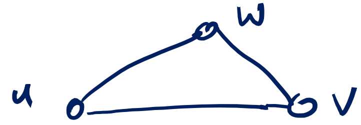
# Chapter 8
# Approximation Algorithms

## Algorithm Theory
## WS 2019/20

Fabian Kuhn

# Metric TSP




**Input:**

- Set $V$ of $n$ nodes (points, cities, locations, sites)
- Distance function $d: V \times V \to \mathbb{R}$, i.e., $d(u,v)$ is dist from $u$ to $v$
- Distances define a metric on $V$:

$$d(u,v) = d(v,u) \geq 0, \qquad d(u,v) = 0 \Leftrightarrow u = v$$
$$\forall u,v,w \in V : d(u,v) \leq d(u,w) + d(w,v)$$ triangle ineq.

**Solution:**

- Ordering/permutation $v_1, v_2, \dots, v_n$ of the vertices
- Length of TSP path: $\sum_{i=1}^{n-1} d(v_i, v_{i+1})$
- Length of TSP tour: $d(v_1, v_n) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$

**Goal:**

- Minimize length of TSP path or TSP tour

# Metric TSP

- The problem is NP-hard
- We have seen that the greedy algorithm (always going to the nearest unvisited node) gives an $O(\log n)$-approximation
- Can we get a constant approximation ratio?
- We will see that we can…

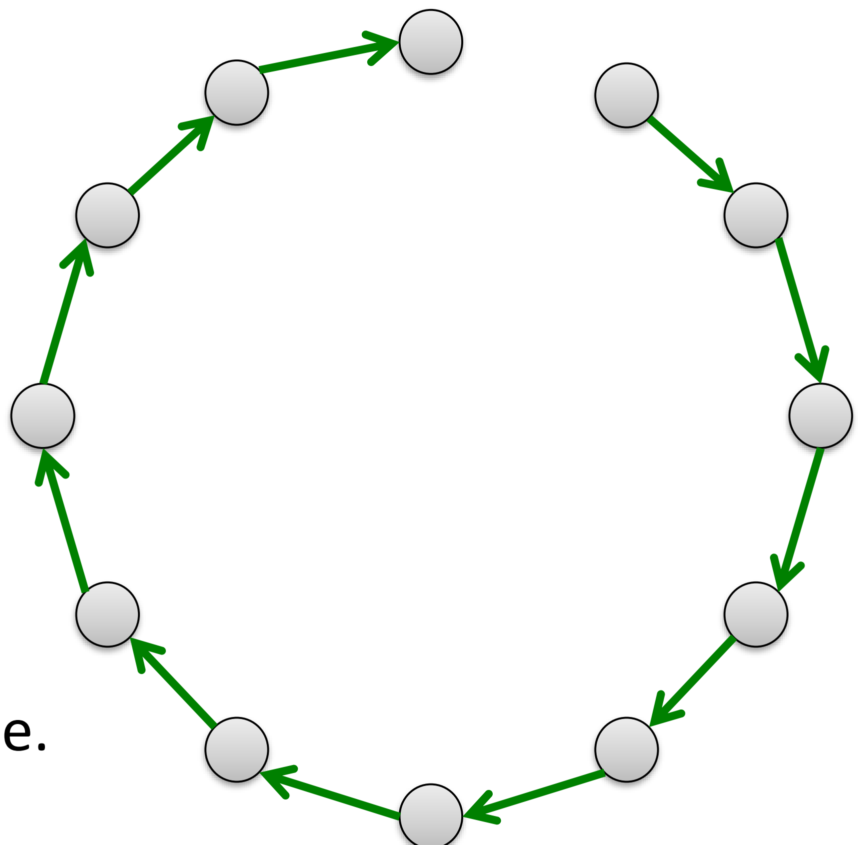
# TSP and MST

**Claim:** The length of an optimal TSP path is lower bounded by the weight of a minimum spanning tree

**Proof:**

- A TSP path is a spanning tree, it's length is the weight of the tree

$w(MST) \leq TSP_{PATH} \leq TSP_{TOUR}$

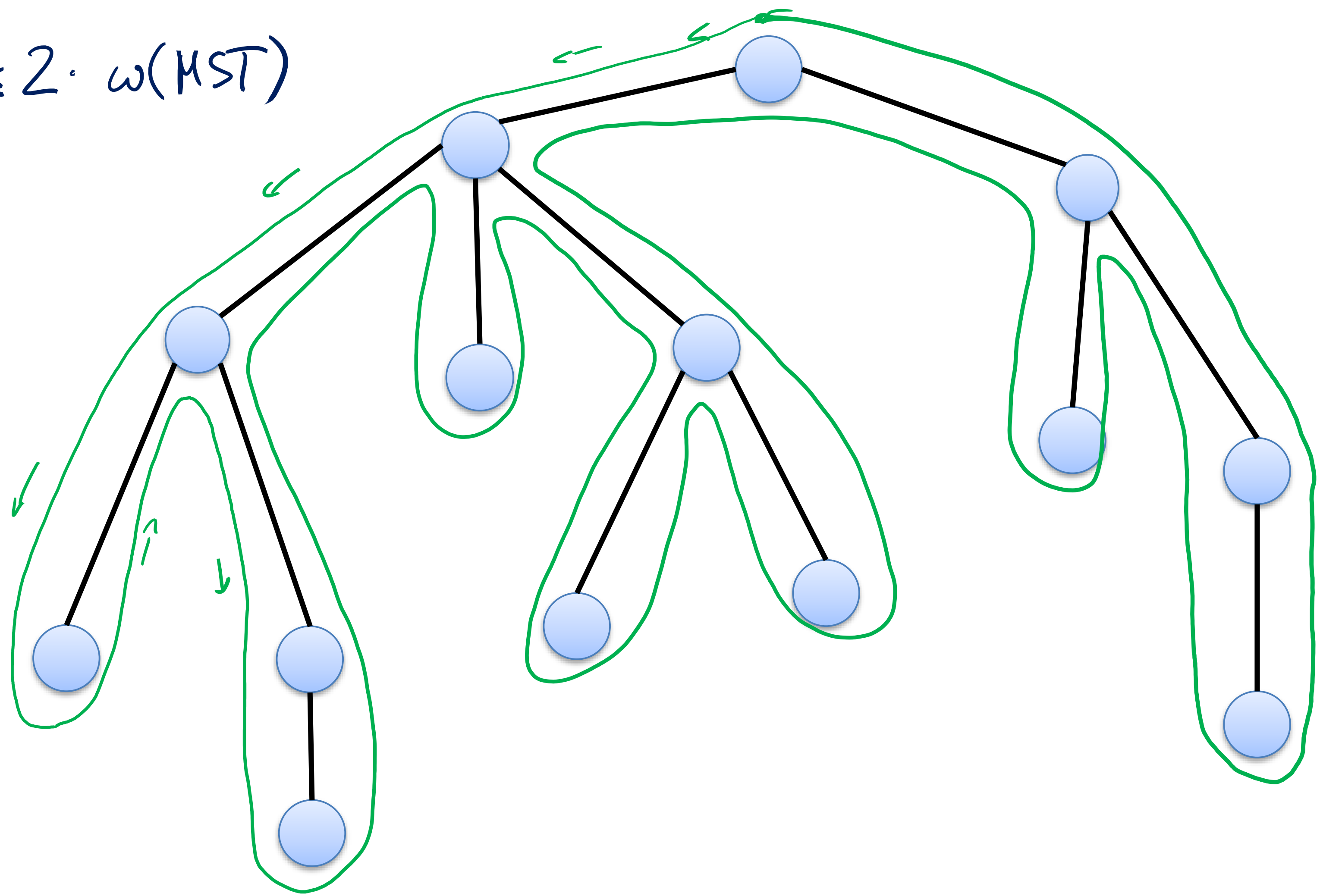


**Corollary:** Since an optimal TSP tour is longer than an optimal TSP path, the length of an optimal TSP tour is also lower bounded by the weight of a minimum spanning tree.

# The MST Tour

Walk around the MST…

length of walk $\leq 2 \cdot w(MST)$

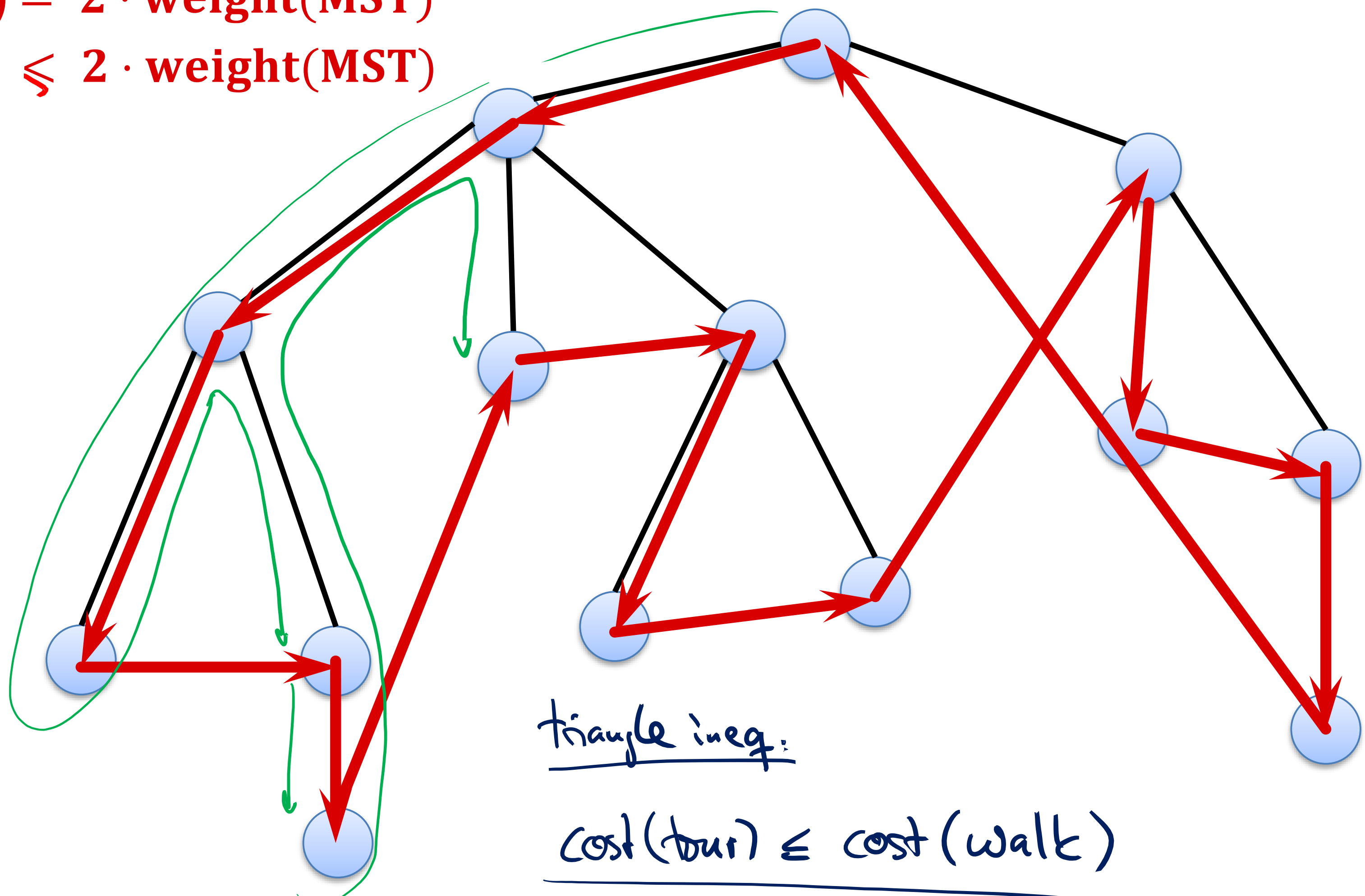


# The MST Tour

Walk around the MST…

**Cost (walk)** $= 2 \cdot \textbf{weight(MST)}$

**Cost (tour)** $\leqslant 2 \cdot \textbf{weight(MST)}$



triangle ineq.:

cost(tour) ≤ cost(walk)

# Approximation Ratio of MST Tour

**Theorem:** The MST TSP tour gives a 2-approximation for the metric TSP problem.

**Proof:**

- Triangle inequality → length of tour is at most $2 \cdot \text{weight(MST)}$
- We have seen that $\text{weight(MST)} < \text{opt. tour length}$

Can we do even better?

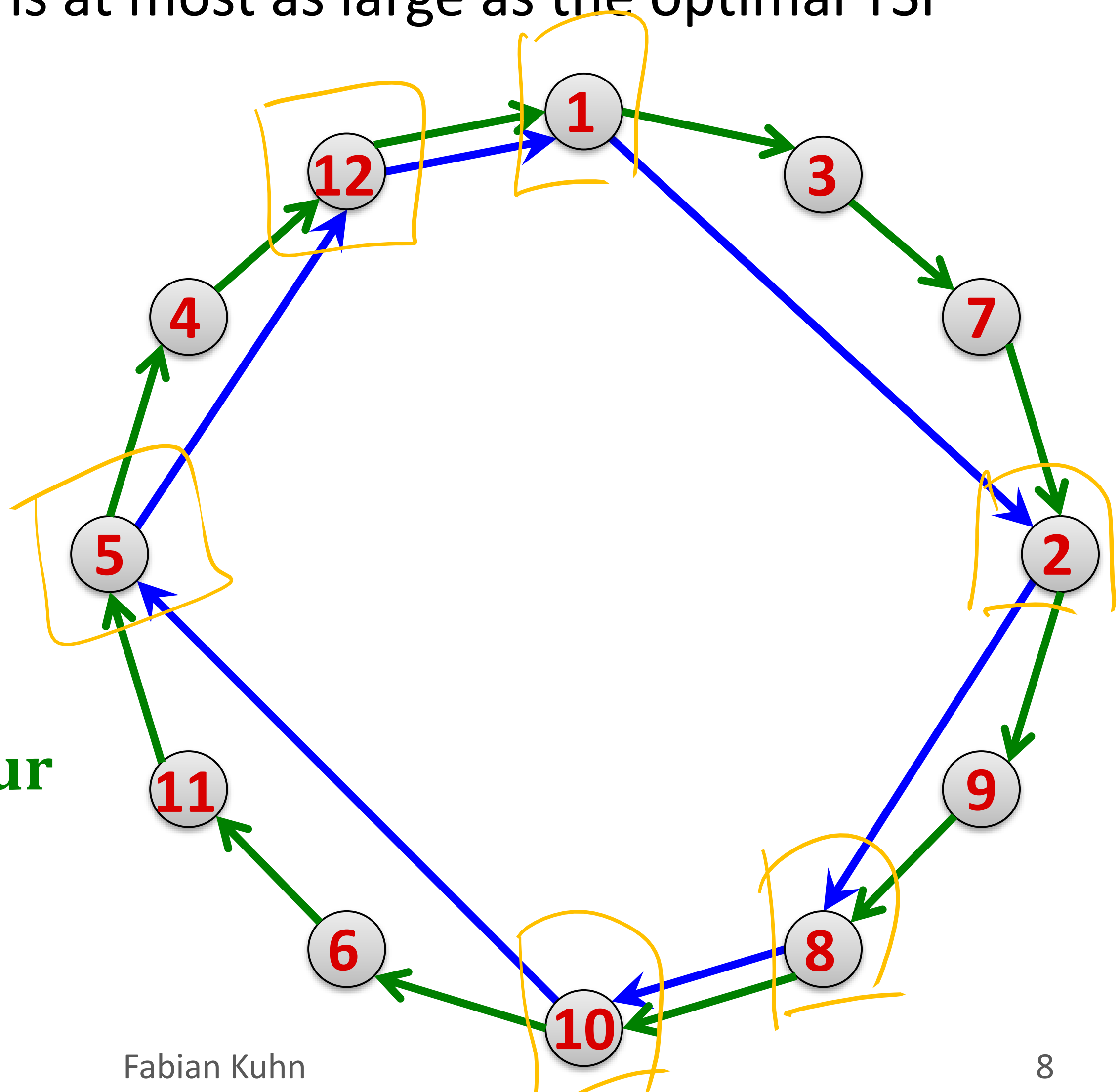
# Metric TSP Subproblems

**Claim:** Given a metric $(V, d)$ and $(V', d)$ for $V' \subseteq V$, the optimal TSP path/tour of $(V', d)$ is at most as large as the optimal TSP path/tour of $(V, d)$.

**Optimal TSP tour of nodes $1, 2, \ldots, 12$**

**Induced TSP tour for nodes $1, 2, 5, 8, 10, 12$**

**blue tour $\leq$ green tour**



# TSP and Matching

- Consider a metric TSP instance $(V, d)$ with an even number of nodes $|V|$
- Recall that a perfect matching is a matching $M \subseteq V \times V$ such that every node of $V$ is incident to an edge of $M$.
- Because $|V|$ is even and because in a metric TSP, there is an edge between any two nodes $u, v \in V$, any partition of $V$ into $|V|/2$ pairs is a perfect matching.
- The weight of a matching $M$ is the sum of the distances represented by all edges in $M$:

$$w(M) = \sum_{\{u,v\} \in M} d(u, v)$$

# TSP and Matching

**Lemma:** Assume we are given a TSP instance $(V, d)$ with an even number of nodes. The length of an optimal TSP tour of $(V, d)$ is at least twice the weight of a minimum weight perfect matching of $(V, d)$.

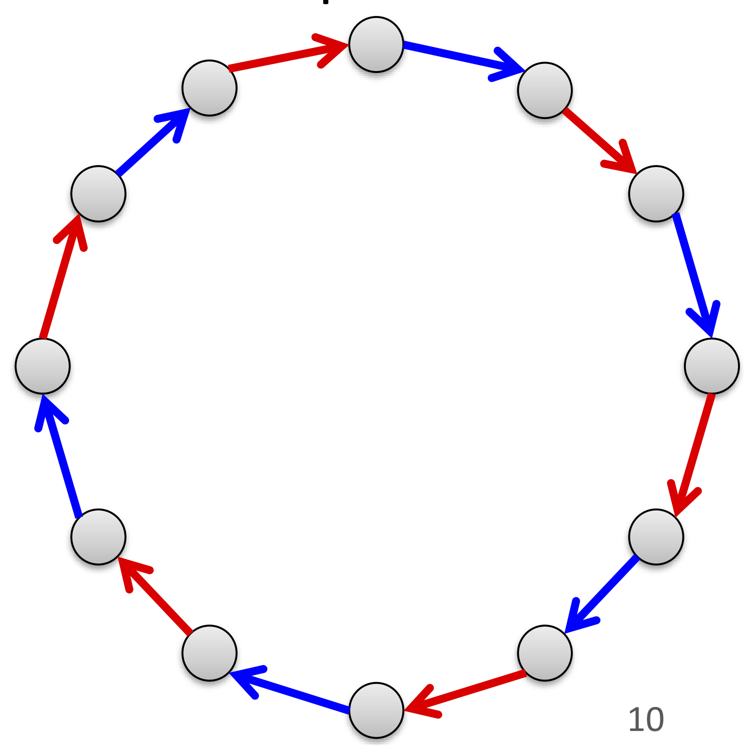
$w(M^*) \leq \frac{1}{2} \text{cost}(TSP_{\text{tour}})$

min. weight perfect matching

**Proof:**

- The edges of a TSP tour can be partitioned into 2 perfect matchings

$$TSP_{OPT} = \underbrace{\underset{\geq}{\text{red}} + \underset{\geq}{\text{blue}}}_{\text{weight of } M^*}$$



# Minimum Weight Perfect Matching

**Claim:** If $|V|$ is even, a minimum weight perfect matching of $(V, d)$ can be computed in polynomial time

**Proof Sketch:**

- We have seen that a minimum weight perfect matching in a complete bipartite graph can be computed in polynomial time
- With a more complicated algorithm, also a minimum weight perfect matching in a complete (non-bipartite) graph can be computed in polynomial time
- The algorithm uses similar ideas as the bipartite weighted matching algorithm and it uses the Blossom algorithm as a subroutine

# Algorithm Outline

Problem of MST algorithm:

- Every edge has to be visited twice

**Goal:**

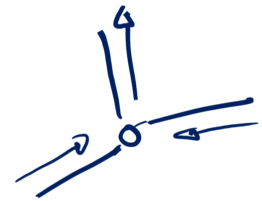
- Get a graph on which every edge only has to be visited once (and where still the total edge weight is small compared to an optimal TSP tour) not possible on a tree

**Euler Tours:**

- A tour that visits each edge of a graph exactly once is called an Euler tour (no self-loops)
- An Euler tour in a (multi-)graph exists if and only if every node of the graph has even degree
- That's definitely not true for a tree, but can we modify our MST suitably?

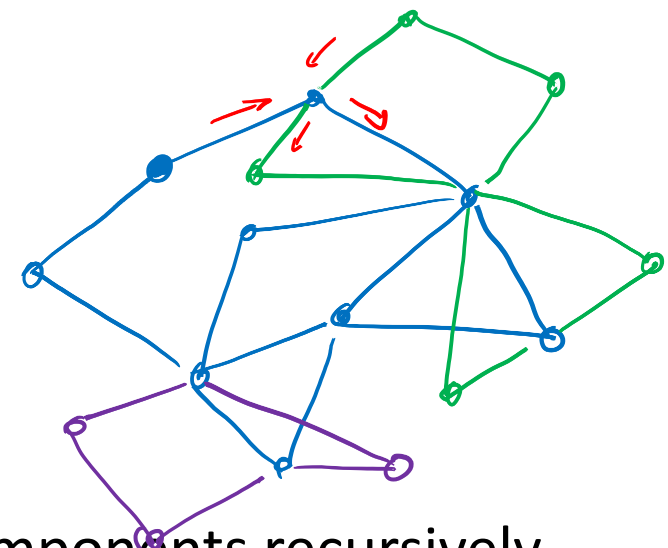
# Euler Tour

**Theorem:** A connected (multi-)graph $G$ has an Euler tour if and only if every node of $G$ has even degree.



**Proof:**

- If $G$ has an odd degree node, it clearly cannot have an Euler tour
- If $G$ has only even degree nodes, a tour can be found recursively:

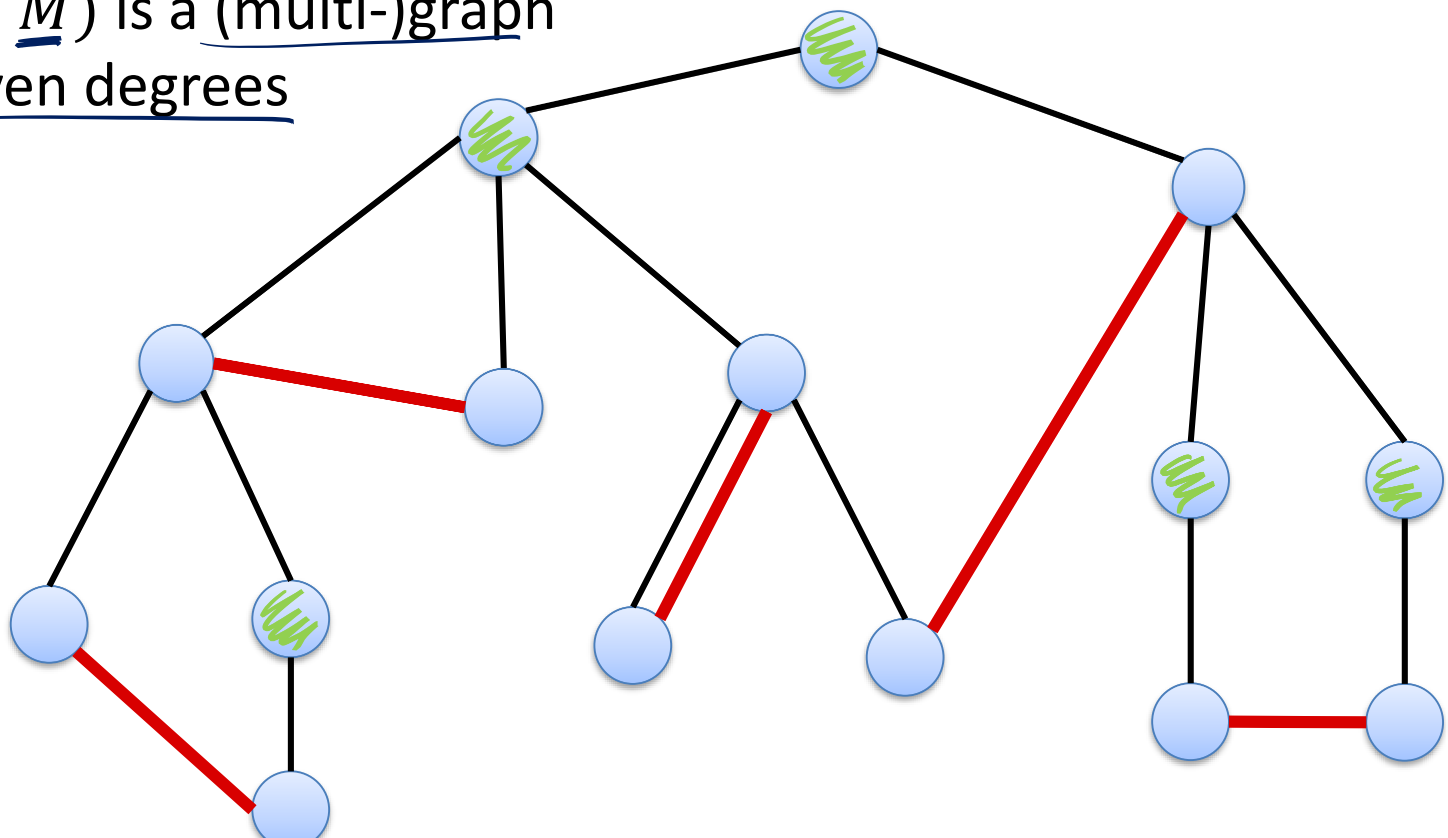


1. Start at some node
2. As long as possible, follow an unvisited edge
   - Gives a partial tour, the remaining graph still has even degree
3. Solve problem on remaining components recursively
4. Merge the obtained tours into one tour that visits all edges

# TSP Algorithm

$\sum_v \deg(v) = 2 \cdot |E|$

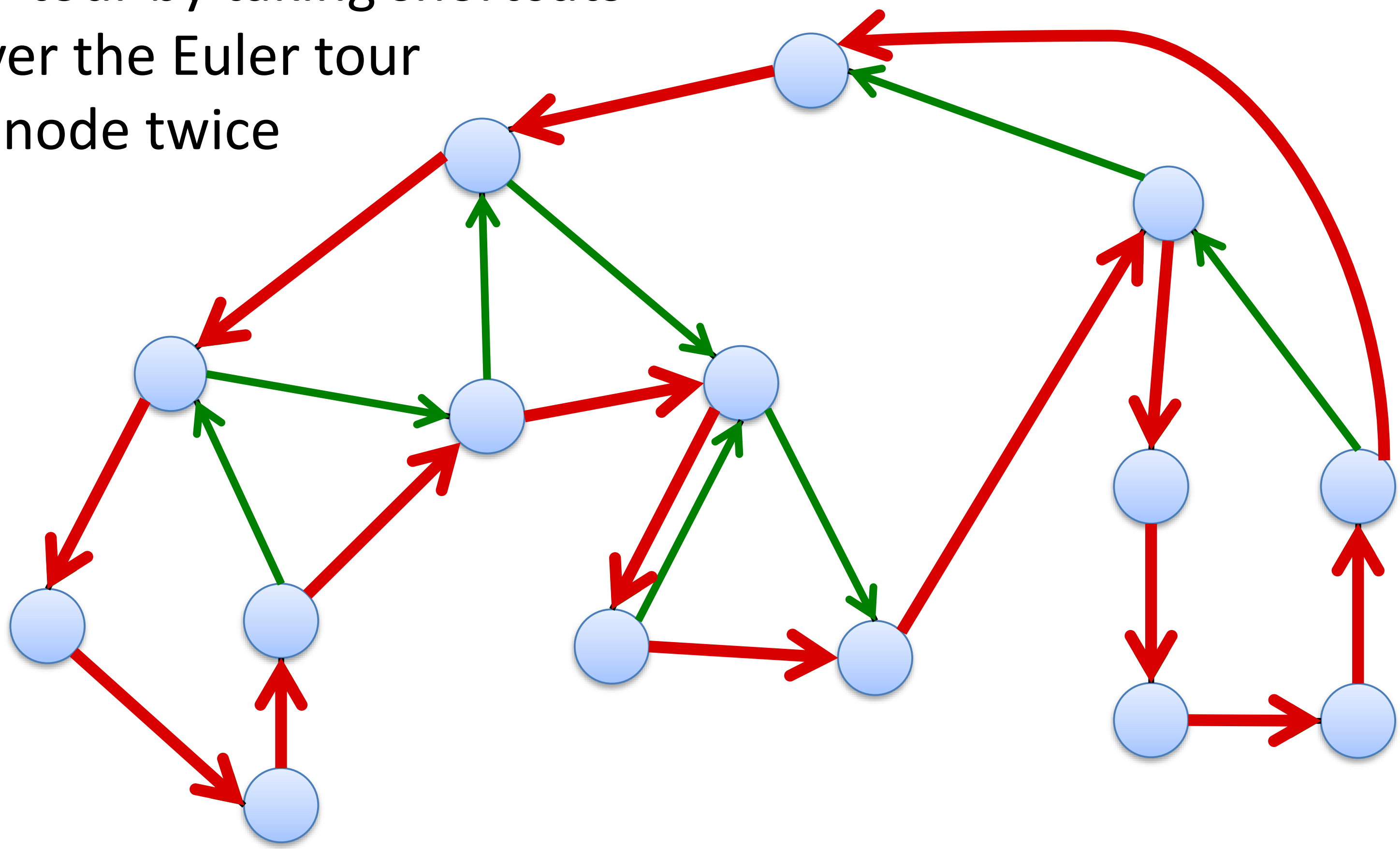
1. Compute MST $T$
2. $V_{\text{odd}}$: nodes that have an odd degree in $T$ ($|V_{\text{odd}}|$ is even)
3. Compute min weight perfect matching $M$ of $(V_{\text{odd}}, d)$
4. $(V, T \cup M)$ is a (multi-)graph with even degrees



# TSP Algorithm

5. Compute Euler tour on $(V, T \cup M)$
6. Total length of Euler tour $\leq \frac{3}{2} \cdot \mathbf{TSP_{OPT}}$
7. Get TSP tour by taking shortcuts wherever the Euler tour visits a node twice

Euler tour = $\omega(MST) + \omega(\text{matching})$



# TSP Algorithm

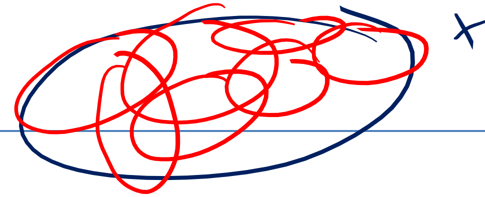
- The described algorithm is by Christofides

**Theorem:** The Christofides algorithm achieves an approximation ratio of at most ${}^{3}/_{2}$.

**Proof:**

- The length of the Euler tour is $\leq {}^{3}/_{2} \cdot \mathrm{TSP}_{\mathrm{OPT}}$
- Because of the triangle inequality, taking shortcuts can only make the tour shorter

# Set Cover



**Input:**

- A set of elements $X$ and a collection $\mathcal{S}$ of subsets $X$, i.e., $\mathcal{S} \subseteq 2^X$
  - such that $\bigcup_{S \in \mathcal{S}} S = X$

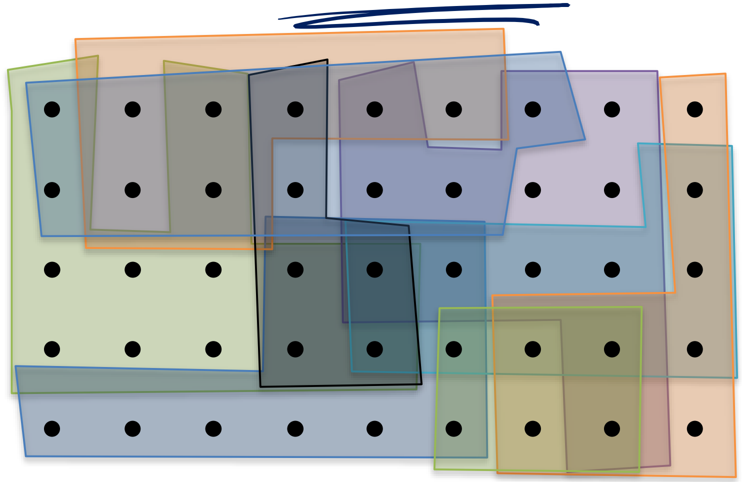
**Set Cover:**

set system

- A set cover $\mathcal{C}$ of $(X, \mathcal{S})$ is a subset of the sets $\mathcal{S}$ which covers $X$:

$$\bigcup_{S \in \mathcal{C}} S = X$$

**Example:**



$\boldsymbol{X}$

# Minimum (Weighted) Set Cover

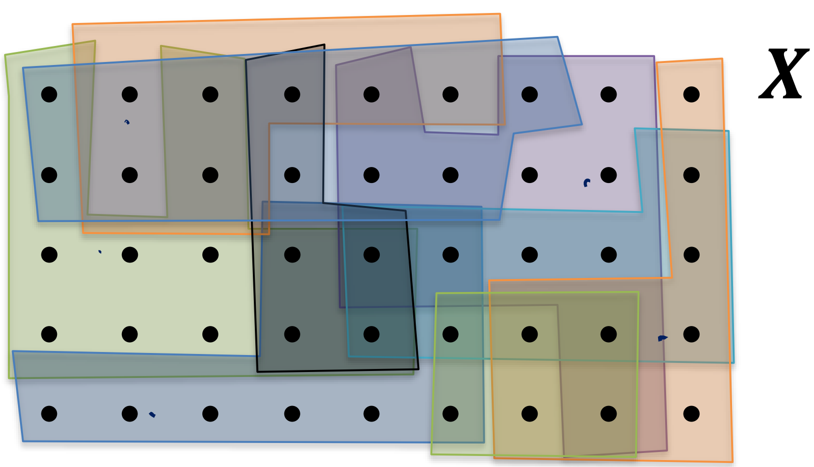
## Minimum Set Cover:

- **Goal:** Find a set cover $\mathcal{C}$ of smallest possible size
  - i.e., over $X$ with as few sets as possible

## Minimum Weighted Set Cover:

- Each set $S \in \mathcal{S}$ has a weight $w_S > 0$
- **Goal:** Find a set cover $\mathcal{C}$ of minimum weight

**Example:**

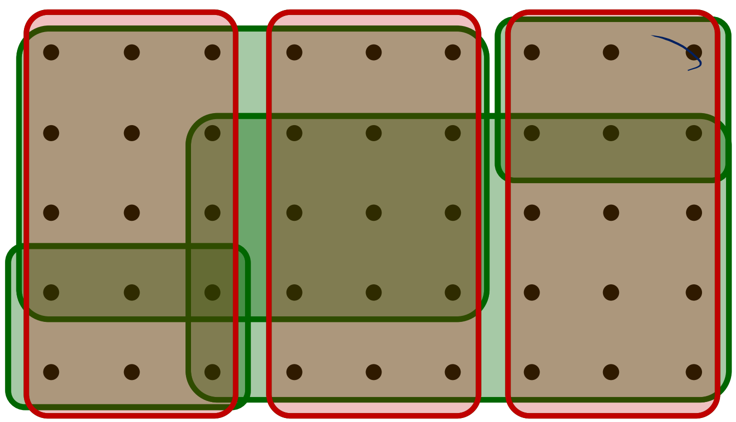


# Minimum Set Cover: Greedy Algorithm

**Greedy Set Cover Algorithm:**

- Start with $\mathcal{C} = \emptyset$
- In each step, add set $S \in \mathcal{S} \setminus \mathcal{C}$ to $\mathcal{C}$ s.t. $S$ covers as many uncovered elements as possible

**Example:**



# Weighted Set Cover: Greedy Algorithm

**Greedy Weighted Set Cover Algorithm:**

$\mathcal{C}$ : current set of subsets of $X$

- Start with $\mathcal{C} = \emptyset$
- In each step, add set $S \in \mathcal{S} \setminus \mathcal{C}$ with the best weight per newly covered element ratio (set with best efficiency):

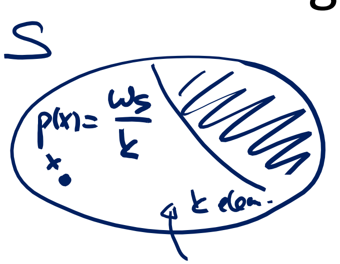
$$S = \arg\min_{S \in \mathcal{S} \setminus \mathcal{C}} \frac{w_S}{|S \setminus \bigcup_{T \in \mathcal{C}} T|}$$

price per newly covered element

# newly cov. elem. by S

**Analysis of Greedy Algorithm:**

- Assign a price $p(x)$ to each element $x \in X$: The efficiency of the set when covering the element
- If covering $x$ with set $S$, if partial cover is $\mathcal{C}$ before adding $S$:




$$p(x) = \frac{w_S}{|S \setminus \bigcup_{T \in \mathcal{C}} T|}$$

at the end (at all times)

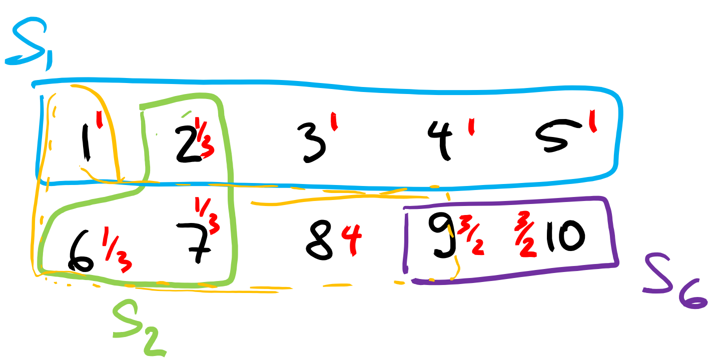
$$\sum_{x \in X} p(x) = \sum_{T \in \mathcal{C}} w_T$$

# Weighted Set Cover: Greedy Algorithm

**Example:**

- Universe $X = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$
- Sets $\mathcal{S} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$

$$
\begin{aligned}
&\cancel{S_1 = \{1, \cancel{2}, 3, 4, 5\},} && w_{S_1} = \underline{4} \leftarrow 2.\\
&\cancel{S_2 = \{\cancel{2}, \cancel{6}, \cancel{7}\},} && w_{S_2} = \underline{1} \leftarrow 1.\\
&S_3 = \{\cancel{1}, \cancel{6}, \cancel{7}, 8, 9\}, && w_{S_3} = \underline{4} \leftarrow 4\\
&S_4 = \{\cancel{2}, \cancel{4}, \cancel{7}, 9, 10\}, && w_{S_4} = 6\\
&S_5 = \{\cancel{1}, \cancel{3}, \cancel{5}, \cancel{6}, \cancel{7}, 8, 9, 10\}, && w_{S_5} = 9\\
&S_6 = \{9, 10\}, && w_{S_6} = \underline{3} \leftarrow 3.
\end{aligned}
$$




total price:

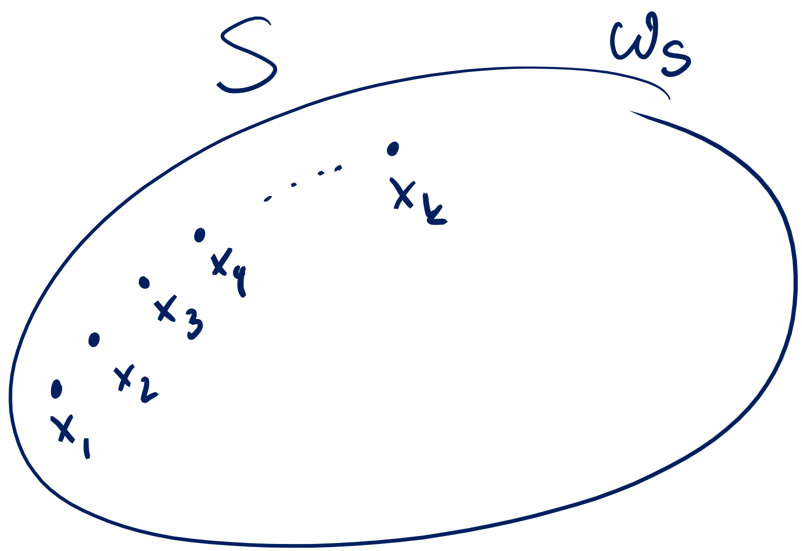
$3 \cdot \frac{1}{3} + 4 \cdot 1 + 2 \cdot \frac{3}{2} + 1 \cdot 4 = 12$

total weight: 12

# Weighted Set Cover: Greedy Algorithm

**Lemma:** Consider a set $S = \{x_1, x_2, \dots, x_k\} \in \mathcal{S}$ be a set and assume that the elements are covered in the order $x_1, x_2, \dots, x_k$ by the greedy algorithm (ties broken arbitrarily).

Then, the price of element $x_i$ is at most $p(x_i) \le \frac{w_S}{k-i+1}$




$$\sum_{x \in S} p(x) \le w_S \Big( \underbrace{\frac{1}{k} + \frac{1}{k-1} + \frac{1}{k-2} + \dots + \frac{1}{1}}_{} \Big)$$

$$H(k) \le \ln k + 1$$

$$p(x_1) \le \frac{w_S}{k}, \quad p(x_2) \le \frac{w_S}{k-1}, \quad p(x_3) \le \frac{w_S}{k-2}$$

# Weighted Set Cover: Greedy Algorithm

**Lemma:** Consider a set $S = \{x_1, x_2, \dots, x_k\} \in \mathcal{S}$ be a set and assume that the elements are covered in the order $x_1, x_2, \dots, x_k$ by the greedy algorithm (ties broken arbitrarily).

Then, the price of element $x_i$ is at most $p(x_i) \leq \frac{w_S}{k-i+1}$

**Corollary:** The total price of a set $S \in \mathcal{S}$ of size $|S| = k$ is

$$\underline{\sum_{x \in S}} p(x) \leq \underline{w_S} \cdot \underline{H_k}, \qquad \text{where } \underline{H_k = \sum_{i=1}^{k} \frac{1}{i}} \leq 1 + \ln k$$

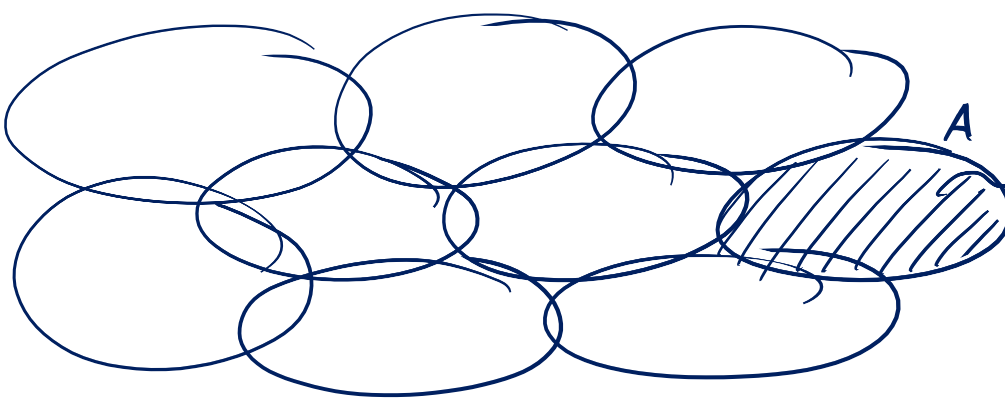
# Weighted Set Cover: Greedy Algorithm

**Corollary:** The total price of a set $S \in \mathcal{S}$ of size $|S| = k$ is

$$\sum_{x \in S} p(x) \leq w_S \cdot H_k, \qquad \text{where } H_k = \sum_{i=1}^{k} \frac{1}{i} \leq 1 + \ln k$$

**Theorem:** The approximation ratio of the greedy minimum (weighted) set cover algorithm is at most $\boldsymbol{H_s \leq 1 + \ln s}$, where $s$ is the cardinality of the largest set ($s = \max_{S \in \mathcal{S}} |S|$).

OPT:




total price

$\leq \omega_A \cdot H(|A|)$

$\leq \omega_A \cdot H(s)$

$$\sum_{x \in X} p(x) \leq \sum_{A \in OPT} \sum_{x \in A} p(x) \leq \sum_{A \in OPT} \omega_A \cdot H(s) = \omega(OPT) \cdot H(s)$$

$$\leq \omega(OPT)(1 + \ln s)$$

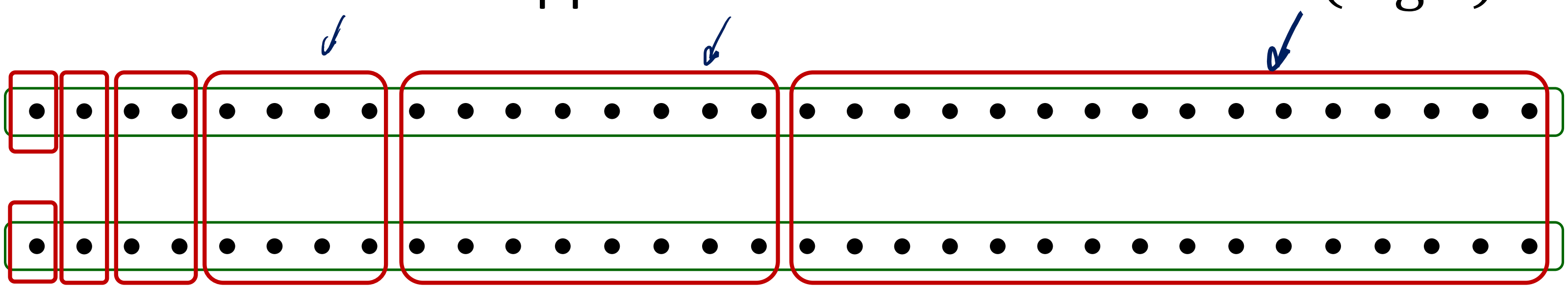
# Set Cover Greedy Algorithm

Can we improve this analysis?

No! Even for the unweighted minimum set cover problem, the approximation ratio of the greedy algorithm is $\geq \big(1 - o(1)\big) \cdot \ln s$.

- if $s$ is the size of the largest set... ($s$ can be linear in $n$)

Let's show that the approximation ratio is at least $\Omega(\log n)$...



$$\mathbf{OPT = 2}$$

$$\mathbf{GREEDY \geq \log_2 n}$$

# Set Cover: Better Algorithm?

An approximation ratio of $\ln n$ seems not spectacular...

Can we improve the approximation ratio?

No, unfortunately not, unless $\mathrm{P} = \mathrm{NP}$

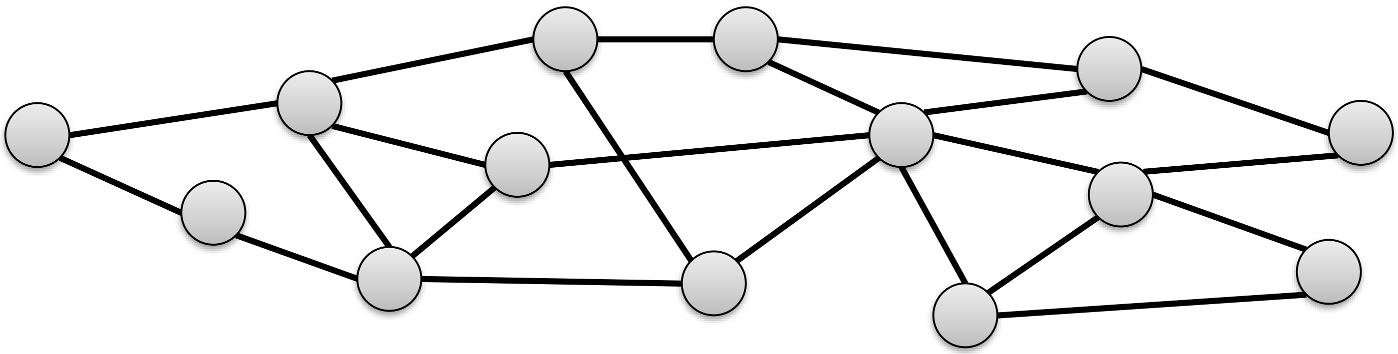
Dinur & Steurer showed in 2013 that unless $\mathrm{P} = \mathrm{NP}$, minimum set cover cannot be approximated better than by a factor $(1 - o(1)) \cdot \ln n$ in polynomial time.

- Proof is based on the so-called PCP theorem
  - PCP theorem is one of the main (relatively) recent advancements in theoretical computer science and the major tool to prove approximation hardness lower bounds
  - Shows that every language in NP has certificates of polynomial length that can be checked by a randomized algorithm by only querying a constant number of bits (for any constant error probability)

# Set Cover: Special Cases

**Vertex Cover:** set $S \subseteq V$ of nodes of a graph $G = (V, E)$ such that

$$\forall \{\boldsymbol{u}, \boldsymbol{v}\} \in \boldsymbol{E}, \qquad \{\boldsymbol{u}, \boldsymbol{v}\} \cap \boldsymbol{S} \neq \emptyset.$$



**Minimum Vertex Cover:**

- Find a vertex cover of minimum cardinality

**Minimum Weighted Vertex Cover:**

- Each node has a weight
- Find a vertex cover of minimum total weight

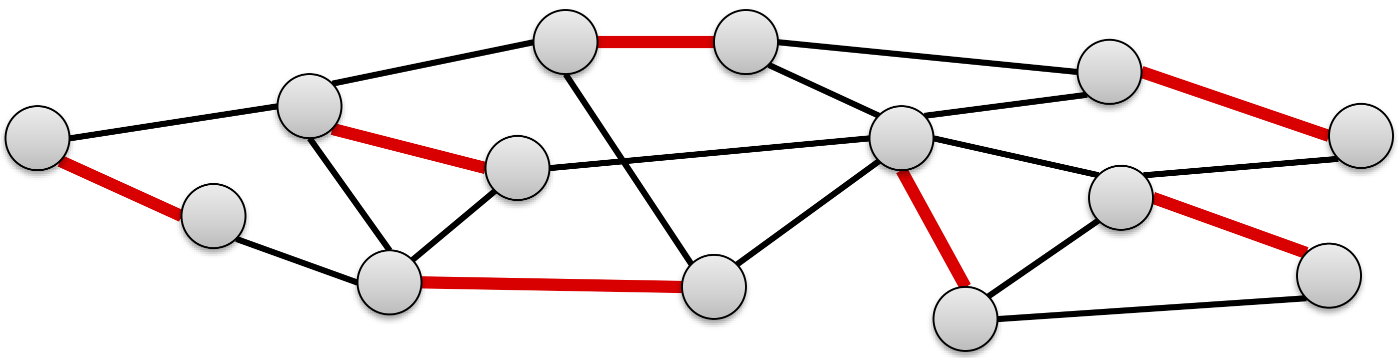
# Vertex Cover vs Matching

Consider a matching $M$ and a vertex cover $S$

**Claim:** $|M| \leq |S|$

**Proof:**

- At least one node of every edge $\{u, v\} \in M$ is in $S$
- Needs to be a different node for different edges from $M$



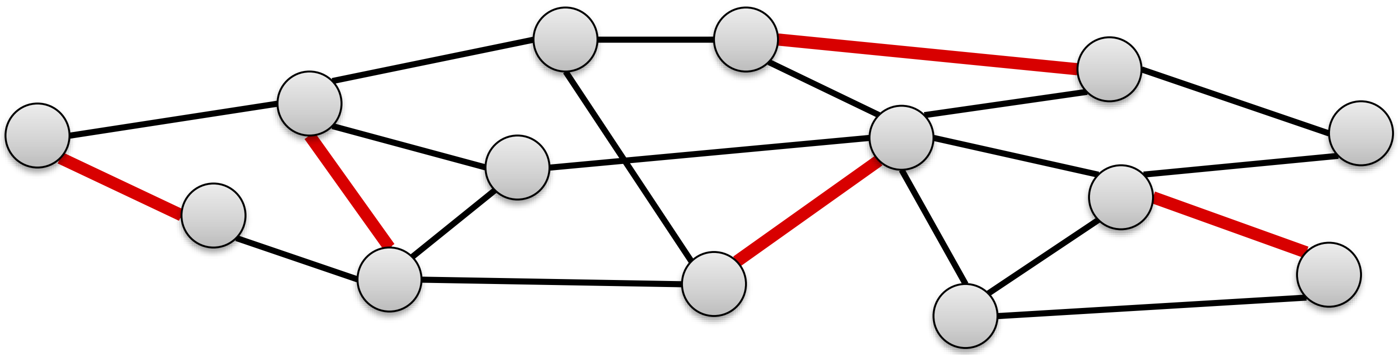
# Vertex Cover vs Matching

Consider a matching $M$ and a vertex cover $S$

**Claim:** If $M$ is maximal and $S$ is minimum, $|S| \leq 2|M|$

**Proof:**

- $M$ is maximal: for every edge $\{u, v\} \in E$, either $u$ or $v$ (or both) are matched



- Every edge $e \in E$ is "covered" by at least one matching edge
- Thus, the set of the nodes of all matching edges gives a vertex cover $S$ of size $|S| = 2|M|$.

# Maximal Matching Approximation

**Theorem:** For any maximal matching $M$ and any maximum matching $M^*$, it holds that

$$|M| \geq \frac{|M^*|}{2}.$$

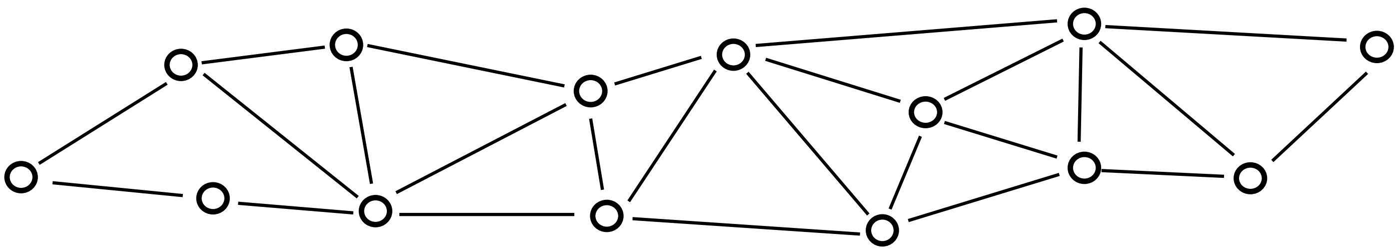
**Proof:**

**Theorem:** The set of all matched nodes of a maximal matching $M$ is a vertex cover of size at most twice the size of a min. vertex cover.

# Set Cover: Special Cases

**Dominating Set:**

Given a graph $G = (V, E)$, a dominating set $S \subseteq V$ is a subset of the nodes $V$ of $G$ such that for all nodes $u \in V \setminus S$, there is a neighbor $v \in S$.



# Minimum Hitting Set

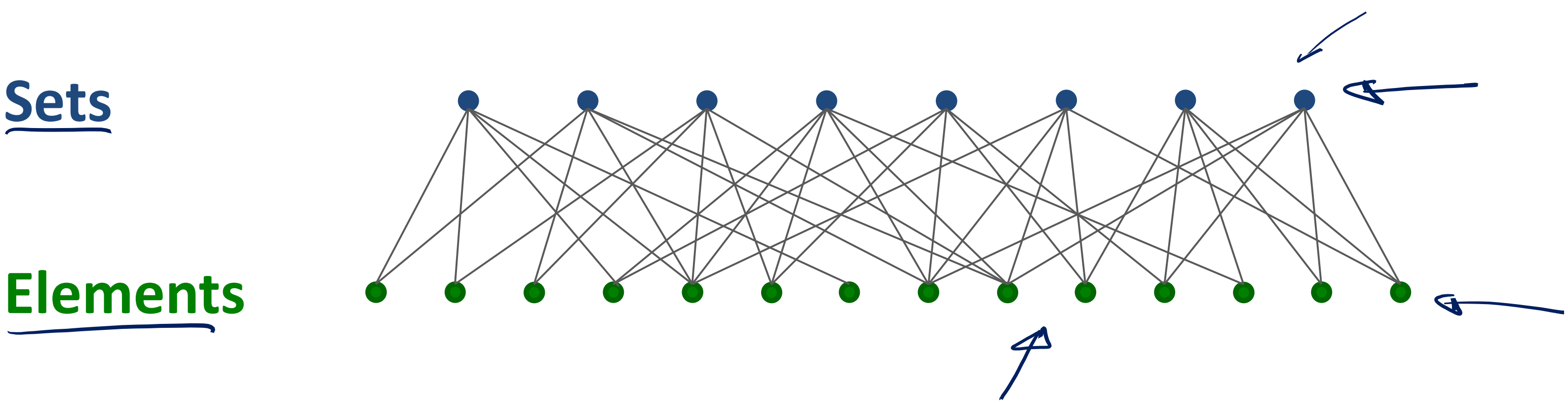
**Given:** Set of elements $X$ and collection of subsets $\mathcal{S} \subseteq 2^X$

– Sets cover $X$: $\bigcup_{S \in \mathcal{S}} S = X$

**Goal:** Find a min. cardinality subset $H \subseteq X$ of elements such that

$$\forall S \in \mathcal{S} : S \cap H \neq \emptyset$$

Problem is equivalent to min. set cover with roles of sets and elements interchanged



# Knapsack

- $n$ items $1, \dots, n$, each item has weight $w_i > 0$ and value $v_i > 0$
- Knapsack (bag) of capacity $W$

- Goal: pack items into knapsack such that total weight is at most $W$ and total value is maximized:

$$\max \sum_{i \in S} v_i$$

$$\text{s.t.} \;\; S \subseteq \{1, \dots, n\} \text{ and } \sum_{i \in S} w_i \leq W$$

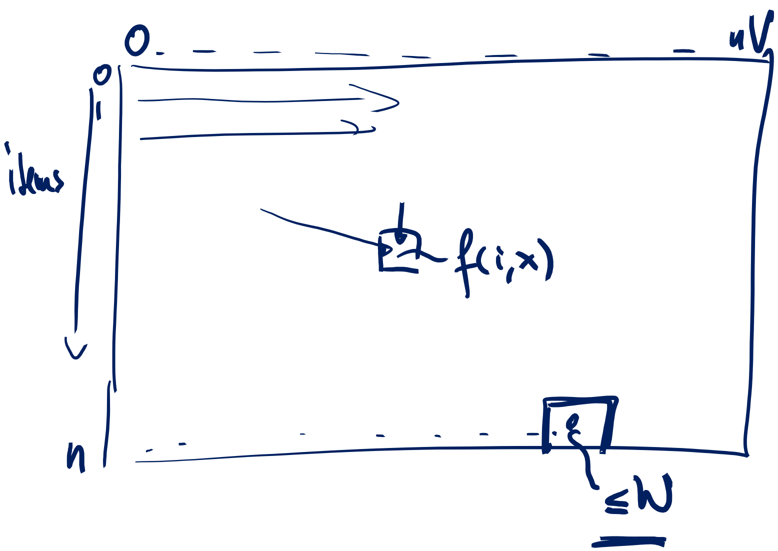
- E.g.: jobs of length $w_i$ and value $v_i$, server available for $W$ time units, try to execute a set of jobs that maximizes the total value

# Knapsack: Dynamic Programming Alg.

**We have shown:**

- If all item weights $w_i$ are integers, using dynamic programming, the knapsack problem can be solved in time $O(nW)$
- If all values $v_i$ are integers, there is another dynamic progr. algorithm that runs in time $O(n^2V)$, where $V$ is the max. value.

$f(i,x)$: min. weight to obtain exactly total value of $x$ when only using items $1,\dots,i$




$f(i,0) = 0$

$f(0,x) = \infty$ (for $x>0$)

$$f(i,x) = \min\begin{cases} f(i-1,x) \\ f(i-1,x-v_i) + w_i \end{cases}$$

$V := \max_i v_i$