Chapter 8
Approximation Algorithms

(next week)
Mon : lecture
Thu : exercises

Algorithm Theory
WS 2016/17

Fabian Kuhn

Approximation Algorithms

- Optimization appears everywhere in computer science
- We have seen many examples, e.g.:
  - scheduling jobs
  - traveling salesperson
  - maximum flow, maximum matching
  - minimum spanning tree
  - minimum vertex cover
  - …
- Many discrete optimization problems are NP-hard
- They are however still important and we need to solve them
- As algorithm designers, we prefer algorithms that produce solutions which are provably good, even if we can't compute an optimal solution.

Approximation Algorithms: Examples

We have already seen two approximation algorithms

- **Metric TSP:** If distances are positive and satisfy the triangle inequality, the greedy tour is only by a log-factor longer than an optimal tour

- **Maximum Matching and Vertex Cover:** A maximal matching gives solutions that are within a factor of 2 for both problems.

Approximation Ratio

An approximation algorithm is an algorithm that computes a solution for an optimization with an objective value that is provably within a bounded factor of the optimal objective value.

Formally:

- $\mathrm{OPT} \geq 0$: optimal objective value
  $\mathrm{ALG} \geq 0$: objective value achieved by the algorithm

- **Approximation Ratio α:** $\alpha \geq 1$

$$\textbf{Minimization:}\ \alpha := \max_{\text{input instances}} \frac{\mathrm{ALG}}{\mathrm{OPT}}$$

example:
Vertex cover / TSP

$$\textbf{Maximization:}\ \alpha := \max_{\text{input instances}} \frac{\mathrm{OPT}}{\mathrm{ALG}}$$

maximum m.
maximal match.

Example: Load Balancing

We are given:

- m machines $M_1, \dots, M_m$
- n jobs, processing time of job i is t_i

Goal:

- Assign each job to a machine such that the makespan is minimized

  makespan: largest total processing time of any machine

The above load balancing problem is NP-hard and we therefore want to get a good approximation for the problem.

Greedy Algorithm

There is a simple greedy algorithm:

- Go through the jobs in an arbitrary order
- When considering job i, assign the job to the machine that currently has the smallest load.

Example: 3 machines, 12 jobs

3	4	2	3	1	6	4	4	3	2	1	5

Greedy Assignment:

M_1: | 3 | 1 | 6 | 1 | 5 |

M_2: | 4 | 4 | 3 |

M_3: | 2 | 3 | 4 | 2 |

makespan: 16

Optimal Assignment:

M_1: | 3 | 4 | 2 | 3 | 1 |

M_2: | 6 | 4 | 3 |

M_3: | 4 | 2 | 1 | 5 |

makespan: 13

Greedy Analysis

- We will show that greedy gives a 2-approximation
- To show this, we need to compare the solution of greedy with an optimal solution (that we can't compute)
- Lower bound on the optimal makespan T^*:

$$T^* \geq \frac{1}{m} \cdot \sum_{i=1}^{n} t_i$$

- Lower bound can be far from T^*:
  - m machines, m jobs of size 1, 1 job of size m

$\sum t_i = 2m$

$$T^* = m, \qquad \frac{1}{m} \cdot \sum_{i=1}^{n} t_i = 2$$

Greedy Analysis

- We will show that greedy gives a 2-approximation
- To show this, we need to compare the solution of greedy with an optimal solution (that we can't compute)
- Lower bound on the optimal makespan T^*:

$$T^* \geq \underline{\frac{1}{m} \cdot \sum_{i=1}^{n} t_i}$$

- Second lower bound on optimal makespan T^*:

$$T^* \geq \underline{\max_{1 \leq i \leq n} t_i}$$

Greedy Analysis

Theorem: The greedy algorithm has approximation ratio ≤ 2, i.e., for the makespan T of the greedy solution, we have $T \leq 2T^*$.

Proof:

- For machine k, let T_k be the time used by machine k
- Consider some machine M_i for which $T_i = T$
- Assume that job j is the last one schedule on M_i:

{ time when t_j gets assigned

$$\boldsymbol{M_i}: \quad \underbrace{\boxed{\quad T - t_j \quad}}\underbrace{\boxed{\; t_j \;}}$$

(assigned)

- When job j is scheduled, M_i has the minimum load

$\hookrightarrow \forall k \in \{1, \ldots, m\} : T_k \geq T - t_j$

Greedy Analysis

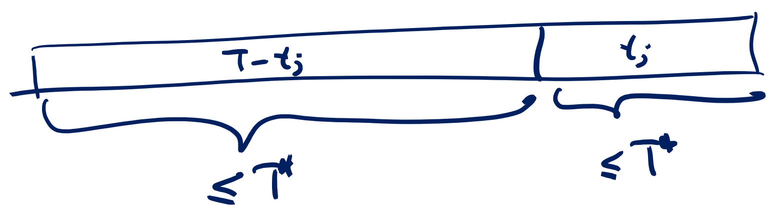
Theorem: The greedy algorithm has approximation ratio ≤ 2, i.e., for the makespan T of the greedy solution, we have $T \leq 2T^*$.

Proof:

- For all machines M_k: load $T_k \geq T - t_j$

$\sum t_i \geq m(T - t_j)$

$\hookrightarrow T^* \geq \frac{1}{m} \sum t_i \geq T - t_j$



$T = T - t_j + t_j \leq \underline{\underline{2T^*}}$

$\square$

Can We Do Better?

The analysis of the greedy algorithm is almost tight:

- Example with $n = m(m-1)+1$ jobs
- Jobs $1, \dots, n-1 = m(m-1)$ have $t_i = 1$, job n has $t_n = m$

Greedy Schedule:

M_1: 1 1 1 1 $\cdots$ 1 $t_n = m$

M_2: 1 1 1 1 $\cdots$ 1   makespan: $2m-1$

M_3: 1 1 1 1 $\cdots$ 1

$\vdots$ $\vdots$

M_m: $\underbrace{1\ 1\ 1\ 1\ \cdots\ 1}_{m-1}$

OPT: makespan $= m$

M_1: $t_n = m$

M_2: $\square \dots \square$

$\vdots$

M_m: $\underbrace{\square \dots \square}_{m}$

Improving Greedy

Bad case for the greedy algorithm:
One large job in the end can destroy everything

Idea: assign large jobs first

if $n \le m$: problem trivial
greedy optimal

Modified Greedy Algorithm:

1. Sort jobs by decreasing length s.t. $t_1 \geq t_2 \geq \cdots \geq t_n$
2. Apply the greedy algorithm as before (in the sorted order)

Lemma: If $n > m$: $T^* \geq t_m + t_{m+1} \geq 2t_{m+1}$

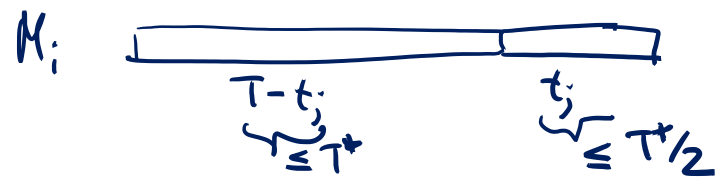
Proof:

- Two of the first $m + 1$ jobs need to be scheduled on the same machine
- Jobs m and $m + 1$ are the shortest of these jobs

Analysis of the Modified Greedy Alg.

Theorem: The modified algorithm has approximation ratio $\leq {}^{3}/_{2}$.

Proof:



- We show that $T \leq {}^{3}/_{2} \cdot T^*$
- As before, we consider the machine M_i with $T_i = T$
- Job j (of length t_j) is the last one scheduled on machine M_i
- If j is the only job on M_i, we have $T = T^*$

$T^* \geq 2t_{m+1}$

- Otherwise, we have $j \geq m + 1$

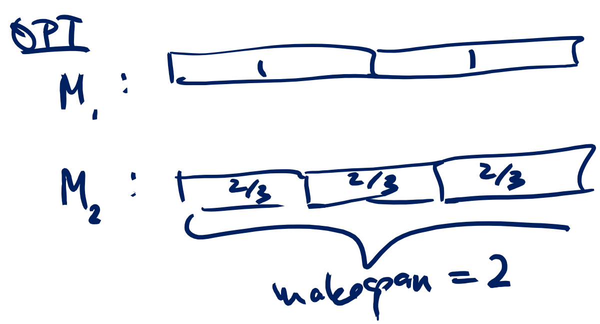
$t_j \leq t_{m+1}$

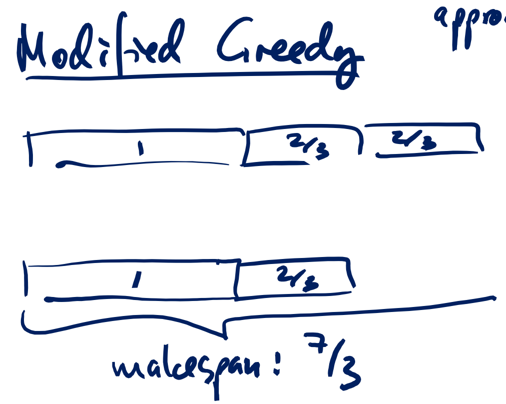
  - The first m jobs are assigned to m distinct machines

approx ratio $\leq 4/3$

jobs: $1, 1, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}$ $m=2$

approx ratio: $\geq 7/6$



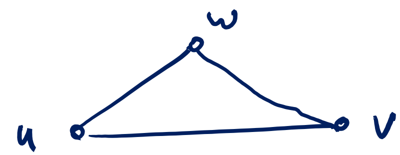


Metric TSP

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:

$$d(u,v) = d(v,u) \geq 0, \qquad d(u,v) = 0 \Leftrightarrow u = v$$
$$\forall u,v,w \in V : \underbrace{d(u,v) \leq d(u,w) + d(w,v)}_{\Delta\text{-inequality}}$$



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…

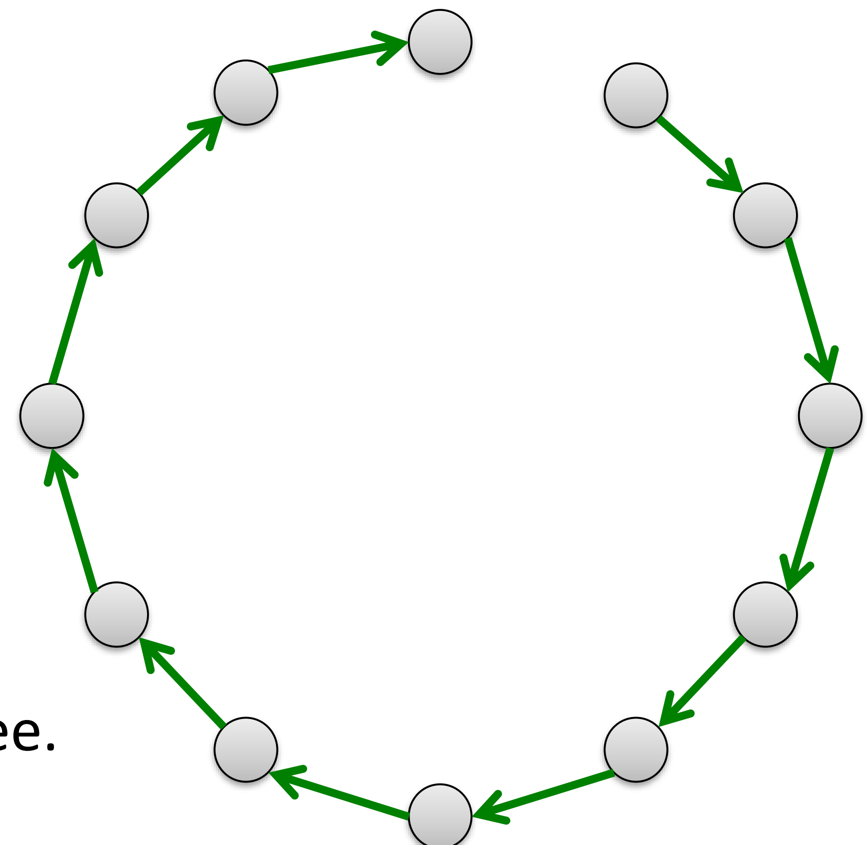
TSP and MST

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

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

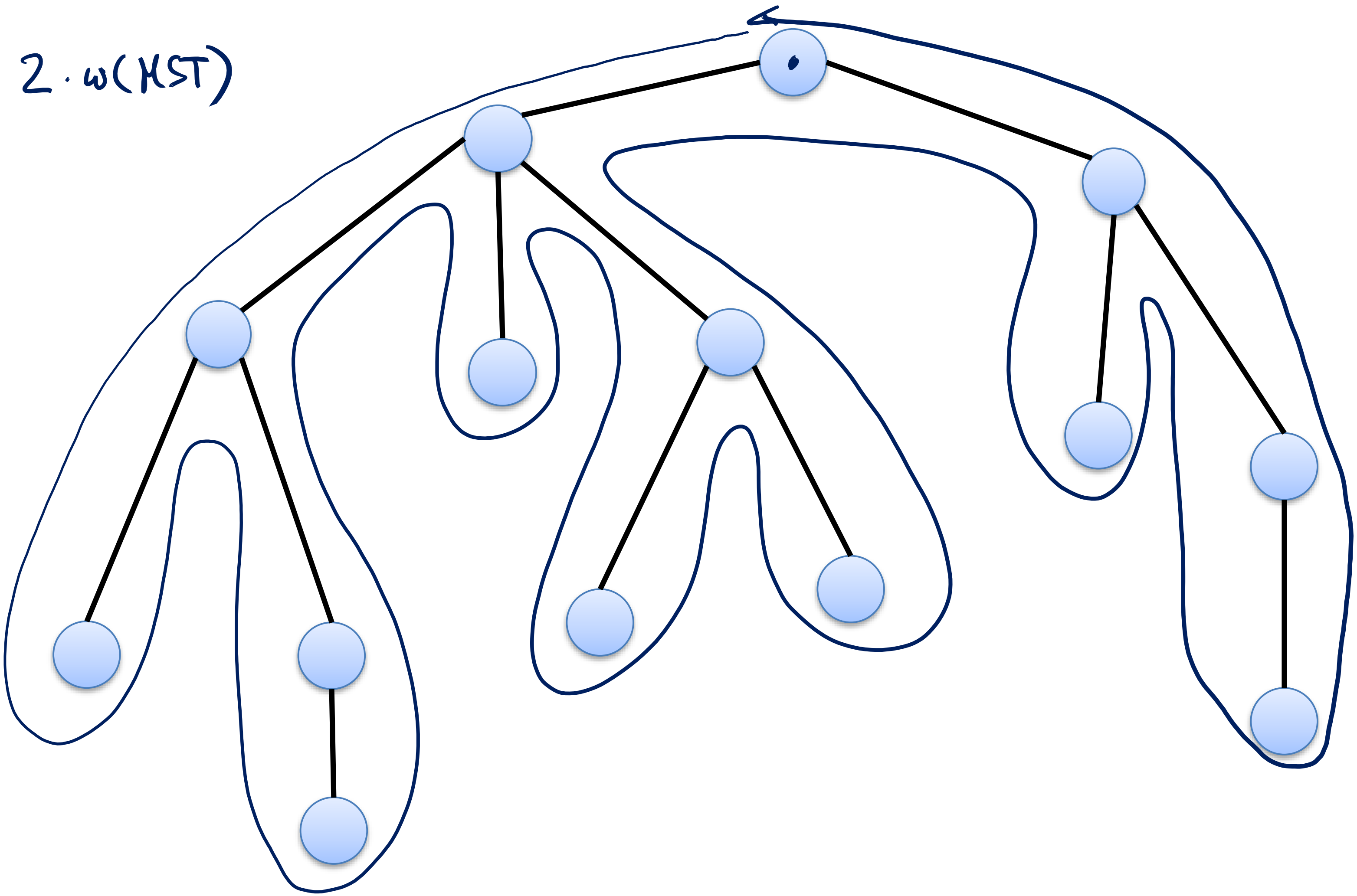


Corollary: Since an optimal TSP tour is longer than an optimal TSP path, the length of an optimal TSP tour is also lower bounded by the weight of a minimum spanning tree.

The MST Tour

Walk around the MST…

length tour $= 2 \cdot w(MST)$

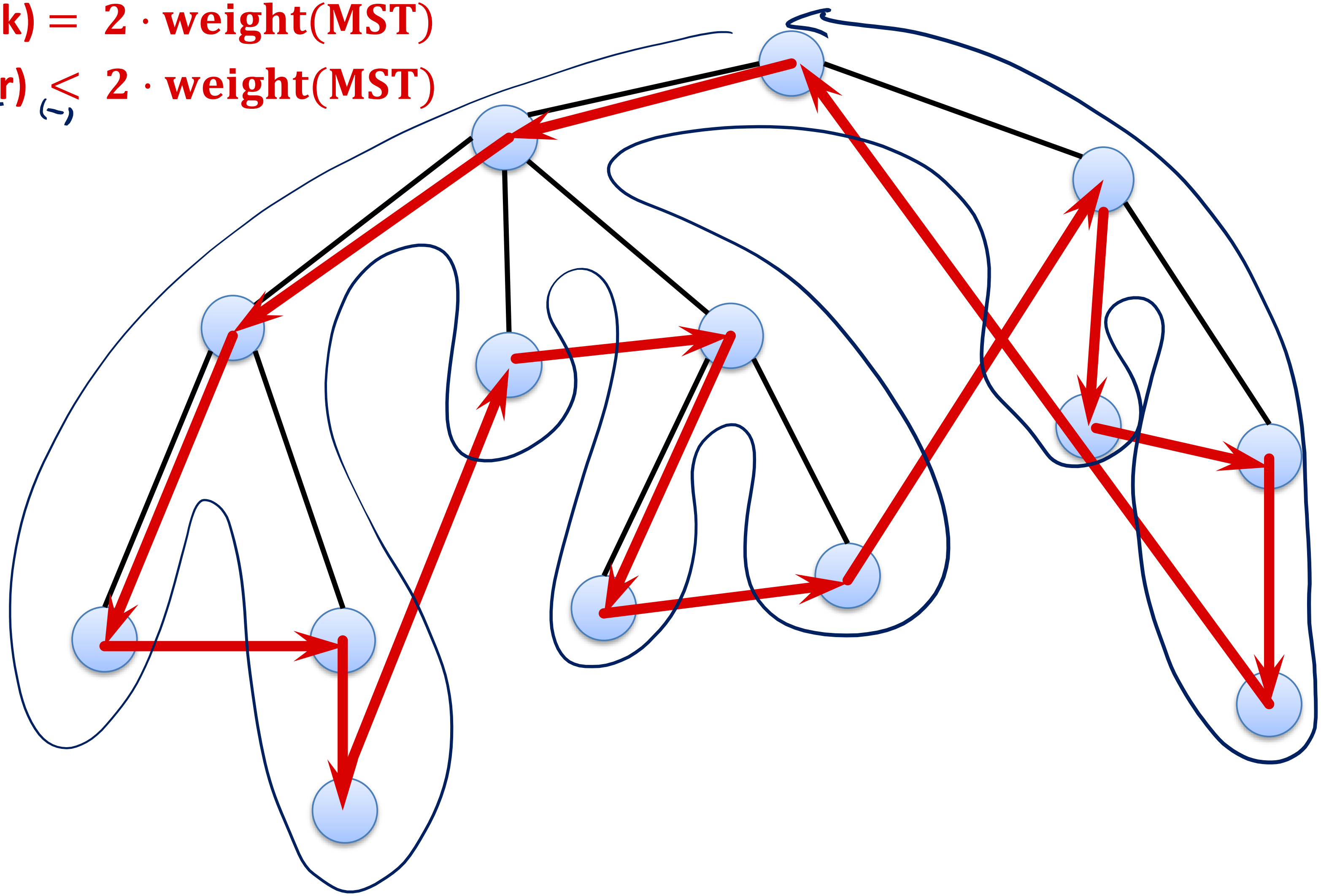


The MST Tour

Walk around the MST…

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

Cost (tour) $\underset{(-)}{\le} \ 2 \cdot \mathbf{weight(MST)}$



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 $2 \cdot \text{weight}(\text{MST})$
- We have seen that $\text{weight}(\text{MST}) < \text{opt. tour length}$

Can we do even better?

Metric TSP Subproblems

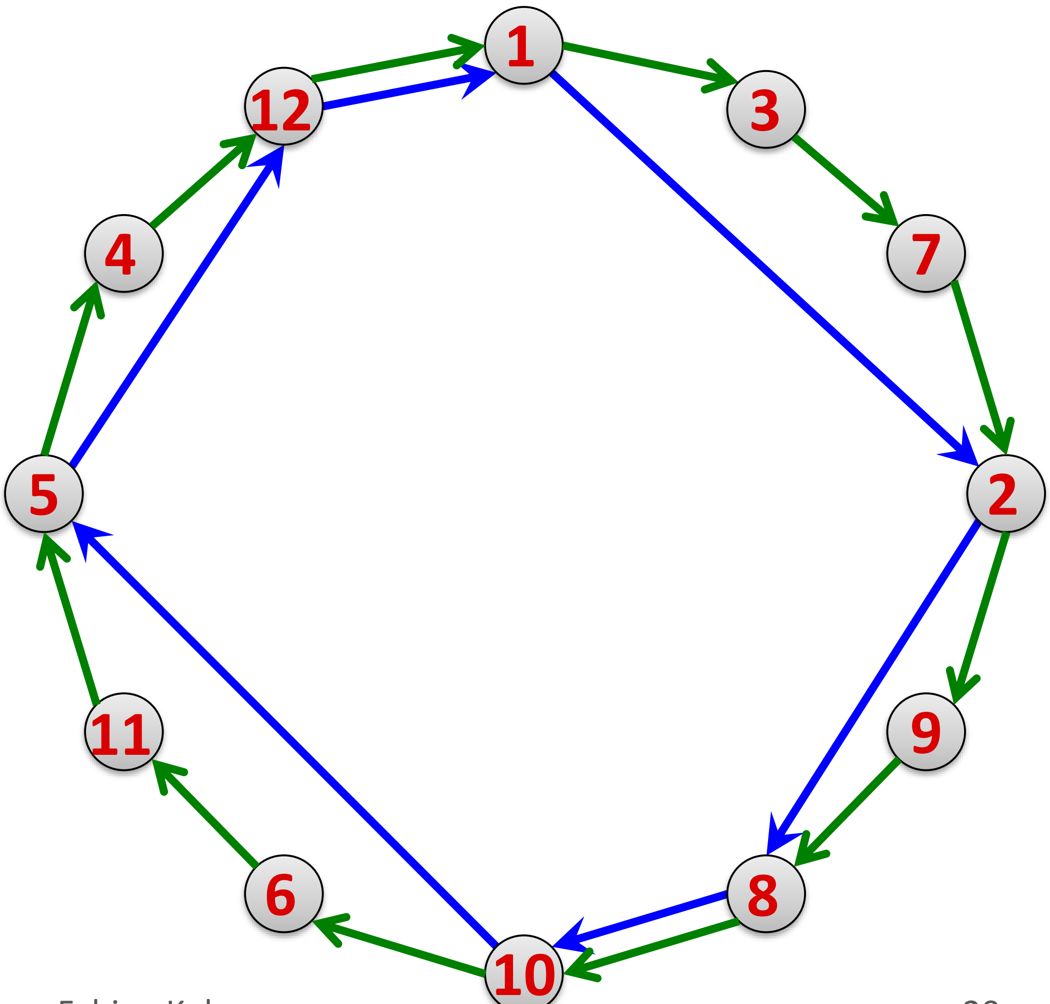
Claim: Given a metric (V, d) and (V', d) for $V' \subseteq V$, the optimal TSP path/tour of (V', d) is at most as large as the optimal TSP path/tour of (V, d).

Optimal TSP tour of nodes $1, 2, \dots, 12$

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

blue tour $\leq$ green tour

triangle ineq.



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:

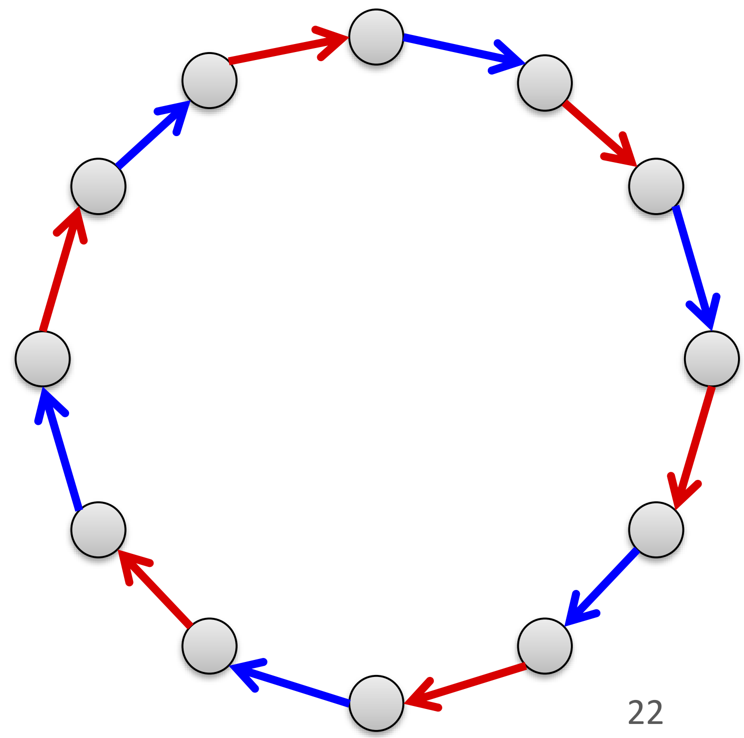
$$w(M) = \sum_{\{u,v\} \in M} d(u, v)$$

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).

Proof:

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



$TSP_{OPT} = \text{red} + \text{blue}$

$\text{red} \geq$ weight of a min. weight perf. match. $\leq \text{blue}$

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 complete (non-bipartite) graphs 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:

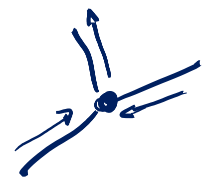
- Get a graph on which every edge only has to be visited once (and where still the total edge weight is small compared to an optimal TSP tour)

Euler Tours:

- A tour that visits each edge of a graph exactly once is called an Euler tour
- 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?

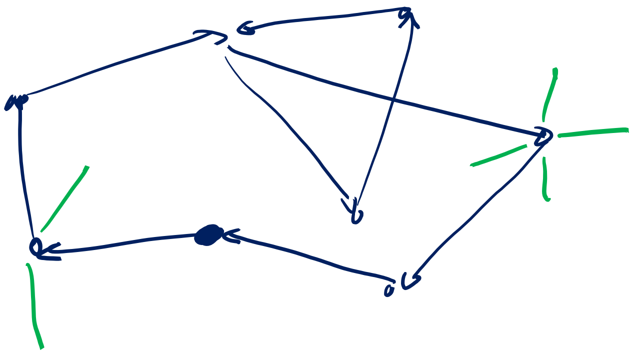
Euler Tour

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:



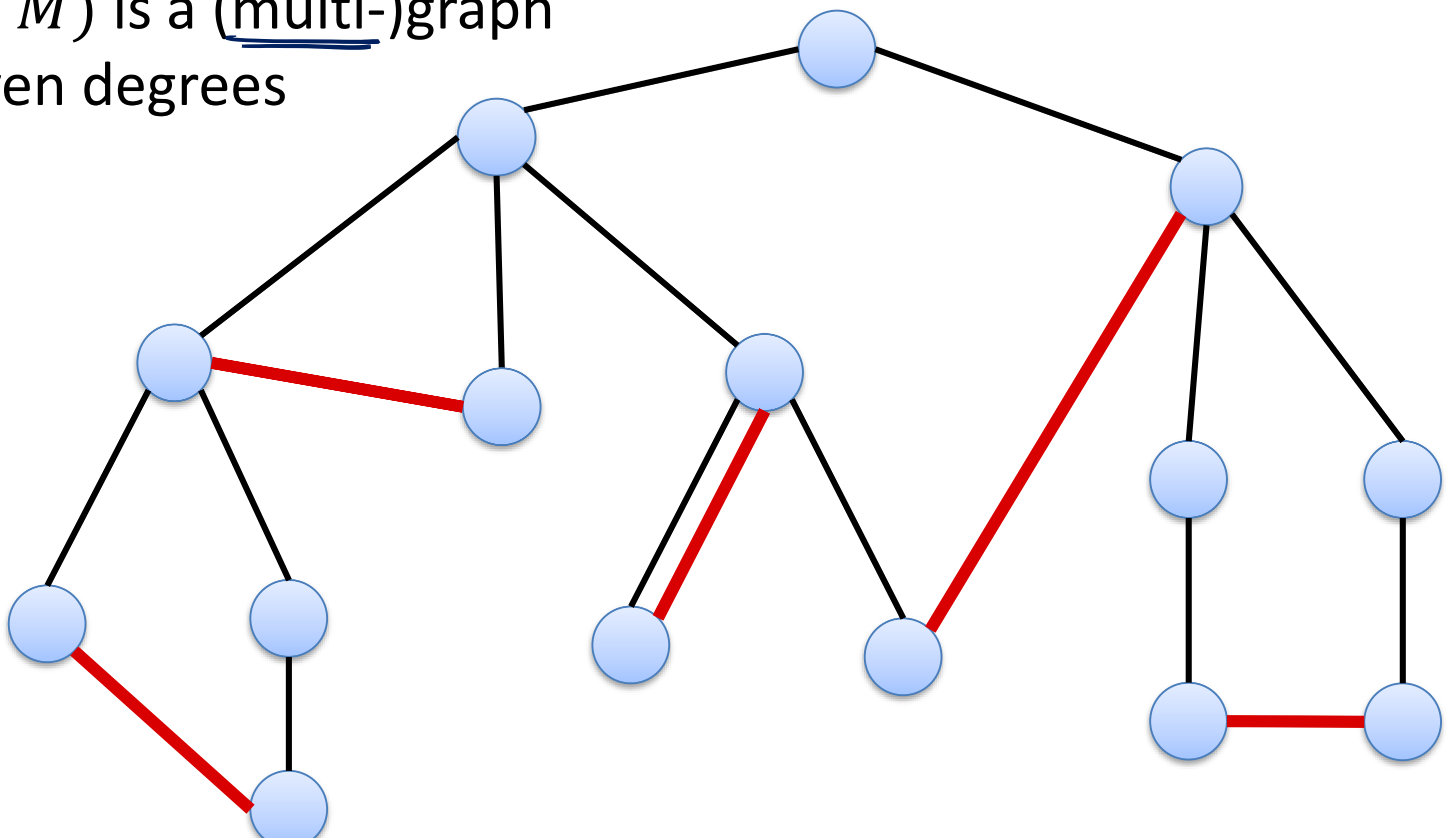
1. Start at some node
2. As long as possible, follow an unvisited edge
   - Gives a partial tour, the remaining graph still has even degree
3. Solve problem on remaining components recursively
4. Merge the obtained tours into one tour that visits all edges

TSP Algorithm

$\sum_{v \in V} deg(v) = 2 \cdot |E|$

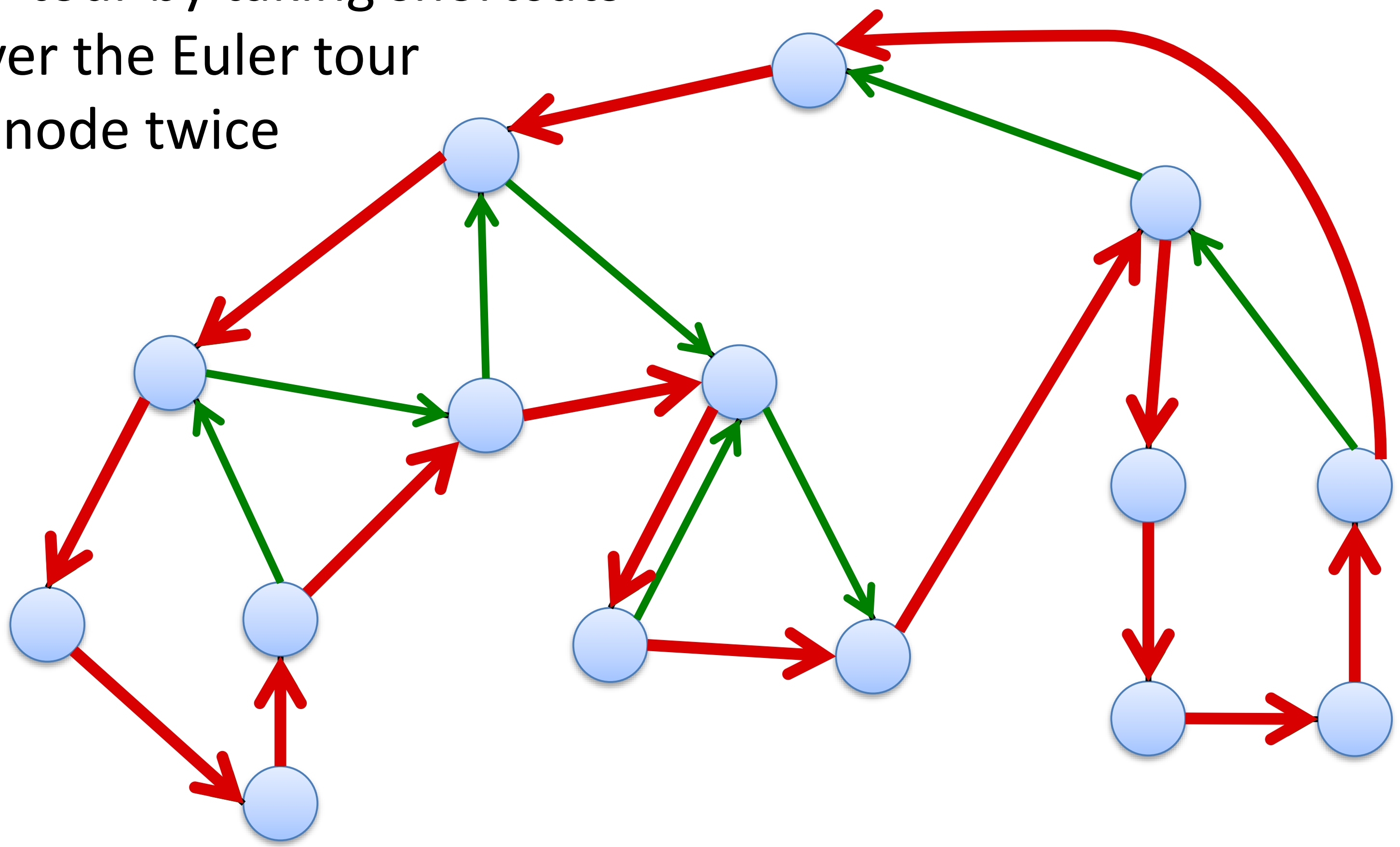
1. Compute MST T
2. V_{odd}: nodes that have an odd degree in T ($|V_{\text{odd}}|$ is even)
3. Compute min weight perfect matching M of (V_{odd}, d)
4. $(V, T \cup M)$ is a (multi-)graph with even degrees

w(all edges together) $\leq \frac{3}{2} \cdot TSP_{TOUR}$



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



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

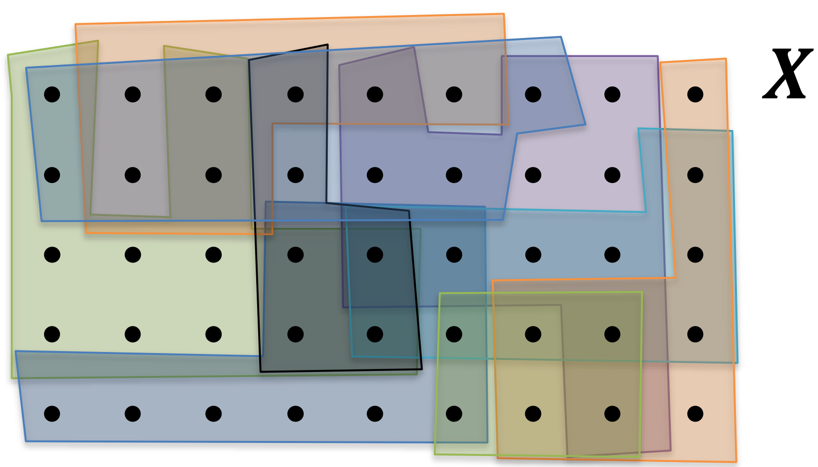
Set system

Set Cover:

- 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:



Minimum (Weighted) Set Cover

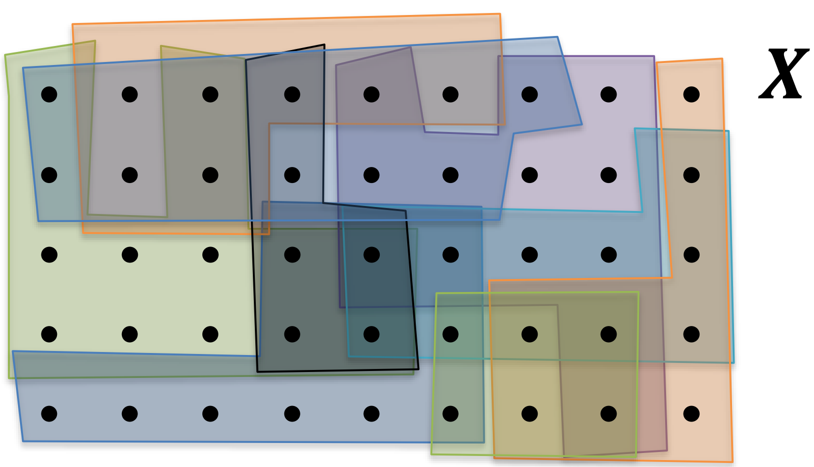
Minimum Set Cover:

- **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:

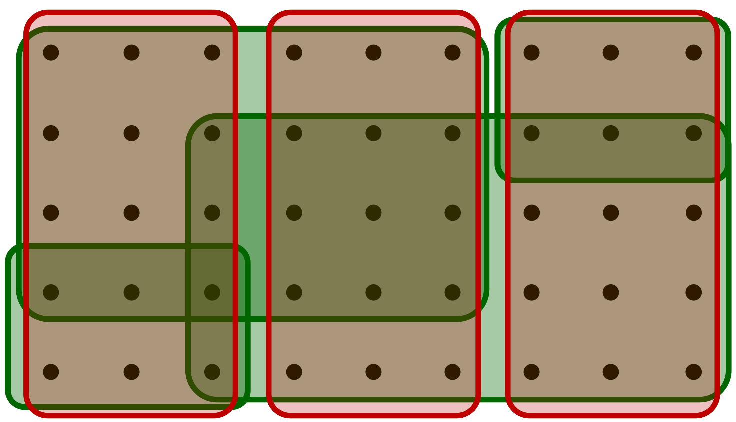


Minimum Set Cover: Greedy Algorithm

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:

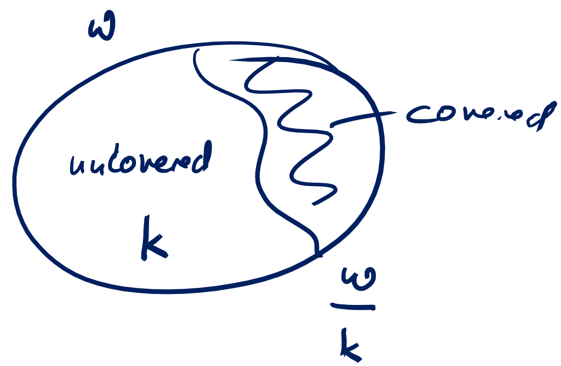


Weighted Set Cover: Greedy Algorithm

Greedy Weighted Set Cover Algorithm:

- 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):

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



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:

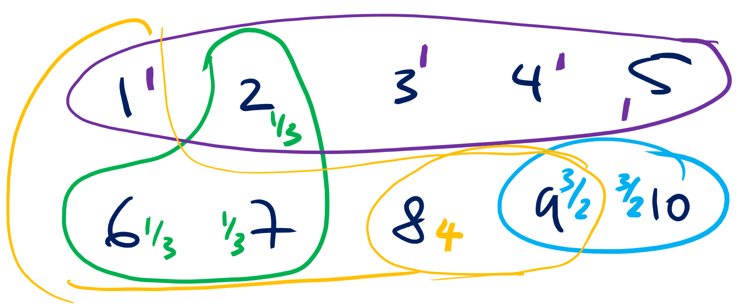
$$p(e) = \frac{w_S}{\left|S \setminus \bigcup_{T \in \mathcal{C}} T\right|}$$

Weighted Set Cover: Greedy Algorithm

Example:

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

$$\cancel{S_1 = \{1, \cancel{2}, 3, 4, 5\},} \qquad \cancel{w_{S_1} = 4}$$
$$\cancel{S_2 = \{2, 6, 7\},} \qquad w_{S_2} = 1$$
$$S_3 = \{\cancel{1}, \cancel{6}, \cancel{7}, 8, 9\}, \qquad w_{S_3} = 4$$
$$S_4 = \{\cancel{2}, \cancel{4}, \cancel{7}, 9, 10\}, \qquad w_{S_4} = 6$$
$$S_5 = \{\cancel{1}, \cancel{3}, \cancel{5}, \cancel{6}, \cancel{7}, 8, 9, 10\}, \qquad w_{S_5} = 9$$
$$S_6 = \{9, 10\}, \qquad w_{S_6} = 3$$



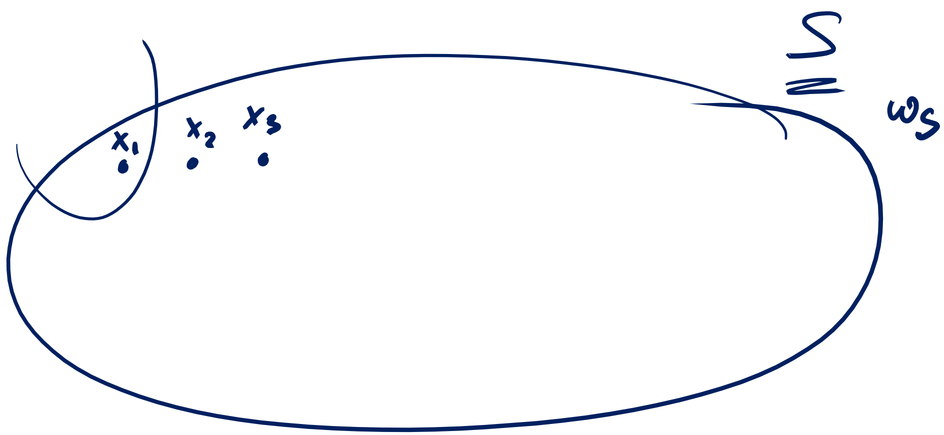
Weighted Set Cover: Greedy Algorithm

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 $p(x_i) \leq \frac{w_S}{k-i+1}$

$\frac{w_S}{k}$

$\frac{w_S}{k-1}$



$$\sum_{x \in S} p(x) \leq w_S \sum_{i=1}^{k} \frac{1}{i}$$

$$= w_S \cdot H(k)$$

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

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

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

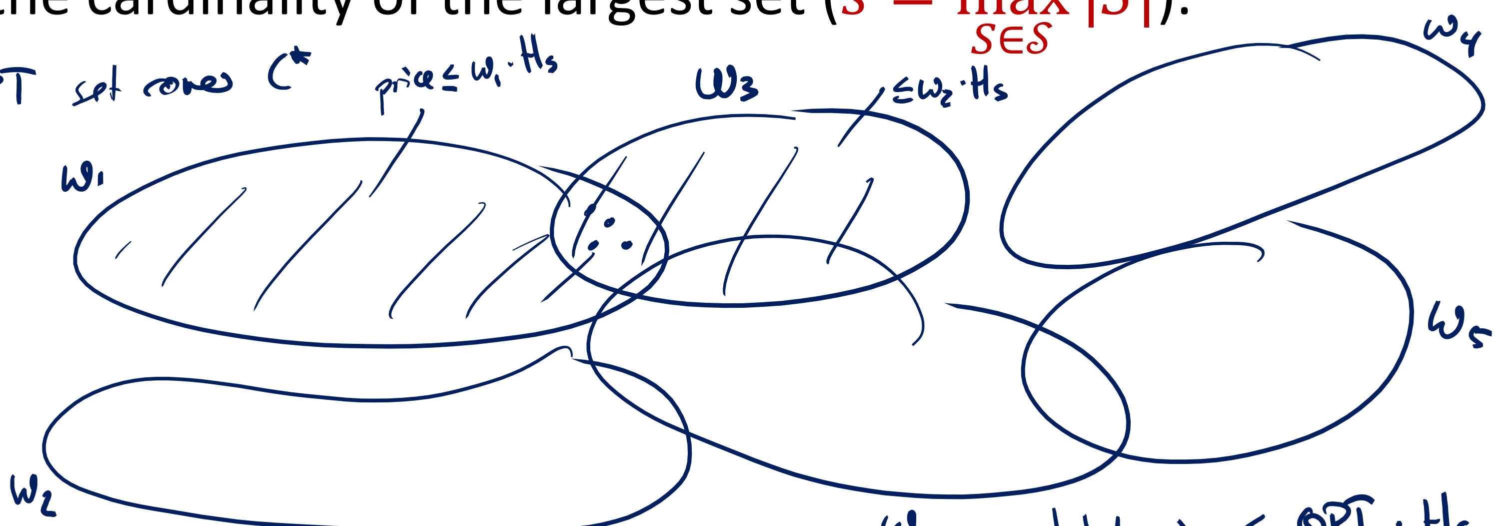
Weighted Set Cover: Greedy Algorithm

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

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

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

OPT set cover C^*



$\underline{\underline{GREEDY}}$ = total price ≤ $\underline{\underline{OPT}} \cdot \underline{\underline{H_s}}$