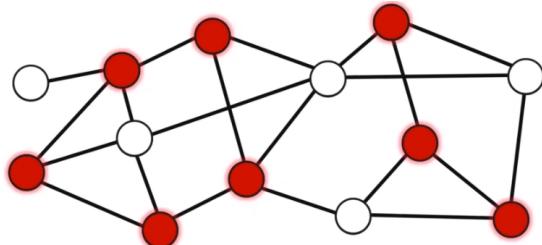


Overview

- close to optimal solutions
 - provable guarantees on the solution quality
 - often a simple greedy algorithm
- Approximation ratio $\rho(n)$** of an algorithm A: if for any n -sized input the algorithm A produces a solution with value C such that
- $\frac{C}{OPT} \leq \rho(n)$ for a **minimization problem**, and
 - $\frac{C}{OPT} \geq \rho(n)$ for a **maximization problem**.
-
- **Minimization problem:** $\rho(n) \geq 1$. **Maximization problem:** $\rho(n) \leq 1$.

Minimum Vertex Cover



A vertex cover of an undirected graph G is a set S of vertices such that every edge in $E(G)$ is incident to at least one vertex in S . In other words, for every edge (u,v) in G , at least one of the vertices u or v is in the set S .

- **Goal:** Compute a vertex cover of **minimum size**.
- NP-complete
- “vertex greedy” algorithm

Input: Graph $G = (V, E)$

Output: Vertex cover C

$C = \text{empty set}$

$F = E(G)$

while F is not empty:

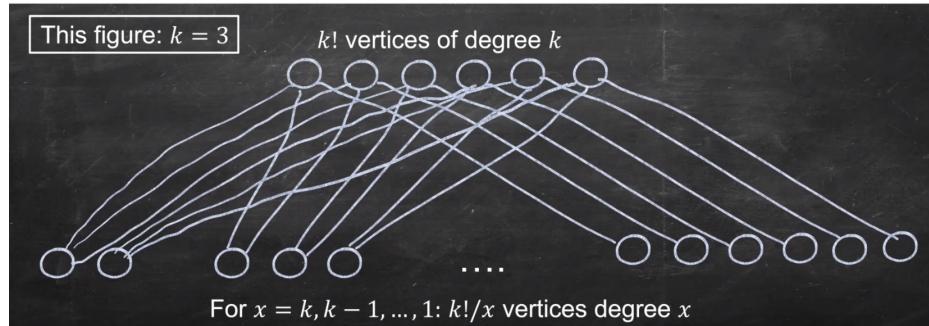
 choose largest degree vertex v in $G = (V - C, F)$

 add v to C

 remove all edges incident on v from F

return C

Theorem: Greedily adding the largest degree vertex (in the graph induced by uncovered edges) is not a constant factor approximation for MVC.



- “edge greedy” algorithm

- ApproximateVertexCover

Input: Graph $G = (V, E)$

Output: Vertex cover C

$C = \text{empty set}$

$F = E(G)$

while F is not empty:

choose any edge (u, v) from F

add u and v to C

remove all edges incident on u and v from F

return C

- better approximation guarantee than “greedy vertex”

- must not be always better

- 2-approximation

F : the set of edges that our greedy algorithm picked. F is a matching in G . Why?
 $C = V(F)$ be the computed vertex cover.

- Clearly C is a vertex cover, as we continue adding vertices until all edges are covered.
- We have $|C| = 2|F|$ (no overlap, as F is a matching in G)

Let C_{OPT} be any optimal solution to MVC.

$|C_{OPT}| \geq |F|$, because C_{OPT} has to cover every edge, including all edges in F .

* But no vertex of G can cover more than a single edge of F , as F is a matching.

$$\Rightarrow |C| = 2|F| \leq 2|C_{OPT}|.$$

*

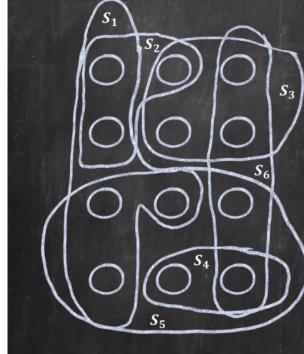
Set Cover

Input:

- **Universe** $X = \{x_1, \dots, x_n\}$ of n elements
- **Collection of Sets** $S = \{S_1, \dots, S_k\}$, each $S_i \subseteq X$

Set Cover: a collection of sets (indices) $I \subseteq \{1, \dots, k\}$
s.t. all elements are covered, i.e.,

$$X \subseteq \bigcup_{i \in I} S_i$$



- **Goal:** Select a minimum size set cover (minimize $|I|$).
- NP-complete
- greedy algorithm

GreedySetCover(Universe, Sets):

$$I = \{\}$$

while X is not empty:

$$\text{MaxSet} = \operatorname{argmax}(\text{Set in Sets}, |\text{Set} \cap X|)$$

$$I = I \cup \{\text{MaxSet}\}$$

$$X = X - \text{MaxSet}$$

return I

- add the set with a maximum number of not yet covered elements
- repeat until all elements are covered \Rightarrow correct
- approximation factor

Let C_{OPT} be an optimal cover, let $t = |C_{OPT}|$
Let X_k be the elements in iteration k . $X_0 = X$

Claim: For all $0 \leq k$, the set X_k can be covered with t sets.

Proof: The original set X can be covered with t sets, so the same is true for $X_k \subseteq X$

In step k , there exists a set that covers at least $|X_k|/t$ elements (pigeonhole principle)

\Rightarrow in step k the greedy algorithm is going to pick a set of size at least $|X_k|/t$

$$\text{For all } k, \text{ we have } |X_{k+1}| \leq \left(1 - \frac{1}{t}\right) |X_k|$$

$$\boxed{\text{By induction for all } k \geq 0 : |X_k| \leq \left(1 - \frac{1}{t}\right)^k |X_0| = \left(1 - \frac{1}{t}\right)^k \cdot |X|}$$

*

When do we stop? How many sets do we choose?

We stop when $X_k = \emptyset$ ($|X_k| < 1$), chosen at most k sets

For $k^* = t \cdot (\lfloor \log |X| \rfloor + 1)$, we obtain

$$|X_{k^*}| \leq \left(1 - \frac{1}{t}\right)^{k^*} \cdot |X| \leq e^{-\frac{k^*}{t}} \cdot |X| = e^{-\frac{k^*}{t} + \log |X|} \leq e^{-1} < 1$$

* **At most $k^* = t \cdot (\lfloor \log |X| \rfloor + 1)$ sets, so we have a $(\lfloor \log |X| \rfloor + 1)$ -approximation.**

$$\blacklozenge 1 - x \leq e^{-x}$$

Partition Problem

Input:

n positive integers s_1, \dots, s_n

Goal:

Partition the set of integers (integer indices) into two sets $A, B \subseteq \{1, \dots, n\}$ to minimize

$$\max \left\{ \sum_{i \in A} s_i, \sum_{i \in B} s_i \right\}$$

- - balance both partitions
- NP-complete
 - Dynamic programming: $O(n \cdot \sum s_i)$ (does not contradict NP-hardness)
 - Brute force by trying all combinations: $O(2^n)$
- Polynomial time approximation scheme

Polynomial time approximation scheme (PTAS): For each $\epsilon > 0$:

- Computes a $(1 \pm \epsilon)$ -approximation
- Runtime is polynomial in input for fixed ϵ .

Fix: $m = \left\lceil \frac{1}{\epsilon} \right\rceil - 1$

Order from largest to smallest $s_1 \geq s_2 \geq \dots \geq s_n$.

Compute an **optimal solution (A, B)** for the first m elements.

Greedy for the rest:

```

For i=m+1,...,n
    If weight(A) ≤ weight(B) :
        add i to A,
    else
        add i to B.

```

Runtime: $O(n \cdot \log n + 2^{\frac{1}{\epsilon}+1} + n)$

- approximation factor

Proof: Wlog assume at the end we have $\text{weight}(A) \geq \text{weight}(B)$
 Let s_k be the last element added to A.

Look at the snapshot after adding k:

We only need to prove $w(A) = w(A_k) \leq (1 + \epsilon)OPT$

but we don't know OPT, how can we compare to it?

* perfect balancing as lower bound

$$L = \frac{1}{2} \sum s_i \leq OPT$$

• ↑

Case 1 (k was added in the first phase) $\Rightarrow k \leq m$

After the first phase, A was optimal for the smaller problem of adding the first m elements. Later, we never add anything to A. We obtain:

$$w(A) = w(A_k) \leq OPT(s_1, \dots, s_m) \leq OPT(s_1, \dots, s_n)$$

* \rightarrow Approximation ratio in this case is 1

Case 2 (k was added in the second phase) $\Rightarrow k > m$

Claim: $s_k \leq 2L/m$

Proof: As $s_1 \geq s_2 \geq \dots \geq s_k$ we get:

$$2L \geq \sum_{i=1}^k s_i \geq \sum_{i=1}^k s_k \geq m \cdot s_k, \text{ claim follows by dividing by } m.$$

$W(A) \leq w(B_{k-1}) + s_k$ (we added k to A because of $w(A_{k-1}) \leq w(B_{k-1})$, and never changed A afterwards)

$$W(A) \leq w(B_{k-1}) + s_k \leq w(B) + s_k = 2L - w(A) + s_k$$

* $W(A) \leq L + s_k \leq (1 + \epsilon)L \leq (1 + \epsilon)OPT$

◆ $s_k \leq \frac{2L}{m}$ since all integers are positive

◆ last line of approximation works because of

■ $m = \lceil \frac{1}{\epsilon} \rceil - 1$

Theorem: For any $\epsilon > 0$ there exists an algorithm that computes a $(1 + \epsilon)$ -

* approximation of the partition problem in time $O(n \cdot \log n + 2^{\frac{1}{\epsilon}+1} + n)$.