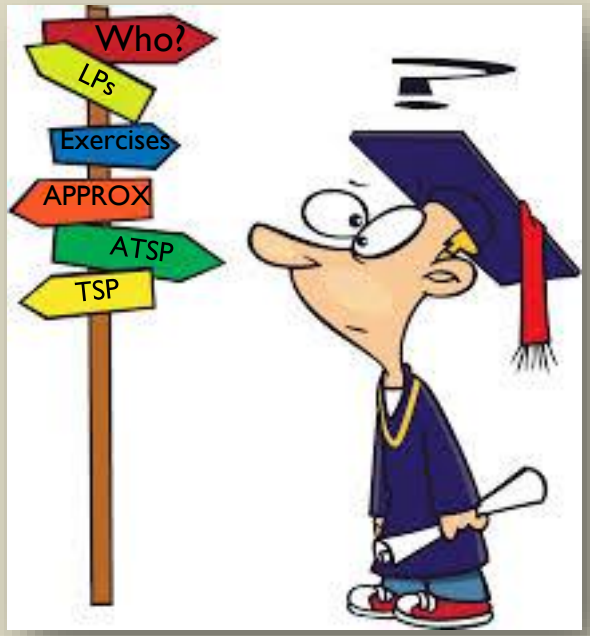
Lecture 1: Symmetric Traveling Salesman Problem

Ola Svensson

ADFOCS 2015



WHO AM I AND WHAT WILL I TALK ABOUT?

About Me

Ola in action

- Ola Svensson
  - ola.svensson@epfl.ch
  - http://theory.epfl.ch/osven
- Assistant Prof at EPFL
  - Research on algorithms
- Happy to get feedback and answer questions

Outline

LECTURE 1: Traveling Salesman Problem

LECTURE 2: Traveling Salesman Problem

LECTURE 3: Traveling Salesman Problem

Outline

LECTURE 1: Traveling Salesman Problem

Symmetric TSP, Christofides' Algorithm, Removable Edges, Open Problems

LECTURE 2: Traveling Salesman Problem

Asymmetric TSP, Cycle Cover Algorithm, Thin trees

LECTURE 3: Traveling Salesman Problem

Continuation of asymmetric TSP, Local-Connectivity Algorithm, Open Problems

The Irresistible Traveling Salesman Problem

What is the cheapest way to visit these cities?

The Irresistible Traveling Salesman Problem

What is the cheapest way to visit these cities?

The Irresistible Traveling Salesman Problem

What is the cheapest way to visit these cities?

Traveling Salesman Problem

The Irresistible Traveling Salesman Problem

The Irresistible Traveling Salesman Problem

33 city contest that Proctor and Gamble ran in 1962

Rich History

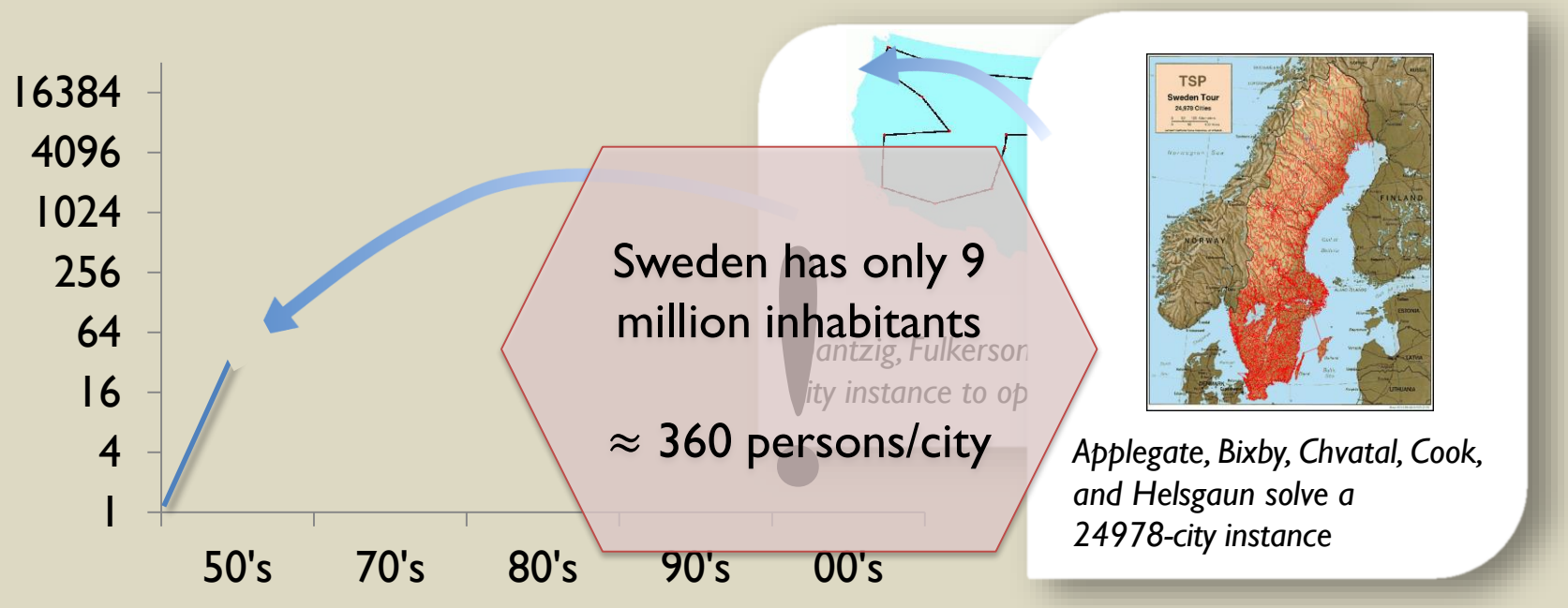
- Variants studied in mathematics by Hamilton and Kirkman already in the 1800's
- Benchmark problem in computer science from the "beginning"
- Today, probably the most studied NP-hard optimization problem
- Intractable: (current) exact algorithms require exponential time

Major open problem what efficient computation can accomplish

HOW TO EVALUATE AN ALGORITHM

Solving intractable problems

- Heuristics
  - good for "typical" instances
  - bad instances do not happen too often



Solving intractable problems

- Approximation Algorithms
  - Perhaps we can efficiently find a reasonably good solution in polytime?

Approximation Ratio:

$$\alpha = \frac{\text{cost}(\text{Found Solution})}{\text{cost}(\text{Optimal Solution})}$$

worst case over all instances

- α=1 is an exact polynomial time algorithm
- α=1.01 then algorithm finds a solution with at most 1% higher cost

MOTIVATION OF TODAY'S LECTURE

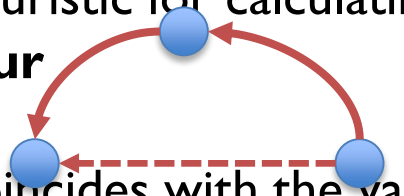
Approximation algorithms for symmetric TSP

What is the best possible algorithm?

1970's

Christofides: **1.5-approximation** algorithm for metric distances

Held & Karp: Heuristic for calculating **lower bound on a tour**



Coincides with the value of a **linear program** known as Held-Karp or Subtour Elimination Relaxation

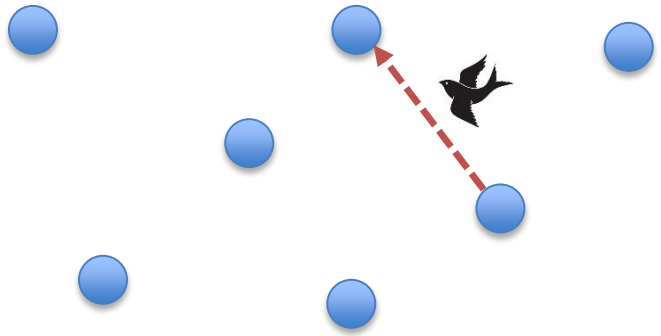
1980's

• • •

1990's

Arora & Mitchell independently:

PTAS for **Euclidian TSP**



1990's

Arora & Mitchell independently:

PTAS for **Euclidian TSP**

Arora et al. and Grigni et al.

PTAS for **planar TSP**

2000's

Papadimitriou & Vempala:

NP-hard to approximate metric TSP within 220/219

Simplified and slightly improved by Lampis'12

Today

Major open problem to understand the approximability of TSP

- **NP-hard** to approximate metric TSP within 220/219
- Christofides' **1.5-approximation algorithm** still best
- Held-Karp relaxation conjectured to give **4/3-approximation**

TODAY'S Lecture

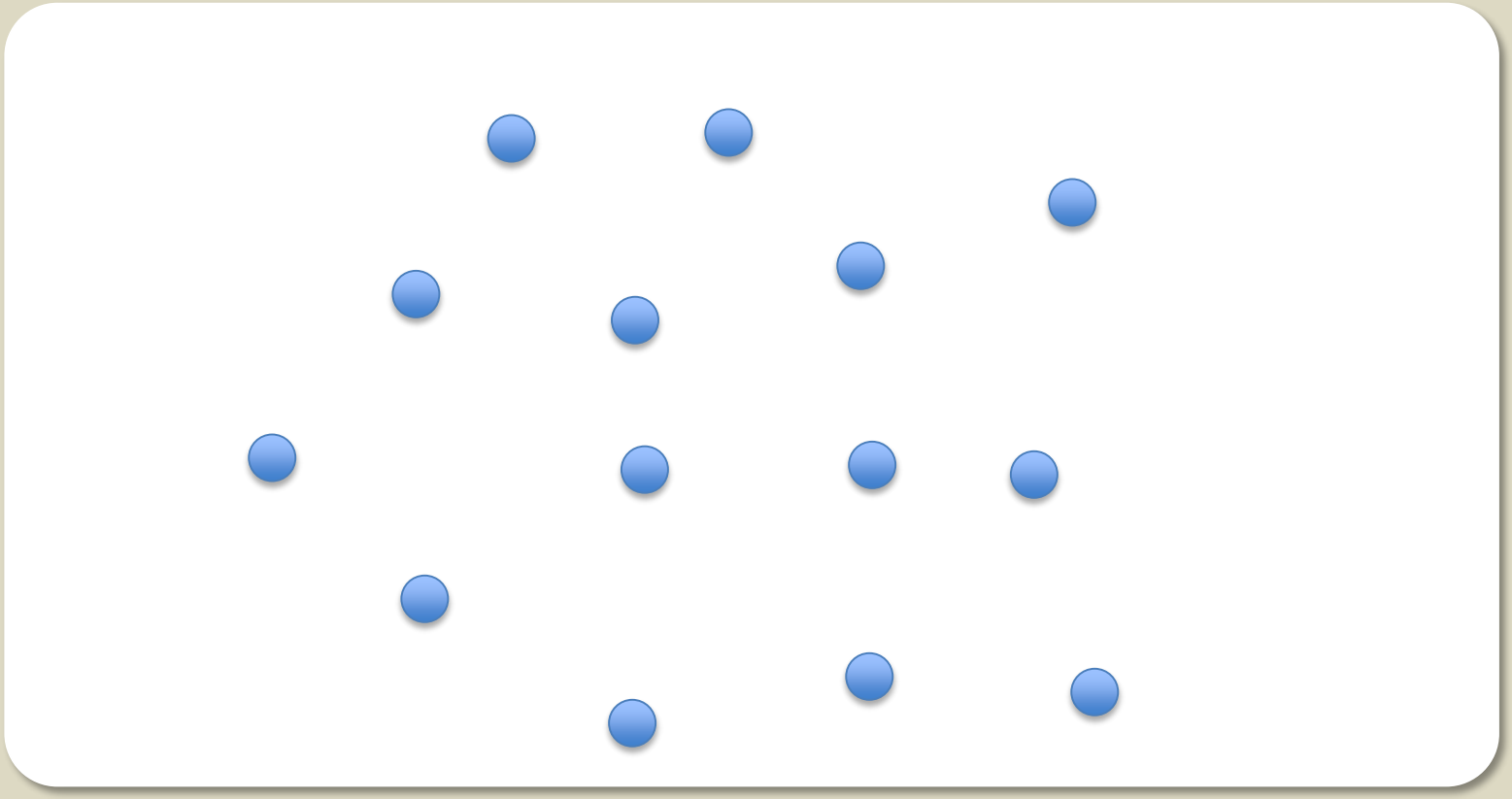
- The first approximation algorithm for TSP
- Christofides' Algorithm
- Recent techniques that improve upon Christofides' algorithm for important special cases

OUR FIRST APPROXIMATION ALGORITHM

The Traveling Salesman Problem

INPUT: n cities with pairwise distances that *satisfy the triangle inequality*

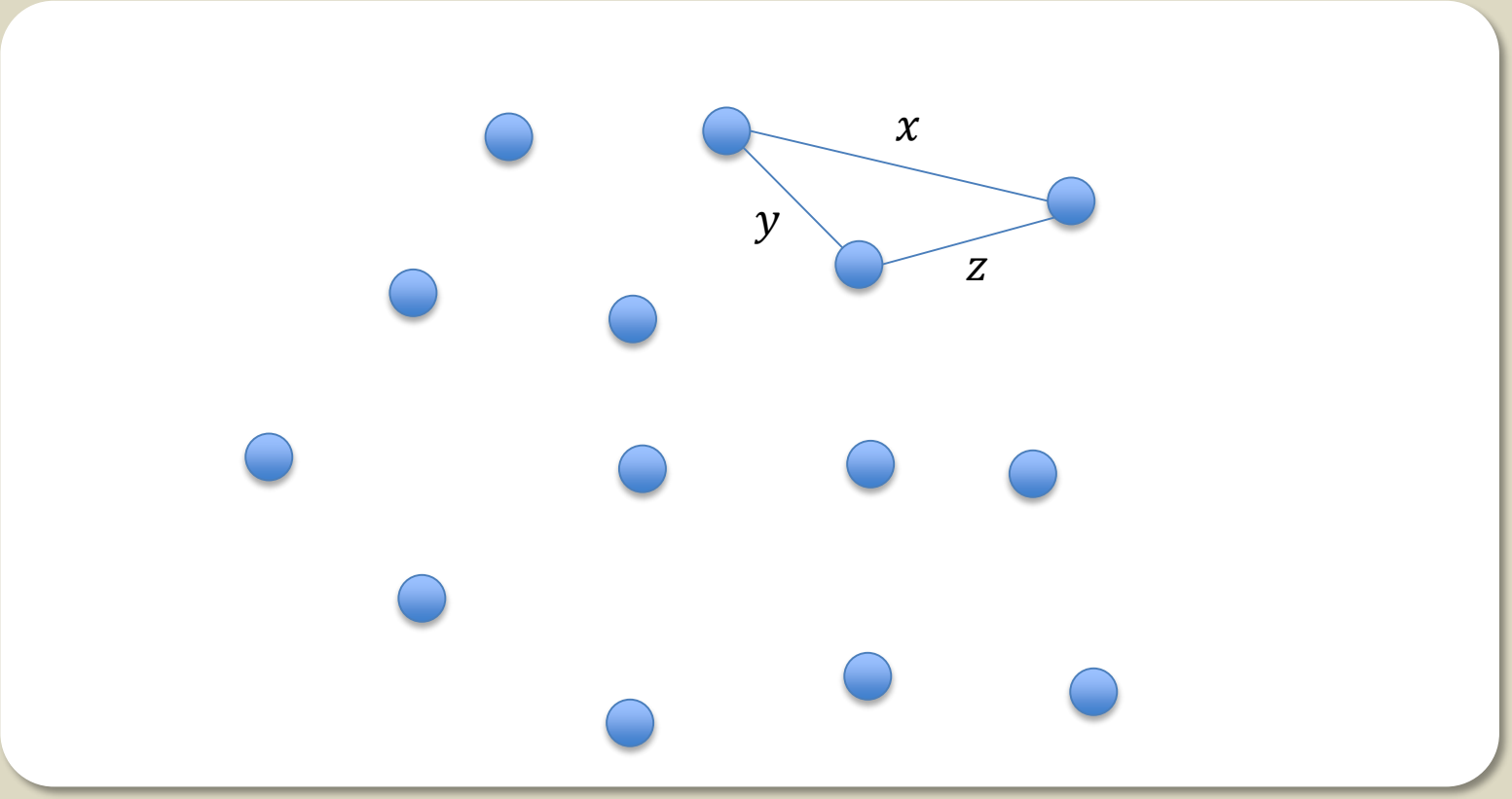
OUTPUT: a tour of minimum total distance that visits each city once



The Traveling Salesman Problem

INPUT: n cities with pairwise distances that *satisfy the triangle inequality*

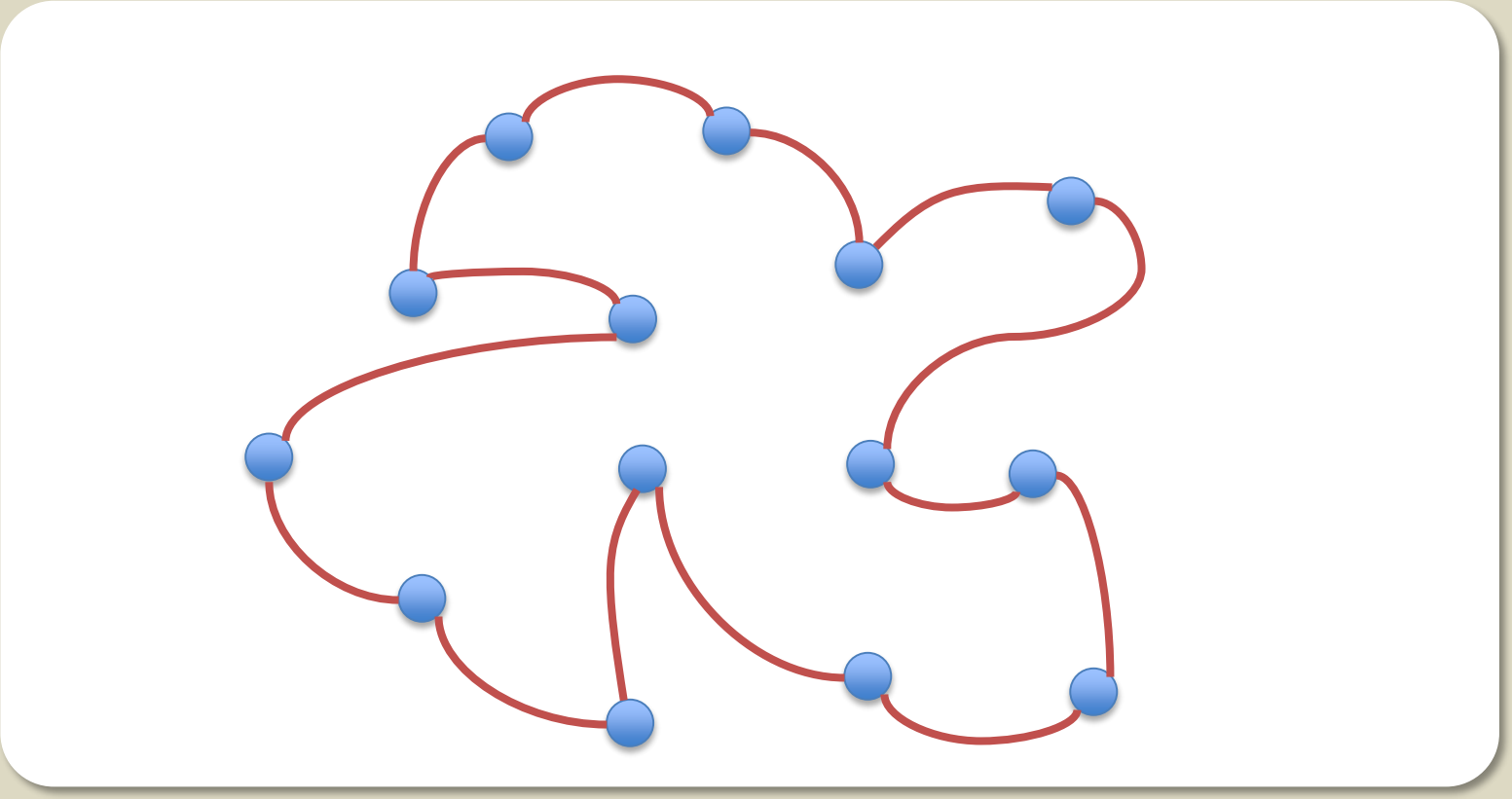
OUTPUT: a tour of minimum total distance that visits each city once



The Traveling Salesman Problem

INPUT: n cities with pairwise distances that *satisfy the triangle inequality*

OUTPUT: a tour of minimum total distance that visits each city once



How to analyze an approximation algorithm?

- Recall that we measure its performance by

Approximation Ratio:

$$\alpha = \frac{\text{cost}(\text{Found Solution})}{\text{cost}(\text{Optimal Solution})}$$

worst case over all instances

- But calculating the cost of the optimal solution is NP-hard…

SOLUTION: Compare with a lower bound on OPT!

What is a Lower bound on OPT?

What is a Lower bound on OPT?

The weight of a minimum spanning tree is at most OPT:

$$w(MST) \leq OPT$$

What is a Lower bound on OPT?

The weight of a minimum spanning tree is at most OPT:

$$w(MST) \leq OPT$$

PROOF:

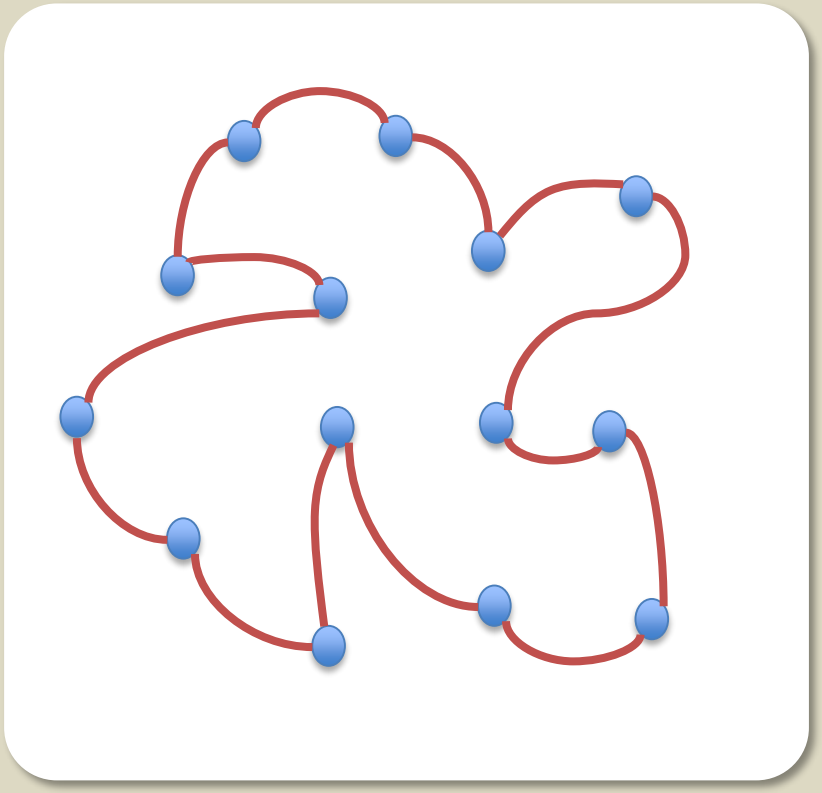
What is a Lower bound on OPT?

The weight of a minimum spanning tree is at most OPT:

$$w(MST) \leq OPT$$

PROOF:

- Take an optimal tour of cost OPT
- Drop an edge to obtain a tree T
- Distances/weights are non-negative so $w(T) \leq OPT$
- Hence, $w(MST) \leq OPT$

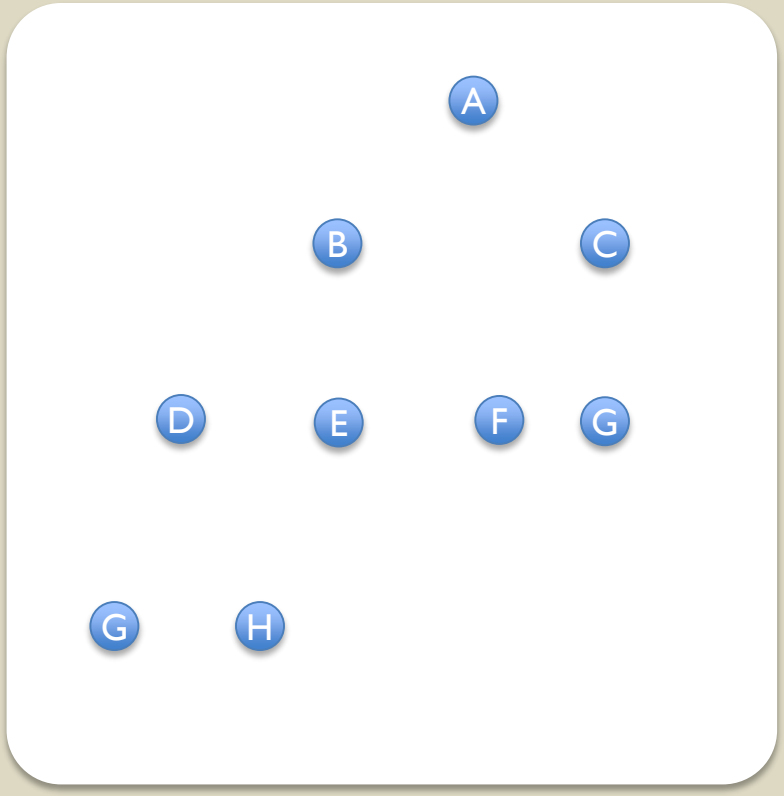


Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

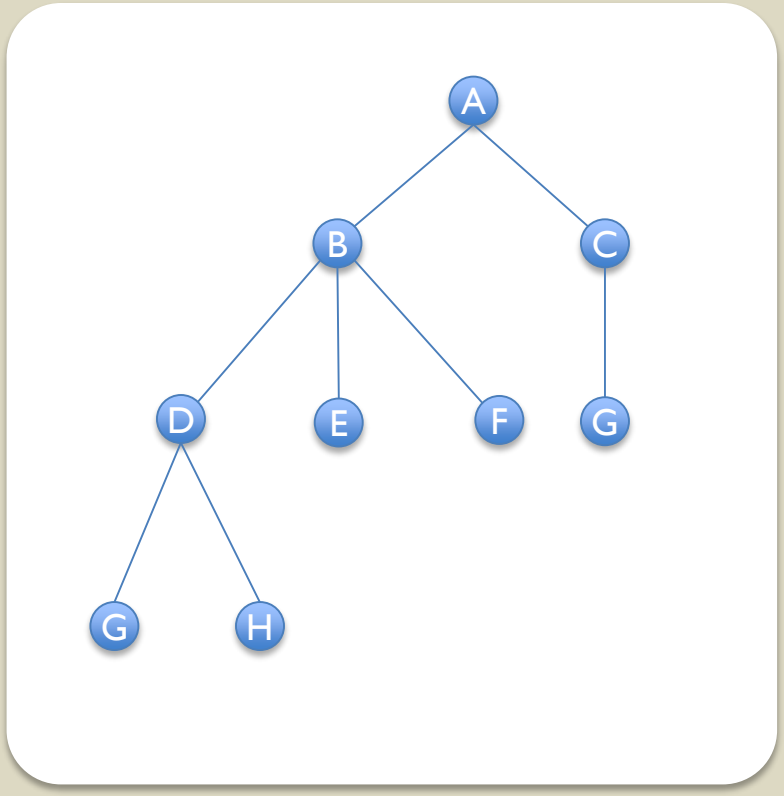


Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

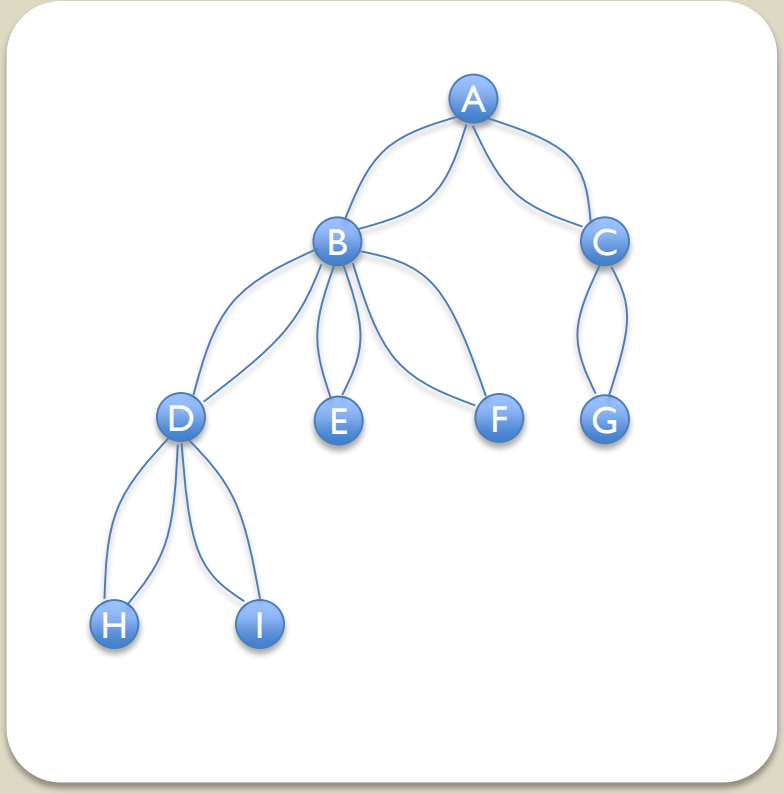


Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

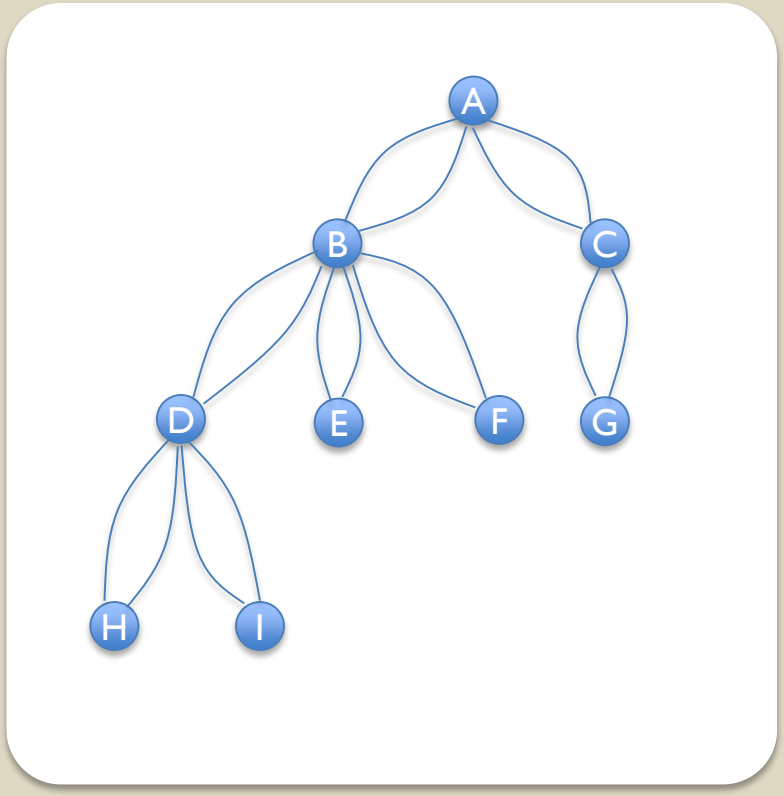


Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting



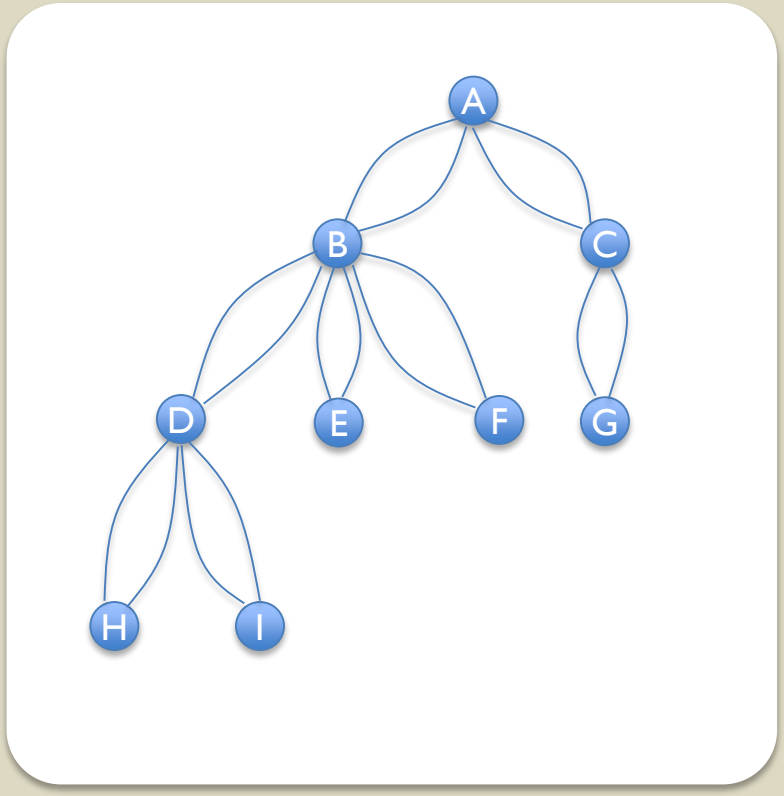
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



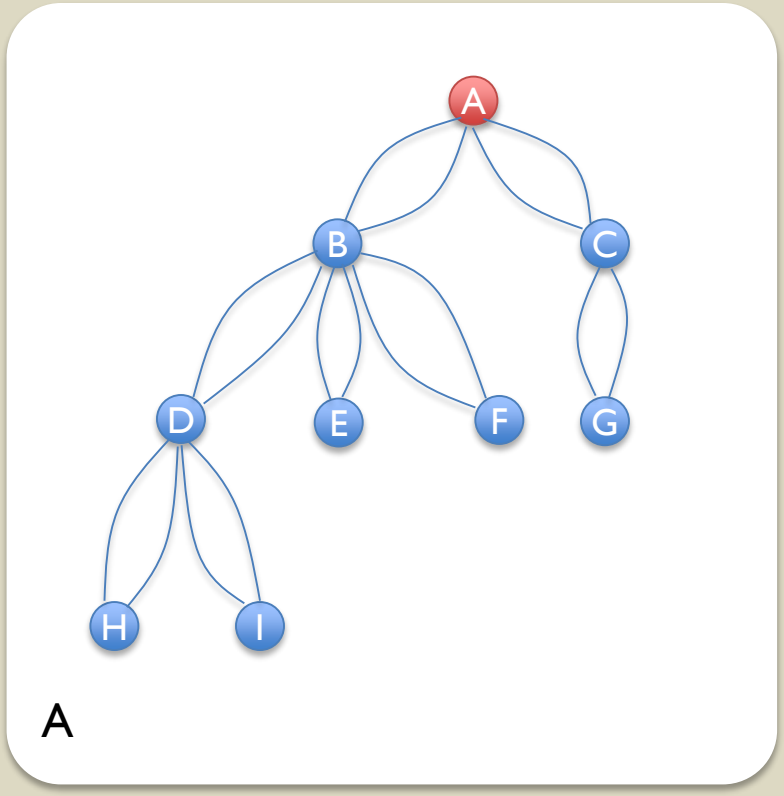
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



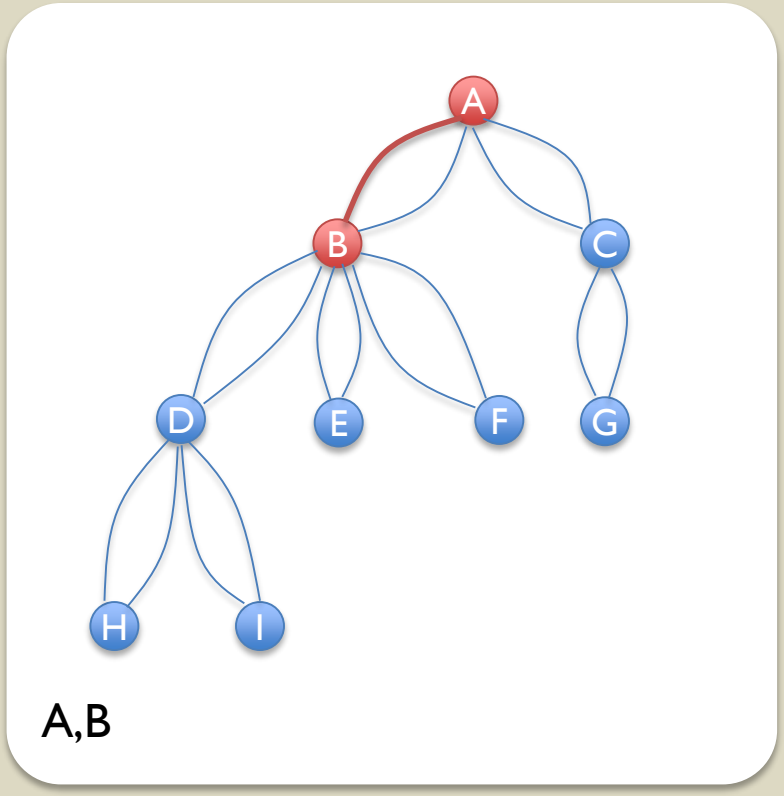
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



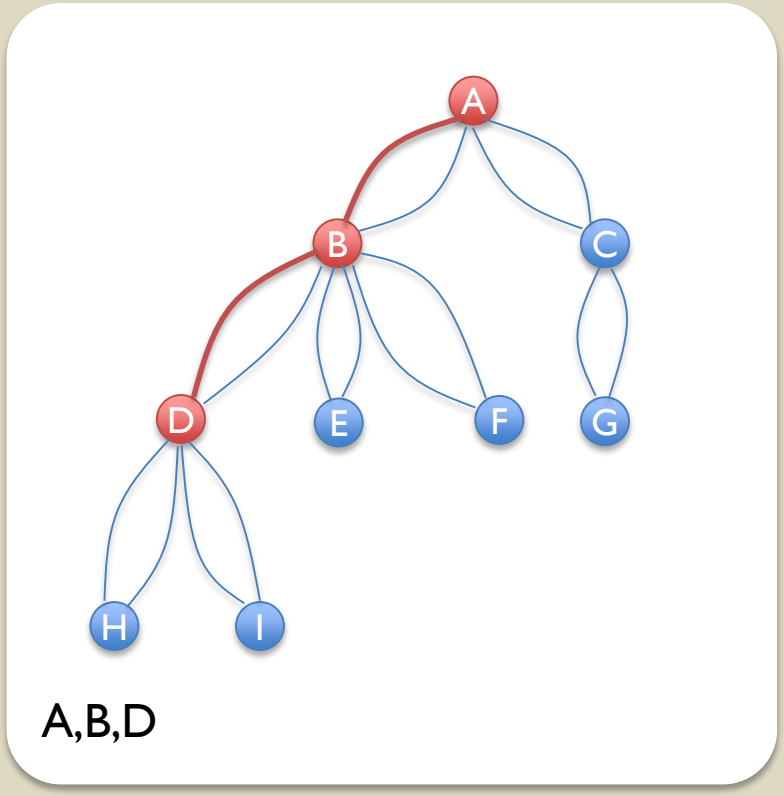
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



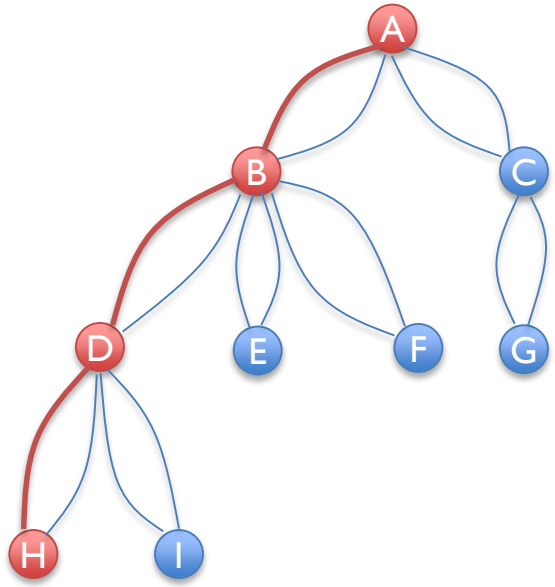
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H

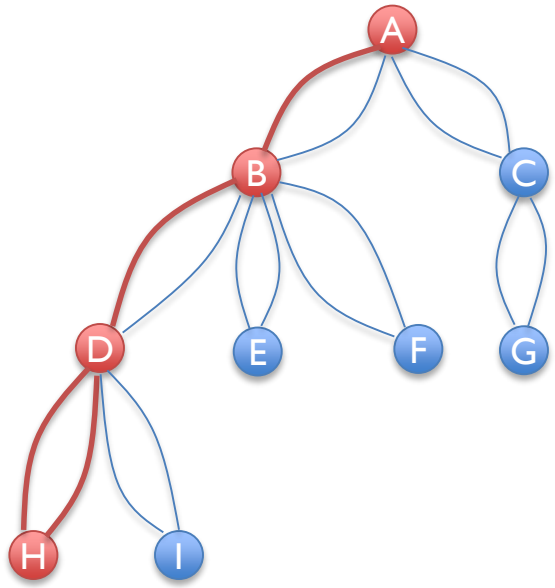
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D

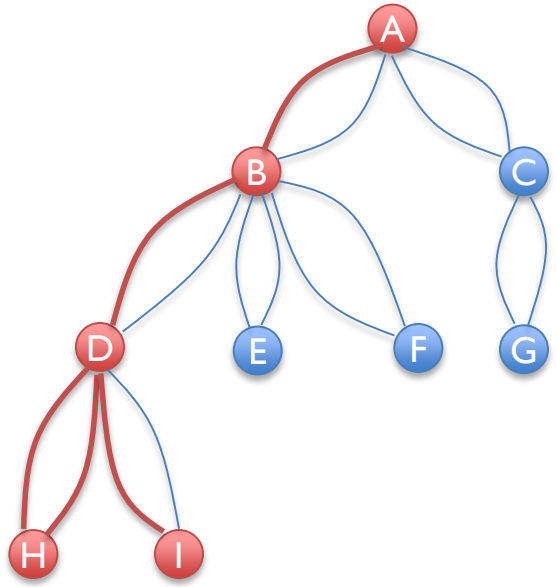
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I

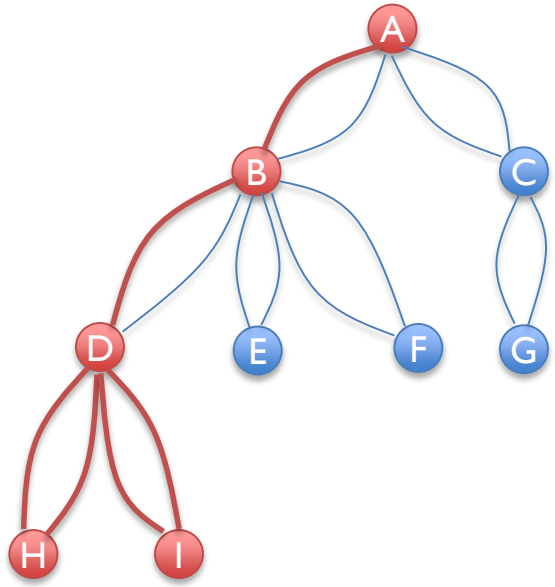
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D

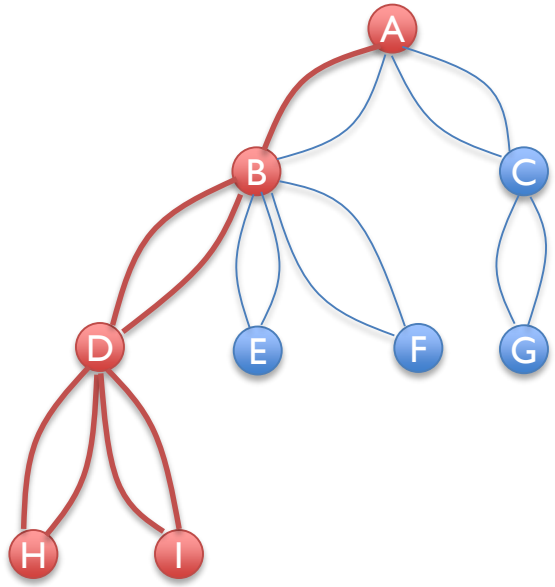
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B

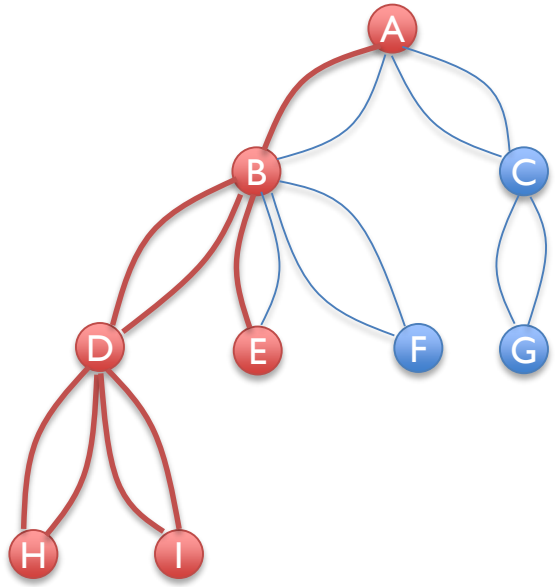
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E

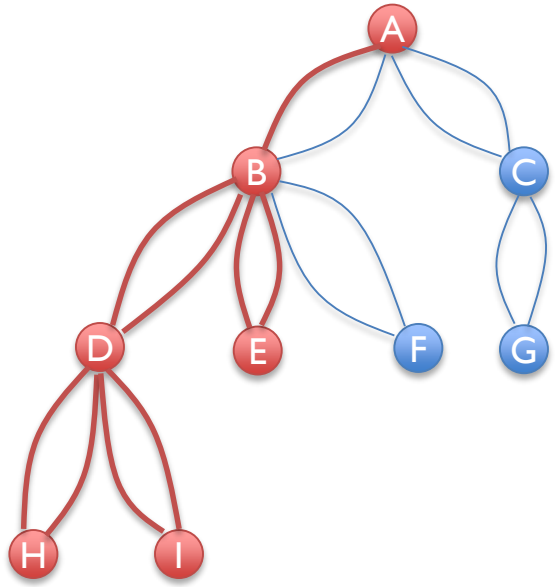
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B

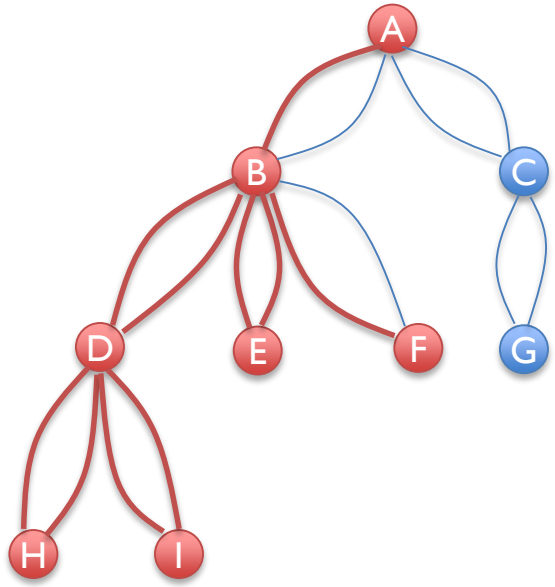
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B,F

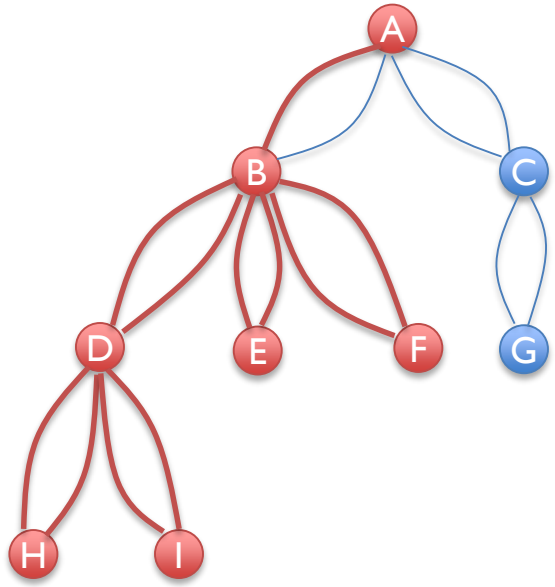
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B,F,B

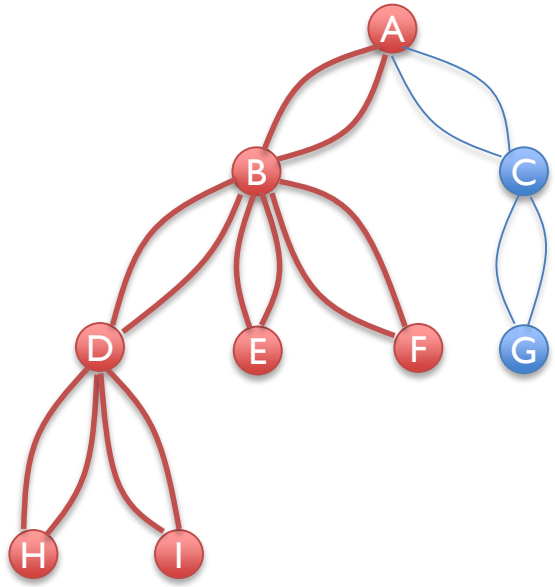
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B,F,B,A

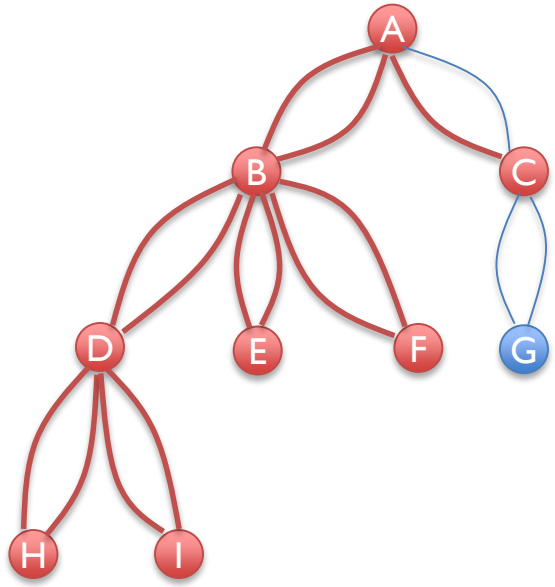
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B,F,B,A,C

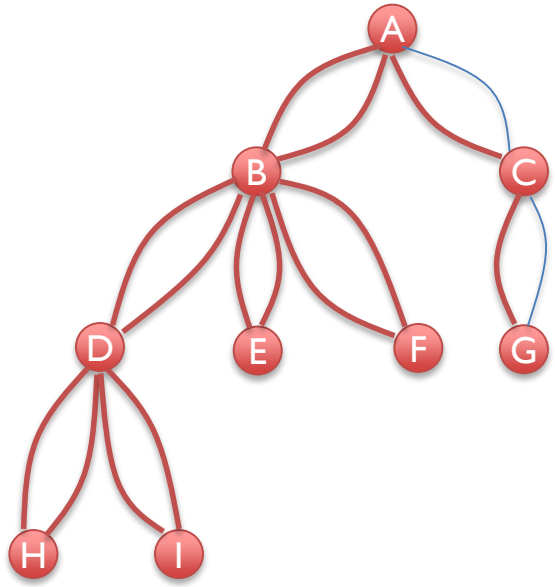
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B,F,B,A,C,G

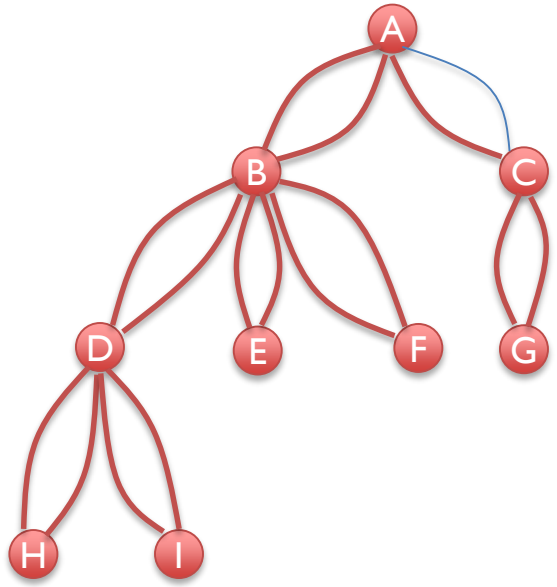
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B,F,B,A,C,G,C

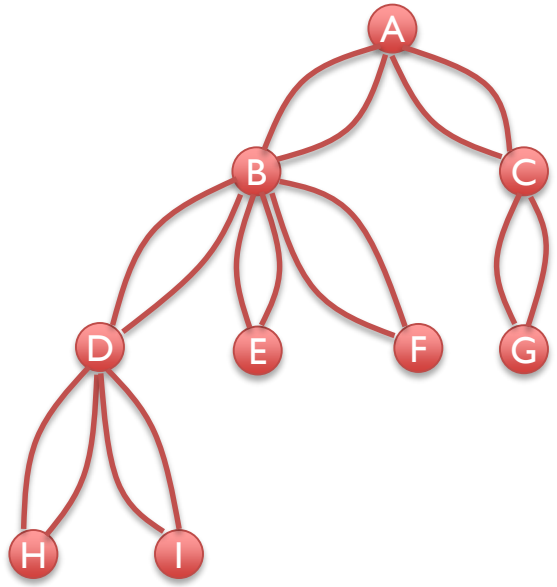
Double Tree Algorithm

Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once



A,B,D,H,D,I,D,B,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

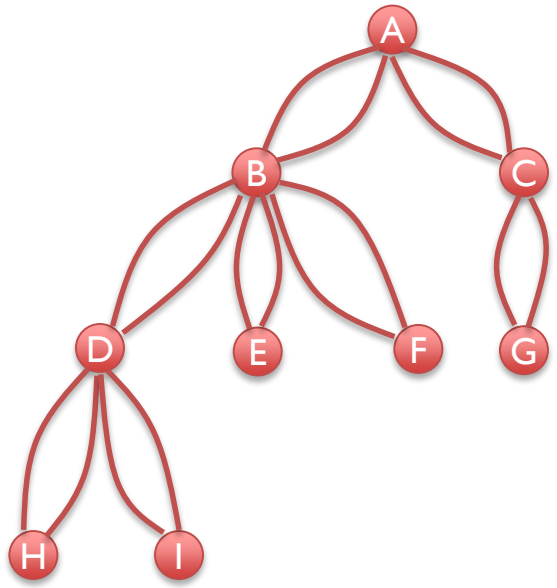
Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once

Short cut to visit each vertex exactly once. By triangle inequality this doesn't increase cost



A,B,D,H,D,I,D,B,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

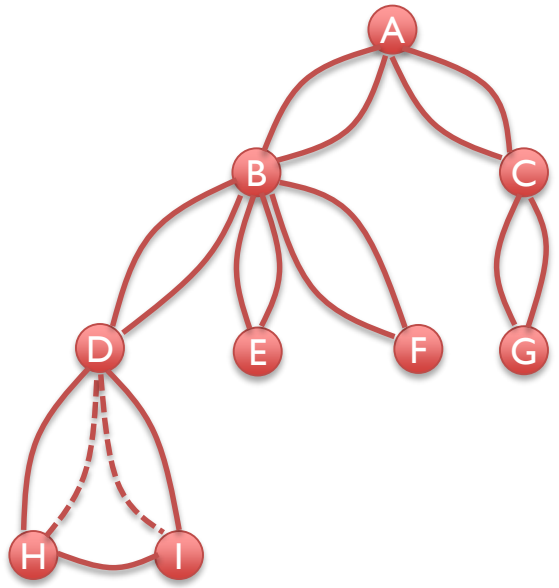
Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once

Short cut to visit each vertex exactly once. By triangle inequality this doesn't increase cost



A,B,D,H,~~D~~,I,D,B,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

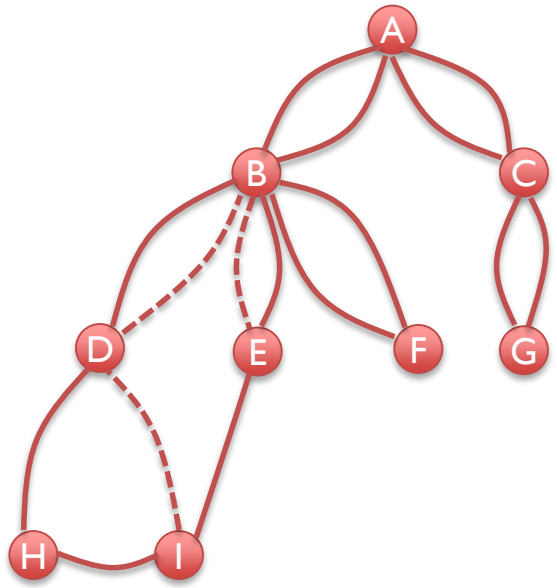
Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once

Short cut to visit each vertex exactly once. By triangle inequality this doesn't increase cost



A,B,D,H,~~D~~,I,~~D~~,~~B~~,E,B,F,B,A,C,G,C,A

Double Tree Algorithm

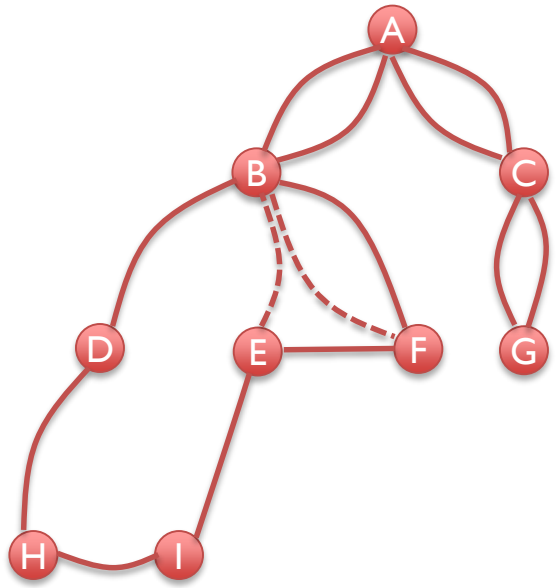
Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once

Short cut to visit each vertex exactly once. By triangle inequality this doesn't increase cost



A,B,D,H,~~D~~,I,~~D~~,~~B~~,E,~~B~~,F,B,A,C,G,C,A

Double Tree Algorithm

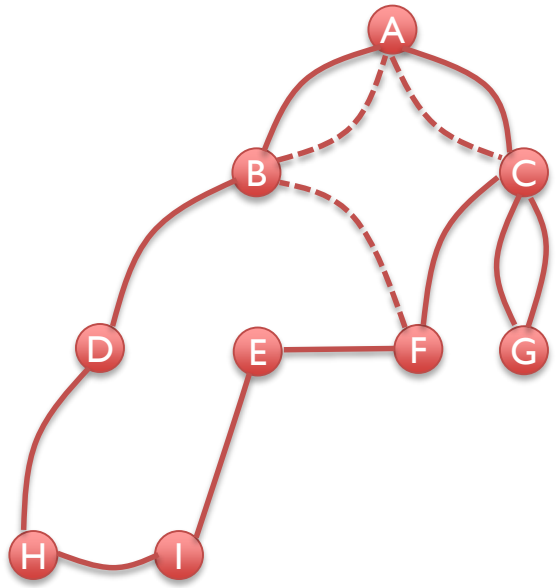
Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once

Short cut to visit each vertex exactly once. By triangle inequality this doesn't increase cost



A,B,D,H,~~D~~,I,~~D~~,~~B~~,E,~~B~~,F,~~B~~,~~A~~,C,G,C,A

Double Tree Algorithm

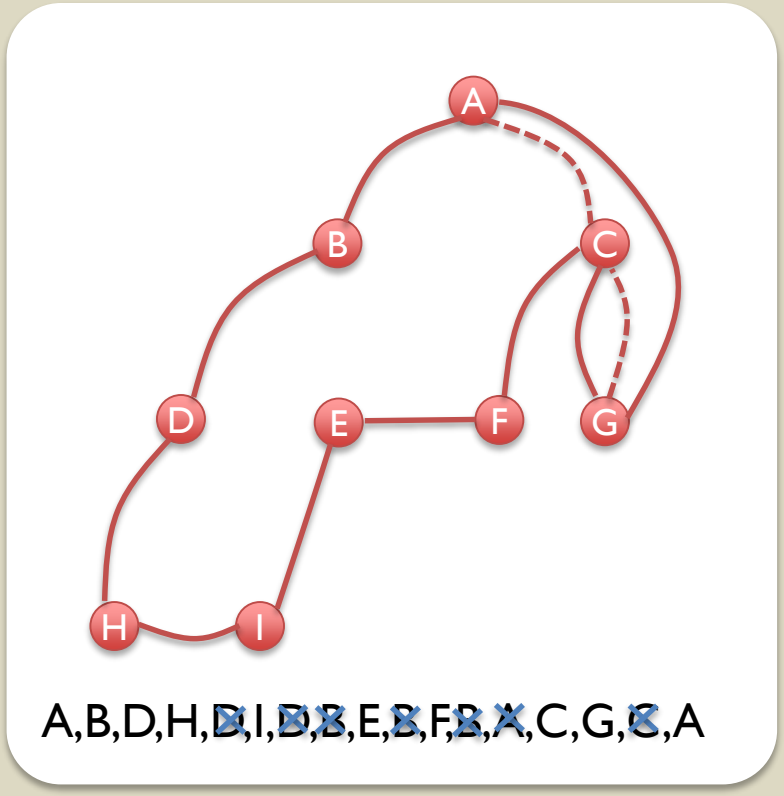
Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once

Short cut to visit each vertex exactly once. By triangle inequality this doesn't increase cost



Double Tree Algorithm

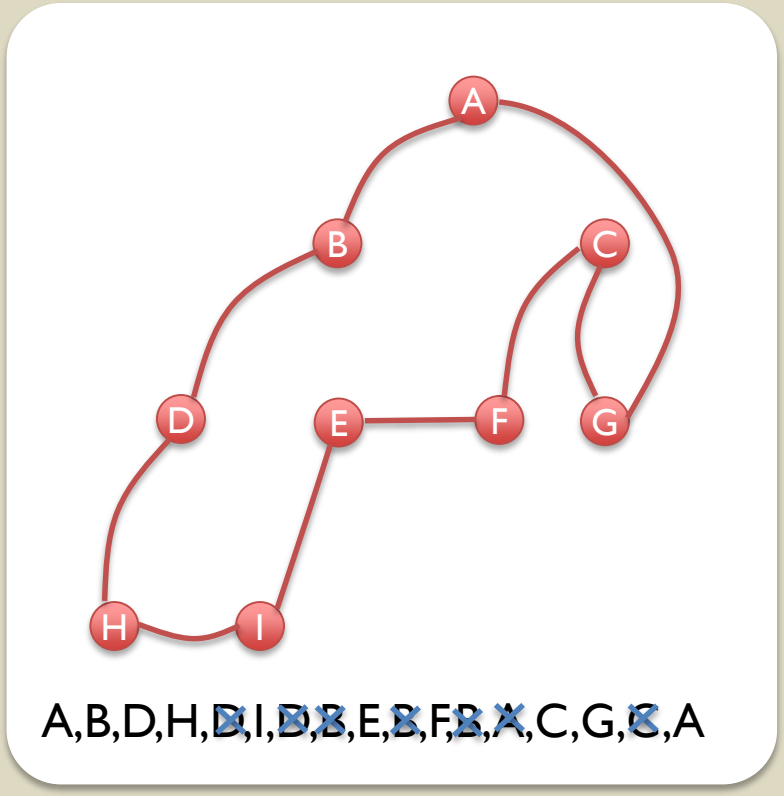
Find minimum spanning tree **T**

Duplicate **T**

Return Eulerian tour plus shortcutting

Eulerian tour: a walk that traverses each edge exactly once

Short cut to visit each vertex exactly once. By triangle inequality this doesn't increase cost



Cost of tour is at most

$$2 \cdot w(T) \leq 2 \cdot OPT$$

Hence, Double Tree Algorithm is a

2-approximation algorithm for TSP

CHRISTOFIDES ALGORITHM

The recipe

Euler'1736

A graph has an Eulerian walk iff each vertex has even degree

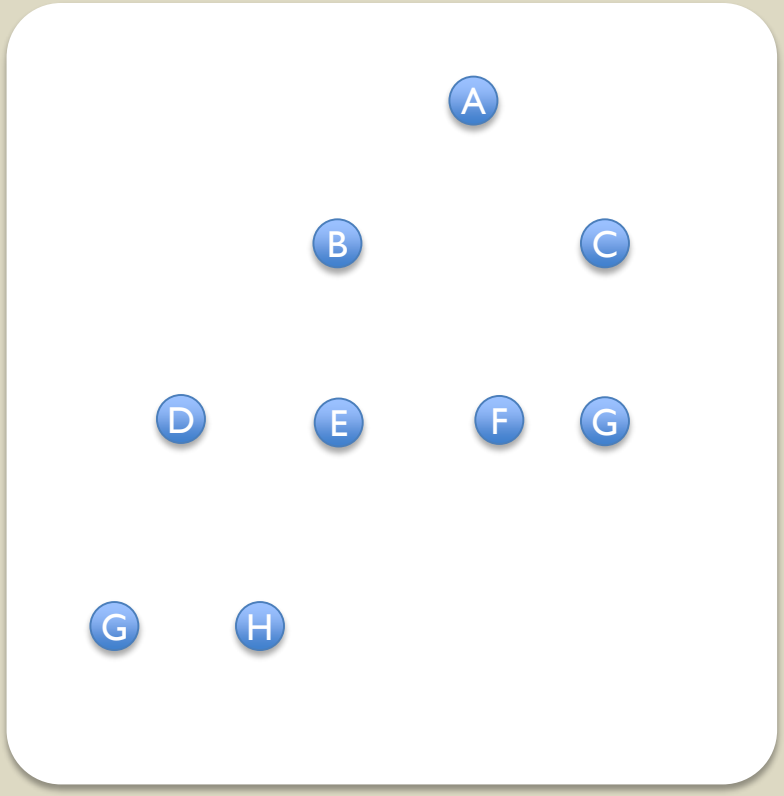
- Hence, to solve TSP it is sufficient to find a cheap connected subgraph so that each vertex has even degree
- Double spanning tree algorithm does this by simply duplicating a spanning tree
- Christofides algorithm will also ensure connectivity by taking a MST but then be more clever in the correction of the parity of vertices

Christofides Algorithm

Find minimum spanning tree **T**

Find …

Return **T + M** (with shortcutting)

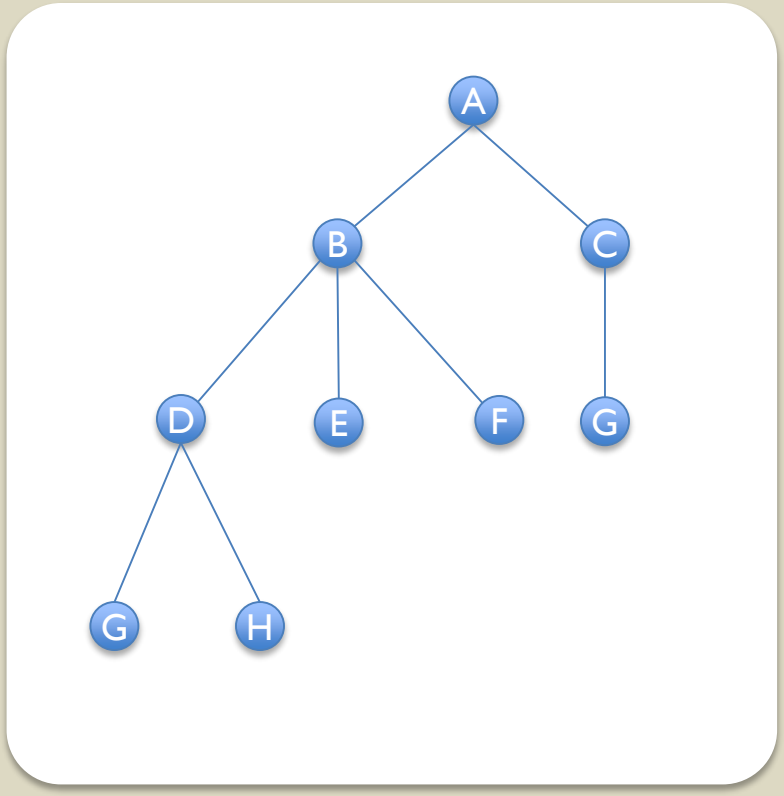


Christofides Algorithm

Find minimum spanning tree **T**

Find …

Return **T + M** (with shortcutting)

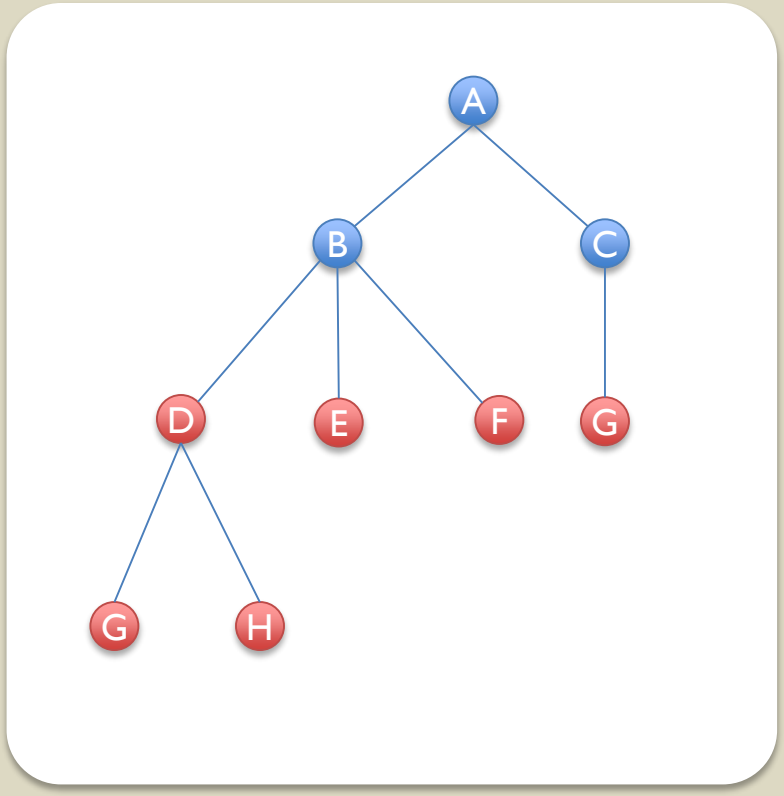


Christofides Algorithm

Find minimum spanning tree **T**

Find …

Return **T + M** (with shortcutting)



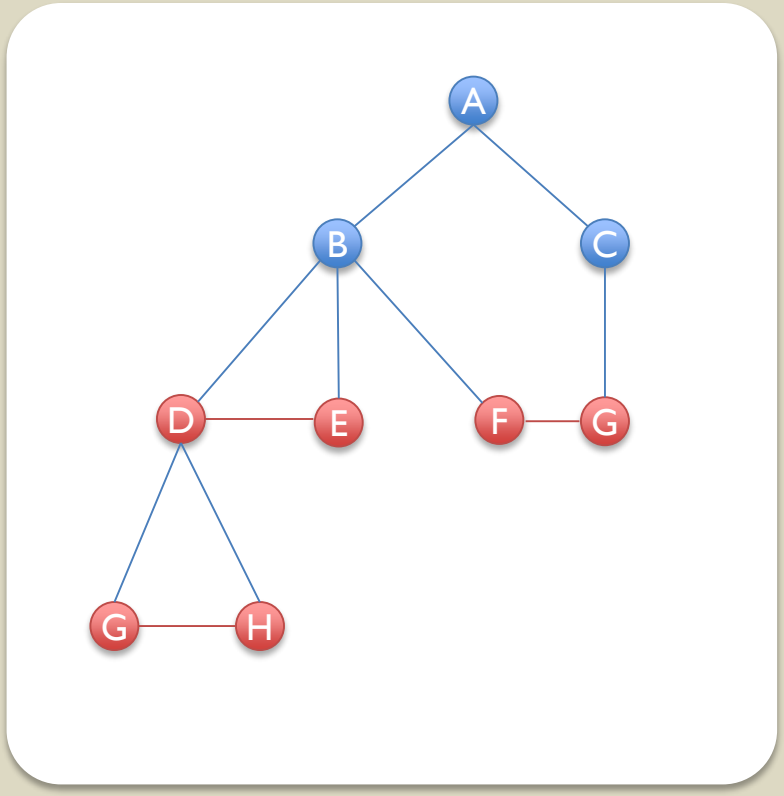
Hint: A graph has always an even number of vertices *(exercise)*

Christofides Algorithm

Find minimum spanning tree **T**

Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

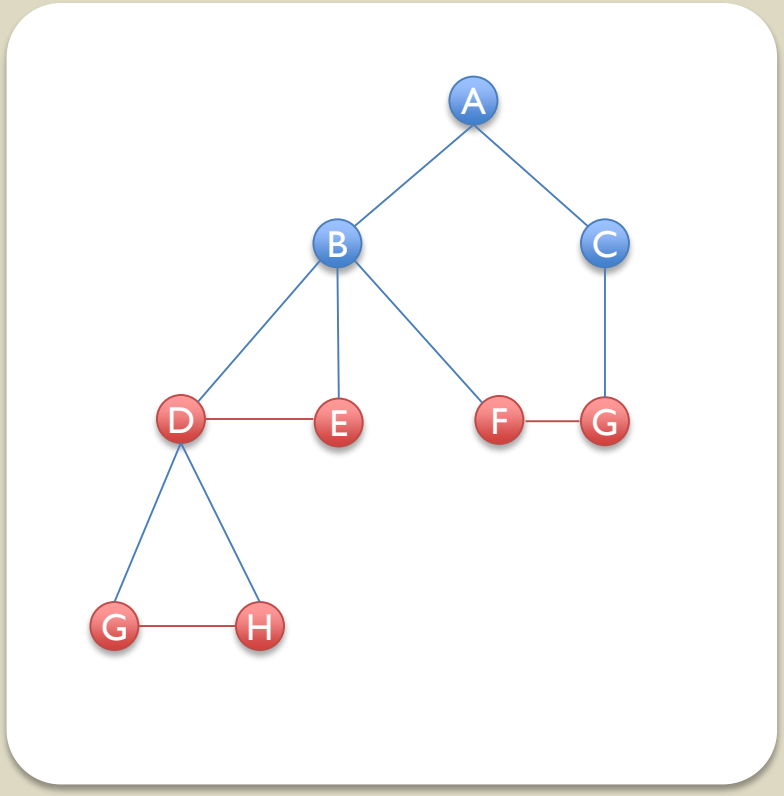


Christofides Algorithm

Find minimum spanning tree **T**

Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

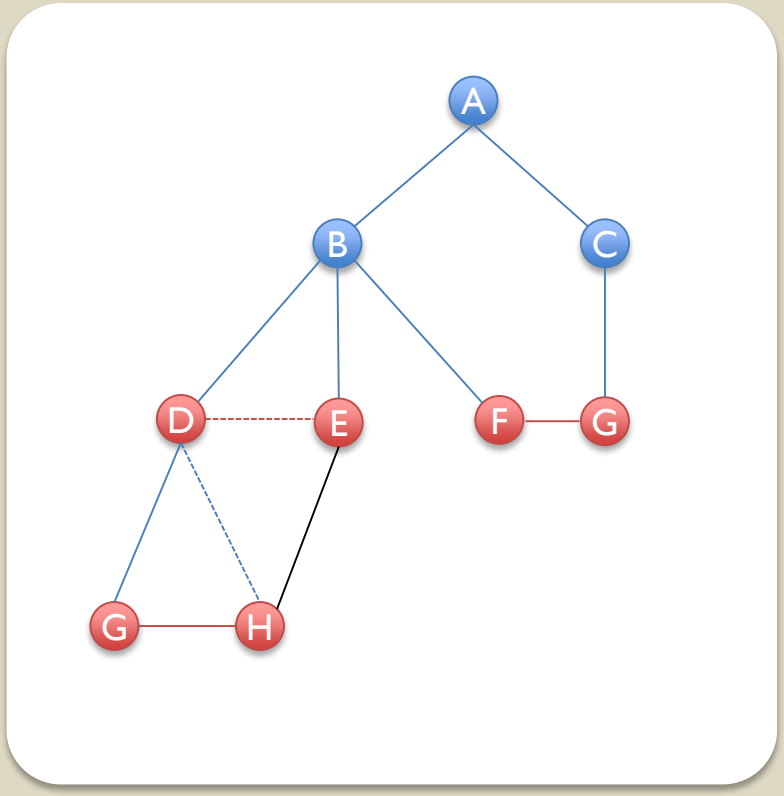


Christofides Algorithm

Find minimum spanning tree **T**

Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

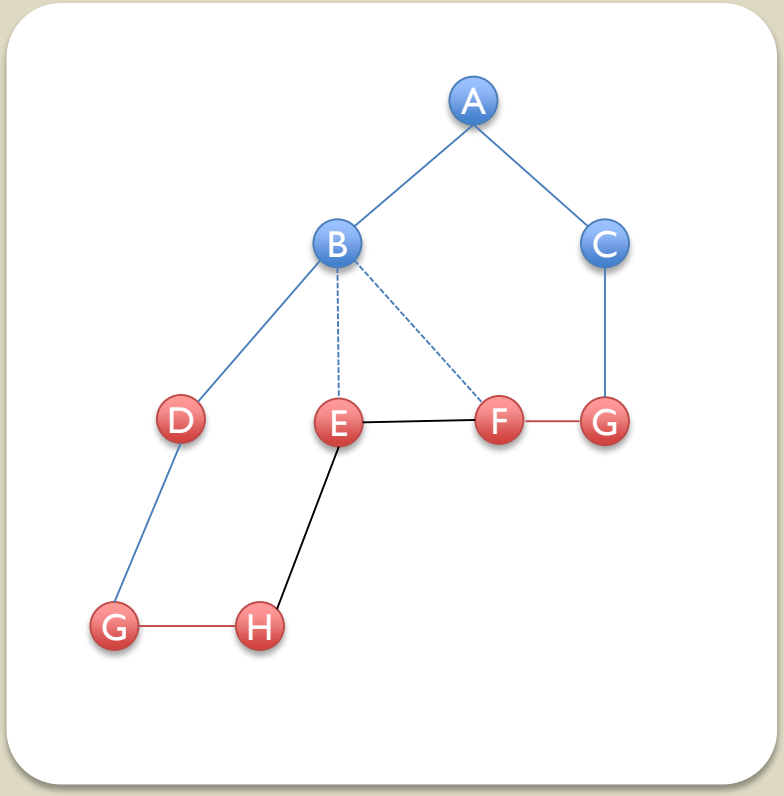


Christofides Algorithm

Find minimum spanning tree **T**

Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

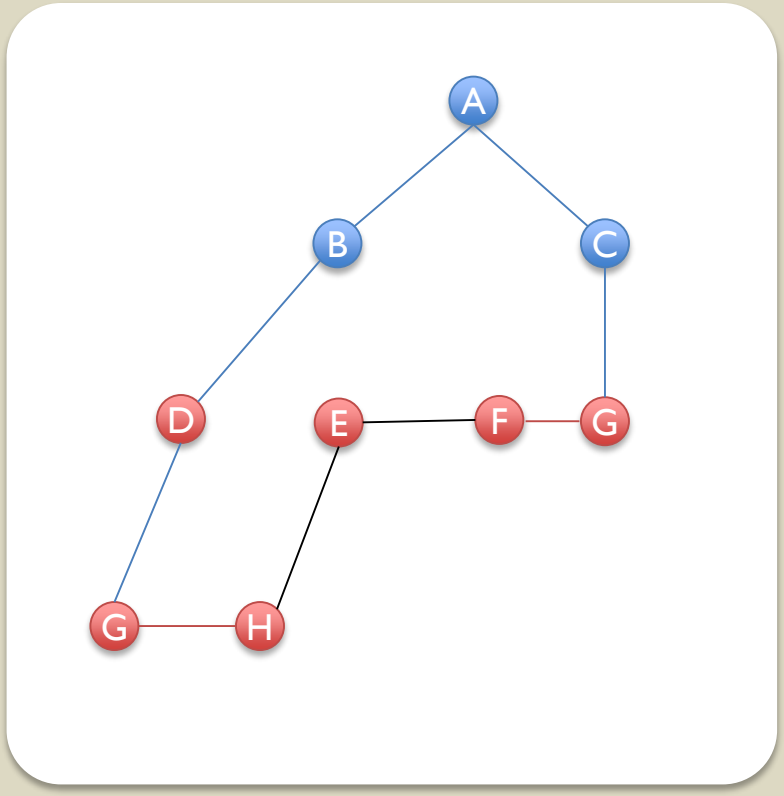


Christofides Algorithm

Find minimum spanning tree **T**

Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)



Cost of tour is at most

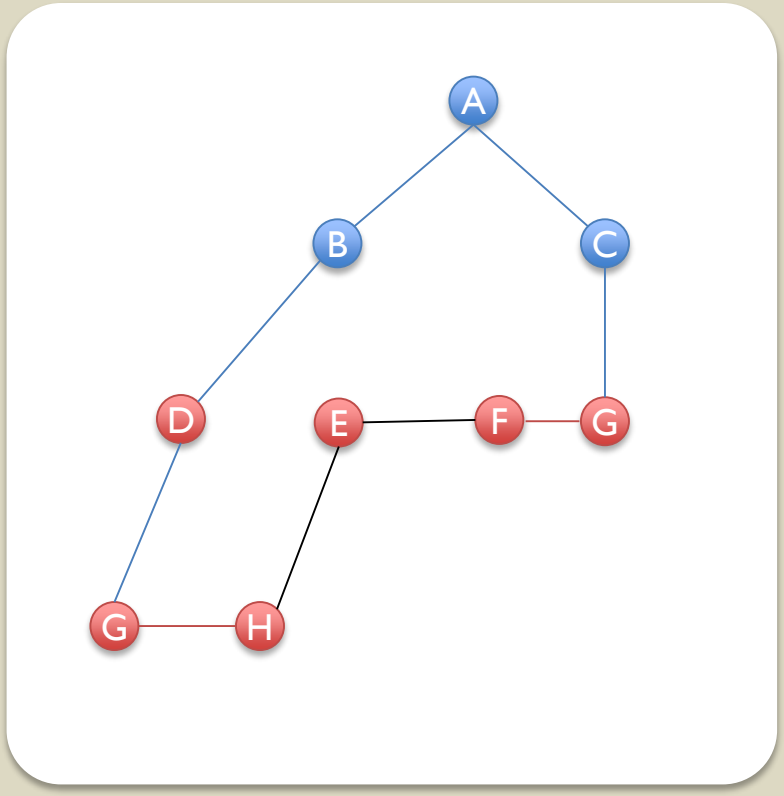
$$w(T) + w(M) \leq w(OPT) + w(M)$$

Christofides Algorithm

Find minimum spanning tree **T**

Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)



Cost of tour is at most

$$w(T) + w(M) \leq w(OPT) + w(M)$$

How can we bound $w(M)$?

Christofides Algorithm

Find minimum spanning tree **T**

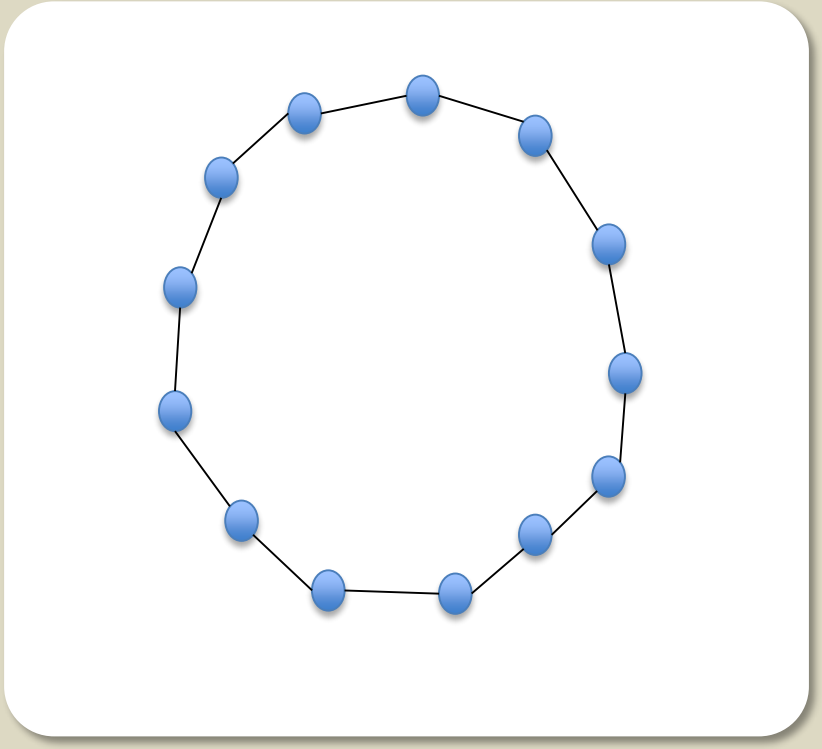
Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

We have $w(M) \leq \frac{OPT}{2}$

PROOF:

- Take an optimal tour of cost OPT



Christofides Algorithm

Find minimum spanning tree **T**

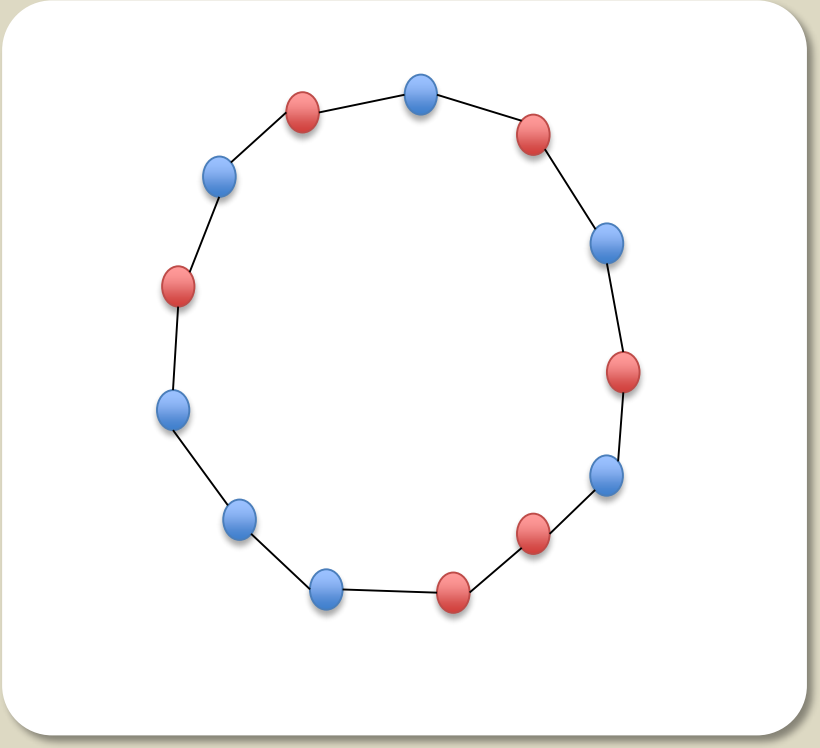
Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

We have $w(M) \leq \frac{OPT}{2}$

PROOF:

- Take an optimal tour of cost OPT
- Consider vertices that are odd in **T**



Christofides Algorithm

Find minimum spanning tree **T**

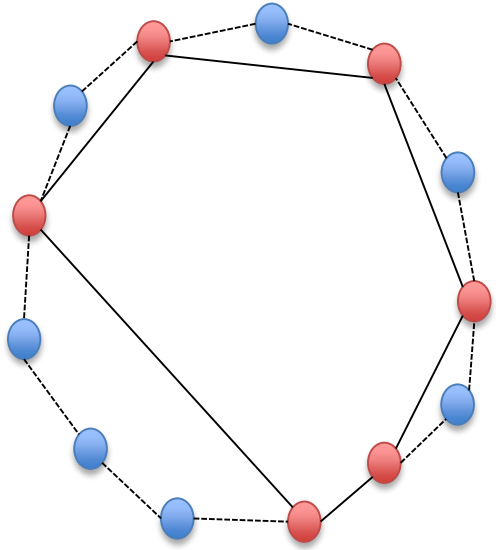
Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

We have $w(M) \leq \frac{OPT}{2}$

PROOF:

- Take an optimal tour of cost OPT
- Consider vertices that are odd in **T**
- Shortcut to obtain tour on odd-degree vertices of no larger cost



Christofides Algorithm

Find minimum spanning tree **T**

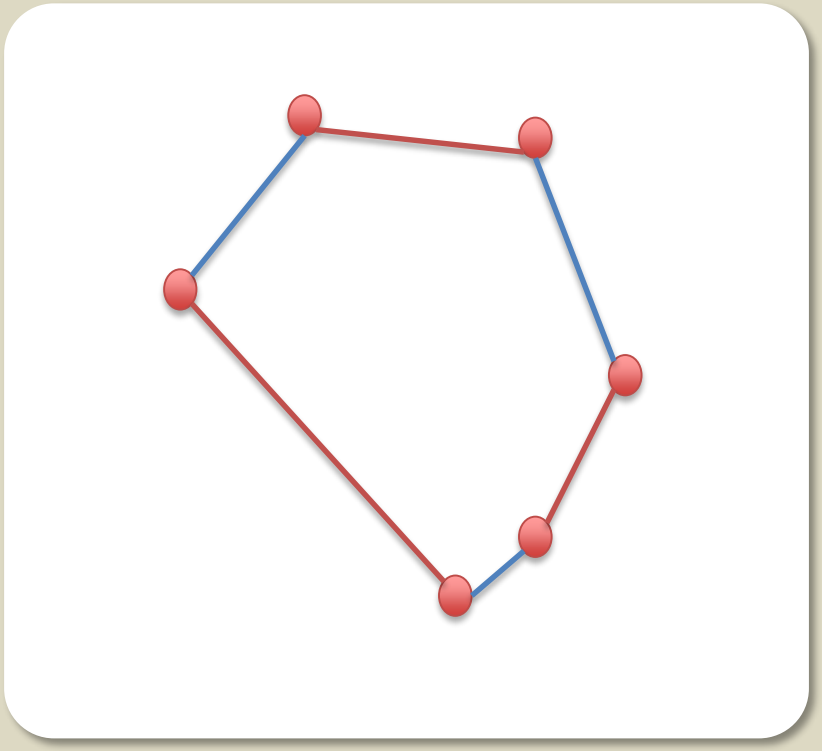
Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

We have $w(M) \leq \frac{OPT}{2}$

PROOF:

- Take an optimal tour of cost OPT
- Consider vertices that are odd in **T**
- Shortcut to obtain tour on odd-degree vertices of no larger cost
- These edges partition into two matchings M_1 and M_2 such that $w(M_1) + w(M_2) \leq OPT$



Christofides Algorithm

Find minimum spanning tree **T**

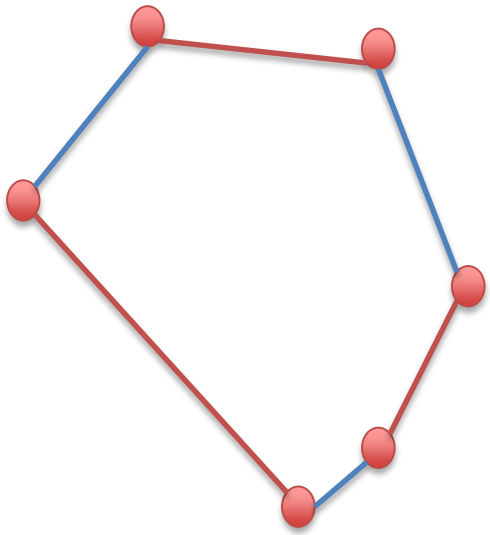
Find min matching **M** of odd degree vertices

Return **T + M** (with shortcutting)

We have $w(M) \leq \frac{OPT}{2}$

PROOF:

- Take an optimal tour of cost OPT
- Consider vertices that are odd in **T**
- Shortcut to obtain tour on odd-degree vertices of no larger cost
- These edges partition into two matchings M_1 and M_2 such that $w(M_1) + w(M_2) \leq OPT$
- Hence the minimum has weight $\leq \frac{OPT}{2}$



Summary sofar

- Saw our first approximation algorithm
- We were a little bit more clever to obtain Christofides algorithm
- This is the best known in spite of a lot of research!

Today

Major open problem to understand the approximability of TSP

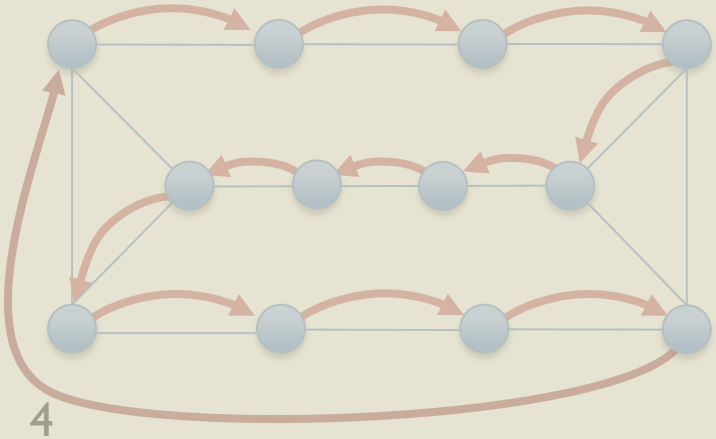
- **NP-hard** to approximate metric TSP within 220/219
- Christofides' **1.5-approximation algorithm** still best
- Held-Karp relaxation conjectured to give **4/3-approximation**

Graph-TSP

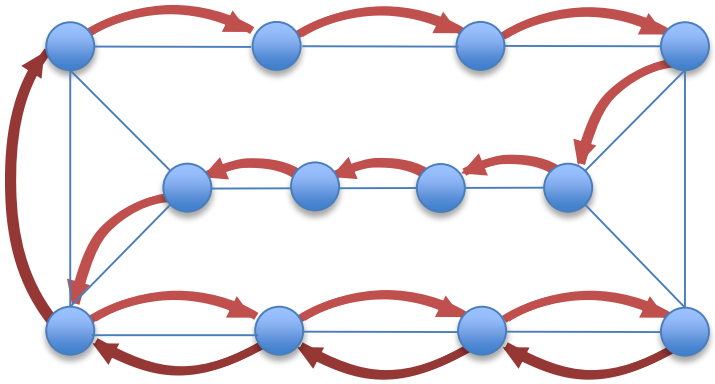
Given an unweighted undirected graph **G=(V,E)**

Find shortest tour where
Find spanning Eulerian multigraph with minimum #edges

d(u,v) = shortest path between **u** and **v**



Cost = #edges



2000 2005 2010

Progress on TSP vs mobiles

Oveis Gharan, Saberi & Singh:

$(1.5 - \epsilon)$-approximation algorithm for graph-TSP

Gamarnik, Lewenstein & Sviridenko'05:

1.487 approximation for cubic 2-edge connected graphs

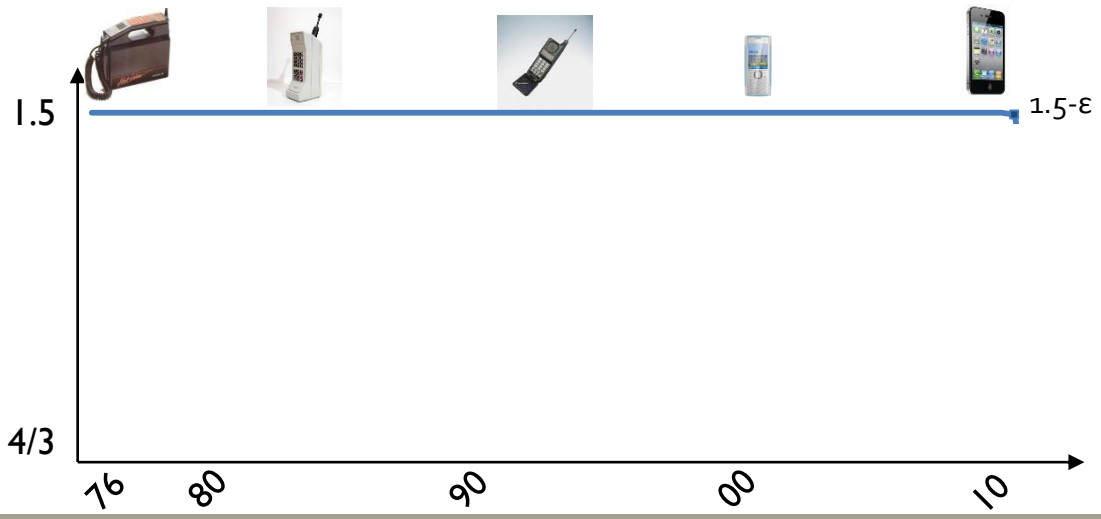
Boyd, Sitters, van der Star & Stougie

4/3 – approximation for cubic graphs

7/5 – approximation for subcubic graphs

Mömke & S.:

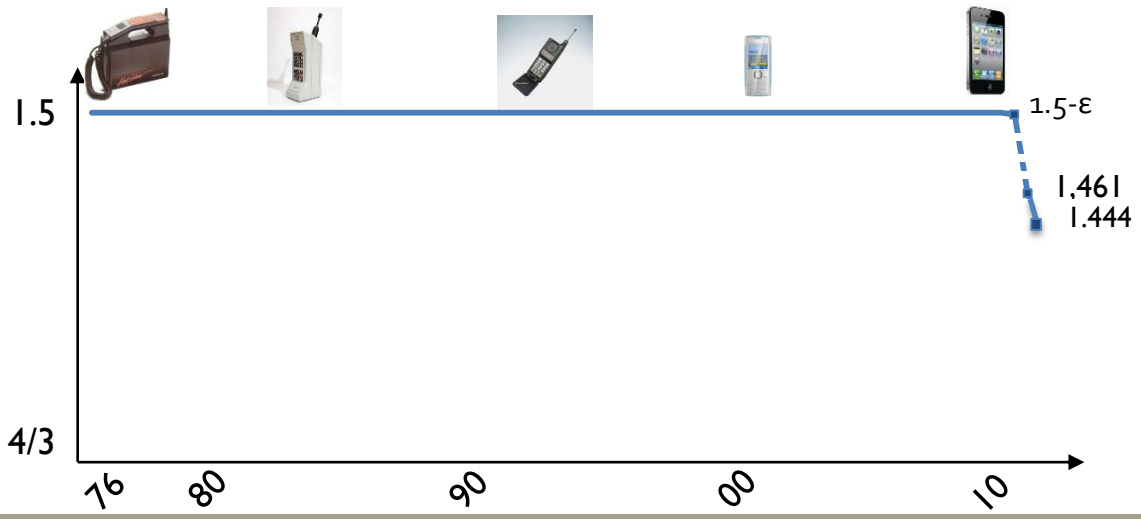
1.461-approximation algorithm for graph-TSP



2000 2005 2010

Mucha:

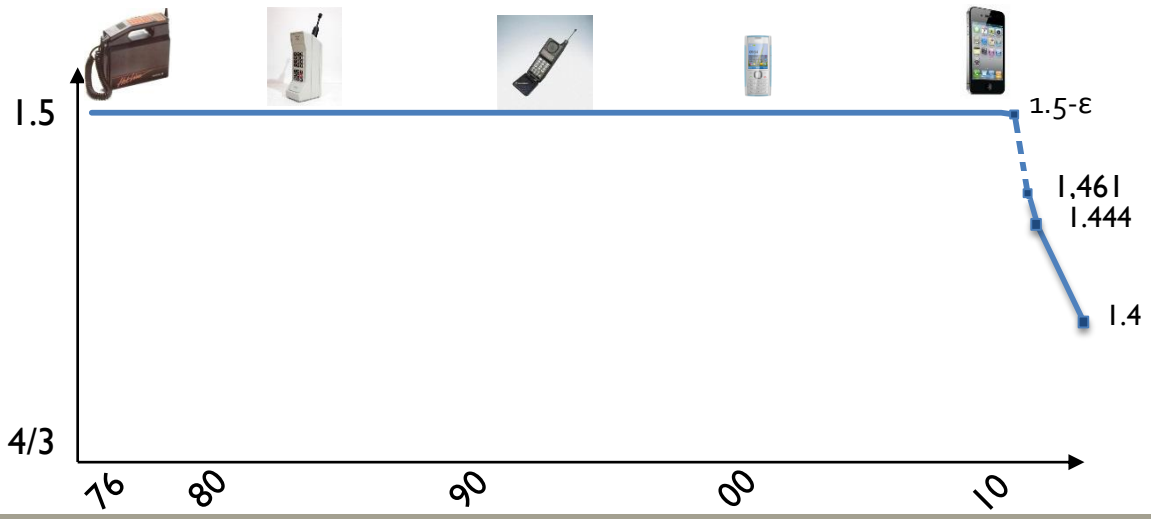
1.444-approximation algorithm for graph-TSP



2000 2005 2010

Sebö & Vygen:

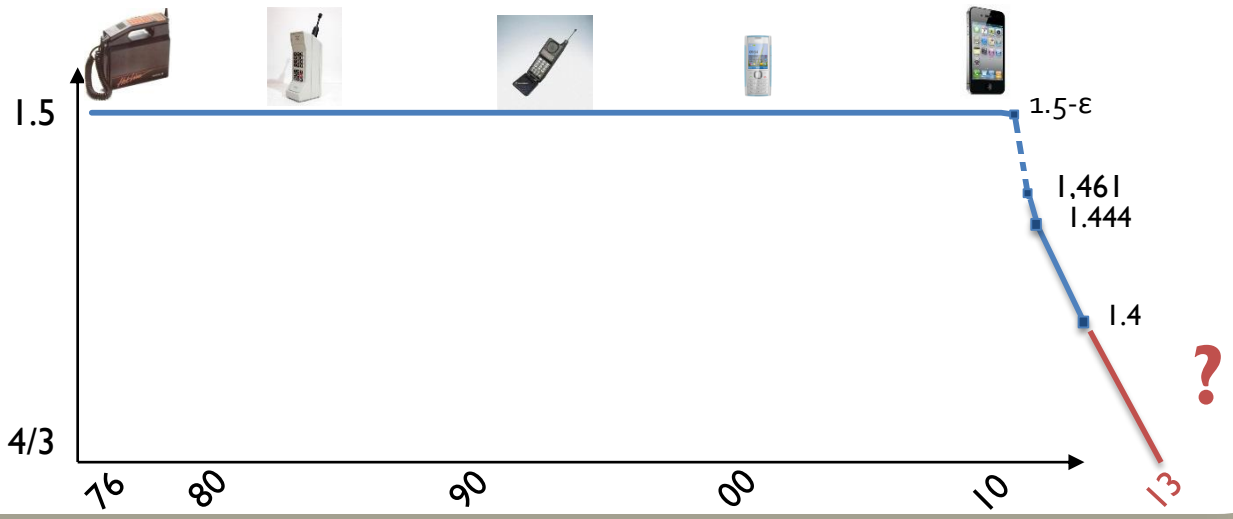
1.4-approximation algorithm for graph-TSP

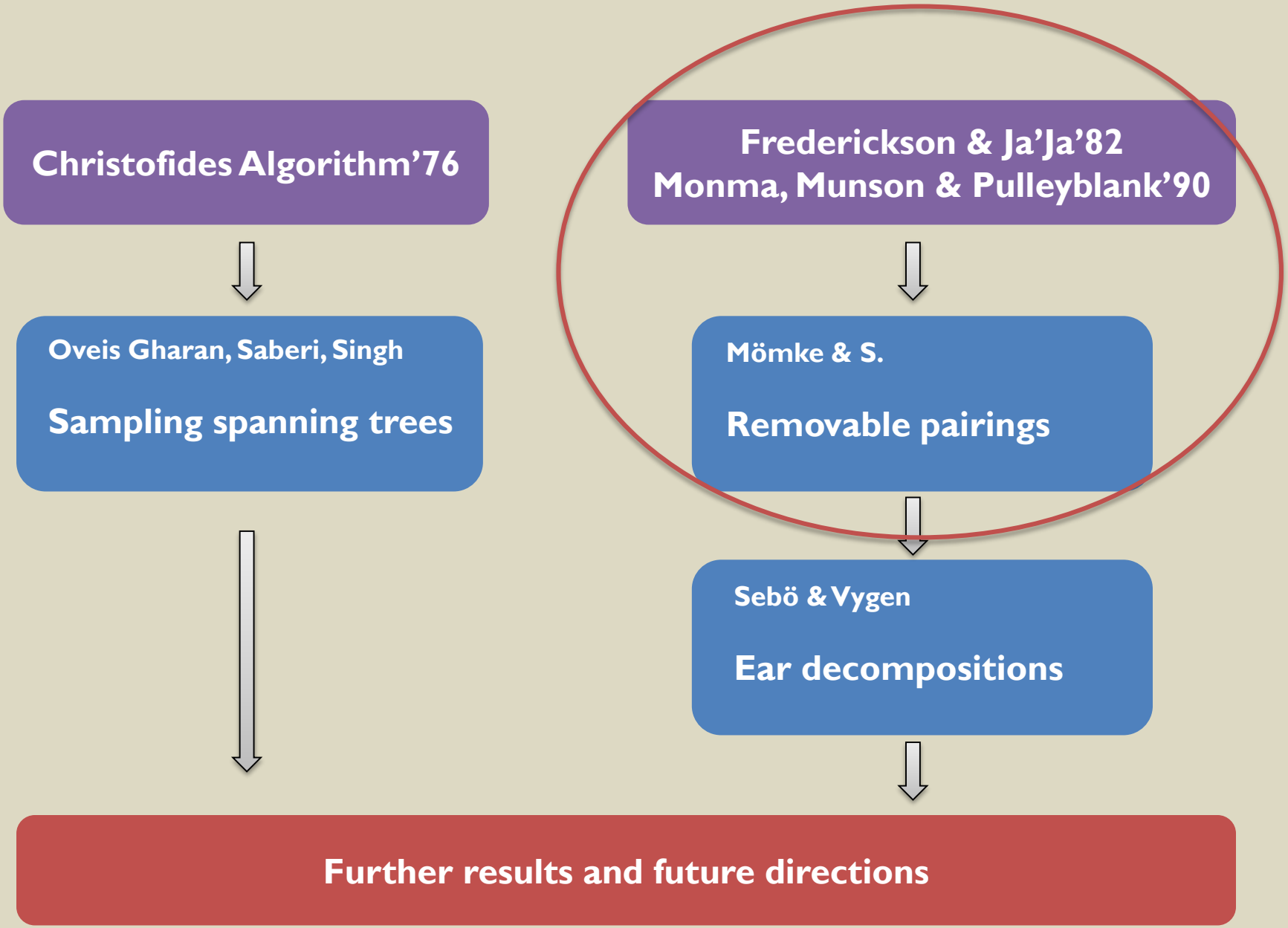


2000 2005 2010

Sebö & Vygen:

1.4-approximation algorithm for graph-TSP




Christofides Algorithm'76
Frederickson & Ja'Ja'82
Monma, Munson & Pulleyblank'90
Oveis Gharan, Saberi, Singh
Sampling spanning trees
Mömke & S.
Removable pairings
Sebö & Vygen
Ear decompositions
Further results and future directions

Approximating TSP by removable pairings

- Different more "graph theoretic approach"
- Promising applications (apart from 1.4 approximation for graph-TSP)
- Among other things settled a conjecture by Boyd et al.

Subcubic 2-edge connected graphs have a tour of length 4n/3-2/3

We will illustrate the techniques by proving above statement for cubic graphs

Relating 2-VC and Tours

Frederickson & Ja'Ja'82 and Monma, Munson & Pulleyblank'90

A 2-VC (cubic) graph **G=(V,E)** has a tour of length at most **4/3|E|**

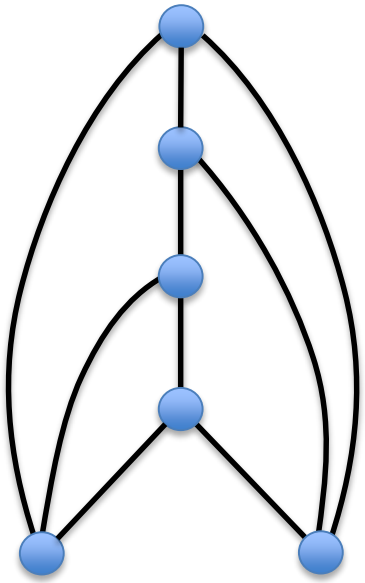
Relating 2-VC and Tours

Sample perfect matching **M** so that

each edge is take with prob. **1/3**

Return **E + M**

Input

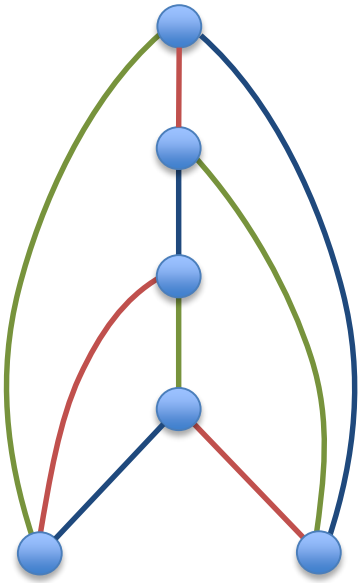


Relating 2-VC and Tours

Sample perfect matching **M** so that each edge is take with prob. **1/3**

Return **E + M**

Input



Berge-Fulkerson Conjecture:

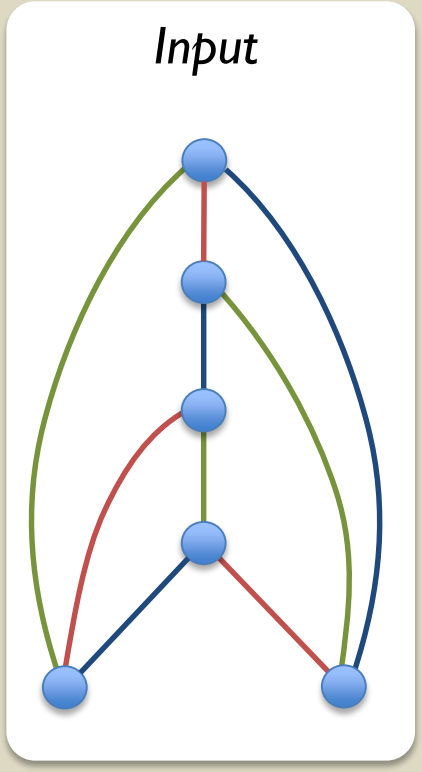
Any cubic 2-edge connected graph has 6 matchings so that each edge appears in exactly two of them?

Relating 2-VC and Tours

Sample perfect matching **M** so that each edge is take with prob. **1/3**

Return **E + M**

How to sample M in general?



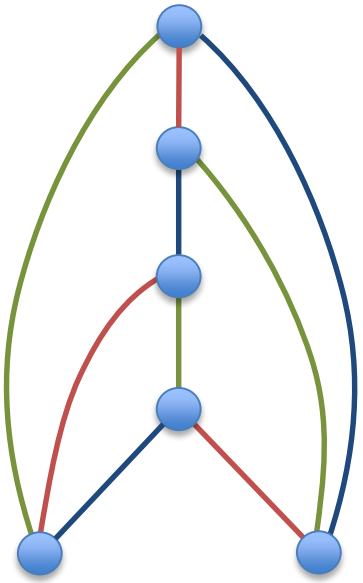
Relating 2-VC and Tours

Sample perfect matching **M** so that each edge is take with prob. **1/3**

Return **E + M**

How to sample M in general?

Input

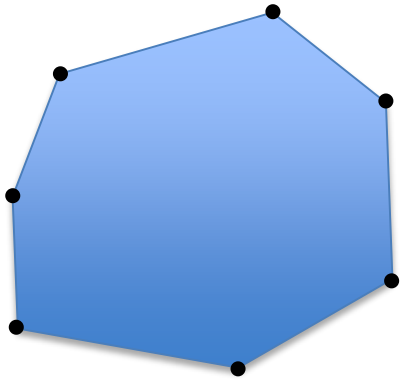


Edmond's perfect matching polytope

$$\sum_{e \in \delta(v)} x_e = 1 \qquad \forall v \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subseteq V, |S| \text{ odd}$$

$$x \geq 0$$



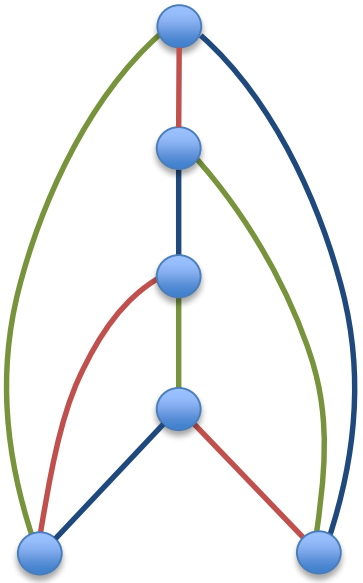
Relating 2-VC and Tours

Sample perfect matching **M** so that each edge is take with prob. **1/3**

Return **E + M**

How to sample M in general?

Input

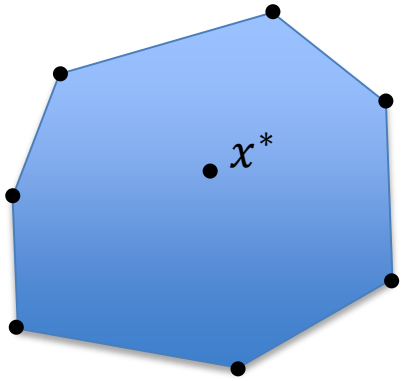


Edmond's perfect matching polytope

$$\sum_{e \in \delta(v)} x_e = 1 \qquad \forall v \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subseteq V, |S| \text{ odd}$$

$$x \geq 0$$



Every extreme point ⇔ perfect matching

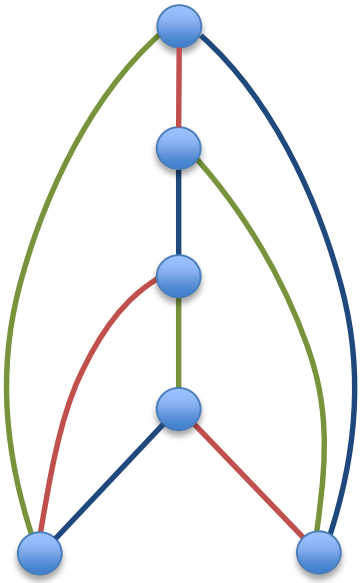
Relating 2-VC and Tours

Sample perfect matching **M** so that each edge is take with prob. **1/3**

Return **E + M**

How to sample M in general?

Input

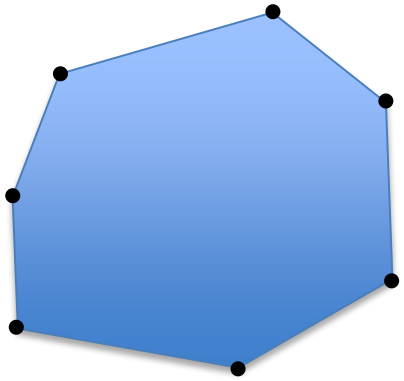


Edmond's perfect matching polytope

$$\sum_{e \in \delta(v)} x_e = 1 \qquad \forall v \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subseteq V, |S| \text{ odd}$$

$$x \geq 0$$



Every extreme point ⇔ perfect matching

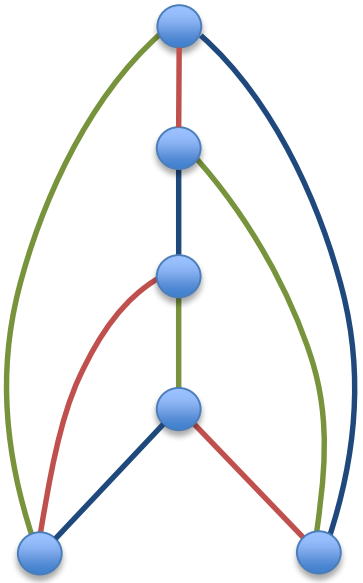
Relating 2-VC and Tours

Sample perfect matching **M** so that each edge is take with prob. **1/3**

Return **E + M**

How to sample M in general?

Input

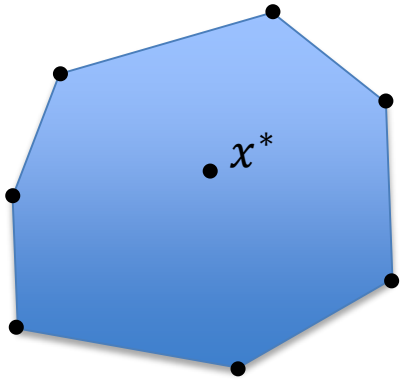


Edmond's perfect matching polytope

$$\sum_{e \in \delta(v)} x_e = 1 \qquad \forall v \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subseteq V, |S| \text{ odd}$$

$$x \geq 0$$



Every extreme point ⇔ perfect matching

- For 2-connected cubic graphs $x_e^* = 1/3$ for is a feasible solution *(exercise)*

Relating 2-VC and Tours

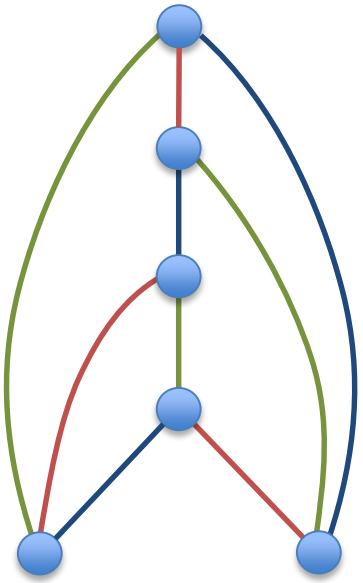
Sample perfect matching **M** so that

each edge is take with prob. **1/3**

Return **E + M**

How to sample M in general?

Input

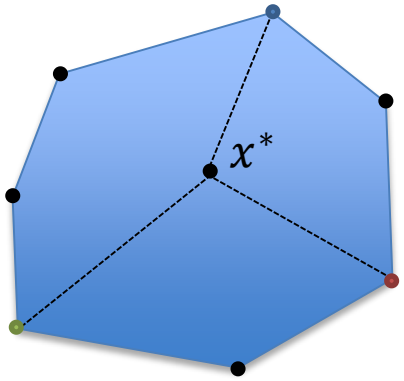


Edmond's perfect matching polytope

$$\sum_{e \in \delta(v)} x_e = 1 \qquad \forall v \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subseteq V, |S| \text{ odd}$$

$$x \geq 0$$



Every extreme point ⇔ perfect matching

- For 2-connected cubic graphs $x_e^* = 1/3$ for is a feasible solution *(exercise)*
- By Edmond, $x^* = \sum \lambda_i M^{(i)}$ can be written as a convex combination of perfect matchings

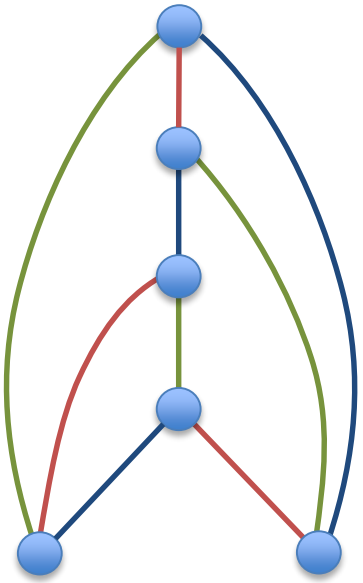
Relating 2-VC and Tours

Sample perfect matching **M** so that each edge is take with prob. **1/3**

Return **E + M**

How to sample M in general?

Input

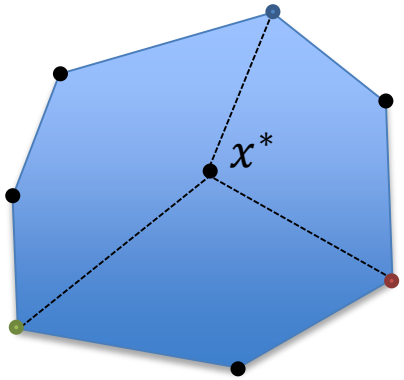


Edmond's perfect matching polytope

$$\sum_{e \in \delta(v)} x_e = 1 \qquad \forall v \in V$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \qquad \forall S \subseteq V, |S| \text{ odd}$$

$$x \geq 0$$



Every extreme point ⇔ perfect matching

- For 2-connected cubic graphs $x_e^* = 1/3$ for is a feasible solution *(exercise)*
- By Edmond, $x^* = \sum \lambda_i M^{(i)}$ can be written as a convex combination of perfect matchings
- Sampling $M^{(i)}$ with probability λ_i gives a distribution of perfect matchings where each edge is taken with probability 1/3

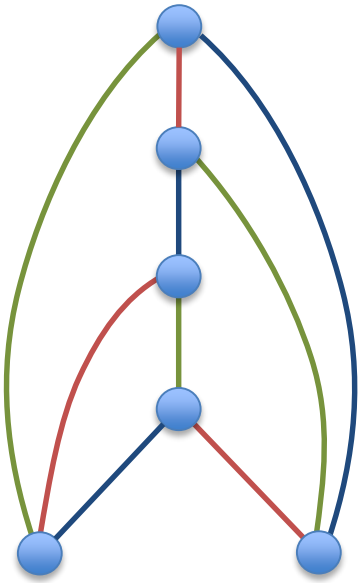
Relating 2-VC and Tours

Sample perfect matching **M** so that

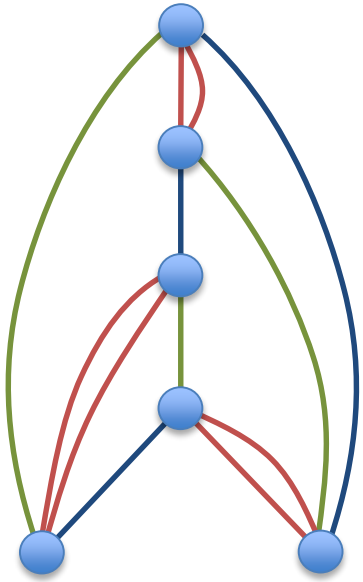
each edge is take with prob. **1/3**

Return **E + M**

Input

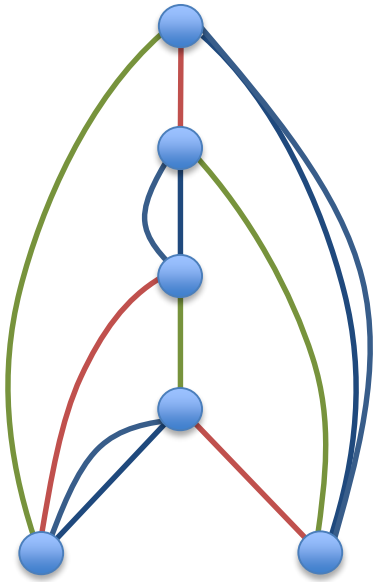


Output: E+M



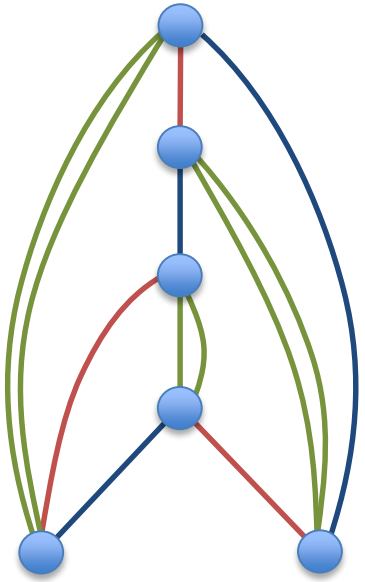
2n = 4/3|E| edges

Output: E+M



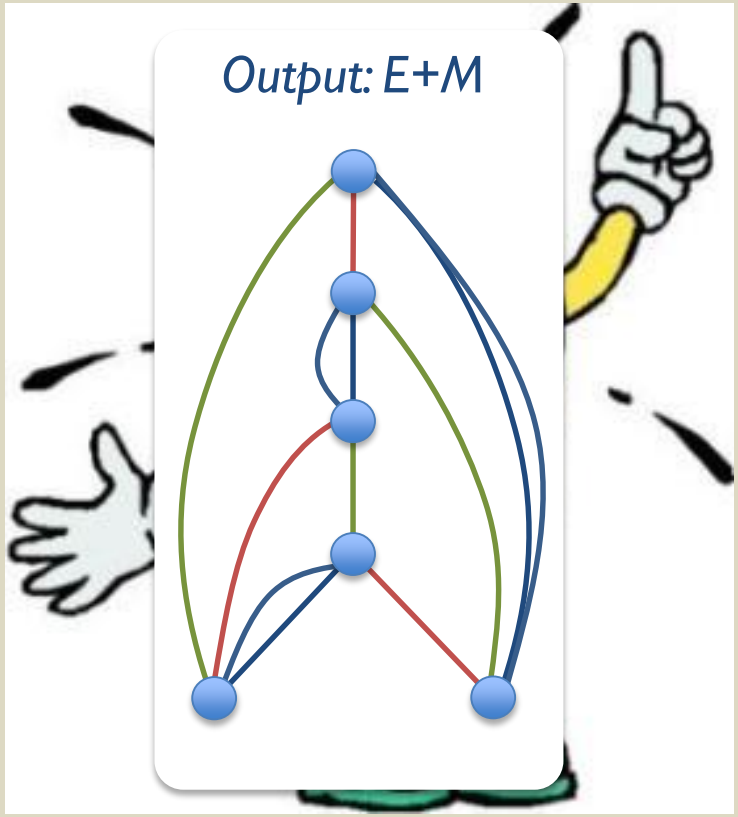
2n = 4/3|E| edges

Output: E+M

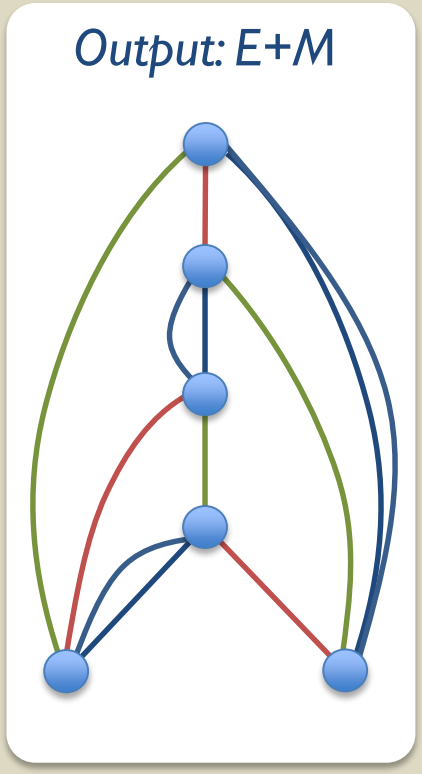


2n = 4/3|E| edges

=> A 2-VC (cubic) graph **G=(V,E)** has a tour of length at most **4/3|E|**



Instead of just adding edges from matching remove some of them



Need to keep connectivity!

Instead of just adding edges from matching remove some of them

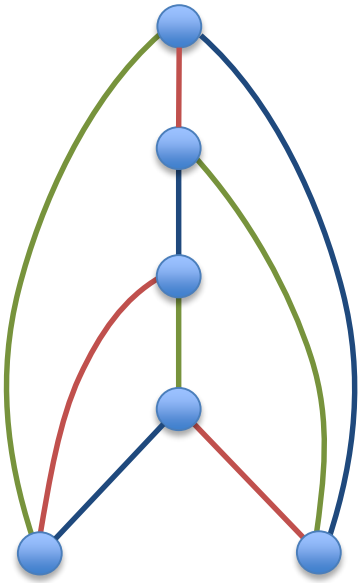
Removing Edges

Take a DFS spanning tree **T**

Same algorithm as before but

return $E + (M \cap T) - (M \setminus T)$

Input



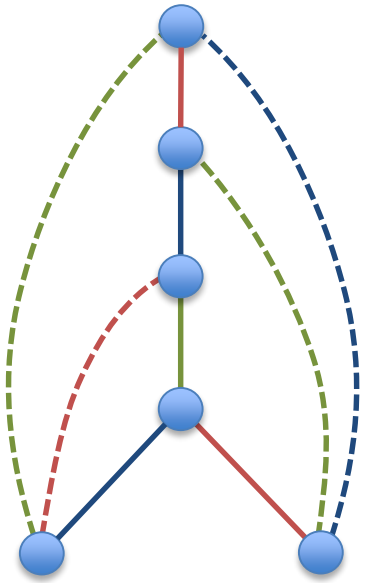
Removing Edges

Take a DFS spanning tree **T**

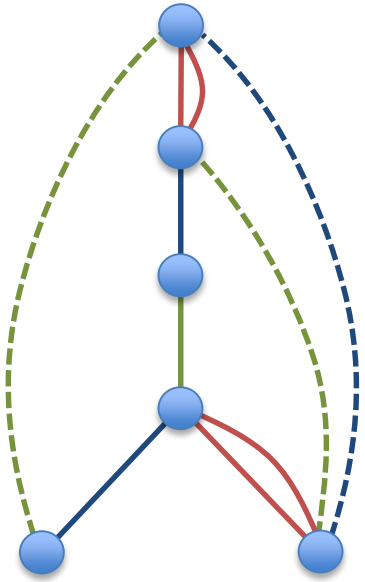
Same algorithm as before but

return $E + (M \cap T) - (M \setminus T)$

Input

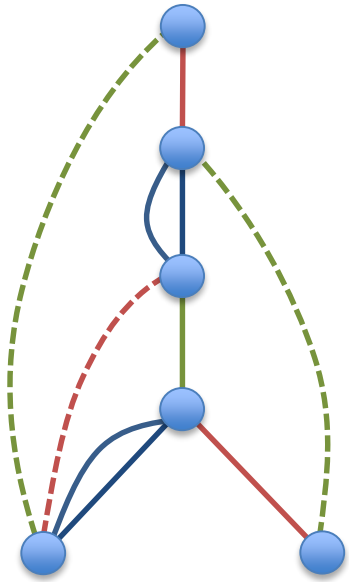


Output



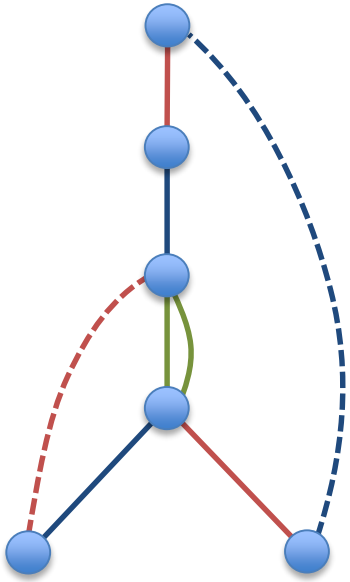
5/3n = 10/9|E| edges

Output



5/3n = 10/9|E| edges

Output

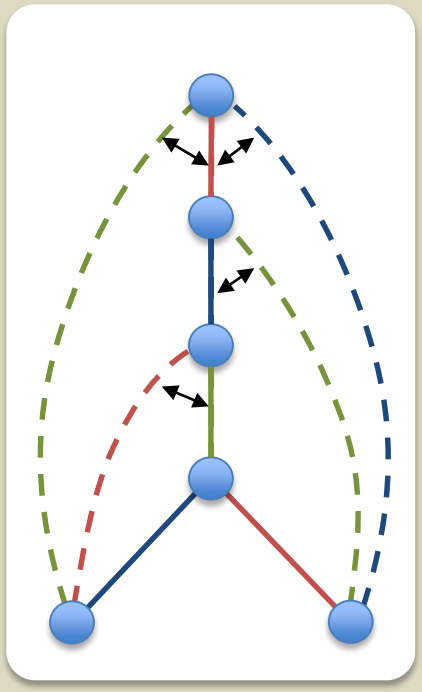


4/3n = 8/10|E| edges

=> A 2-VC (cubic) graph **G=(V,E)** has a tour of length at most **2/3(n-1) + 2/3|E|**

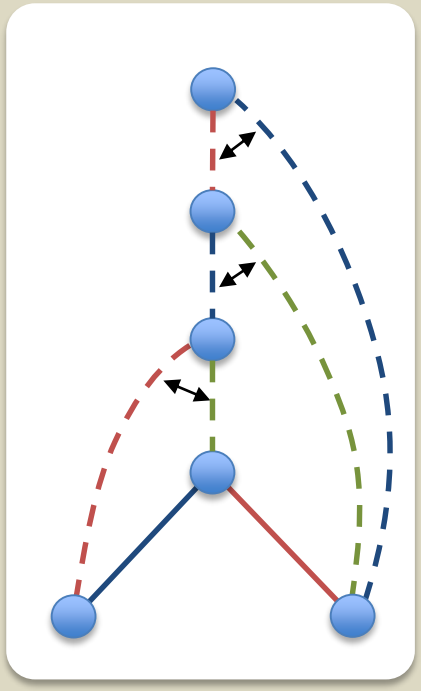
Increasing number of removable edges

- Use structure of perfect matching to increase the set of removable edges
- Define a "removable pairing"
  - Pair of edges: only one edge in each pair can occur in a matching
  - Graph obtained by removing at most one edge in each pair is connected



R contains all back edges and paired tree edges

$2b - 1$ removable edges where b is number of back edges

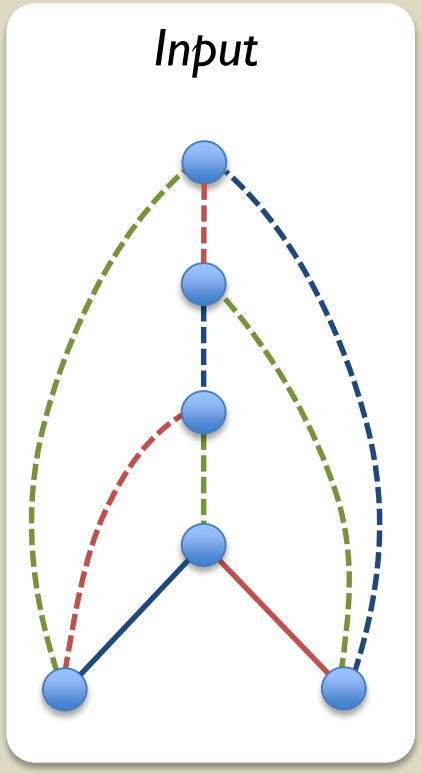


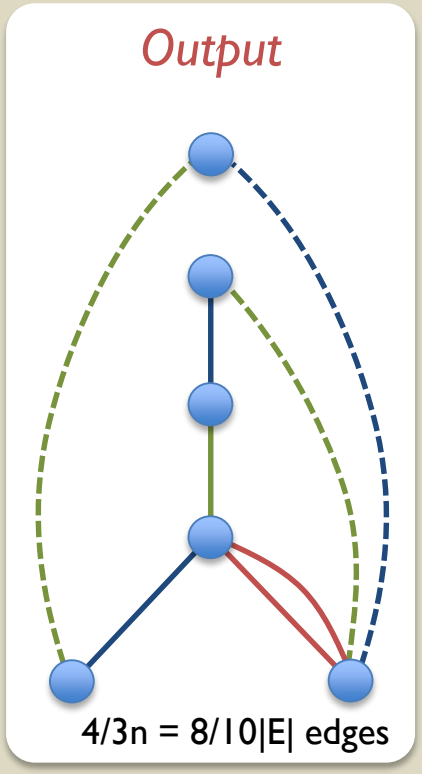
Removable Pairings

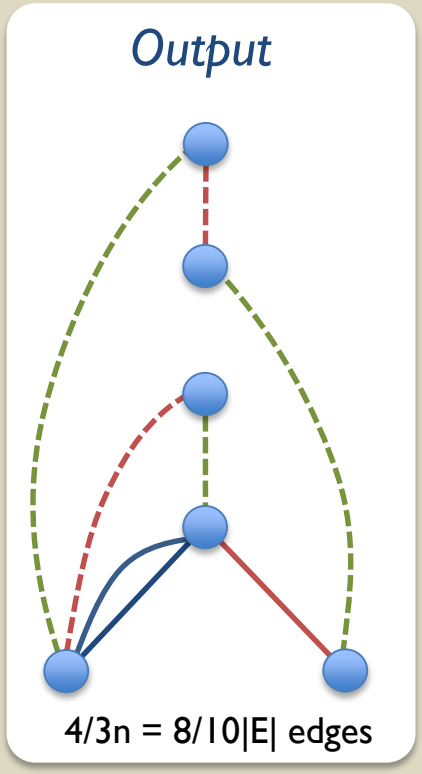
Take a DFS spanning tree **T**

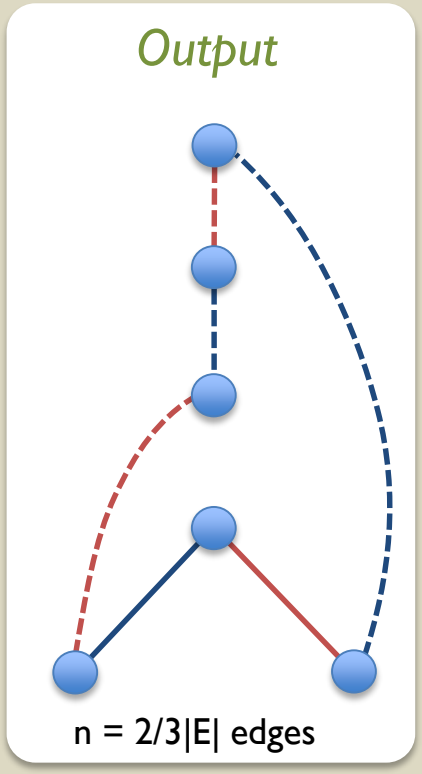
Same algorithm as before but

return $E + (M \setminus R) - (M \cap R)$

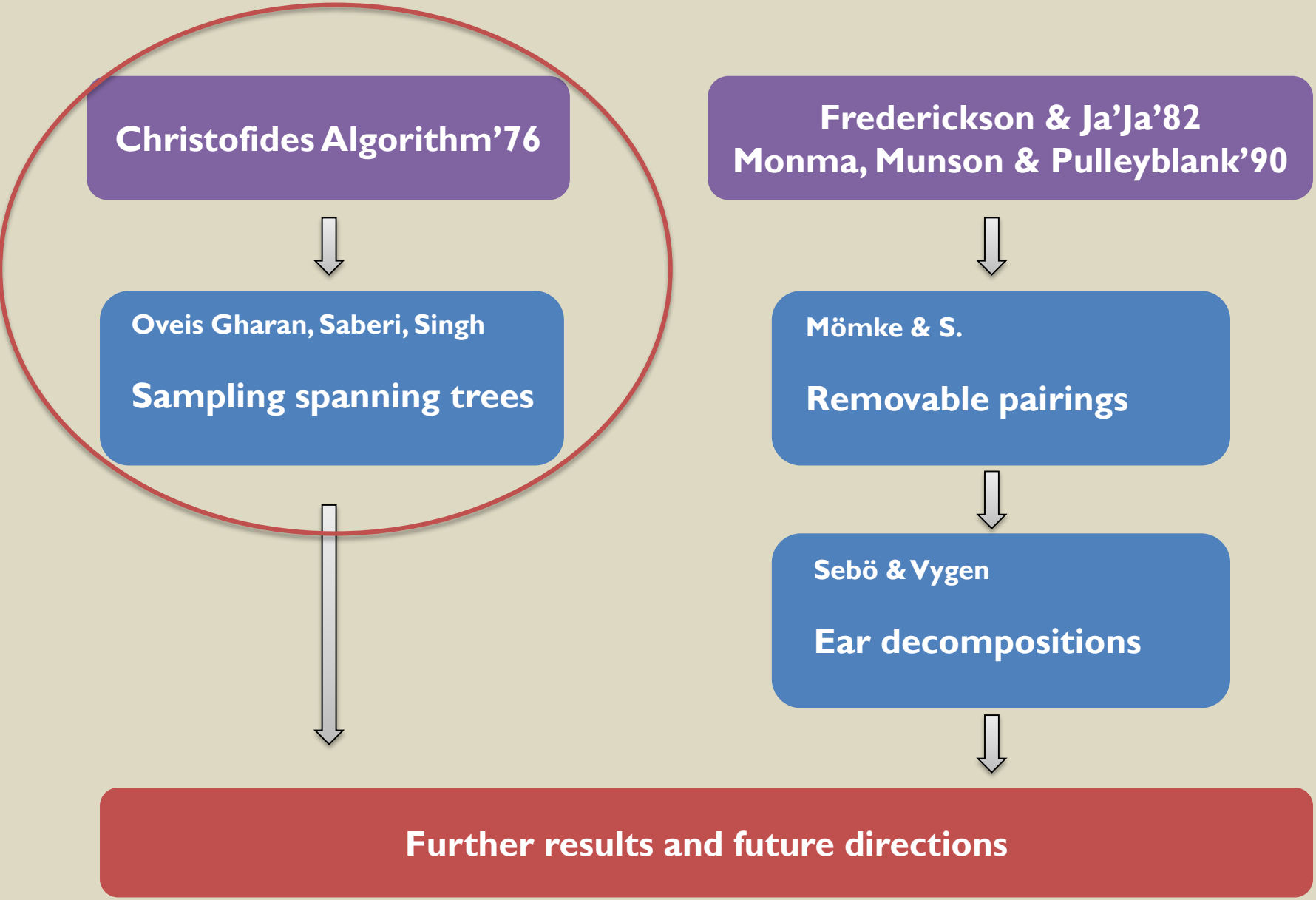








=> A 2-VC subcubic graph **G=(V,E)** has a tour of length at most **4/3n-2/3**


Christofides Algorithm'76
Oveis Gharan, Saberi, Singh
Sampling spanning trees
Frederickson & Ja'Ja'82
Monma, Munson & Pulleyblank'90
Mömke & S.
Removable pairings
Sebö & Vygen
Ear decompositions
Further results and future directions

Instead of minimum spanning tree
sample one from the Held-Karp relaxation

Inspired by work on asymmetric traveling salesman problem (Asadpour et al'10)

Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**

Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**

Held-Karp Relaxation

$$\text{minimize} \sum x_e d(e)$$

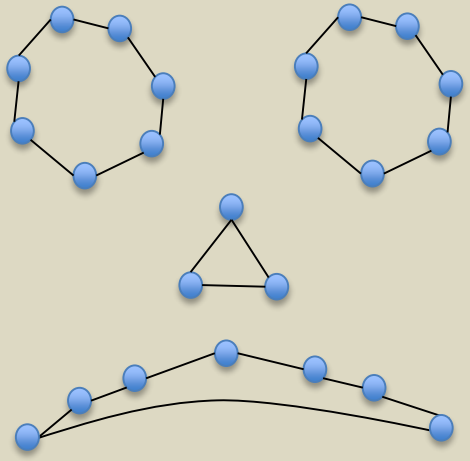
$$\sum_{e \in E} x_e = |V|$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subset V$$

$$x \geq 0$$

A tour (after shortcutting) has |V| edges

No subtours



Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**

Held-Karp Relaxation

$$\text{minimize} \sum x_e d(e)$$

$$\sum_{e \in E} x_e = |V|$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subset V$$

$$x \geq 0$$

Spanning-Tree Polytope

$$\sum_{e \in E} x_e = |V| - 1$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad \forall S \subset V$$

$$x \geq 0$$

$$\left(1 - \frac{1}{|V|}\right) x$$

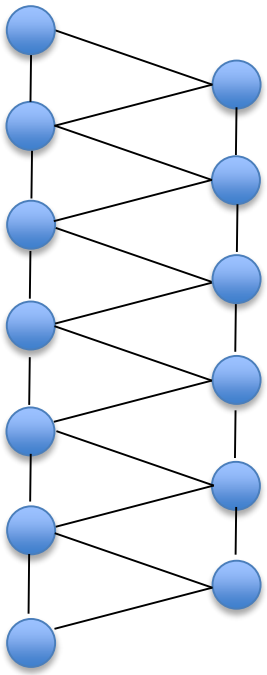
=> Sample spanning tree whose expected cost equals value of Held-Karp

Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**

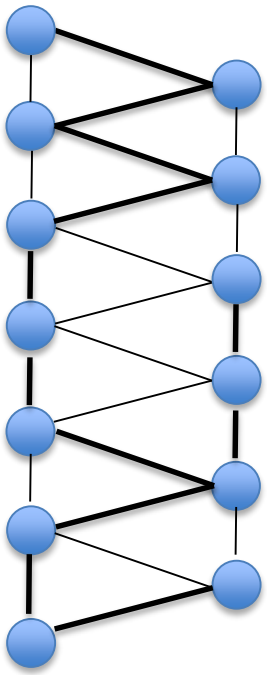


Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**



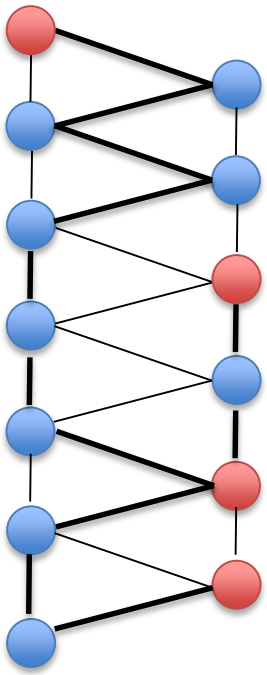
Cost of spanning tree $|V| - 1$

Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**



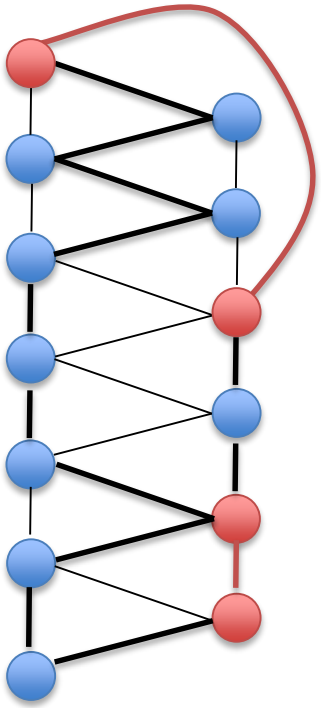
Cost of spanning tree $|V| - 1$

Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**



Cost of spanning tree $|V| - 1$

Cost of matching is at most $\frac{|V|}{2 + \epsilon}$

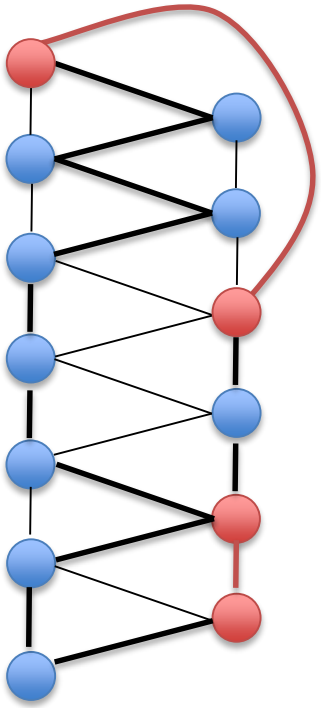
Sample Spanning Tree

Solve Held-Karp relaxation

Sample spanning tree **T** from solution

Run Christofides starting from **T**

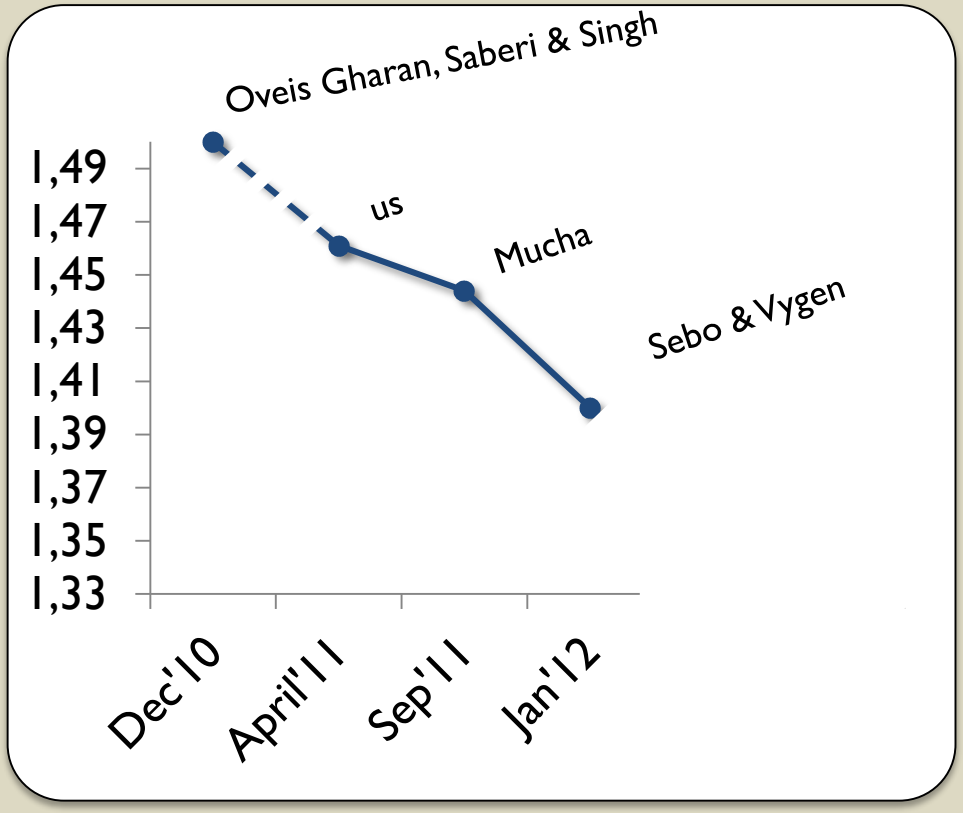
- Involved proof by Oveis Gharan et al
  - maximum entropy distribution of spanning trees
  - Structure of near-min cuts etc.



Cost of spanning tree $|V| - 1$

Cost of matching is at most $\frac{|V|}{2+\epsilon}$

Summary sofar



- Two very different approaches for improved algorithms for graph-TSP
- Many different concepts from graph theory
  - Structure of perfect matchings, Ear decompositions … hopefully more

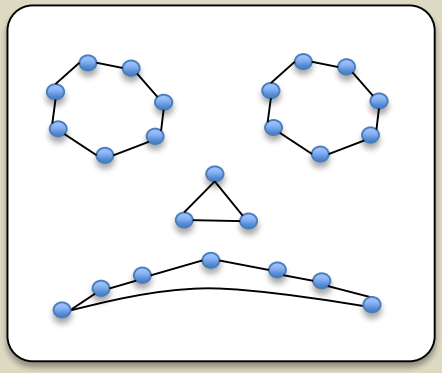
Future results and future directions

GENERAL METRICS

No progress on TSP yet but …

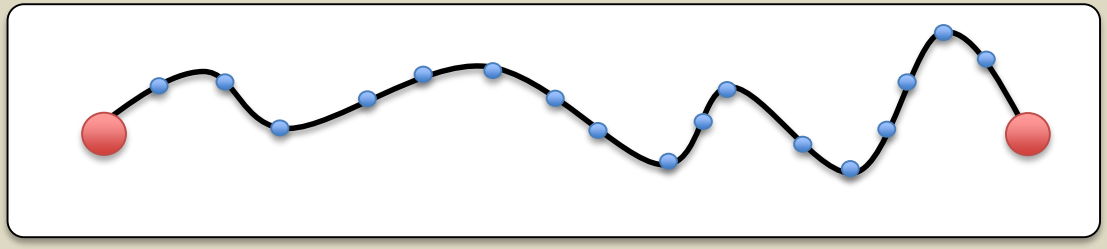
A proof of the Boyd-Carr conjecture (Schalekamp, Williamson, van Zuylen'12)

- Tight analysis of cost of 2-matching vs Held-Karp relaxation



Improved algorithms for st-path TSP (An, Kleinberg, Shmoys'12, Sebö'13)

- Tight analysis of cost of 2-matching vs Held-Karp relaxation

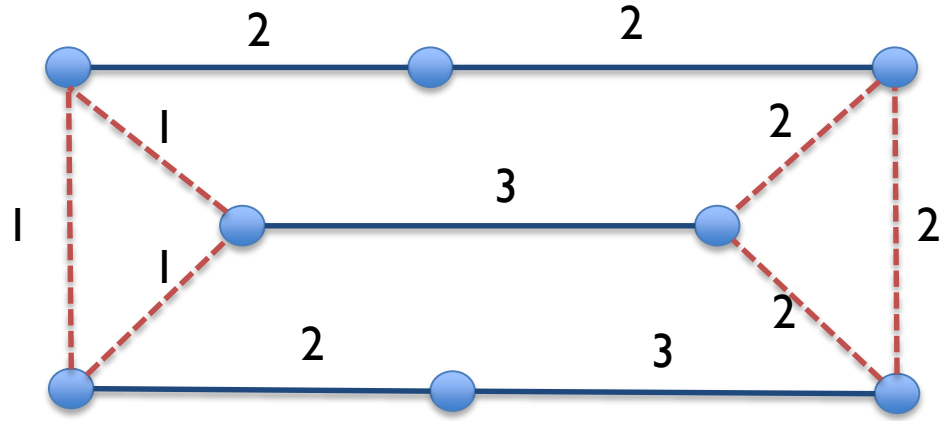


2-Matching Extreme Points

- A graph consisting of disjoint odd cycles connected a matching of paths
- Arbitrary distances on edges
- Held-Karp value = half the weight of cycles + total weight of paths

Can you always find a tour of cost at most 4/3 of Held-Karp?

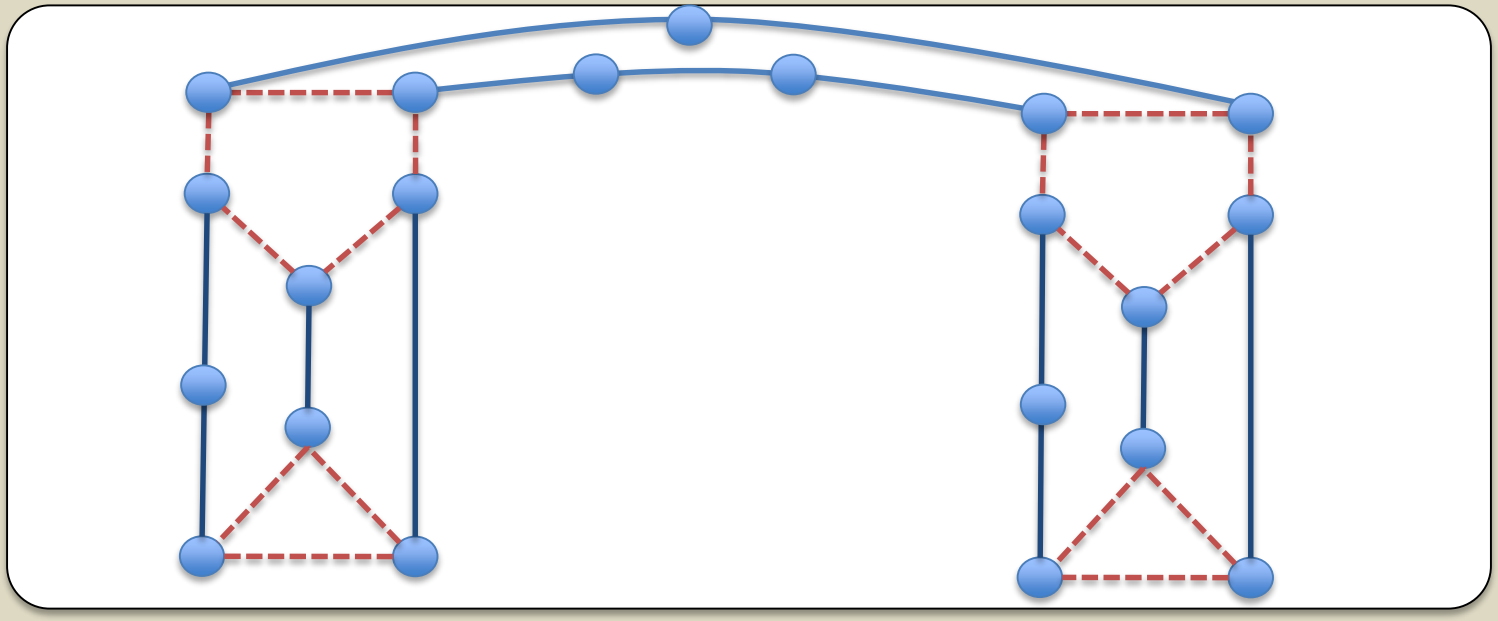
$$\mathrm{HK} = \frac{1+1+1+2+2+2}{2} + 2 + 2 + 3 + 2 + 3$$



2-Matching Extreme Points

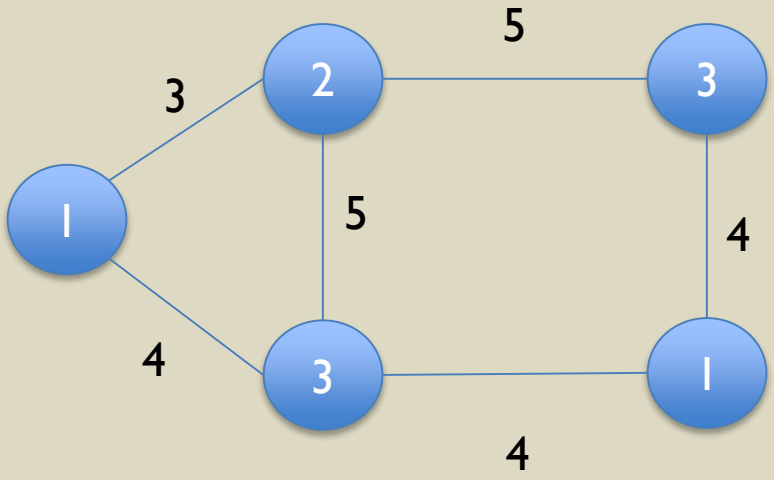
- A graph consisting of disjoint odd cycles connected a matching of paths
- Arbitrary distances on edges
- Held-Karp value = half the weight of cycles + total weight of paths

Can you always find a tour of cost at most 4/3 of Held-Karp?



Node-Weighted Symmetric TSP

- distance of $\{u, v\} \in E$ is $w(u) + w(v)$



Can you do better than Christofides (1.5)?

Summary

- Classic algorithms for TSP
  - Our first approximation algorithm + Christofides
- Two new approaches
- Nice technique: Interpreting a fractional solution in an integral polytope as a distribution
- Great open questions!