Algorithmic Graph Theory
Part V - Approximation Algorithms for Graph Problems

Martin Milanič
martin.milanic@upr.si

University of Primorska, Koper, Slovenia

Dipartimento di Informatica
Università degli Studi di Verona, March 2013

Coping with NP-hardness

There are several approaches on how to deal with (the intractability of) NP-hard problems:

- polynomial algorithms for particular input instances
- approximation algorithms
- heuristics, local optimization
- “efficient” exponential algorithms
- randomized algorithms
- parameterized complexity
  (fixed-parameter tractable (FPT) algorithms)

What we'll do

1. Basic Definitions.
2. 2-Approximation Algorithm for Vertex Cover.
3. Approximation Algorithms for the Metric TSP.

BASICS OF APPROXIMATION ALGORITHMS.

Coping with NP-hardness

Heuristics:

- intuitive algorithms;

Coping with NP -hardness

Heuristics:

- intuitive algorithms;
- guaranteed to run in polynomial time;

Coping with NP -hardness

Heuristics:

- intuitive algorithms;
- guaranteed to run in polynomial time;
- no guarantee on quality of solution.

Coping with NP-hardness

Heuristics:

- intuitive algorithms;
- guaranteed to run in polynomial time;
- no guarantee on quality of solution.

Approximation algorithms:

- guaranteed to run in polynomial time;

Coping with NP -hardness

Heuristics:

- intuitive algorithms;
- guaranteed to run in polynomial time;
- no guarantee on quality of solution.

Approximation algorithms:

- guaranteed to run in polynomial time;
- guaranteed to find “high quality” solution, say within 1% of optimum;

Coping with NP -hardness

Heuristics:

- intuitive algorithms;
- guaranteed to run in polynomial time;
- no guarantee on quality of solution.

Approximation algorithms:

- guaranteed to run in polynomial time;
- guaranteed to find “high quality” solution, say within 1% of optimum;
- Obstacle:
  need to prove a solution’s value is close to optimum,
  without even knowing what the optimum value is!

Approximation Algorithms and Schemes

ρ-**approximation algorithm**:

- An algorithm A for an optimization problem Π that runs in polynomial time.

Approximation Algorithms and Schemes

ρ**-approximation algorithm**:

- An algorithm A for an optimization problem Π that runs in polynomial time.
- For every instance of Π, A outputs a feasible solution with objective function value within ratio ρ of true optimum for that instance.

Approximation Algorithms and Schemes

ρ-**approximation algorithm**:

- An algorithm A for an optimization problem Π that runs in polynomial time.
- For every instance of Π, A outputs a feasible solution with objective function value within ratio ρ of true optimum for that instance.

More specifically:

- for minimization problems:
  for every instance I, we have $f_A(I) \leq \rho \cdot \mathrm{OPT}(I)$, where

Approximation Algorithms and Schemes

ρ-**approximation algorithm**:

- An algorithm A for an optimization problem Π that runs in polynomial time.
- For every instance of Π, A outputs a feasible solution with objective function value within ratio ρ of true optimum for that instance.

More specifically:

- for minimization problems:
  for every instance I, we have $f_A(I) \leq \rho \cdot \mathrm{OPT}(I)$, where $f_A(I)$ is the value of the solution returned by the algorithm, and

Approximation Algorithms and Schemes

ρ-**approximation algorithm**:

- An algorithm A for an optimization problem Π that runs in polynomial time.
- For every instance of Π, A outputs a feasible solution with objective function value within ratio ρ of true optimum for that instance.

More specifically:

- for minimization problems:
  for every instance I, we have $f_A(I) \le \rho \cdot \mathrm{OPT}(I)$, where
  $f_A(I)$ is the value of the solution returned by the algorithm, and
  $\mathrm{OPT}(I)$ is the optimal solution value.
- for maximization problems: $f_A(I) \ge \mathrm{OPT}(I)/\rho$.

Approaches to the Design of Approximation Algorithms

There exist several approaches to the design of approximation algorithms:

- combinatorial algorithms,

Approaches to the Design of Approximation Algorithms

There exist several approaches to the design of approximation algorithms:

- combinatorial algorithms,
- algorithms based on linear programming,

Approaches to the Design of Approximation Algorithms

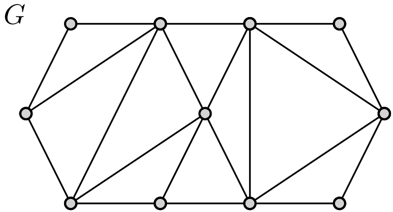
There exist several approaches to the design of approximation algorithms:

- combinatorial algorithms,
- algorithms based on linear programming,
- randomized algorithms,
- etc.

2-Approximation Algorithm for the Vertex Cover Problem

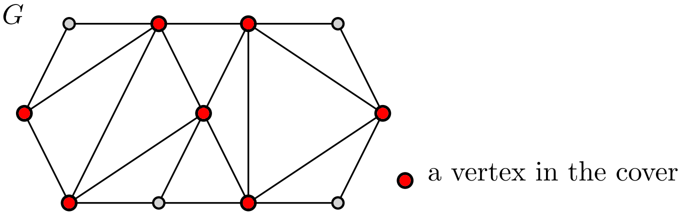
The Vertex Cover Problem

Recall:
vertex cover in a graph $G = (V, E)$:
a subset $C \subseteq V$ such that for all $e \in E$, $e \cap C \neq \emptyset$



The Vertex Cover Problem

Recall:
vertex cover in a graph $G = (V, E)$:
a subset $C \subseteq V$ such that for all $e \in E$, $e \cap C \neq \emptyset$



The Vertex Cover Problem

Consider the optimization version of the VERTEX COVER problem:

MINIMUM VERTEX COVER

Input: Graph $G = (V, E)$.

Task: Find a minimum vertex cover in G.

The Vertex Cover Problem

Consider the optimization version of the VERTEX COVER problem:

MINIMUM VERTEX COVER

Input: Graph $G = (V, E)$.

Task: Find a minimum vertex cover in G.

In bipartite graphs, the problem can be solved optimally in polynomial time.

The Vertex Cover Problem

Consider the optimization version of the VERTEX COVER problem:

MINIMUM VERTEX COVER	
Input:	Graph $G = (V, E)$.
Task:	Find a minimum vertex cover in G.

In bipartite graphs, the problem can be solved optimally in polynomial time.
For general graphs, the problem is NP-hard.

2-Approximation Algorithm for Vertex Cover

Approx-Cover:

2-Approximation Algorithm for Vertex Cover

Approx-Cover:
$C := \emptyset$;

2-Approximation Algorithm for Vertex Cover

Approx-Cover:
$C := \emptyset$;
while $(\exists e = uv \in E)(u, v \in V \setminus C)$ **do**

2-Approximation Algorithm for Vertex Cover

Approx-Cover:
$C := \emptyset$;
while $(\exists e = uv \in E)(u, v \in V \setminus C)$ **do**
    $C := C \cup \{u, v\}$

2-Approximation Algorithm for Vertex Cover

Approx-Cover:
$C := \emptyset$;
while $(\exists e = uv \in E)(u, v \in V \setminus C)$ **do**
    $C := C \cup \{u, v\}$
end while

2-Approximation Algorithm for Vertex Cover

Approx-Cover:
$C := \emptyset$;
while $(\exists e = uv \in E)(u, v \in V \setminus C)$ **do**
    $C := C \cup \{u, v\}$
end while
return C.

The algorithm computes an inclusion-wise maximal matching M and returns the union of all edges in the matching.

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* Minimum Vertex Cover *problem.*

Proof:

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

Proof:

The stopping criterion of the **while** loop guarantees that C is a cover.

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

Proof:

The stopping criterion of the **while** loop guarantees that C is a cover.

Clearly, the algorithm can be implemented to run in polynomial time.

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

Proof:

The stopping criterion of the **while** loop guarantees that C is a cover.

Clearly, the algorithm can be implemented to run in polynomial time.

Let M be the maximal matching consisting of all edges chosen by the algorithm.

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

Proof:

The stopping criterion of the **while** loop guarantees that C is a cover.

Clearly, the algorithm can be implemented to run in polynomial time.

Let M be the maximal matching consisting of all edges chosen by the algorithm.

Every vertex cover must contain at least one vertex of each edge of M,

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

Proof:
The stopping criterion of the **while** loop guarantees that C is a cover.
Clearly, the algorithm can be implemented to run in polynomial time.
Let M be the maximal matching consisting of all edges chosen by the algorithm.
Every vertex cover must contain at least one vertex of each edge of M,
hence $\mathrm{OPT} \geq |M|$

2-Approximation Algorithm for Vertex Cover

Claim

Approx-Cover *is a 2-approximation algorithm for the* MINIMUM VERTEX COVER *problem.*

Proof:

The stopping criterion of the **while** loop guarantees that C is a cover.

Clearly, the algorithm can be implemented to run in polynomial time.

Let M be the maximal matching consisting of all edges chosen by the algorithm.

Every vertex cover must contain at least one vertex of each edge of M,

hence $\mathrm{OPT} \geq |M|$

and consequently

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

Other (In)approximability issues

The factor of 2 in the analysis cannot be improved:

Other (In)approximability issues

The factor of 2 in the analysis cannot be improved:

- If $G = K_{n,n}$, then the algorithm returns $C = V(K_{n,n})$, while an optimal solution is of size n (either part of the bipartition).

Other (In)approximability issues

The factor of 2 in the analysis cannot be improved:

- If $G = K_{n,n}$, then the algorithm returns $C = V(K_{n,n})$, while an optimal solution is of size n (either part of the bipartition).

Remark:

- It can be shown that a natural **greedy algorithm** is not a ρ-approximation, for any ρ.

Other (In)approximability issues

The factor of 2 in the analysis cannot be improved:

- If $G = K_{n,n}$, then the algorithm returns $C = V(K_{n,n})$, while an optimal solution is of size n (either part of the bipartition).

Remark:

- It can be shown that a natural **greedy algorithm** is not a ρ-approximation, for any ρ.
  [**Greedy:** start with $C = \emptyset$. Until C is not a vertex cover, peel off and add to C a vertex of maximum degree in the current graph.]

Other (In)approximability issues

The factor of 2 in the analysis cannot be improved:

- If $G = K_{n,n}$, then the algorithm returns $C = V(K_{n,n})$, while an optimal solution is of size n (either part of the bipartition).

Remark:

- It can be shown that a natural **greedy algorithm** is not a ρ-approximation, for any ρ.
  [**Greedy:** start with $C = \emptyset$. Until C is not a vertex cover, peel off and add to C a vertex of maximum degree in the current graph.]

Inapproximability of vertex cover:

- If there exists a polynomial 1.36-approximation algorithm for MINIMUM VERTEX COVER, then P = NP (Dinur-Safra 2005).

Other (In)approximability issues

The factor of 2 in the analysis cannot be improved:

- If $G = K_{n,n}$, then the algorithm returns $C = V(K_{n,n})$, while an optimal solution is of size n (either part of the bipartition).

Remark:

- It can be shown that a natural **greedy algorithm** is not a ρ-approximation, for any ρ.
  [**Greedy:** start with $C = \emptyset$. Until C is not a vertex cover, peel off and add to C a vertex of maximum degree in the current graph.]

Inapproximability of vertex cover:

- If there exists a polynomial 1.36-approximation algorithm for MINIMUM VERTEX COVER, then P = NP (Dinur-Safra 2005).
- No ρ-approximation algorithm for MINIMUM VERTEX COVER is known with $\rho < 2$.

Approximation Algorithms for the Traveling Salesman Problem

The Traveling Salesman Problem

TRAVELING SALESMAN (TSP)

Input:	Graph $G = (V, E)$, a cost function $c : E \to \mathbb{R}+$.
Task:	Find a Hamiltonian cycle in G of smallest total cost.

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP*.*

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP*.*

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP.

Proof:

Suppose that *A* is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using *A*, whether a given graph *G* is Hamiltonian (which is an NP-complete problem).

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP*.*

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using A, whether a given graph G is Hamiltonian (which is an NP-complete problem).
HAMILTONIAN CYCLE: determine whether a given graph contains a cycle going through every vertex exactly once.

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP*.*

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using A, whether a given graph G is Hamiltonian (which is an NP-complete problem).
HAMILTONIAN CYCLE: determine whether a given graph contains a cycle going through every vertex exactly once.
I = instance for HAMILTONIAN CYCLE: a graph $G = (V, E)$.

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP*.*

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using A, whether a given graph G is Hamiltonian (which is an NP-complete problem).
HAMILTONIAN CYCLE: determine whether a given graph contains a cycle going through every vertex exactly once.
I = instance for HAMILTONIAN CYCLE: a graph $G = (V, E)$.
Let $G' = K_V$ (complete graph), $c : \binom{V}{2} \to \mathbb{R}_+$, where

$$c(e) = \begin{cases} 1, & \text{if } e \in E; \\ \rho|V| + 1, & \text{otherwise.} \end{cases}$$

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP.

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using A, whether a given graph G is Hamiltonian (which is an NP-complete problem).
HAMILTONIAN CYCLE: determine whether a given graph contains a cycle going through every vertex exactly once.
I = instance for HAMILTONIAN CYCLE: a graph $G = (V, E)$.
Let $G' = K_V$ (complete graph), $c : \binom{V}{2} \to \mathbb{R}_+$, where

$$c(e) = \begin{cases} 1, & \text{if } e \in E; \\ \rho|V| + 1, & \text{otherwise.} \end{cases}$$

Notice: G is Hamiltonian if and only if G' has a Hamiltonian cycle of total cost $|V|$.

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP.

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using A, whether a given graph G is Hamiltonian (which is an NP-complete problem).
HAMILTONIAN CYCLE: determine whether a given graph contains a cycle going through every vertex exactly once.
I = instance for HAMILTONIAN CYCLE: a graph $G = (V, E)$.
Let $G' = K_V$ (complete graph), $c : \binom{V}{2} \to \mathbb{R}_+$, where

$$c(e) = \begin{cases} 1, & \text{if } e \in E; \\ \rho|V| + 1, & \text{otherwise.} \end{cases}$$

Notice: G is Hamiltonian if and only if G' has a Hamiltonian cycle of total cost $|V|$.
Let Γ be the solution to the TSP given (K_V, c), computed by our algorithm A.

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP.

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using A, whether a given graph G is Hamiltonian (which is an NP-complete problem).
HAMILTONIAN CYCLE: determine whether a given graph contains a cycle going through every vertex exactly once.
I = instance for HAMILTONIAN CYCLE: a graph $G = (V, E)$.
Let $G' = K_V$ (complete graph), $c : \binom{V}{2} \to \mathbb{R}_+$, where

$$c(e) = \begin{cases} 1, & \text{if } e \in E; \\ \rho|V| + 1, & \text{otherwise.} \end{cases}$$

Notice: G is Hamiltonian if and only if G' has a Hamiltonian cycle of total cost $|V|$.
Let Γ be the solution to the TSP given (K_V, c), computed by our algorithm A.

(a) If $c(\Gamma) \leq \rho|V|$, then $\Gamma \subseteq E$, hence G is Hamiltonian.

The Traveling Salesman Problem

Proposition

If there exists a ρ-approximation algorithm for TSP for some $\rho \geq 1$, then P = NP.

Proof:
Suppose that A is a ρ-approximation algorithm for TSP.
We will show how to decide in polynomial time, using A, whether a given graph G is Hamiltonian (which is an NP-complete problem).
HAMILTONIAN CYCLE: determine whether a given graph contains a cycle going through every vertex exactly once.
I = instance for HAMILTONIAN CYCLE: a graph $G = (V, E)$.
Let $G' = K_V$ (complete graph), $c : \binom{V}{2} \to \mathbb{R}_+$, where

$$c(e) = \begin{cases} 1, & \text{if } e \in E; \\ \rho|V| + 1, & \text{otherwise.} \end{cases}$$

Notice: G is Hamiltonian if and only if G' has a Hamiltonian cycle of total cost $|V|$.
Let Γ be the solution to the TSP given (K_V, c), computed by our algorithm A.

(a) If $c(\Gamma) \leq \rho|V|$, then $\Gamma \subseteq E$, hence G is Hamiltonian.

(b) If $c(\Gamma) > \rho|V|$, then $\mathrm{OPT} > |V|$, hence G is not Hamiltonian. □

A Heuristic for TSP

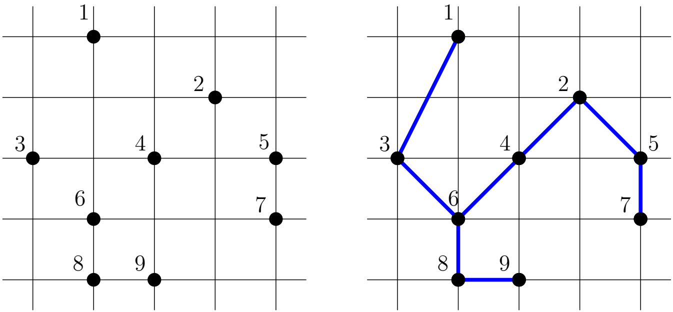
Approx-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).

A Heuristic for TSP

Approx-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).



input instance

minimum spanning tree T

(The cost of an egde connecting two vertices is equal to the **Euclidean distance** between them.)

A Heuristic for TSP

Approx-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).
- $W \leftarrow$ walk in which each edge appears exactly twice (obtained for example with depth-first search)

A Heuristic for TSP

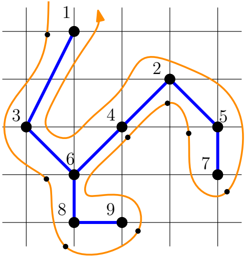
Approx-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).
- $W \leftarrow$ walk in which each edge appears exactly twice (obtained for example with depth-first search)
- $H \leftarrow$ cycle that visits vertices in the order as they appear in W for the first time

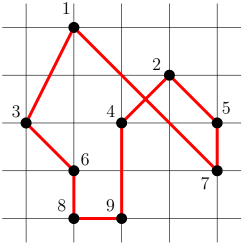
A Heuristic for TSP

Approx-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).
- $W \leftarrow$ walk in which each edge appears exactly twice (obtained for example with depth-first search)
- $H \leftarrow$ cycle that visits vertices in the order as they appear in W for the first time



walk W
(obtained with DFS)

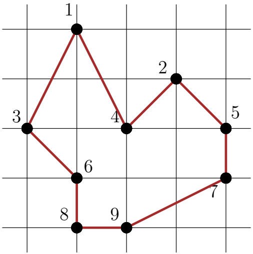


traveling salesman tour determined by W

A Heuristic for TSP

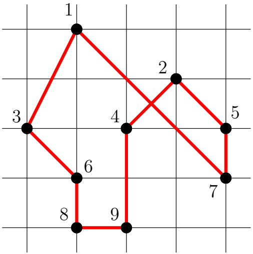
Approx-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).
- $W \leftarrow$ walk in which each edge appears exactly twice (obtained for example with depth-first search)
- $H \leftarrow$ cycle that visits vertices in the order as they appear in W for the first time



optimal solution

cost $\approx 13,95$



traveling salesman tour determined by W

cost $\approx 15,72$

The Metric TSP

METRIC TSP: TSP in which the cost function obeys the triangle inequality:
For all $u, v, w \in V$:

$$c(uw) \leq c(uv) + c(vw) .$$

The Metric TSP

METRIC TSP: TSP in which the cost function obeys the triangle inequality:
For all $u, v, w \in V$:

$$c(uw) \leq c(uv) + c(vw) .$$

A reduction from the HAMILTONIAN CYCLE problem shows:

Proposition

METRIC TSP *is NP-hard.*

The Metric TSP: a 2-approximation

Proposition

APPROX-TSP *is a 2-approximation algorithm for the* METRIC TSP *problem.*

The Metric TSP: a 2-approximation

Proposition

APPROX-TSP *is a 2-approximation algorithm for the* METRIC TSP *problem.*

Proof:

Let H^* be an optimal tour. We need to show: $c(H) \leq 2 \cdot c(H^*)$.

The Metric TSP: a 2-approximation

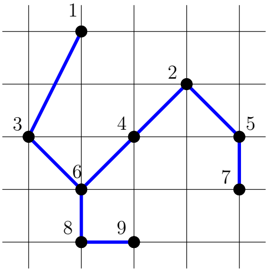
Proposition

APPROX-TSP *is a 2-approximation algorithm for the* METRIC TSP *problem.*

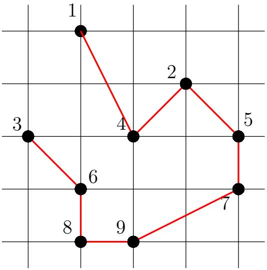
Proof:

Let H^* be an optimal tour. We need to show: $c(H) \leq 2 \cdot c(H^*)$.

- $c(T) \leq c(H^*)$, since removing any edge of H^* results in a spanning tree.



minimum spanning tree T



optimal solution H^* minus one edge

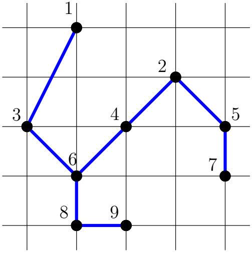
The Metric TSP: a 2-approximation

Proposition

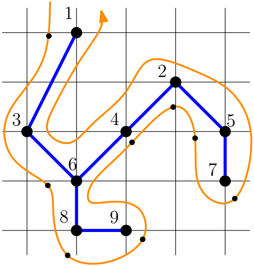
APPROX-TSP *is a 2-approximation algorithm for the* METRIC TSP *problem.*

Proof:

- $c(T) \leq c(H^*)$, since removing any edge of H^* results in a spanning tree.
- $c(W) = 2 \cdot c(T)$, since every edge is visited exactly twice.



minimum spanning tree T



walk W

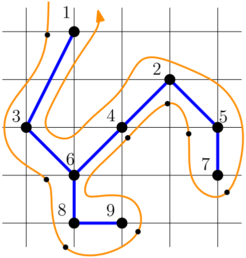
The Metric TSP: a 2-approximation

Proposition

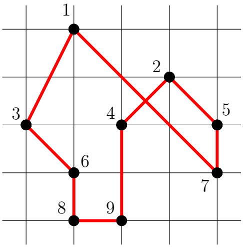
APPROX-TSP *is a 2-approximation algorithm for the* METRIC TSP *problem.*

Proof:

- $c(T) \leq c(H^*)$, since removing any edge of H^* results in a spanning tree.
- $c(W) = 2 \cdot c(T)$, since every edge is visited exactly twice.
- $c(H) \leq \cdot c(W)$, due to the triangle inequality.



walk W



traveling salesman tour determined by W

The Metric TSP: Christofides' Algorithm

Theorem (Christofides, 1976)

There exists a 1.5*-approximation algorithm for the* METRIC TSP *problem.*

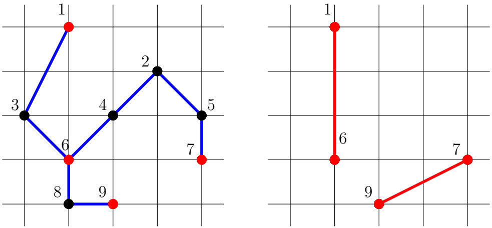
The Metric TSP: Christofides' Algorithm

Theorem (Christofides, 1976)

There exists a 1.5*-approximation algorithm for the* METRIC TSP *problem.*

Christofides-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).
- Find a minimum cost perfect matching M connecting vertices of odd degree in T.



minimum spanning tree T
vertices of odd degree are red

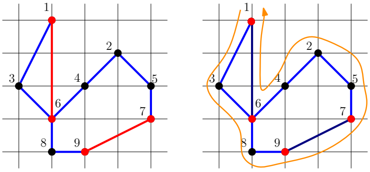
matching M

Theorem (Christofides, 1976)

There exists a 1.5*-approximation algorithm for the* METRIC TSP *problem.*

Christofides-TSP(G, c)

- Find a minimum spanning tree $T = (V, E_T)$ for (G, c).
- Find a minimum cost perfect matching M connecting vertices of odd degree in T.
- $G' \leftarrow T \cup M$
- $E \leftarrow$ Euler tour in G' (graph G' is Eulerian, since it is connected and has all vertices of even degree).



G' $\quad$ E = Euler tour in G'

The Metric TSP: Christofides' Algorithm

Proof:

Let H^* be an optimal tour. We need to show: $c(H) \leq 1.5 \cdot c(H^*)$.

The Metric TSP: Christofides' Algorithm

Proof:

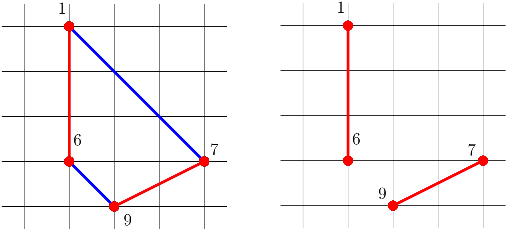
Let H^* be an optimal tour. We need to show: $c(H) \leq 1.5 \cdot c(H^*)$.

- $c(T) \leq c(H^*)$, as before

Proof:

Let H^* be an optimal tour. We need to show: $c(H) \leq 1.5 \cdot c(H^*)$.

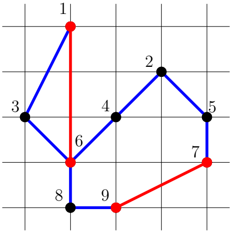
- $c(T) \leq c(H^*)$, as before
- $c(M) \leq (1/2) \cdot c(\Gamma^*) \leq (1/2) \cdot c(H^*)$, where Γ^* is an optimal cycle on the odd vertices of T.
  - The first inequality follows since Γ^* is the union of two perfect matchings.
  - The second inequality follows from the triangle inequality.



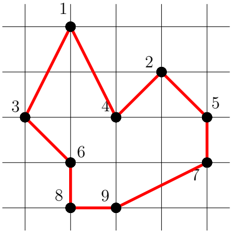
cycle Γ^* matching M

Proof:

- $c(T) \leq c(H^*)$, as before
- $c(M) \leq (1/2) \cdot c(\Gamma^*) \leq (1/2) \cdot c(H^*)$, where Γ^* is an optimal cycle on the odd vertices of T.
- Due to the triangle inquality:
  $c(H) \leq c(M) + c(T) \leq (3/2) \cdot c(H^*)$.



G'



H = traveling salesman tour

What we did – Week 1

1. Tue March 5: Review of basic notions in graph theory, algorithms and complexity ✓
2. Wed March 6: Graph colorings ✓
3. Thu March 7: Perfect graphs and their subclasses, part 1 ✓
4. Fri March 8: Perfect graphs and their subclasses, part 2 ✓

What we'll do – Week 2

1. Tue March 19: Further examples of tractable problems, part 1 ✓
2. Wed March 20:
   Further examples of tractable problems, part 2 ✓
   Approximation algorithms for graph problems ✓
3. Thu March 21: Lectio Magistralis lecture, *"Graph classes: interrelations, structure, and algorithmic issues"*

Questions?

Thank you for your attention!

martin.milanic@upr.si