Chapter 8

Approximation Algorithms

Algorithm Theory
WS 2016/17

Fabian Kuhn

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:

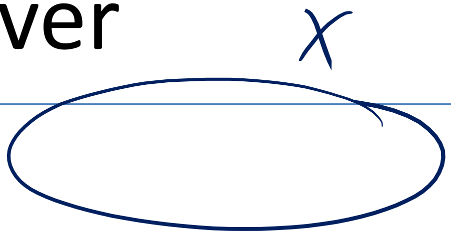
- $\mathrm{OPT} \geq 0$: optimal objective value
  $\mathrm{ALG} \geq 0$: objective value achieved by the algorithm

- **Approximation Ratio α:**

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

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

Minimum (Weighted) Set Cover



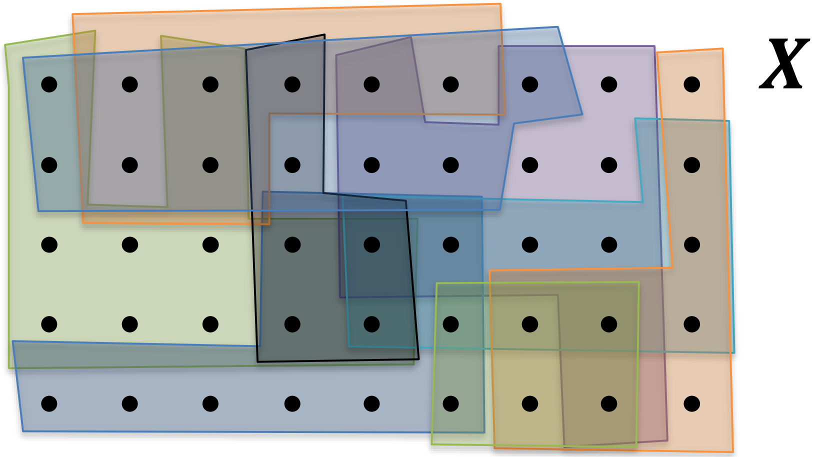
Minimum Set Cover:

- **Goal:** Find a set cover $\mathcal{C}$ of smallest possible size $\mathcal{C} \subseteq \mathcal{S} \subseteq 2^X$
  - 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:



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 $\underline{p(x)}$ 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:

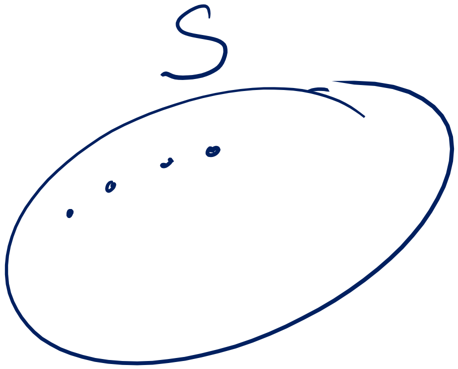
$$p(\cancel{x}) = \frac{w_S}{\left|S \setminus \bigcup_{T \in \mathcal{C}} T\right|}$$

Weighted Set Cover: Greedy Algorithm

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

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

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



$$\sum_{x \in S} p(x) \le H_{|S|}$$

H_s

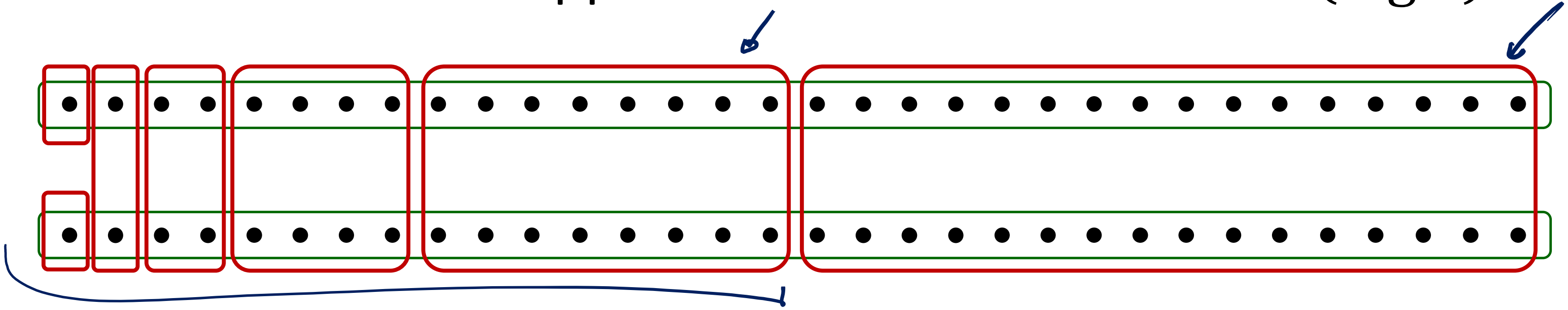
Set Cover Greedy Algorithm

Can we improve this analysis? $\ln s \;\; +1$

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 $\mathrm{P} \approx \mathrm{NP}$

$e^{O(n)} = n^{O(n/\log n)}$

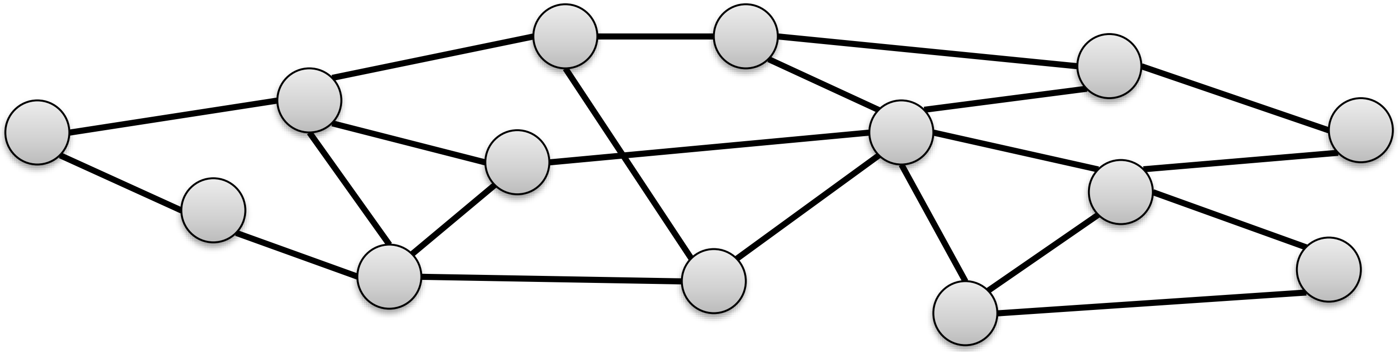
Feige showed that unless NP has deterministic $n^{O(\log\log n)}$-time algorithms, minimum set cover cannot be approximated better than by a factor $(1 - o(1)) \cdot \ln n$ in polynomial time.

- 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:

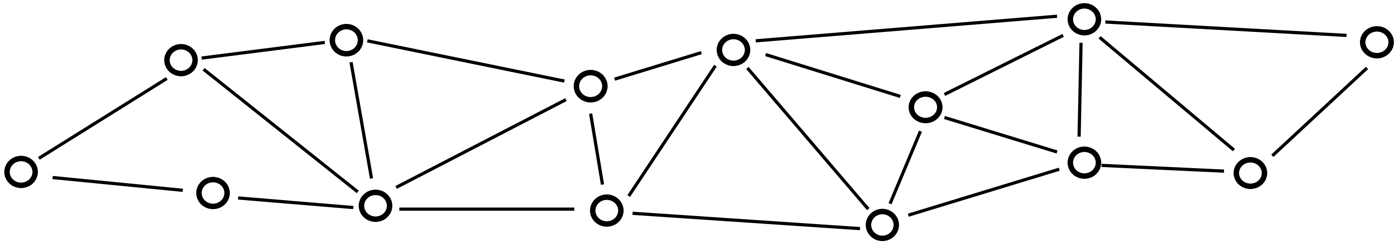
- Each node has a weight
- Find a vertex cover of minimum total weight

Set Cover: Special Cases

Dominating Set:

Δ $H_{\Delta+1}$

Given a graph $G = (V, E)$, a dominating set $\underline{S \subseteq V}$ 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

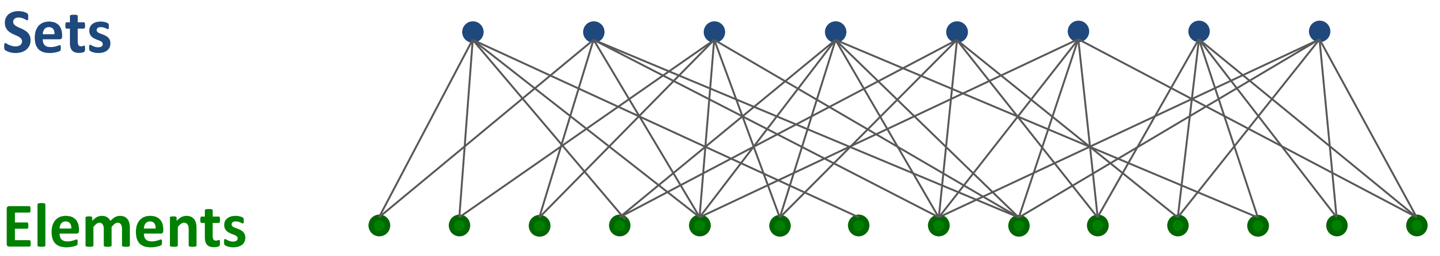
Given: Set of elements X and collection of subsets $\mathcal{S} \subseteq 2^X$

- 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 $\underline{W}$

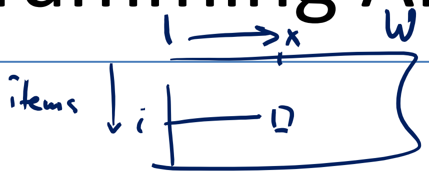
- Goal: pack items into knapsack such that total weight is at most W and total value is maximized:

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

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

- 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

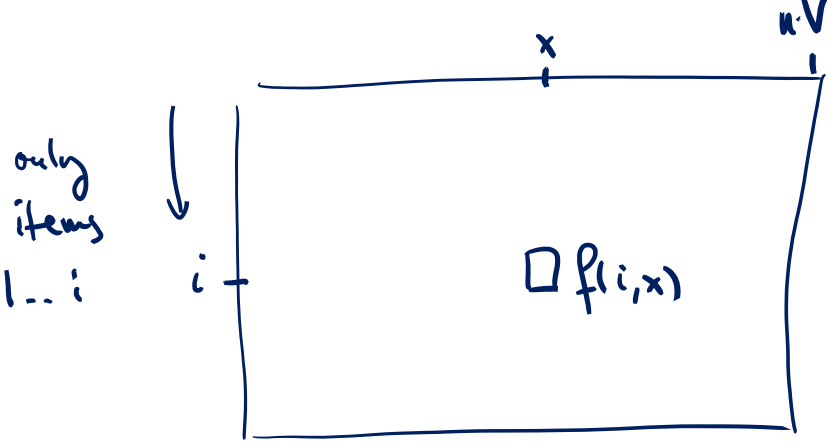
Knapsack: Dynamic Programming Alg.



We have shown:

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

$f(i,x)$: min. weight to get value exactly x



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

$$f(i,0) = 0$$

$$f(0,x) = \infty \quad (\text{for } x>0)$$

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 $O(n\underline{W})$
- If all values v_i are integers, there is another dynamic progr. algorithm that runs in time $O(n^2\underline{V})$, where V is the max. value.

Problems:

- If W and V are large, the algorithms are not polynomial in n
- If the values or weights are not integers, things are even worse (and in general, the algorithms cannot even be applied at all)

Idea:

- Can we adapt one of the algorithms to at least compute an approximate solution?

Approximation Algorithm

- The algorithm has a parameter $\varepsilon > 0$
- We assume that each item alone fits into the knapsack
- We define:

$$\underline{V} := \underline{\max_{1\le i\le n} v_i}, \qquad \forall i: \underline{\hat{v}_i} := \left\lceil \frac{v_i n}{\varepsilon V} \right\rceil, \qquad \underline{\hat{V}} := \underline{\max_{1\le i\le n} \hat{v}_i} = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

- We solve the problem with integer values $\hat{v}_i$ and weights w_i using dynamic programming in time $O(n^2 \cdot \hat{V})$
- If solution value $\underline{< V}$, we take item with value V instead

Theorem: The described algorithm runs in time $O(n^3/\varepsilon)$.

Proof:

$$\hat{V} = \max_{1\le i\le n} \hat{v}_i = \max_{1\le i\le n} \left\lceil \frac{v_i n}{\varepsilon V} \right\rceil = \left\lceil \frac{V n}{\varepsilon V} \right\rceil = \underline{\underline{\left\lceil \frac{n}{\varepsilon} \right\rceil}}$$

Approximation Algorithm

Theorem: The approximation algorithm computes a feasible solution with approximation ratio at most $1 + \varepsilon$.

Proof:

- Define the set of all feasible solutions (subsets of $[n]$)

$$\underline{\mathcal{S}} := \left\{ S \subseteq \{1, \dots, n\} : \sum_{i \in S} w_i \leq \underline{W} \right\}$$

$S \in \mathcal{S}$

$v(S) = \sum_{i \in S} v_i$

- $v(S)$: value of solution S w.r.t. values $v_1, v_2, \dots$
  $\hat{v}(S)$: value of solution S w.r.t. values $\hat{v}_1, \hat{v}_2, \dots$
- S^*: an optimal solution w.r.t. values $v_1, v_2, \dots$
  $\underline{\hat{S}}$: an optimal solution w.r.t. values $\hat{v}_1, \hat{v}_2, \dots$
- Weights are not changed at all, hence, $\hat{S}$ is a feasible solution

Approximation Algorithm

Theorem: The approximation algorithm computes a feasible solution with approximation ratio at most $1 + \varepsilon$.

Proof:

- We have

$$V \leq \underline{v(S^*)} = \sum_{i \in S^*} v_i = \max_{S \in \mathcal{S}} \sum_{i \in S} v_i\,,$$

$$\underline{\hat{v}(\hat{S})} = \sum_{i \in \hat{S}} \hat{v}_i = \max_{S \in \mathcal{S}} \sum_{S \in \mathcal{S}} \hat{v}_i$$

- Because every item fits into the knapsack, we have

$$\hat{v}_i \geq \frac{v_i n}{\varepsilon V}$$

$$\forall i \in \{1, \dots, n\}\colon\ v_i \leq V \leq \sum_{j \in S^*} v_j$$

- Also: $\hat{v}_i = \left\lceil \frac{v_i n}{\varepsilon V} \right\rceil \implies \underline{\underline{v_i \leq \frac{\varepsilon V}{n} \cdot \hat{v}_i}},$ and $\underline{\hat{v}_i \leq \frac{v_i n}{\varepsilon V} + 1}$

Approximation Algorithm $\mathcal{O}(n^3/\varepsilon)$

Theorem: The approximation algorithm computes a feasible solution with approximation ratio at most $1+\varepsilon$.

Proof: $v_i \le \frac{\varepsilon V}{n}\cdot \hat{v}_i$ $\hat{v}_i \le \frac{v_i n}{\varepsilon V} + 1$

- We have

$$\underline{\underline{v(S^*)}} = \sum_{i\in S^*} v_i \le \frac{\varepsilon V}{n}\cdot\sum_{i\in S^*}\hat{v}_i \le \frac{\varepsilon V}{n}\cdot\sum_{i\in \hat{S}}\hat{v}_i \le \frac{\varepsilon V}{n}\cdot\sum_{i\in \hat{S}}\left(1+\frac{v_i n}{\varepsilon V}\right)$$

- Therefore

$$\underline{\underline{v(S^*)}} = \sum_{i\in S^*} v_i \le \frac{\varepsilon V}{n}\cdot\left|\hat{S}\right| + \sum_{i\in \hat{S}} v_i \le \underline{\underline{\varepsilon V}} + \underline{\underline{v\left(\hat{S}\right)}} \le \varepsilon\, v(\hat{S}) + v(\hat{S})$$

- If $\underline{v\left(\hat{S}\right) \ge V}$: $\quad v(S^*) \le (1+\varepsilon)\cdot v\left(\hat{S}\right)$

- Otherwise: algorithm solution value is V and $v(\hat{S}) = V$

$$v(S^*) \le (1+\varepsilon)\cdot V = (1+\varepsilon)\, v(\hat{S})$$

Approximation Schemes

- For every parameter $\varepsilon > 0$, the knapsack algorithm computes a $(1+\varepsilon)$-approximation in time $O(n^3/\varepsilon)$.
- For every fixed ε, we therefore get a polynomial time approximation algorithm
- An algorithm that computes an $(1+\varepsilon)$-approximation for every $\varepsilon > 0$ is called an approximation scheme.
- If the running time is polynomial for every fixed ε, we say that the algorithm is a polynomial time approximation scheme (PTAS)
- If the running time is also polynomial in $1/\varepsilon$, the algorithm is a fully polynomial time approximation scheme (FPTAS)
- Thus, the described alg. is an FPTAS for the knapsack problem