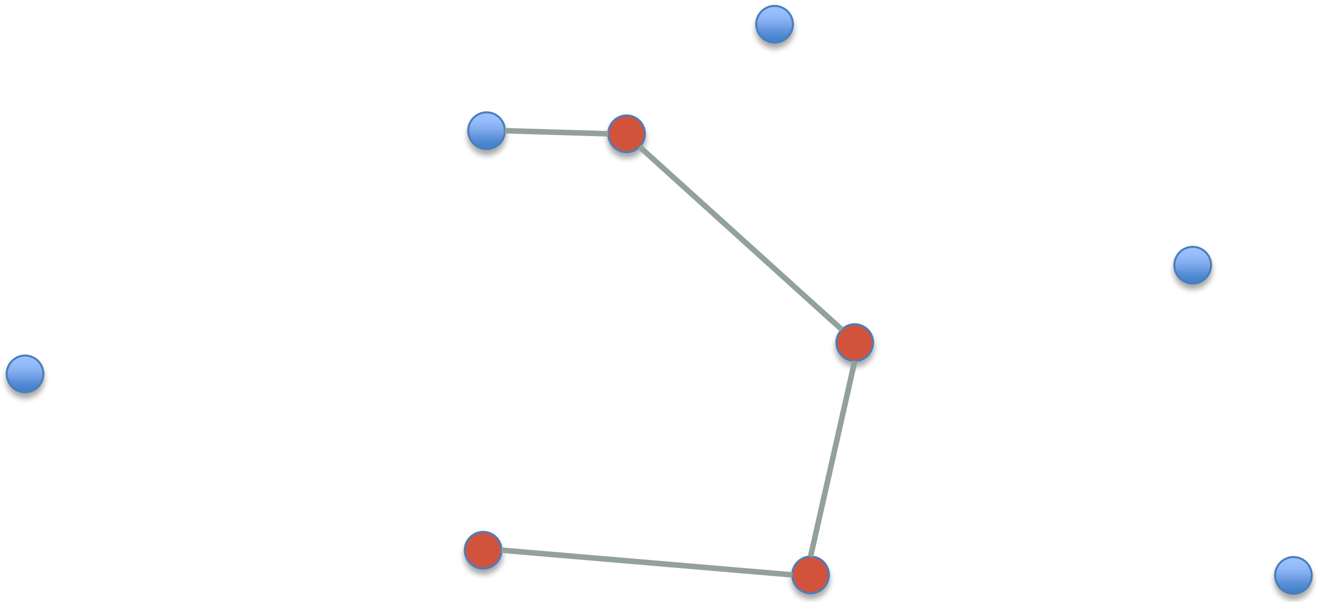
Prim's Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



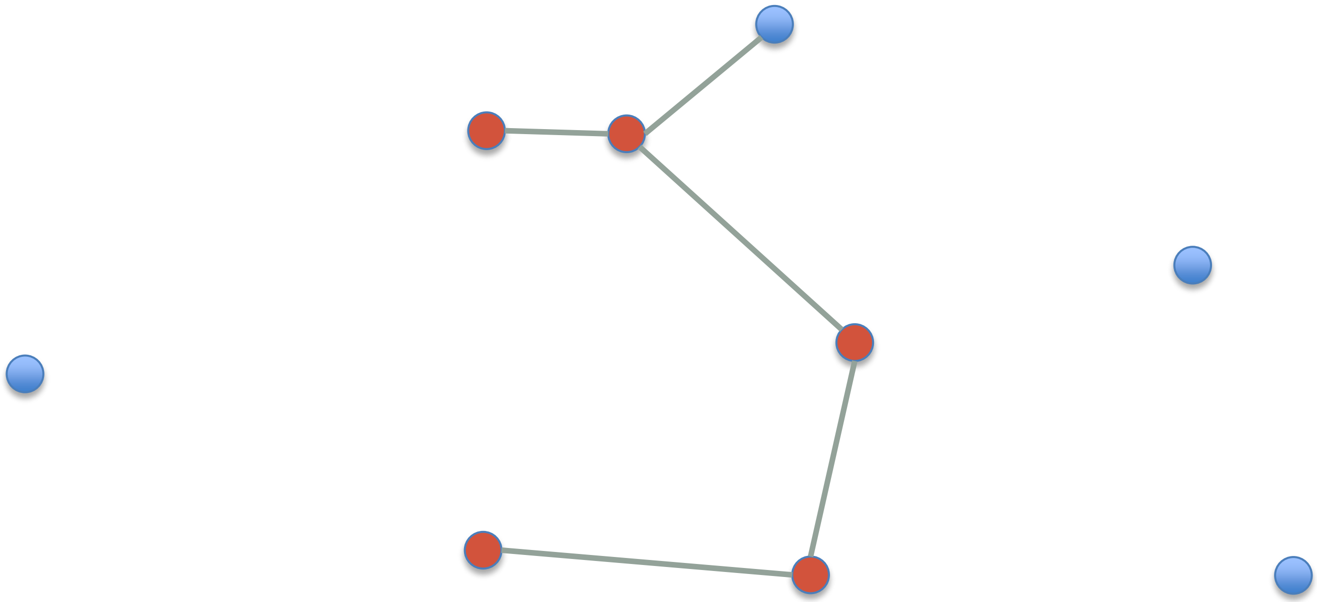
Prim's Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



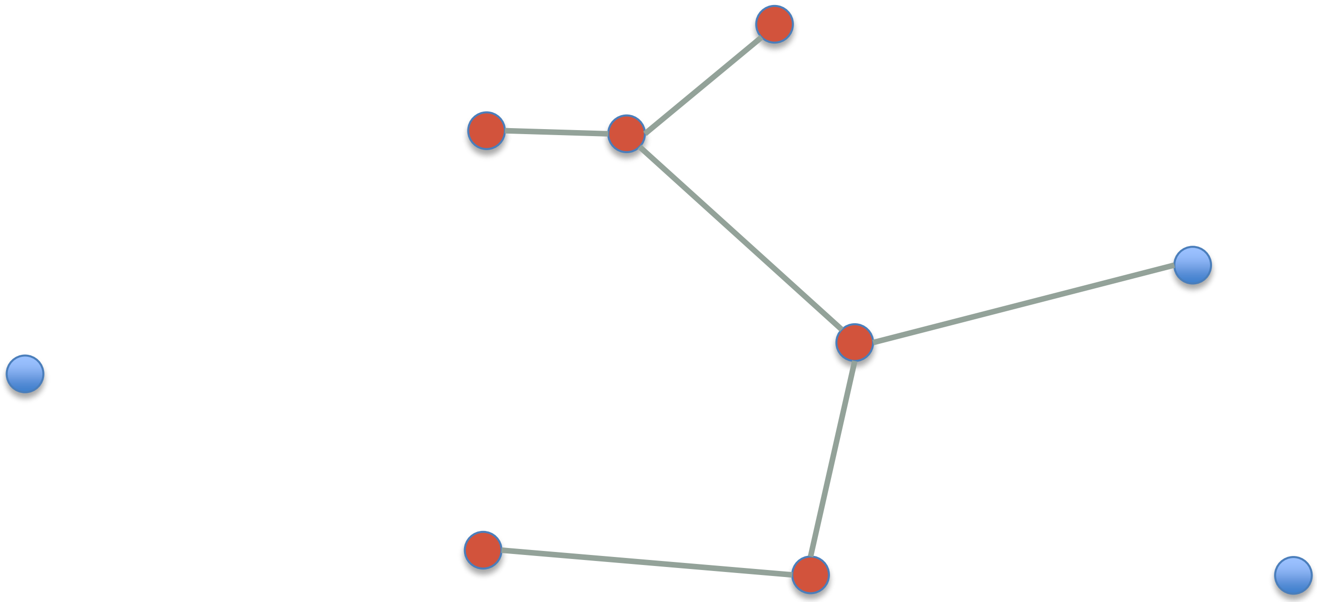
Prim's Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



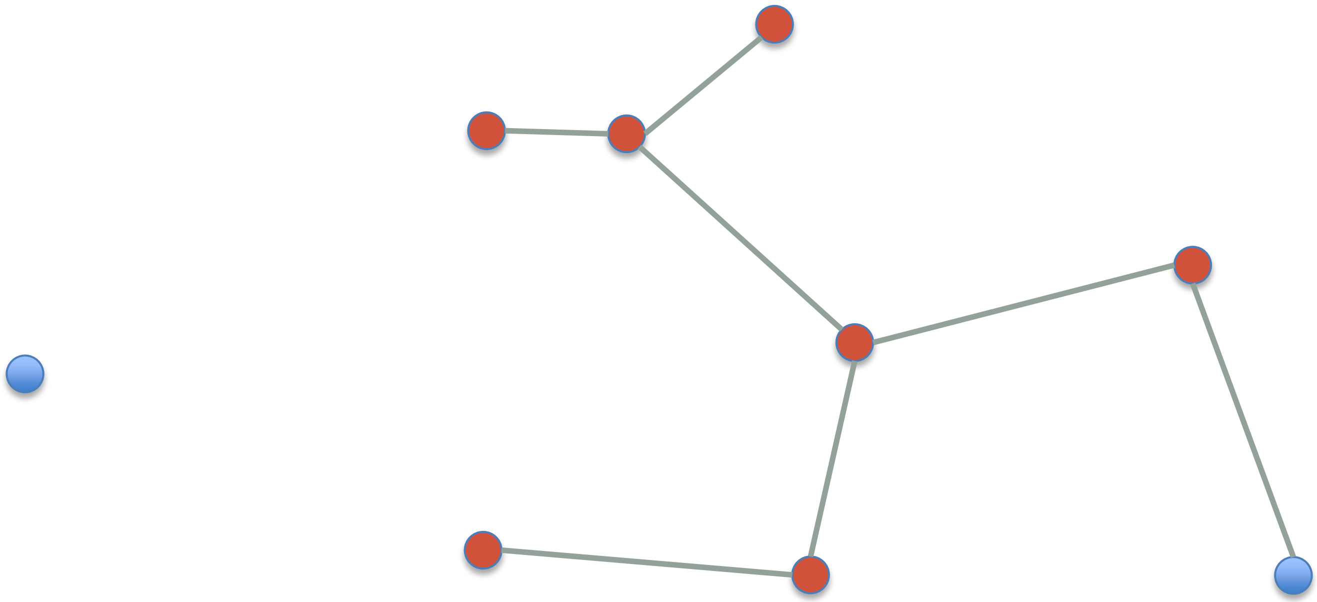
Prim's Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



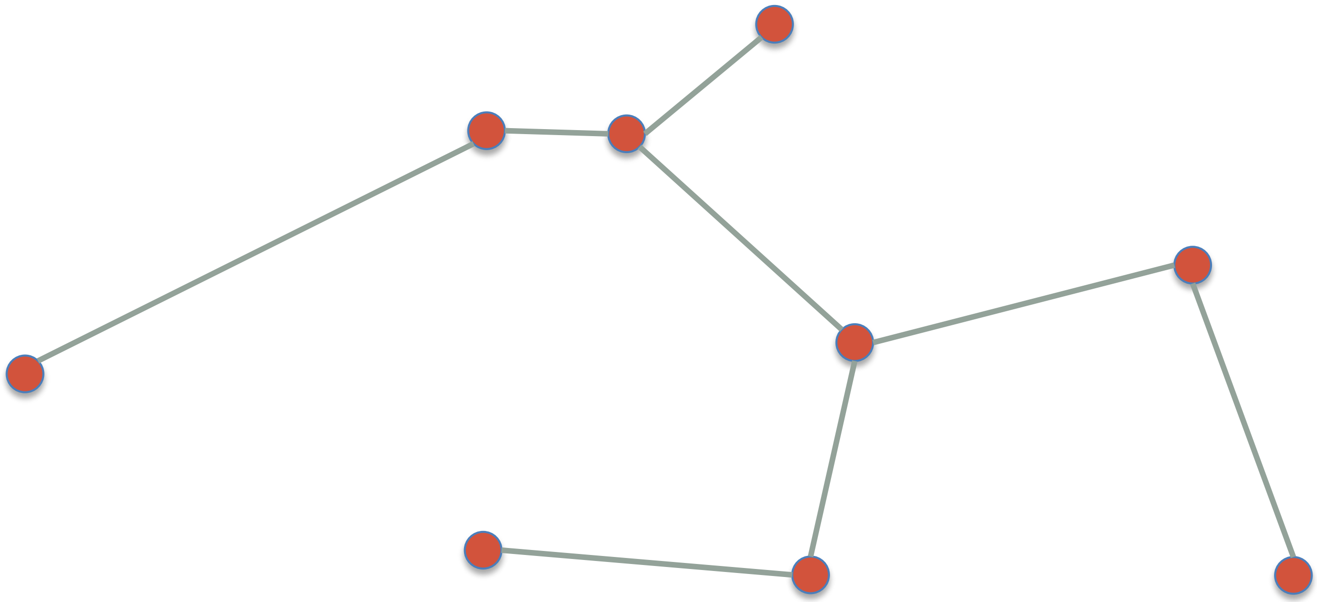
Prim’s Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



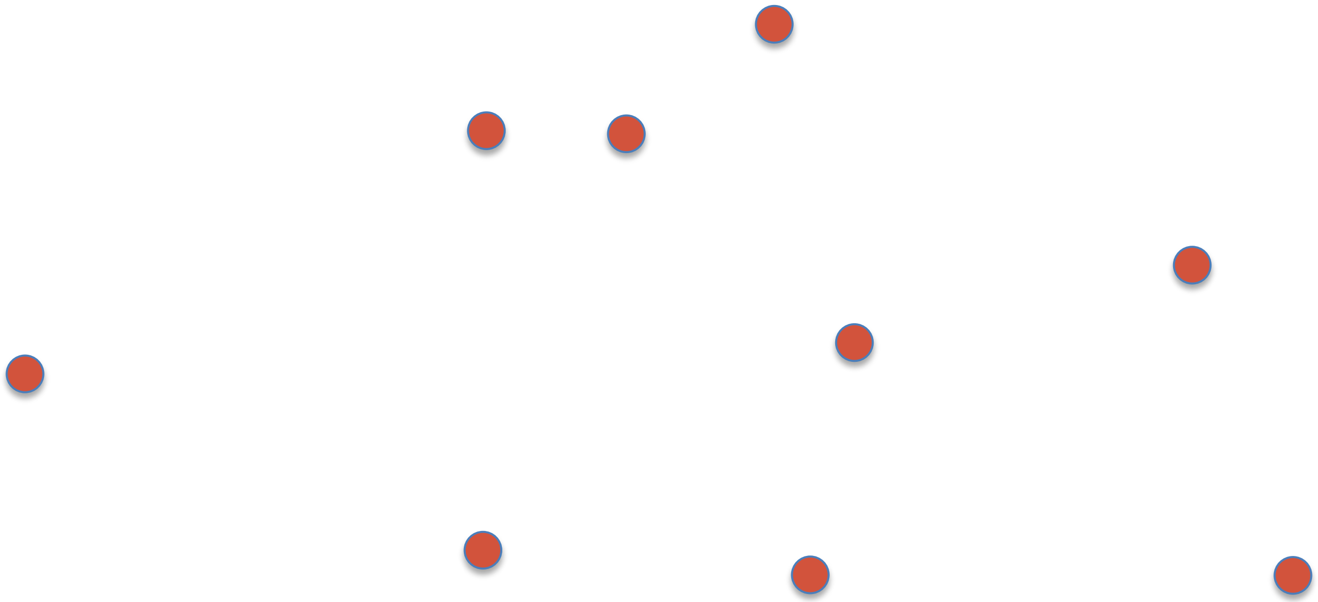
Prim’s Minimum Spanning Tree (MST) algorithm:

Start with any point. Keep adding the point that is nearest to the already chosen points.



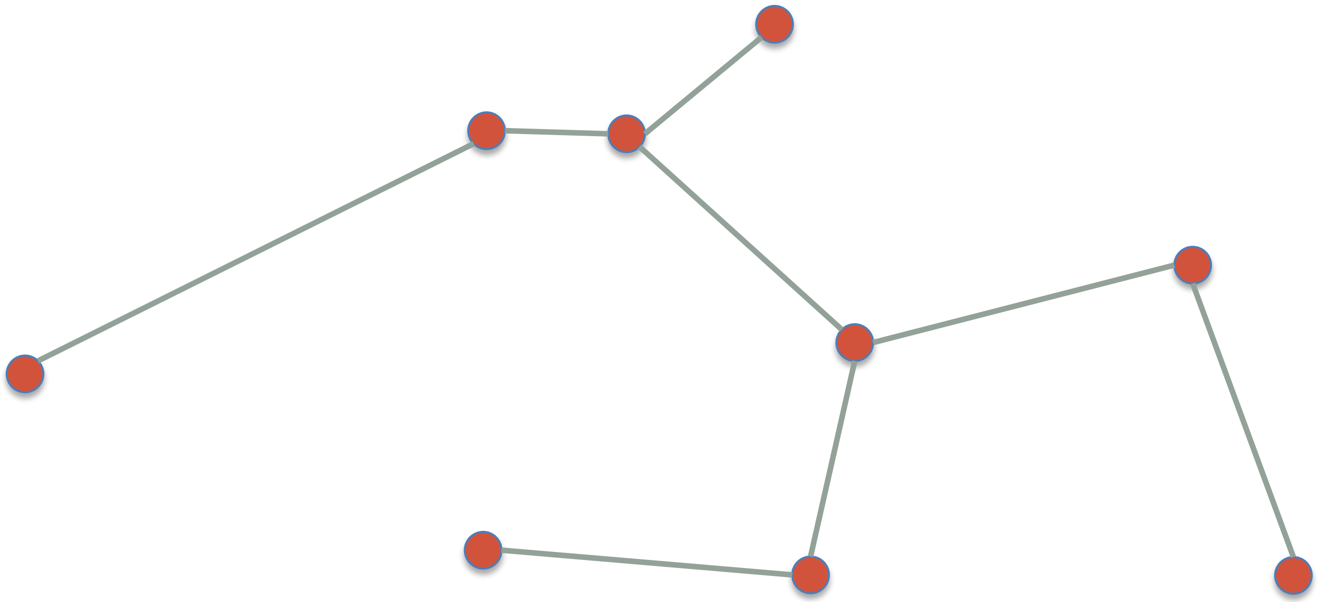
Algorithm Double Tree:

1. Find an MST
2. Double the edges
3. Find an Euler tour
4. Cut short



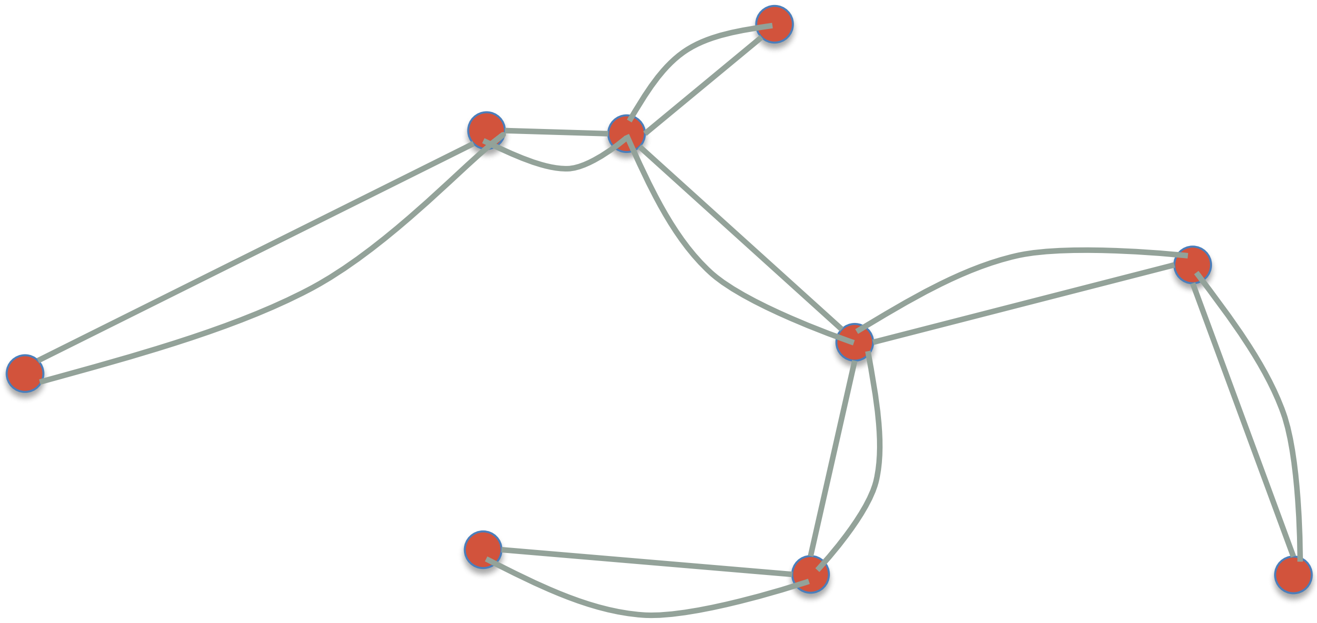
Algorithm Double Tree:

1. Find an MST
2. Double the edges
3. Find an Euler tour
4. Cut short



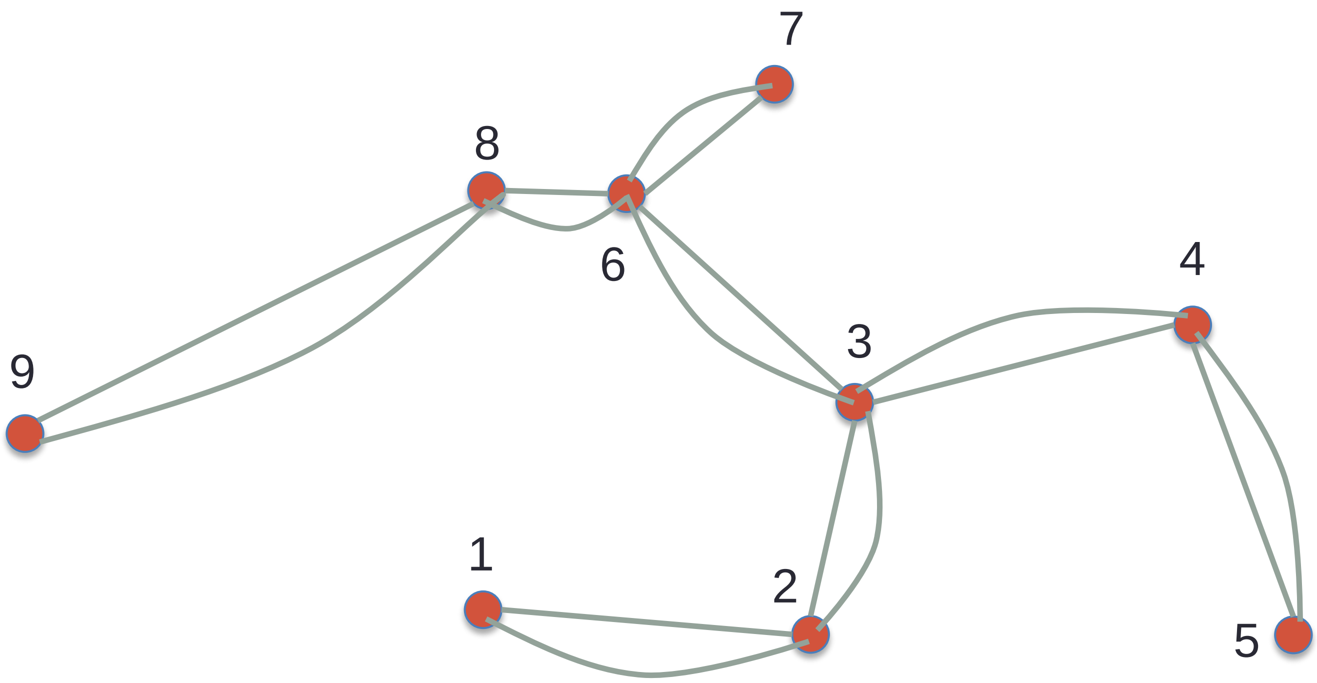
Algorithm Double Tree:

1. Find an MST
2. Double the edges
3. Find an Euler tour
4. Cut short



Algorithm Double Tree:

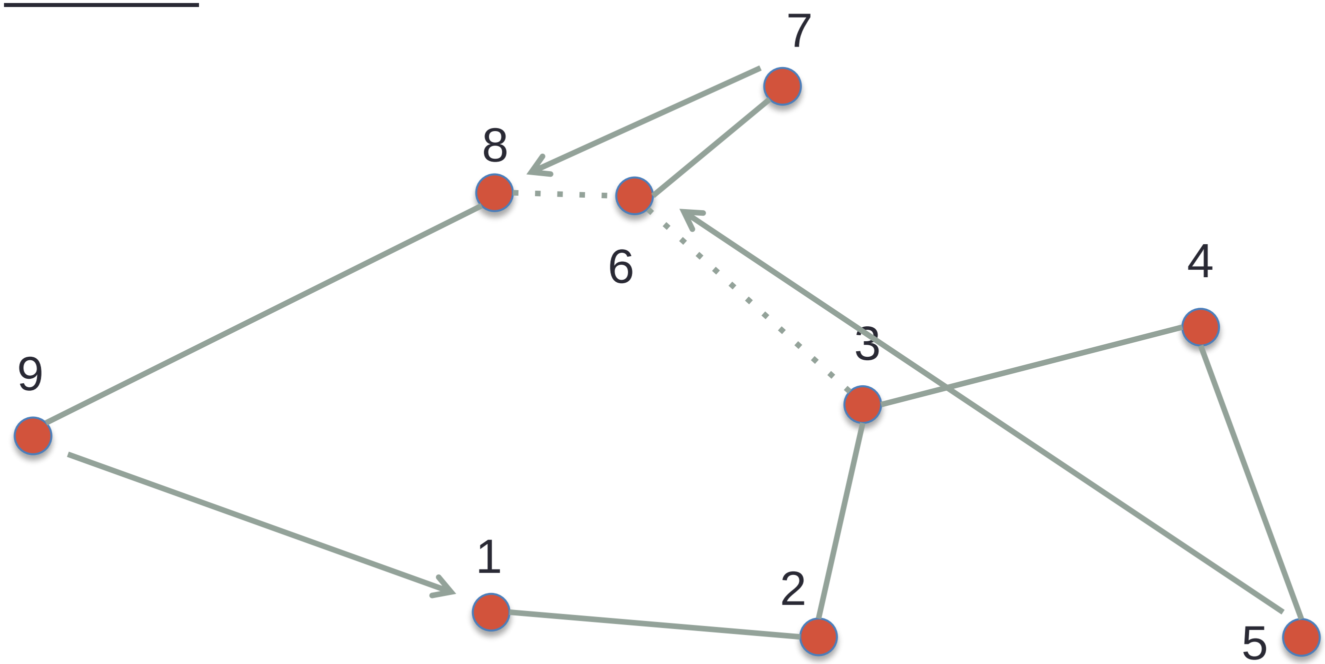
1. Find an MST
2. Double the edges
3. Find an Euler tour
4. Cut short



Euler tour: 1,2,3,4,5,4,3,6,7,6,8,9,8,6,3,2,1

Algorithm Double Tree:

1. Find an MST
2. Double the edges
3. Find an Euler tour
4. <u>Cut short</u>



TSP tour: 1,2,3,4,5,~~4,3~~,6,7,~~6~~,8,9,~~8,6,3,2~~,1

Theorem Double Tree is a 2-approximation algorithm

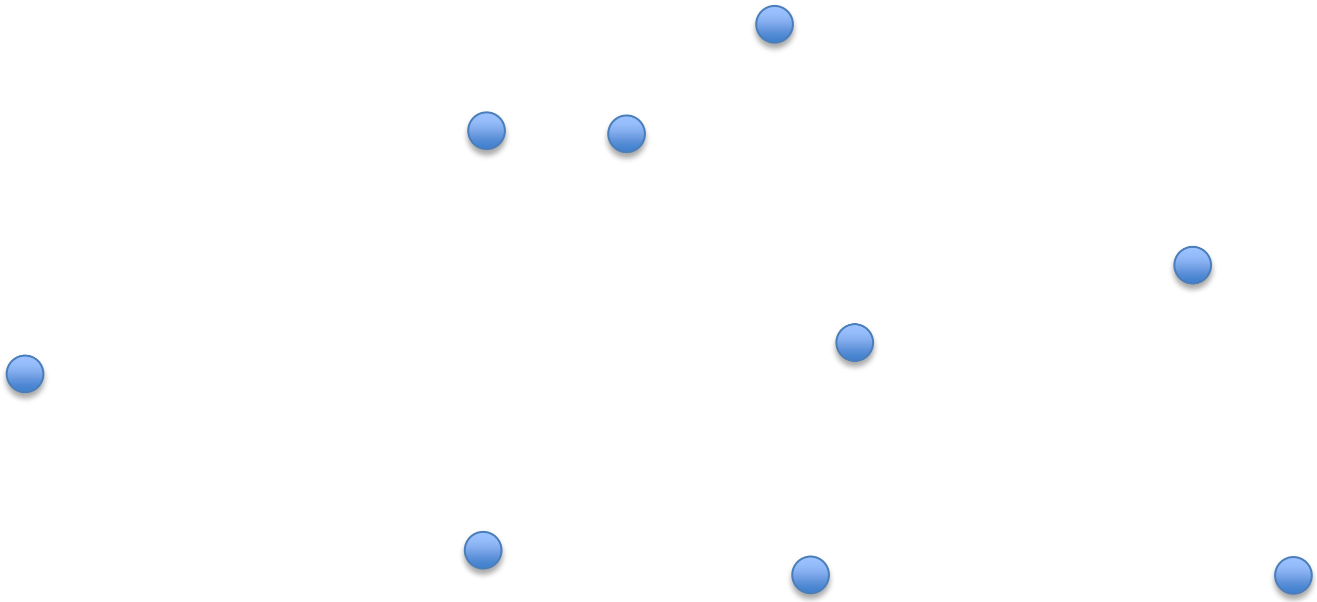
Proof [1] running time. OK
[2] Feasible OK
[3] Ratio:?

> **Lemma**
> Let T be an MST and OPT the value of the smallest TSP tour.
> Then cost(T) < OPT.
>
> **Proof** Take an optimal TSP tour and remove one edge. This gives a spanning tree. The cost of this tree is no more than cost(T).

Cost of the Eulertour is 2cost(T) < 2OPT. Shortcutting does not increase the length since triangle inequality holds.

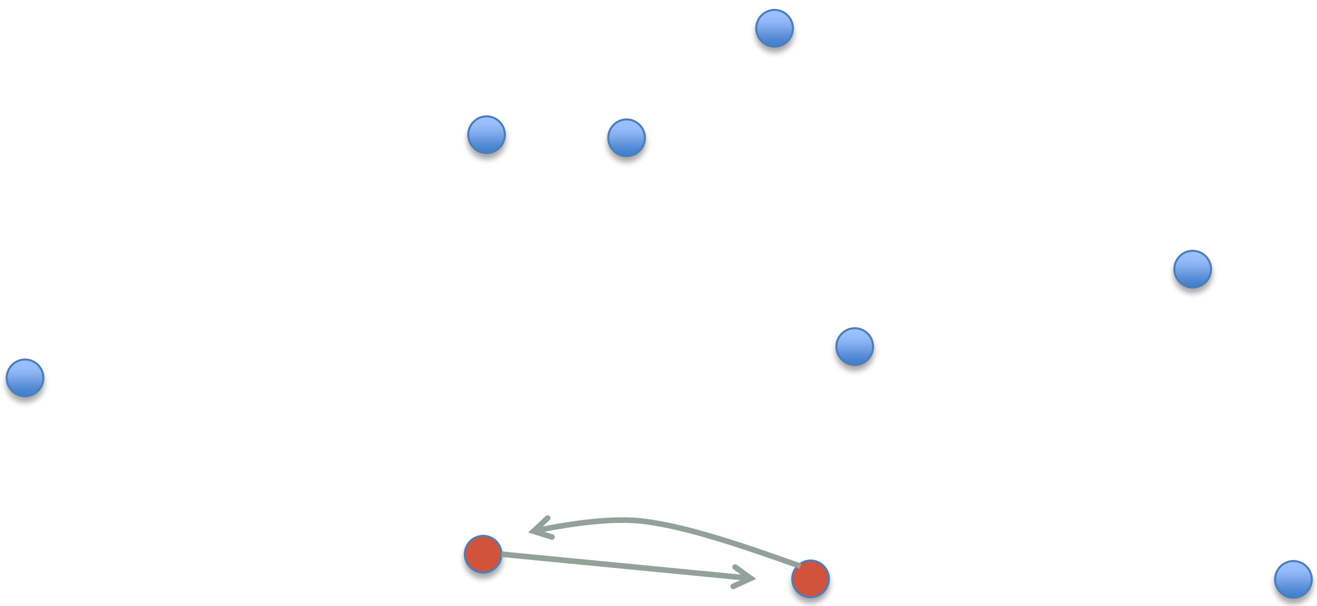
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



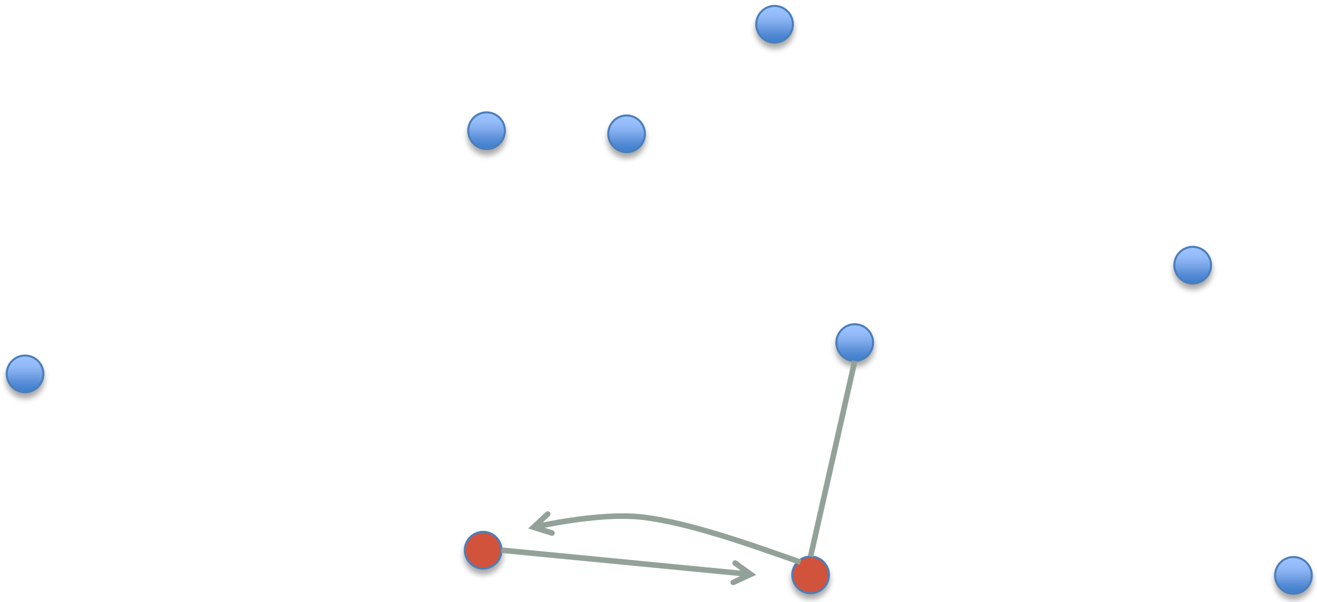
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



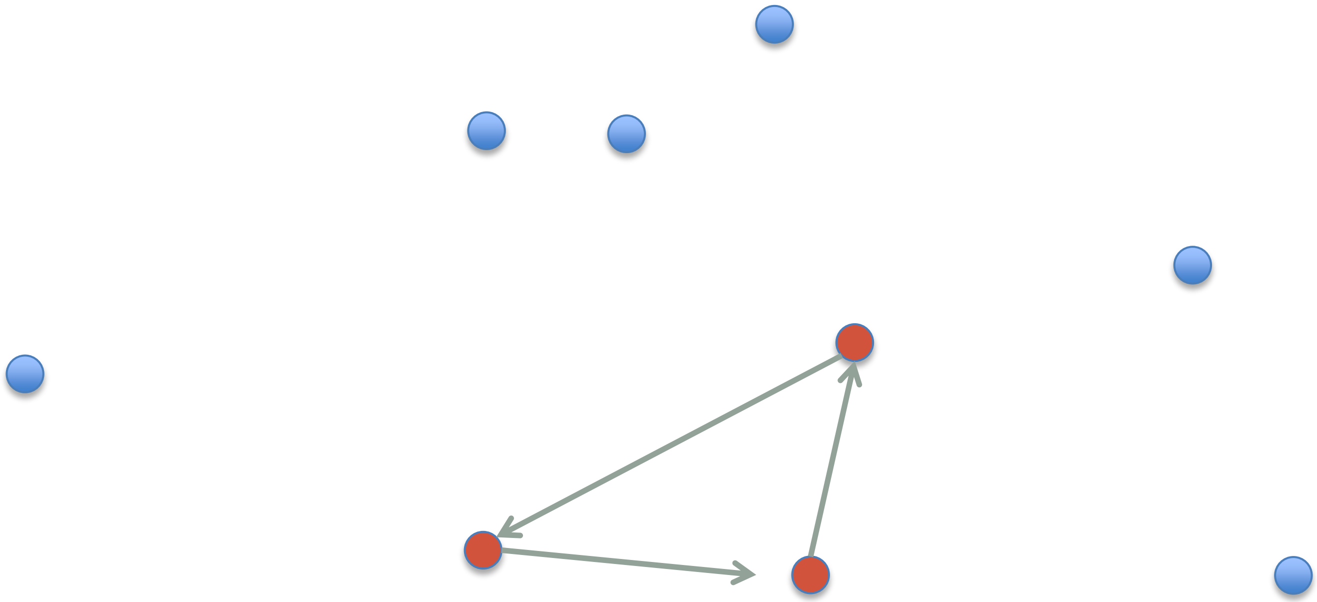
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



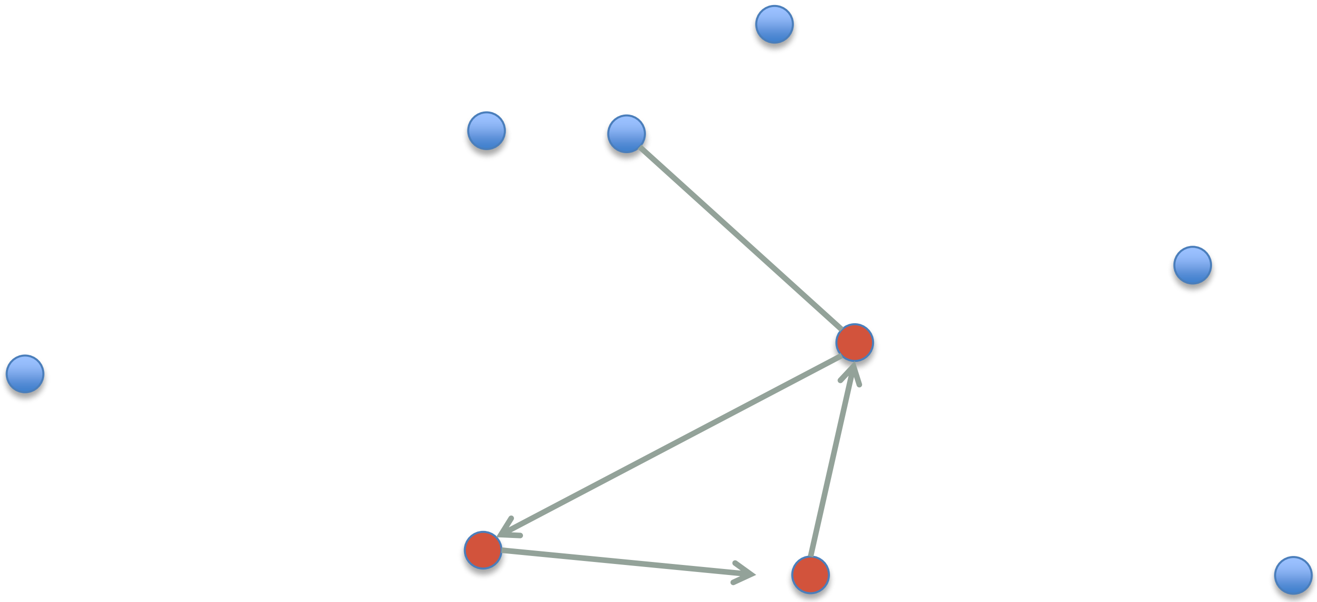
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



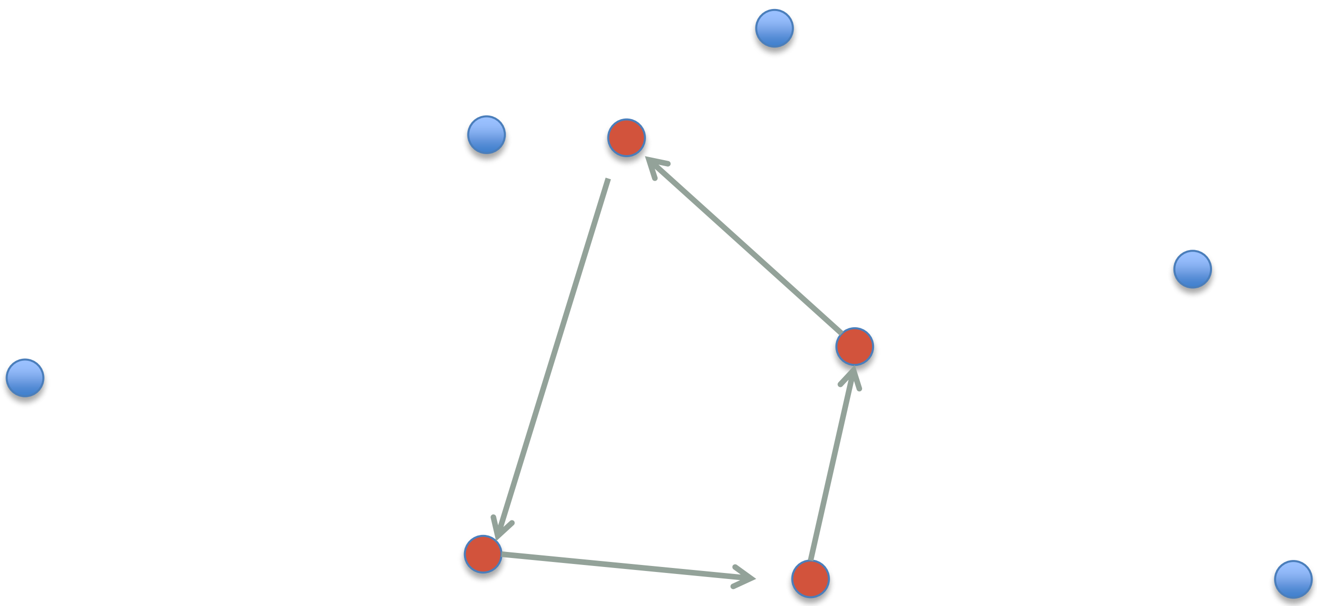
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



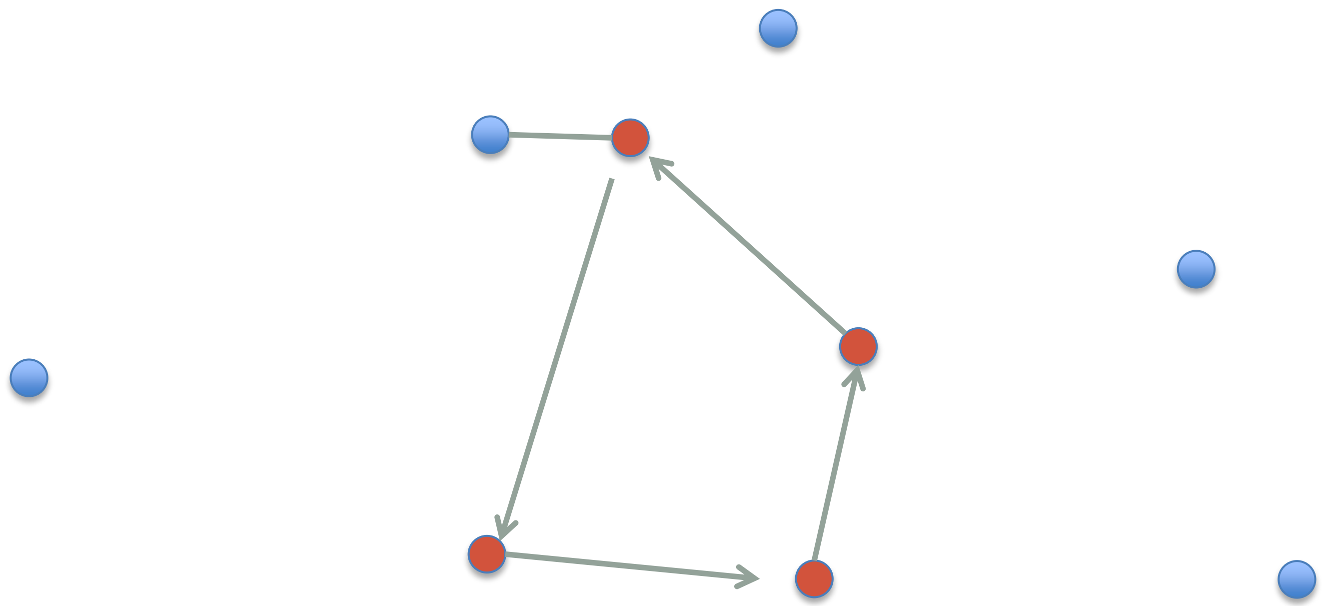
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



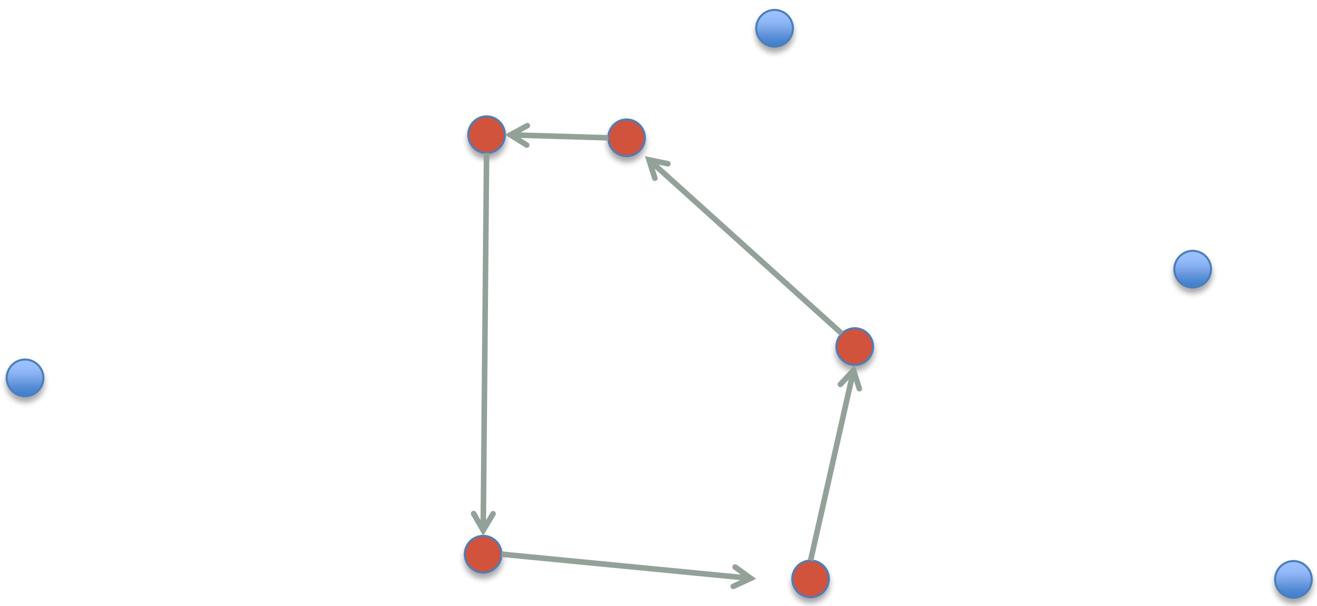
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



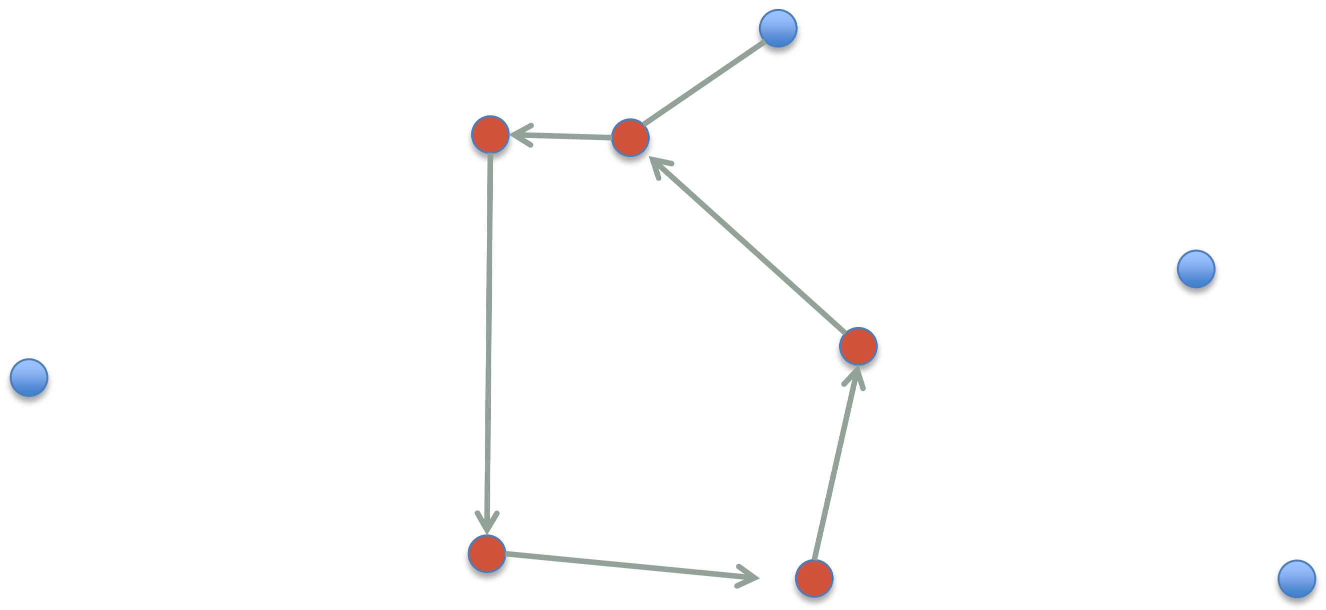
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



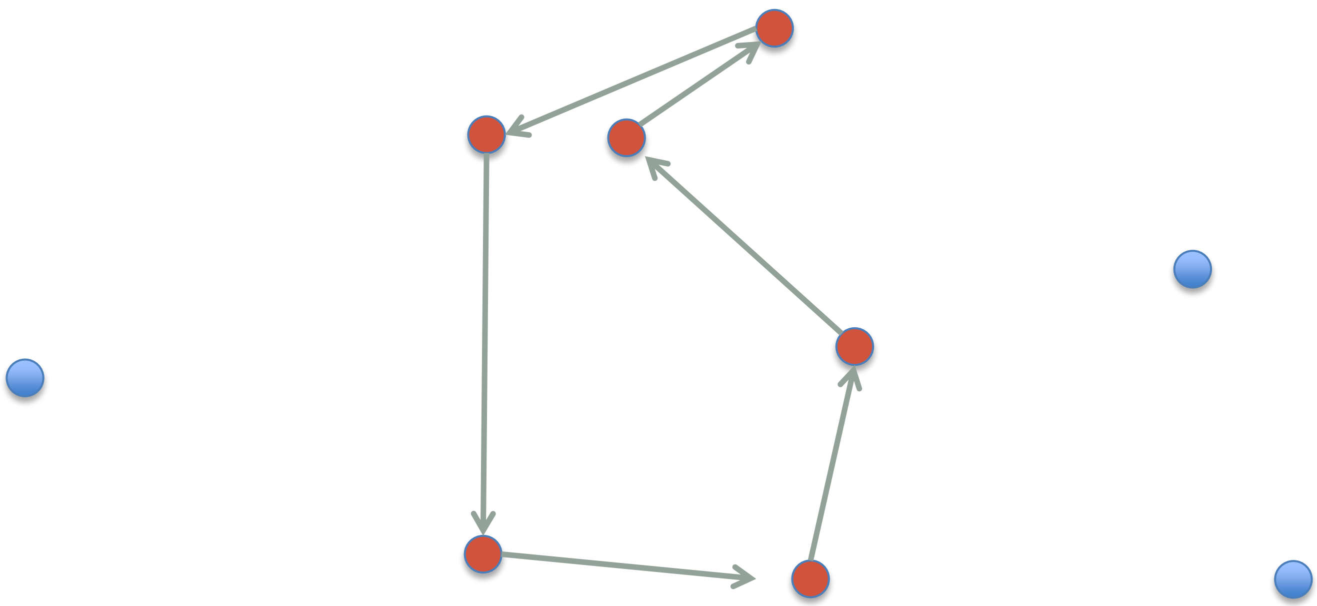
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



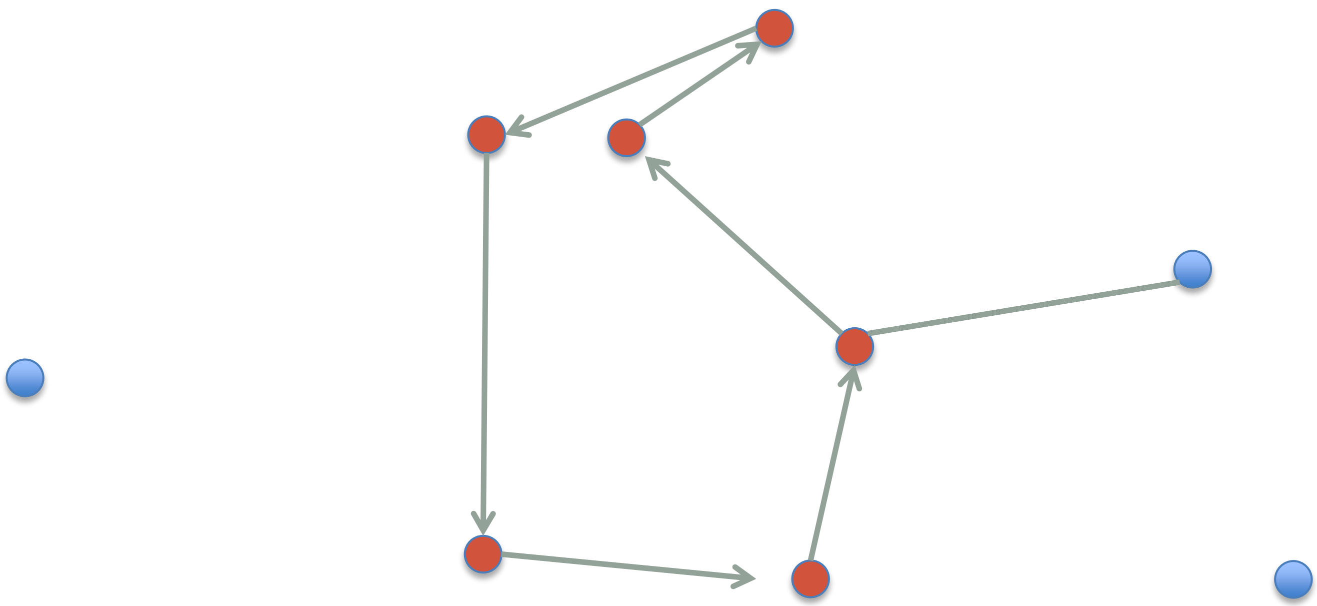
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



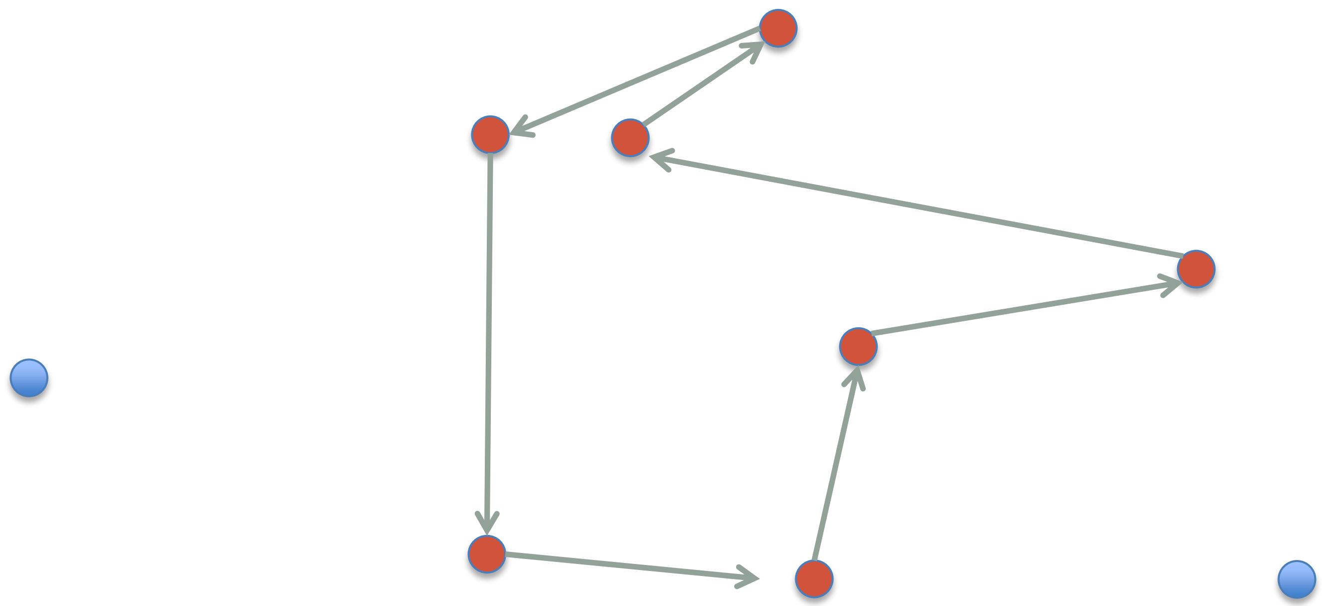
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



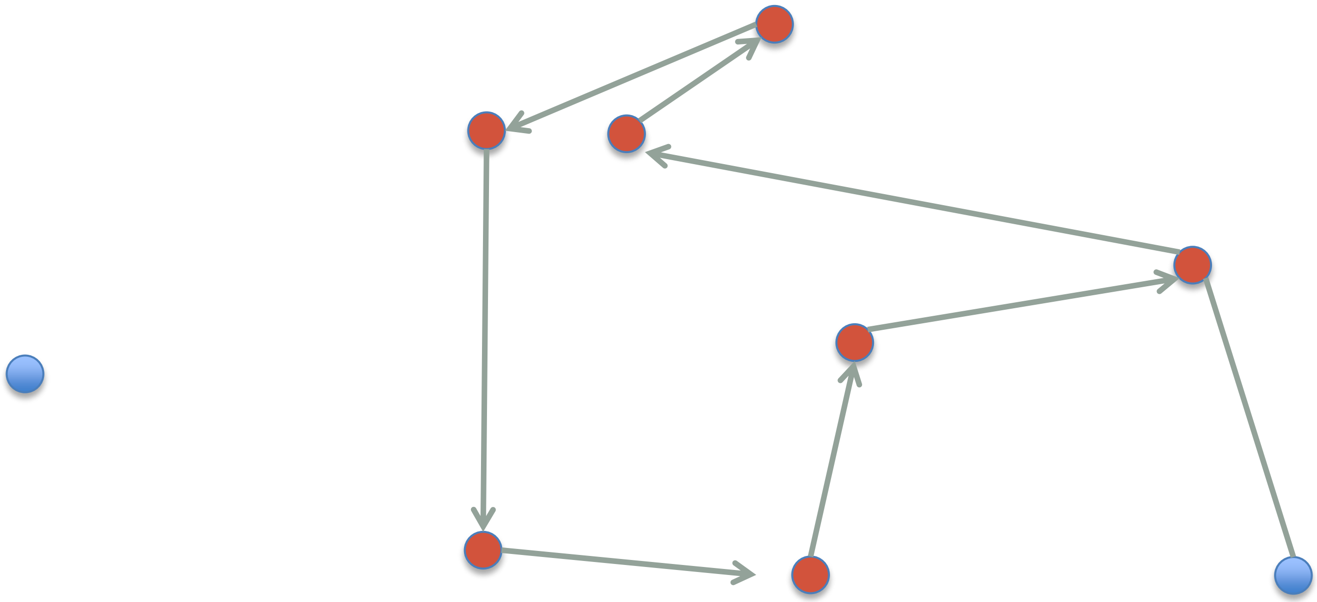
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



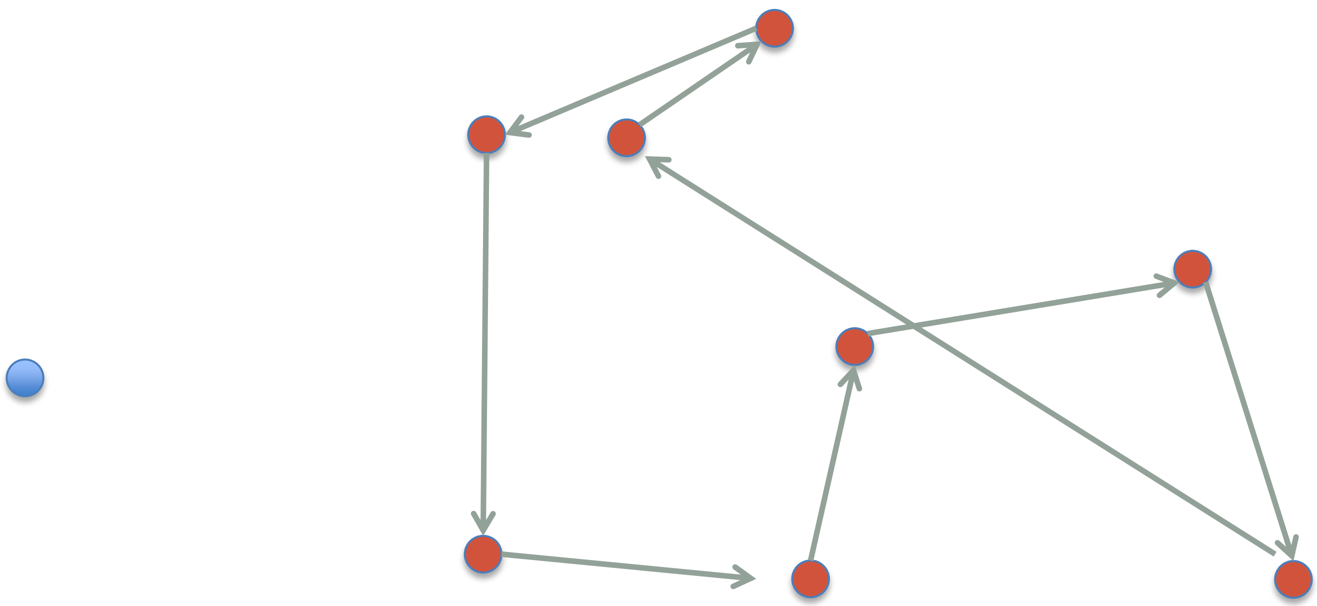
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



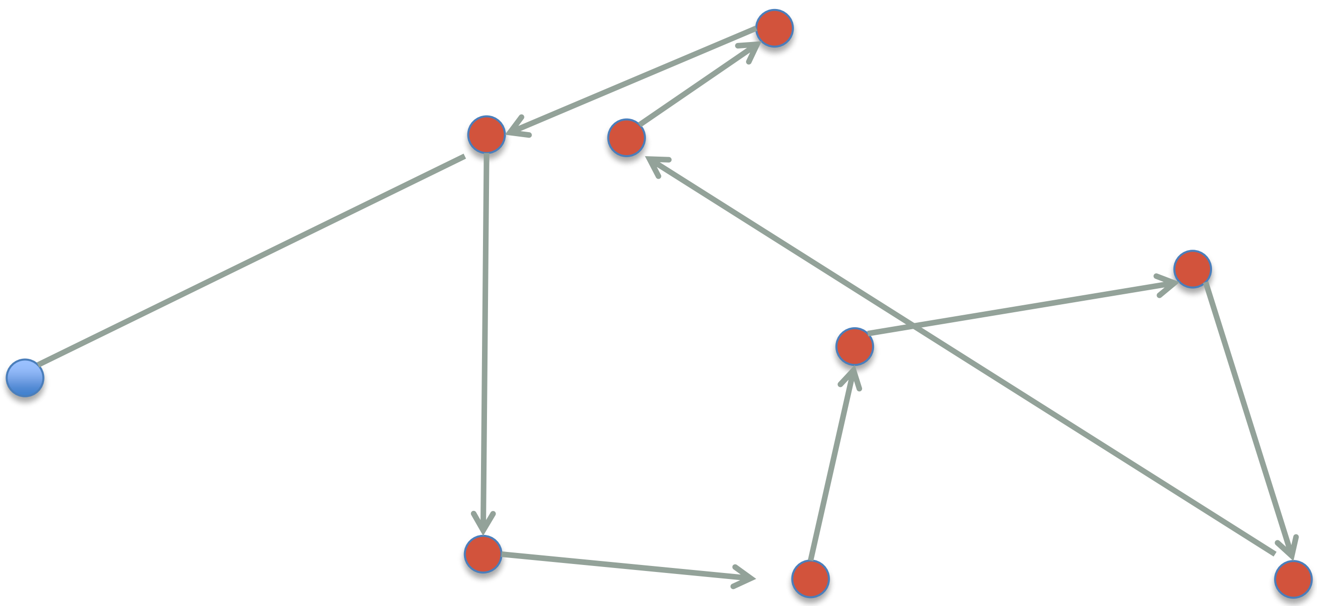
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



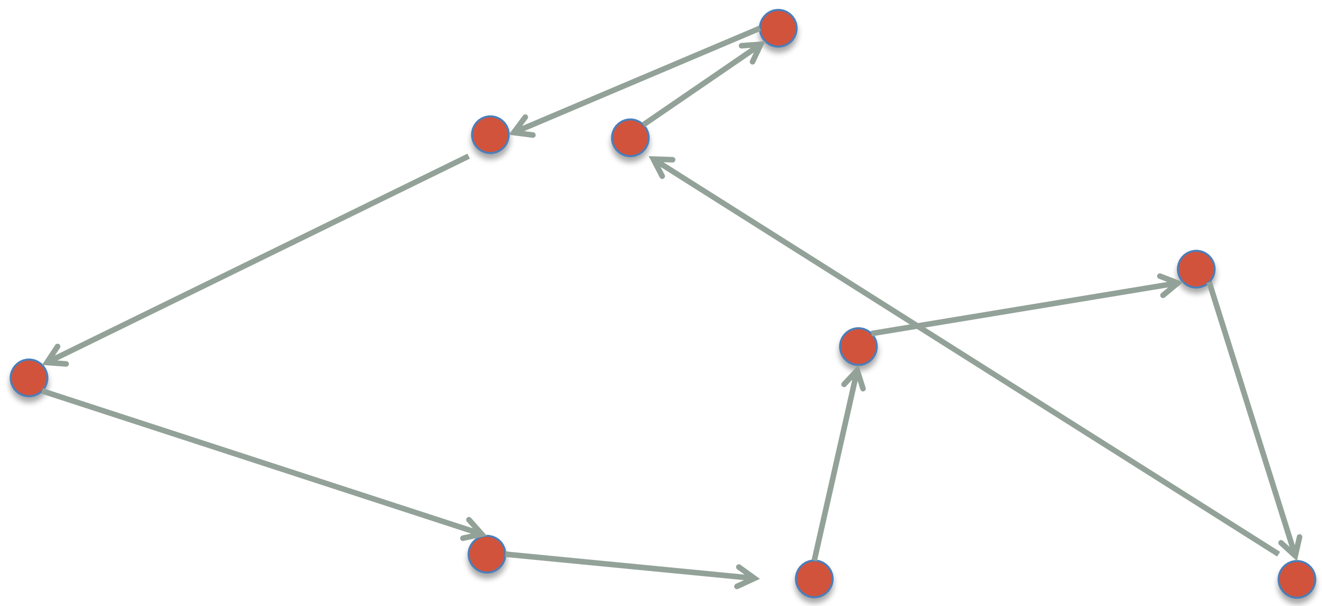
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



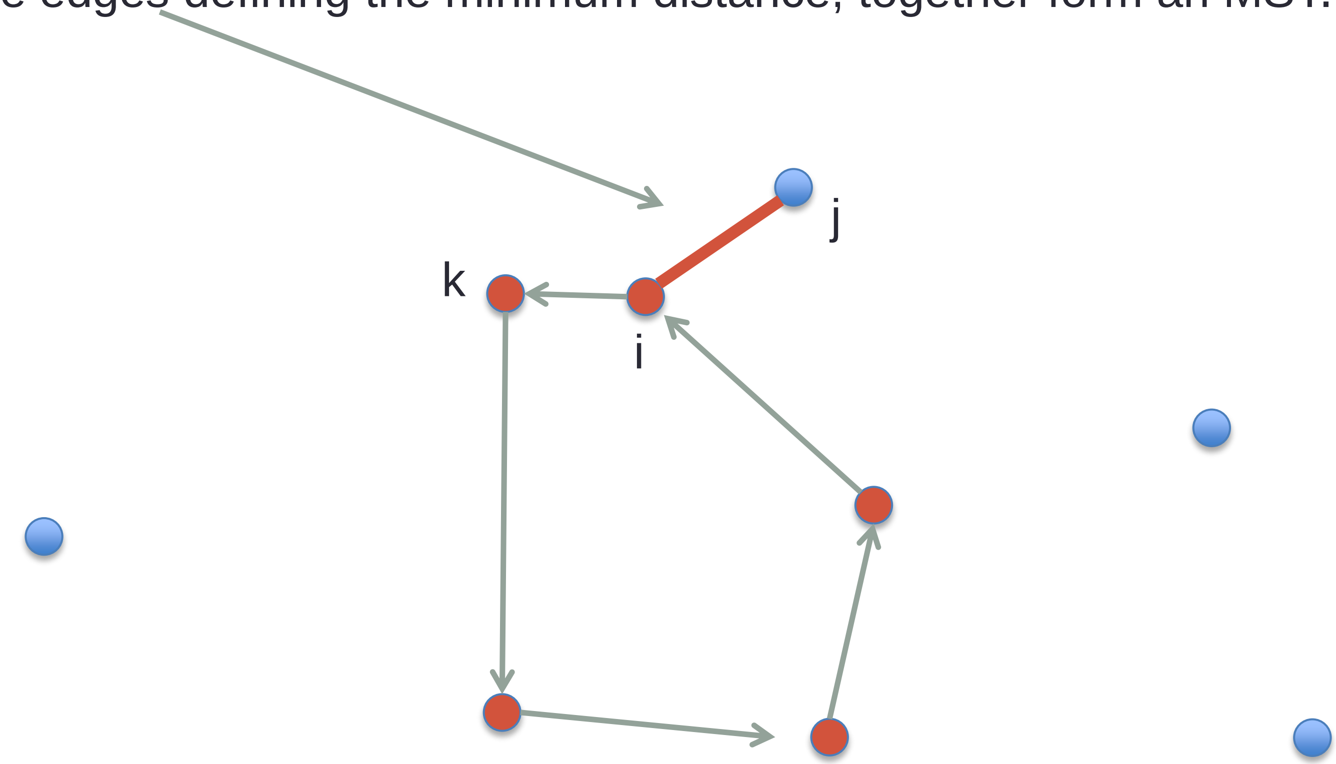
Nearest addition:

1. Start with a tour on 2 points
2. Keep adding the nearest point.



Theorem Nearest Addition is a 2-approximation algorithm

Proof [1] running time. OK
[2] Feasible OK
[3] Ratio:
The algorithm behaves exactly like Prim's MST algorithm:
The edges defining the minimum distance, together form an MST.

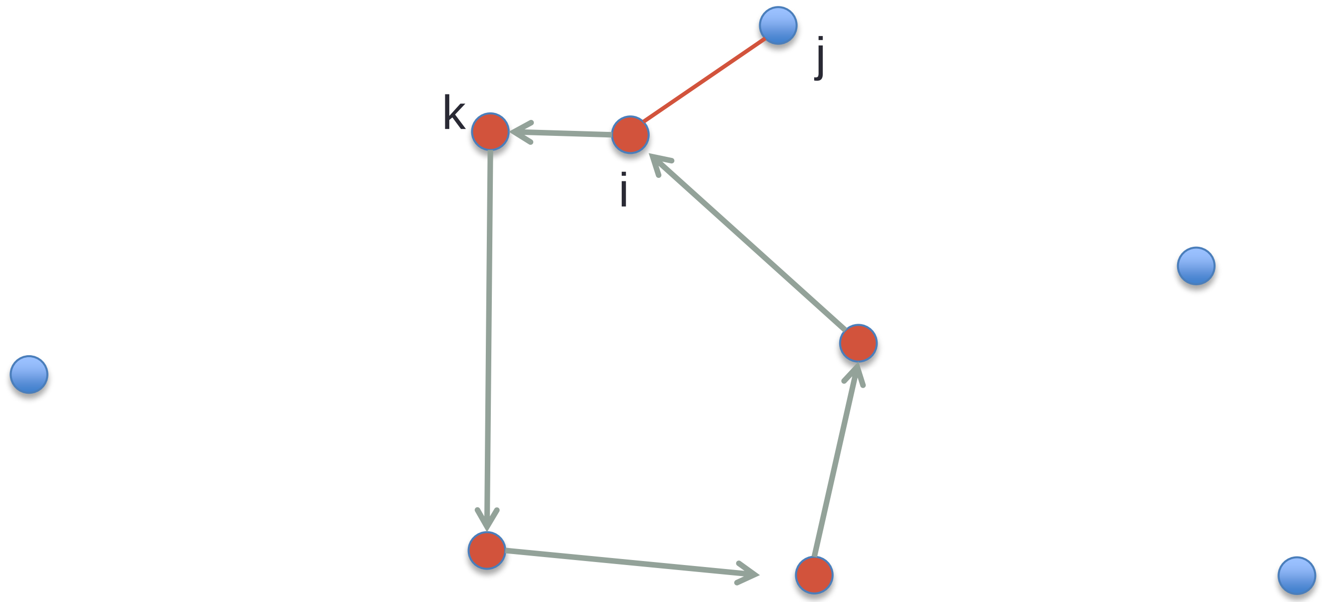


Theorem Nearest Addition is a 2-approximation algorithm

Proof [1] running time. OK
[2] Feasible OK
[3] Ratio:

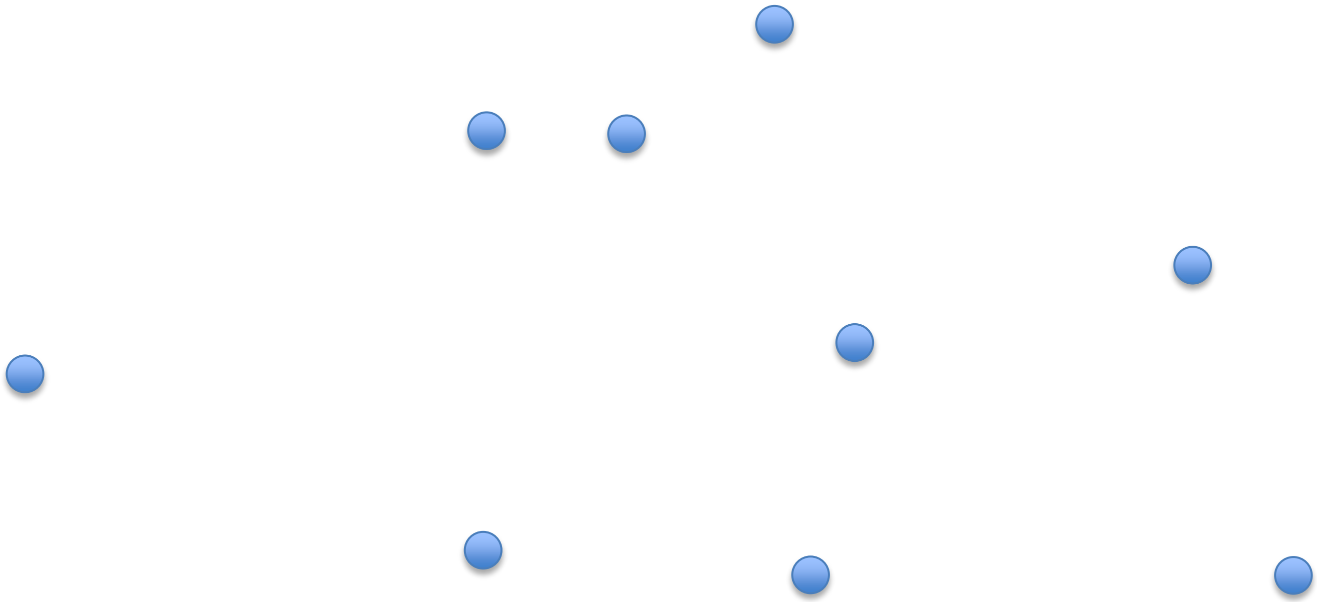
By triangle inequality: $c_{jk} \le c_{ji} + c_{ik} \rightarrow c_{jk} - c_{ik} \le c_{ji}$

Cost in this step: $c_{ij} + c_{jk} - c_{ik} \le 2c_{ij}$. $\rightarrow$ Total cost $\le$ 2cost(MST) $\le$ 2OPT.



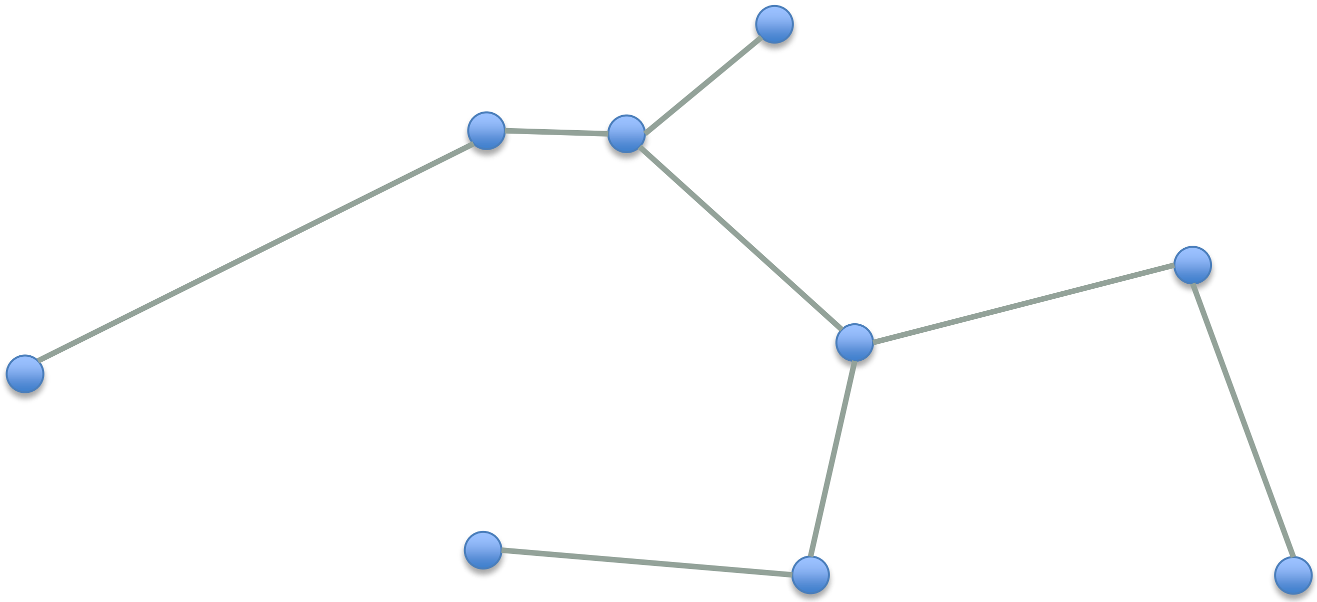
Christofides' Algorithm

1. Find a minimum spanning tree T
2. Find a minimum matching M for the odd-degree vertices in T
3. Add M to T
4. Find an Euler tour
5. Cut short



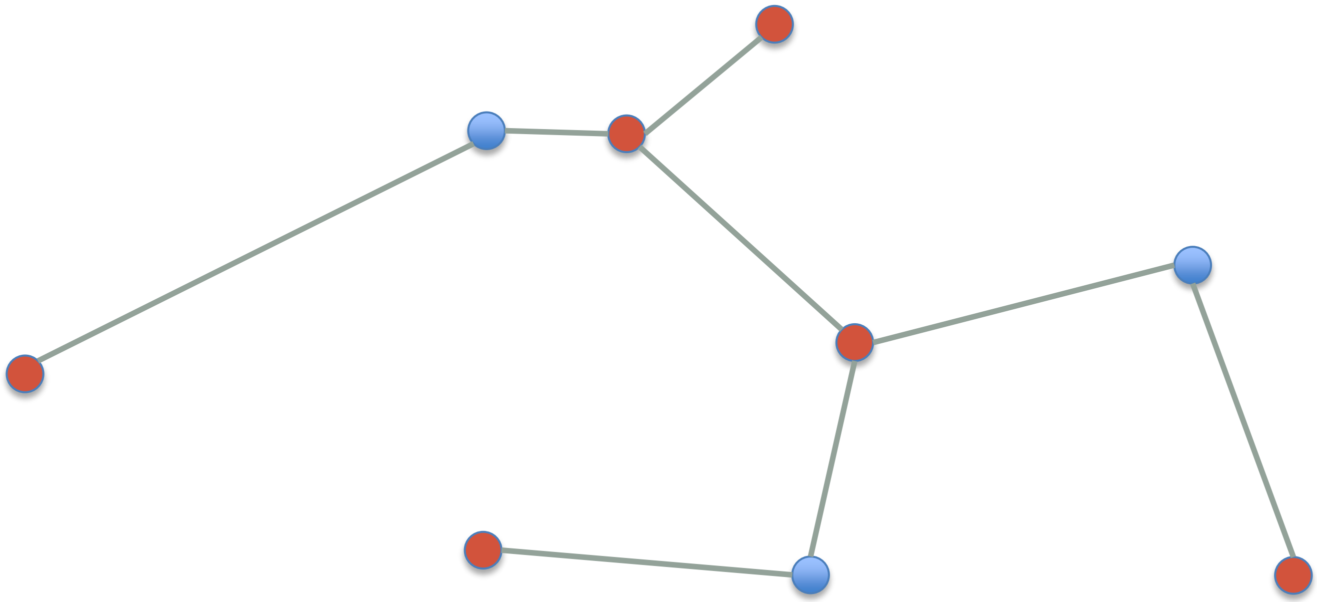
Christofides' Algorithm

1. Find a minimum spanning tree T
2. Find a minimum matching M for the odd-degree vertices in T
3. Add M to T
4. Find an Euler tour
5. Cut short



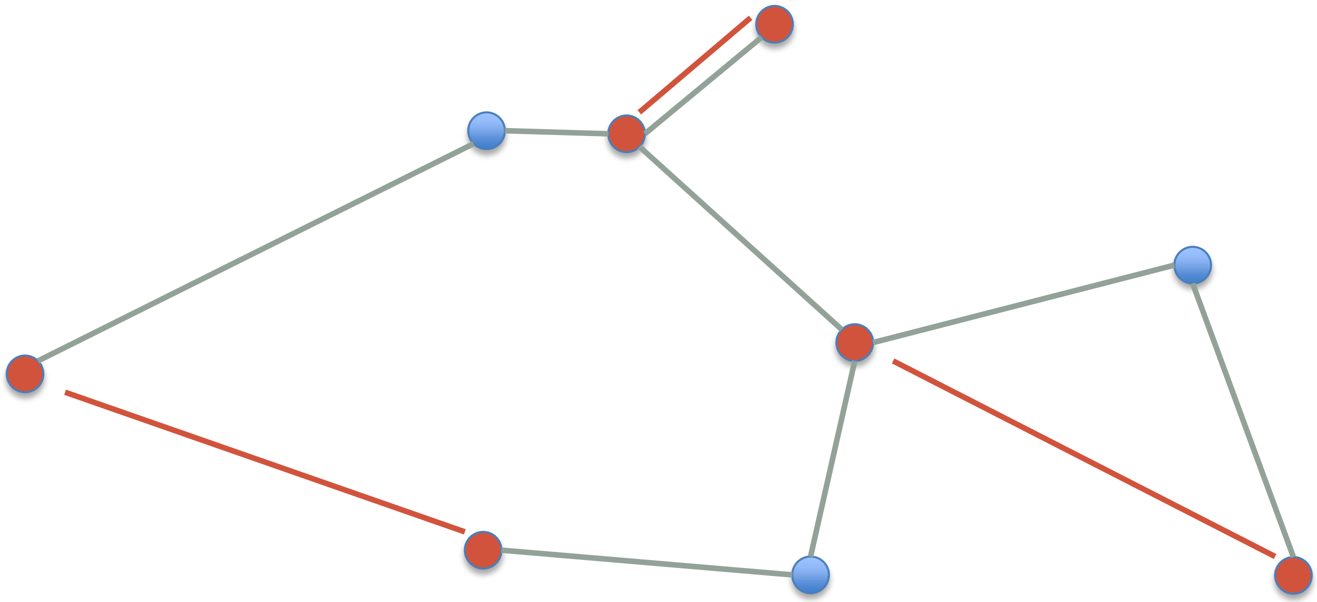
Christofides' Algorithm

1. Find a minimum spanning tree T
2. Find a minimum matching M for the odd-degree vertices in T
3. Add M to T
4. Find an Euler tour
5. Cut short



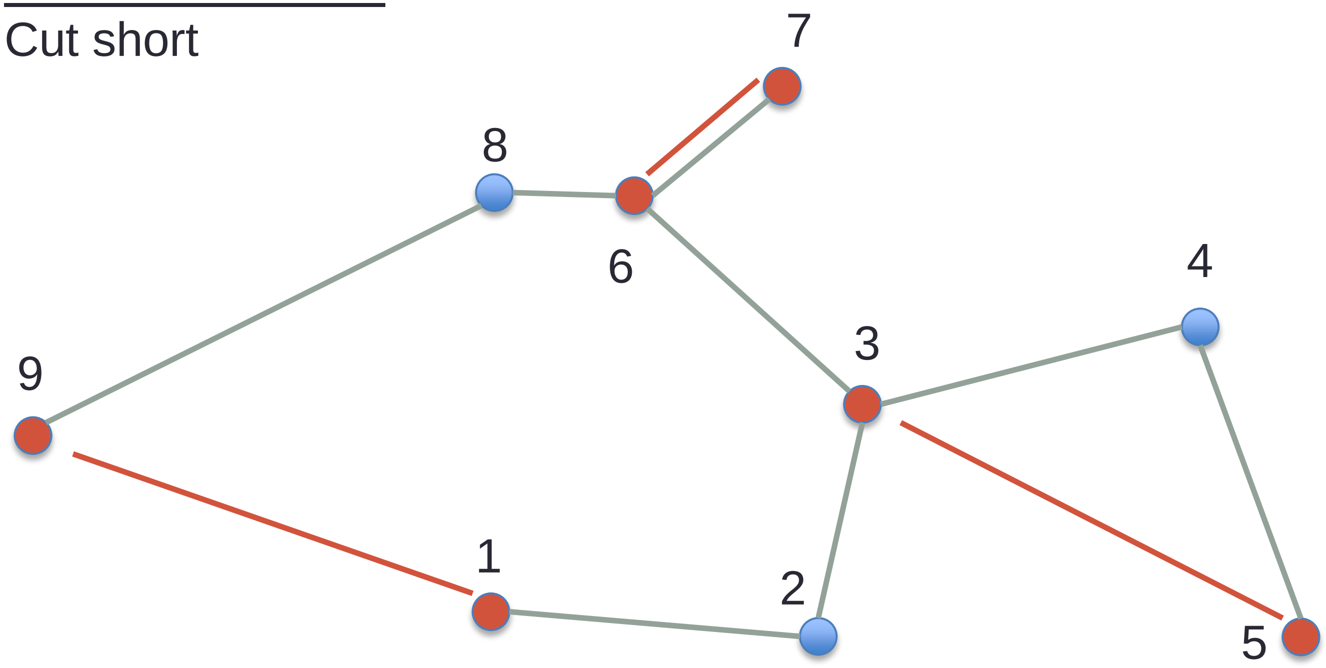
Christofides' Algorithm

1. Find a minimum spanning tree T
2. <u>Find a minimum matching M for the odd-degree vertices in T</u>
3. <u>Add M to T</u>
4. Find an Euler tour
5. Cut short



Christofides' Algorithm

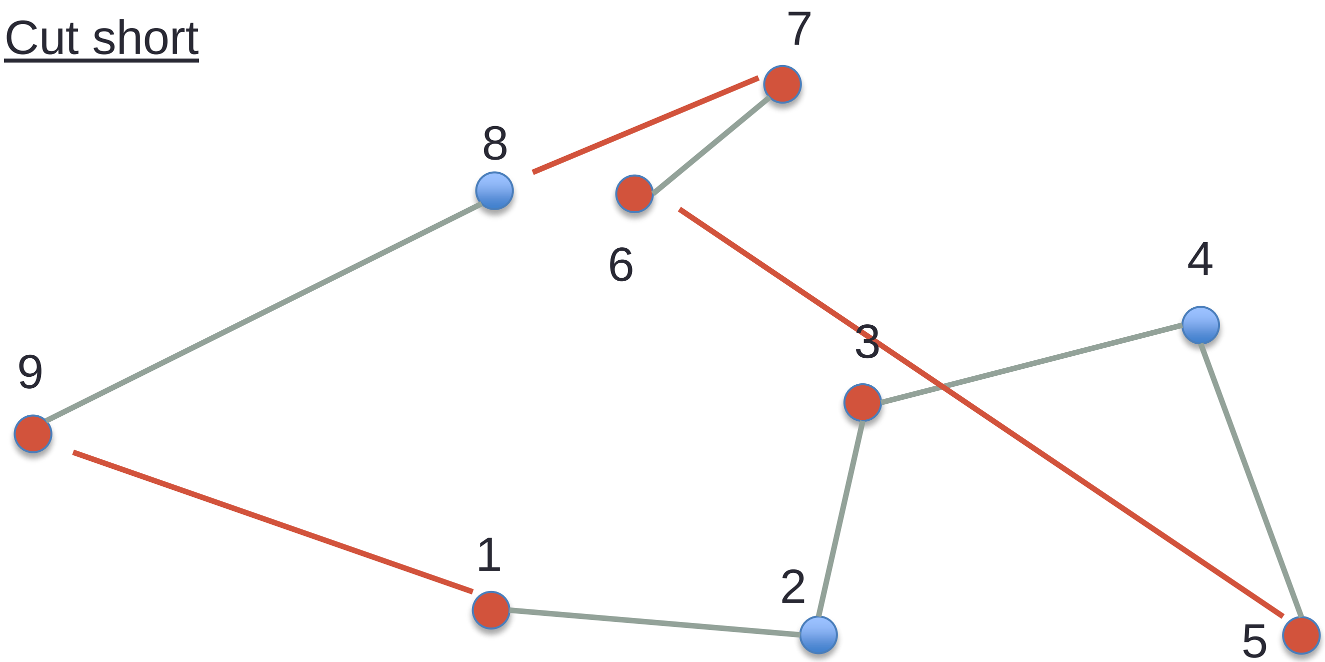
1. Find a minimum spanning tree T
2. Find a minimum matching M for the odd-degree vertices in T
3. Add M to T
4. <u>Find an Euler tour</u>
5. Cut short



Euler tour: 1,2,3,4,5,3,6,7,6,8,9,1

Christofides' Algorithm

1. Find a minimum spanning tree T
2. Find a minimum matching M for the odd-degree vertices in T
3. Add M to T
4. Find an Euler tour
5. <u>Cut short</u>



Short cut: 1,2,3,4,5,~~3,~~6,7~~,6~~,8,9,1

Theorem
Christofides' Algorithm is a 1.5-approximation algorithm

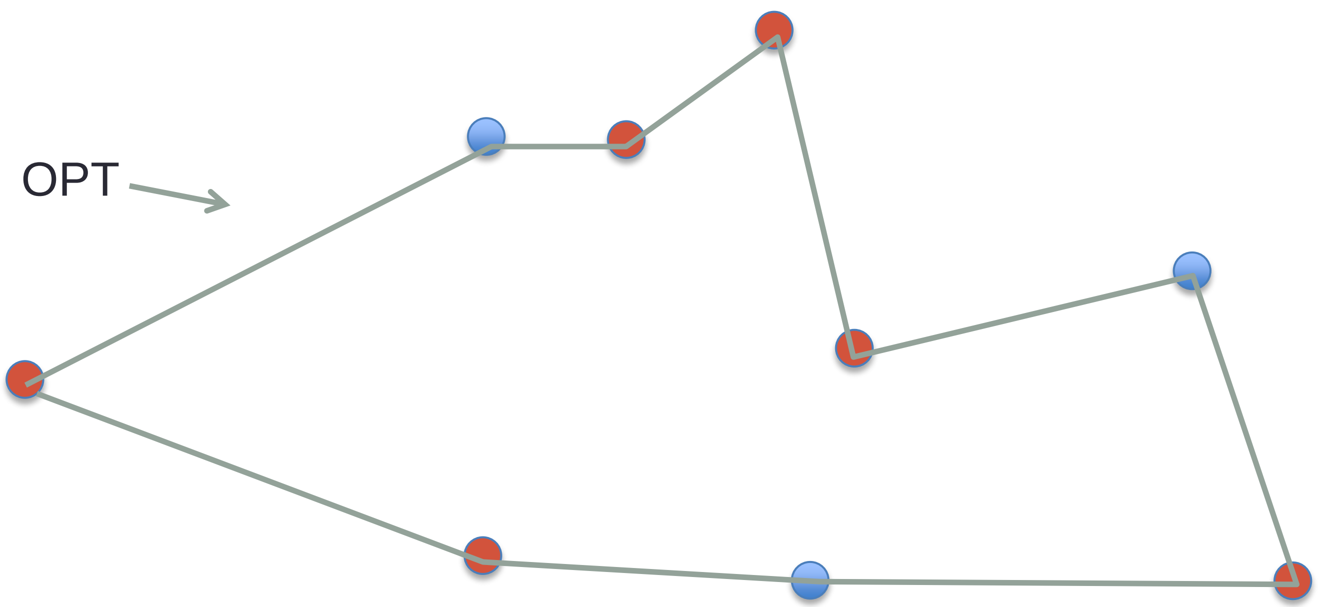
Proof
[1] Time? MST and Matching can be found in polynomial time.
[2] Feasible. OK
[3] Cost of TSP is at most the cost of the Euler tour.
Cost of Euler tour is cost(T)+cost(M)

Know: cost(T) $<$ OPT.

Claim: cost(M) $\leq$ OPT/2.

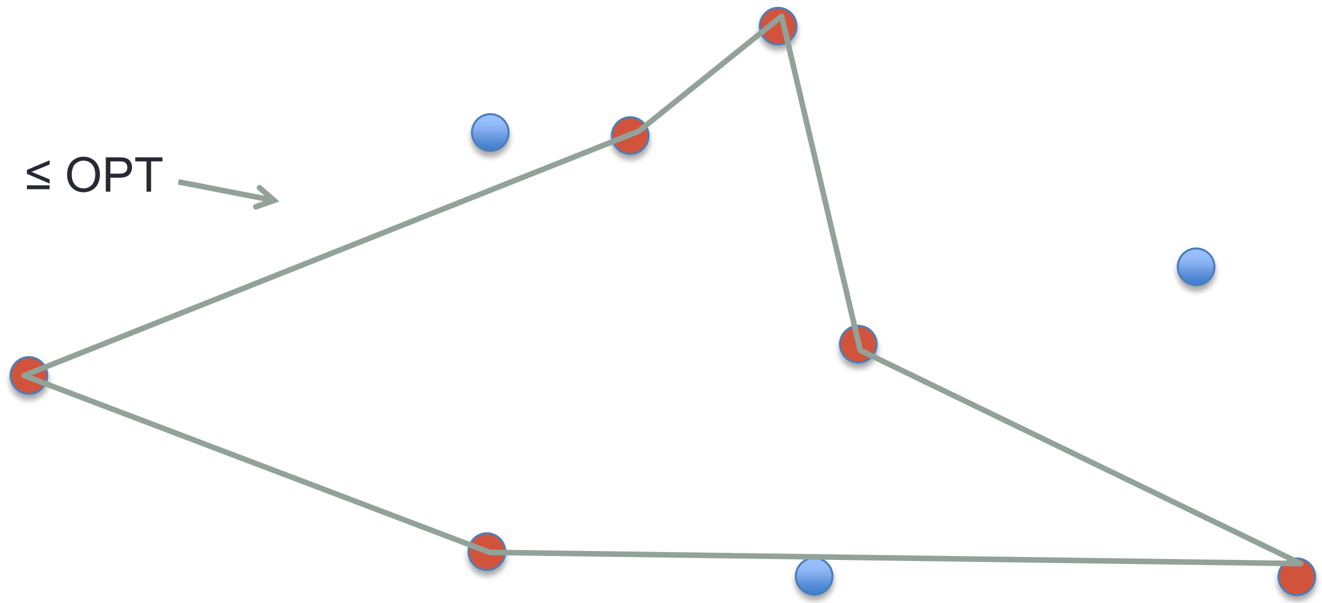
Theorem
Christofides' Algorithm is a 1.5-approximation algorithm

Proof
Claim: cost(M) ≤ OPT/2.



Theorem
Christofides' Algorithm is a 1.5-approximation algorithm

Proof
Claim: cost(M) ≤ OPT/2.



Theorem
Christofides' Algorithm is a 1.5-approximation algorithm

Proof
Claim: cost(M) ≤ OPT/2.

Two matchings.
Both have cost ≥ cost(M)
→ OPT ≥ 2cost(M)