Approximation algorithms

An algorithm has approximation ratio r if it outputs solutions with cost such that

$c/c^* \leq r$ and $c^*/c \leq r$

where c^* is the optimal cost.

We focus on ratio (as opposed to difference) because that appears to be more natural for problems of interest

- Problem: Cover edges by vertexes

Input: Graph
Output: A minimal set of nodes that touches every edge

Algorithm:
While there is an edge (u, v)
    Add both u and v to your cover.
    Erase all edges adjacent to either u or v.

- Claim: This is a 2 approximation

- Proof:

Consider the set A of edges picked by the algorithm.
Note any cover must have at least one node for each edge,
and so size at least |A|. ■

- Problem: Cover edges by weighted vertexes

Input: Graph, weights for vertexes
Output: A minimal-cost set of nodes that touches every edge

Formulate problem as integer program:
min $\sum x(v)\, w(v)$:
$x(u) + x(v) \geq 1 \;\forall\, (u,v) \in E$,
$x(u) \in \{0,1\} \;\forall\, u \in V$

Integer programs should not be solvable efficiently

- Problem: Cover edges by weighted vertexes

Input: Graph, weights for vertexes
Output: A minimal-cost set of nodes that touches every edge

Relax to linear programming
min $\sum$ x(v) w(v) :
$x(u) + x(v) \geq 1 \ \forall \ (u,v) \in E$,
$x(u) \in [0,1] \ \forall \ u \in V$

- Algorithm:

Solve relaxation

Round: Take nodes with $x(u) \geq 1/2$.

Claim: This is a cover.
Proof: Because $x(u) + x(v) \geq 1/2$ for every edge (u,v) ■

Claim: This is a 2 approximation
Proof: Let C^* be an optimal solution.
z be cost of relaxed linear program
C be cost of output of algorithm

Obviously, $z \leq C^*$ since solution space is bigger

Now note $z = \sum x(v) w(v) \geq \sum_{v \,:\, x(v) \geq 1/2} w(v) / 2 = C/2$.

So $C/2 \leq z \leq C^*$

■

Paradigm:

Believed infeasible		Feasible
	Relaxation	
Integer program	→	linear program
Quadratic program	→	vector program
	Rounding	
Integral solution	←	

Max Cut: given a graph want cut that separates as many edges as possible.

2-approximation:

How?

Max Cut: given a graph want cut that separates as many edges as possible.

2-approximation:

Pick the cut at random. You expect to cut 1/2 of the edges

Possible to do deterministically

We now improve 2 to $1 / 0.87... < 2$

Max Cut: given a graph want cut that separates as many edges as possible.

Maximize $1/2 \sum_{(i,j) \in E} 1 - y_i y_j \; : \; y_i \in \{-1,1\}$

Relax to vector program:

$y_i \rightarrow$ vector $v_i \in R^d$ (where d = polynomial in $|V|$)

$y_i y_j \rightarrow$ inner product $< v_i , v_j >$

$y_i \in \{-1,1\} \rightarrow |v_i| = 1$

Algorithm:

Solve vector program

Round: Take random vector r of length 1.
One side of the cut is $\{ i : < v_i , r > \geq 0\}$

Max Cut: given a graph want cut that separates as many edges as possible.

Analysis:

Expected size of cut is $\sum_{(i,j)}$ Pr[v_i and v_j are separated]

$= \sum \theta_{i,j} / \pi$ (lemma)

$\geq \alpha \sum (1 - \cos \theta_{i,j}) /2$ ($\exists\, \alpha = 0.87...$: this is true $\forall\, \theta$)

$\geq \alpha \sum (1 - \langle v_i , v_j \rangle) /2$ ($\langle v_i , v_j \rangle = \cos \theta_{i,j}$)

$= \alpha$ cost of vector program

$\geq \alpha$ optimal cost

Problem: Cover points by sets

Input: A family of sets over n points.
Output: A minimal number of sets that covers every point.

Algorithm:
Greedily pick a set that covers as much as possible of what's left.

Claim: This is a log(n) approximation
Proof:
Fix an execution of the algorithm: $(S_1, S_2, \ldots,)$
S_i is the i-th set picked by algorithm.
Given this, for each element x, define cost
c_x := 1/ # of new elements covered by set that covers x first
= (if S_i covers x first) $1/| S_i - U_{j < i} S_j |$

Note cost of algorithm $|C| = \sum_x c_x$

Also, let C^* be optimal.

Have $|C| \leq \sum_{S \in C^*} \sum_{x \in S} c_x$, since every point is covered

We wil show $\forall$ S, $\sum_{x \in S} c_x \leq O(\log n)$,

yielding $|C| \leq O(|C^*| \log n)$.

Claim: $\forall$ S, $\sum_{x \in S} c_x \leq O(\log n)$,

Proof: Fix S. u_i := # elements in S uncovered after i-th iteration of algorithm = $|S - \cup_{j \leq i} S_j|$

$u_0 = |S|$

Let k be the first such that $u_k = 0$.

Note u is decreasing, $u_{i-1} - u_i$ is # elements in S covered first time by S_i .

$$
\begin{aligned}
\sum_{x \in S} c_x &= \sum_{1 \leq i \leq k} (u_{i-1} - u_i)/ |S_i - \cup_{j < i} S_j| \\
&\leq \sum_{1 \leq i \leq k} (u_{i-1} - u_i)/ |S - \cup_{j < i} S_j| \quad \text{(greedy choice)} \\
&= \sum_{1 \leq i \leq k} (u_{i-1} - u_i)/ u_{i-1} \\
&= \sum_{1 \leq i \leq k,\ 1+ui \leq j \leq u(i-1)} 1/ u_j \\
&= \sum_{1+uk \leq i \leq u0} 1/i = O(H(u_0)) = O(H(|S|)) = O(\log |S|) \quad \blacksquare
\end{aligned}
$$

Problem: Given n numbers x_1 , x_2 , …, x_n
integer t, compute maximum size of subset of numbers not exceeding t

This problem has fully polynomial-time approximation algorithm: in time poly(n,1/ ε) finds a sum that does not exceed t and is within 1+ ε of largest not exceeding t.

Naive approach:
$L_0 = \emptyset$
For every i: $L_{i+1} = L_i + x_i$; Remove elements bigger than t
Return Max in L_n

Problem ?

Problem: Given n numbers x_1 , x_2 , …, x_n
integer t, compute maximum size of subset of numbers not exceeding t

This problem has fully polynomial-time approximation algorithm: in time poly(n,1/ ε) finds a sum that does not exceed t and is within 1+ ε of largest not exceeding t.

Naive approach:
$L_0 = Ø$
For every i: $L_{i+1} = L_i + x_i$; Remove elements bigger than t
Return Max in L_n

Problem, list gets too big.
For approximation, don't keep elements close to each other.

Trim(L, δ) : Go through elements in L in sorted order.
Add element y in L ⇔ bigger than 1 + δ of what you have already

Approximation algorithm(x_1 , …, x_n , t, ε)

$L_0 = \emptyset$

For every i: $L_{i+1} = L_i + x_i$

Trim(L_{i+1}, ε /2n)

Remove elements bigger than t

Return Max in L_n

● Correctness:

Claim:
Let P_i be set of possible sums of first i elements
$\forall\, y \in P_i \;\; \exists\, z \in L_i : y/(1+ \varepsilon/2n)^i \leq z \leq y$

i.e., $\forall\, y\; \exists$ a close lower bound z

Proof by induction. Won't see

Given claim, easy to see algorithm gives an ε approximation.

● Running time:
We bound length of lists. Let $\delta = \varepsilon / 2n$
By construction $| L_i | \leq \log_{1 + \delta} t$
$= O(\log t / \delta)$
$= O(n/ \varepsilon) \log t$ ■