Lecture 21	Approximation Algorithms	L21.1 6.046 05/09/2017

Today

- Vertex Cover
- Set Cover
- Partition
- Linear Programming Approximation Algs.

Reading: CLRS Chpt 35, except § 35.5

Why do we need Approximation Algorithms?

Want to solve:

① hard problems (NP-hard)

② using fast algorithms (polynomial time)

③ to obtain exact solutions (correct)

We can obtain solutions satisfying any two of these conditions, but not all three (currently).

- Much of this subject & 6.006 focuses on ② + ③
- To achieve ① + ②, we use approximation algorithms
  - run in polynomial time
  - allow suboptimal solutions, but with bounds on suboptimality.

When proving NP-hardness and NP-completeness, we were especially interested in decision versions of problems; here we are concerned with optimization versions.

Formalism. Given an optimization problem of size n,

let C^* = "cost" of optimal soln (not running time but, e.g., tour length)

C = "cost" of approx. soln

then

ratio bound, $\rho(n)$: $\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n)$, $\forall n$

minimization ↑ ↑ maximization

and we say we have a $\rho(n)$-approximation alg.

An approximation scheme takes as input $\varepsilon > 0$ and provides a $(1+\varepsilon)$-approximation algorithm.

- A polynomial-time approximation scheme (PTAS) provides algorithm that is polynomial in n but not necessarily in $\frac{1}{\varepsilon}$ (e.g., $O(n^{2/\varepsilon})$).
- A fully polynomial-time approximation scheme (FPTAS) provides algorithm polynomial in n and $\frac{1}{\varepsilon}$ (e.g., $O(\frac{n}{\varepsilon})$)

Vertex Cover - optimization version (we saw decision vers.) | L21.3

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

Output: Set of vertices $S \subseteq V$ s.t.

$$\forall e = (u, v) \in E, \quad S \cap e \neq \emptyset$$

Objective: Minimize $|S|$

2-approximation algorithm for VC

- Pick any edge $(u, v) \in E$
- Add both u & v to S (which was initially empty)
- Remove all edges from E that are incident on u or v
- Repeat until $E = \emptyset$

Note:

① Running time $O(V+E)$, using adjacency list to represent graph

② Non-deterministic — the output obtained depends upon the order in which edges are selected.

③ The output S is always a valid vertex cover. Edges are only removed once they are covered by a vertex in S.

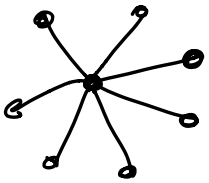
Claim: $|S_{APX}| \leq 2|S_{OPT}|$ (Alg is a 2-approximation)

Proof:

- Let A be the set of edges picked by alg.
- No edges in A share an endpoint (because removed other edges incident on u or v)
- $|S_{APX}| = 2|A|$
- Optimal vertex cover must include at least one endpoint for each edge in A.
- $|A| \leq |S_{OPT}| \Rightarrow$ $\boxed{|S_{APX}| \leq 2|S_{OPT}|}$

Further notes:

① Picking only one endpoint is not an improvement. Consider graph:



(could have cover of size $n-1$ when size 1 would do)

② Being even greedier in edge selection doesn't help (always pick edge whose endpoints have highest degree)
- does not improve worst-case bound
- complicates analysis

③ Showed 2-approximation without determining $|S_{OPT}|$

④ Post-processing can help
- remove extra vertices at end
- can't weaken approximation bound
- can be useful in practical situations.

* We revisit Vertex Cover later in this lecture.

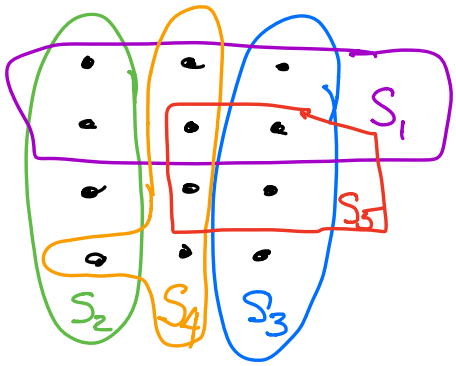
Set Cover

NP-hard problem

Given: A set X of n points and m subsets S_i of X such that $\bigcup_{i=1}^{m} S_i = S_1 \cup S_2 \cup \dots \cup S_m = X$

Find: Cover $C \subseteq \{1, \dots, m\}$ such that $\bigcup_{i \in C} S_i = X$ while $|C|$ is minimized.

Example:



$C_{OPT} = \{2, 3, 4\}$ and the optimal set cover has size 3.

Greedy Algorithm:

REPEAT until all elements covered
- Choose a new set S_i containing maximum # uncovered elements
- Add i to C (C is initially empty)
- Mark all elements from S_i as covered

RETURN C

returns a valid Set Cover that may not be optimal

Running Time: Polynomial
- # of iterations of loop is $O(\min(n, m))$ — at least one element and one set's elements removed per iteration
- each iteration is $O(m \cdot n)$
- Product is $O(m \cdot n \cdot \min(m, n))$

Greedy Set Cover is a $(\lceil \ln n \rceil + 1)$-approximation

The idea of this analysis is that at each iteration, the algorithm covers a "large" fraction $\left(\frac{1}{|C_{OPT}|}\right)$ of the remaining elements.

Let $t = |C_{OPT}|$

At each iteration i, let X_i be the set of remaining elements

- X_i can be covered by t sets (or fewer) because $X_i \subseteq X$, and X can be covered by t sets
- $\exists$ a set that covers $\geq \frac{|X_i|}{t}$ elements (Otherwise, could not cover $|X_i|$ elements with t sets)
- by picking set with max cover of X_i, our algorithm picks set that covers $\geq \frac{|X_i|}{t}$ elements

note: $\left(1 - \frac{1}{k}\right)^k \leq \frac{1}{e}$

$\Rightarrow \forall i: \; |X_{i+1}| \leq \left(1 - \frac{1}{t}\right)|X_i|$

$$|X_i| \leq \left(1 - \frac{1}{t}\right)^i |X| = \left(1 - \frac{1}{t}\right)^{t \ln n} \cdot n \leq e^{-\ln n} \cdot n = 1$$

(At step $i = t \ln n$; $|X| = n$)

So at step $t \ln n + 1$, alg has terminated

Thus, $|C_{OPT}| = t$ and $|C_{APX}| \leq t \ln n + 1$

So alg is $(\ln n) + \frac{1}{t} \Rightarrow (\ln n) + 1$-approx

$\Rightarrow |C_{APX}| \leq (\lceil \ln n \rceil + 1)|C_{OPT}|$

(depends on n.)

Partition

NP-hard

Given: A sorted list of n positive numbers $S_1 \geq S_2 \geq \ldots \geq S_n$

Find: A partition of the indices $\{1,\ldots,n\}$ into two sets A and B such that

① $\{1,\ldots,n\} = A \cup B$

and ② $\max\left\{\sum_{i\in A} S_i, \sum_{i\in B} S_i\right\}$ is minimized.

Note: Optimization problem seeking most balanced partition.

Example: 12 10 9 7 4 3 2

A: 12 10 — B: 9 7 4 3 — A: 2

$w(A) = \sum_{i\in A} S_i = 24$

$w(B) = \sum_{i\in B} S_i = 23$

Note that a 2-approximation would be trivial

Consider the following approximation algorithm

Define: $m = \lceil \frac{1}{\varepsilon} \rceil - 1 \longrightarrow \varepsilon \geq \frac{1}{m+1}$

First Phase: Find optimal partition A', B' for the first m numbers $S_1, S_2, \ldots, S_m$.

Such as by brute force, which would take $O(2^m)$ time, which can be large but is constant with respect to n.

Second Phase: Set $A \leftarrow A'$ and $B \leftarrow B'$

```
for i = m+1 to n
    if w(A) ≤ w(B): A ← A ∪ {i}
    else:           B ← B ∪ {i}
return (A, B)
```

$O(n)$

This returns a valid partition. How close to optimal?

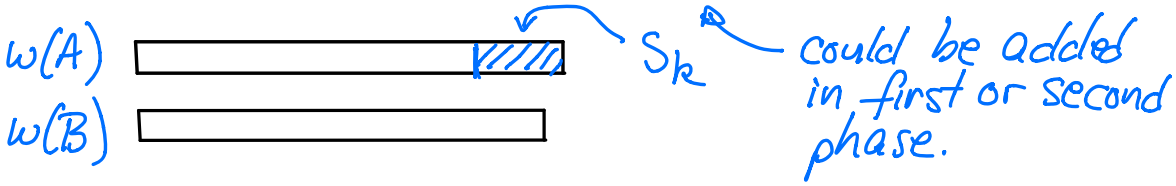
This is a $(1+\varepsilon)$-approximation algorithm

WLOG, let $w(A) \geq w(B)$

Let $2L = \sum_{i=1}^{n} S_i$, so $w(A) \geq L$ and $w_{OPT}(A) \geq L$

Define approximation ratio as $\frac{w(A)}{L}$ (upper bound)

Let k be the last index added to set A.



Case 1: k added to A in first phase. Implies $A = A'$. Implies further that we have the optimal partition for all n numbers, because A' was optimal for first m numbers and the remaining $n-m$ numbers maximally reduced the remaining imbalance.

Case 2: k added to A in second phase.

Algorithm tells us $w(A) - S_k \leq w(B)$ when S_k was added. Also true at termination because additional numbers only added to B.

- $w(A) - S_k \leq w(B) = 2L - w(A)$ Note: $w(A)+w(B)=2L$
- $w(A) \leq L + \frac{S_k}{2}$
- $S_1, S_2, \ldots, S_m$ all $\geq S_k$ due to sorted ordering
- $2L \geq (m+1) S_k$ because $k > m$
- Approx. ratio $= \frac{w(A)}{L} \leq 1 + \frac{S_k}{2L} \leq 1 + \frac{1}{m+1} \leq 1+\varepsilon$ ∎

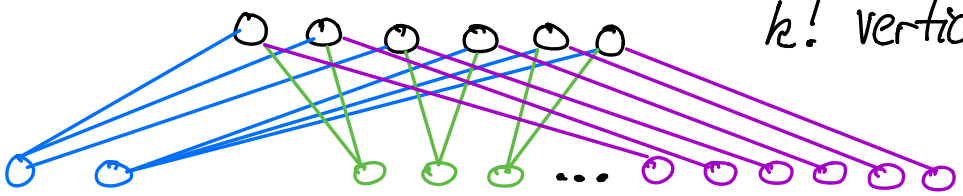
Revisit Vertex Cover

What happens if we consider a greedy algorithm that selects vertices of maximum degree.

REPEAT UNTIL $E = \emptyset$ (initialized empty)
  pick $v \in V$ with max degree, $S \leftarrow S \cup \{v\}$
  remove v and edges incident to v from G
RETURN S

Polynomial time and returns a valid VC

But, consider the following example input:



$\frac{k!}{k}$ vertices of degree k

$\frac{k!}{k-1}$ vertices of degree $k-1$

$k!$ vertices of degree 1

Greedy algorithm may pick all of the bottom vertices.

$|S| = k!\left(\frac{1}{k} + \frac{1}{k-1} + \ldots + 1\right) \approx k! \log k$ ← $\log k$ worse than edge picking

Linear Programming Relaxation for Vertex Cover

Assign an LP variable x_i to each vertex $v_i \in V$

$x_i = 1$: select v_i and add to S

$x_i = 0$: do not select v_i

ILP = integer linear program allows $x_i \in \{0,1\}$

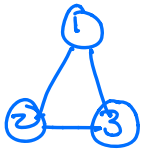
Then minimize $\sum_{i=1}^{n} x_i$ ← size of vertex cover ($n = |V|$)

subject to $0 \le x_i \le 1$ ← temporarily allow x_i to be real, to fit LP formalism

$x_i + x_j \ge 1 \quad \forall e = (v_i, v_j) \in E$ ← Each edge must be covered

Optimal solution x^* may contain non-integer x_i

Example:



optimal VC: $S = \{v_1, v_2\}$ ← $x_1 = x_2 = 1, x_3 = 0$

$\sum_{i=1}^{3} x_i = 2$

LP soln: $x_1^* = x_2^* = x_3^* = \frac{1}{2}$

$\sum_{i=1}^{3} x_i = 1.5$

Need to round x_i^* to obtain integers x_i^{**}

Idea 1: $x_i^{**} = 1$ iff $x_i^* = 1$ → Produces sets S that are not valid covers.

Idea 2: Ensure constraint specifying cover is satisfied.

If $x_i + x_j \ge 1$, then at least one of $x_i, x_j \ge \frac{1}{2}$

Suggests approach: $x_i^{**} = 1$ iff $x_i^* \ge \frac{1}{2}$

$S = \{v_i \mid x_i^* \ge \frac{1}{2}\}$ is a valid vertex cover

Check: $\sum_{i=1}^{n} x_i^{**} \leq 2 \sum_{i=1}^{n} x_i^{*}$

Because $x_i^{**} \leq 2 x_i^{*} \quad \forall x_i^{*}, x_i^{**}$

Thus, we've shown that LP relaxation and rounding gives a cover that is no more that twice the cost at the LP optimum.

Can you complete the argument to show that LP relaxation and rounding is a 2-approximation for Vertex Cover?

$$|S_{APX}| = \sum_{i=1}^{n} x_i^{**} \leq 2 \sum_{i=1}^{n} x_i^{*} \leq 2 |S_{OPT}|$$

✓

Can you show this is true?