

# Recent Progress in Approximation Algorithms for the Traveling Salesman Problem

Lecture 1: Some basic algorithms

David P. Williamson  
Cornell University

July 18-22, 2016  
São Paulo School of Advanced Science on  
Algorithms, Combinatorics, and Optimization



# The traveling salesman problem

---

## TRAVELING SALESMAN PROBLEM (TSP)

### Input:

- A complete, undirected graph  $G = (V, E)$ ;
- Edge costs  $c(e) \equiv c(i, j) \geq 0$  for all  $e = (i, j) \in E$ .

**Goal:** Find the min-cost tour that visits each city exactly once.

Costs are *symmetric* ( $c(i, j) = c(j, i)$ ) and obey the *triangle inequality* ( $c(i, k) \leq c(i, j) + c(j, k)$ ).

*Asymmetric* TSP (ATSP) input has complete directed graph, and  $c(i, j)$  may not equal  $c(j, i)$ .

# ATSP example

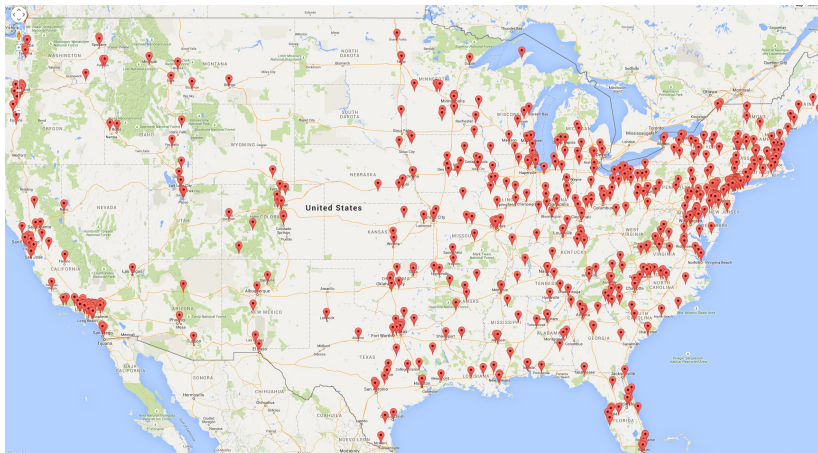
---



Example due to Ola Svensson (EPFL)

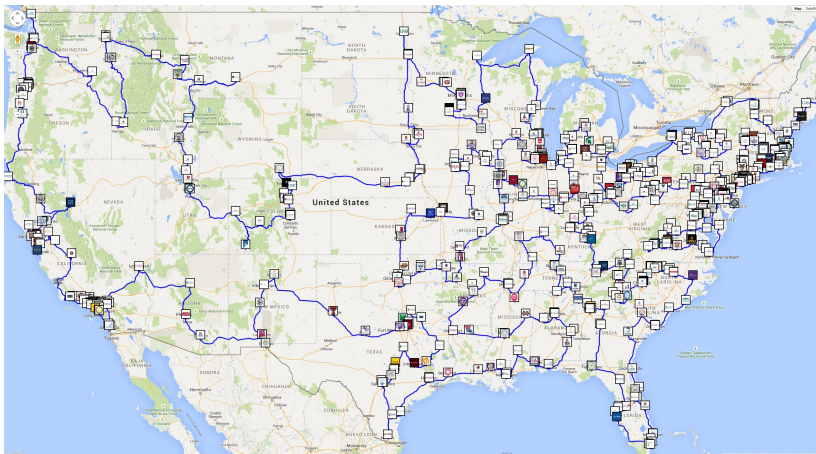
# The traveling salesman problem

---



From Bill Cook, tour of 647 US colleges  
([www.math.uwaterloo.ca/tsp/college](http://www.math.uwaterloo.ca/tsp/college))

# The traveling salesman problem



From Bill Cook, tour of 647 US colleges  
([www.math.uwaterloo.ca/tsp/college](http://www.math.uwaterloo.ca/tsp/college))

# Approximation Algorithms

---

## Definition

An  $\alpha$ -approximation algorithm is a polynomial-time algorithm that returns a solution of cost at most  $\alpha$  times the cost of an optimal solution.

## A simple approximation algorithm for TSP

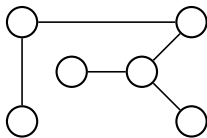
---

- Compute a minimum spanning tree (MST)  $F$  on  $G$ .

## A simple approximation algorithm for TSP

---

- Compute a minimum spanning tree (MST)  $F$  on  $G$ .

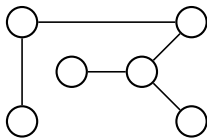




## A simple approximation algorithm for TSP

---

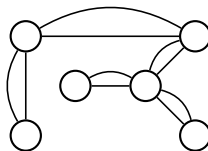
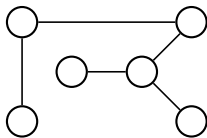
- Compute a minimum spanning tree (MST)  $F$  on  $G$ .
- Double every edge in  $F$ . The result is an *Eulerian graph*: it is connected, and every vertex has even degree.



## A simple approximation algorithm for TSP

---

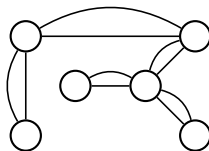
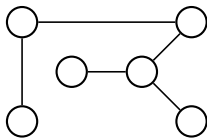
- Compute a minimum spanning tree (MST)  $F$  on  $G$ .
- Double every edge in  $F$ . The result is an *Eulerian graph*: it is connected, and every vertex has even degree.



## A simple approximation algorithm for TSP

---

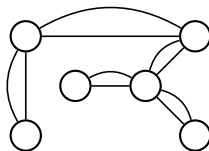
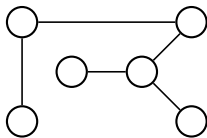
- Compute a minimum spanning tree (MST)  $F$  on  $G$ .
- Double every edge in  $F$ . The result is an *Eulerian graph*: it is connected, and every vertex has even degree.
- An Eulerian graph has an *traversal* that is easy to compute; it starts at any vertex  $v$ , visits every *edge*, and returns to  $v$ .



## A simple approximation algorithm for TSP

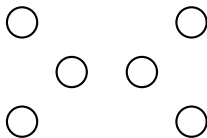
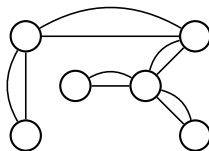
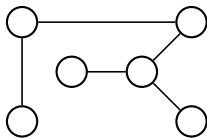
---

- Compute a minimum spanning tree (MST)  $F$  on  $G$ .
- Double every edge in  $F$ . The result is an *Eulerian graph*: it is connected, and every vertex has even degree.
- An Eulerian graph has an *traversal* that is easy to compute; it starts at any vertex  $v$ , visits every *edge*, and returns to  $v$ .
- Compute the traversal, and follow it; if the next edge goes back to a previously visited vertex, *shortcut it*, and go on to the next vertex in the traversal.



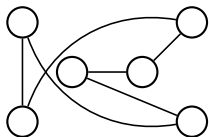
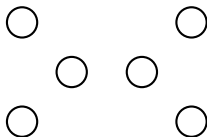
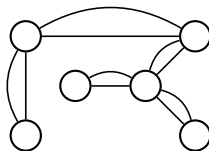
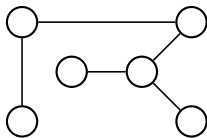
## A simple approximation algorithm for TSP

- Compute a minimum spanning tree (MST)  $F$  on  $G$ .
- Double every edge in  $F$ . The result is an *Eulerian graph*: it is connected, and every vertex has even degree.
- An Eulerian graph has an *traversal* that is easy to compute; it starts at any vertex  $v$ , visits every *edge*, and returns to  $v$ .
- Compute the traversal, and follow it; if the next edge goes back to a previously visited vertex, *shortcut it*, and go on to the next vertex in the traversal.



## A simple approximation algorithm for TSP

- Compute a minimum spanning tree (MST)  $F$  on  $G$ .
- Double every edge in  $F$ . The result is an *Eulerian graph*: it is connected, and every vertex has even degree.
- An Eulerian graph has an *traversal* that is easy to compute; it starts at any vertex  $v$ , visits every *edge*, and returns to  $v$ .
- Compute the traversal, and follow it; if the next edge goes back to a previously visited vertex, *shortcut it*, and go on to the next vertex in the traversal.



# Theorem

---

## Theorem

*This algorithm is a 2-approximation algorithm for the TSP.*

Let  $c(F)$  be the cost of the edges in the MST. Let  $OPT$  be the cost of the optimal tour.

## Lemma

$$c(F) \leq OPT.$$

## Proof.



# Proof of Theorem

---

Proof.

The cost of the tour given by the algorithm is at most  
 $2c(F) \leq 2OPT$ . □



## Proof of Theorem

---

Proof.

The cost of the tour given by the algorithm is at most  
 $2c(F) \leq 2OPT$ . □

Next: a better approximation algorithm due to Christofides (1976).

# Christofides' algorithm

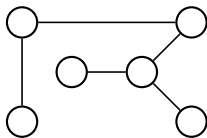
---

- Compute minimum spanning tree (MST)  $F$  on  $G$

# Christofides' algorithm

---

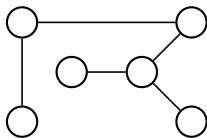
- Compute minimum spanning tree (MST)  $F$  on  $G$



# Christofides' algorithm

---

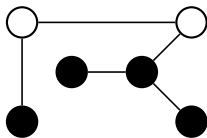
- Compute minimum spanning tree (MST)  $F$  on  $G$
- Compute a *minimum-cost perfect matching*  $M$  on odd-degree vertices of  $F$



# Christofides' algorithm

---

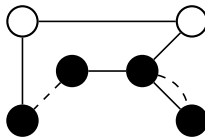
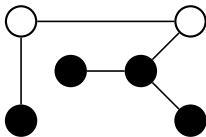
- Compute minimum spanning tree (MST)  $F$  on  $G$
- Compute a *minimum-cost perfect matching*  $M$  on odd-degree vertices of  $F$



## Christofides' algorithm

---

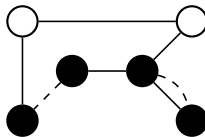
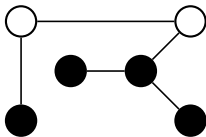
- Compute minimum spanning tree (MST)  $F$  on  $G$
- Compute a *minimum-cost perfect matching*  $M$  on odd-degree vertices of  $F$



## Christofides' algorithm

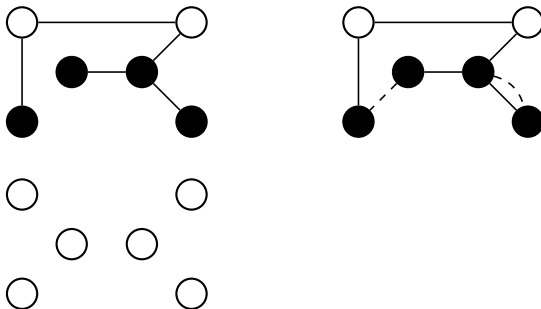
---

- Compute minimum spanning tree (MST)  $F$  on  $G$
- Compute a *minimum-cost perfect matching*  $M$  on odd-degree vertices of  $F$
- “Shortcut” Eulerian traversal in resulting Eulerian graph of  $F \cup M$



# Christofides' algorithm

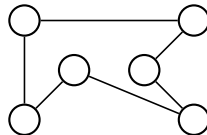
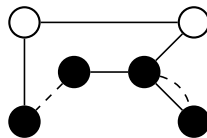
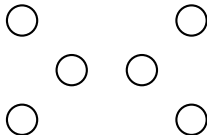
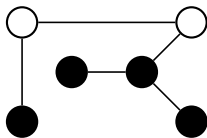
- Compute minimum spanning tree (MST)  $F$  on  $G$
- Compute a *minimum-cost perfect matching*  $M$  on odd-degree vertices of  $F$
- “Shortcut” Eulerian traversal in resulting Eulerian graph of  $F \cup M$





# Christofides' algorithm

- Compute minimum spanning tree (MST)  $F$  on  $G$
- Compute a *minimum-cost perfect matching*  $M$  on odd-degree vertices of  $F$
- “Shortcut” Eulerian traversal in resulting Eulerian graph of  $F \cup M$



# Theorem

---

## Theorem

*Christofides' algorithm is a  $\frac{3}{2}$ -approximation algorithm for the TSP.*

Let  $c(F)$  be the cost of the edges in the MST,  $c(M)$  the cost of the edges in the matching Let  $OPT$  be the cost of the optimal tour.

## Lemma

$$c(F) \leq OPT.$$

## Lemma

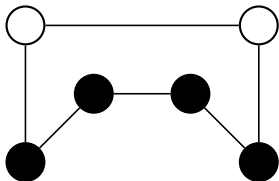
$$c(M) \leq \frac{1}{2}OPT.$$

Then the cost of the algorithm's tour is at most

$$c(F) + c(M) \leq OPT + \frac{1}{2}OPT = \frac{3}{2}OPT.$$

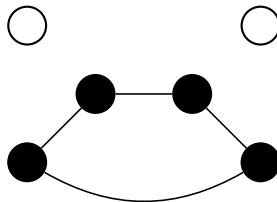
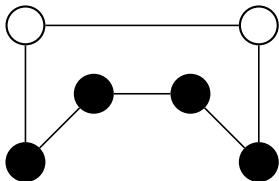
# Proof of Lemma

---



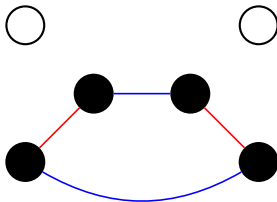
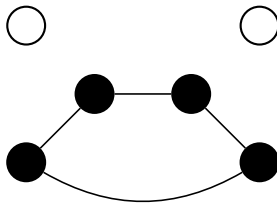
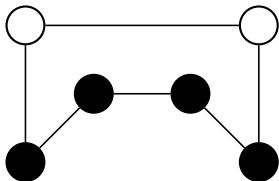
# Proof of Lemma

---



# Proof of Lemma

---



# Linear programs

---

A tool we'll start using frequently: *integer* and *linear programming*. We want to devise an integer program for the traveling salesman problem.

## Linear programs

---

A tool we'll start using frequently: *integer* and *linear programming*. We want to devise an integer program for the traveling salesman problem.

For each edge  $e \in E$ , we introduce a decision variable  $x_e$ , in which we want

$$x(e) = \begin{cases} 1 & \text{if tour uses edge } e \\ 0 & \text{otherwise} \end{cases}$$

## Linear programs

---

A tool we'll start using frequently: *integer* and *linear programming*. We want to devise an integer program for the traveling salesman problem.

For each edge  $e \in E$ , we introduce a decision variable  $x_e$ , in which we want

$$x(e) = \begin{cases} 1 & \text{if tour uses edge } e \\ 0 & \text{otherwise} \end{cases}$$

Then our *objective function* is to

$$\text{Minimize } \sum_{e \in E} c(e)x(e).$$



## Initial constraint

---

Let  $\delta(i)$  represent the set of all edges  $e$  that have  $i \in V$  as one endpoint. Then we want

$$\sum_{e \in \delta(i)} x(e) = 2$$

for all  $i \in V$ .

## Initial constraint

---

Let  $\delta(i)$  represent the set of all edges  $e$  that have  $i \in V$  as one endpoint. Then we want

$$\sum_{e \in \delta(i)} x(e) = 2$$

for all  $i \in V$ .

An initial integer programming formulation of the problem is then:

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in \delta(i)} x(e) = 2 \quad \forall i \in V,$$

$$x(e) \in \{0, 1\} \quad \forall e \in E.$$

## Initial constraint

---

Let  $\delta(i)$  represent the set of all edges  $e$  that have  $i \in V$  as one endpoint. Then we want

$$\sum_{e \in \delta(i)} x(e) = 2$$

for all  $i \in V$ .

An initial integer programming formulation of the problem is then:

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in \delta(i)} x(e) = 2 \quad \forall i \in V,$$

$$x(e) \in \{0, 1\} \quad \forall e \in E.$$

What might be a problem with this formulation?

# Initial IP

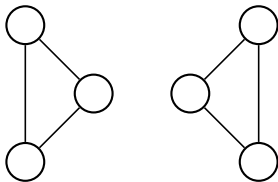
---

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in \delta(i)} x(e) = 2 \quad \forall i \in V$$
$$x(e) \in \{0, 1\} \quad \forall e \in E.$$

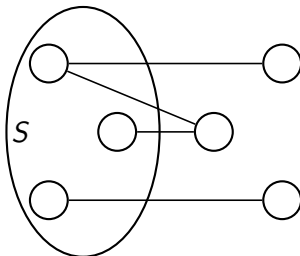
The following solution is feasible:



## Subtour elimination constraints

---

For  $S \subseteq V$ , let  $\delta(S)$  be the set of edges with one endpoint in  $S$ .



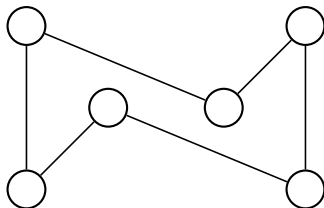
Then we want that

$$\sum_{e \in \delta(S)} x(e) \geq 2$$

for any set  $S$ .

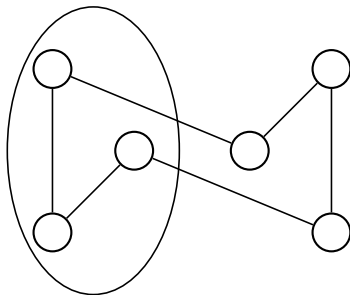
# LP relaxation

---



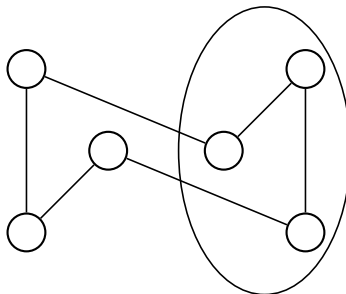
# LP relaxation

---



# LP relaxation

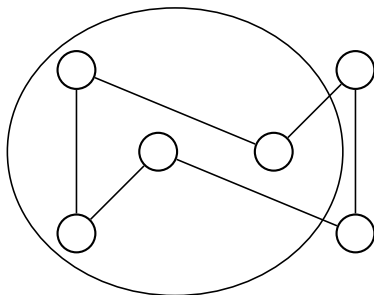
---





# LP relaxation

---



# Subtour LP

---

Thus an initial integer programming formulation is

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in \delta(i)} x(e) = 2 \quad \forall i \in V$$

$$\sum_{e \in \delta(S)} x(e) \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

$$x(e) \in \{0, 1\} \quad \forall e \in E.$$

## Subtour LP

---

Thus an initial integer programming formulation is

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in \delta(i)} x(e) = 2 \quad \forall i \in V$$

$$\sum_{e \in \delta(S)} x(e) \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

$$x(e) \in \{0, 1\} \quad \forall e \in E.$$

Replace  $x(e) \in \{0, 1\}$  by  $0 \leq x(e) \leq 1$  to obtain a *linear programming relaxation* called the *Subtour LP*.

## Subtour LP

---

Thus an initial integer programming formulation is

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in \delta(i)} x(e) = 2 \quad \forall i \in V$$

$$\sum_{e \in \delta(S)} x(e) \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

$$x(e) \in \{0, 1\} \quad \forall e \in E.$$

Replace  $x(e) \in \{0, 1\}$  by  $0 \leq x(e) \leq 1$  to obtain a *linear programming relaxation* called the *Subtour LP*.

If  $OPT_{IP}$  is value of IP optimal,  $OPT_{LP}$  value of LP optimal, how do they compare?

## Subtour LP

---

Thus an initial integer programming formulation is

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\sum_{e \in \delta(i)} x(e) = 2 \quad \forall i \in V$$

$$\sum_{e \in \delta(S)} x(e) \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

$$x(e) \in \{0, 1\} \quad \forall e \in E.$$

Replace  $x(e) \in \{0, 1\}$  by  $0 \leq x(e) \leq 1$  to obtain a *linear programming relaxation* called the *Subtour LP*.

If  $OPT_{IP}$  is value of IP optimal,  $OPT_{LP}$  value of LP optimal, how do they compare?

$$OPT_{LP} \leq OPT_{IP}.$$

## Subtour LP

---

We'll sometimes use the shorthand  $x(F) = \sum_{e \in F} x(e)$ , or  $c(F) = \sum_{e \in F} c(e)$ . For example, rewrite LP as

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$x(\delta(i)) = 2 \quad \forall i \in V$$

$$x(\delta(S)) \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

$$0 \leq x(e) \leq 1 \quad \forall e \in E.$$

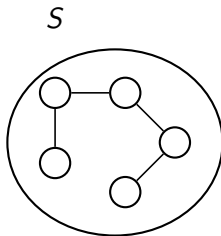
## Equivalent constraints

---

An equivalent way to write the subtour elimination constraints is via a constraint that says no cycles in any strict subset. Let  $E(S)$  be the set of edges with both endpoints in  $S$ ; then

$$x(E(S)) \leq |S| - 1$$

for all  $S \subset V, |S| \geq 2$ .



# Equivalent LP

---

So an LP that's equivalent to the subtour LP is the following:

$$\text{Minimize} \quad \sum_{e \in E} c(e)x(e)$$

subject to:

$$\begin{aligned} x(\delta(i)) &= 2 & \forall i \in V \\ x(E(S)) &\leq |S| - 1 & \forall S \subset V, |S| \geq 2 \\ 0 \leq x(e) &\leq 1 & \forall e \in E. \end{aligned}$$



## Christofides' again

---

Let  $OPT_{LP}$  be the optimal value of the linear programming. We can now (almost) prove the following.

Theorem (Wolsey (1980), Shmoys, W (1990))

*Christofides' algorithm returns a tour of cost at most  $\frac{3}{2}OPT_{LP}$ .*

## Christofides' again

Let  $OPT_{LP}$  be the optimal value of the linear programming. We can now (almost) prove the following.

Theorem (Wolsey (1980), Shmoys, W (1990))

*Christofides' algorithm returns a tour of cost at most  $\frac{3}{2}OPT_{LP}$ .*

To prove this, we need to show that for MST  $F$  and matching  $M$  on odd-degree vertices,

Lemma

$$c(F) \leq OPT_{LP}$$

Lemma

$$c(M) \leq \frac{1}{2}OPT_{LP}$$

## Proof of first lemma

---

The minimum spanning tree can be found as the solution to the following LP (Edmonds 1971):

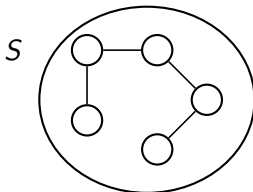
$$\text{Minimize} \quad \sum_{e \in E} c(e)z(e)$$

subject to:

$$z(E) = |V| - 1$$

$$z(E(S)) \leq |S| - 1 \quad \forall S \subset V, |S| \geq 2$$

$$z(e) \geq 0 \quad \forall e \in E.$$



# Proof of first lemma

---

$$c(F) \leq OPT_{LP}$$

$$\text{Min } \sum_{e \in E} c(e)x(e)$$

$$x(\delta(i)) = 2 \quad \forall i \in V$$

$$x(E(S)) \leq |S| - 1 \quad \forall S \subset V, |S| \geq 2$$

$$0 \leq x(e) \leq 1 \quad \forall e \in E.$$

$$\text{Min } \sum_{e \in E} c(e)z(e)$$

$$z(E) = |V| - 1$$

$$z(E(S)) \leq |S| - 1 \quad \forall S \subset V, |S| \geq 2$$

$$z(e) \geq 0 \quad \forall e \in E.$$

## Proof of second lemma

---

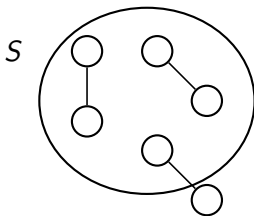
The minimum-cost perfect matching can be found as the solution to the following LP (Edmonds 1965):

$$\text{Minimize} \quad \sum_{e \in E} c(e)z(e)$$

subject to:

$$z(\delta(i)) = 1 \quad \forall i \in V$$

$$z(\delta(S)) \geq 1 \quad \forall S \subset V, |S| \text{ odd.}$$



# Proof of second lemma

---

$$c(M) \leq \frac{1}{2} OPT_{LP}$$

$$\text{Min } \sum_{e \in E} c(e)x(e)$$

$$x(\delta(i)) = 2 \quad \forall i \in V$$

$$x(\delta(S)) \geq 2 \quad \forall S \subset V, S \neq \emptyset$$

$$0 \leq x(e) \leq 1 \quad \forall e \in E.$$

$$\text{Min } \sum_{e \in E} c(e)z(e)$$

$$z(\delta(i)) = 1 \quad \forall i \in V$$

$$z(\delta(S)) \geq 1 \quad \forall S \subset V, |S| \text{ odd}$$

$$z(e) \geq 0 \quad \forall e \in E.$$

## Missing part

---

We also need that the value of the subtour LP can only go down as we remove vertices from the instance (Shmoys, W 1990), so that we can consider a matching only on the odd-degree vertices.

## Missing part

---

We also need that the value of the subtour LP can only go down as we remove vertices from the instance (Shmoys, W 1990), so that we can consider a matching only on the odd-degree vertices.

### Theorem

$$c(F) + c(M) \leq \frac{3}{2} OPT_{LP},$$

*so that Christofides' algorithm returns a solution of cost at most this much.*



# Update

---

“But I thought you were going to talk about *recent* approximation algorithms for the traveling salesman problem...”

# Update

---

“But I thought you were going to talk about *recent* approximation algorithms for the traveling salesman problem...”

For TSP, no better approximation algorithm known that Christofides' algorithm.

# Graph TSP

---

Progress made for two different special cases: *graph TSP* and the *s-t path TSP*.

# Graph TSP

---

Progress made for two different special cases: *graph TSP* and the *s-t path TSP*.

Graph TSP: Input is connected graph  $G = (V, E)$  and cost  $c(i, j)$  is number of edges in shortest path from  $i$  to  $j$  in  $G$ .

Oveis Gharan, Saberi, Singh	2010	$3/2 - \epsilon$	
Mömke, Svensson	2011	1.461	
Mömke, Svensson	2011	$4/3$	if graph <i>subcubic</i>
Mucha	2011	$13/9$	
Sebő and Vygen	2012	1.4	

## $s$ - $t$ path TSP

---

The  $s$ - $t$  path TSP:

Usual TSP input plus  $s, t \in V$ , find a min-cost path from  $s$  to  $t$  visiting all other nodes in between (an  $s$ - $t$  *Hamiltonian path*).

Hoogeveen (1991) shows that the natural variant of Christofides' algorithm gives a  $\frac{5}{3}$ -approximation algorithm.

## $s$ - $t$ path TSP

---

The  $s$ - $t$  path TSP:

Usual TSP input plus  $s, t \in V$ , find a min-cost path from  $s$  to  $t$  visiting all other nodes in between (an  $s$ - $t$  *Hamiltonian path*).

Hoogeveen (1991) shows that the natural variant of Christofides' algorithm gives a  $\frac{5}{3}$ -approximation algorithm.

An, Kleinberg, Shmoys	2012	1.618
Sebő	2013	1.6
Vygen	2015	1.599
Gottschalk and Vygen	2015	1.56
Sebő and Van Zuylen	2016	1.52

# Agenda

---

For the rest of the week:

- Wednesday: Graph TSP (4/3 for cubic, 2-vertex-connected graphs)
- Thursday:  $s$ - $t$  path TSP
- Friday:  $s$ - $t$  path TSP for graph TSP, open questions