Approximation algorithms 1
Set cover, vertex cover

CS240 Spring 2022

Rui Fan

Approximation algorithms

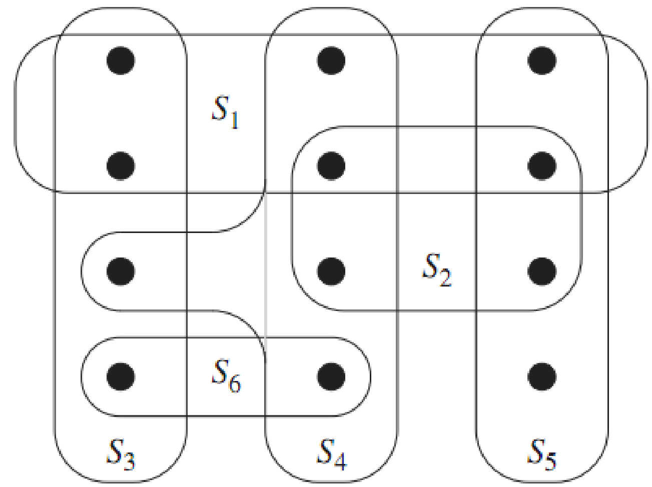
- Up to now, most of our algorithms have been exact. I.e. they find an optimal solution.
- But there are many problems for which we don't know how to find an optimal solution.
  - A key example is NP-complete problems. We don't know efficient algorithms for any NPC problem.
- Many such problems are important in practice. What do we do?
- If we can't get find the best answer, let's try for good enough.
- Approximation algorithms find an approximately optimal answer.

Approximation ratio

- Let X be a maximization problem. Let A be an algorithm for X.
- Let $\alpha>1$ be a constant.
- A is an α-approximation algorithm for X if A always returns an answer that's at least $1/\alpha$ times the optimal.
  - Ex If X is max-flow, A is a 2-approx algorithm if it always returns a flow that's at least ½ the optimal.
  - The closer α is to 1, the better the approximation.
- If X is a minimization problem, A is an α-approximation algorithm for X if it always returns an answer that's at most α times larger than the optimal.
  - Ex If X is min-cut, A is a 2-approx algorithm if it always returns a cut that's at most 2 times the size of the optimal.
  - Again, the closer α is to 1, the better the approximation.

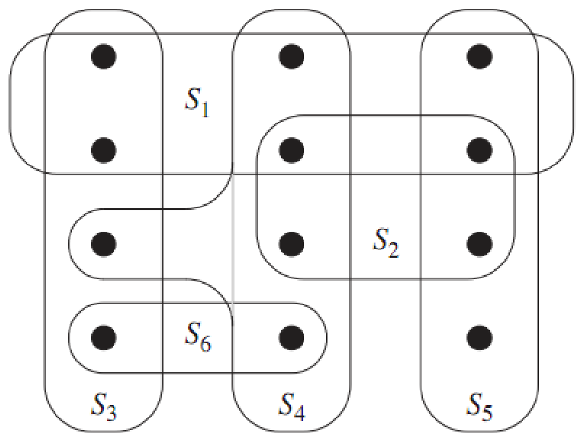
Coverings

- Suppose there's a set of teachers, and each can teach a certain set of classes.
  - Let S_i be the set of classes teach i can teach.
- The entire set of classes is X.
- We want to pick the minimum set of teachers to teach all the classes.
  - Let T be set of teachers we pick.
  - We want $\bigcup_{i \in T} S_i = X$, and T to be the smallest possible.



Set covering

- Input A collection F of sets. Each set has a cost. The union of all the sets is X.
- Output A subset G of F, whose union is X.
- Goal Minimize the total cost of the sets in G.



If all sets have same cost, S_3, S_4 and S_5 is a min cost set cover of X.

- Minimum cost set cover is NP-complete.
- We'll see a ln(n)-approximation algorithm, where n=|X|.

A greedy approximation alg

- A natural greedy heuristic is to choose sets which cover points most cheaply.
  - For each set, let c be its cost, and m be the number of points it covers.
  - We want to use the set with the smallest c/m value, because this is the cheapest way to cover some new points.
- After we pick this set, remove all the points it covers. Then we consider the per unit cost of the remaining sets and again choose the cheapest.

A greedy approximation alg

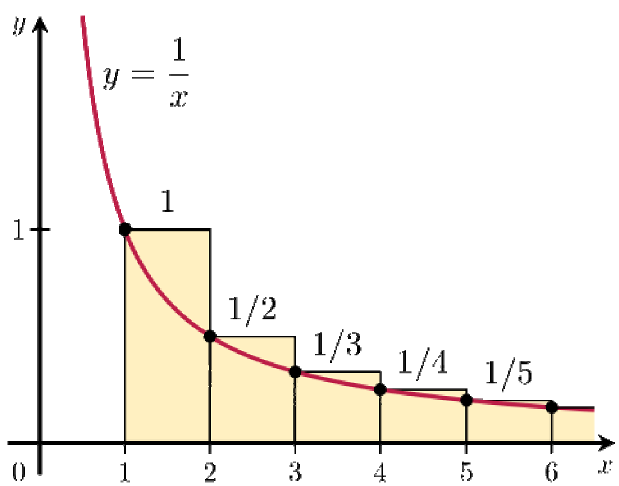
- F is the entire collection of sets. The union of these sets is X.
- Each set S in F has a cost cost(S).
- U is the set of elements of X we haven't covered yet.
- C is the set cover we eventually output.

- $U = X$
- $C = \varnothing$
- while $U \neq \varnothing$
  - choose $S \in F\text{-}C$ with min $|\text{cost}(S)|/|S \cap U|$
  - $C = C \cup \{S\}$
  - $U = U - S$
- output C

Per unit cost to cover new elements.

Proof of correctness

- We always output a set cover, because the while loop continues till X is covered.
- We'll prove the approximation ratio is at most $1+1/2+1/3+...+1/n \approx \ln(n)$.
  - If the min cost of a set cover is V, our set cover costs at most ln(n)*V.
- The basic plan is to bound the cost of the set cover the algorithm outputs using the "average cost" per element.



Proof of correctness

- Order the sets in C by when they're added to C, earliest set first.
  - Let the order be $S_1, S_2, \ldots, S_m$.
- Cost of the set cover is $L = \Sigma_i \text{cost}(S_i)$.
- Order the elements in X by when they're added, earliest element first.
  - Let the order be $e_1, e_2, \ldots, e_n$.
  - So, the first few e's are added by S_1, the next few added by S_2, etc.
  - Every element in X is in the list, because C covers X.

Proof of correctness

- Let n_i be the number of new elements S_i covers.
  - So, n_i is the number of elements in S_i, but not in $S_1,\ldots,S_{i-1}$.
- Divide the cost of S_i evenly among the new elements it covers.
  - If e is newly covered by S_i, then cost(e)= $\text{cost}(S_i)/n_i$.
- $\Sigma_k \text{cost}(e_k) = \Sigma_i n_i * \text{cost}(S_i)/n_i = \Sigma_i \text{cost}(S_i) = L$.
  - Every element is covered by some S_i, and S_i covers n_i new elements.
- We'll prove $\text{cost}(e_k) \leq \text{OPT}/(n-k+1)$, for any k.
- Suppose this is true, then

$$L = \Sigma_k \text{cost}(e_k) \leq \Sigma_k \text{OPT}/(n-k+1) \approx \ln(n)*\text{OPT}$$

The per element cost

- Let's focus on some element e_k, and let S_j be the set which covers e_k for the first time.
- Let $C_1,\ldots,C_r$ be the sets in an optimal cover, each of which covers some elements of $U=\{e_k,e_{k+1},e_{k+2},\ldots,e_n\}$.
  - Let $n'_1,\ldots,n'_r$ be the number of elements of U which $C_1,\ldots,C_r$ cover.
- Obs 1 $\Sigma_i\, n'_i \geq n-k+1$.
  - Because $C_1,\ldots,C_r$ cover U.
- Obs 2 $\Sigma_i\, \mathrm{cost}(C_i) \leq OPT$.
  - Because $C_1,\ldots,C_r$ are a subset of an optimal cover, which has cost OPT.

The per element cost

- Obs 3 None of $C_1,\ldots,C_r$ are among $S_1,\ldots,S_{j-1}$.
  - If some C_i is among $S_1,\ldots,S_{j-1}$, then since C_i covers some e in U, e would be covered by $\{S_1,\ldots,S_{j-1}\}$. So, e would be among the first k-1 elements covered. Contradiction.
- Obs 4 There exists some C_i among $C_1,\ldots,C_r$ with $\mathrm{cost}(C_i)/n'_i \leq OPT/(n-k+1)$.
  - If every C_i in $C_1,\ldots,C_r$ has $\mathrm{cost}(C_i)/n'_i > OPT/(n-k+1)$, then

$OPT \geq \Sigma_i \mathrm{cost}(C_i) = \Sigma_i n'_i * \mathrm{cost}(C_i)/n'_i >$

$\Sigma_i n'_i * OPT/(n-k+1) \geq OPT/(n-k+1)\ \Sigma_i n'_i \geq$

$OPT/(n-k+1)*(n-k+1) = OPT.$

Contradiction.

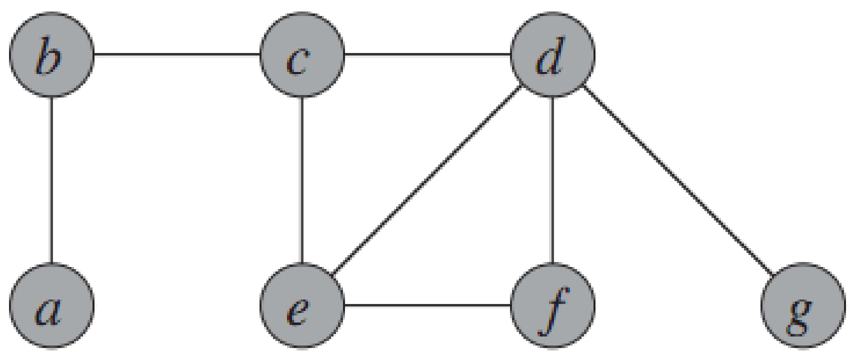
Proof of approximation ratio

- Lemma $cost(S_j)/n_j \le OPT/(n-k+1)$.
- Proof When choosing S_j, the only sets the algorithm is not allowed to choose are $S_1,\ldots,S_{j-1}$.
  - By obs 3, $C_1,\ldots,C_r$ aren't in $S_1,\ldots,S_{j-1}$.
  - By obs 4, there's some C_i in $C_1,\ldots,C_r$, with $cost(C_i)/n'_i \le OPT/(n-k+1)$.
  - S_j was chosen so that $cost(S_j)/n_j$ is min among all sets not in $S_1,\ldots,S_{j-1}$.
  - So $cost(S_j)/n_j \le cost(C_i)/n'_i \le OPT/(n-k+1)$.
- Since $cost(S_j)/n_j = cost(e_k)$, we have $cost(e_k) \le OPT/(n-k+1)$.
- The approx ratio follows because

$$L = \Sigma_k \, cost(e_k) = \Sigma_k \, OPT/(n-k+1) \approx \ln(n)*OPT$$

Vertex cover

- Input A graph with vertices V and edges E.
- Output A subset V' of the vertices, so that every edge in E touches some vertex in V'.
- Goal Make |V'| as small as possible.



source: *Introduction to Algorithms*, Cormen et al.

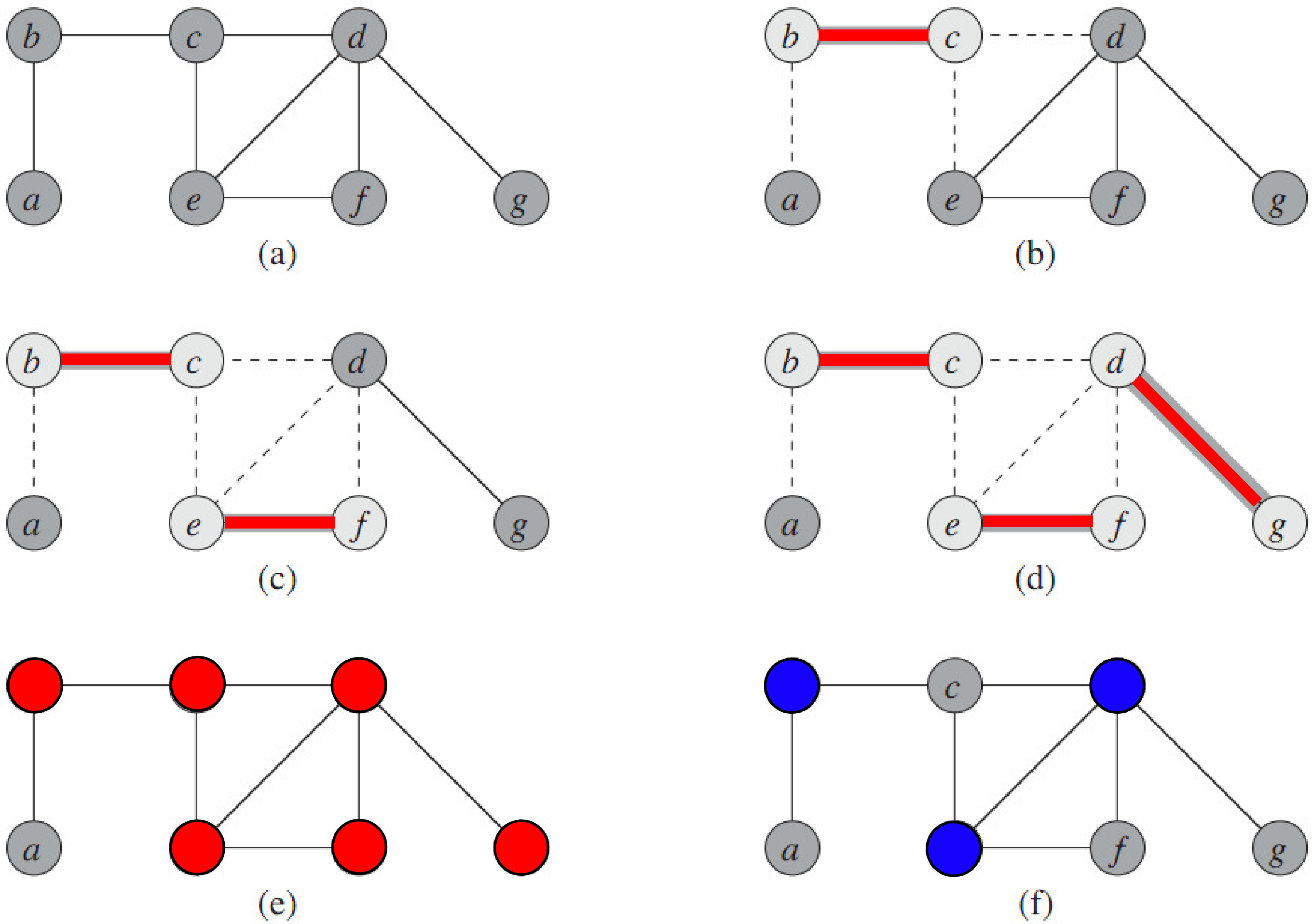
- Finding the minimum vertex cover is NP-complete.
- Vertex cover is a special case of (unweighted) set cover, where each element (edge) can be covered by at most two sets (vertices).
- We'll see a simple 2 approximation for this problem.

A vertex cover algorithm

- Initially, let D be all the edges in the graph, and C be the empty set.
  - C is our eventual vertex cover.
- Repeat as long as there are edge left in D.
  - Take any edge (u,v) in D.
  - Add {u,v} to C.
  - Remove all the edges adjacent to u or v from D.
- Output C as the vertex cover.

Example

source: CLRS



Algorithm's vertex cover

Optimal vertex cover

Proof of correctness

- The output is certainly a vertex cover.
  - In each iteration, we only take out edges that get covered.
  - We keep adding vertices till all edges are covered.
- Now, we show it's a 2 approximation.
- Let C* be an optimal vertex cover.
- Let A be the set of edges the algorithm picked.

Proof of Correctness

- None of the edges in A touch each other.
  - Each time we pick an edge, we remove all adjacent edges.
- So each vertex in C* covers at most one edge in A.
  - The edges covered by a vertex all touch each other.
- Every edge in A is covered by a vertex in C*.
  - Because C* is a vertex cover.
- So $|C^*| \geq |A|$.
- The number of vertices the algorithm uses is $2|A|$.
  - If alg picks edge (u,v), it uses {u,v} in the cover.
- So (# vertices alg uses) / (# vertices in opt cover) = $2|A| / |C^*| \leq 2|A| / |A| = 2$.