Traveling Salesman

Given a set of cities ($\{1, \dots, n\}$) and a symmetric matrix $C = (c_{ij})$, $c_{ij} \geq 0$ that specifies for every pair $(i,j) \in [n] \times [n]$ the cost for travelling from city i to city j. Find a permutation π of the cities such that the round-trip cost

$$c_{\pi(1)\pi(n)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$$

is minimized.

Traveling Salesman

Theorem 2
There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

Traveling Salesman

Theorem 2

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $O(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 2

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 2

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 2
There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Traveling Salesman

Theorem 2

There does not exist an $O(2^n)$-approximation algorithm for TSP.

Hamiltonian Cycle:
For a given undirected graph $G = (V, E)$ decide whether there exists a simple cycle that contains all nodes in G.

- Given an instance to HAMPATH we create an instance for TSP.
- If $(i, j) \notin E$ then set c_{ij} to $n2^n$ otw. set c_{ij} to 1. This instance has polynomial size.
- There exists a Hamiltonian Path iff there exists a tour with cost n. Otw. any tour has cost strictly larger than $n2^n$.
- An $\mathcal{O}(2^n)$-approximation algorithm could decide btw. these cases. Hence, cannot exist unless $P = NP$.

Metric Traveling Salesman

In the metric version we assume for every triple $i, j, k \in \{1, \dots, n\}$

$$c_{ij} \le c_{ij} + c_{jk} \ .$$

It is convenient to view the input as a complete undirected graph $G = (V, E)$, where c_{ij} for an edge (i, j) defines the distance between nodes i and j.

Metric Traveling Salesman

In the metric version we assume for every triple $i, j, k \in \{1, \dots, n\}$

$$c_{ij} \le c_{ij} + c_{jk} \enspace .$$

It is convenient to view the input as a complete undirected graph $G = (V, E)$, where c_{ij} for an edge (i, j) defines the distance between nodes i and j.

TSP: Lower Bound I

Lemma 3

The cost $\mathrm{OPT}_{TSP}(G)$ *of an optimum traveling salesman tour is at least as large as the weight* $\mathrm{OPT}_{MST}(G)$ *of a minimum spanning tree in* G.

Proof:

- Take the optimum TSP-tour.
- Delete one edge.
- This gives a spanning tree of cost at most $\mathrm{OPT}_{TSP}(G)$.

TSP: Lower Bound I

Lemma 3
The cost $\mathrm{OPT}_{TSP}(G)$ of an optimum traveling salesman tour is at least as large as the weight $\mathrm{OPT}_{MST}(G)$ of a minimum spanning tree in G.

Proof:

- Take the optimum TSP-tour.
- Delete one edge.
- This gives a spanning tree of cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

TSP: Lower Bound I

Lemma 3
The cost $\mathrm{OPT}_{TSP}(G)$ of an optimum traveling salesman tour is at least as large as the weight $\mathrm{OPT}_{MST}(G)$ of a minimum spanning tree in G.

Proof:

- Take the optimum TSP-tour.
- Delete one edge.
- This gives a spanning tree of cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

TSP: Lower Bound I

Lemma 3

The cost $\mathrm{OPT}_{TSP}(G)$ *of an optimum traveling salesman tour is at least as large as the weight* $\mathrm{OPT}_{MST}(G)$ *of a minimum spanning tree in* G*.*

Proof:

- Take the optimum TSP-tour.
- Delete one edge.
- This gives a spanning tree of cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

TSP: Greedy Algorithm

- Start with a tour on a subset S containing a single node.
- Take the node v closest to S. Add it S and expand the existing tour on S to include v.
- Repeat until all nodes have been processed.

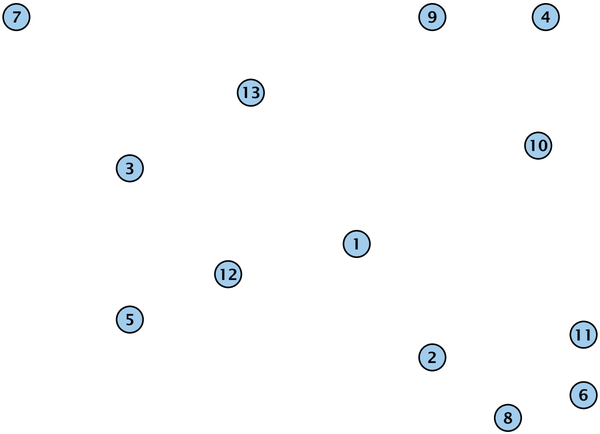
TSP: Greedy Algorithm

- Start with a tour on a subset S containing a single node.
- Take the node v closest to S. Add it S and expand the existing tour on S to include v.
- Repeat until all nodes have been processed.

TSP: Greedy Algorithm

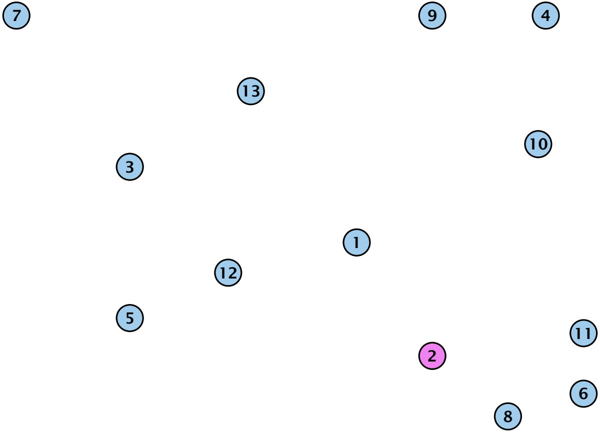
- Start with a tour on a subset S containing a single node.
- Take the node v closest to S. Add it S and expand the existing tour on S to include v.
- Repeat until all nodes have been processed.

TSP: Greedy Algorithm



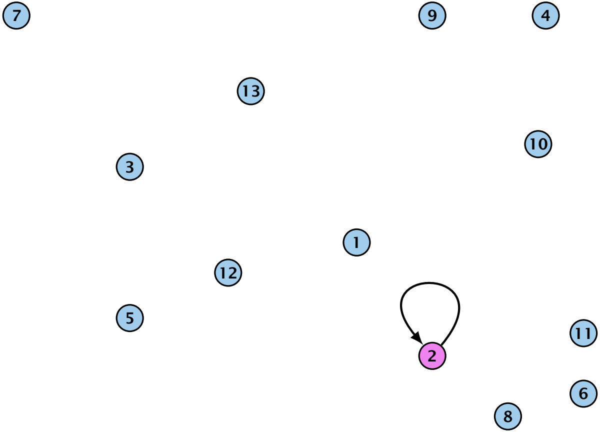
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



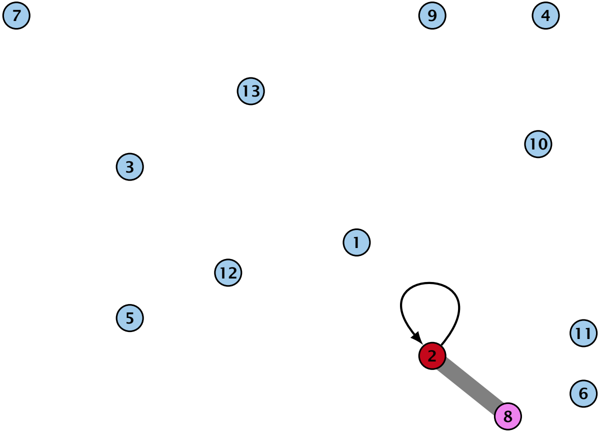
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



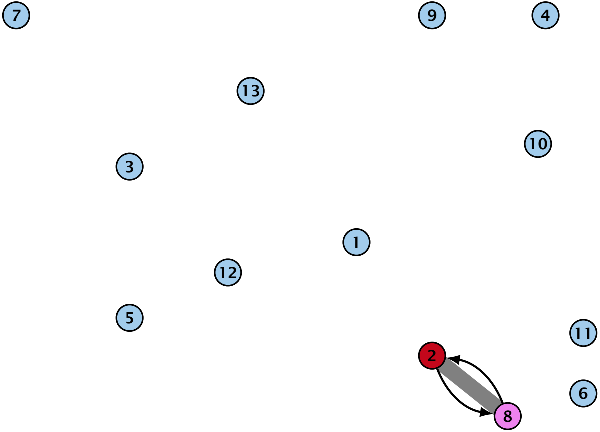
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



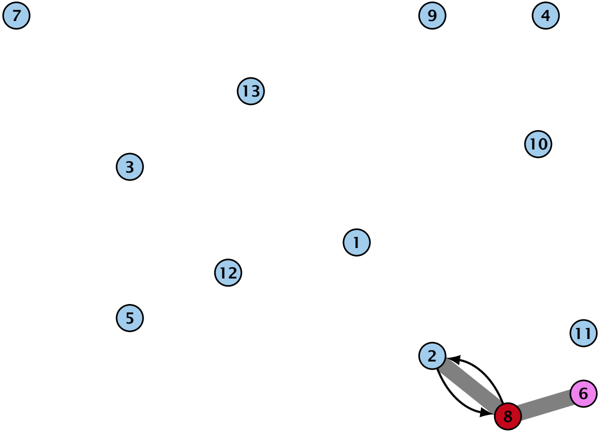
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



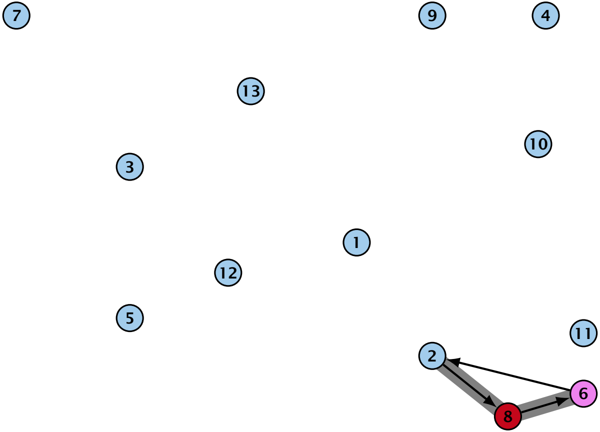
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



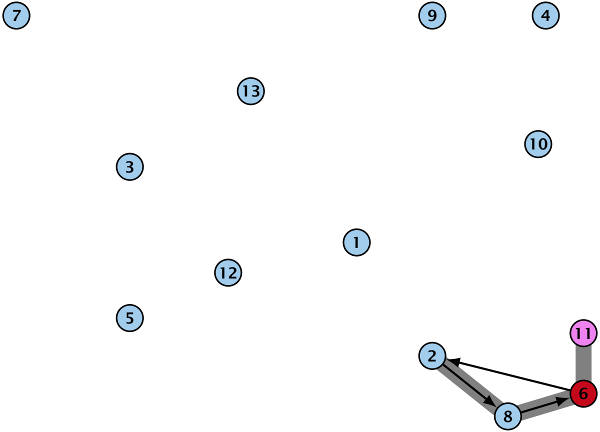
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



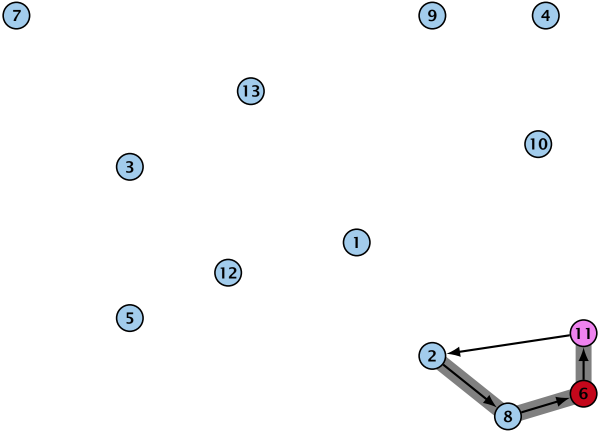
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



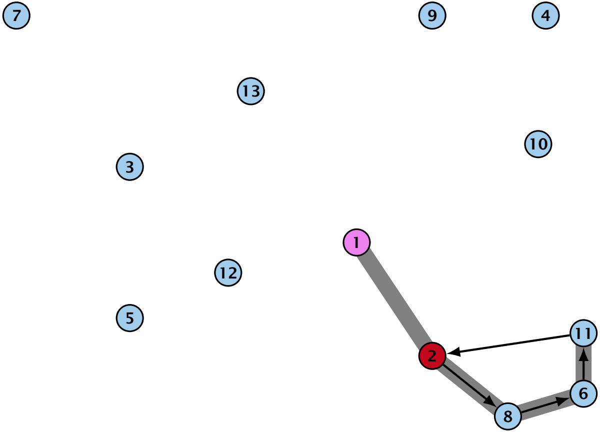
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



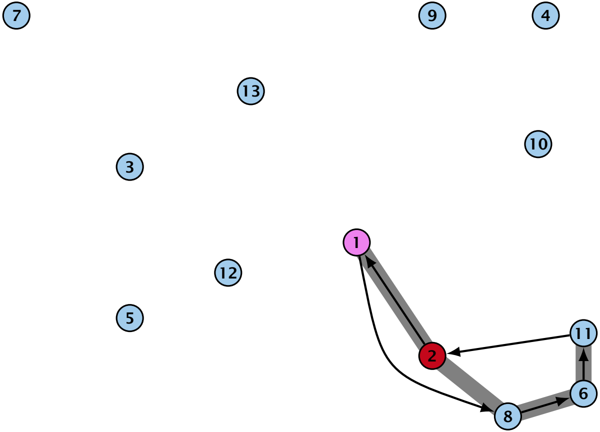
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



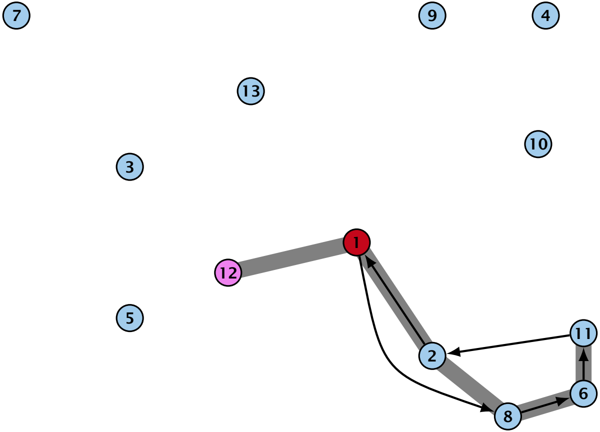
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



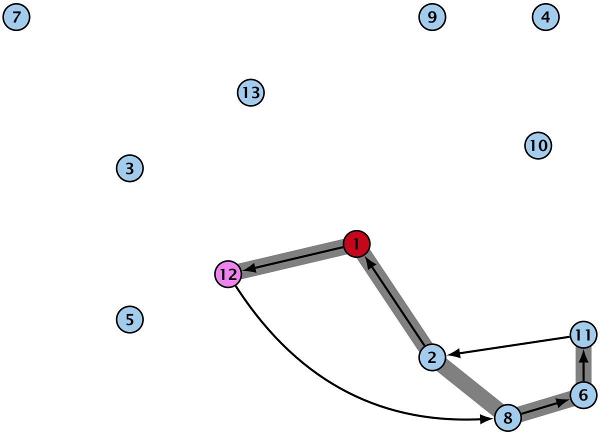
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



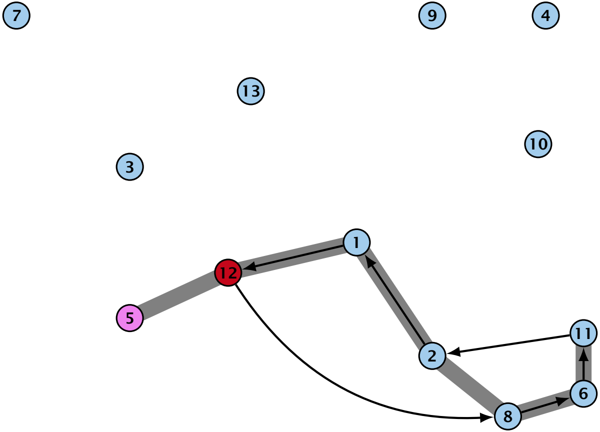
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



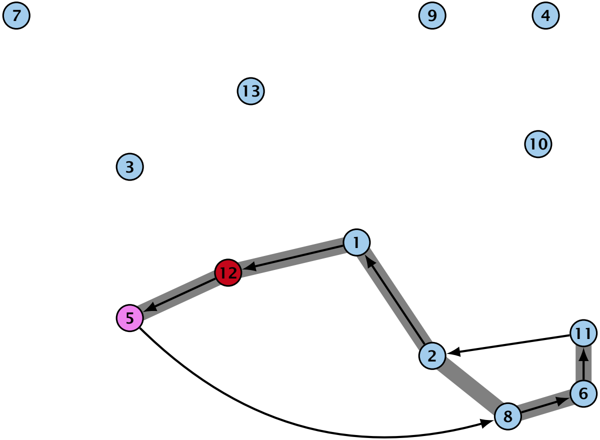
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



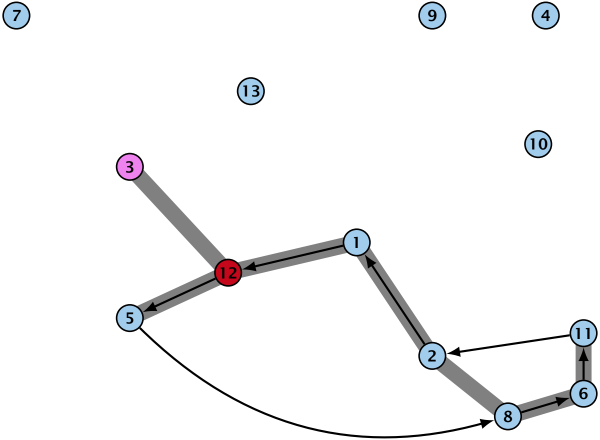
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



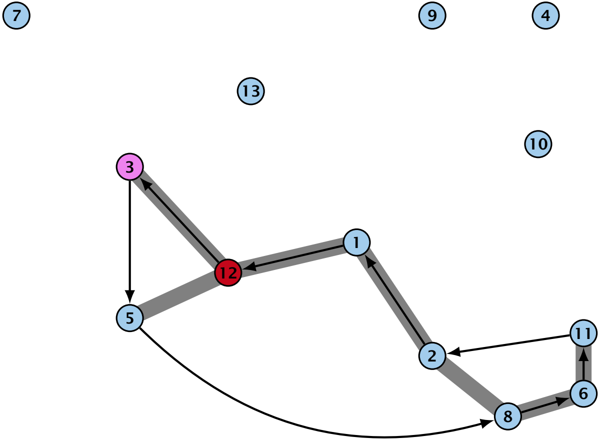
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



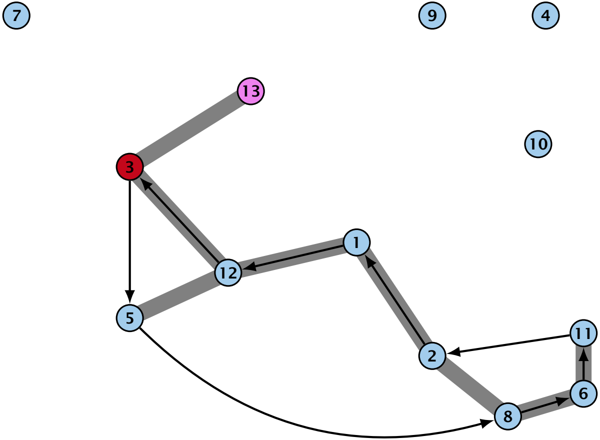
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



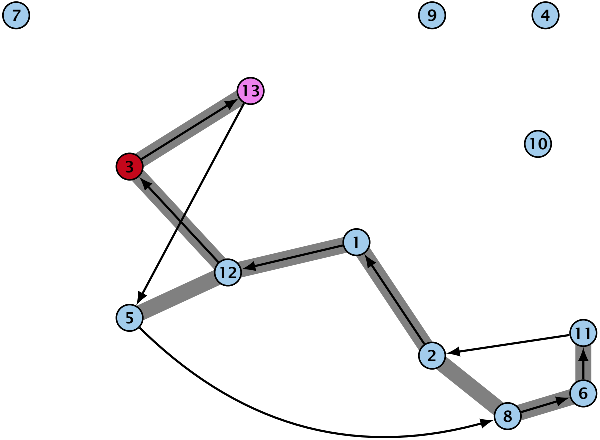
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



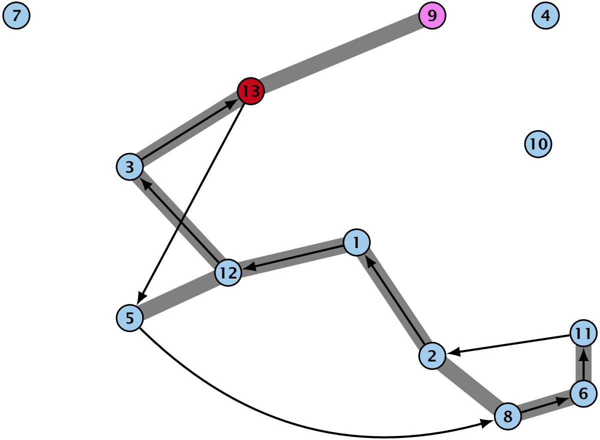
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



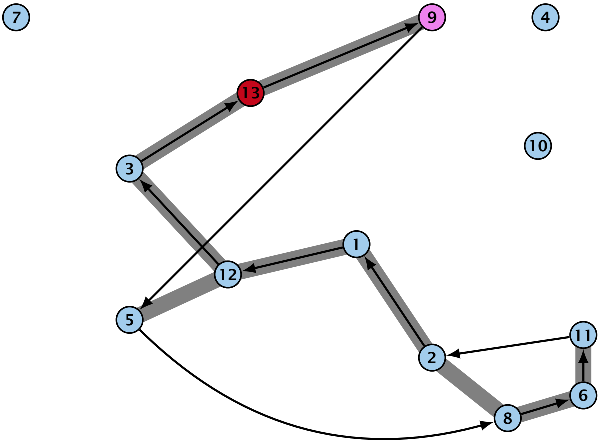
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



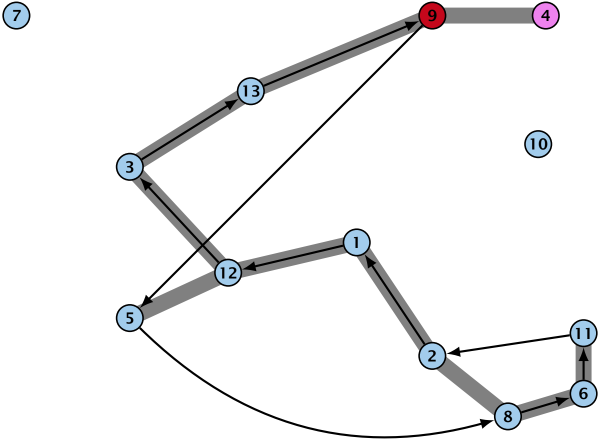
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



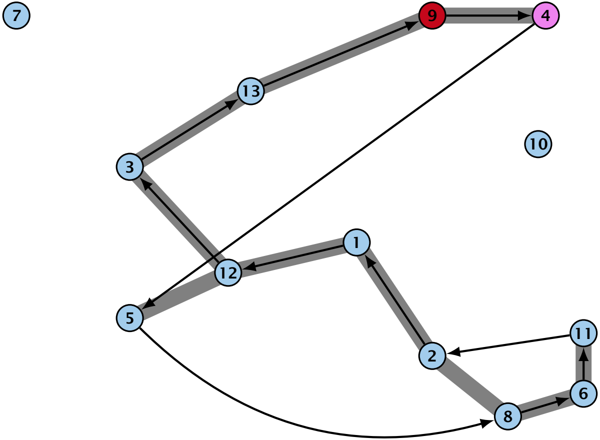
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



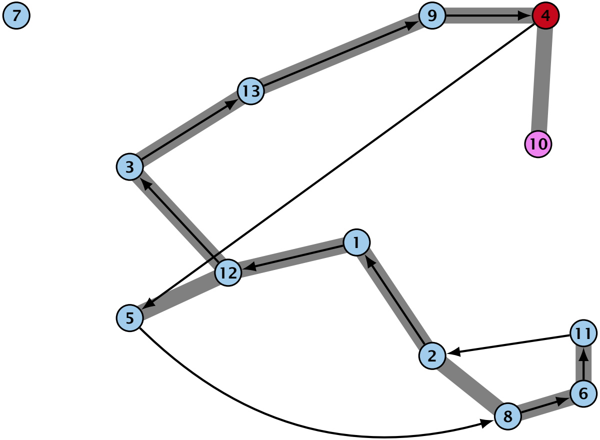
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



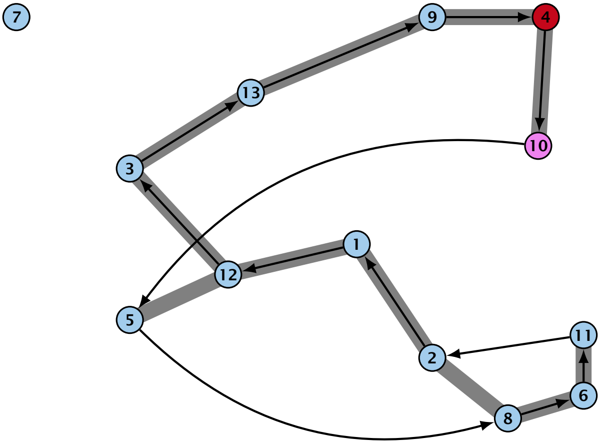
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



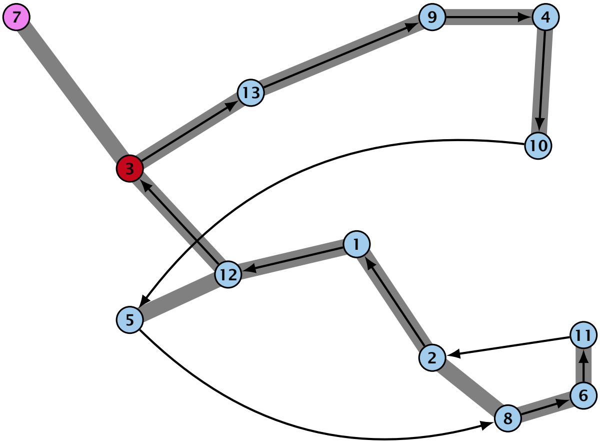
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



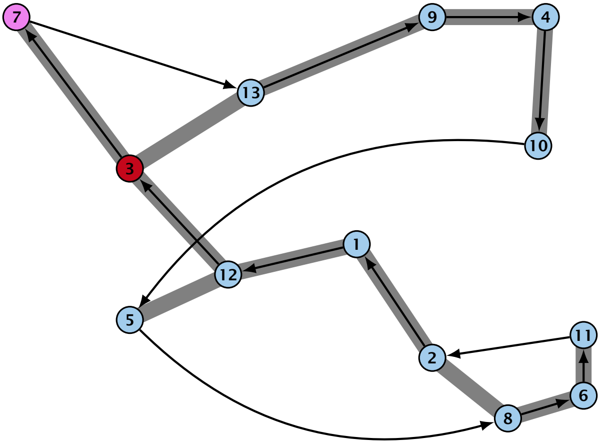
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



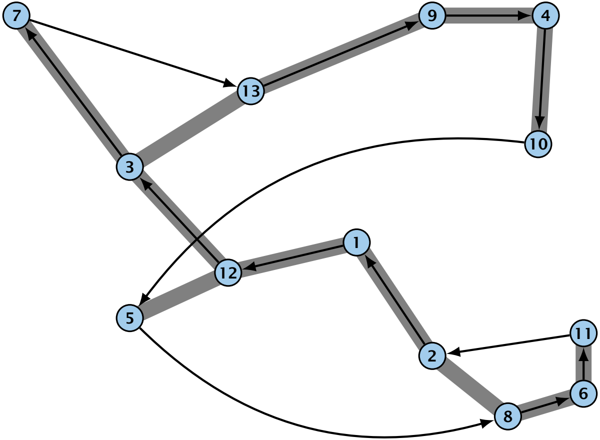
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



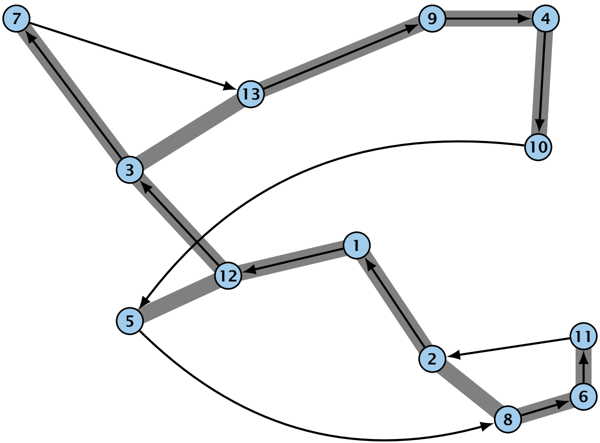
The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm



The gray edges form an MST, because exactly these edges are taken in Prims algorithm.

TSP: Greedy Algorithm

Lemma 4

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i-th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i.

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i).

This increases the cost by

$$c_{s_i,v_i} + c_{v_i,r_i} - c_{s_i,r_i} \leq 2c_{s_i,v_i}$$

TSP: Greedy Algorithm

Lemma 4

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i-th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i.

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i).

This increases the cost by

$$c_{s_i,v_i} + c_{v_i,r_i} - c_{s_i,r_i} \le 2c_{s_i,v_i}$$

TSP: Greedy Algorithm

Lemma 4
The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i-th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i.

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i).

This increases the cost by

$$c_{s_i,v_i} + c_{v_i,r_i} - c_{s_i,r_i} \le 2c_{s_i,v_i}$$

TSP: Greedy Algorithm

Lemma 4
The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i-th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i.

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i).

This increases the cost by

$$c_{s_i,v_i} + c_{v_i,r_i} - c_{s_i,r_i} \le 2c_{s_i,v_i}$$

TSP: Greedy Algorithm

Lemma 4

The Greedy algorithm is a 2-approximation algorithm.

Let S_i be the set at the start of the i-th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i.

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i).

This increases the cost by

$$c_{s_i,v_i} + c_{v_i,r_i} - c_{s_i,r_i} \le 2c_{s_i,v_i}$$

TSP: Greedy Algorithm

Lemma 4
The Greedy algorithm is a 2*-approximation algorithm.*

Let S_i be the set at the start of the i-th iteration, and let v_i denote the node added during the iteration.

Further let $s_i \in S_i$ be the node closest to $v_i \in S_i$.

Let r_i denote the successor of s_i in the tour before inserting v_i.

We replace the edge (s_i, r_i) in the tour by the two edges (s_i, v_i) and (v_i, r_i).

This increases the cost by

$$c_{s_i,v_i} + c_{v_i,r_i} - c_{s_i,r_i} \le 2c_{s_i,v_i}$$

TSP: Greedy Algorithm

The edges (s_i, v_i) considered during the Greedy algorithm are exactly the edges considered during PRIMs MST algorithm.

Hence,

$$\sum_i c_{s_i,v_i} = \mathrm{OPT}_{\mathrm{MST}}(G)$$

which with the previous lower bound gives a 2-approximation.

TSP: Greedy Algorithm

The edges (s_i, v_i) considered during the Greedy algorithm are exactly the edges considered during PRIMs MST algorithm.

Hence,

$$\sum_i c_{s_i,v_i} = \mathrm{OPT}_{\mathrm{MST}}(G)$$

which with the previous lower bound gives a 2-approximation.

TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i,j) \in E'$ $c'(i,j) \geq c(i,j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of G'.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.

TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i, j) \in E'$ $c'(i, j) \geq c(i, j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of G'.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.

TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i,j) \in E'$ $c'(i,j) \geq c(i,j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of G'.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.

TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i,j) \in E'$ $c'(i,j) \geq c(i,j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of G'.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.

TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i,j) \in E'$ $c'(i,j) \geq c(i,j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of G'.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.

TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i,j) \in E'$ $c'(i,j) \ge c(i,j)$.

Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

- Find an Euler tour of G'.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.

TSP: A different approach

Suppose that we are given an Eulerian graph $G' = (V, E', c')$ of $G = (V, E, c)$ such that for any edge $(i,j) \in E'$ $c'(i,j) \geq c(i,j)$.

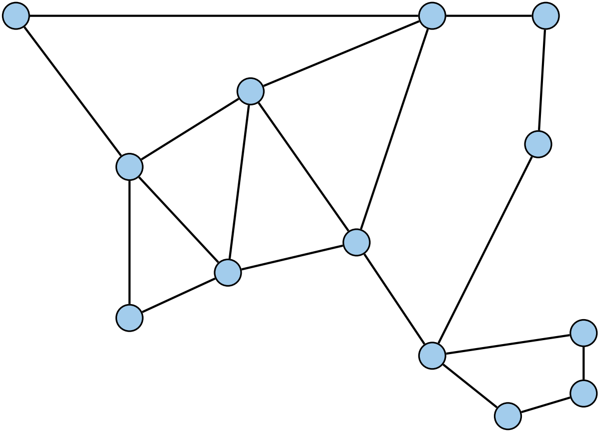
Then we can find a TSP-tour of cost at most

$$\sum_{e \in E'} c'(e)$$

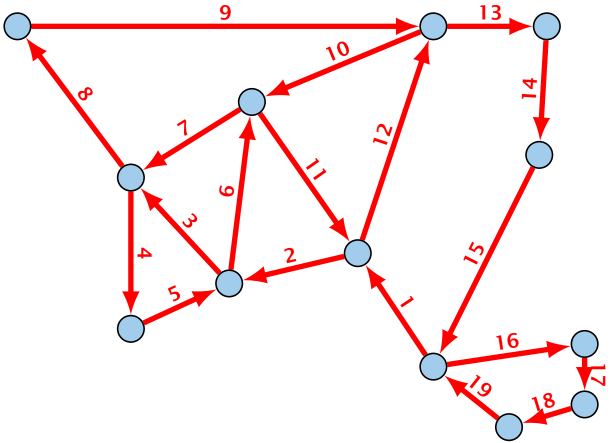
- Find an Euler tour of G'.
- Fix a permutation of the cities (i.e., a TSP-tour) by traversing the Euler tour and only note the first occurrence of a city.
- The cost of this TSP tour is at most the cost of the Euler tour because of triangle inequality.

This technique is known as short cutting the Euler tour.

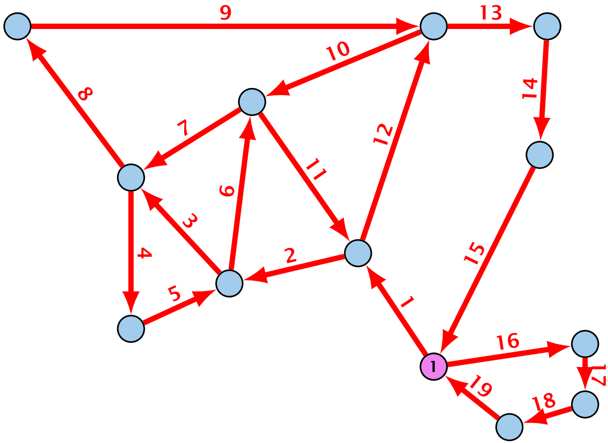
TSP: A different approach



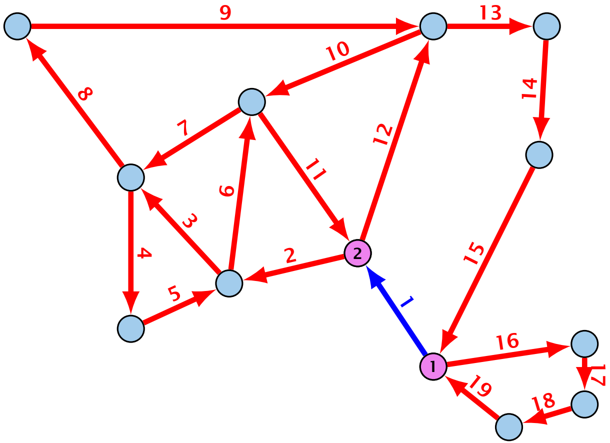
TSP: A different approach



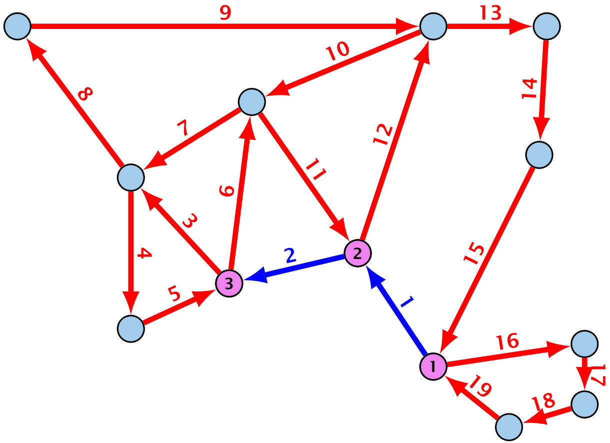
TSP: A different approach



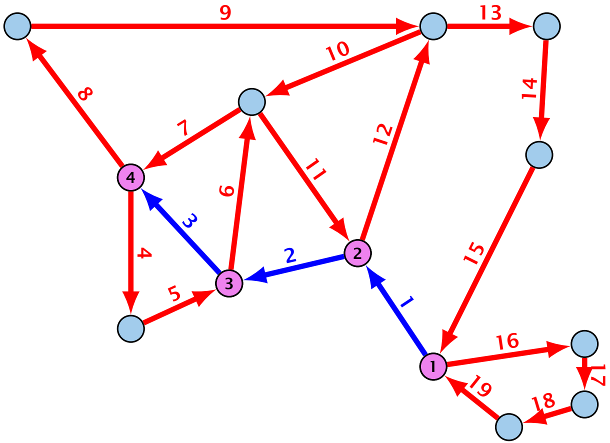
TSP: A different approach



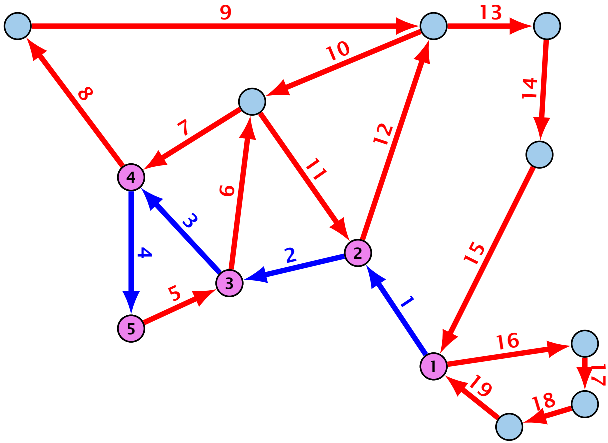
TSP: A different approach



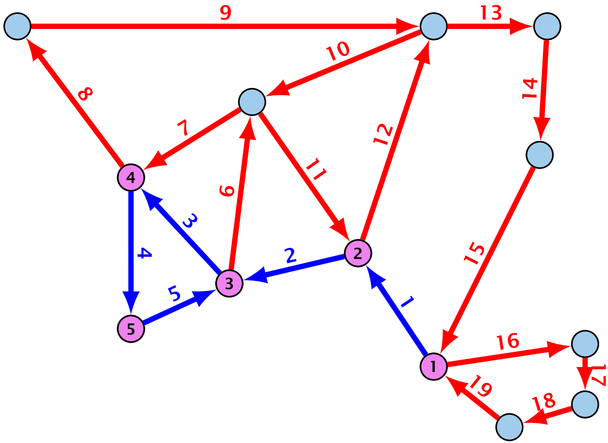
TSP: A different approach



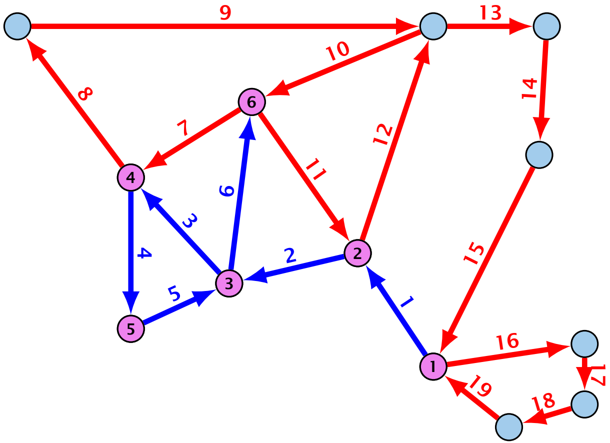
TSP: A different approach



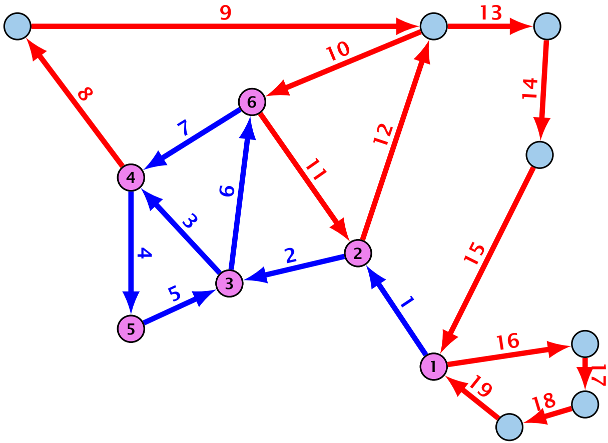
TSP: A different approach



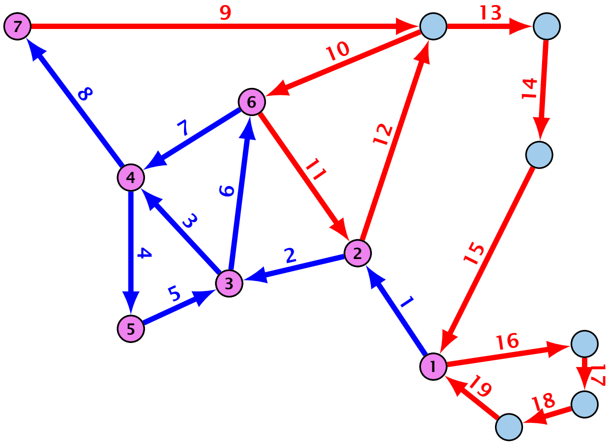
TSP: A different approach



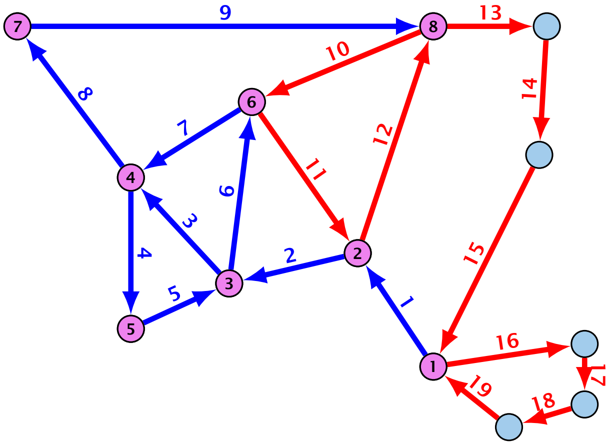
TSP: A different approach



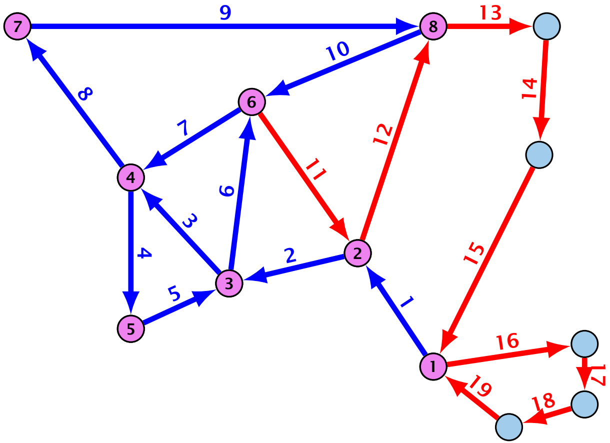
TSP: A different approach



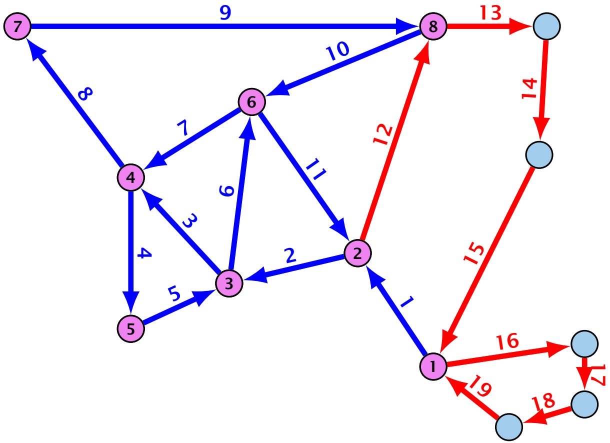
TSP: A different approach



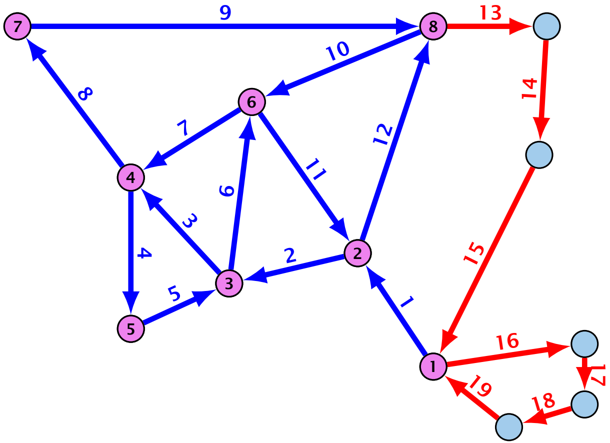
TSP: A different approach



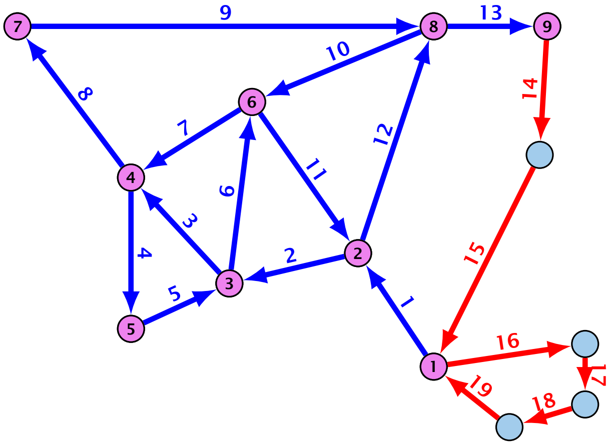
TSP: A different approach



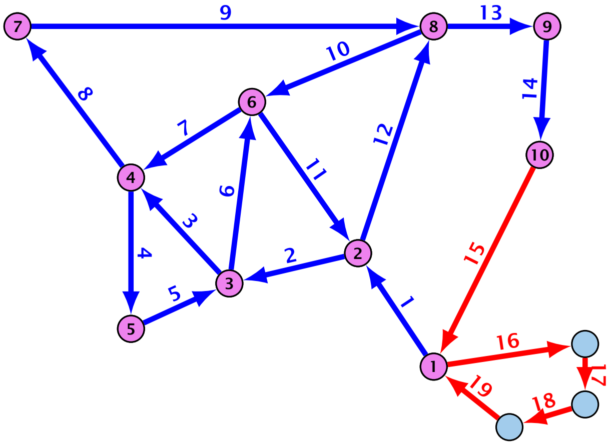
TSP: A different approach



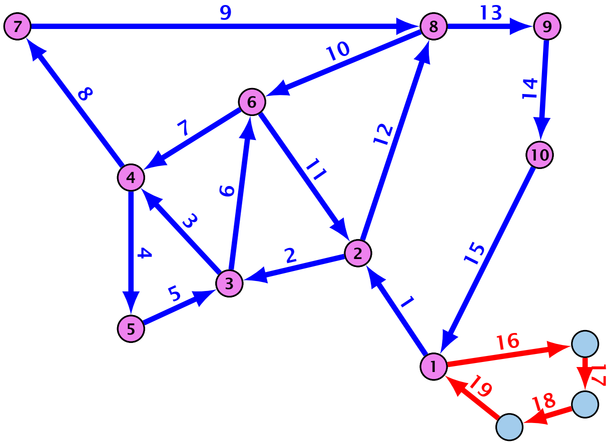
TSP: A different approach



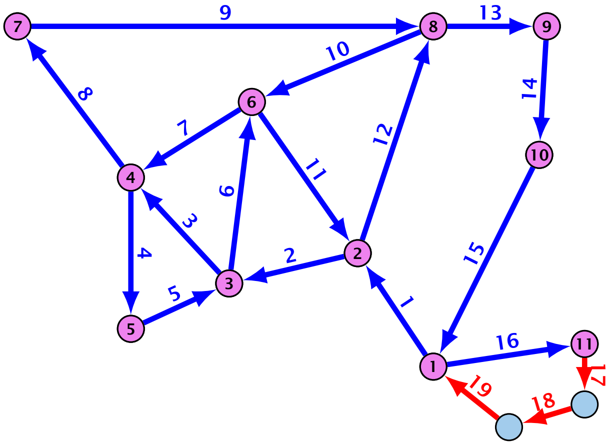
TSP: A different approach



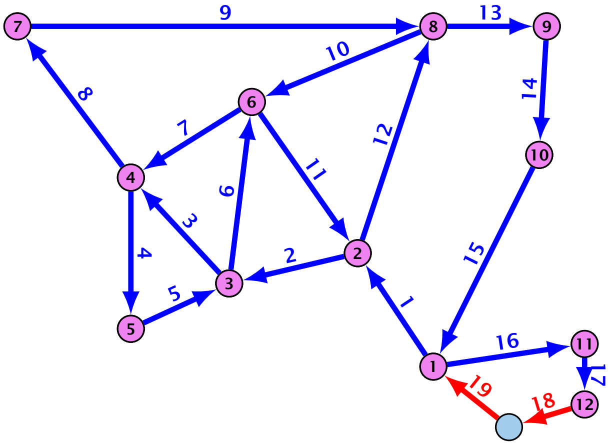
TSP: A different approach



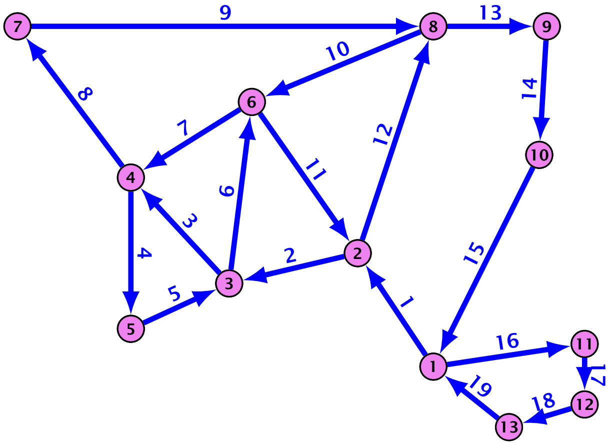
TSP: A different approach



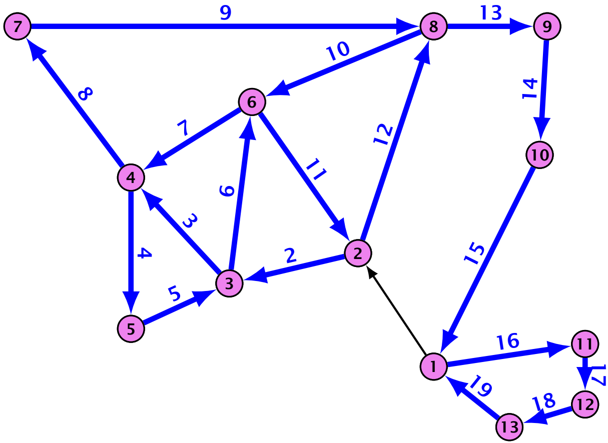
TSP: A different approach



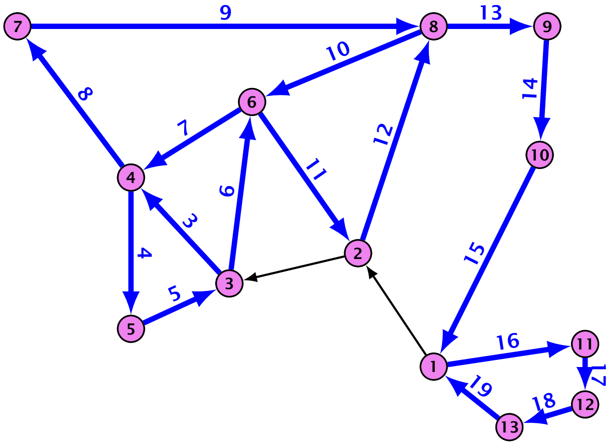
TSP: A different approach



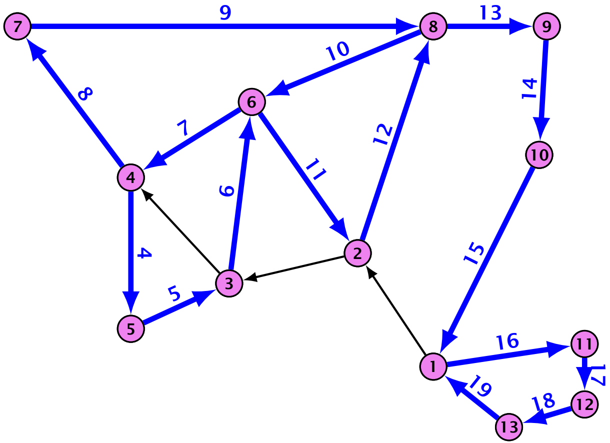
TSP: A different approach



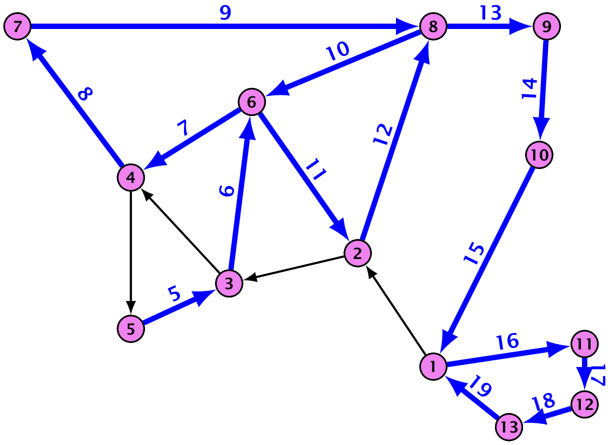
TSP: A different approach



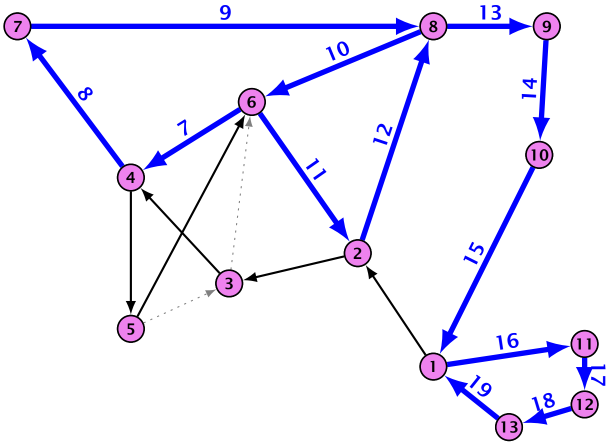
TSP: A different approach



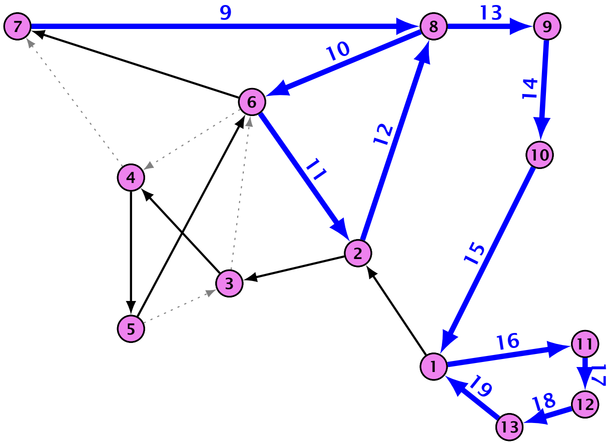
TSP: A different approach



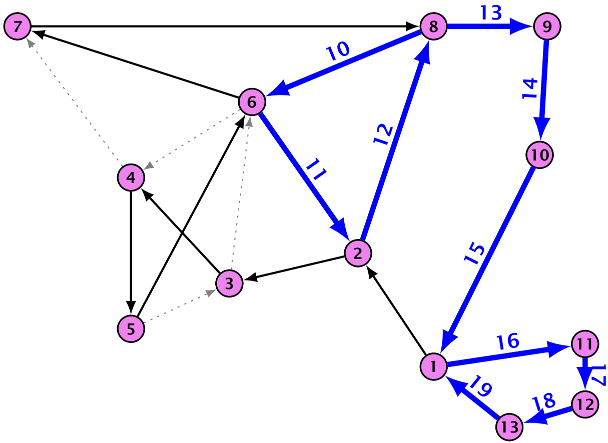
TSP: A different approach



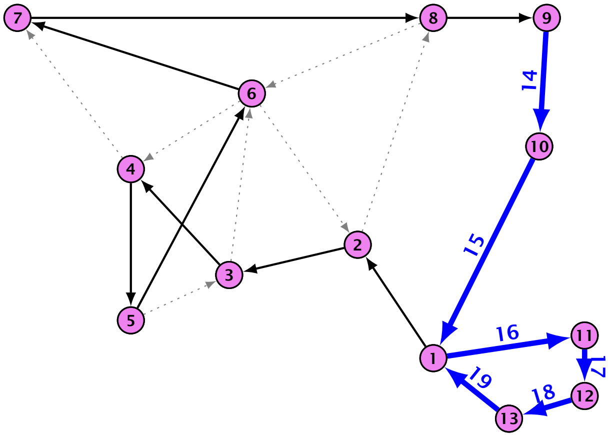
TSP: A different approach



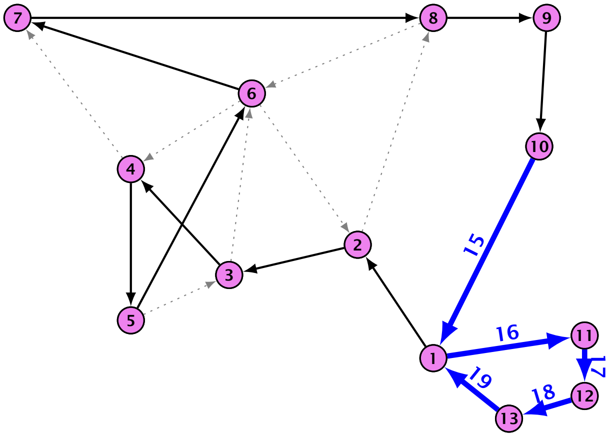
TSP: A different approach



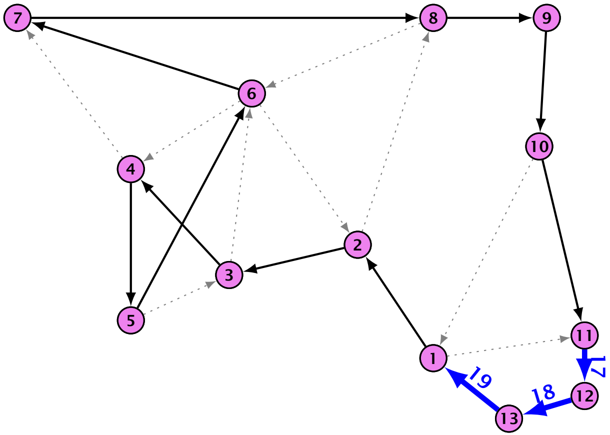
TSP: A different approach



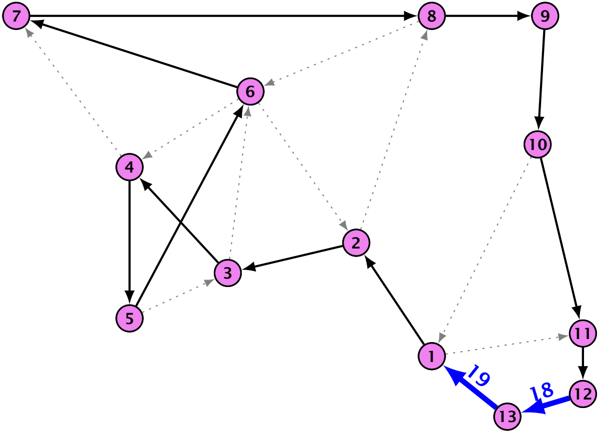
TSP: A different approach



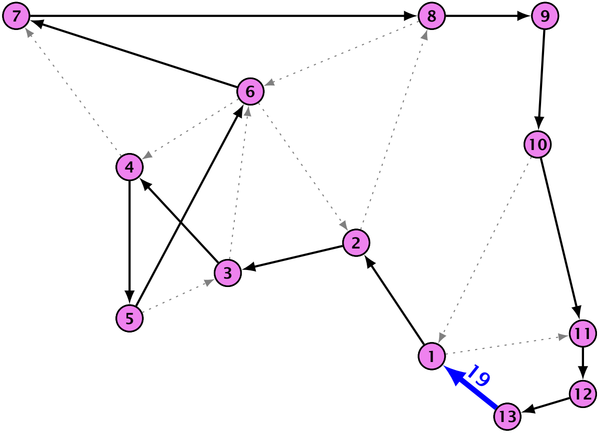
TSP: A different approach



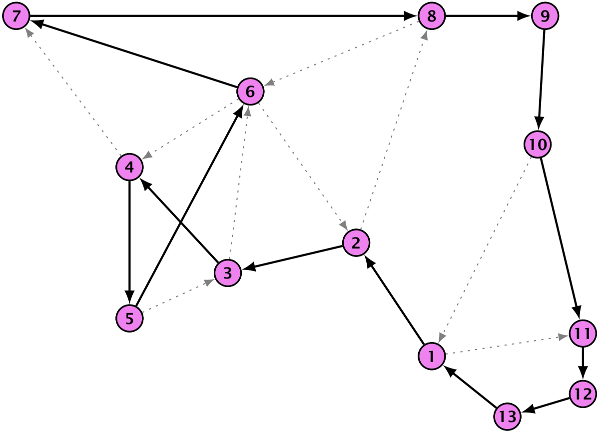
TSP: A different approach



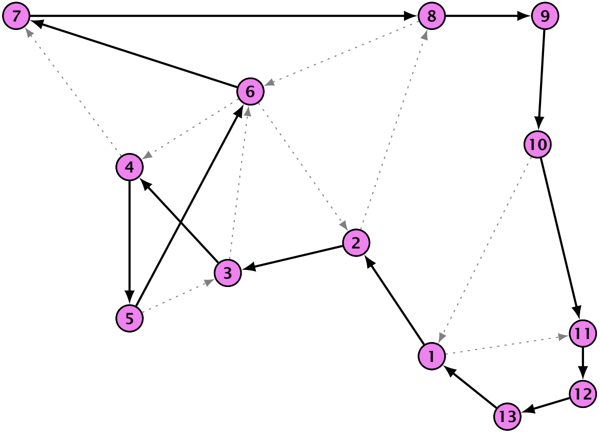
TSP: A different approach



TSP: A different approach



TSP: A different approach



TSP: A different approach

Consider the following graph:

- Compute an MST of G.
- Duplicate all edges.

This graph is Eulerian, and the total cost of all edges is at most $2 \cdot \mathrm{OPT}_{\mathrm{MST}}(G)$.

Hence, short-cutting gives a tour of cost no more than $2 \cdot \mathrm{OPT}_{\mathrm{MST}}(G)$ which means we have a 2-approximation.

TSP: A different approach

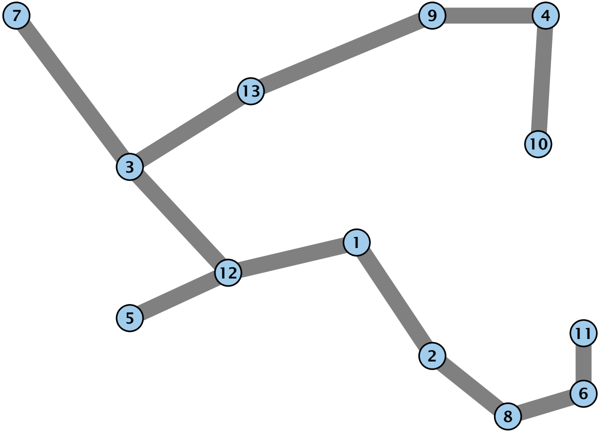
Consider the following graph:

- Compute an MST of G.
- Duplicate all edges.

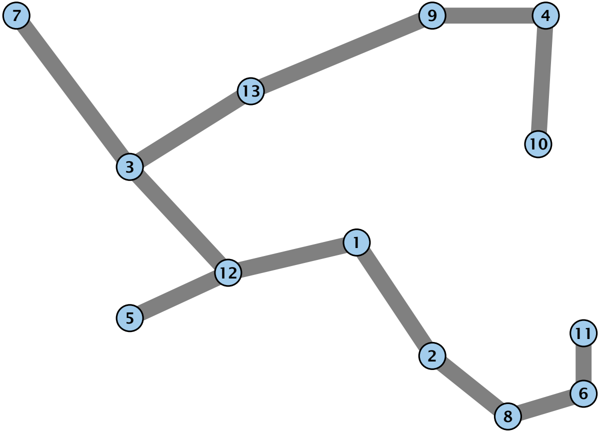
This graph is Eulerian, and the total cost of all edges is at most $2 \cdot \mathrm{OPT}_{\mathrm{MST}}(G)$.

Hence, short-cutting gives a tour of cost no more than $2 \cdot \mathrm{OPT}_{\mathrm{MST}}(G)$ which means we have a 2-approximation.

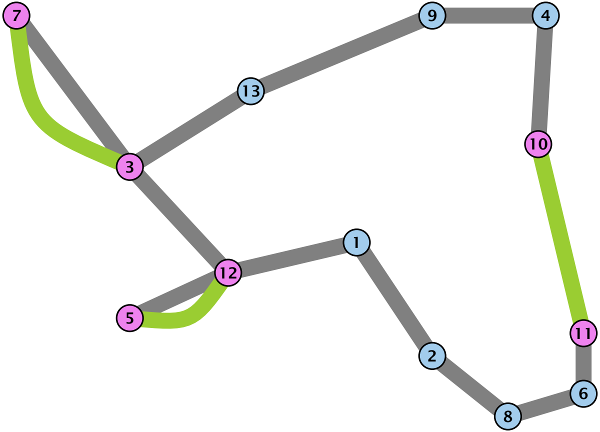
TSP: Can we do better?



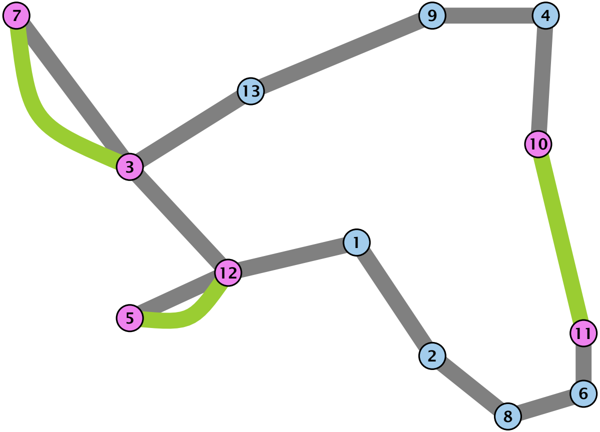
TSP: Can we do better?



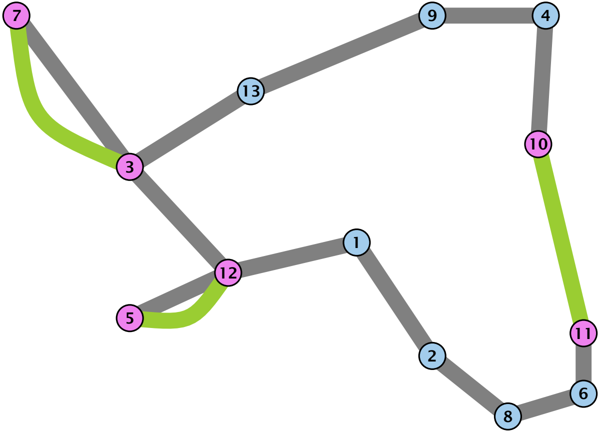
TSP: Can we do better?



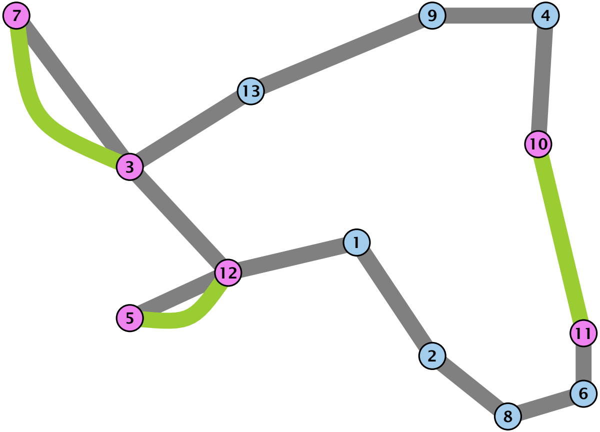
TSP: Can we do better?



TSP: Can we do better?



TSP: Can we do better?



TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

Duplicating all edges in the MST seems to be rather wasteful.

We only need to make the graph Eulerian.

For this we compute a Minimum Weight Matching between odd degree vertices in the MST (note that there are an even number of them).

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\mathrm{OPT}_{\mathrm{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\mathrm{OPT}_{\mathrm{MST}}(G) + \mathrm{OPT}_{\mathrm{TSP}}(G)/2 \leq \frac{3}{2}\mathrm{OPT}_{\mathrm{TSP}}(G) ,$$

Short cutting gives a $\frac{3}{2}$-approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\mathrm{OPT}_{\mathrm{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\mathrm{OPT}_{\mathrm{MST}}(G) + \mathrm{OPT}_{\mathrm{TSP}}(G)/2 \leq \frac{3}{2}\mathrm{OPT}_{\mathrm{TSP}}(G) \ ,$$

Short cutting gives a $\frac{3}{2}$-approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\mathrm{OPT}_{\mathrm{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\mathrm{OPT}_{\mathrm{MST}}(G) + \mathrm{OPT}_{\mathrm{TSP}}(G)/2 \le \frac{3}{2}\mathrm{OPT}_{\mathrm{TSP}}(G) \enspace ,$$

Short cutting gives a $\frac{3}{2}$-approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\mathrm{OPT}_{\mathrm{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\mathrm{OPT}_{\mathrm{MST}}(G) + \mathrm{OPT}_{\mathrm{TSP}}(G)/2 \le \frac{3}{2}\mathrm{OPT}_{\mathrm{TSP}}(G) \ ,$$

Short cutting gives a $\frac{3}{2}$-approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\mathrm{OPT}_{\mathrm{TSP}}(G)/2$.

Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\mathrm{OPT}_{\mathrm{MST}}(G) + \mathrm{OPT}_{\mathrm{TSP}}(G)/2 \leq \frac{3}{2}\mathrm{OPT}_{\mathrm{TSP}}(G) \enspace ,$$

Short cutting gives a $\frac{3}{2}$-approximation for metric TSP.

This is the best that is known.

TSP: Can we do better?

An optimal tour on the odd-degree vertices has cost at most $\mathrm{OPT}_{\mathrm{TSP}}(G)$.

However, the edges of this tour give rise to two disjoint matchings. One of these matchings must have weight less than $\mathrm{OPT}_{\mathrm{TSP}}(G)/2$.

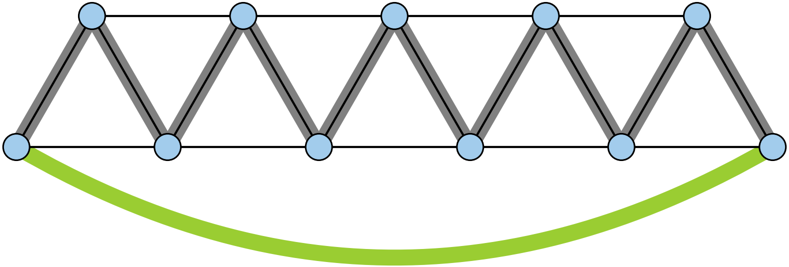
Adding this matching to the MST gives an Eulerian graph with edge weight at most

$$\mathrm{OPT}_{\mathrm{MST}}(G) + \mathrm{OPT}_{\mathrm{TSP}}(G)/2 \leq \frac{3}{2}\mathrm{OPT}_{\mathrm{TSP}}(G) \ ,$$

Short cutting gives a $\frac{3}{2}$-approximation for metric TSP.

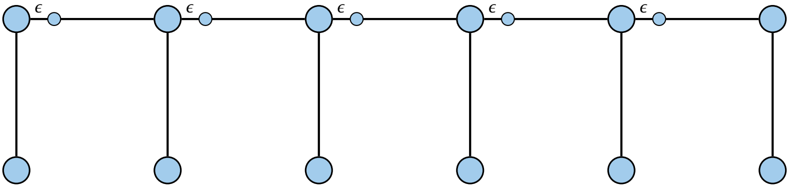
This is the best that is known.

Christofides. Tight Example



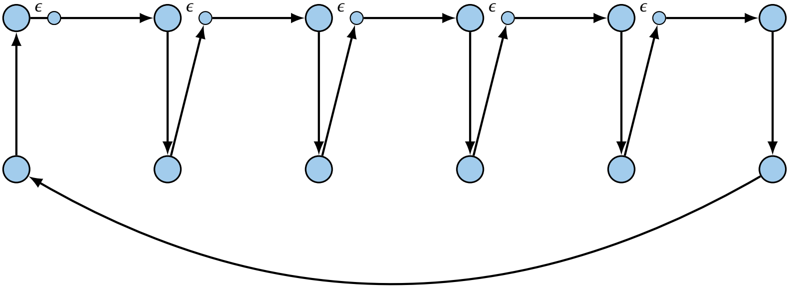
- optimal tour: n edges.
- MST: $n - 1$ edges.
- weight of matching $(n+1)/2 - 1$
- MST+matching $\approx 3/2 \cdot n$

Tree shortcutting. Tight Example



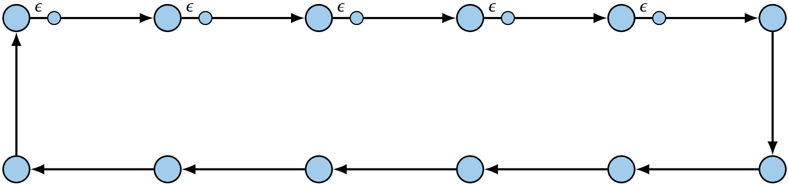
- edges have Euclidean distance.

Tree shortcutting. Tight Example



- edges have Euclidean distance.

Tree shortcutting. Tight Example



- edges have Euclidean distance.