Algorithms and Complexity (AC), week 4

Marie Schmidt

based on slides by Jesper Nederlof

LNMB, Sep–Nov 2019

Program for this week and the next

Dealing with NP-hard problems: Approximation

Basic definitions
Ad-hoc approaches
LP-based approaches
Approximation Schemes
In-approximability

Basic definitions

We leave decision problems, and return to optimization problems

Basic definitions

We leave decision problems, and return to optimization problems

Definition

Let X be a minimization problem.
The optimal objective value of instance I is denoted $\mathrm{opt}(I)$.
The objective value returned by algorithm A is denoted $A(I)$.

The *(worst-case) approximation ratio* of algorithm A is $\sup_I A(I)/\mathrm{opt}(I)$.

Basic definitions

We leave decision problems, and return to optimization problems

Definition

Let X be a minimization problem.
The optimal objective value of instance I is denoted $\mathrm{opt}(I)$.
The objective value returned by algorithm A is denoted $A(I)$.

The *(worst-case) approximation ratio* of algorithm A is $\sup_I A(I)/\mathrm{opt}(I)$.

- approximation ratio always ≥ 1

small approximation ratio = good

Basic definitions

We leave decision problems, and return to optimization problems

Definition

Let X be a minimization problem.
The optimal objective value of instance I is denoted $\text{opt}(I)$.
The objective value returned by algorithm A is denoted $A(I)$.

The *(worst-case) approximation ratio* of algorithm A is $\sup_I A(I)/\text{opt}(I)$.

- approximation ratio always ≥ 1

small approximation ratio = good

For maximization problems

Basic definitions

We leave decision problems, and return to optimization problems

Definition

Let X be a minimization problem.
The optimal objective value of instance I is denoted $\text{opt}(I)$.
The objective value returned by algorithm A is denoted $A(I)$.

The *(worst-case) approximation ratio* of algorithm A is $\sup_I A(I)/\text{opt}(I)$.

- approximation ratio always ≥ 1

small approximation ratio = good

For maximization problems approximation ratio is $\inf_I A(I)/\text{opt}(I)$
always ≤ 1; large guarantee = good

Basic definitions

We leave decision problems, and return to optimization problems

Definition

Let X be a minimization problem.
The optimal objective value of instance I is denoted $\mathrm{opt}(I)$.
The objective value returned by algorithm A is denoted $A(I)$.

The *(worst-case) approximation ratio* of algorithm A is $\sup_I A(I)/\mathrm{opt}(I)$.

- approximation ratio always ≥ 1

small approximation ratio = good

For maximization problems approximation ratio is $\inf_I A(I)/\mathrm{opt}(I)$
always ≤ 1; large guarantee = good

We aim for polynomial time algorithms with good approximation ratios

Basic definitions

We leave decision problems, and return to optimization problems

Definition

Let X be a minimization problem.
The optimal objective value of instance I is denoted $\mathrm{opt}(I)$.
The objective value returned by algorithm A is denoted $A(I)$.

The *(worst-case) approximation ratio* of algorithm A is $\sup_I A(I)/\mathrm{opt}(I)$.

- approximation ratio always ≥ 1

small approximation ratio = good

For maximization problems approximation ratio is $\inf_I A(I)/\mathrm{opt}(I)$
always ≤ 1; large guarantee = good

We aim for polynomial time algorithms with good approximation ratios
still possible for problems whose decision versions are NP-complete!

Ad-hoc approaches

Vertex cover (VC)

Instance: undirected graph $G = (V, E)$
Goal: find a vertex cover of smallest possible size
(vertex cover = subset of vertices that touches every edge)

How can we find a 'good' vertex cover?

Vertex cover (VC)

Instance: undirected graph $G = (V, E)$
Goal: find a vertex cover of smallest possible size
(vertex cover = subset of vertices that touches every edge)

How can we find a 'good' vertex cover?

Matching

Subset $M \subseteq E$ of disjoint edges (e.g. $e \cap e' = \emptyset$, for distinct $e, e' \in M$)

Vertex cover (VC)

Instance: undirected graph $G = (V, E)$
Goal: find a vertex cover of smallest possible size
(vertex cover = subset of vertices that touches every edge)

How can we find a 'good' vertex cover?

Matching

Subset $M \subseteq E$ of disjoint edges (e.g. $e \cap e' = \emptyset$, for distinct $e, e' \in M$)
Say M is maximal if there is no matching M' such that $M \subset M'$.

Vertex cover (VC)

Instance: undirected graph $G = (V, E)$
Goal: find a vertex cover of smallest possible size
(vertex cover = subset of vertices that touches every edge)

How can we find a 'good' vertex cover?

Matching

Subset $M \subseteq E$ of disjoint edges (e.g. $e \cap e' = \emptyset$, for distinct $e, e' \in M$)
Say M is maximal if there is no matching M' such that $M \subset M'$.

Approximation algorithm for Vertex Cover

Vertex cover (VC)

Instance: undirected graph $G = (V, E)$
Goal: find a vertex cover of smallest possible size
(vertex cover = subset of vertices that touches every edge)

How can we find a 'good' vertex cover?

Matching

Subset $M \subseteq E$ of disjoint edges (e.g. $e \cap e' = \emptyset$, for distinct $e, e' \in M$)
Say M is maximal if there is no matching M' such that $M \subset M'$.

Approximation algorithm for Vertex Cover

1. Find a maximal matching M (iteratively pick edges not yet touched)

Vertex cover (VC)

Instance: undirected graph $G = (V, E)$
Goal: find a vertex cover of smallest possible size
(vertex cover = subset of vertices that touches every edge)

How can we find a 'good' vertex cover?

Matching

Subset $M \subseteq E$ of disjoint edges (e.g. $e \cap e' = \emptyset$, for distinct $e, e' \in M$)
Say M is maximal if there is no matching M' such that $M \subset M'$.

Approximation algorithm for Vertex Cover

1. Find a maximal matching M (iteratively pick edges not yet touched)
2. Output $S = \bigcup_{\{u,v\} \in M} \{u, v\}$ (e.g. all endpoint of M)

Vertex cover (VC)

Instance: undirected graph $G = (V, E)$
Goal: find a vertex cover of smallest possible size
(vertex cover = subset of vertices that touches every edge)

How can we find a 'good' vertex cover?

Matching

Subset $M \subseteq E$ of disjoint edges (e.g. $e \cap e' = \emptyset$, for distinct $e, e' \in M$)
Say M is maximal if there is no matching M' such that $M \subset M'$.

Approximation algorithm for Vertex Cover

1. Find a maximal matching M (iteratively pick edges not yet touched)
2. Output $S = \bigcup_{\{u,v\} \in M} \{u, v\}$ (e.g. all endpoint of M)

Theorem

This poly-time approximation algorithm has approximation ratio 2.

Makespan minimization

Instance: m machines; n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to machines so that the maximum workload (= makespan) is minimized

Decision variant in NP, NP-hard (thus NP-complete) already for $m = 2$:

Makespan minimization

Instance: m machines; n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to machines so that the maximum workload (= makespan) is minimized

Decision variant in NP, NP-hard (thus NP-complete) already for $m = 2$: use a reduction from 2-PARTITION

Makespan minimization

Instance: m machines; n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to machines so that the maximum workload (= makespan) is minimized

Decision variant in NP, NP-hard (thus NP-complete) already for $m = 2$: use a reduction from 2-PARTITION

List scheduling algorithm

Work through the job list one by one; in each step
assign current job to machine with currently smallest workload

Makespan minimization

Instance: m machines; n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to machines so that the maximum workload (= makespan) is minimized

Decision variant in NP, NP-hard (thus NP-complete) already for $m = 2$: use a reduction from 2-PARTITION

List scheduling algorithm

Work through the job list one by one; in each step
assign current job to machine with currently smallest workload

Theorem

Makespan minimization

Instance: m machines; n jobs with processing times $p_1, \ldots, p_n$
Goal: assign jobs to machines so that the maximum workload (= makespan) is minimized

Decision variant in NP, NP-hard (thus NP-complete) already for $m = 2$: use a reduction from 2-PARTITION

List scheduling algorithm

Work through the job list one by one; in each step
assign current job to machine with currently smallest workload

Theorem

List scheduling has approximation ratio 2.

Theorem

List scheduling has approximation ratio 2.

Proof:

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$$\mathrm{opt}(I) \geq \max p_i$$
$$\mathrm{opt}(I) \geq \frac{1}{m}\sum_{i=1}^{n} p_i$$

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$$\mathrm{opt}(I) \geq \max p_i$$
$$\mathrm{opt}(I) \geq \frac{1}{m}\sum_{i=1}^{n} p_i$$

2.) Consider machine j that determines the makespan

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$$\begin{aligned} \mathrm{opt}(I) &\geq \max p_i \\ \mathrm{opt}(I) &\geq \frac{1}{m}\sum_{i=1}^{n} p_i \end{aligned}$$

2.) Consider machine j that determines the makespan

Consider last job i assigned to machine j

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$$\begin{aligned} \mathrm{opt}(I) &\geq \max p_i \\ \mathrm{opt}(I) &\geq \frac{1}{m} \sum_{i=1}^{n} p_i \end{aligned}$$

2.) Consider machine j that determines the makespan
Consider last job i assigned to machine j
Consider the moment when i was assigned to j

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$\mathrm{opt}(I) \;\geq\; \max p_i$
$\mathrm{opt}(I) \;\geq\; \frac{1}{m}\sum_{i=1}^{n} p_i$

2.) Consider machine j that determines the makespan
Consider last job i assigned to machine j
Consider the moment when i was assigned to j
At this moment j has workload at most $\frac{1}{m}\sum_{i'=1}^{n} p_{i'} \;\leq\; \mathrm{opt}(I)$

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$\mathrm{opt}(I) \;\geq\; \max p_i$

$\mathrm{opt}(I) \;\geq\; \frac{1}{m}\sum_{i=1}^{n} p_i$

2.) Consider machine j that determines the makespan

Consider last job i assigned to machine j

Consider the moment when i was assigned to j

At this moment j has workload at most $\frac{1}{m}\sum_{i'=1}^{n} p_{i'} \;\leq\; \mathrm{opt}(I)$

job i adds $p_i \leq \max_{i'} p_{i'} \leq \mathrm{opt}(I)$ to workload of j

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$\text{opt}(I) \;\geq\; \max p_i$

$\text{opt}(I) \;\geq\; \frac{1}{m}\sum_{i=1}^{n} p_i$

2.) Consider machine j that determines the makespan

Consider last job i assigned to machine j

Consider the moment when i was assigned to j

At this moment j has workload at most $\frac{1}{m}\sum_{i'=1}^{n} p_{i'} \;\leq\; \text{opt}(I)$

job i adds $p_i \leq \max_{i'} p_{i'} \leq \text{opt}(I)$ to workload of j

Thus $A(I) \leq \frac{1}{m}\sum_{i'=1}^{n} p_{i'} + p_i \leq 2OPT(I)$

Theorem

List scheduling has approximation ratio 2.

Proof:

1.) Lower bounds:

$$\mathrm{opt}(I) \;\geq\; \max p_i$$
$$\mathrm{opt}(I) \;\geq\; \frac{1}{m}\sum_{i=1}^{n} p_i$$

2.) Consider machine j that determines the makespan
Consider last job i assigned to machine j
Consider the moment when i was assigned to j
At this moment j has workload at most $\frac{1}{m}\sum_{i'=1}^{n} p_{i'} \;\leq\; \mathrm{opt}(I)$
job i adds $p_i \leq \max_{i'} p_{i'} \leq \mathrm{opt}(I)$ to workload of j
Thus $A(I) \leq \frac{1}{m}\sum_{i'=1}^{n} p_{i'} + p_i \leq 2OPT(I)$

Is this bound tight?

Theorem

List scheduling has approximation ratio 2.

Proof:
1.) Lower bounds:
$\text{opt}(I) \;\geq\; \max p_i$
$\text{opt}(I) \;\geq\; \frac{1}{m}\sum_{i=1}^{n} p_i$

2.) Consider machine j that determines the makespan
Consider last job i assigned to machine j
Consider the moment when i was assigned to j
At this moment j has workload at most $\frac{1}{m}\sum_{i'=1}^{n} p_{i'} \;\leq\; \text{opt}(I)$
job i adds $p_i \leq \max_{i'} p_{i'} \leq \text{opt}(I)$ to workload of j
Thus $A(I) \leq \frac{1}{m}\sum_{i'=1}^{n} p_{i'} + p_i \leq 2OPT(I)$

Is this bound tight?

Can sharpen above analysis to guarantee $2 - 1/m$ (exercise).

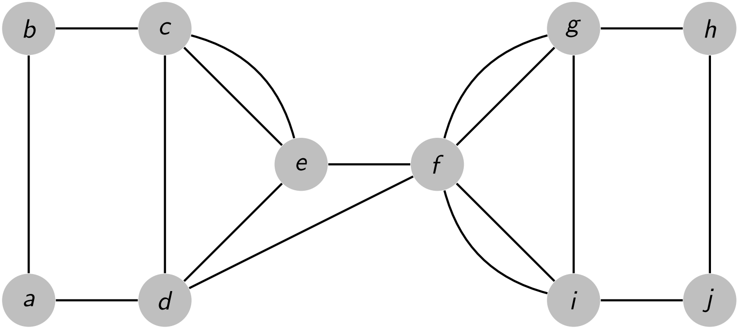
Intermezzo: Euler tours

Intermezzo: Euler tours

- 7 bridges of Köningsberg (Kalingrad): can a tour cross all bridges once?

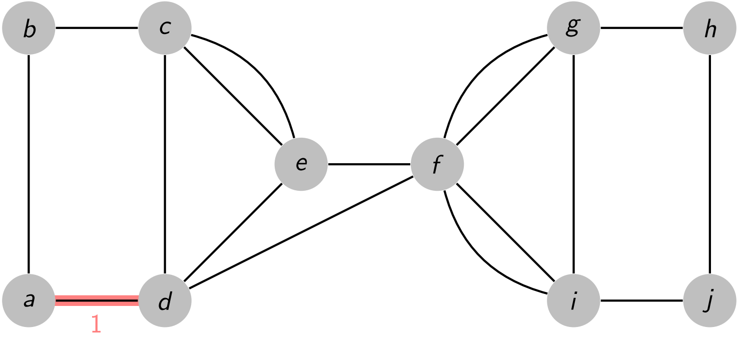
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



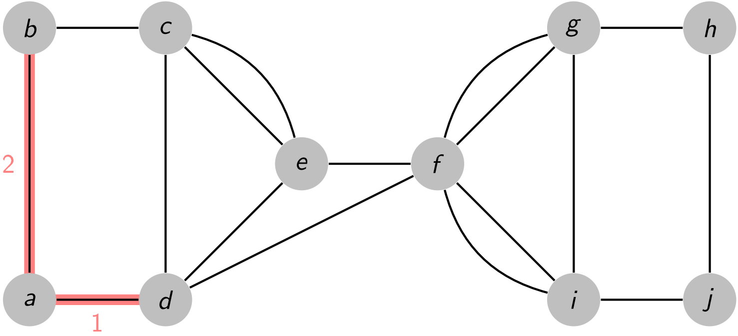
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



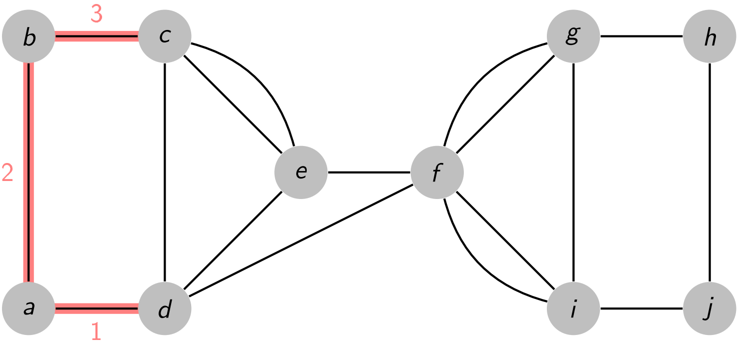
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



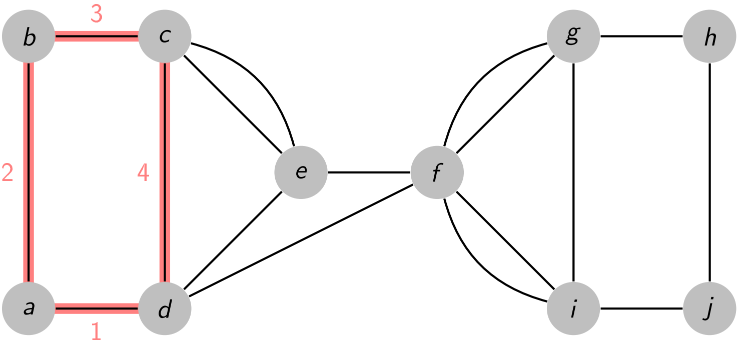
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



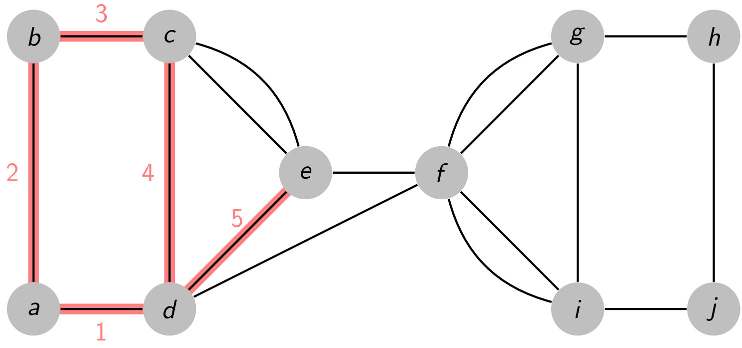
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



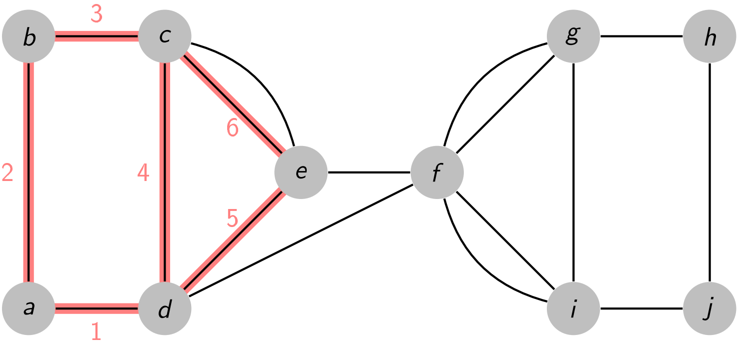
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



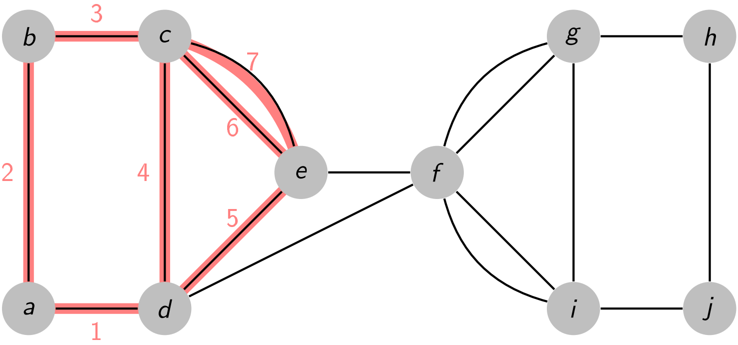
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



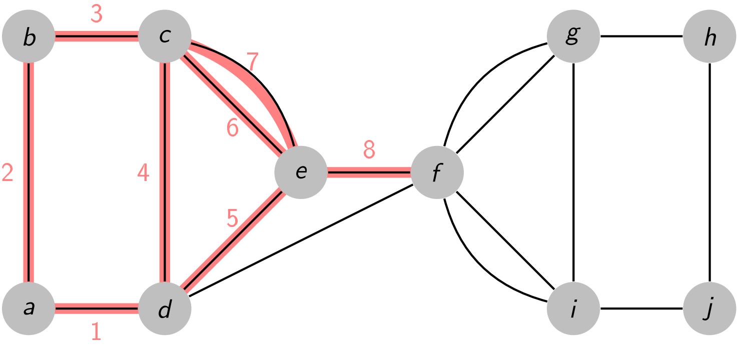
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



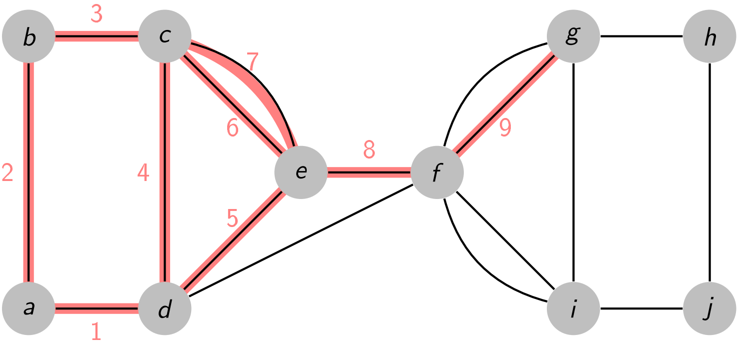
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



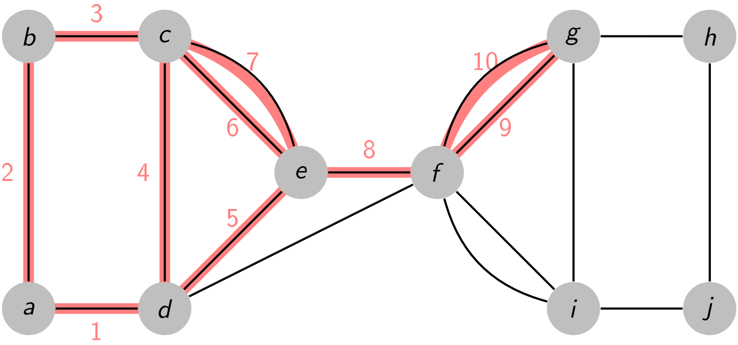
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



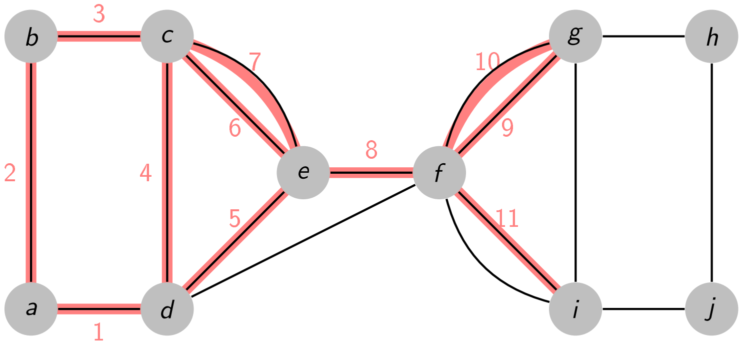
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



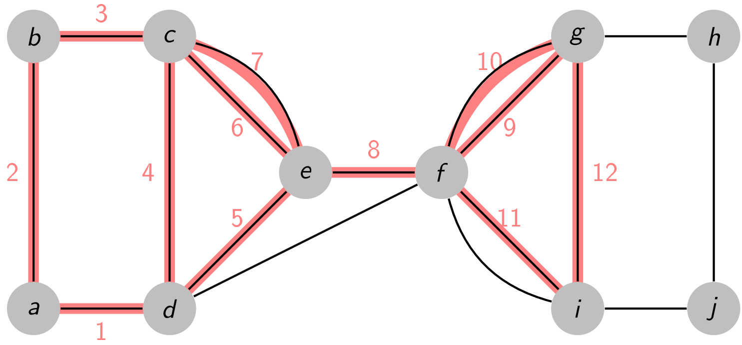
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



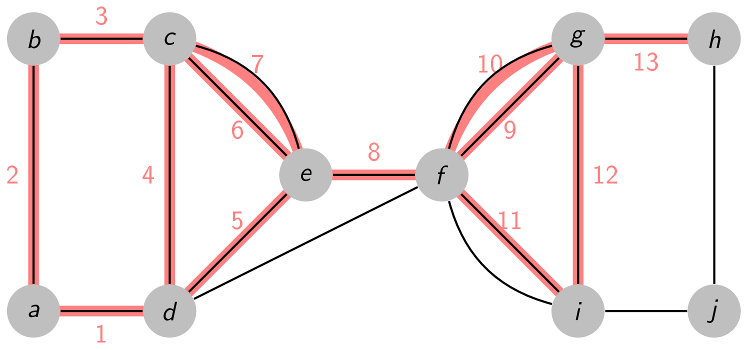
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



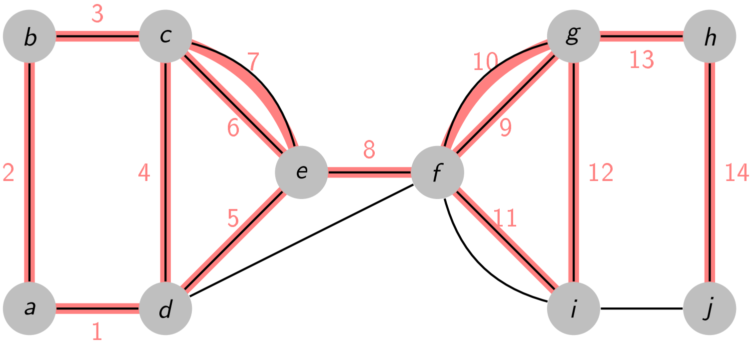
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



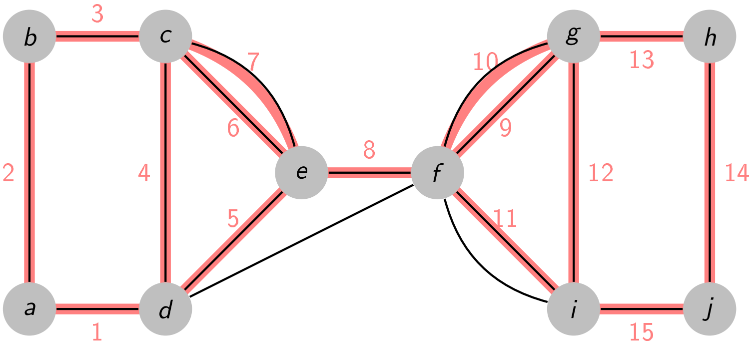
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



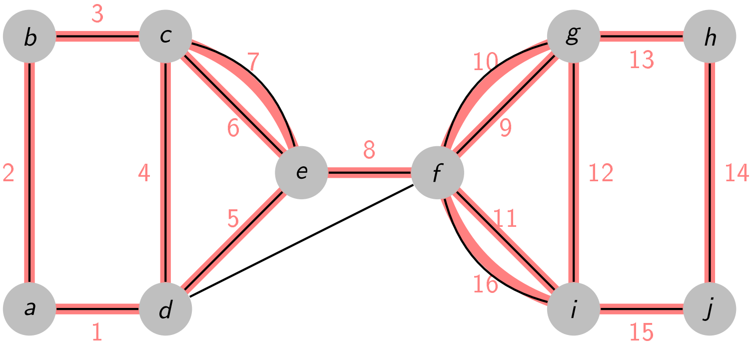
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



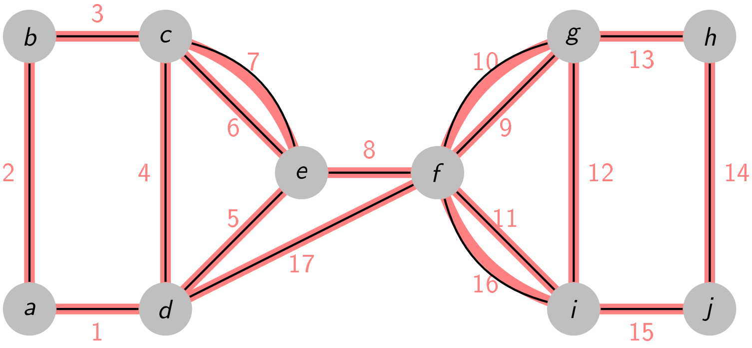
Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



Intermezzo: Euler tours

- Multi-graph: $G = (V, E)$ but now E may be a multi-set
  e.g. some edge $\{u, v\}$ may occur multiple times
- A Eulerian tour of a graph is a tour visiting all *edges* exactly once.



Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Theorem (Euler in 1736, first theorem in graph theory!!)

Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Theorem (Euler in 1736, first theorem in graph theory!!)

A connected graph has a Euler tour iff all vertices have even degree.

(In a multi-graph $G = (V, E)$ the degree of vertex $v \in V$ is the number of edges $\{u, v\} \in E$.)

Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Theorem (Euler in 1736, first theorem in graph theory!!)

A connected graph has a Euler tour iff all vertices have even degree.

(In a multi-graph $G = (V, E)$ the degree of vertex $v \in V$ is the number of edges $\{u, v\} \in E$.)
Proof sketch:

Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Theorem (Euler in 1736, first theorem in graph theory!!)

A connected graph has a Euler tour iff all vertices have even degree.

(In a multi-graph $G = (V, E)$ the degree of vertex $v \in V$ is the number of edges $\{u, v\} \in E$.)
Proof sketch:
($\rightarrow$) the tour enters and leaves vertices consecutively; ends at start.

Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Theorem (Euler in 1736, first theorem in graph theory!!)

A connected graph has a Euler tour iff all vertices have even degree.

(In a multi-graph $G = (V, E)$ the degree of vertex $v \in V$ is the number of edges $\{u, v\} \in E$.)

Proof sketch:

($\rightarrow$) the tour enters and leaves vertices consecutively; ends at start.
($\leftarrow$) start at any vertex v walk along unused edges as long as possible
If we end at v and edges are left incident to a visited vertex u,

Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Theorem (Euler in 1736, first theorem in graph theory!!)

A connected graph has a Euler tour iff all vertices have even degree.

(In a multi-graph $G = (V, E)$ the degree of vertex $v \in V$ is the number of edges $\{u, v\} \in E$.)
Proof sketch:

($\rightarrow$) the tour enters and leaves vertices consecutively; ends at start.
($\leftarrow$) start at any vertex v walk along unused edges as long as possible
If we end at v and edges are left incident to a visited vertex u,
walk from u and insert obtained tour in previous tour.

Intermezzo: Euler tours

Euler tour

Instance: A graph $G = (V, E)$.
Question: Is there an Euler tour?

Theorem (Euler in 1736, first theorem in graph theory!!)

A connected graph has a Euler tour iff all vertices have even degree.

(In a multi-graph $G = (V, E)$ the degree of vertex $v \in V$ is the number of edges $\{u, v\} \in E$.)
Proof sketch:

($\rightarrow$) the tour enters and leaves vertices consecutively; ends at start.
($\leftarrow$) start at any vertex v walk along unused edges as long as possible
If we end at v and edges are left incident to a visited vertex u,
walk from u and insert obtained tour in previous tour.

(This is a constructive polynomial-time algorithm)

Travelling Salesman Problem

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length

NP-complete (reduction from Hamiltonian cycle)

Travelling Salesman Problem

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length

NP-complete (reduction from Hamiltonian cycle)

Assumption: *metric* TSP

We now assume that the distances satisfy the triangle inequality
$d(x, y) + d(y, z) \geq d(x, z)$ for all cities x, y, z

Important example: Points in the plane (triangle ineq. by Pythagoras).
Even here, still NP-complete (harder reduction, beyond scope for us)

Travelling Salesman Problem

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i,j)$
Goal: find roundtrip of smallest possible length

NP-complete (reduction from Hamiltonian cycle)

Assumption: *metric* TSP

We now assume that the distances satisfy the triangle inequality
$d(x,y) + d(y,z) \;\geq\; d(x,z)$ for all cities x, y, z

Important example: Points in the plane (triangle ineq. by Pythagoras).
Even here, still NP-complete (harder reduction, beyond scope for us)

Can we approximate the metric TSP?

Travelling Salesman Problem

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length

NP-complete (reduction from Hamiltonian cycle)

Assumption: *metric* TSP

We now assume that the distances satisfy the triangle inequality
$d(x, y) + d(y, z) \;\geq\; d(x, z)$ for all cities x, y, z

Important example: Points in the plane (triangle ineq. by Pythagoras).
Even here, still NP-complete (harder reduction, beyond scope for us)

Can we approximate the metric TSP?

Lower bounds:

Travelling Salesman Problem

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length

NP-complete (reduction from Hamiltonian cycle)

Assumption: *metric* TSP

We now assume that the distances satisfy the triangle inequality
$d(x, y) + d(y, z) \;\geq\; d(x, z)$ for all cities x, y, z

Important example: Points in the plane (triangle ineq. by Pythagoras).
Even here, still NP-complete (harder reduction, beyond scope for us)

Can we approximate the metric TSP?

Lower bounds:

- $\mathrm{opt}(I) \geq$ length of minimum spanning tree MST

Travelling Salesman Problem

TSP (Optimization version)

Instance: cities $1, \ldots, n$; distances $d(i, j)$
Goal: find roundtrip of smallest possible length

NP-complete (reduction from Hamiltonian cycle)

Assumption: *metric* TSP

We now assume that the distances satisfy the triangle inequality
$d(x, y) + d(y, z) \;\geq\; d(x, z)$ for all cities x, y, z

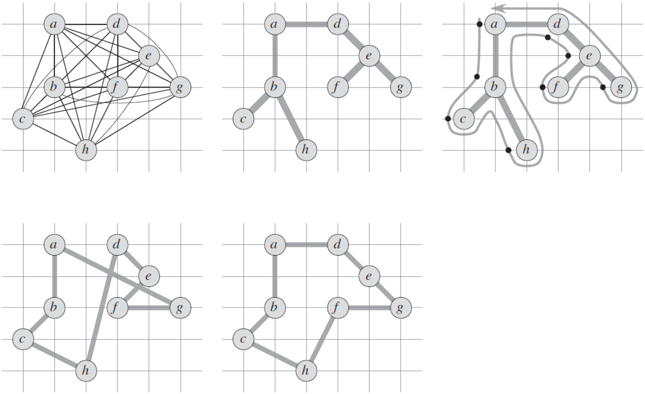
Important example: Points in the plane (triangle ineq. by Pythagoras).
Even here, still NP-complete (harder reduction, beyond scope for us)

Can we approximate the metric TSP?

Lower bounds:

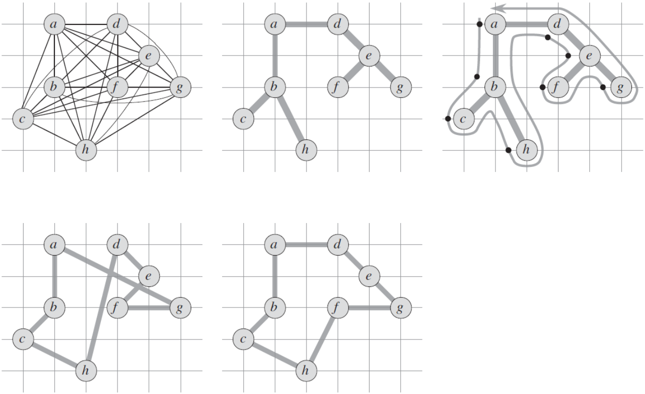
- $\mathrm{opt}(I) \geq$ length of minimum spanning tree MST
- $\mathrm{opt}(I) \geq$ twice the length of min. weight perfect matching, if n even

Double-tree algorithm



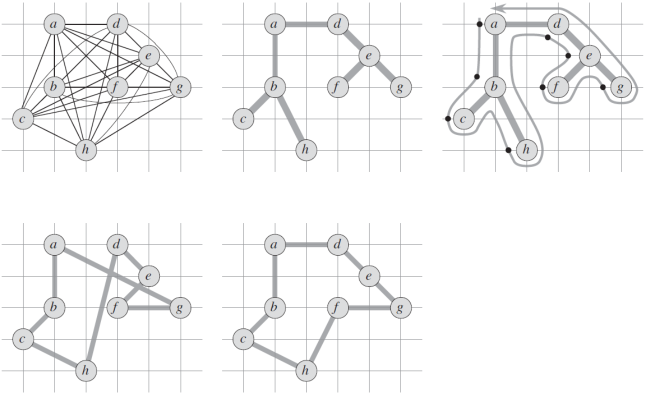
Double-tree algorithm

1. Compute a minimum spanning tree MST



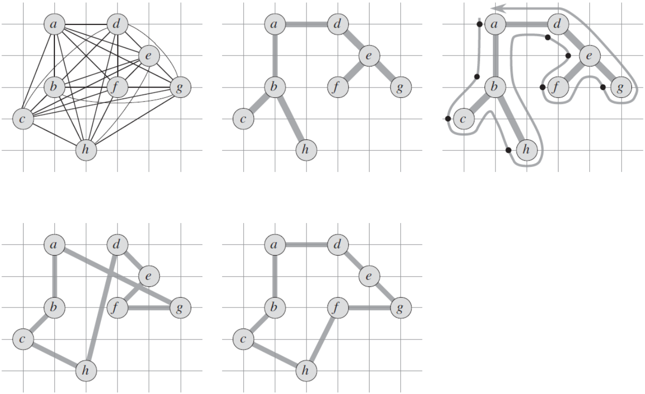
Double-tree algorithm

1. Compute a minimum spanning tree MST
2. Double every edge in MST to get a Eulerian graph



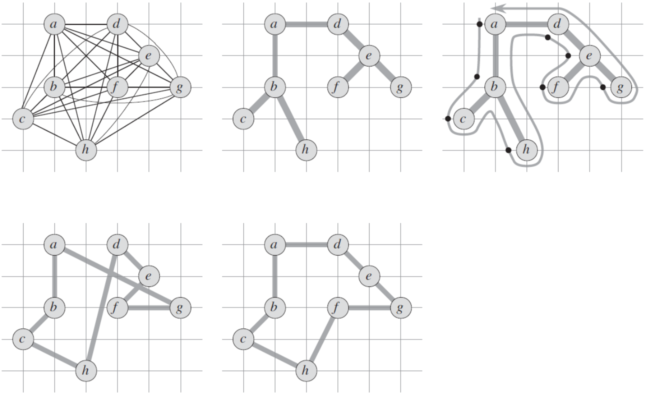
Double-tree algorithm

1. Compute a minimum spanning tree MST
2. Double every edge in MST to get a Eulerian graph
3. Compute a Euler tour in the doubled MST



Double-tree algorithm

1. Compute a minimum spanning tree MST
2. Double every edge in MST to get a Eulerian graph
3. Compute a Euler tour in the doubled MST
4. Shortcut the Euler tour to a TSP tour



Double-tree algorithm

1. Compute a minimum spanning tree MST
2. Double every edge in MST to get a Eulerian graph
3. Compute a Euler tour in the doubled MST
4. Shortcut the Euler tour to a TSP tour

Double-tree algorithm

1. Compute a minimum spanning tree MST
2. Double every edge in MST to get a Eulerian graph
3. Compute a Euler tour in the doubled MST
4. Shortcut the Euler tour to a TSP tour

Theorem

The Double-tree algorithm has approximation ratio 2.

Double-tree algorithm

1. Compute a minimum spanning tree MST
2. Double every edge in MST to get a Eulerian graph
3. Compute a Euler tour in the doubled MST
4. Shortcut the Euler tour to a TSP tour

Theorem

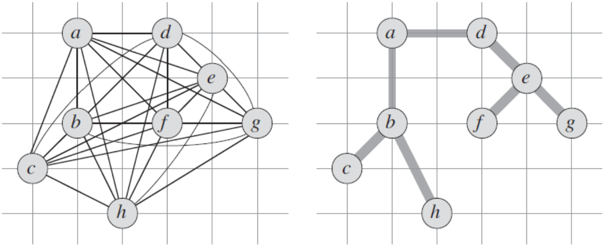
The Double-tree algorithm has approximation ratio 2.

$A(I) \leq$ twice length of MST $\leq 2 \cdot \mathrm{opt}(I)$

Christofides-Serdyukov algorithm

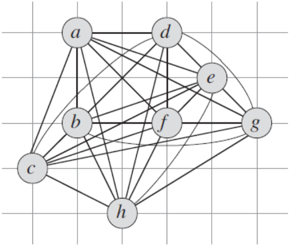
Christofides-Serdyukov algorithm

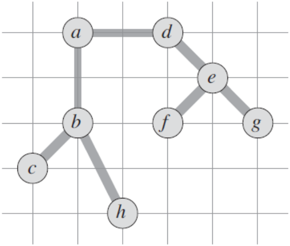
1. Compute a minimum spanning tree MST



Christofides-Serdyukov algorithm

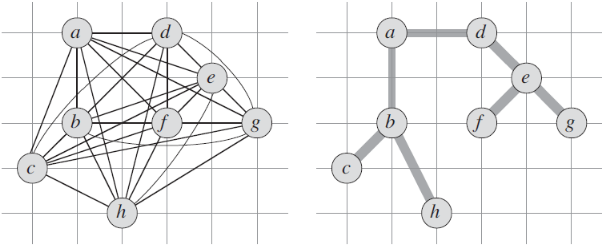
1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST





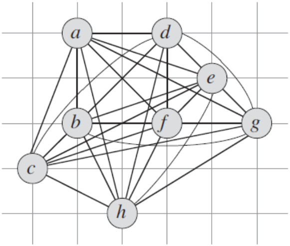
Christofides-Serdyukov algorithm

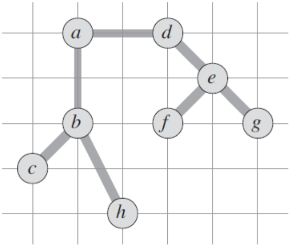
1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*



Christofides-Serdyukov algorithm

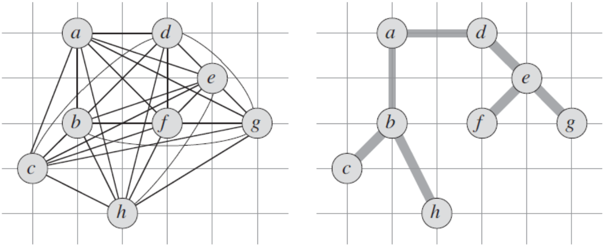
1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph





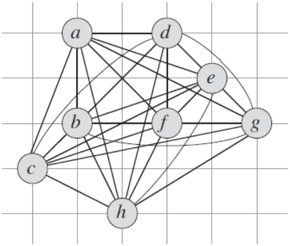
Christofides-Serdyukov algorithm

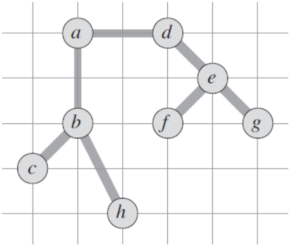
1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M



Christofides-Serdyukov algorithm

1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M
6. Shortcut the Euler tour to a TSP tour





Christofides-Serdyukov algorithm

1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M
6. Shortcut the Euler tour to a TSP tour

Christofides-Serdyukov algorithm

1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M
6. Shortcut the Euler tour to a TSP tour

Theorem

The Christofides-Serdyukov algorithm has approximation ratio 1.5.

Proof: $A(I) \leq$ length of MST plus weight of matching

Christofides-Serdyukov algorithm

1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M
6. Shortcut the Euler tour to a TSP tour

Theorem

The Christofides-Serdyukov algorithm has approximation ratio 1.5*.*

Proof: $A(I) \leq$ length of MST plus weight of matching
weight of matching $\leq 1/2$ length of a tour visiting all odd degree vertices

Christofides-Serdyukov algorithm

1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M
6. Shortcut the Euler tour to a TSP tour

Theorem

The Christofides-Serdyukov algorithm has approximation ratio 1.5.

Proof: $A(I) \leq$ length of MST plus weight of matching
weight of matching $\leq 1/2$ length of a tour visiting all odd degree vertices
$\leq 1/2$ length of a tour visiting all vertices

Christofides-Serdyukov algorithm

1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST
   number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M
6. Shortcut the Euler tour to a TSP tour

Theorem

The Christofides-Serdyukov algorithm has approximation ratio 1.5.

Proof: $A(I) \leq$ length of MST plus weight of matching
weight of matching $\leq 1/2$ length of a tour visiting all odd degree vertices
$\leq 1/2$ length of a tour visiting all vertices

$A(I) \leq$ length of MST plus weight of matching $\leq \mathrm{opt}(I) + \mathrm{opt}(I)/2$

Christofides-Serdyukov algorithm

1. Compute a minimum spanning tree MST
2. Compute a minimum perfect matching M for odd-degree cities in MST number of odd-degree cities always even! (sum of degrees even)
3. *Can find min perfect matching in poly time (non-trivial)*
4. Construct the union of MST and M to get a Eulerian graph
5. Construct a Euler tour in MST union M
6. Shortcut the Euler tour to a TSP tour

Theorem

The Christofides-Serdyukov algorithm has approximation ratio 1.5.

Proof: $A(I) \leq$ length of MST plus weight of matching
weight of matching $\leq 1/2$ length of a tour visiting all odd degree vertices
$\leq 1/2$ length of a tour visiting all vertices

$A(I) \leq$ length of MST plus weight of matching $\leq \text{opt}(I) + \text{opt}(I)/2$

Where did we use the triangle inequality?

LP-based approaches

LP-based approaches

1. Find an exact ILP formulation

LP-based approaches

1. Find an exact ILP formulation
2. Relax integrality constraints (ILP $\rightarrow$ LP)

LP-based approaches

1. Find an exact ILP formulation
2. Relax integrality constraints (ILP $\rightarrow$ LP)
3. Solve the LP relaxation in polynomial time

LP-based approaches

1. Find an exact ILP formulation
2. Relax integrality constraints (ILP $\rightarrow$ LP)
3. Solve the LP relaxation in polynomial time
4. Round the optimal LP solution to approximate ILP solution (preserving feasibility!)

Weighted vertex cover (VC)

Instance: a graph $G = (V, E)$; weights $w : V \to \mathbb{R}^+$
Goal: find a vertex cover of smallest possible weight
(e.g. find a vertex cover $X \subseteq V$ minimizing $\sum_{v \in V} w(v)$)

Weighted vertex cover (VC)

Instance: a graph $G = (V, E)$; weights $w : V \to \mathbb{R}^+$
Goal: find a vertex cover of smallest possible weight
(e.g. find a vertex cover $X \subseteq V$ minimizing $\sum_{v \in V} w(v)$)

ILP formulation

$$
\begin{array}{lll}
\text{minimize} & \sum_{v \in V} w(v) \cdot x_v & \\
\text{subject to} & x_u + x_v \geq 1 & \text{for every edge } \{u, v\} \in E \\
 & \color{red}{x_v \in \{0, 1\}} & \text{for every vertex } v \in V
\end{array}
$$

Weighted vertex cover (VC)

Instance: a graph $G = (V, E)$; weights $w : V \to \mathbb{R}^+$
Goal: find a vertex cover of smallest possible weight
(e.g. find a vertex cover $X \subseteq V$ minimizing $\sum_{v \in V} w(v)$)

ILP formulation

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} w(v) \cdot x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \text{for every edge } \{u, v\} \in E \\ & x_v \in \{0, 1\} & \text{for every vertex } v \in V \end{array}$$

LP relaxation

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} w(v) \cdot x_v & \\ \text{subject to} & x_u + x_v \geq 1 & \text{for every edge } \{u, v\} \in E \\ & 0 \leq x_v \leq 1 & (\text{or simply } 0 \leq x_v) \quad \text{for } v \in V \end{array}$$

Approximation algorithm

1. Compute the optimal LP solution x_v^*

Approximation algorithm

1. Compute the optimal LP solution x_v^*
2. Round the LP solution x_v^* to a feasible ILP-solution $\tilde{x}_v$:

Approximation algorithm

1. Compute the optimal LP solution x_v^*
2. Round the LP solution x_v^* to a feasible ILP-solution $\tilde{x}_v$:
   If $x_v^* < 1/2$ then $\tilde{x}_v = 0$

Approximation algorithm

1. Compute the optimal LP solution x_v^*
2. Round the LP solution x_v^* to a feasible ILP-solution $\tilde{x}_v$:
   If $x_v^* < 1/2$ then $\tilde{x}_v = 0$
   If $x_v^* \geq 1/2$ then $\tilde{x}_v = 1$

Approximation algorithm

1. Compute the optimal LP solution x_v^*
2. Round the LP solution x_v^* to a feasible ILP-solution $\tilde{x}_v$:
   If $x_v^* < 1/2$ then $\tilde{x}_v = 0$
   If $x_v^* \geq 1/2$ then $\tilde{x}_v = 1$

Theorem

Approximation algorithm

1. Compute the optimal LP solution x_v^*
2. Round the LP solution x_v^* to a feasible ILP-solution $\tilde{x}_v$:
   If $x_v^* < 1/2$ then $\tilde{x}_v = 0$
   If $x_v^* \geq 1/2$ then $\tilde{x}_v = 1$

Theorem

This poly-time approximation algorithm has approximation ratio 2.

Proof:

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$\mathrm{opt}_{LP} \;\leq\; \mathrm{opt}_{ILP} \;\leq\; \mathrm{app} \;\leq\; 2\mathrm{opt}_{LP}$

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$\text{opt}_{LP} \leq \text{opt}_{ILP} \leq \text{app} \leq 2\text{opt}_{LP}$

For minimization, want opt_{LP} to be a good lower bound of opt_{ILP}.

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$opt_{LP} \leq opt_{ILP} \leq app \leq 2opt_{LP}$

For minimization, want opt_{LP} to be a good lower bound of opt_{ILP}.

Integrality gap

Define the *integrality gap* of an LP-relaxation to be
the supremum (over all instances) of opt_{ILP}/opt_{LP}.
(for maximization, use infimum)

Two examples (with unit weights)

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$\text{opt}_{LP} \leq \text{opt}_{ILP} \leq \text{app} \leq 2\text{opt}_{LP}$

For minimization, want opt_{LP} to be a good lower bound of opt_{ILP}.

Integrality gap

Define the *integrality gap* of an LP-relaxation to be
the supremum (over all instances) of $\text{opt}_{ILP}/\text{opt}_{LP}$.
(for maximization, use infimum)

Two examples (with unit weights)

- Odd cycle on $2k+1$ vertices yields

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$\text{opt}_{LP} \leq \text{opt}_{ILP} \leq \text{app} \leq 2\text{opt}_{LP}$

For minimization, want opt_{LP} to be a good lower bound of opt_{ILP}.

Integrality gap

Define the *integrality gap* of an LP-relaxation to be
the supremum (over all instances) of $\text{opt}_{ILP}/\text{opt}_{LP}$.
(for maximization, use infimum)

Two examples (with unit weights)

- Odd cycle on $2k+1$ vertices yields
  $\text{opt}_{LP} = k + \frac{1}{2}, \quad \text{opt}_{ILP} = k+1, \quad \text{app} = 2k+1.$

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$\mathrm{opt}_{LP} \leq \mathrm{opt}_{ILP} \leq \mathrm{app} \leq 2\mathrm{opt}_{LP}$

For minimization, want opt_{LP} to be a good lower bound of opt_{ILP}.

Integrality gap

Define the *integrality gap* of an LP-relaxation to be
the supremum (over all instances) of $\mathrm{opt}_{ILP}/\mathrm{opt}_{LP}$.
(for maximization, use infimum)

Two examples (with unit weights)

- Odd cycle on $2k+1$ vertices yields
  $\mathrm{opt}_{LP} = k + \frac{1}{2}$, $\mathrm{opt}_{ILP} = k+1$, $\mathrm{app} = 2k+1$.
- Complete graph on $2k$ vertices yields

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$\mathrm{opt}_{LP} \leq \mathrm{opt}_{ILP} \leq \mathrm{app} \leq 2\mathrm{opt}_{LP}$

For minimization, want opt_{LP} to be a good lower bound of opt_{ILP}.

Integrality gap

Define the *integrality gap* of an LP-relaxation to be
the supremum (over all instances) of $\mathrm{opt}_{ILP}/\mathrm{opt}_{LP}$.
(for maximization, use infimum)

Two examples (with unit weights)

- Odd cycle on $2k+1$ vertices yields
  $\mathrm{opt}_{LP} = k + \frac{1}{2}$, $\mathrm{opt}_{ILP} = k+1$, $\mathrm{app} = 2k+1$.
- Complete graph on $2k$ vertices yields
  $\mathrm{opt}_{LP} = k$, $\mathrm{opt}_{ILP} = 2k-1$, $\mathrm{app} = 2k$.

General approach is centered around three values: opt_{ILP}, opt_{LP}, app (result of the rounding)

Observation

$\mathrm{opt}_{LP} \ \leq \ \mathrm{opt}_{ILP} \ \leq \ \mathrm{app} \ \leq \ 2\mathrm{opt}_{LP}$

For minimization, want opt_{LP} to be a good lower bound of opt_{ILP}.

Integrality gap

Define the *integrality gap* of an LP-relaxation to be
the supremum (over all instances) of $\mathrm{opt}_{ILP}/\mathrm{opt}_{LP}$.
(for maximization, use infimum)

Two examples (with unit weights)

- Odd cycle on $2k+1$ vertices yields
  $\mathrm{opt}_{LP} = k + \frac{1}{2}$, $\mathrm{opt}_{ILP} = k+1$, $\mathrm{app} = 2k+1$.
- Complete graph on $2k$ vertices yields
  $\mathrm{opt}_{LP} = k$, $\mathrm{opt}_{ILP} = 2k-1$, $\mathrm{app} = 2k$.

Therefore the integrality gap of the LP relaxation is $\mathrm{opt}_{ILP}/\mathrm{opt}_{LP} = 2$