Coping with NP-completeness: Approximation Algorithms

Alexander S. Kulikov

Steklov Institute of Mathematics at St. Petersburg
Russian Academy of Sciences

Advanced Algorithms and Complexity
Data Structures and Algorithms

Outline

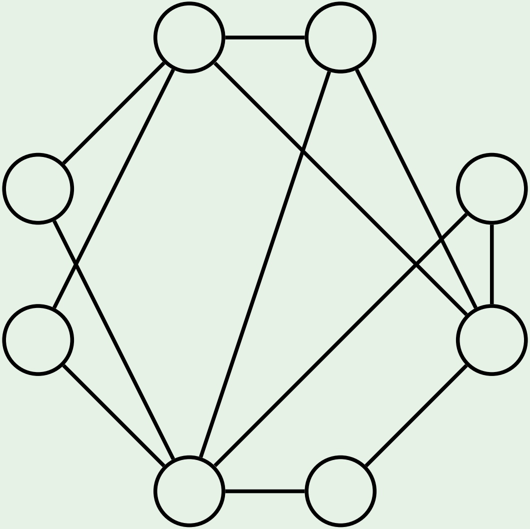
1. Vertex cover
2. Traveling salesman
   - Metric TSP
   - Local search

Vertex cover (optimization version)

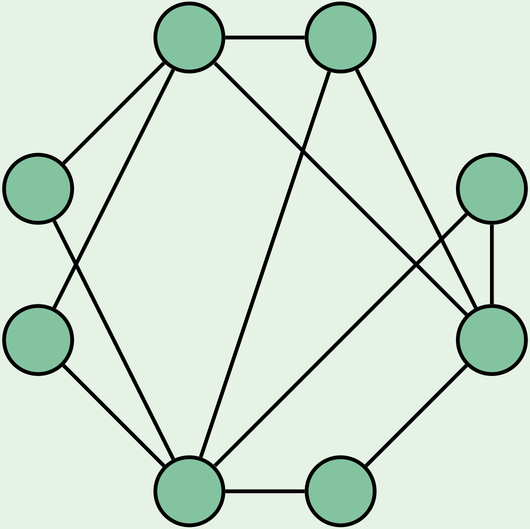
Input: A graph.

Output: A subset of vertices of minimum size that touches every edge.

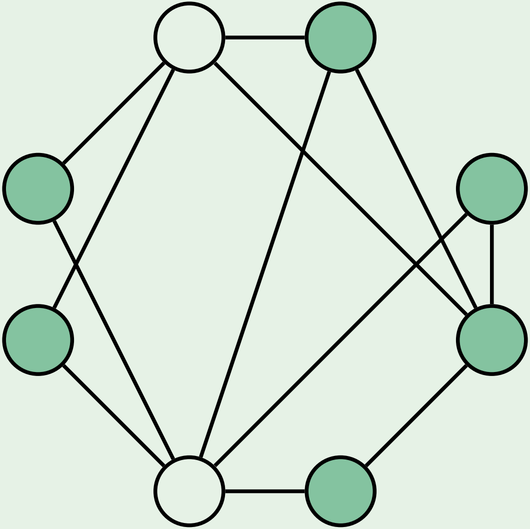
Example



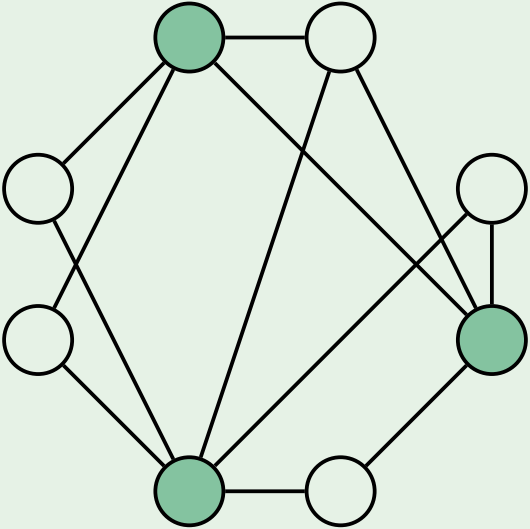
Example



Example



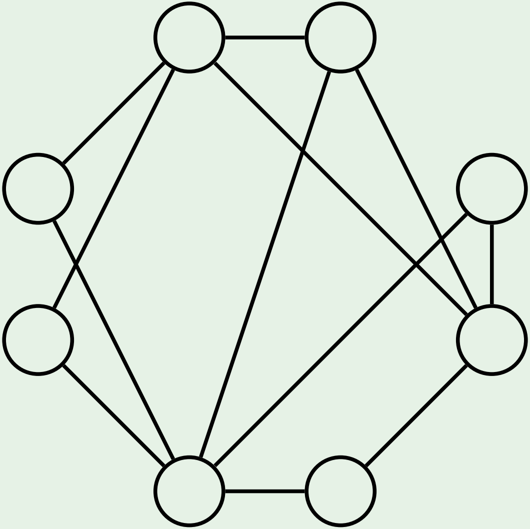
Example



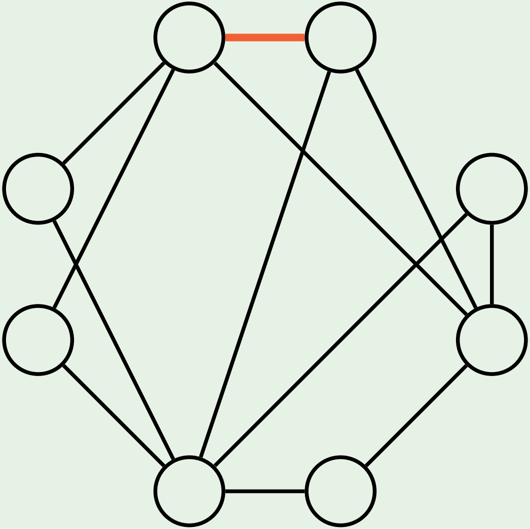
```
ApproxVertexCover(G(V, E))

C ← empty set
while E is not empty:
  {u, v} ← any edge from E
  add u, v to C
  remove from E all edges incident to u, v
return C
```

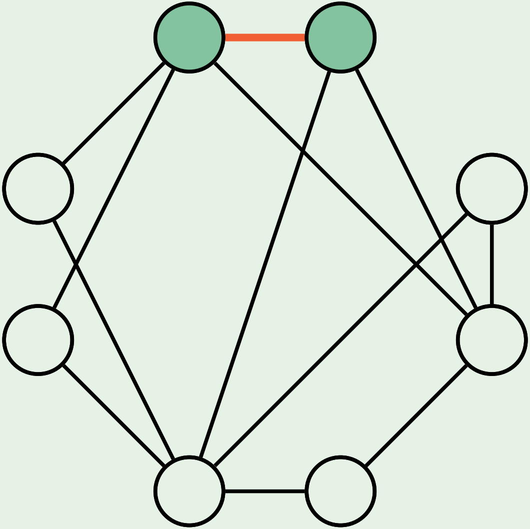
Example



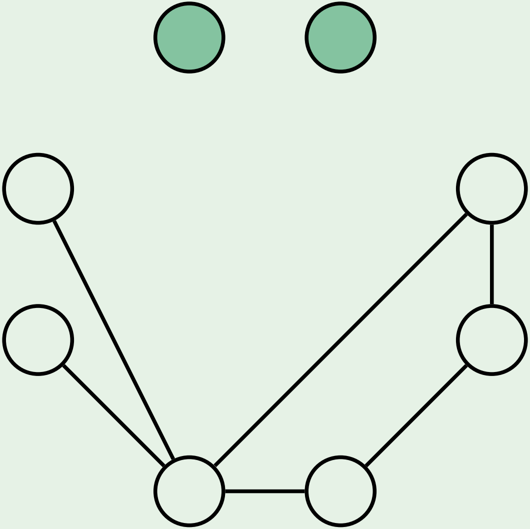
Example



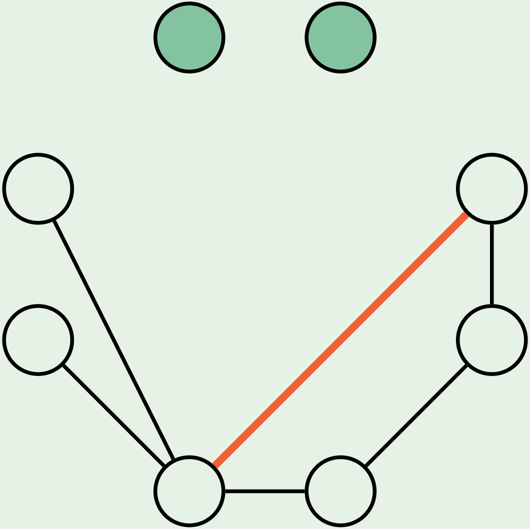
Example



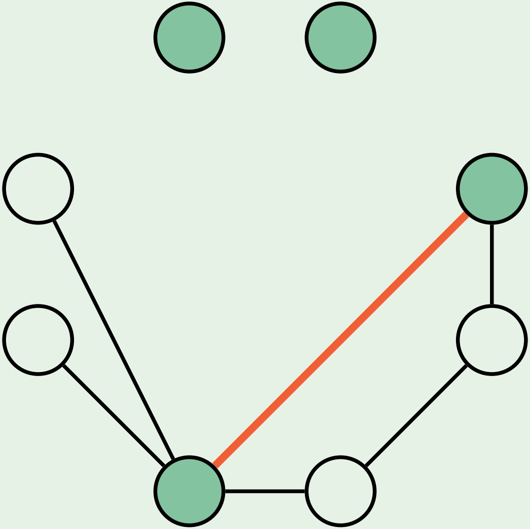
Example



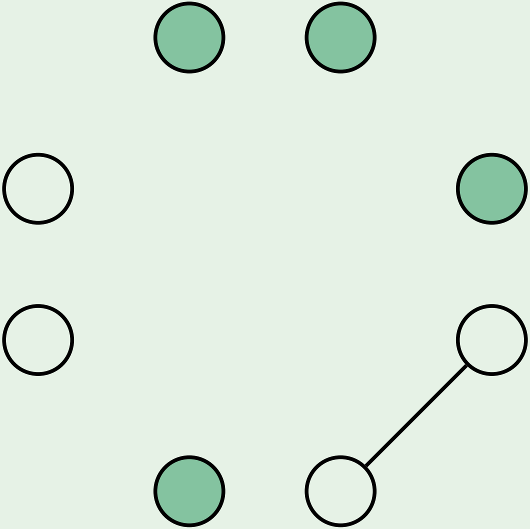
Example



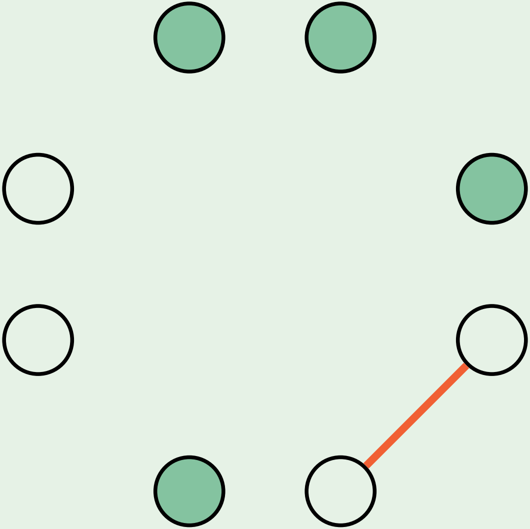
Example



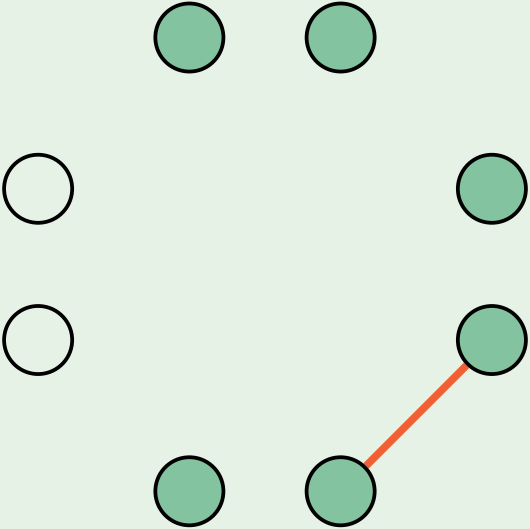
Example



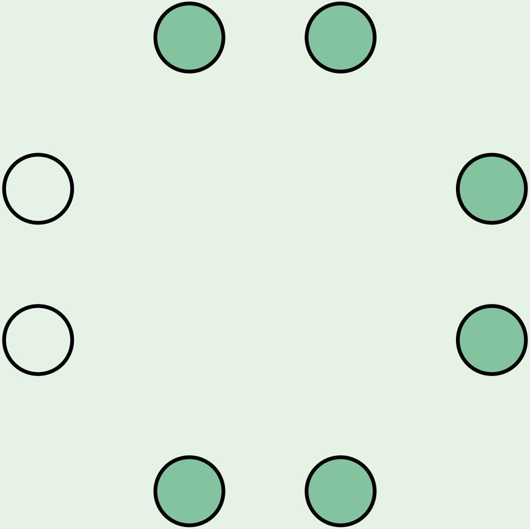
Example



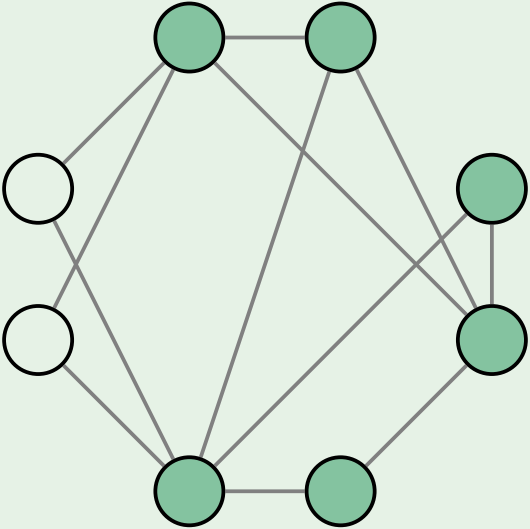
Example



Example



Example



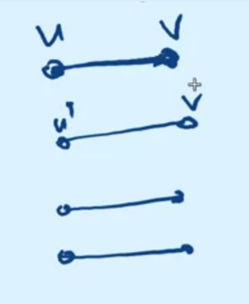
Lemma

The algorithm `ApproxVertexCover` is 2-approximate: it returns a vertex cover that is at most twice as large as an optimal one and runs in polynomial time.

Proof

- The set M of all edges selected by the algorithm forms a matching

This means that all the endpoints of edges selected by our algorithm is DISJOINT because after selecting an edge our algorithm discards all the edges adjacent to u and v which means the endpoints of other selected edges u' and v' (not v as shown in fig) are not going to coincide with u and v



Proof

- The set M of all edges selected by the algorithm forms a matching
- Any vertex cover of the graph has size at least $|M|$

Proof

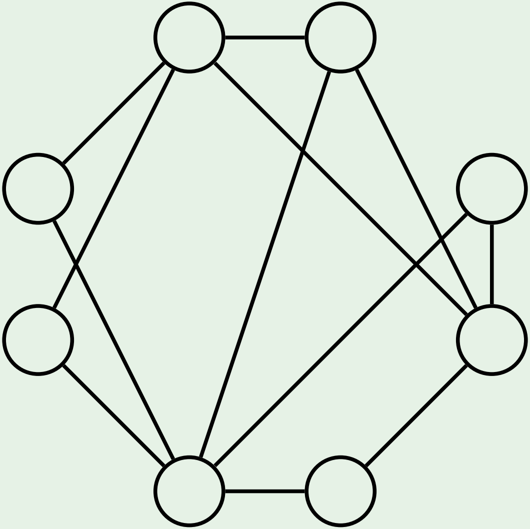
- The set M of all edges selected by the algorithm forms a matching
- Any vertex cover of the graph has size at least $|M|$
- The algorithm returns a vertex cover C of size $2|M|$, hence

$$|C| = 2 \cdot |M| \leq 2 \cdot \mathrm{OPT}$$

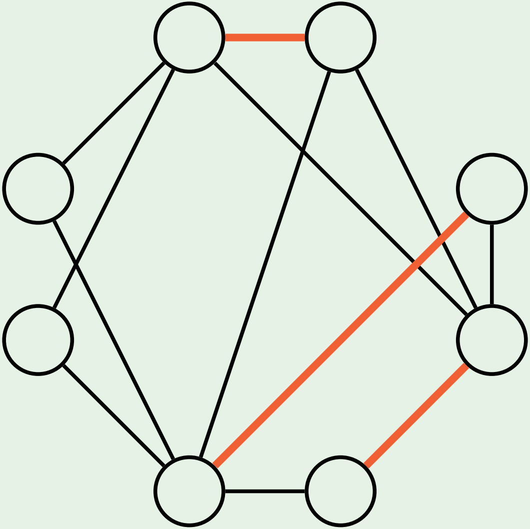
We know that M is atmost optimal Vertex value

$\square$

Example



Example



Summary

- We don't know the value of OPT, but we've managed to prove that

$$|C| \leq 2 \cdot \mathrm{OPT}$$

So how it can possibly be that we proved some upper bound on C in terms of OPT without knowing the exact value of OPT ?

Ans: This is because we know the lower bound of OPT which is M and size of vertex cover is 2|M|. hence this allows us to find the upper bound not in terms of some quantities that we don't know namely OPT but in terms of the size of M which we can compute quickly

- C in terms of OPT that we don't know
- OPT atleast M
- C is equal to 2M
- hence C is atmost 2M and we can compute M quickly

Summary

- We don't know the value of OPT, but we've managed to prove that

$$|C| \leq 2 \cdot \mathrm{OPT}$$

- This is because we know a lower bound on OPT: it is at least the size of any matching

$$|C| = 2 \cdot |M| \leq 2 \cdot \mathrm{OPT}$$

Final Remarks

- The bound is tight: there are graphs for which the algorithm returns a vertex cover of size twice the minimum size.

note here the optimal solution in bipartite graph is of size n but the verte cover returned by our graph is of size 2n

Question

We've just discussed an algorithm that finds in polynomial time a vertex cover that is at most twice the size of a smallest vertex cover. Recall that in any graph the complement of a vertex cover is an independent set. This automatically gives us a 2-approximation algorithm for the independent set problem! That is, we run the 2-approximation algorithm for the vertex cover problem and then return the complement of the constructed vertex cover. It is easy to see that the resulting independent set has size at least half of a largest independent set. Is this correct?

- (•) Yes, sure!
- ( ) Of course, not!

✕ **Incorrect**

No, this is not correct, in fact. Consider a complete bipartite graph with parts of size $n/2$. The minimum size of a vertex cover as well as the maximum size of an independent set in this graph is $n/2$ (take any of the two parts). However the 2-approximation vertex cover algorithm in this case returns all n vertices. The complement of this set is just an empty set.

Final Remarks

- The bound is tight: there are graphs for which the algorithm returns a vertex cover of size twice the minimum size.
- No 1.99-approximation algorithm is known.

Outline

1. Vertex cover

2. Traveling salesman
   - Metric TSP
   - Local search

Outline

1. Vertex cover
2. Traveling salesman
   - Metric TSP
   - Local search

Metric TSP (optimization version)

Input: An undirected graph $G(V, E)$ with non-negative edge weights satisfying the triangle inequality: for all $u, v, w \in V$, $d(u, v) + d(v, w) \geq d(u, w)$.

Output: A cycle of minimum total length visiting each vertex exactly once .

Lower Bound

- We are going to design a 2-approximation algorithm: it returns a cycle that is at most twice as long as an optimal cycle: $C \leq 2 \cdot \mathrm{OPT}$

Lower Bound

- We are going to design a 2-approximation algorithm: it returns a cycle that is at most twice as long as an optimal cycle: $C \leq 2 \cdot \mathrm{OPT}$
- Since we don't know the value of OPT, we need a good lower bound L on OPT:

$$C \leq 2 \cdot L \leq 2 \cdot \mathrm{OPT}$$

We are going to use MST as L

Minimum Spanning Trees

Lemma

Let G be an undirected graph with non-negative edge weights. Then $\mathrm{MST}(G) \leq \mathrm{TSP}(G)$.

Minimum Spanning Trees

Lemma

Let G be an undirected graph with non-negative edge weights. Then $\mathrm{MST}(G) \leq \mathrm{TSP}(G)$.

$\mathrm{MST}(G) \leq P \leq \mathrm{TSP}(G)$

Proof

By removing any edge from an optimum TSP cycle one gets a spanning tree of G. $\square$

`ApproxMetricTSP`(G)

$T \leftarrow$ `minimum spanning tree of` G

ApproxMetricTSP(G)

$T \leftarrow$ minimum spanning tree of G

$D \leftarrow T$ with each edge doubled

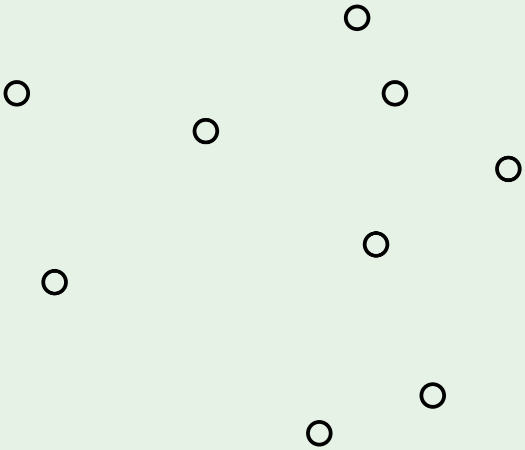
`ApproxMetricTSP`(G)

$T \leftarrow$ `minimum spanning tree of` G
$D \leftarrow T$ `with each edge doubled`
`find an Eulerian cycle` C `in` D

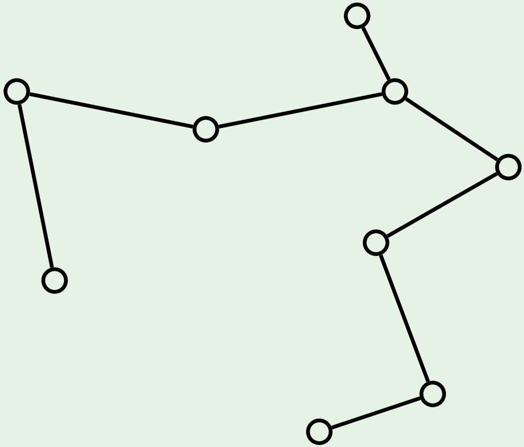
ApproxMetricTSP(G)

$T \leftarrow$ minimum spanning tree of G
$D \leftarrow T$ with each edge doubled
find an Eulerian cycle C in D
return a cycle that visits vertices in
  the order of their first appearance in C

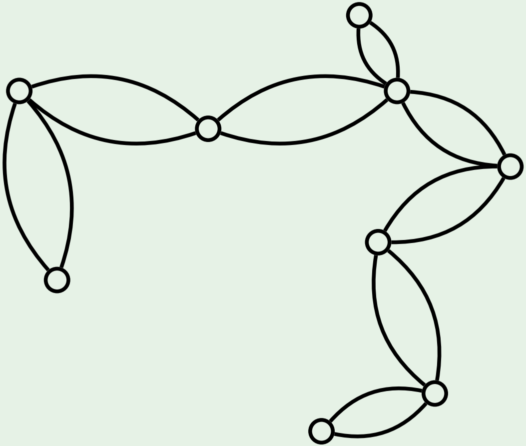
Example: points on a plane



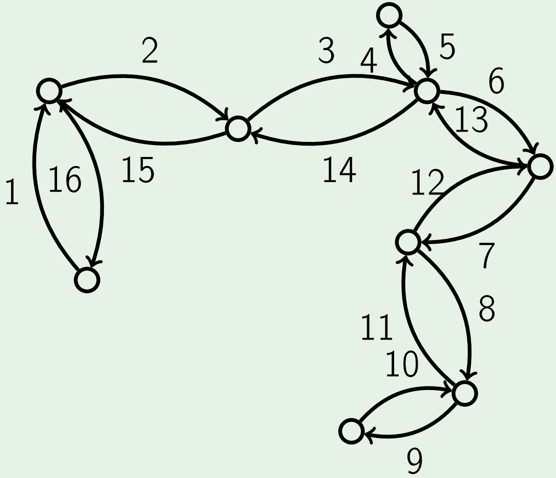
Example: points on a plane



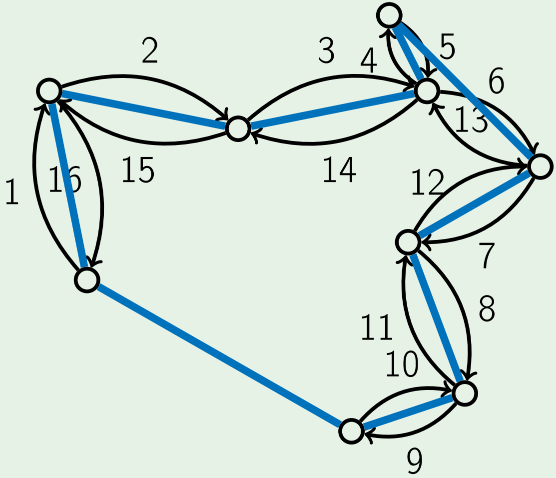
Example: points on a plane



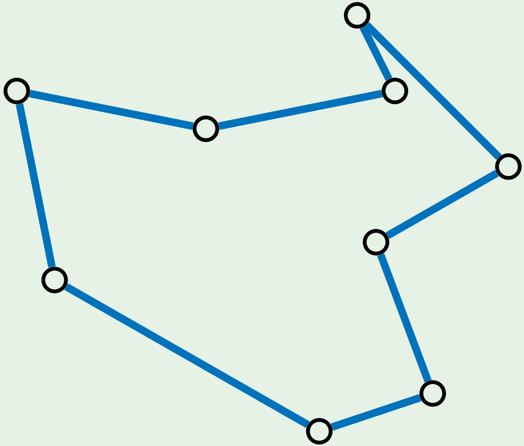
Example: points on a plane



Example: points on a plane



Example: points on a plane



Lemma

The algorithm `ApproxMetricTSP` is 2-approximate.

Lemma

The algorithm `ApproxMetricTSP` is 2-approximate.

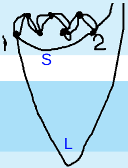
Proof

- The total length of the MST T is at most OPT. because MST will always be one less then optimal TSP as TSP is a loop

Lemma

The algorithm `ApproxMetricTSP` is 2-approximate.

Note that the approximate TSP algorithm won't hold good if triangle ineqaulity is not satisfied meaning in the image shown 1 can connect to 2 via the short direct path S and not the long direct path L



Proof

- The total length of the MST T is at most OPT. hence MST <= OPT
- Bypasses can only decrease the total length.

When we doubled each edge of the tree T, we get a graph whose total weight is atmost 2OPT (hence 2MST <= 2OPT) and then when we transform this eulerian cycle into hamiltonian cycle we can only decrease the total weight of this cycle hence ApproximateTSP will return the distance less then 2OPT and we can also use the fact that the edge weights of our graph satisfies the triangle inequality that is a direct connection of u to w is less then or equal to the sum of the distance from u to v and v to w

Final Remarks

- The currently best known approximation algorithm for metric TSP is Christofides' algorithm that achieves a factor of 1.5

Question

In the video, we've proved that for any undirected graph with non-negative edge weights the length of an optimum traveling salesman problem is at least the length of an minimum spanning tree. Is it true for general graphs or just for metric graphs, that is, graphs with edge weights satisfying the triangle inequalities)?

- (selected) Yes, for this inequality to hold, the edge weights should satisfy the triangle inequality.
- No, the triangle inequality is not needed for this inequality to hold.

✗ **Incorrect**

No, in fact, the triangle inequality is not needed. Recall the proof of this inequality. Take an optimal TSP cycle and remove any edge. This gives us a spanning tree (in fact, a spanning path) and this can only decrease the total length of a cycle (since edge weights are non-negative). Hence, at this point we don't need the triangle inequality. This inequality is used essentially in the analysis of the 2-approximation algorithm when bypassing an Eulerian cycle.

Final Remarks

- The currently best known approximation algorithm for metric TSP is Christofides' algorithm that achieves a factor of 1.5
- If $\mathbf{P} \neq \mathbf{NP}$, then there is no α-approximation algorithm for the general version of TSP for any polynomial time computable function α

Outline

1. Vertex cover
2. Traveling salesman
   - Metric TSP
   - Local search

LocalSearch

```
s ← some initial solution
while there is a solution s′ in the
neighborhood of s which is better than s:
  s ← s′
return s
```

LocalSearch

```
s ← some initial solution
while there is a solution s′ in the
neighborhood of s which is better than s:
  s ← s′
return s
```

- Computes a local optimum instead of a global optimum

LocalSearch

```
s ← some initial solution
while there is a solution s' in the
neighborhood of s which is better than s:
  s ← s'
return s
```

- Computes a local optimum instead of a global optimum
- The larger is the neighborhood, the better is the resulting solution and the higher is the running time

Thus this algorithm will be similar to brute force algorithm when neighborhood is large

Local Search for TSP

- Let s and s' be two cycles visiting each vertex of the graph exactly once

Local Search for TSP

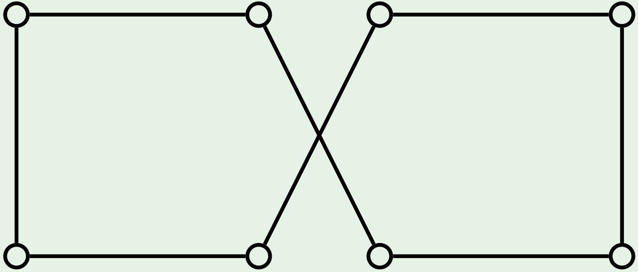
- Let s and s' be two cycles visiting each vertex of the graph exactly once
- The distance between s and s' is at most d, if one can get s' by deleting d edges from s and adding other d edges

Local Search for TSP

- Let s and s' be two cycles visiting each vertex of the graph exactly once
- The distance between s and s' is at most d, if one can get s' by deleting d edges from s and adding other d edges
- Neighborhood $N(s, r)$ with center s and radius r: all cycles with distance at most r from s

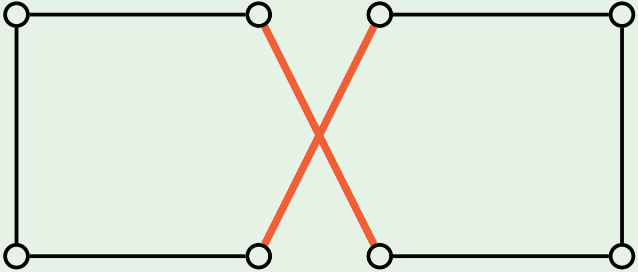
Example

Changing two edges in a suboptimal solution:



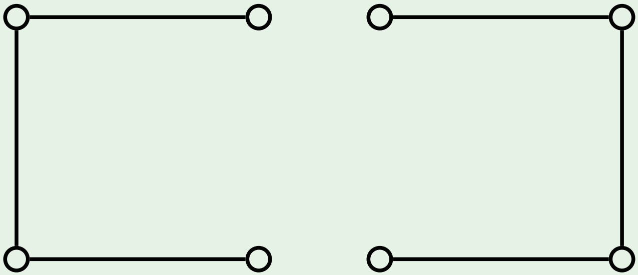
Example

Changing two edges in a suboptimal solution:



Example

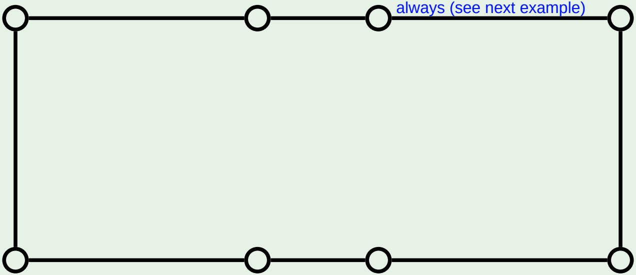
Changing two edges in a suboptimal solution:



Example

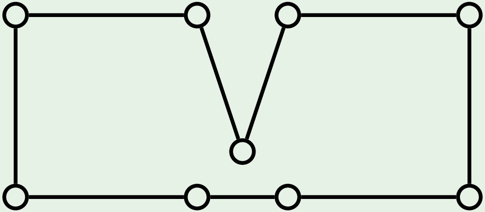
Changing two edges in a suboptimal solution:

[continue] Can give optimal solution sometimes but not always (see next example)



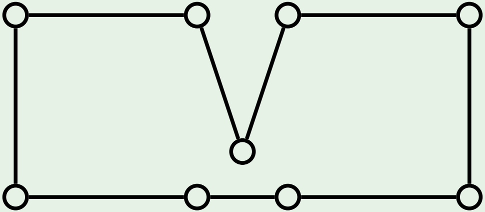
Example

A suboptimal solution that cannot be improved by changing two edges:



Example

A suboptimal solution that cannot be improved by changing two edges:



Need to allow changing three edges to improve this solution

Performance

- Trade-off between quality and running time of a single iteration

Meaning if we increase the size of neighborhood we increase the chances of finding better solution but running time increases

Performance

- Trade-off between quality and running time of a single iteration
- Still, the number of iterations may be exponential and the quality of the found cycle may be poor

Performance

- Trade-off between quality and running time of a single iteration
- Still, the number of iterations may be exponential and the quality of the found cycle may be poor

Algorithms can be designed like running the local search for limited number of steps for one starting point and if the satisfactory solution is not found then run the local search for completely different starting point likewise having many starting points

- But works well in practice

Coping with NP-completeness

- special cases
- intelligent exhaustive search
- approximation algorithms