Approximation Algorithms

Lecture 1:
Introduction and Vertex Cover

Part I:
Organizational

Alexander Wolff

Winter 2022/23

Organizational

Lectures: on site (in German :-)

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

possibly some lectures via inverted classroom

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

possibly some lectures via inverted classroom

Tutorials: roughly one exercise sheet per lecture

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

possibly some lectures via inverted classroom

Tutorials: roughly one exercise sheet per lecture

discussing old solutions and solving new tasks

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

possibly some lectures via inverted classroom

Tutorials: roughly one exercise sheet per lecture

discussing old solutions and solving new tasks

Tue, 10:15–11:45 (SE I)

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

possibly some lectures via inverted classroom

Tutorials: roughly one exercise sheet per lecture

discussing old solutions and solving new tasks

Tue, 10:15–11:45 (SE I)

Bonus ($+0.3$ on final grade) for $\geq 50\%$ points

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

possibly some lectures via inverted classroom

Tutorials: roughly one exercise sheet per lecture

discussing old solutions and solving new tasks

Tue, 10:15–11:45 (SE I)

Bonus (+0.3 on final grade) for $\geq 50\%$ points

Questions/Tasks during the lecture

Organizational

Lectures: on site (in German :-)

Fri, 10:15–11:45 (ÜR I)

possibly some lectures via inverted classroom

Tutorials: roughly one exercise sheet per lecture

discussing old solutions and solving new tasks

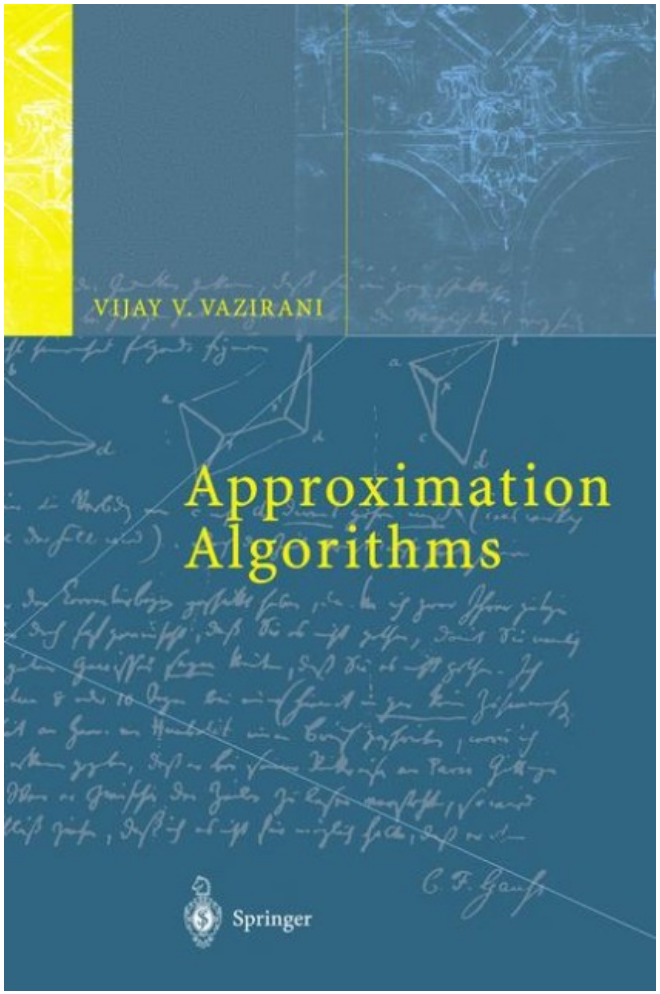
Tue, 10:15–11:45 (SE I)

Bonus (+0.3 on final grade) for $\geq 50\%$ points

Questions/Tasks during the lecture

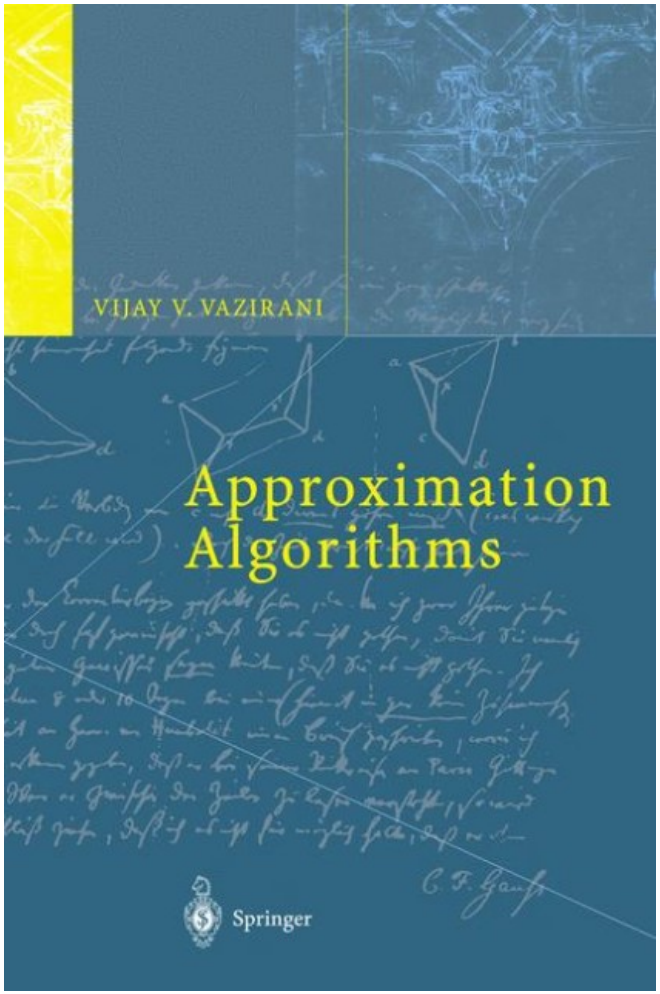
Most slides are due to Joachim Spoerhase. Thanks!

Textbooks

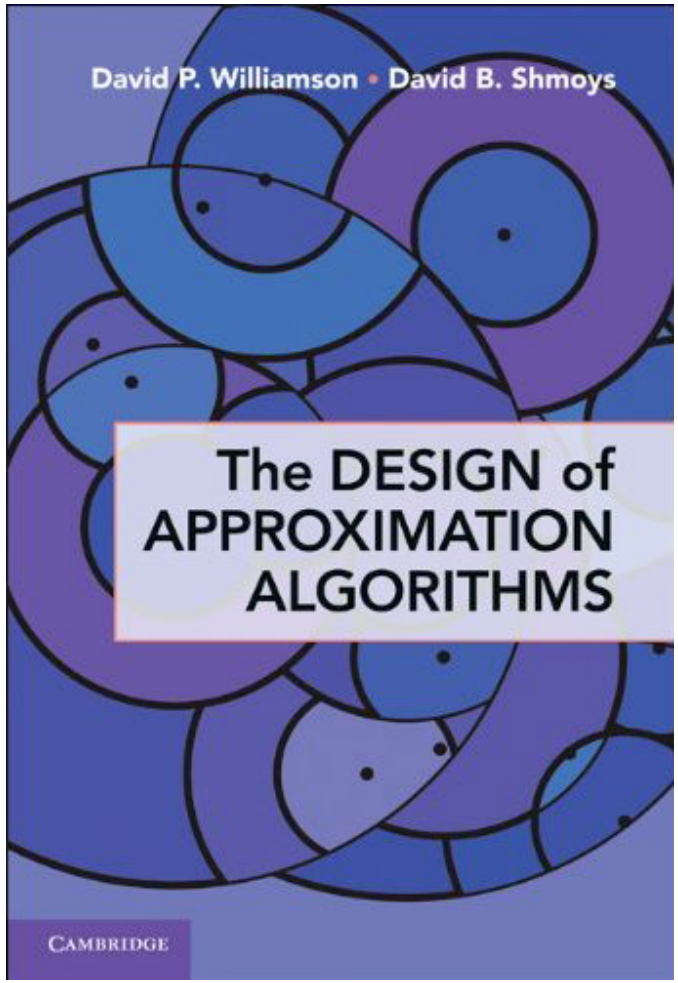


Vijay V. Vazirani:
Approximation Algorithms
Springer-Verlag, 2003.

Textbooks



Vijay V. Vazirani:
Approximation Algorithms
Springer-Verlag, 2003.



D. P. Williamson & D. B. Shmoys:
The Design of Approximation Algorithms
Cambridge-Verlag, 2011.

http://www.designofapproxalgs.com/

Approximation Algorithms

„All exact science is dominated by the idea of approximation.“

– Bertrand Russell
(1872 – 1970)

Approximation Algorithms

- Many optimization problems are NP-hard!
  (For example, the traveling salesperson problem.)

Approximation Algorithms

- Many optimization problems are NP-hard!
  (For example, the traveling salesperson problem.)
- $\rightsquigarrow$ an optimal solution cannot be efficiently computed unless P=NP.

Approximation Algorithms

- Many optimization problems are NP-hard!
  (For example, the traveling salesperson problem.)
- $\rightsquigarrow$ an optimal solution cannot be efficiently computed unless P=NP.
- However, good approximate solutions can often be found efficiently!

Approximation Algorithms

- Many optimization problems are NP-hard!
  (For example, the traveling salesperson problem.)
- $\rightsquigarrow$ an optimal solution cannot be efficiently computed unless P=NP.
- However, good approximate solutions can often be found efficiently!
- **Techniques** for the design and analysis of approximation algorithms arise from studying specific optimization problems.

Overview

Combinatorial algorithms

- Introduction (Vertex Cover)
- Set Cover via Greedy
- Shortest Superstring
  via reduction to SC
- Steiner Tree via MST
- Multiway Cut via Greedy
- k-Center via Parametrized Pruning
- Min-Degree Spanning Tree
  and local search
- Knapsack via DP and Scaling
- Euclidean TSP via Quadtrees

Overview

Combinatorial algorithms

- Introduction (Vertex Cover)
- Set Cover via Greedy
- Shortest Superstring via reduction to SC
- Steiner Tree via MST
- Multiway Cut via Greedy
- k-Center via Parametrized Pruning
- Min-Degree Spanning Tree and local search
- Knapsack via DP and Scaling
- Euclidean TSP via Quadtrees

LP-based algorithms

- introduction to LP-Duality
- Set Cover via LP Rounding
- Set Cover via Primal–Dual Schema
- Maximum Satisfiability
- Scheduling und Extreme Point Solutions
- Steiner Forest via Primal–Dual

Approximation Algorithms

Lecture 1:
Introduction and Vertex Cover

Part II:
(Cardinality) Vertex Cover

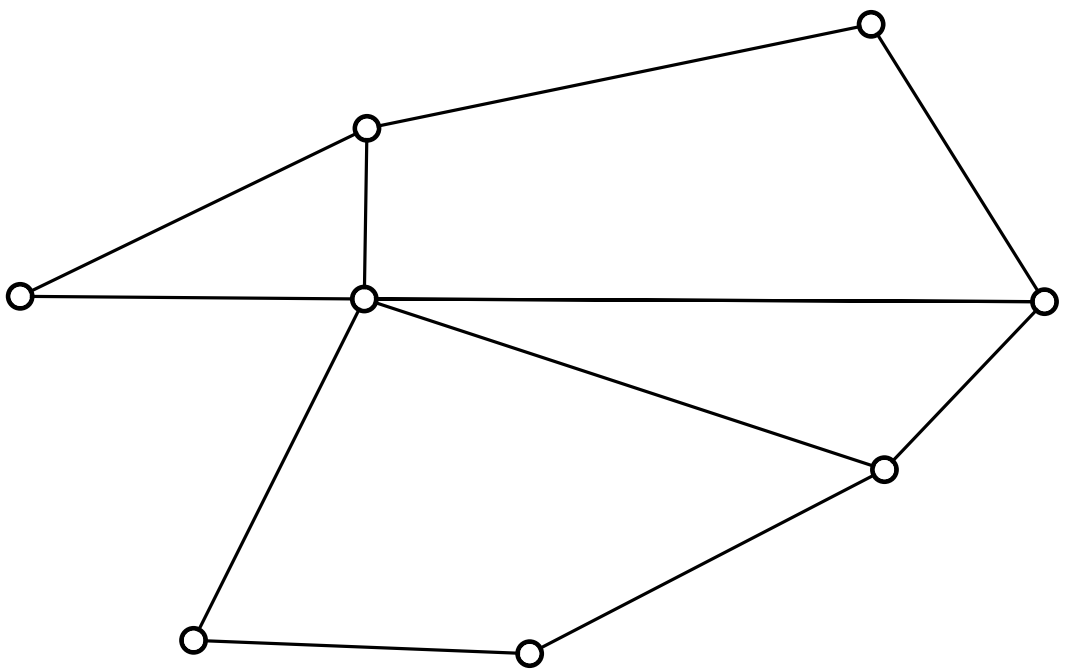
Alexander Wolff

Winter 2022/23

VertexCover (card.)

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

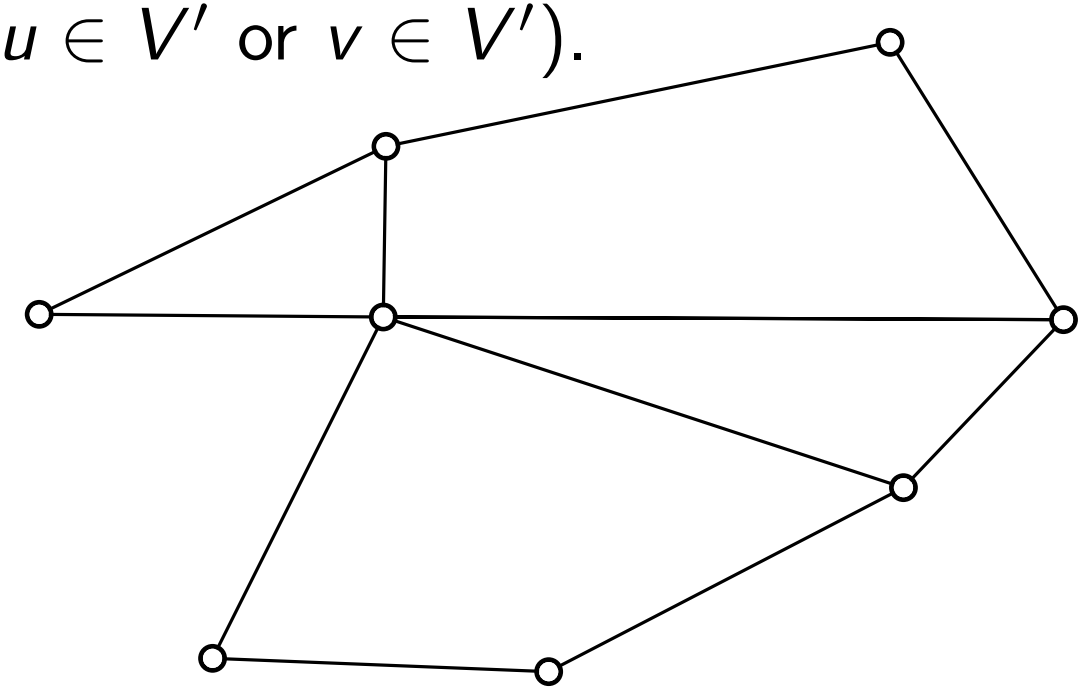
Output:



VertexCover (card.)

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

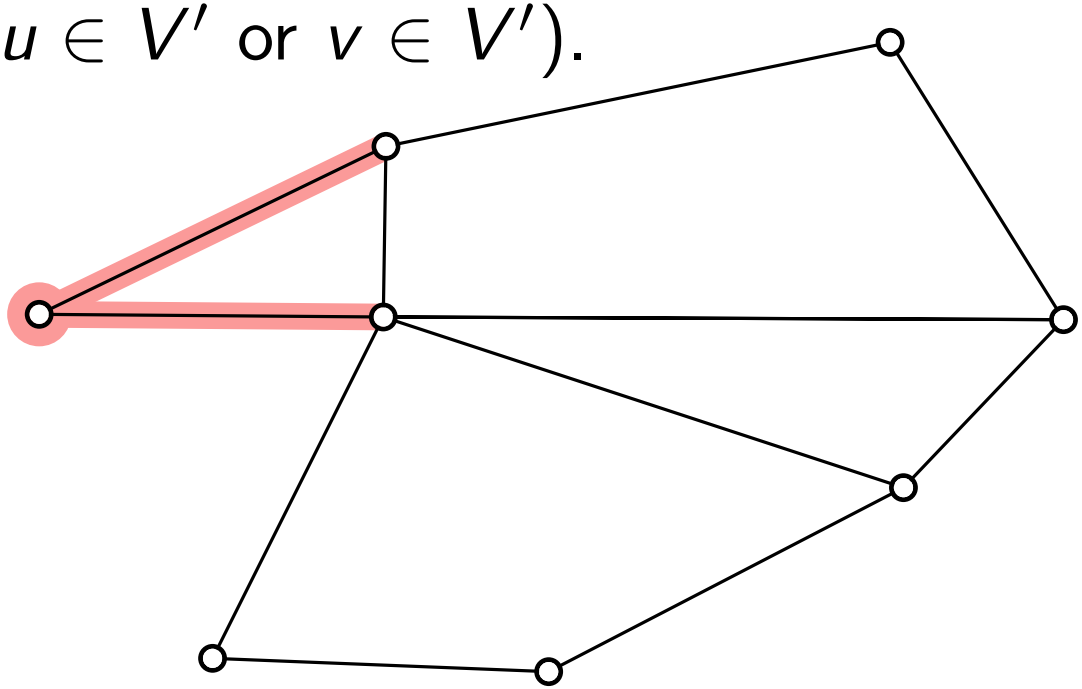
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

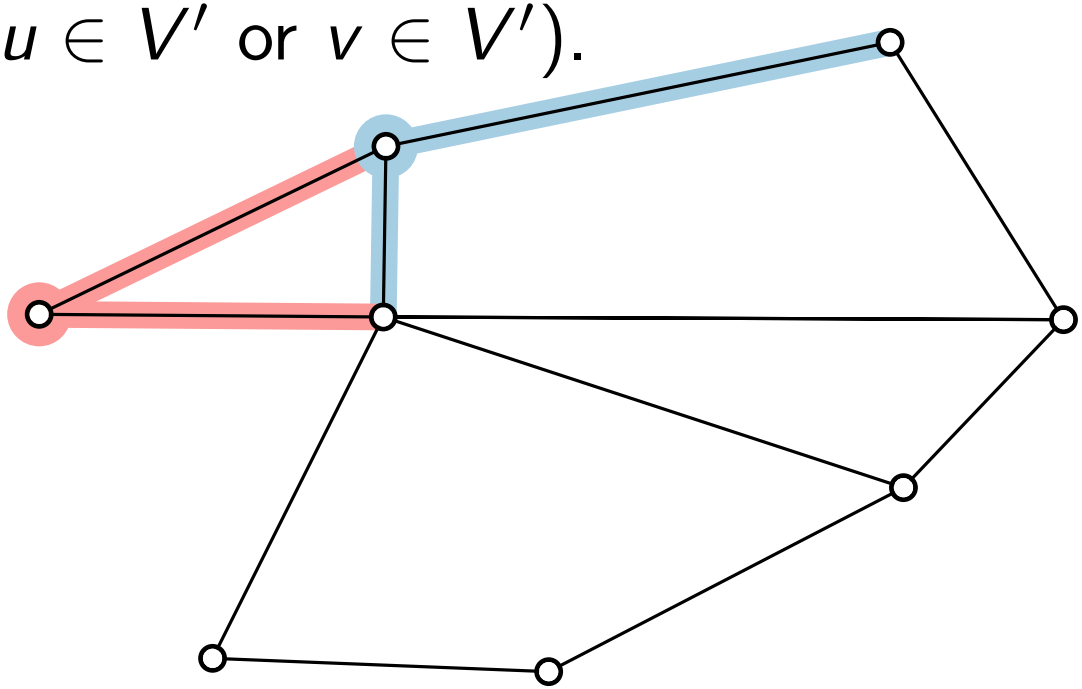
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

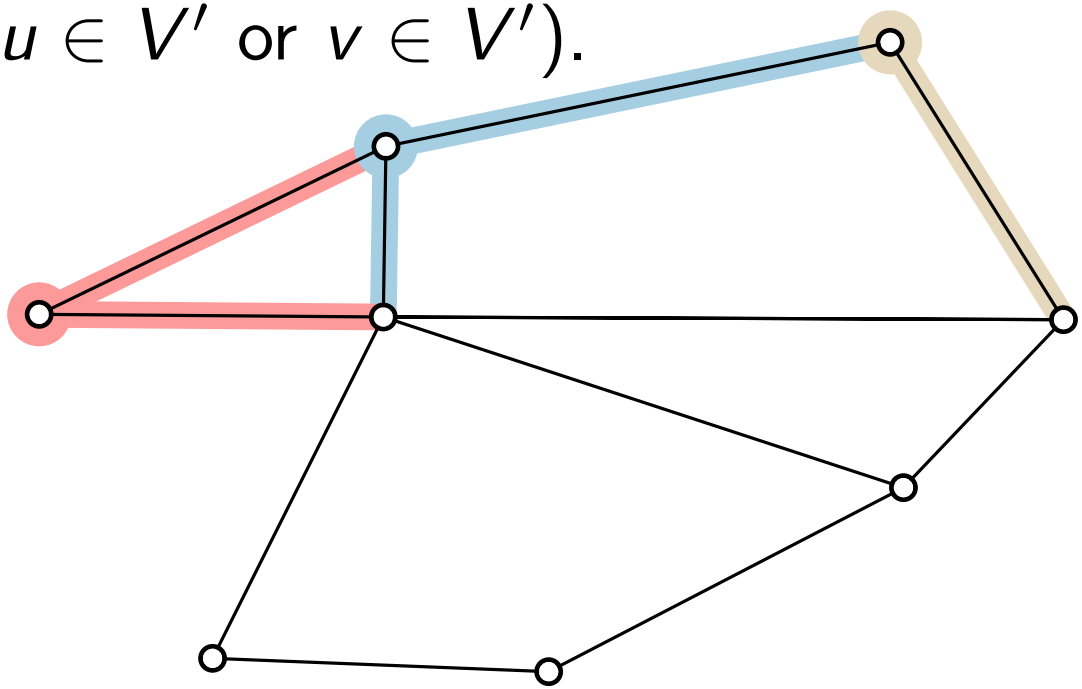
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

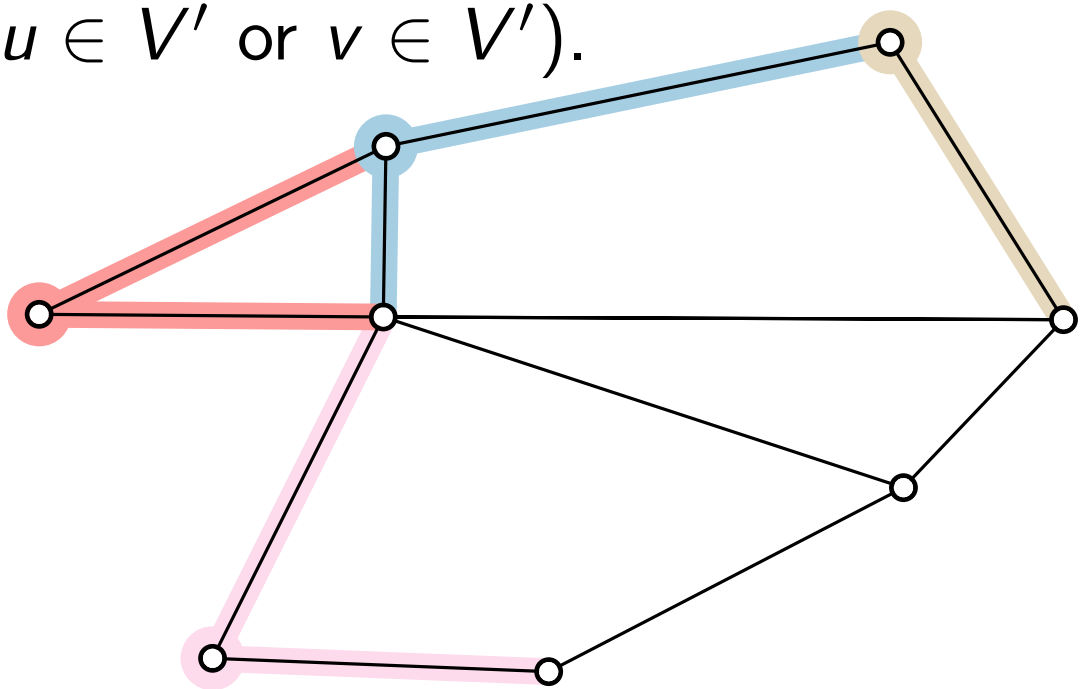
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

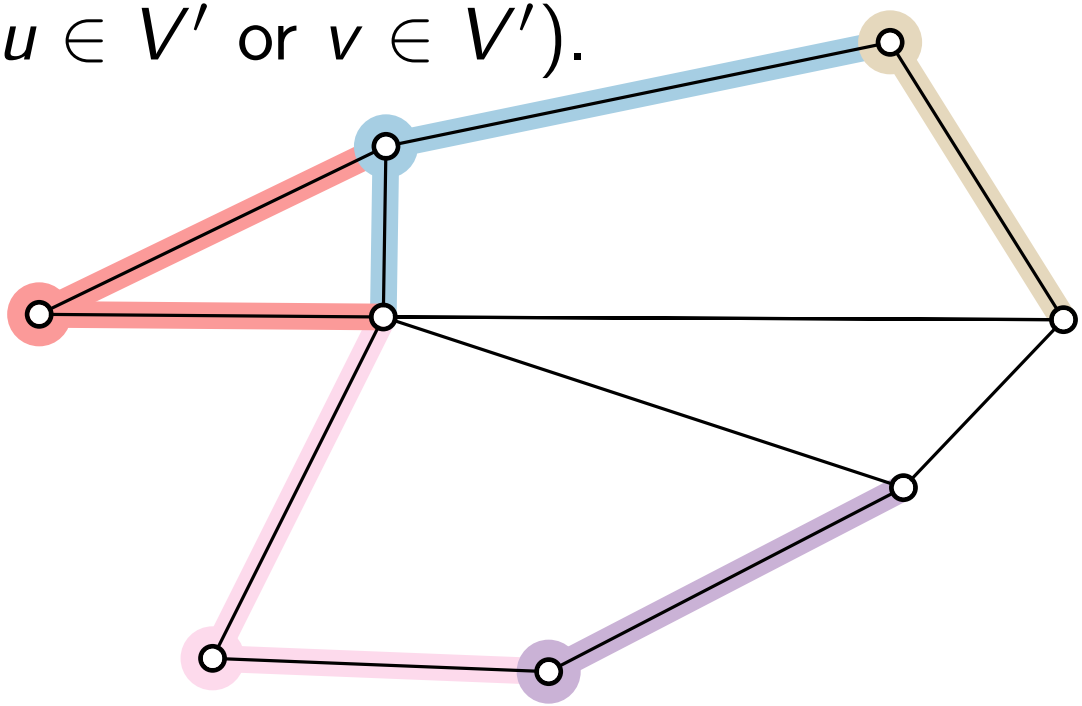
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

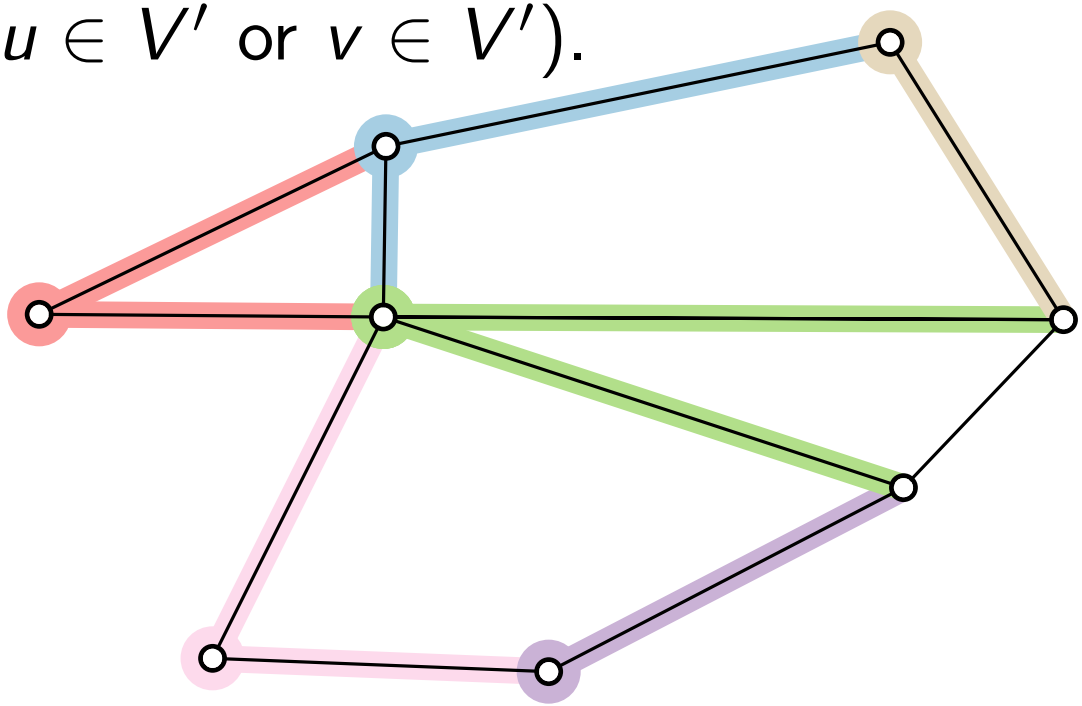
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

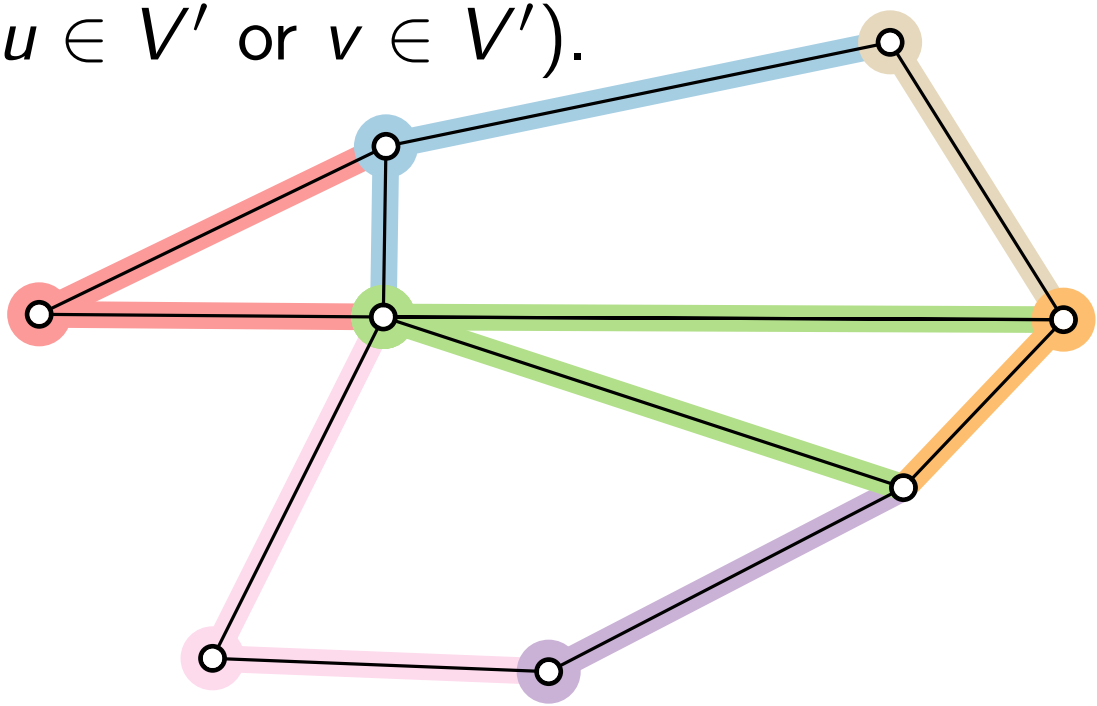
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

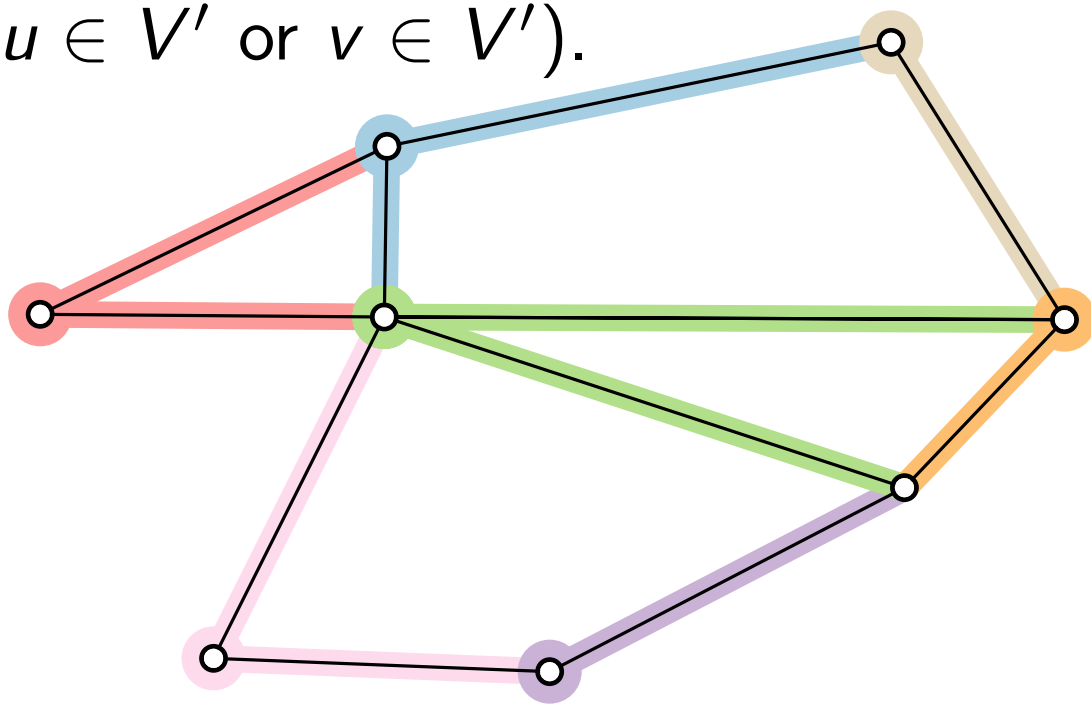
Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



VertexCover (card.)

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

Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).

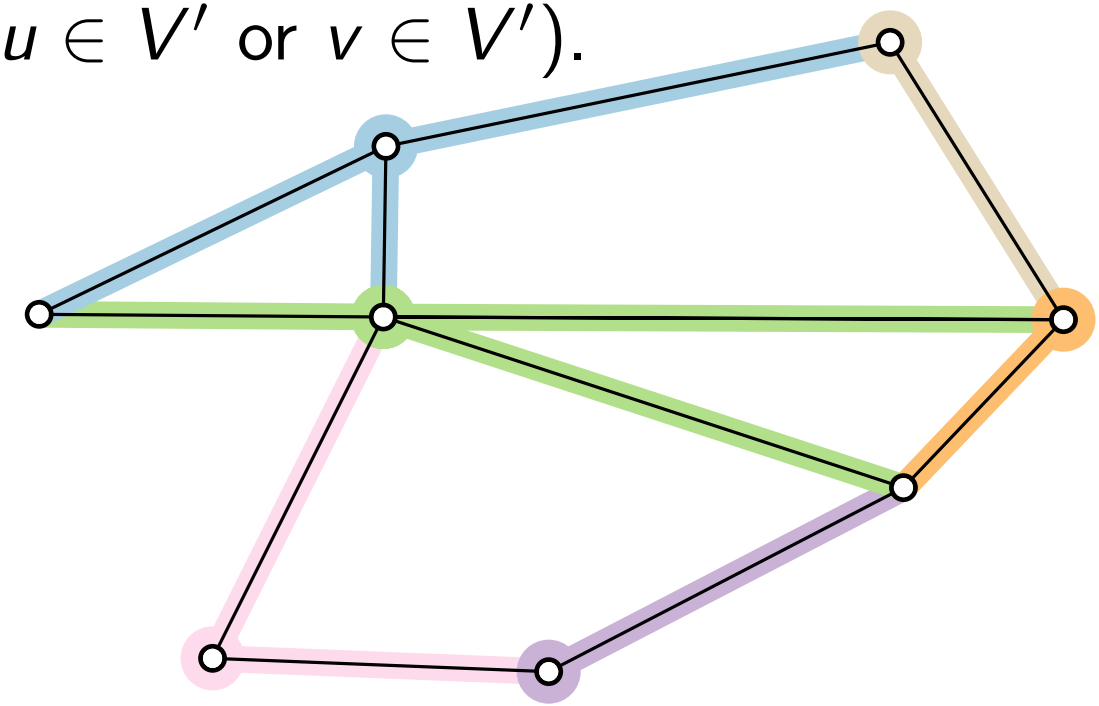


any vertex cover

VertexCover (card.)

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

Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).

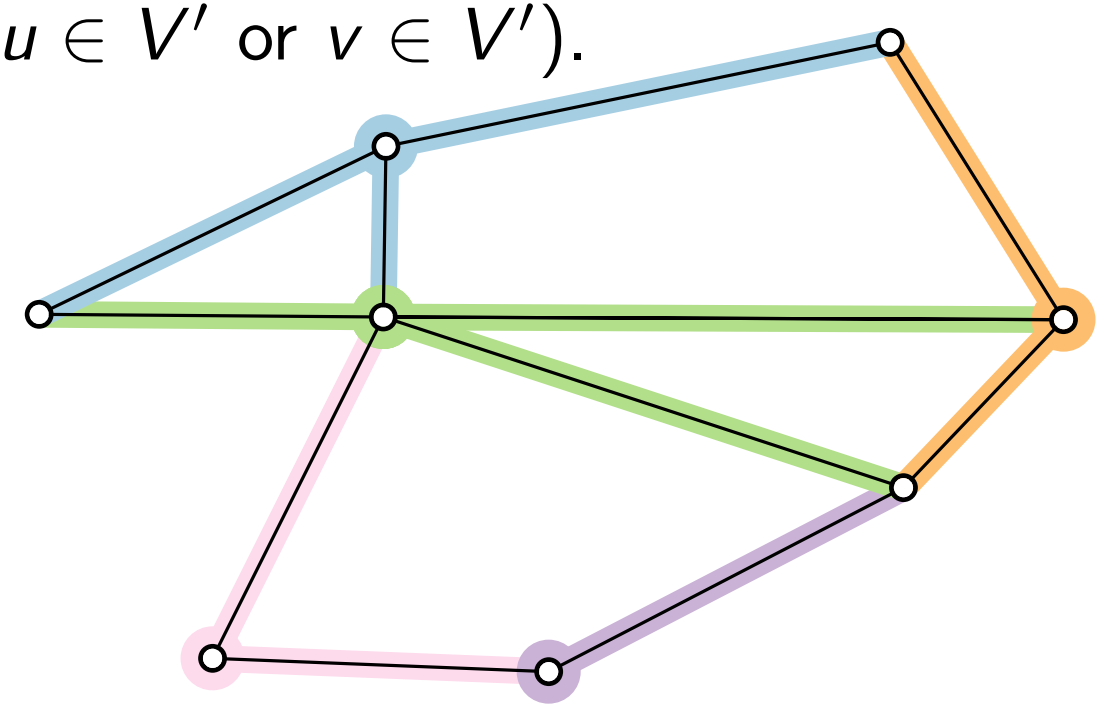


any vertex cover

VertexCover (card.)

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

Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).

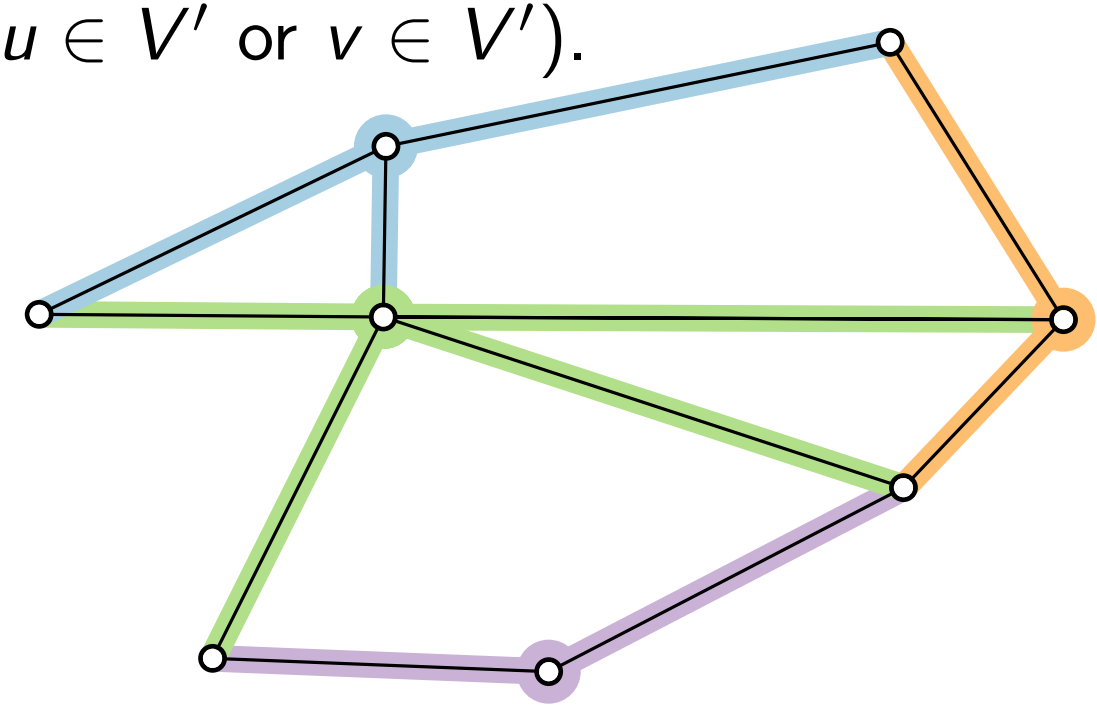


any vertex cover

VertexCover (card.)

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

Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).

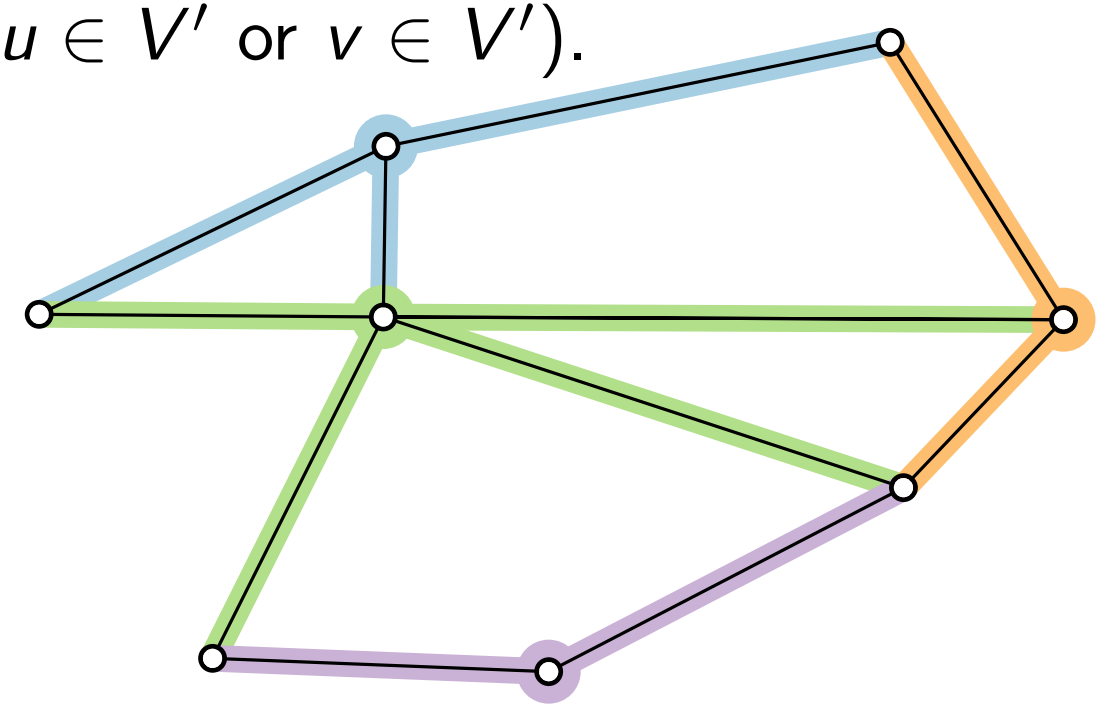


any vertex cover

VertexCover (card.)

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

Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).

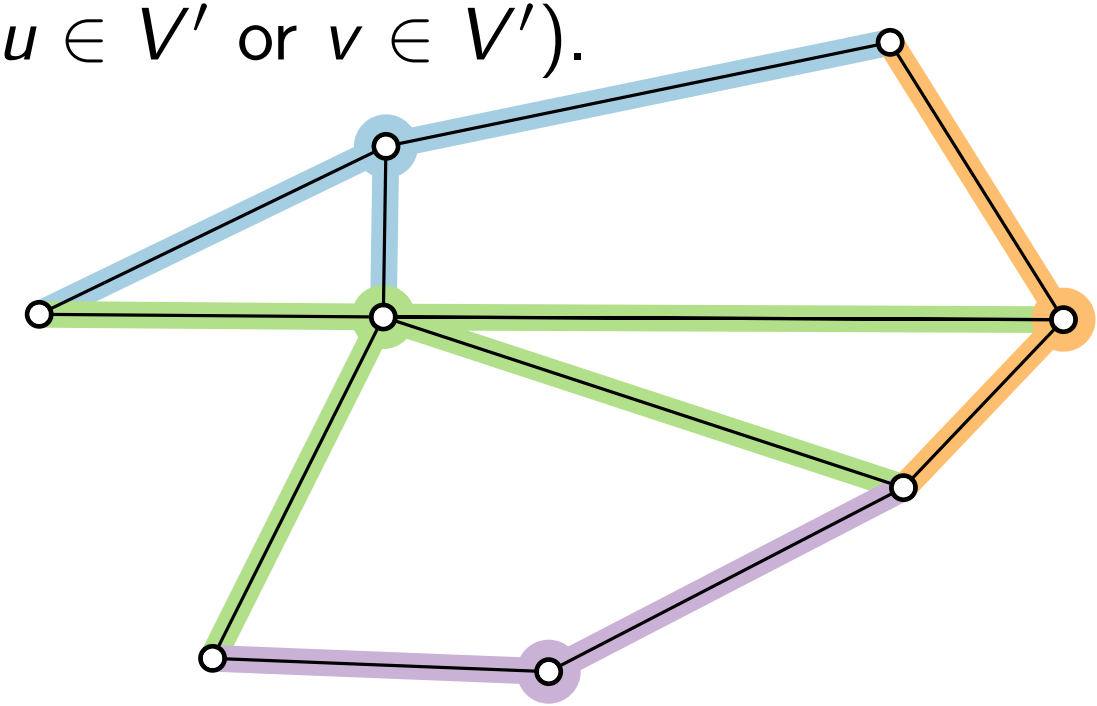


Optimum (OPT $= 4$)

VertexCover (card.)

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

Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).

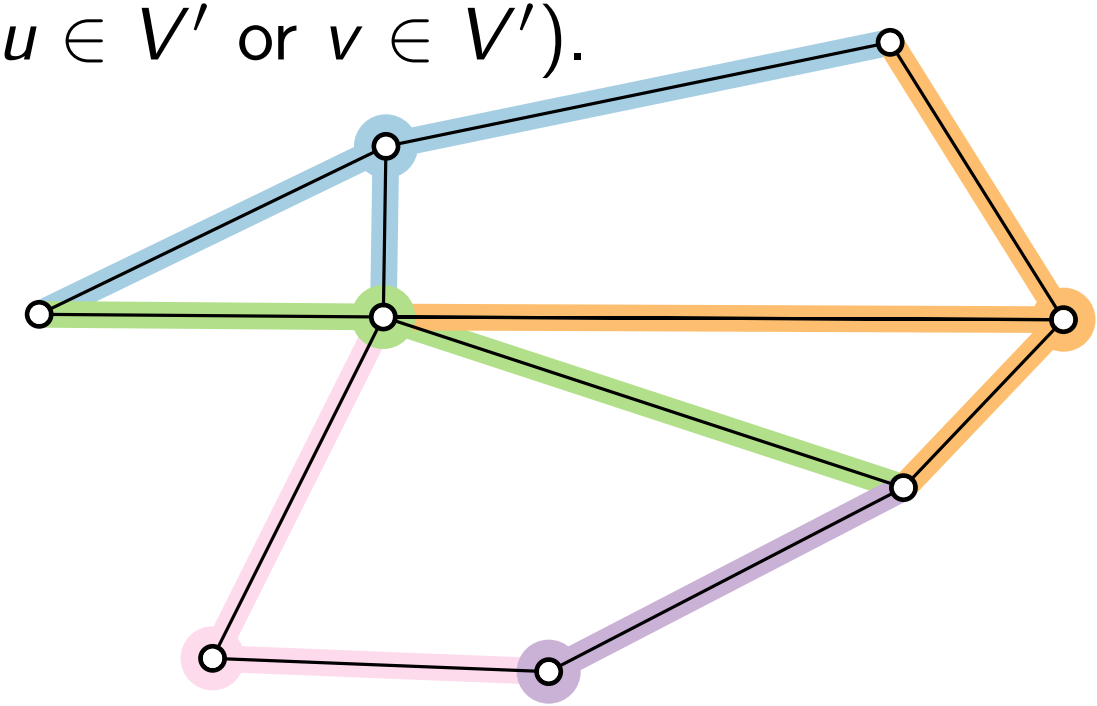


Optimum (OPT $= 4$) – but in general NP-hard to find :-(

VertexCover (card.)

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

Output: a minimum **vertex cover**, that is,
a minimum-cardinality vertex set $V' \subseteq V$ such that every edge is **covered** (i.e., for every $uv \in E$, it holds that $u \in V'$ or $v \in V'$).



"good" (5/4-) approximate solution

Approximation Algorithms

Lecture 1:
Introduction and Vertex Cover

Part III:
NP-Optimization Problem

Alexander Wolff

Winter 2022/23

NP-Optimization Problem

An **NP-optimization problem** Π is given by:

NP-Optimization Problem

An **NP-optimization problem** Π is given by:

- A set D_Π of **instances**.
  We denote the size of an instance $I \in D_\Pi$ by $|I|$.

NP-Optimization Problem

An **NP-optimization problem** Π is given by:

- A set D_Π of **instances**.
  We denote the size of an instance $I \in D_\Pi$ by $|I|$.
- For each instance $I \in D_\Pi$,
  a set $S_\Pi(I) \neq \emptyset$ of **feasible solutions** for I such that:

NP-Optimization Problem

An **NP-optimization problem** Π is given by:

- A set D_Π of **instances**.
  We denote the size of an instance $I \in D_\Pi$ by $|I|$.
- For each instance $I \in D_\Pi$,
  a set $S_\Pi(I) \neq \emptyset$ of **feasible solutions** for I such that:
  - for each solution $s \in S_\Pi(I)$,
    its size $|s|$ is polynomially bounded in $|I|$, and

NP-Optimization Problem

An **NP-optimization problem** Π is given by:

- A set D_Π of **instances**.
  We denote the size of an instance $I \in D_\Pi$ by $|I|$.
- For each instance $I \in D_\Pi$,
  a set $S_\Pi(I) \neq \emptyset$ of **feasible solutions** for I such that:
  - for each solution $s \in S_\Pi(I)$,
    its size $|s|$ is polynomially bounded in $|I|$, and
  - there is a polynomial-time algorithm that decides,
    for each pair (s, I), whether $s \in S_\Pi(I)$.

NP-Optimization Problem

An **NP-optimization problem** Π is given by:

- A set D_Π of **instances**.
  We denote the size of an instance $I \in D_\Pi$ by $|I|$.
- For each instance $I \in D_\Pi$,
  a set $S_\Pi(I) \neq \emptyset$ of **feasible solutions** for I such that:
  - for each solution $s \in S_\Pi(I)$,
    its size $|s|$ is polynomially bounded in $|I|$, and
  - there is a polynomial-time algorithm that decides,
    for each pair (s, I), whether $s \in S_\Pi(I)$.
- A polynomial time computable **objective function** obj_Π
  which assigns a positive objective value $\mathrm{obj}_\Pi(I, s) \geq 0$ to
  any given pair (s, I) with $s \in S_\Pi(I)$.

NP-Optimization Problem

An **NP-optimization problem** Π is given by:

- A set D_Π of **instances**.
  We denote the size of an instance $I \in D_\Pi$ by $|I|$.
- For each instance $I \in D_\Pi$,
  a set $S_\Pi(I) \neq \emptyset$ of **feasible solutions** for I such that:
  - for each solution $s \in S_\Pi(I)$,
    its size $|s|$ is polynomially bounded in $|I|$, and
  - there is a polynomial-time algorithm that decides,
    for each pair (s, I), whether $s \in S_\Pi(I)$.
- A polynomial time computable **objective function** obj_Π
  which assigns a positive objective value $\mathrm{obj}_\Pi(I, s) \geq 0$ to
  any given pair (s, I) with $s \in S_\Pi(I)$.
- Π is either a minimization or maximization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$

For $I \in D_\Pi$: $|I| =$

$S_\Pi(I) =$

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?

- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$?

$\text{obj}_\Pi(I, s) =$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$

$S_\Pi(I) =$

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?

- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$?

$\text{obj}_\Pi(I, s) =$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$

$G=(V,E)$ $S_\Pi(I) =$

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?

- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$?

$\text{obj}_\Pi(I, s) =$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$ Number of vertices $|V|$

$G=(V,E)$ $S_\Pi(I) =$

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?

- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$?

$\text{obj}_\Pi(I, s) =$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$ Number of vertices $|V|$

$G=(V,E)$ $S_\Pi(I) =$ Set of all vertex covers of G

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?

- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$?

$\text{obj}_\Pi(I, s) =$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$ Number of vertices $|V|$

$G=(V,E)$ $S_\Pi(I) =$ Set of all vertex covers of G

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?

  $s \subseteq V \Rightarrow |s| \leq |V| = |I|$

- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$?

$\text{obj}_\Pi(I, s) =$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$ Number of vertices $|V|$

$G=(V,E)$ $S_\Pi(I) =$ Set of all vertex covers of G

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?
  $s \subseteq V \Rightarrow |s| \leq |V| = |I|$
- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$? Test whether all edges are covered.

$\text{obj}_\Pi(I, s) =$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$ Number of vertices $|V|$

$G=(V,E)$ $S_\Pi(I) =$ Set of all vertex covers of G

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?
  $s \subseteq V \Rightarrow |s| \leq |V| = |I|$
- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$? Test whether all edges are covered.

$\text{obj}_\Pi(I, s) = |s|$

Π is a m...imization problem.

VertexCover: NP-Optimization Problem

Task: Fill in the gaps for $\Pi =$ Vertex Cover.

$D_\Pi =$ Set of all graphs

For $I \in D_\Pi$: $|I| =$ Number of vertices $|V|$

$G=(V,E)$ $S_\Pi(I) =$ Set of all vertex covers of G

- Why is $|s| \in \text{poly}(|I|)$ for every $s \in S_\Pi(I)$?
  $s \subseteq V \Rightarrow |s| \leq |V| = |I|$
- For a given pair (s, I), how can we efficiently decide whether $s \in S_\Pi(I)$? Test whether all edges are covered.

$\text{obj}_\Pi(I, s) = |s|$

Π is a minimization problem.

Optimum and Optimal Objective Value

Let Π be a minimization problem and $I \in D_\Pi$ an instance of Π.

Optimum and Optimal Objective Value

Let Π be a minimization problem and $I \in D_\Pi$ an instance of Π.

A feasible solution $s^* \in S_\Pi(I)$ is **optimal** if

$\text{obj}_\Pi(I, s^*)$ is minimal among the objective values

attained by the feasible solutions of I.

Optimum and Optimal Objective Value

Let Π be a maximization problem / minimization problem and $I \in D_\Pi$ an instance of Π.

A feasible solution $s^* \in S_\Pi(I)$ is **optimal** if

$\text{obj}_\Pi(I, s^*)$ is maximal / minimal among the objective values

attained by the feasible solutions of I.

Optimum and Optimal Objective Value

Let Π be a maximization problem minimization problem and $I \in D_\Pi$ an instance of Π.

A feasible solution $s^* \in S_\Pi(I)$ is **optimal** if

$\text{obj}_\Pi(I, s^*)$ is maximal minimal among the objective values

attained by the feasible solutions of I.

The optimal value $\text{obj}_\Pi(I, s^*)$ of the objective function is denoted by $\text{OPT}_\Pi(I)$ or simply by OPT in context.

Approximation Algorithms

Let Π be a minimization problem and $\alpha \in \mathbb{Q}^+$.

Approximation Algorithms

Let Π be a minimization problem and $\alpha \in \mathbb{Q}^+$.

A factor-α approximation algorithm for Π is an efficient algorithm that provides, for **any** instance $I \in D_\Pi$,
a feasible solution $s \in S_\Pi(I)$ such that

Approximation Algorithms

Let Π be a minimization problem and $\alpha \in \mathbb{Q}^+$.

A factor-α approximation algorithm for Π is an efficient algorithm that provides, for **any** instance $I \in D_\Pi$, a feasible solution $s \in S_\Pi(I)$ such that

$$\frac{\text{obj}_\Pi(I, s)}{\text{OPT}_\Pi(I)}$$

Approximation Algorithms

Let Π be a minimization problem and $\alpha \in \mathbb{Q}^+$.

A factor-α approximation algorithm for Π is an efficient algorithm that provides, for **any** instance $I \in D_\Pi$, a feasible solution $s \in S_\Pi(I)$ such that

$$\frac{\mathrm{obj}_\Pi(I, s)}{\mathrm{OPT}_\Pi(I)} \leq \alpha.$$

Approximation Algorithms

Let Π be a minimization problem and ~~$\alpha \in \mathbb{Q}^+$~~ $\alpha: \mathbb{N} \to \mathbb{Q}$.

A factor-α approximation algorithm for Π is an efficient algorithm that provides, for **any** instance $I \in D_\Pi$, a feasible solution $s \in S_\Pi(I)$ such that

$$\frac{\mathrm{obj}_\Pi(I, s)}{\mathrm{OPT}_\Pi(I)} \leq \alpha(|I|).$$

Approximation Algorithms

maximization problem $\alpha: \mathbb{N} \to \mathbb{Q}$

Let Π be a minimization problem and ~~$\alpha \in \mathbb{Q}^+$~~.

A factor-α approximation algorithm for Π is an efficient algorithm that provides, for **any** instance $I \in D_\Pi$, a feasible solution $s \in S_\Pi(I)$ such that

$$\frac{\text{obj}_\Pi(I, s)}{\text{OPT}_\Pi(I)} \begin{matrix} \geq \\ = \\ \leq \end{matrix} \not{\alpha}. \quad \alpha(|I|)$$

Approximation Algorithms

Lecture 1:
Introduction and Vertex Cover

Part IV:
Approximation Algorithm for VertexCover

Alexander Wolff

Winter 2022/23

Approximation Alg. for VertexCover

Ideas?

Approximation Alg. for VertexCover

Ideas?

- Edge-Greedy

Approximation Alg. for VertexCover

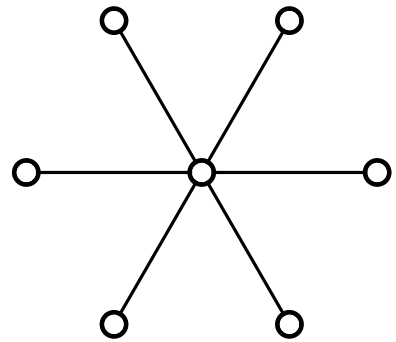
Ideas?

- Edge-Greedy
- Vertex-Greedy

Approximation Alg. for VertexCover

Ideas?

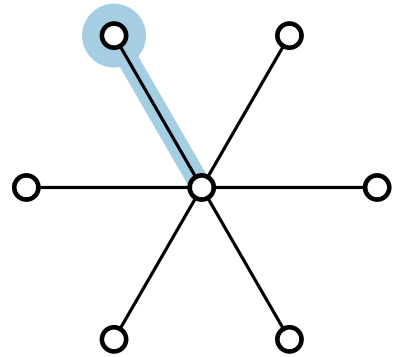
- Edge-Greedy
- Vertex-Greedy



Approximation Alg. for VertexCover

Ideas?

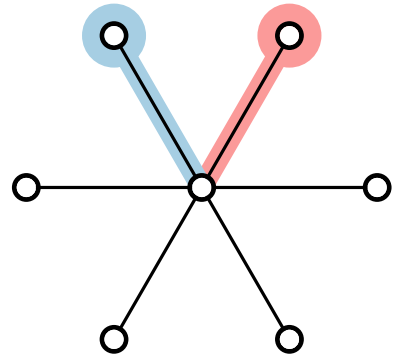
- Edge-Greedy
- Vertex-Greedy



Approximation Alg. for VertexCover

Ideas?

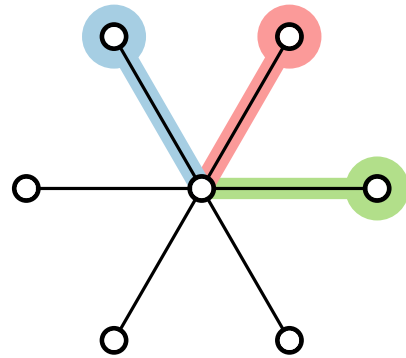
- Edge-Greedy
- Vertex-Greedy



Approximation Alg. for VertexCover

Ideas?

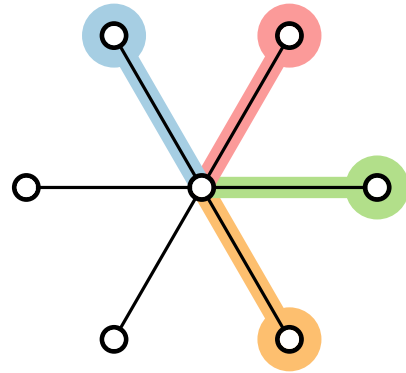
- Edge-Greedy
- Vertex-Greedy



Approximation Alg. for VertexCover

Ideas?

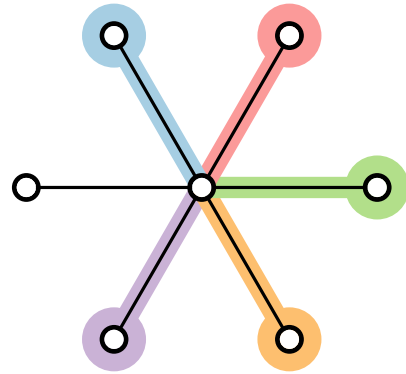
- Edge-Greedy
- Vertex-Greedy



Approximation Alg. for VertexCover

Ideas?

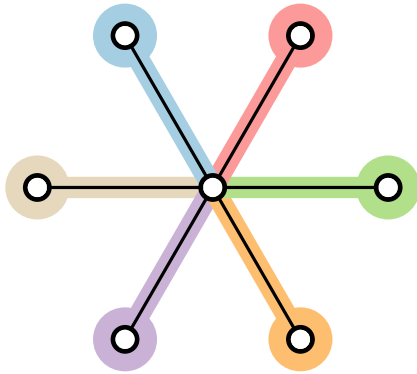
- Edge-Greedy
- Vertex-Greedy



Approximation Alg. for VertexCover

Ideas?

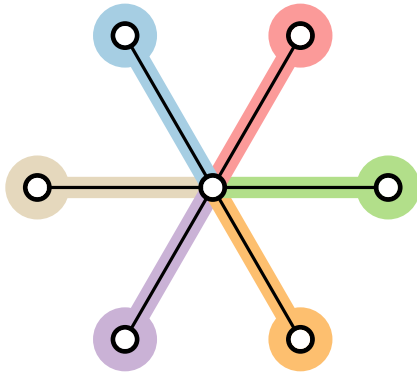
- Edge-Greedy
- Vertex-Greedy



Approximation Alg. for VertexCover

Ideas?

- Edge-Greedy
- Vertex-Greedy

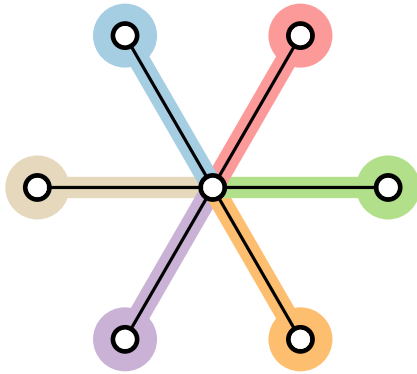


Quality?

Approximation Alg. for VertexCover

Ideas?

- Edge-Greedy
- Vertex-Greedy



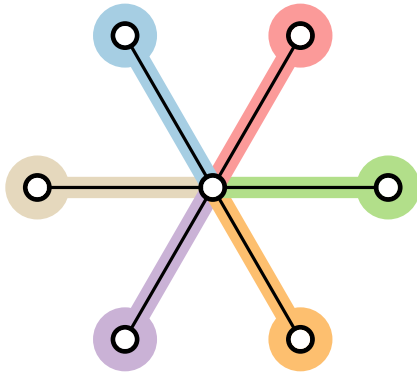
Quality?

Problem: How can we estimate $\text{obj}_\Pi(I, s)/\text{OPT}$, when it is hard to compute OPT?

Approximation Alg. for VertexCover

Ideas?

- Edge-Greedy
- Vertex-Greedy



Quality?

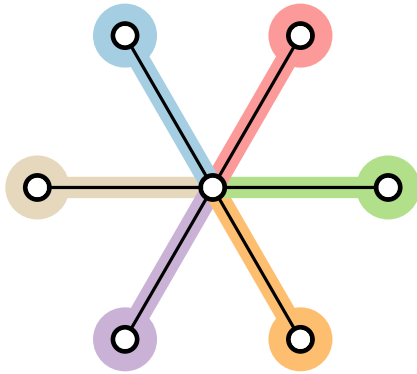
Problem: How can we estimate $\text{obj}_\Pi(I, s)/\text{OPT}$, when it is hard to compute OPT?

Idea: Find a "good" lower bound $L \leq \text{OPT}$ for OPT and compare it to our approximate solution.

Approximation Alg. for VertexCover

Ideas?

- Edge-Greedy
- Vertex-Greedy



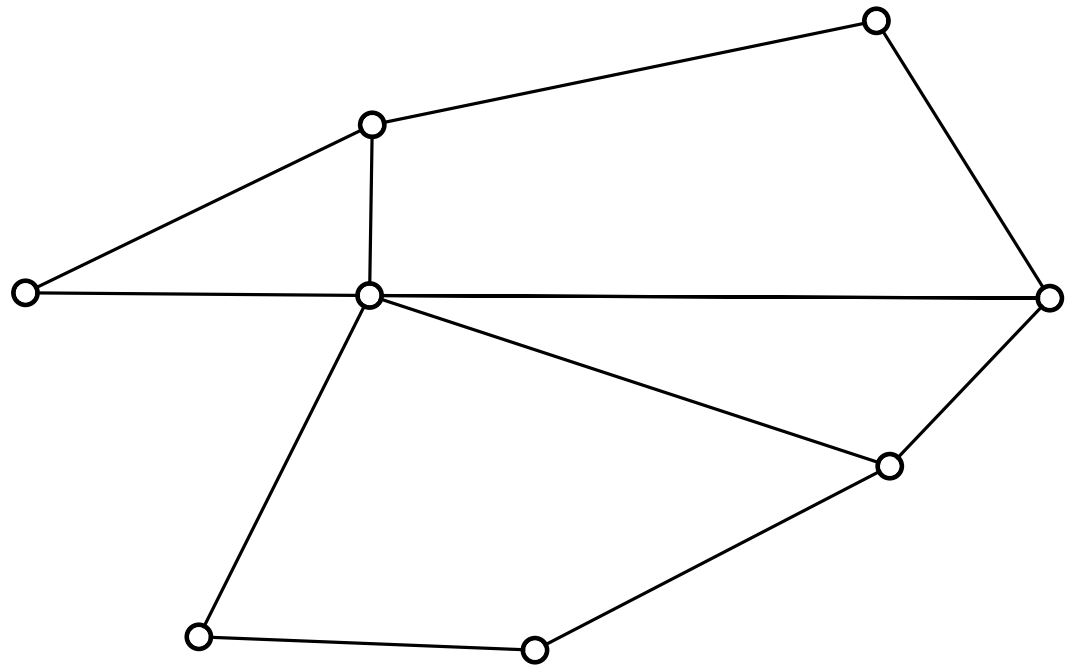
Quality?

Problem: How can we estimate $\text{obj}_\Pi(I, s)/\text{OPT}$, when it is hard to compute OPT?

Idea: Find a "good" lower bound $L \leq \text{OPT}$ for OPT and compare it to our approximate solution.

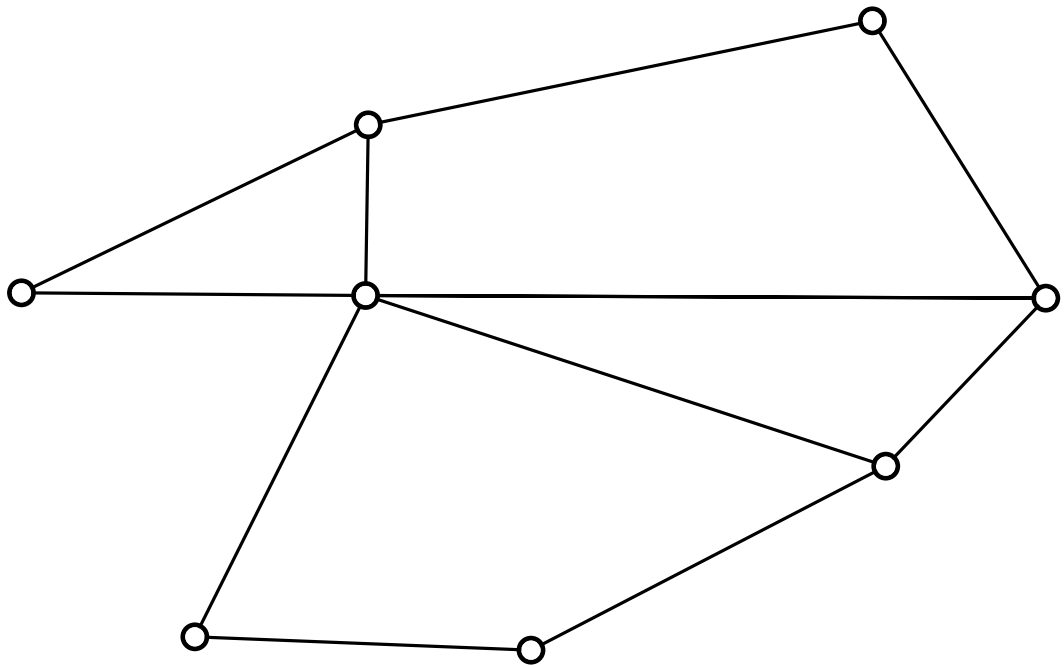
$$\frac{\text{obj}_\Pi(I, s)}{\text{OPT}} \leq \frac{\text{obj}_\Pi(I, s)}{L}$$

Lower Bound by Matchings



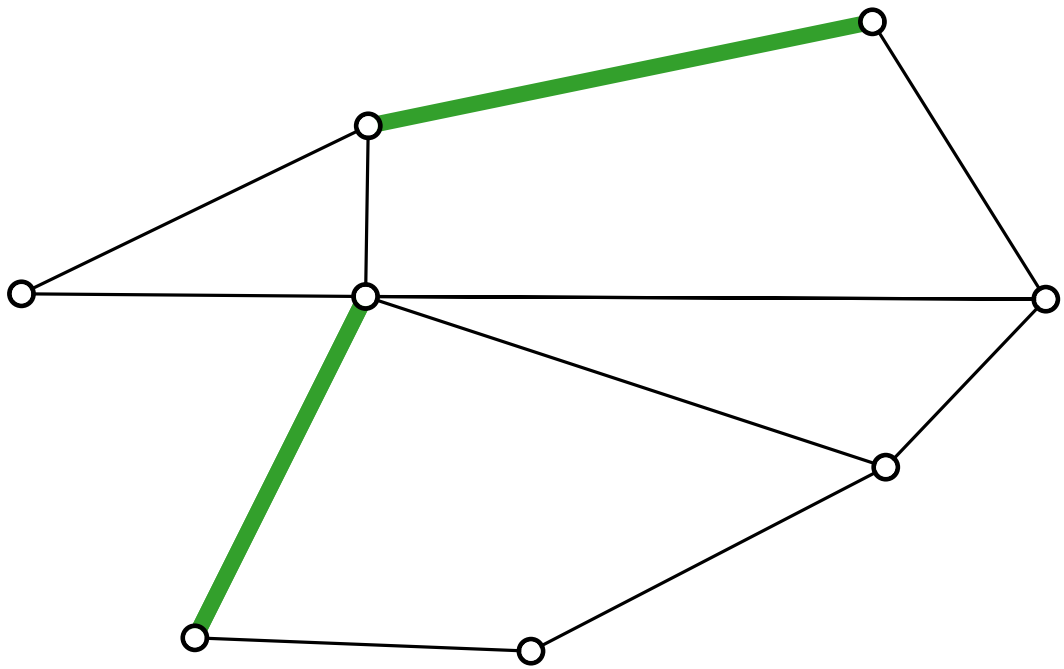
Lower Bound by Matchings

An edge set $M \subseteq E$ of a graph $G = (V, E)$ is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).



Lower Bound by Matchings

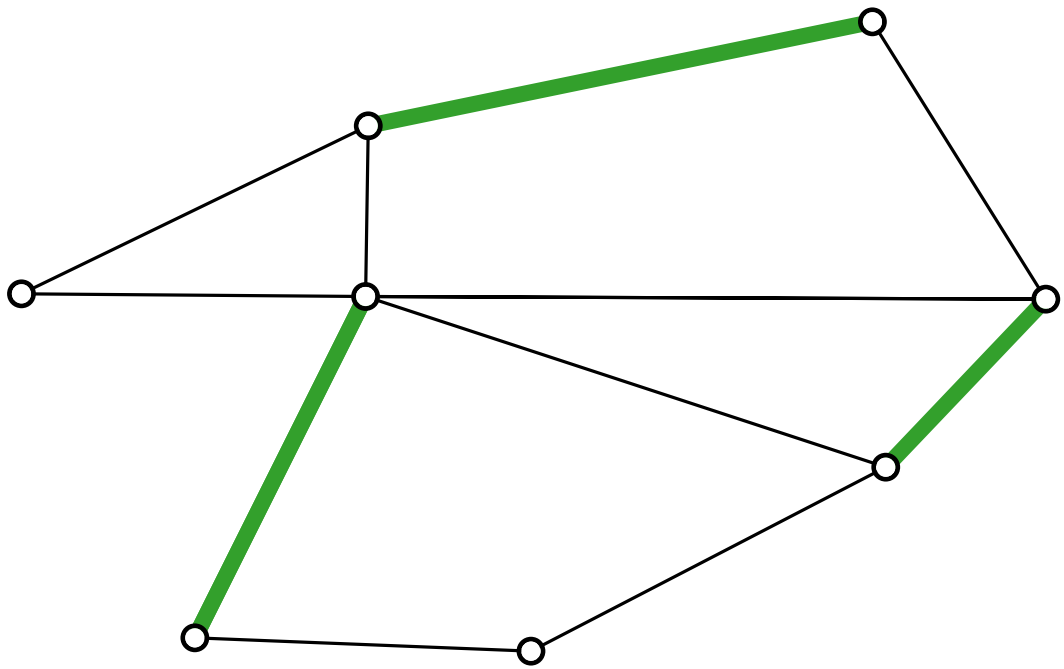
An edge set $M \subseteq E$ of a graph $G = (V, E)$ is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).



Lower Bound by Matchings

An edge set $M \subseteq E$ of a graph $G = (V, E)$ is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.

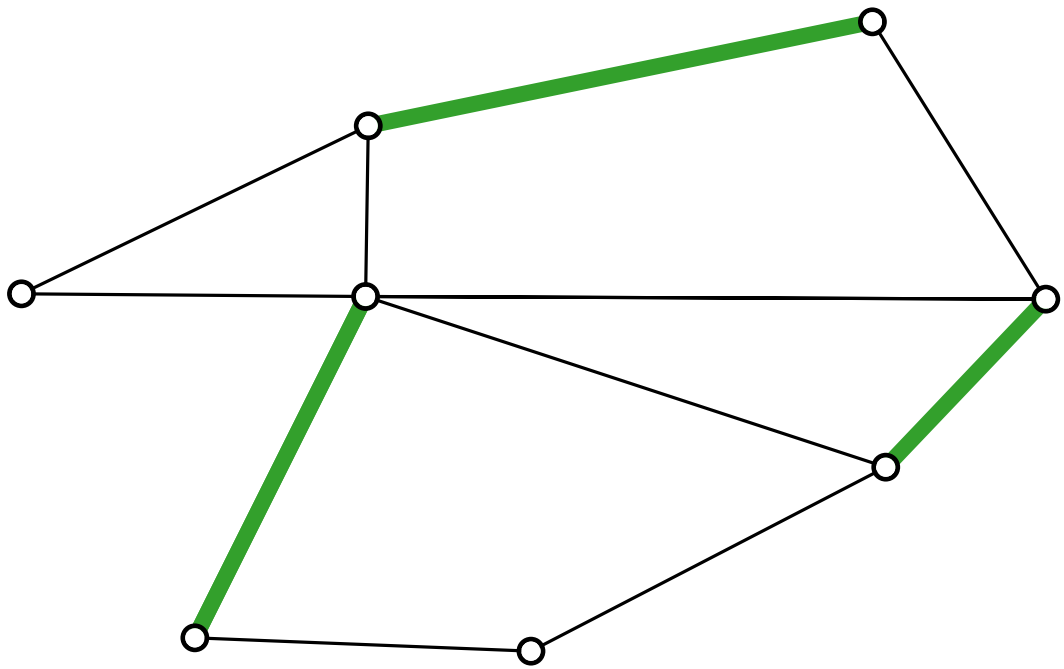


Lower Bound by Matchings

An edge set $M \subseteq E$ of a graph $G = (V, E)$ is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.

$\mathrm{OPT} \geq$



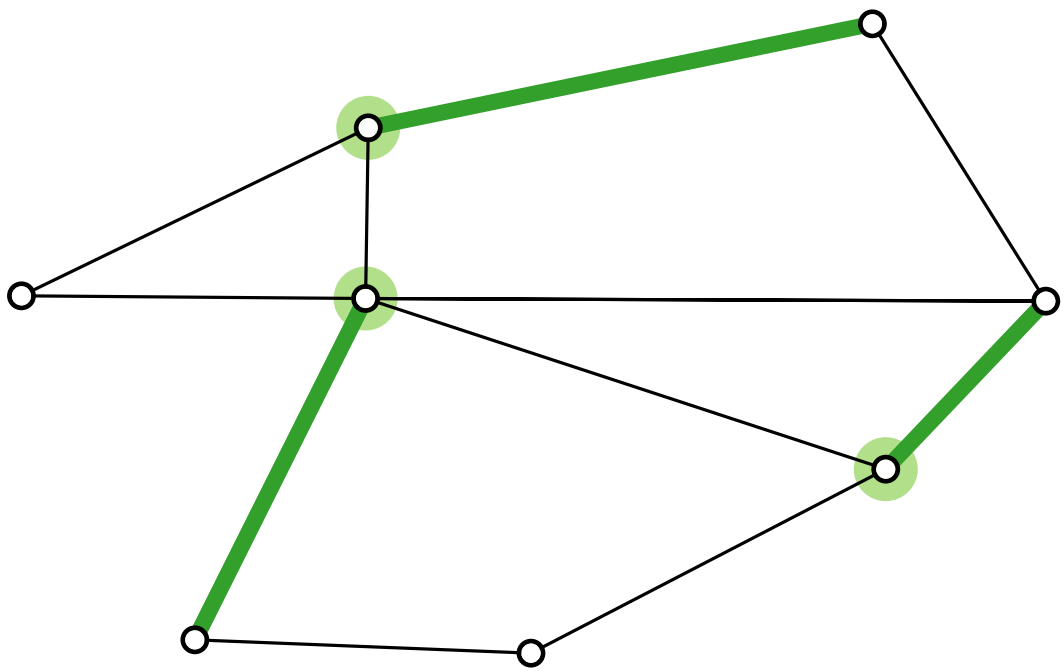
Lower Bound by Matchings

An edge set $M \subseteq E$ of a graph $G = (V, E)$ is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.

$\mathrm{OPT} \geq$

Vertex cover of M



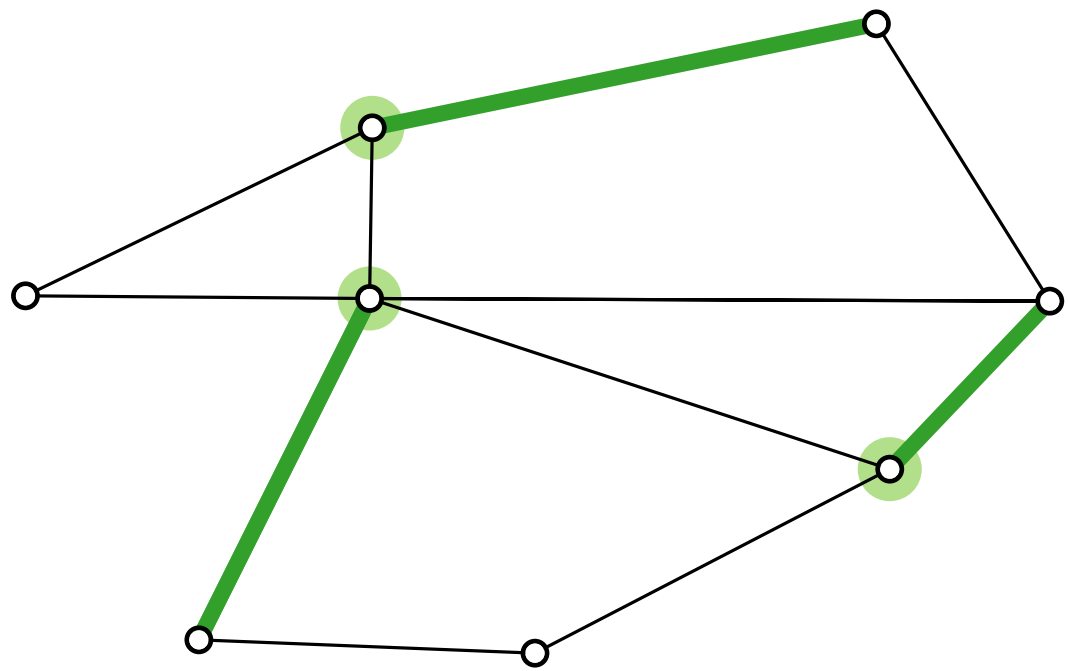
Lower Bound by Matchings

An edge set $M \subseteq E$ of a graph $G = (V, E)$ is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.

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

Vertex cover of M



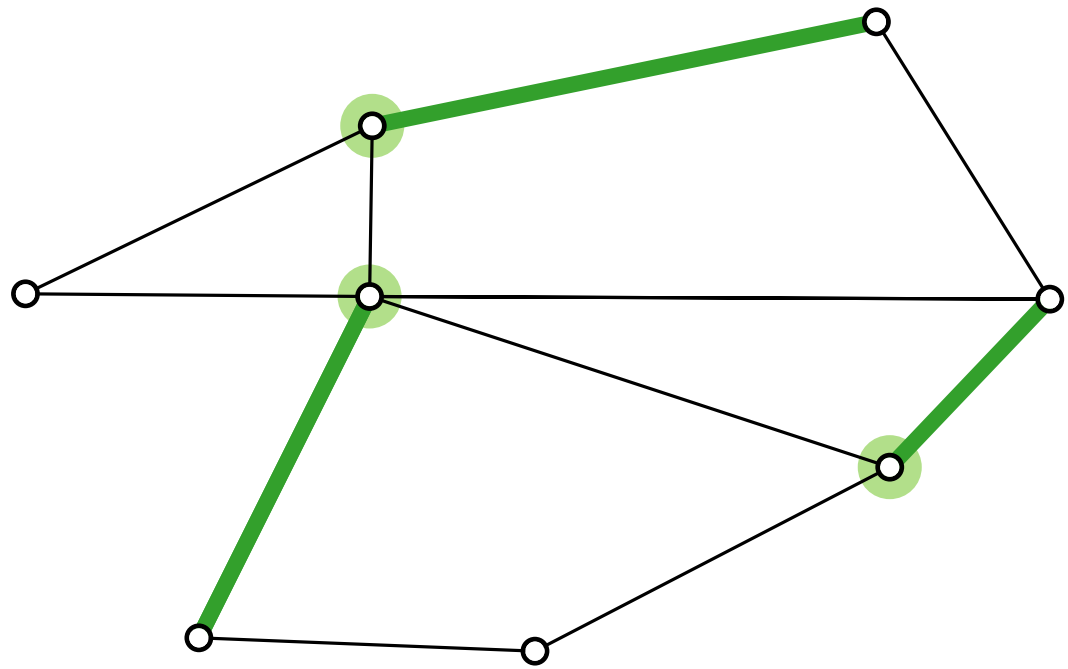
Lower Bound by Matchings

Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.

$\mathrm{OPT} \geq |M|$
$\mathrm{OPT} = |M|$?

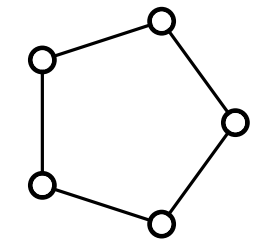
Vertex cover of M



Lower Bound by Matchings

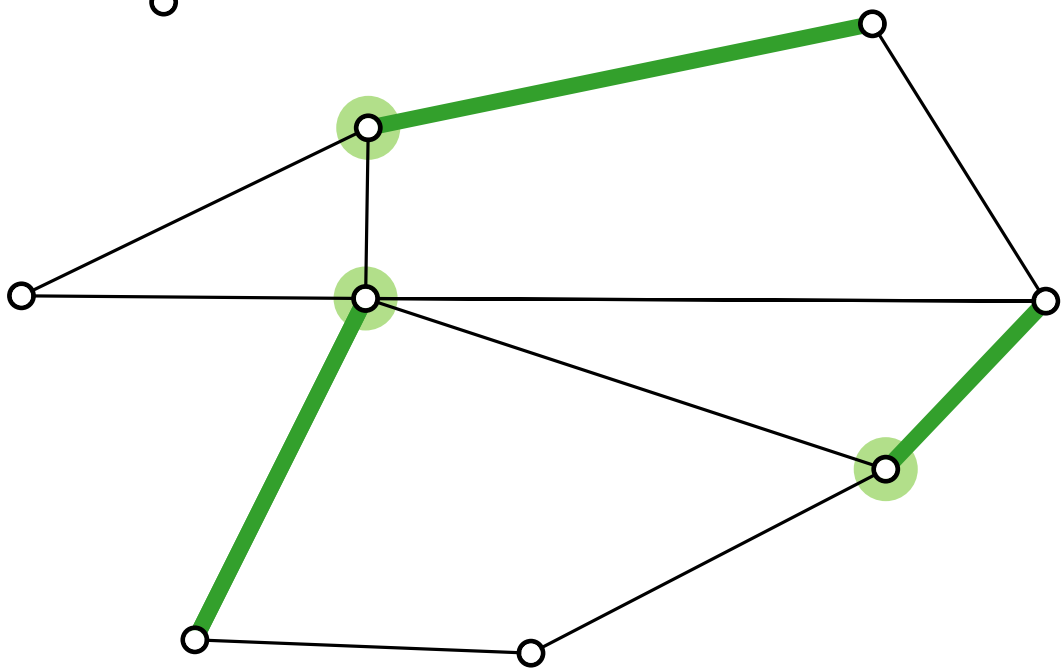
Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.



$\mathrm{OPT} \geq |M|$
$\mathrm{OPT} = |M|$?

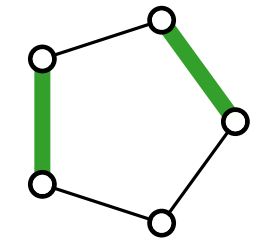
Vertex cover of M



Lower Bound by Matchings

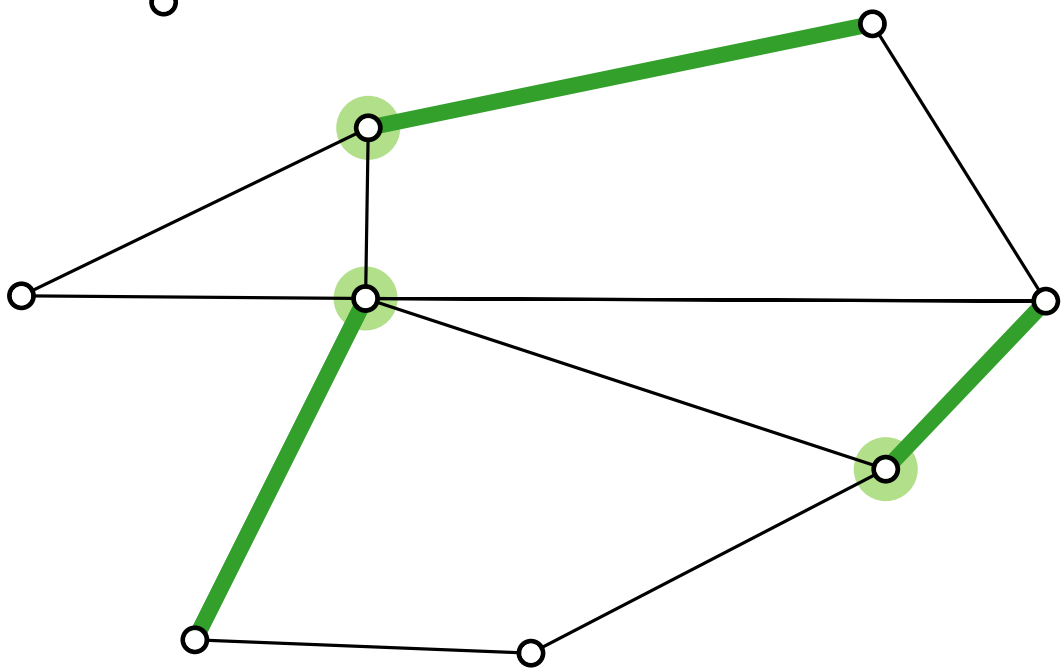
Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.



$\mathrm{OPT} \geq |M|$
$\mathrm{OPT} = |M|$?

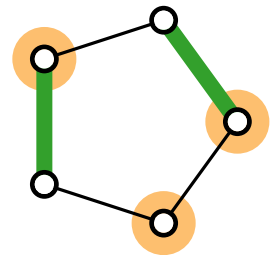
Vertex cover of M



Lower Bound by Matchings

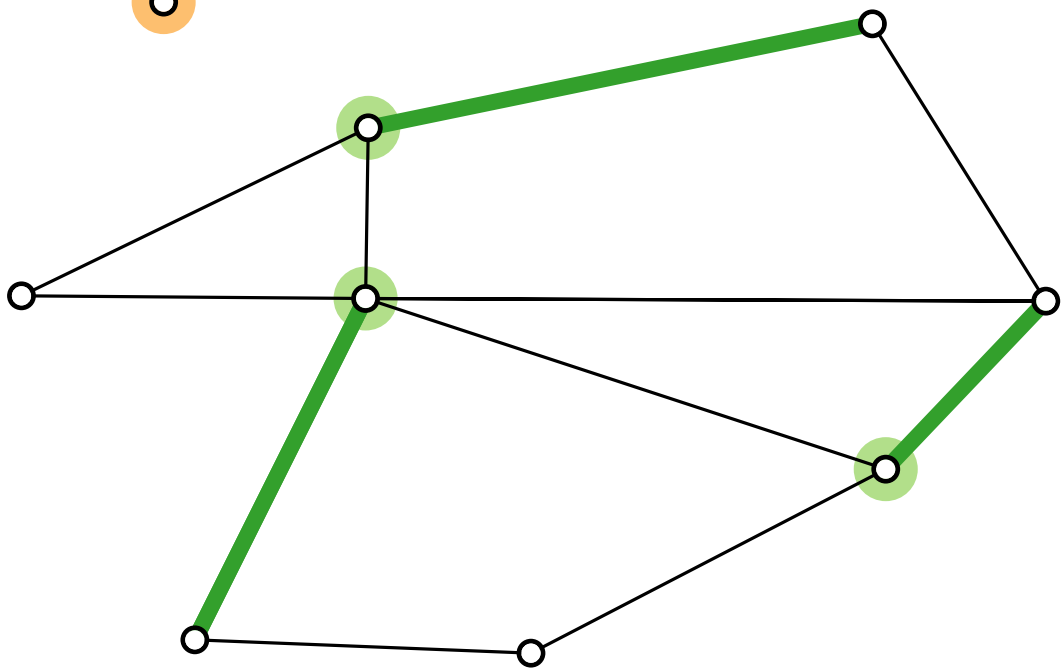
Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.



$\mathrm{OPT} \geq |M|$
$\mathrm{OPT} = |M|$?

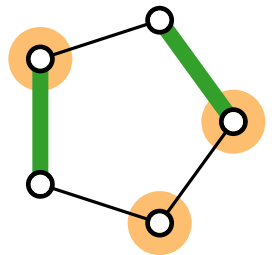
Vertex cover of M



Lower Bound by Matchings

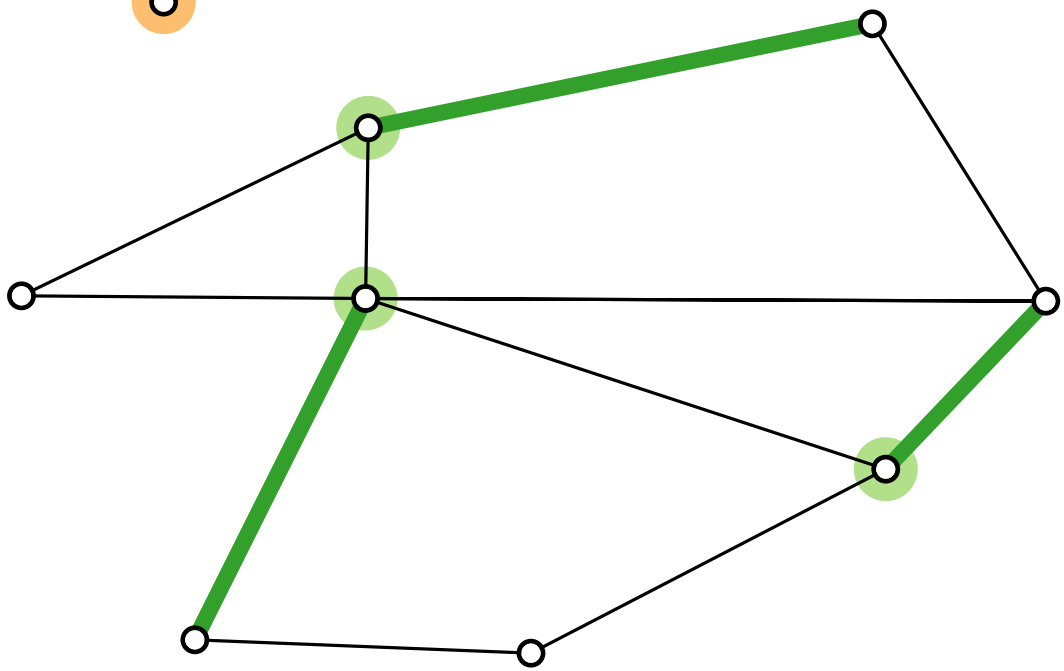
Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.



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

~~$\mathrm{OPT} = |M|$~~ ?



Vertex cover of M

Lower Bound by Matchings

Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

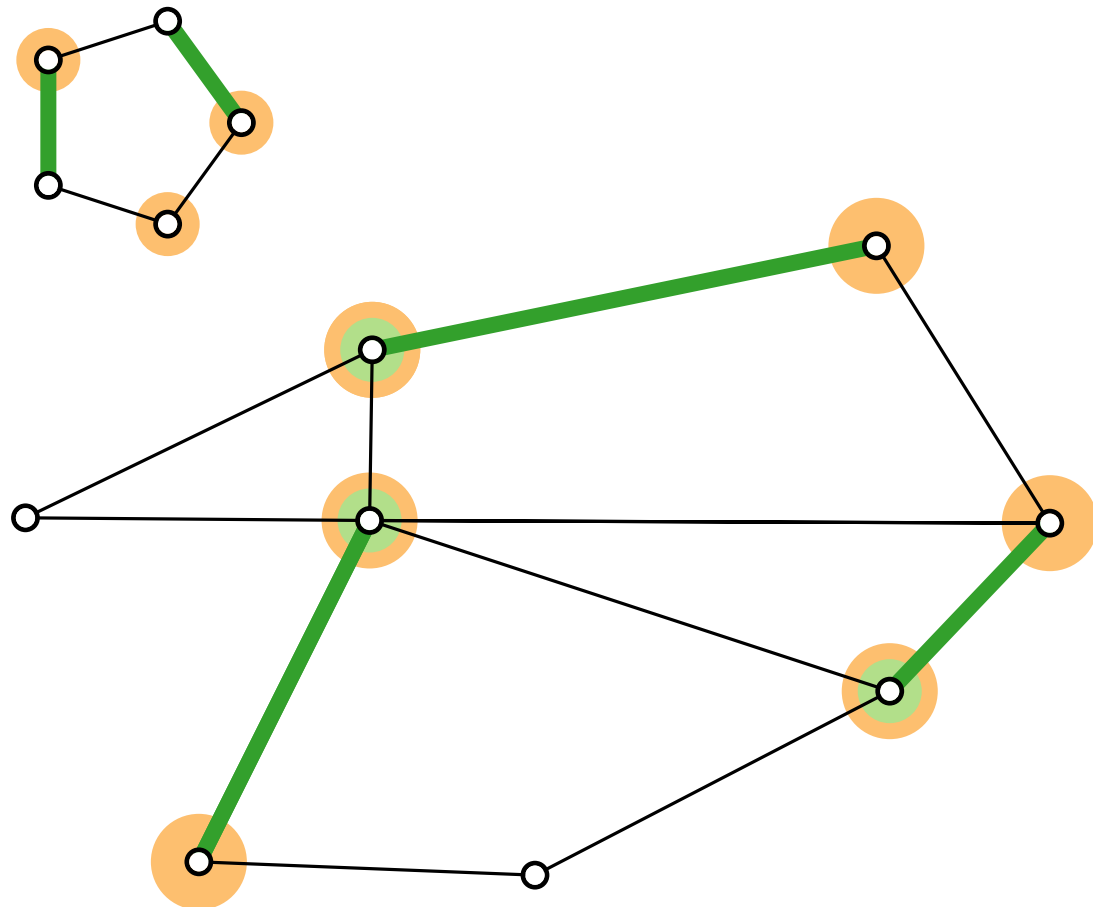
M is **maximal** if there is no matching M' with $M' \supsetneq M$.

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

~~$\mathrm{OPT} = |M|$~~ ?

Vertex cover of M

Vertex cover of E



Lower Bound by Matchings

Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

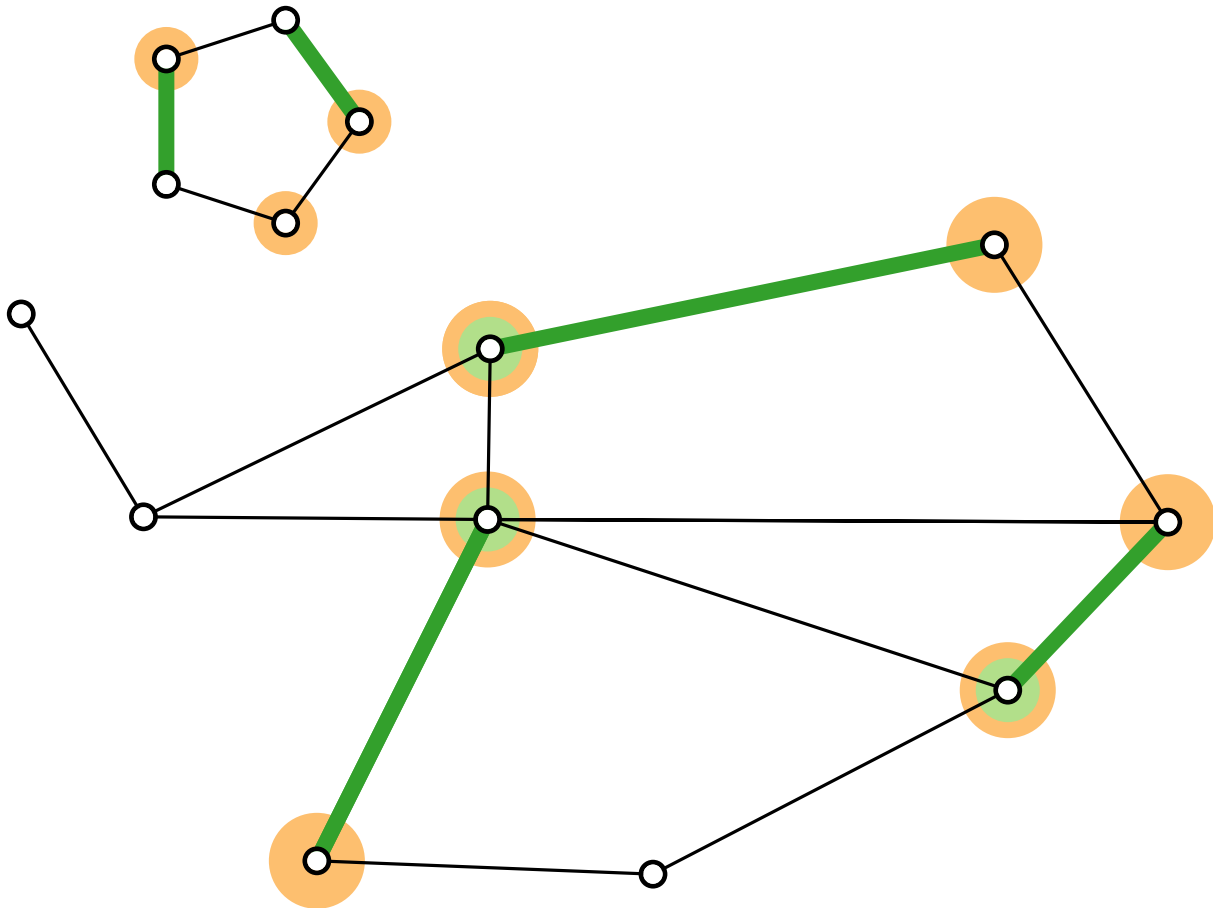
M is **maximal** if there is no matching M' with $M' \supsetneq M$.

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

~~$\mathrm{OPT} = |M|$~~ ?

Vertex cover of M

Vertex cover of E



Lower Bound by Matchings

Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

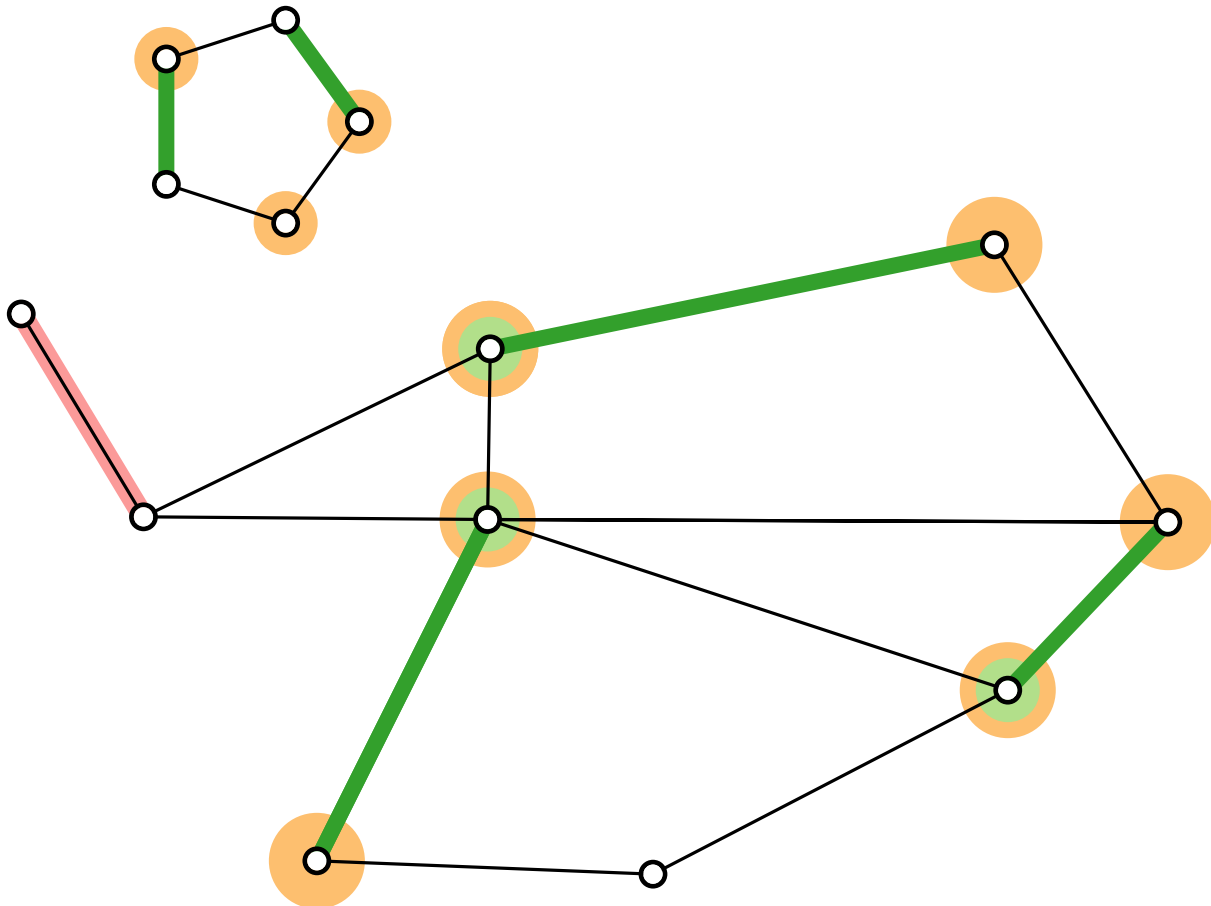
M is **maximal** if there is no matching M' with $M' \supsetneq M$.

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

~~$\mathrm{OPT} = |M|$?~~

Vertex cover of M

Vertex cover of E



Lower Bound by Matchings

Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

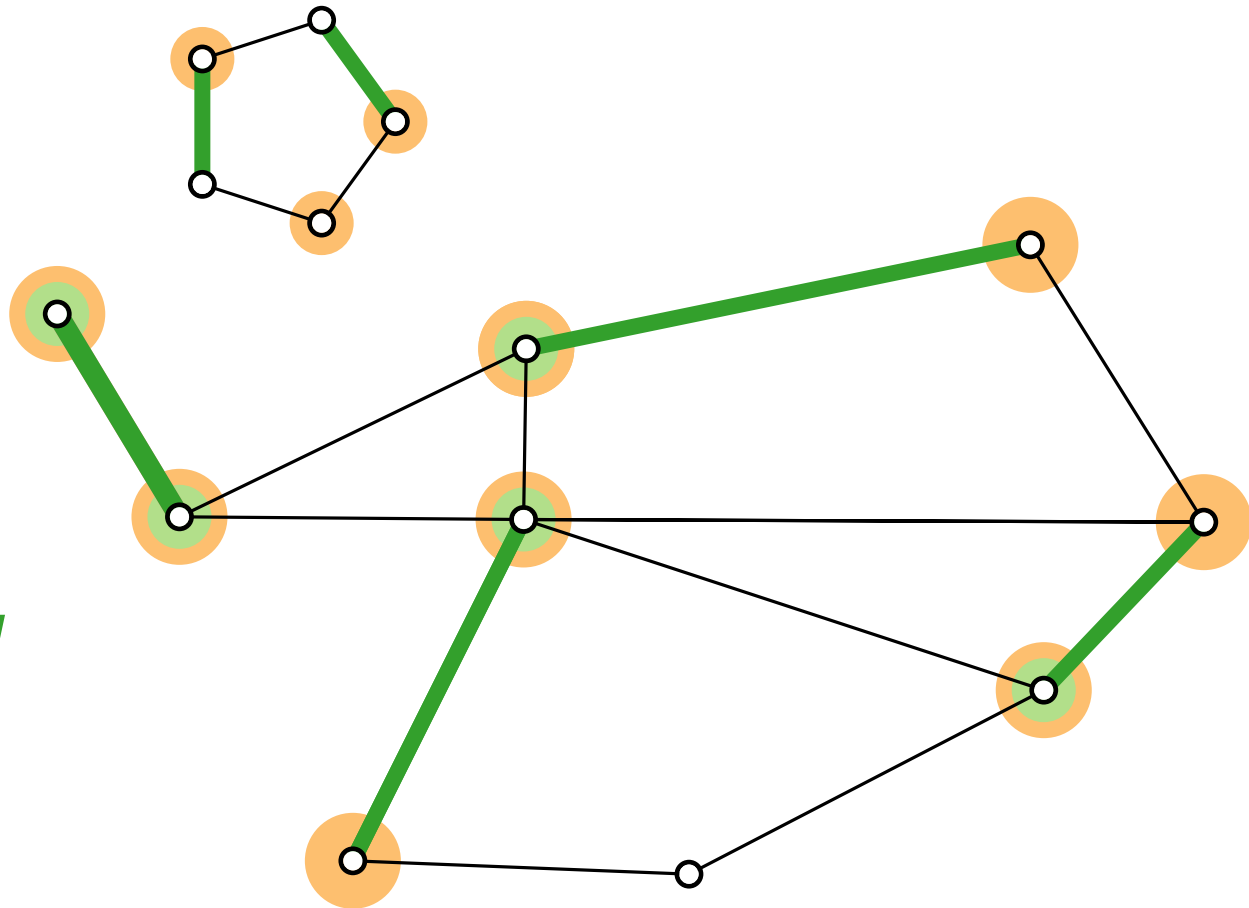
M is **maximal** if there is no matching M' with $M' \supsetneq M$.

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

~~$\mathrm{OPT} = |M|$?~~

Vertex cover of M

Vertex cover of E



Lower Bound by Matchings

Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.

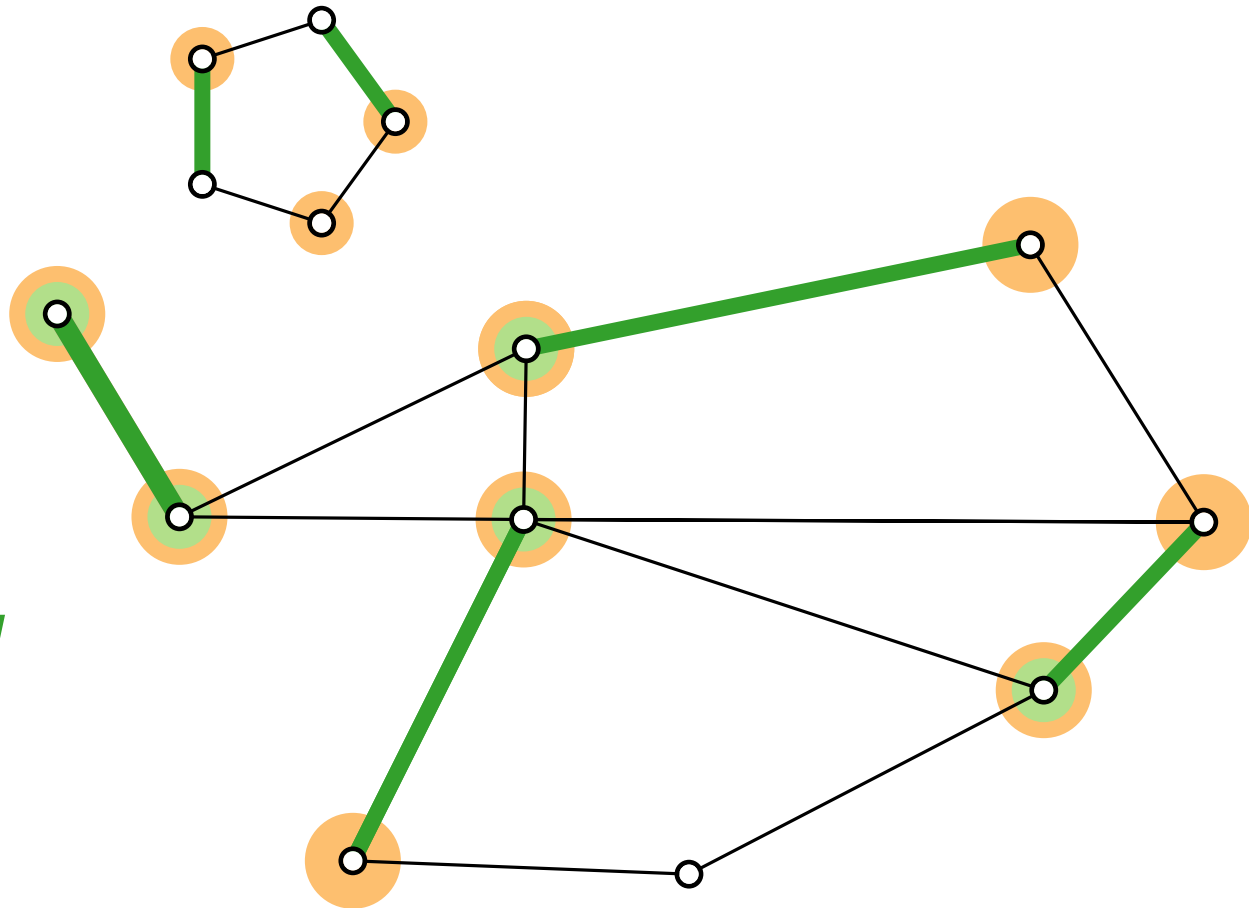
$\mathrm{OPT} \geq |M|$

~~$\mathrm{OPT} = |M|$?~~

Vertex cover of M

Vertex cover of E

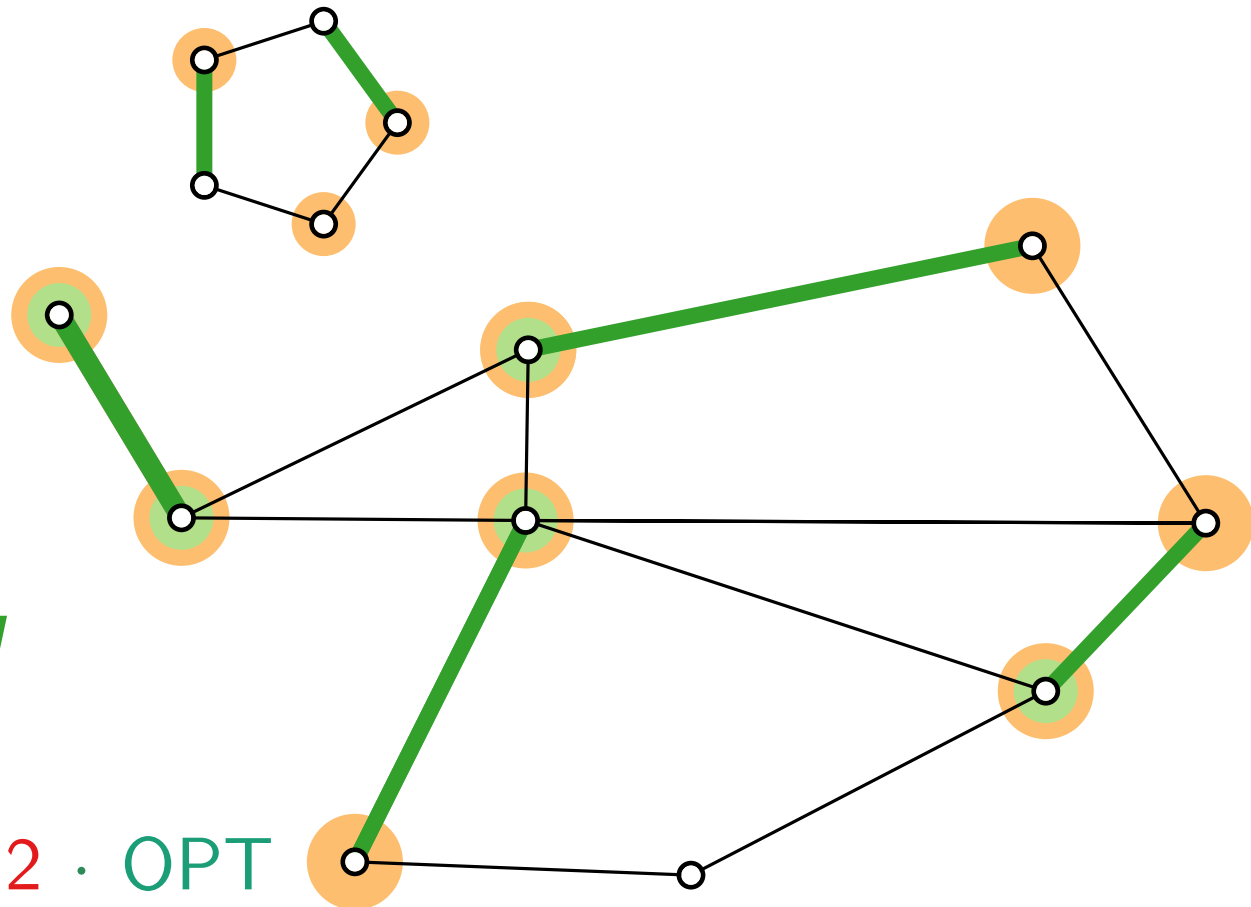
$\mathrm{ALG} = 2 \cdot |M| \leq$



Lower Bound by Matchings

Given a graph G, a set M of edges of G is a **matching** if no two edges of M are adjacent (i.e., share an end vertex).

M is **maximal** if there is no matching M' with $M' \supsetneq M$.



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

~~$\mathrm{OPT} = |M|$?~~

Vertex cover of M

Vertex cover of E

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

Approximation Alg. for VertexCover

Algorithm VertexCover(G)

$M \leftarrow \emptyset$

Approximation Alg. for VertexCover

Algorithm VertexCover(G)

$M \leftarrow \emptyset$

foreach $e \in E(G)$ **do**

Approximation Alg. for VertexCover

Algorithm VertexCover(G)

$M \leftarrow \emptyset$

foreach $e \in E(G)$ **do**

  if e is not adjacent to any edge in M **then**

Approximation Alg. for VertexCover

Algorithm VertexCover(G)

$M \leftarrow \emptyset$

foreach $e \in E(G)$ **do**

  if e is not adjacent to any edge in M **then**

    $M \leftarrow M \cup \{e\}$

Approximation Alg. for VertexCover

Algorithm VertexCover(G)

$M \leftarrow \emptyset$

foreach $e \in E(G)$ **do**

  if e is not adjacent to any edge in M **then**

    $M \leftarrow M \cup \{e\}$

return $\{ u, v \mid uv \in M \}$

Approximation Alg. for VertexCover

Algorithm VertexCover(G)

$M \leftarrow \emptyset$

foreach $e \in E(G)$ **do**

  if e is not adjacent to any edge in M **then**

    $M \leftarrow M \cup \{e\}$

return $\{ u, v \mid uv \in M \}$

Theorem. The above algorithm is a factor-2 approximation algorithm for VertexCover.

Approximability of Vertex Cover

The best known approximation factor for VertexCover is

Approximability of Vertex Cover

The best known approximation factor for VertexCover is

$$2 - \Theta(1/\sqrt{\log n}).$$

Approximability of Vertex Cover

The best known approximation factor for VertexCover is

$$2 - \Theta(1/\sqrt{\log n}).$$

If P $\neq$ NP, VertexCover cannot be approximated within a factor of 1.3606.

Approximability of Vertex Cover

The best known approximation factor for VertexCover is

$$2 - \Theta(1/\sqrt{\log n}).$$

If P $\neq$ NP, VertexCover cannot be approximated within a factor of 1.3606.

VertexCover cannot be approximated within a factor of $2 - \Theta(1)$ – if the *Unique Games Conjecture* holds.