



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:

- $OPT \geq 0$: optimal objective value
 $ALG \geq 0$: objective value achieved by the algorithm
- **Approximation Ratio α :**

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

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

Minimum (Weighted) Set Cover

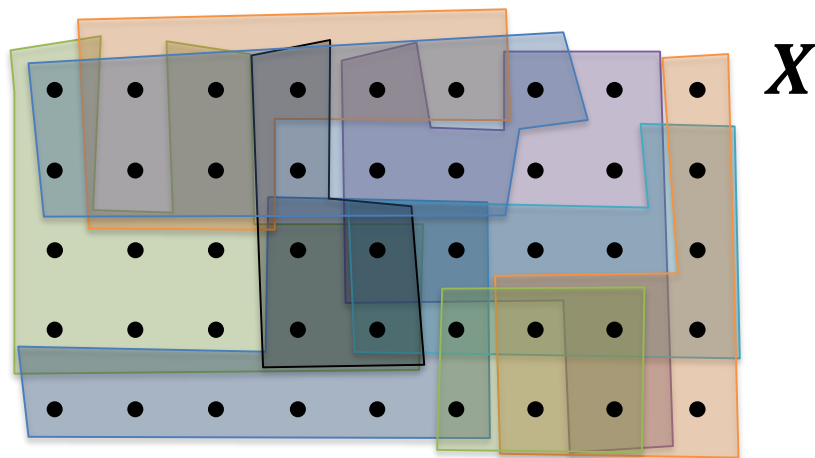
Minimum Set Cover:

- **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:



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):

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

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 :

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

Weighted Set Cover: Greedy Algorithm

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

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

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

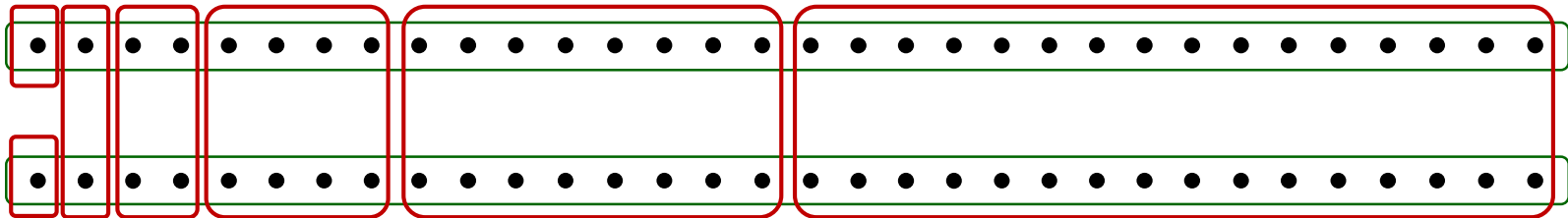
Set Cover Greedy Algorithm

Can we improve this analysis?

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

- 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 $\Omega(\log n)$...



OPT = 2

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 $P \approx NP$

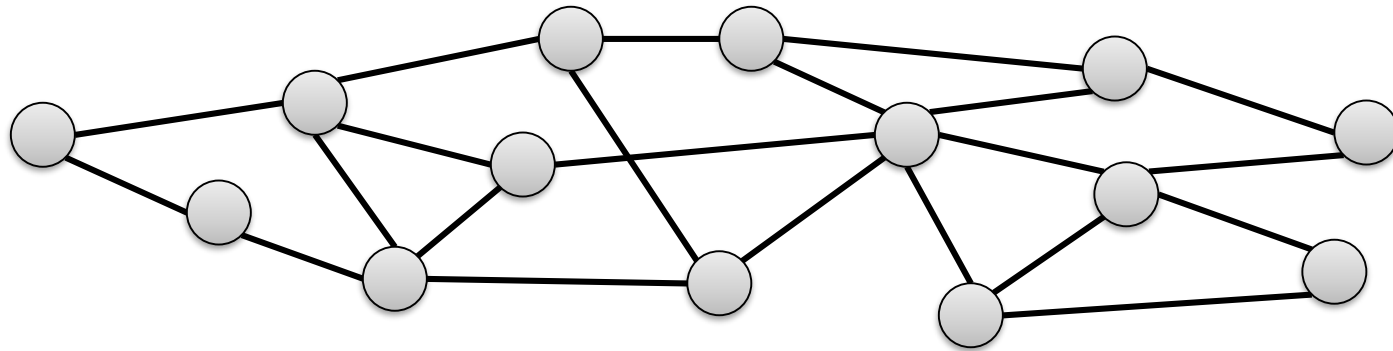
Feige showed that unless NP has deterministic $n^{O(\log \log n)}$ -time algorithms, minimum set cover cannot be approximated better than by a factor $(1 - o(1)) \cdot \ln n$ in polynomial time.

- 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

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



Minimum Vertex Cover:

- Find a vertex cover of minimum cardinality

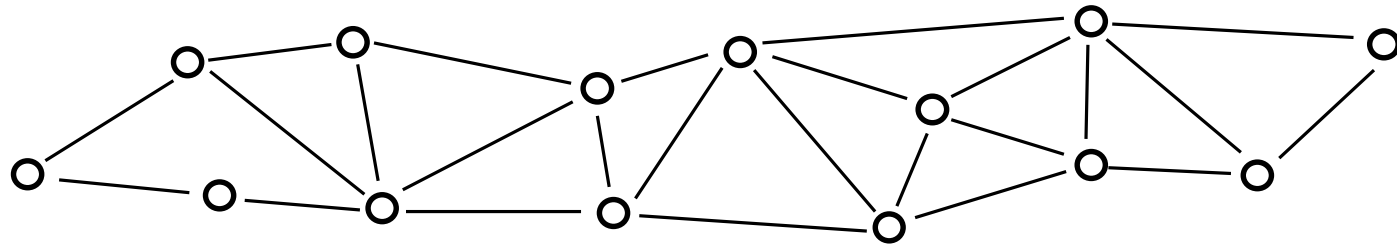
Minimum Weighted Vertex Cover:

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

Set Cover: Special Cases

Dominating Set:

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

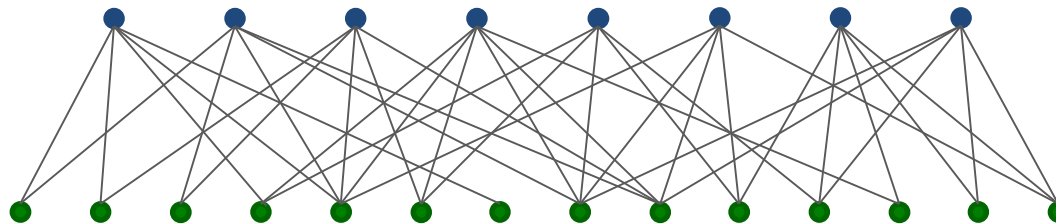
Goal: Find a min. cardinality subset $H \subseteq X$ of elements such that

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

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

Sets

Elements



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**:

$$\begin{aligned} \max \quad & \sum_{i \in S} v_i \\ \text{s. t.} \quad & S \subseteq \{1, \dots, n\} \text{ and } \sum_{i \in S} w_i \leq W \end{aligned}$$

- 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

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 programming algorithm that runs in time $O(n^2V)$, where V is the max. value.

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 programming algorithm that runs in time $O(n^2V)$, 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:

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

- 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 $< V$, we take item with value V instead

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

Proof:

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

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 $[n]$)

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

- $v(S)$: 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
 \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

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

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

- Because every item fits into the knapsack, we have

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

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

Approximation Algorithm

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

Proof:

- We have

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

- Therefore

$$v(S^*) = \sum_{i \in S^*} v_i \leq \frac{\varepsilon V}{n} \cdot |\hat{S}| + \sum_{i \in \hat{S}} v_i \leq \varepsilon V + v(\hat{S})$$

- If $v(\hat{S}) \geq V$: $v(S^*) \leq (1 + \varepsilon) \cdot v(\hat{S})$
- Otherwise: algorithm solution value is V and $v(S^*) \leq (1 + \varepsilon) \cdot V$

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