Chapter 8
Approximation Algorithms

Algorithm Theory
WS 2018/19

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:**

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

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

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

Optimal Assignment:

M_1: 3 | 4 | 2 | 3 | 1

M_2: 6 | 4 | 3

M_3: 4 | 2 | 1 | 5

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

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

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^*:

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

- Second lower bound on optimal makespan T^*:

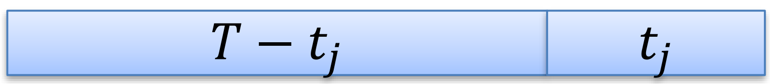
$$T^* \geq \max_{1 \leq i \leq n} t_i$$

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:

$\boldsymbol{M_i}$**:** 

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

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:

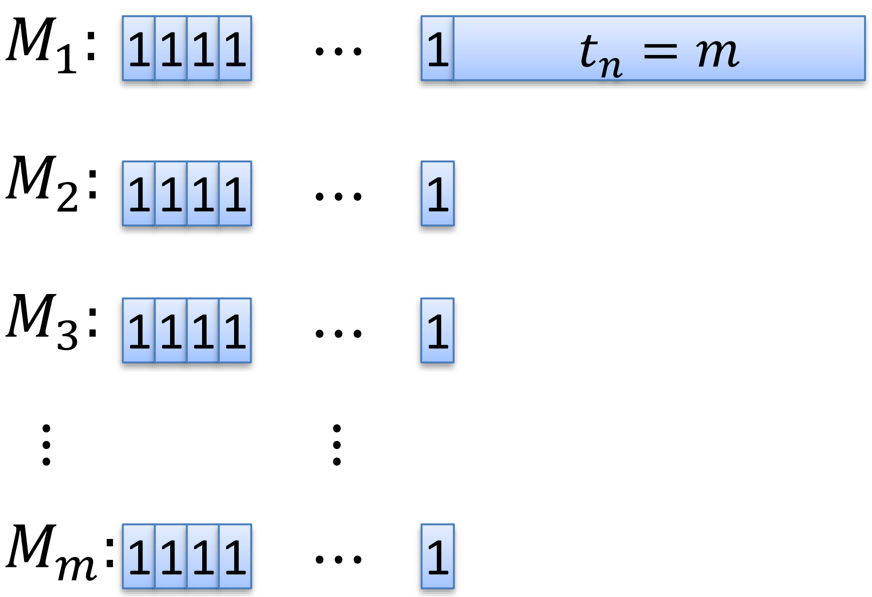
- For all machines M_k: load $T_k \geq T - t_j$

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:



Improving Greedy

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

Idea: assign large jobs first

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 $T = T^*$
- Otherwise, we have $j \geq m + 1$
  - The first m jobs are assigned to m distinct machines

Set Cover

Input:

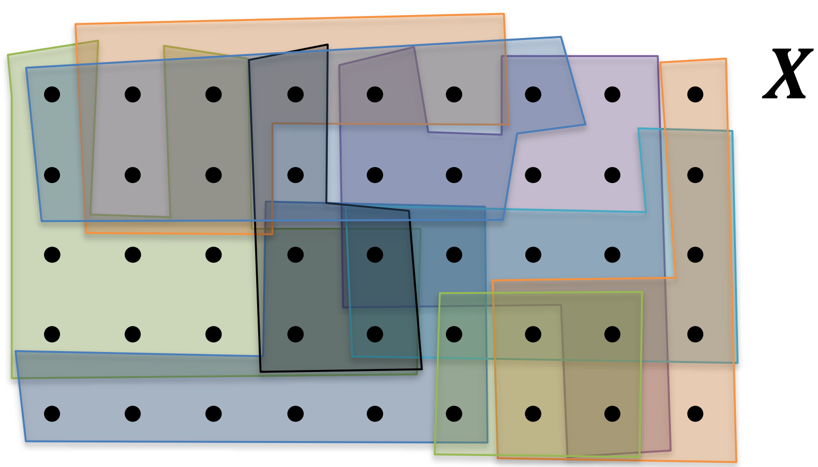
- A set of elements X and a collection $\mathcal{S}$ of subsets X, i.e., $\mathcal{S} \subseteq 2^X$
  - such that $\bigcup_{S \in \mathcal{S}} S = X$

Set Cover:

- 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

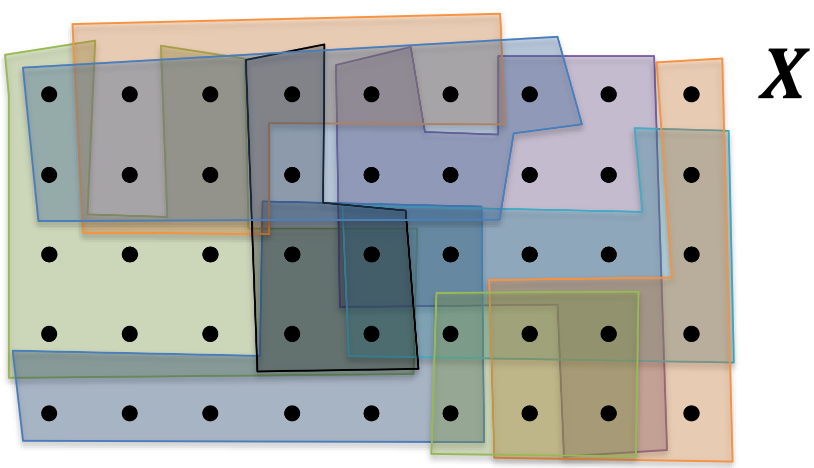
Minimum Set Cover:

- **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:

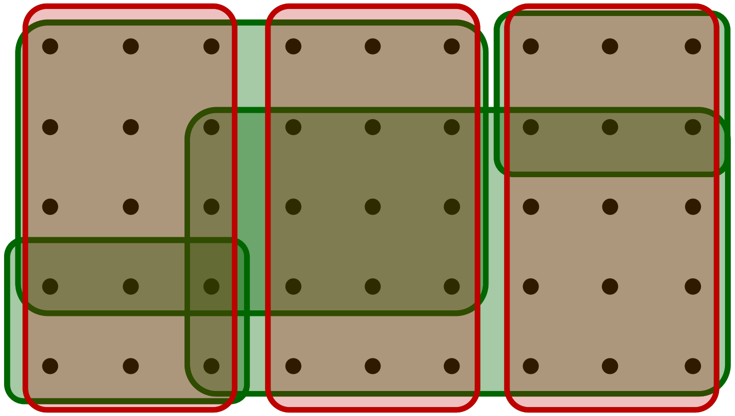


Minimum Set Cover: Greedy Algorithm

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 $\mathcal{S} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$

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

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

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

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

Weighted Set Cover: Greedy Algorithm

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

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

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 ($s = \max_{S \in \mathcal{S}} |S|$).

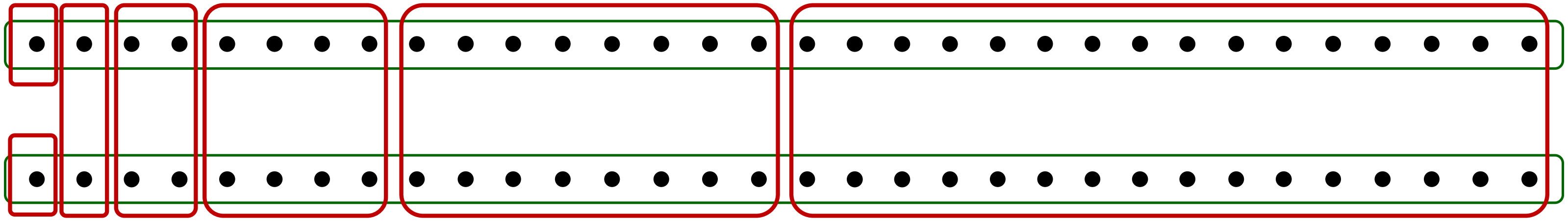
Set Cover Greedy Algorithm

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

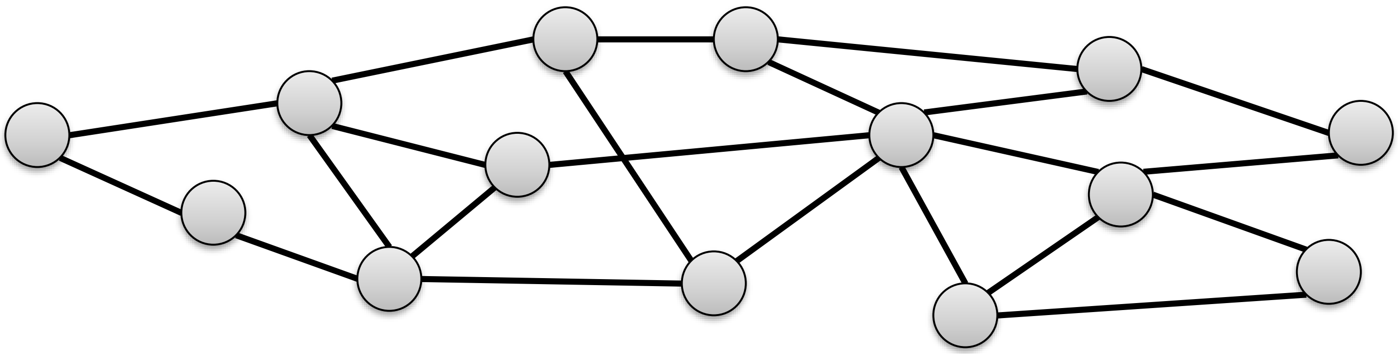
Feige showed that unless NP has deterministic $n^{O(\log\log n)}$-time algorithms, minimum set cover cannot be approximated better than by a factor $\big(1 - o(1)\big) \cdot \ln n$ in polynomial time.

- 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

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



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

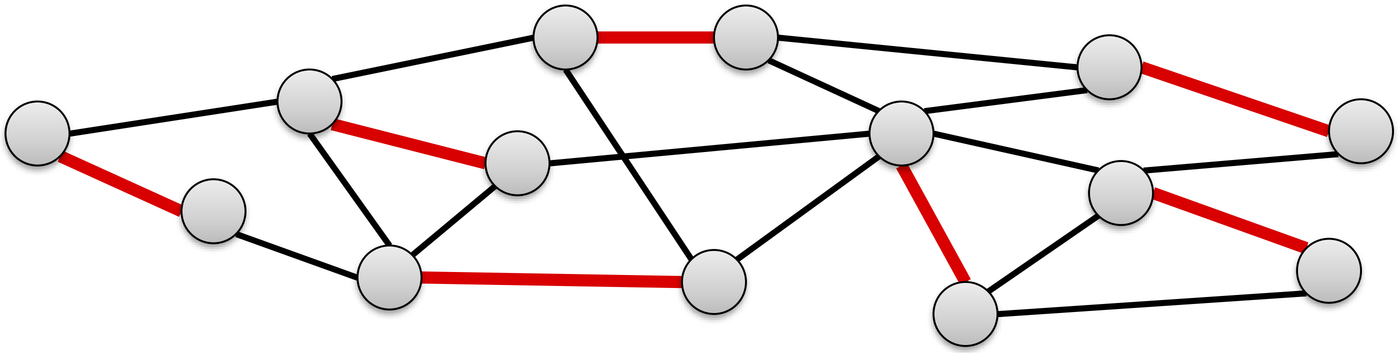
Vertex Cover vs Matching

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



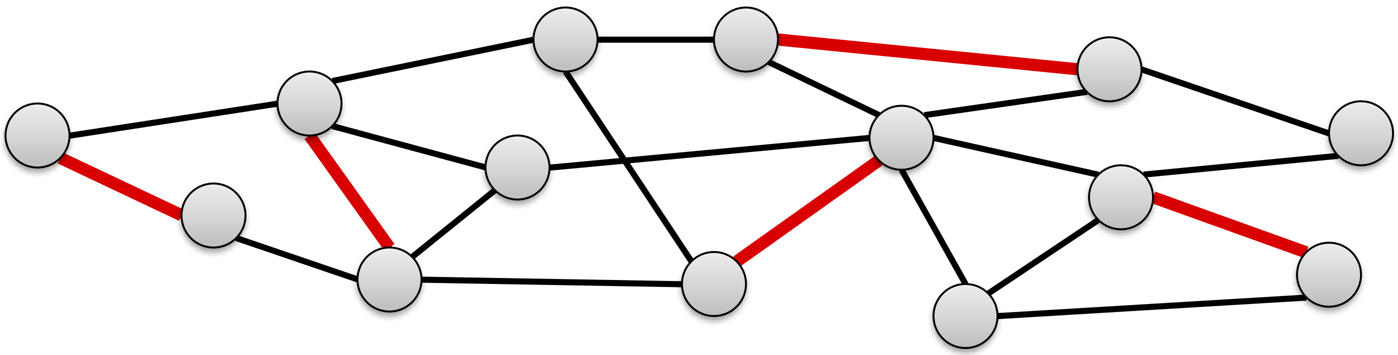
Vertex Cover vs Matching

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 $|S| = 2|M|$.

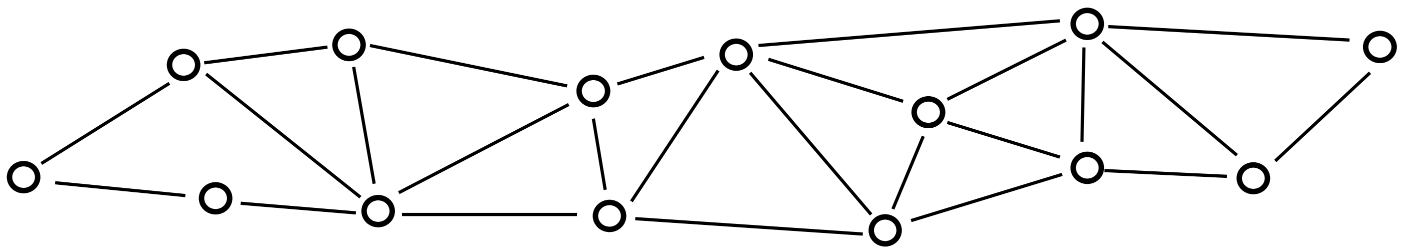
Maximal Matching Approximation

Theorem: 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