ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Special Topics on Algorithms
FALL 2022

Vertex Cover, Set Cover

Vangelis Markakis-George Zois

markakis@gmail.com

georzois@gmail.com

Vertex Cover and Set Cover (Greedy Approximation Algorithms)

Vertex Cover (VC)

Recall the (optimization) version:

<u>VERTEX COVER (VC)</u>:

I: A graph G = (V,E)

Q: Find a cover $C \subseteq V$ of minimum size, i.e., a set $C \subseteq V$, s.t. $\forall$ (u, v) $\in$ E, either u $\in$ C or v $\in$ C (or both)

Weighted version:

<u>WEIGHTED VERTEX COVER (WVC)</u>:

I: A graph G = (V,E), and a weight w(u) for every vertex $u \in V$

Q: Find a subset $C \subseteq V$ covering all edges of G, s.t. $W = \sum_{u \in C} w(u)$ is minimized

Many different approximation techniques have been "tested" on vertex cover

Vertex Cover (VC)

We will focus first on the unweighted version

Natural greedy algorithms: start picking nodes according to some criterion until all edges are covered

1st approach:

Greedy-any-node

C := $\emptyset$;

while E $\neq \emptyset$ do

{ choose arbitrarily a vertex u $\in$ V;

delete u and its incident edges from G;

Add u to C }

What is the approximation ratio this algorithm ?

Vertex Cover (VC)

2nd natural approach: start picking nodes and at each step choose the node with the maximum degree

Greedy-best-node

$C := \emptyset$;

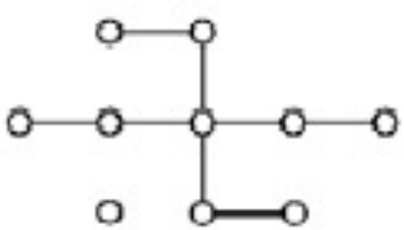
while $E \neq \emptyset$ do

{ choose the vertex $u \in V$ with the largest degree; (break ties arbitrarily)

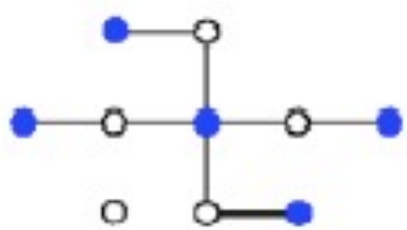
delete u and its incident edges from G;

Add u to C }

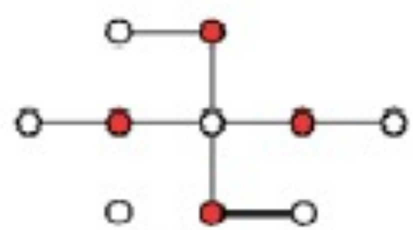
This is not Optimal!



A graph instance



A vertex cover of size 5 obtained by the greedy algorithm.



A vertex cover of size 4 optimal solution!!

Vertex Cover (VC)

2^{nd} natural approach: start picking nodes and at each step choose the node with the maximum degree

Greedy-best-node

C := $\emptyset$;

while E $\neq \emptyset$ do

{ choose the vertex u $\in$ V with the largest degree; (break ties arbitrarily)

delete u and its incident edges from G;

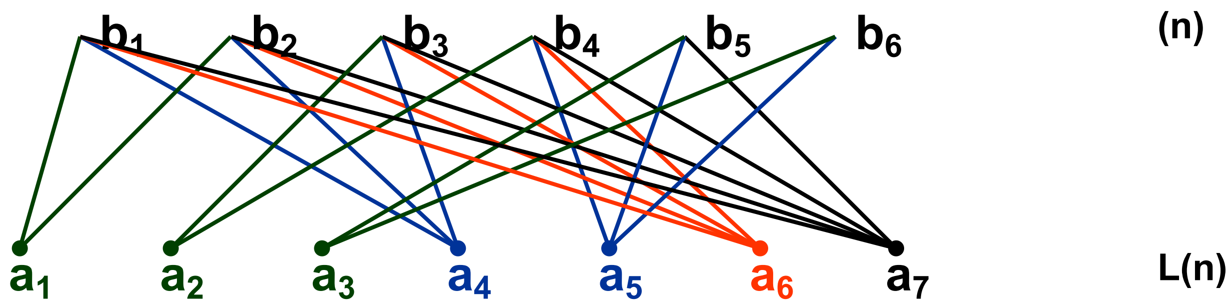
Add u to C }

Theorem: Greedy-best-node is an O(log n)-approximation algorithm

(see slides 25-26 for a proof)

Vertex Cover (VC)

The O(logn) ratio of Greedy-best-node is tight, i.e. the algorithm cannot achieve a better ratio.



- Partition b-nodes into pairs, triples, quadtuples,…,(n-1)tuples
- Connect the nodes in each i-tuple above with a new a-node

$$L(n) = \sum_{j=2}^{n-1} \left\lfloor \frac{n}{j} \right\rfloor = n \sum_{j=2}^{n-1} \left\lfloor \frac{1}{j} \right\rfloor \leq n \sum_{j=1}^{n} \frac{1}{j} = nO(\log n)$$

Vertex Cover (VC)

The O(logn) ratio of Greedy-best-node is tight

C = $\{a_7,a_6,a_5,a_4,a_4,a_2,a_1\}$
OPT = $\{b_1,b_2,b_3,b_4,b_5,b_6\}$

$$\frac{C}{OPT} = \frac{L(n)}{n} = O(\log n)$$

n	6	10	30	100	1000	…
C/OPT	2.17	2.6	3.67	4.8	7	…

Greedy-best-node is not a constant approximation algorithm.

Vertex Cover (VC)

Detour on matchings

Consider a graph G = (V, E)

Definition: A matching M is a collection of edges $M \subseteq E$, such that no 2 edges share a common vertex

Given a matching M, a vertex u is called *matched* if there exists an edge $e \in M$ such that e has u as one of its endpoints

Vertex Cover (VC)

Detour on matchings

Types of matchings we are interested in:

- Maximal matching: find a matching where no more edges can be added
- Maximum matching: find a matching with the maximum possible number of edges
- Perfect matching: find a matching where every vertex is matched (if one exists)
- Maximum weight matching: given a weighted graph, find a matching with maximum possible total weight
- Minimum weight perfect matching: given a weighted graph, find a perfect matching with minimum cost

All the above problems can be solved in polynomial time (several algorithms and publications over the last decades)

Vertex Cover (VC)

A different approach:

- We will resort to matching
- Let M be any matching in the graph
- Observation: OPT ≥ |M|
  - The optimal solution needs at least one vertex to cover each of the matched edges
- But we cannot just pick any matching, since it may not be a cover

Matching-based VC

C = ∅;

Find a maximal matching M;

For every (u, v) ∈ M, add both u and v to C

Output C

Theorem: Matching-based VC is a 2-approximation algorithm

Vertex Cover (VC)

Is it easy to find a maximal matching?
Trivial! Keep adding edges until it is not feasible to add more

A way to implement the maximal matching based algorithm

Greedy-any-edge

C := $\emptyset$;

while E $\neq \emptyset$ do

{ choose arbitrarily an edge (u,v) $\in$ E ;

Add u and v to C;

delete u and v and their incident edges from G;

}

The edges selected by the algorithm form a maximal matching (no 2 edges share a common vertex)

Note: In contrast to greedy-any-node, greedy-any-edge achieves a constant factor approximation

Vertex Cover (VC)

Theorem: Matching-based VC is a 2-approximation algorithm

Proof:

a) The solution – say C - returned by the algorithm is a vertex cover

- Suppose not
- Then there is an uncovered edge (u, v)
- But then we could add this edge to the matching M
- Contradiction with the fact that M is a maximal matching

b) 2-approximation ratio

- Let M be the set of edges selected by Greedy-any-edge
- Each selected edge adds two vertices to C: $|C| = 2\,|M|$
  - No two edges in M share a vertex (since M is a maximal matching)
  - Edges incident to the endpoints of a selected edge are removed

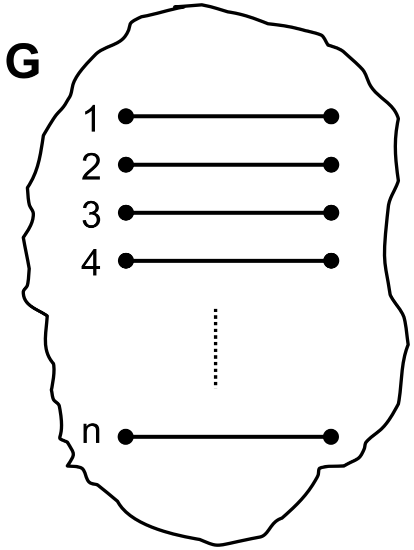
Cost of the solution: $|C| = 2\,|M| \leq 2\,OPT$ (by the observation)

Hence a 2-approximation

Vertex Cover (VC)

Tightness of the 2-approximation

Example:



C = 2n

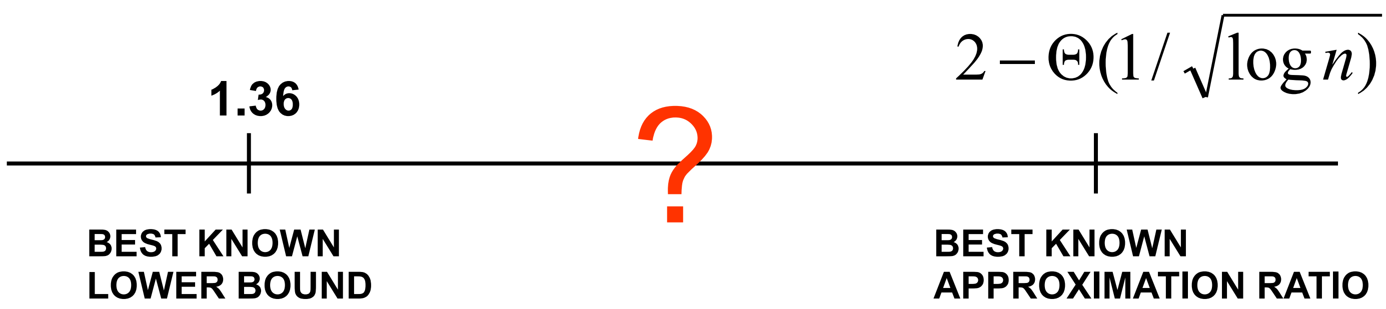
OPT = n

Vertex Cover (VC)

Greedy-any-edge is almost the best known for VC

Is there a better approximation algorithm ?

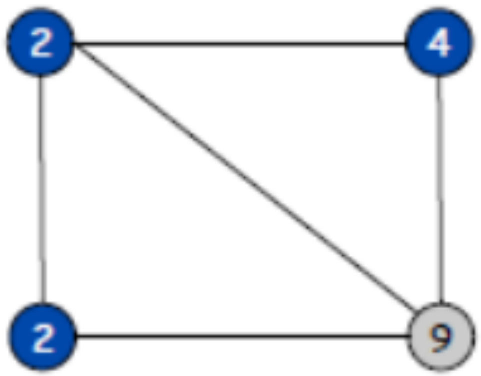
We know a lower bound of 1.36 on the approximation factor for VC, i.e.,
Unless P=NP, VC cannot be approximated with a ratio smaller than 1.36

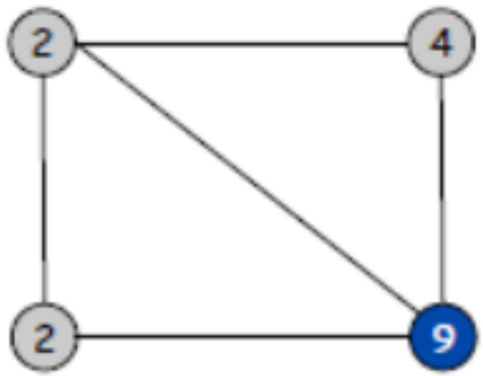


Big open problem!!

Weighted Vertex Cover (WVC)

Find a minimum weight subset C $\subseteq$ V covering all edges of G





weight = 2 + 2 + 4

Cover

weight = 9

No Cover

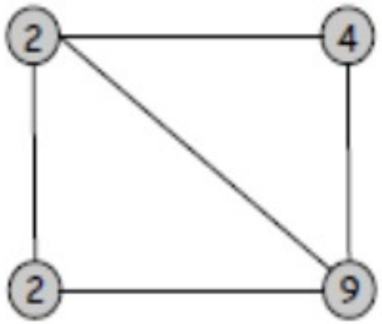
What is the min weighted cover here?

Weighted Vertex Cover (WVC)

Pricing method. Each edge must be covered by some vertex i. Edge e pays price $p_e \geq 0$ to use vertex i.

Fairness. Edges incident to vertex i should pay $\leq w_i$ in total.

for each vertex i: $\sum_{e=(i,j)} p_e \leq w_i$



Claim. For any vertex cover S and any fair prices p_e: $\sum_e p_e \leq w(S)$.

Proof. ▪

$$\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S).$$

each edge e covered by at least one node in S

sum fairness inequalities for each node in S

Weighted Vertex Cover (WVC)

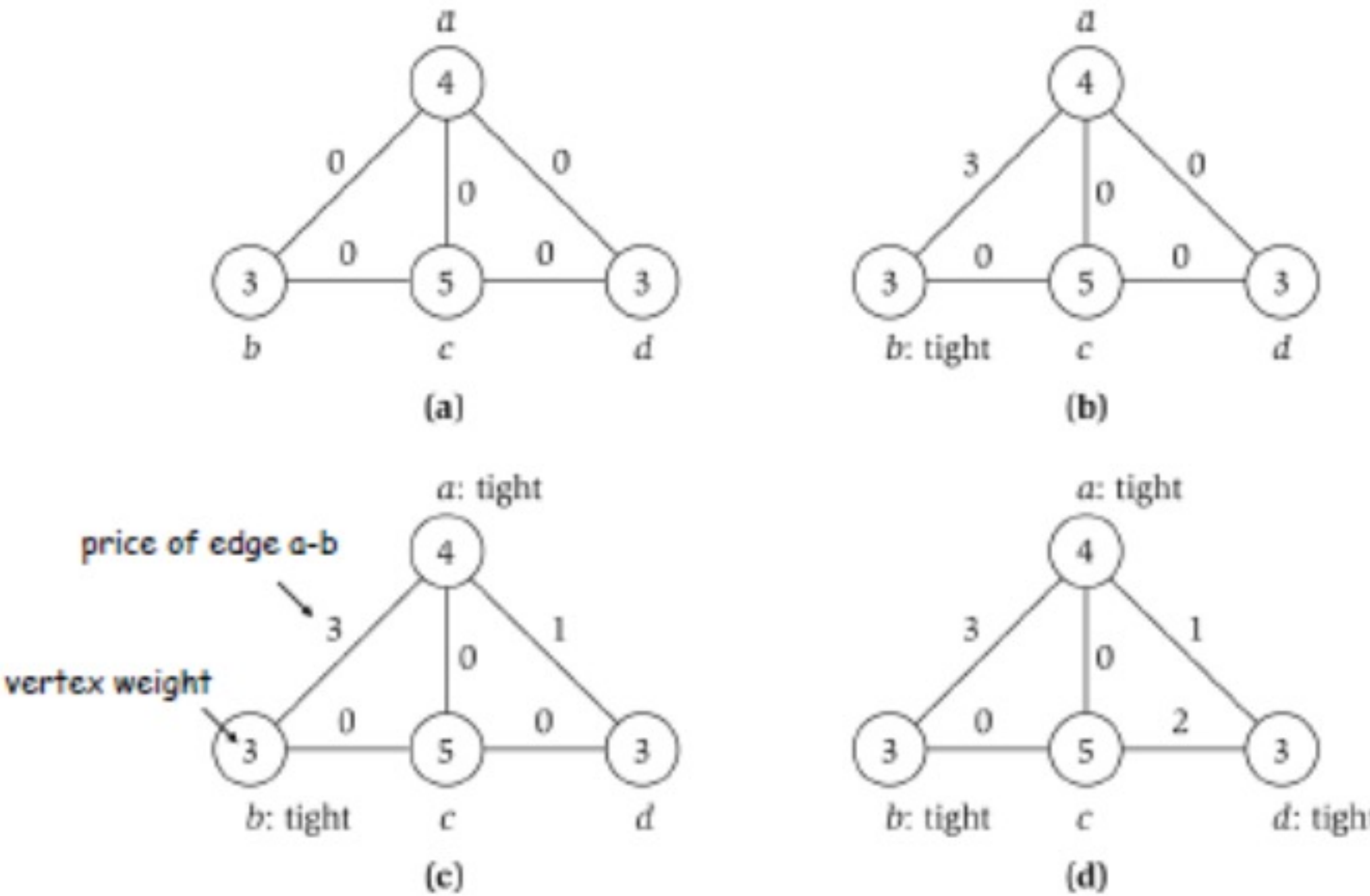
```
Weighted-Vertex-Cover-Approx(G, w) {
   foreach e in E
      pe = 0

   while (∃ edge i-j such that neither i nor j are tight)
      select such an edge e
      increase pe without violating fairness
   }

   S ← set of all tight nodes
   return S
}
```

$$\sum_{e=(i,j)} p_e = w_i$$

Weighted Vertex Cover (WVC)



Weighted Vertex Cover (WVC)

Theorem. Pricing method is a 2-approximation.

Pf.

- Algorithm terminates since at least one new node becomes tight after each iteration of while loop.

- Let S = set of all tight nodes upon termination of algorithm. S is a vertex cover: if some edge i-j is uncovered, then neither i nor j is tight. But then while loop would not terminate.

- Let S^* be optimal vertex cover. We show $w(S) \le 2w(S^*)$.

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \le \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e \le 2w(S^*).$$

↑ all nodes in S are tight

↑ $S \subseteq V$, prices ≥ 0

↑ each edge counted twice

↑ fairness lemma

Tightness ?

Set Cover

SET COVER (SC):

I: a set U of n elements

a family F = $\{S_1, S_2, \ldots, S_m\}$ of subsets of U

Q: Find a minimum size subset $C \subseteq F$ covering all elements of U, i.e.:

$$\bigcup_{S_i \in C} S_i = U \text{ and } |C| \text{ is minimized}$$

Weighted version:

WEIGHTED SET COVER (WSC):

I: a set U of n elements

a family F = $\{S_1, S_2, \ldots, S_m\}$ of subsets of U

a weight $w(S_i)$ for each set S_i

Q: Find a minimum weight subset $C \subseteq F$ covering all elements of U, i.e.,

$$\bigcup_{S_i \in C} S_i = U \text{ and } W = \sum_{S_i \in C} w(S_i) \text{ is minimized}$$

Set Cover vs Vertex Cover

(WEIGHTED) VERTEX COVER

I: (weighted) graph $G=(V,E)$

Q: find $C \subseteq V$ s.t.

C covers E and

C is of min size (cost)

(WEIGHTED) SET COVER

I: $U=E$ (i.e., we need to cover the edges)

$|F| = |V|$,

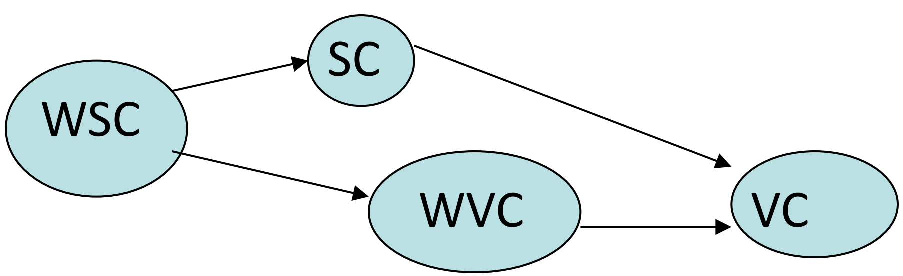
One set per vertex: $S_u = \{(u,v) \mid (u,v) \in E\}$

(in the weighted case: weight of S_u is $w(u)$)

Q: find $C \subseteq F$ s.t.

C covers U and

C is of min size (cost)



Hence, all WSC, SC, and WVC problems are NP-complete

as generalizations of VC

Set Cover-Example

Input: Set U of n elements and m subsets, $S_1,S_2,\ldots,S_m$ of U.
Question: Find the minimum number of subsets covering U.

Example:

U: a set of n cities

Consider that the ministry of education is planing to place/build new schools such that no city is more than 30km away from a school.

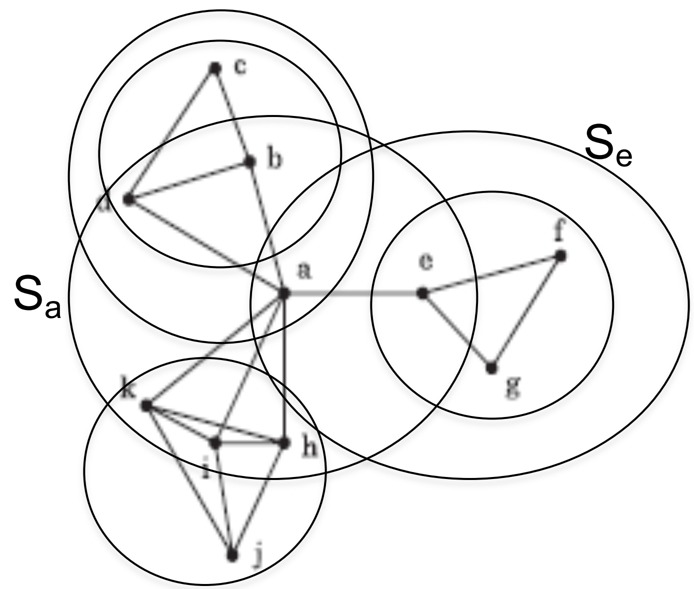
Subsets: For each city i, S_i is the subset of cities which are at most 30km away from i.

Find which is the minimum number of schools to be built?

Set Cover-Example

Input: Set U of n elements and m subsets, $S_1,S_2,\ldots,S_m$ of U.
Question: Find the minimum number of subsets covering U.

U: 11 cities

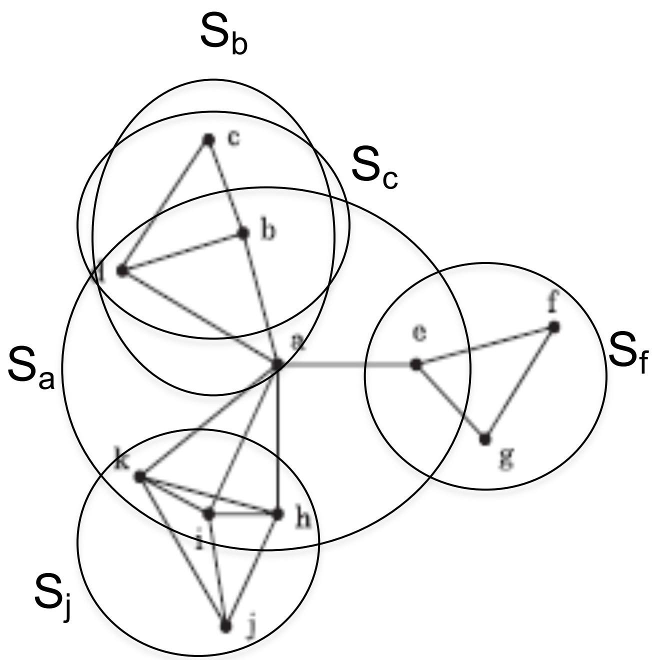


S_a, S_b, …,S_k : cities
which are away at most 30km
from each candidate location

Set Cover-Greedy Algorithm

Greedy Idea: While there are uncovered cities:

Choose the subset with the greatest number of uncovered cities-elements.



S_i: S_a, S_b, …,S_k, cities which are away at most 30km

Greedy Solution:

1. S_a
2. S_f
3. S_c
4. S_j

C=4 (# συνόλων)

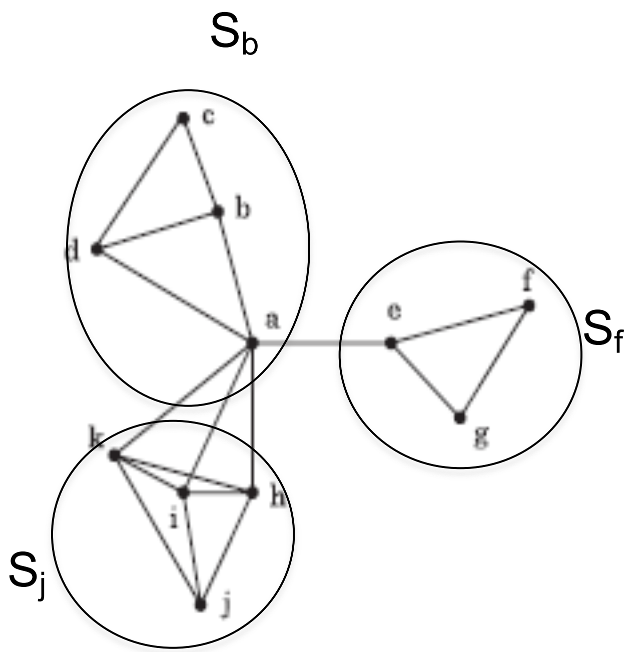
OPT=?

Is the greedy optimal?

Set Cover-Greedy Algorithm

Greedy Idea: While there are uncovered cities:

Choose the subset with the greatest number of uncovered elements.



S_i: S_a, S_b, …,S_k, cities which are away at most 30km

Greedy is not optimal

1. S_b
2. S_j
3. S_f

OPT=3

- What is the approximation ratio of Greedy?
- We will analyze a generalization of the greedy algorithm for Weighted Set Cover

Weighted Set Cover (WSC)

In a similar spirit as for (greedy best-node) Vertex Cover:

<u>Greedy-best-set</u>

C := ∅ ;

while C ≠ U do

{ choose the best set S;

remove S from F;

C := C U S;

W(C) = W(C)+W(S);

}

C: elements covered before iteration i

S: Set chosen at iteration i

Weighted Set Cover (WSC)

Q: What does "best set" mean ?

- S covers |S-C| new elements
- Covering those elements costs *w(S)*
- Every element x $\in$ S-C essentially costs

$$\frac{w(S)}{|S-C|} = p(x)$$ = "cost-effectiveness" of S

Best set: the set with the smallest cost-effectiveness

Weighted Set Cover (WSC)

Analysis of Greedy-best-set

Let $x_{1,} x_{2,} \dots, x_{k,} \dots, x_n$ be the order in which the elements of U are covered

$S_{1,} S_{2,} \dots \; S_{i,} \; \dots$ be the order in which sets are chosen by the algorithm

Suppose set S_i covers element x_k

Claim: $p(x_k) \le \dfrac{OPT}{n-k+1}$

$C = \bigcup_{j=1}^{i-1} S_j$ elements covered by iterations 1,2,…,i-1

- U-C: uncovered elements before iteration i
- |U-C| ≥ n-k+1, since element x_k is covered in iteration i

Weighted Set Cover (WSC)

- These elements of U-C are covered in the optimal solution by some sets at a cost of at most OPT
- Among them there must be one set with cost-effectiveness at most

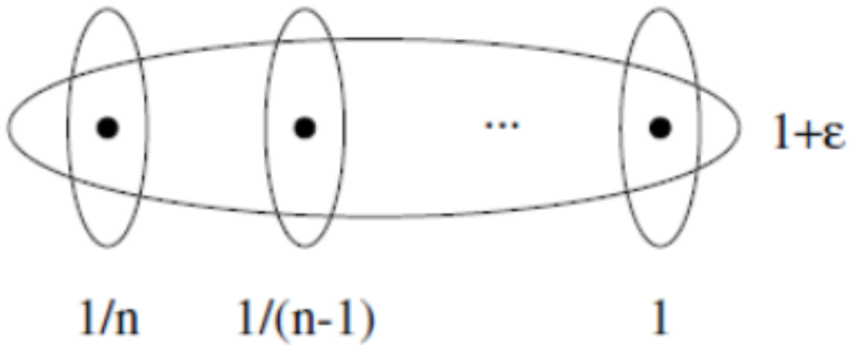
$$\le \frac{OPT}{|U-C|} \le \frac{OPT}{n-k+1}$$

- the set S_i was picked by the algorithm as the set with the smallest cost-effectiveness at that moment (and it covered x_k)
- that is $p(x_k) \le \frac{OPT}{n-k+1}$

$$W = \sum_{k=1}^{n} p(x_k) \le \sum_{k=1}^{n} \frac{OPT}{n-k+1} = OPT \sum_{i=1}^{n} \frac{1}{k} = OPT \cdot H_n = O(\log n) OPT$$

Weighted Set Cover (WSC)

Tightness



The greedy algorithm outputs the n singleton sets with total cost

$$W = \frac{1}{n} + \frac{1}{n-1} + \cdots + 1 = H_n$$

The optimal cover takes only the other set of cost $1+\varepsilon$

Weighted Set Cover (WSC)

Q: Is there a better approximation ?

- Several failed attempts over the years
- [Lund, Yannakakis '94]: There can be no logn/2 = 0.72ln(n)-approximation
- [Feige '98] There can be no (1-ε)ln(n) approximation
  - *Proof based on the PCP theorem*
- Complexity assumption for these results: NP cannot be solved in time $n^{O(\log\log n)}$