CSC 2421 Approximation algorithms - Spring 2003
Lecture #2, Tuesday, Jan 14th, 2003
Lecturer: Allan Borodin
Scribe: Mark Braverman and Pixing Zhang

Lecture #2

2.1 The set cover problem

We first recall the statement of the set cover problem.

Definition 1 *Given a collection of sets $S = \{S_1, S_2, \ldots, S_r\}$ over the universe $U = \{e_1, e_2, \ldots, e_n\}$ (i.e. $S_i \subseteq U$ for $i = 1, 2, \ldots, r$), then the (unweighted)***set cover problem** *is to find the smallest possible k such that there is a subset $\{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ of S which covers the whole universe U, i.e. $U = \bigcup_{j=1}^{k} S_{i_j}$.*

The **weighted** version of the problem is very similar. The only difference is in the objective function: we now attach costs to sets.

Definition 2 *Let the sets U and S be as in the previous definition. And suppose we have the* **cost** *function $c : S \longrightarrow \mathbb{Q}^{\geq 0}$, then the* **weighted set cover problem** *is to find the smallest possible c_0 such that there is a subset $\{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ of S with the total cost $\sum_{j=1}^{k} c(S_{i_j}) = c_0$, which covers the whole universe U, i.e. $U = \bigcup_{j=1}^{k} S_{i_j}$.*

Denote $n = |U|$, $r = |S|$. We define two parameters associated with the set cover problem, which we will use later.

$$\kappa = \max_{1 \leq i \leq r} |S_i| \quad \text{and} \quad f = \text{maximal frequency} = \max_{1 \leq j \leq n} |\{l | e_j \in S_l\}|.$$

The vertex cover problem can be viewed as a particular case of the set cover problem. To view the vertex cover problem in this way, for a graph $G = (V, E)$ with $V = \{v_1, v_2, \ldots, v_r\}$, the corresponding set cover problem is to set $U = E$, and

$$S_i = \{e \in E | e \text{ is one of the edges of } v_i\}, \quad i = 1, 2, \ldots, r.$$

Note that in this case if E is nonempty, then $f = 2$.

The **hitting set problem** is equivalent to the set cover problem. Given a finite ground set E, a cost function $c : E \longrightarrow \mathbb{Q}$ and subsets $T_1, T_2, \ldots, T_q$ of E. We want to find $A \subseteq E$ such that $A \cap T_i = \emptyset$ for $1 \leq i \leq q$ and the total cost $c(A) = \sum_{a \in A} c(a)$ is minimized.

This problem is equivalent the set cover problem by taking $U = \{T_1, T_2, \ldots, T_q\}$, $S = \{S_e : e \in E\}$, where S_e is the subset of U corresponding to e in a natural way $S_e = \{T_i | e \in T_i\}$ Finding the hitting set A is equivalent to finding a covering set $\{S_a | a \in A\}$.

2.2 The "obvious" greedy algorithm for the Vertex Cover

We analyze the following "obvious greedy algorithm":

Take $A = U$ be the set of all uncovered elements.
Take $\mathcal{S} = \emptyset$ be the set of all sets we've chosen so far to be in the cover.
While $A \neq \emptyset$
  Choose S_i not in $\mathcal{S}$ such that the value of $\frac{c(S_i)}{|A \cap S_i|}$ is minimal
   (the "cost per uncovered element" is minimal).
  add S_i to $\mathcal{S}$.
  $A \leftarrow A \backslash S_i$

This algorithm achieves an approximation ratio of $H_\kappa \leq H_n$. Recall that κ is the size of the largest set and H_n is the harmonic number

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}.$$

It is known that for all n, $\ln n \leq H_n \leq 1 + \ln n$.

We will see that this algorithm does not achieve a good approximation ratio in the unweighted vertex cover case, where we've seen a simple 2-approximation algorithm.

For this algorithm the H_n bound is tight (even in the unweighted case):

(a) (Johnson, 1974) Can force $H_\kappa = H_\Delta$ lower bound for the vertex cover on graphs of maximal degree Δ.
(b) (Slavik) There exists a vertex cover instance which forces a lower bound of $\ln n - \ln \ln n - 0.31$ for this algorithm.

Now we give the H_κ lower bound example (for the obvious greedy algorithm above).

Theorem 1 *The following example is a tight exmaple for the obvious greedy algorithm, with a lower bound of H_k.*

Proof: We construct a bipartite graph in the following manner: on the left side we take a set U_0 with $\kappa!$ nodes of degree κ. On the right side we take κ groups of nodes $V_\kappa, V_{\kappa-1}, \ldots, V_1$. In the group V_i there are $\frac{\kappa!}{i}$ nodes of degree i. There are exactly $\kappa!$ edges going out of each group on the right, and we connect them in a manner such that for each i, each node in U_0 is connected to exactly one node in each V_i, $1 \leq i \leq \kappa$.

Now we run the algorithm on this graph, assuming that we are "unlucky" in breaking ties; that is we assume an adversary is breaking the ties. During the first $\frac{\kappa!}{\kappa}$ steps we might have picked the vertices from V_κ which have the maximal degree κ in the graph. Then the maximal degree among the uncovered edges becomes $\kappa - 1$, and on the next $\frac{\kappa!}{\kappa-1}$ steps we might have choosen the nodes from $V_{\kappa-1}$. We proceed in the same manner, and end up choosing all the nodes from the sets V_i, $1 \leq i \leq \kappa$.

So we obtain a vertex cover with

$$\frac{\kappa!}{\kappa} + \frac{\kappa!}{\kappa - 1} + \ldots + \frac{\kappa!}{1} = H_\kappa \kappa!$$

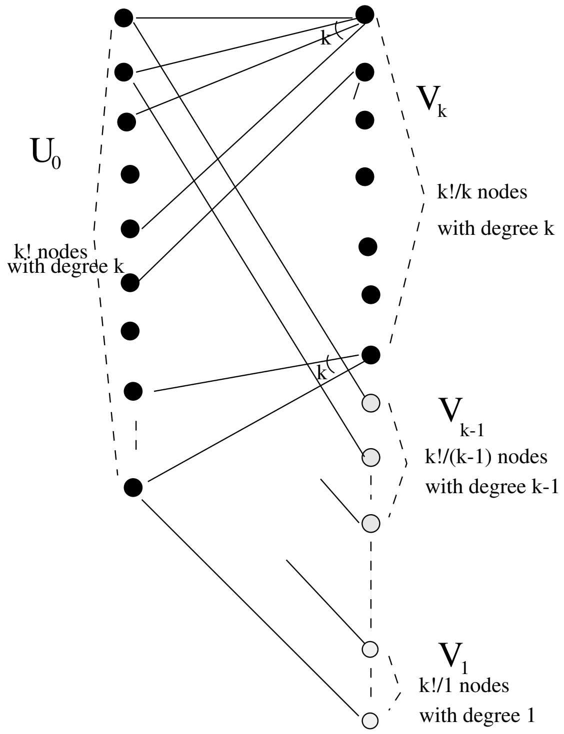

Figure 1: the H_k lower bound example for the greedy algorithm of Set Cover

vertices. However it is obvious that U_0 is a vertex cover for the graph with $\kappa!$ vertices. So we obtain a lower bound of

$$\frac{H_\kappa \kappa!}{\kappa!} = H_\kappa,$$

which shows the H_κ lower bound for the obvious greedy algorithm. ■

Our next goal is to prove that the algorithm gives the desired approximation ratio. Instead of proving the H_κ upper bound on the ratio of the approximation, we prove a weaker H_n upper bound, and leave the H_k lower bound to a later lecture.

Theorem 2 *The obvious greedy algorithm is an H_n factor approximation algorithm for the Set Cover problem.*

Proof: Suppose that the elements of U were covered by the chosen set in the order $\{e_1, e_2, \ldots, e_n\}$, where we do not care about the order between elements chosen in the same iteration of the while loop. We define the **price** function p on the elements of U as follows. If e_k was first covered by the set S_{i_k}, then

$$p(e_k) = \frac{c(S_{i_k})}{|\{e \in S_{i_k} \mid S_{i_k} \text{ was the first set to cover } e_k\}|}.$$

So on each step the sum of the prices of all the elements added is $c(S_{i_k})$ – the cost of the new set added. So if $\mathcal{S}_g$ is the output of the algorithm, we have

$$\sum_{S_i \in \mathcal{S}_g} c(S_i) = \sum_{k=1}^{n} p(e_k).$$

We also observe that from the way the algorithm works, on each step we choose the set S_i so that the price of the newly added elements is the smallest possible, and the sooner we add a set S_j, the smaller will be the price of each element in it. This implies that $p(e_1), p(e_2), \dots, p(e_n)$ is non-decreasing.

Denote the optimal choice of sets by $\mathcal{S}_{opt}$, and

$$c_{opt} = \sum_{S_i \in \mathcal{S}_{opt}} c(S_i)$$

Now suppose that we have covered $\{e_1, \dots, e_{k-1}\}$, we want to estimate $p(e_k)$. Denote $A = \{e_k, e_{k+1}, \dots, e_n\}$. For each $S_i \in S_{opt}$ we have, by the definition of the price

$$p(e_k) \leq \frac{c(S_i)}{|A \cap S_i|}, \quad \text{so} \quad c(S_i) \geq p(e_k) \cdot |A \cap S_i|.$$

Summing over $s_i \in S_{opt}$ we obtain

$$c_{opt} = \sum_{S_i \in S_{opt}} c(S_i) \geq p(e_k) \cdot \sum_{S_i \in S_{opt}} |A \cap S_i| \geq p(e_k)(n-k+1).$$

So

$$p(e_k) \leq \frac{c_{opt}}{n-k+1}.$$

It is trivial to see that this inequality also holds for k's where e_k is not necessarily the beginning of a new chosen set.

So we obtain

$$\sum_{S_i \in S_g} c(S_i) = \sum_{k=1}^{n} p(e_k) \leq \sum_{k=1}^{n} \frac{c_{opt}}{n-k+1} = H_n c_{opt}.$$

So the multiplicative factor is at most H_n, which completes the proof. ■

2.3 The definition for greedy algorithms

Allan Borodin, Morten N. Nielsen and Charles Rackoff [1] formalize the idea of greedy algorithms and define fixed priority algorithms (such as Kruskal's Min Spanning Tree algorithm) and adaptive priority algorithms. The definitions are given below.

Fixed Priority Algorithm

1. Sort the input items in a fair way:$J_1, J_2, ..., J_n$
2. For i from 1 to n do

        consider J_i and make irrevocable decision about it

    Endfor

Adaptive Priority Algorithm

For i from 1 to n do

    Sort the unprocessed inputs in a fair way:$J_i, J_2, ..., J_n$

    consider J_i and make irrevocable decision about it

End for

For the fixed priority algorithm, the sorting function should behave by the following way:

$$\Pi : \mathcal{S} \longrightarrow \mathbb{R}^{\geq 0},$$

where $\mathcal{S}$ is the set of all possible inputs(the fixed priority algorithm always takes the input itme with the largest(or smallest) value of all potential inputs). The ties can be broken arbitrarily, for example an index is given to the input.

For the adaptive priority algorithm, the sorting function can use the whole information from the history, but not from the future;

$$\Pi_{history\ so\ far} : \mathcal{S} \longrightarrow \mathbb{R}^{\geq 0}.$$

For example, Longest Processing Time Job First algorithm is a typical fixed priority algorithm for the MakeSpan problem, in which a set of jobs with different processing times and several identical machines are given, and the object is to minimize the maximaum load of any machine.

The maximal matching algorithm for the Vertex Cover problem and the "obvious greedy algorithm" for set cover can be viewed as adaptive priority algorithms.

In the following, we can again use the above adaptive priority framework to give what Vazirani calls a **layer** algorithm for the weighted VC problem, whose approximation ratio is 2 (see [2]).

A priority function need to be designed first to guide the greedy decision.

Since the layer algorithm is designed for the weighted vertex cover, the degrees have to be related to the weights by some way. In the layer algorithm, the "degree weighted function" manages to do that.

Let $w : V \longrightarrow \mathbb{Q}^{\geq 0}$ be the function assigning weights to the vertices of the given graph $G = (V, E)$.

Definition 3 Degree-Weighted Function. *A function assigning vertex weights is degree-weighted if there is a constant $c > 0$ such that the weight of each vertex $v \in V$ is $c \cdot deg(v)$.*

The significance of such a weight function is captured by the following lemma:

Lemma 3 *[2] Let $w : V \longrightarrow \mathbb{Q}^{\geq 0}$ be a degree-weighted function. Then $w(V) \leq 2 \cdot OPT$.*

Proof: Let c be the constant such that $w(v) = c \cdot deg(v)$, and set U be an optimal vertex cover in G. Since U covers all the edges,

$$\sum_{u \in U} deg(v) \geq |E|$$

Therefore, $w(U) \geq c|E|$. Now since $\sum_{v \in V} deg(v) = 2|E|$, $w(V) = 2c|E|$. The lemma follows. ■

Now we give the priority function for the layer algorithm:

Definition 4 Largest degree-weighted function *in w is obtained as follows: remove all degree zero vertices from the graph, and over the remaining vertices, compute $c = min\{w(v)/deg(v)\}$, then $t(v) = c \cdot deg(v)$ is the desired function.*

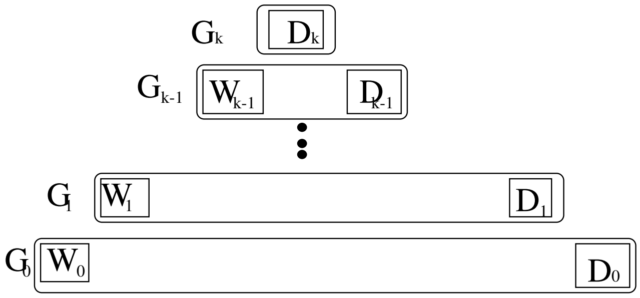

Figure 2: the illustration of the layer algorithm

Furthermore, we call $w'(v) = w(v) - t(v)$ the residual weight function.

Having defined the priority function, it is ready to gave the layer(priority) algorithm.The algorithm is given as follows.

Let $G_0 = G$. Remove all degree zero vertices from G_0, say this set is D_0, and compute the largest degree-weighted function in w. Let W_0 be vertices of zero residual weight; these vertices are included in the vertex cover. Let G_1 be the graph induced on $V - (D_0 \cup W_0)$. Now, the entire process is repeated on G_1 w.r.t(with regard to) the residual weight function. The process terminates when all vertices are of degree zero; let G_k denote this graph. Note that in each iteration, all vertices of maximum (current) degree are chosen for the cover. The process is schematically shown in the figure.

Let $t_0, ..., t_{k-1}$ be the degree-weighted functions defined on graphs $G_0, ..., G_{k-1}$. The vertex cover chosen is $C = W_0 \cup ... \cup W_{k-1}$. Clearly, $V - C = D_0 \cup ... \cup D_k$.

Theorem 4 *[2] The layer algorithm achieves an approximation guarantee of factor 2 for the vertex cover problem, assuming arbitrary vertex weights.*

Proof: We need to show that set C is a vertex cover for G and $w(C) \leq 2 \cdot OPT$. Assuming, for contradiction, that C is not a vertex cover for G. Then there must be an edge (u, v) with $u \in D_i$ and $v \in D_j$, for some i, j. Assume $i \leq j$. Therefore, (u, v) is present in G_i, contradicting the fact that u is a degree zero vertex.

Let C^* be an optimal vertex cover. For proving the second part, consider a vertex $v \in C$. If $v \in W_j$, its weight can be decomposed as

$$w(v) = \sum_{i \leq j} t_i(v)$$

next, consider a vertex $v \in V - C$. If $v \in D_j$, a lower bound on its weight is given by

$$w(v) \geq \sum_{i < j} t_i(v)$$

.

The important observation is that in each layer i, $C^* \cap G_i$ is a vertex cover for G_i, since G_i is a vertex-induced graph. Therefore, by the above lemma, $t_i(C \cap G_i) \leq 2 \cdot t_i(C^* \cap G_i)$.

By the decomposition of weights given above, we get

$$w(C) = \sum_{i=0}^{k-1} t_i(C \cap G_i) \leq 2 = \sum_{i=0}^{k-1} t_i(C^* \cap G_i) \leq 2 \cdot w(C^*)$$

.

This finishes the proof of the theorem. ∎

2.4 An approximation algorithm designed by an IP/LP formualation and rounding

Another polynomial time f-approximation algorithm (in some sense, a more obvious algorithm) for weighted Set Covering problem when f is the max frequency, is designed by using an Integer Programming/Linear Programming relaxation approach.

The standard form of Linear Programming is as follows.

For a minimization problem.

$$\begin{aligned} \text{Min } & c_1x_1 + \dots + c_nx_n \\ \text{s.t. } & \sum_{j=1}^{n} a_{ij}x_j \geq b_i, \text{ for } 1 \leq i \leq m \\ & x_j \geq 0 \end{aligned}$$

For a maximization problem.

$$\begin{aligned} \text{Max } & c_1x_1 + \dots + c_nx_n \\ \text{s.t. } & \sum_{j=1}^{n} a_{ij}x_j \leq b_i, \text{ for } 1 \leq i \leq m \\ & x_j \geq 0 \end{aligned}$$

For simplicity, we also use the following vector form:

$$\begin{aligned} \text{Min } & \vec{c} \cdot \vec{x} \\ \text{s.t. } & A\vec{x} \geq \vec{b} \\ & \vec{x} \geq 0 \end{aligned}$$

An LP problem can be solved in polynomial time, although usually not as efficiently as a "combinatorial algorithm".

For the IP problem, its only difference from LP is that $x_j \in \mathbb{Z}$, for all j. And for a $\{0,1\}$ IP problem, $x_j \in \{0,1\}$ should hold for all j.

The basic idea of designing an approximation algorithm by linear programming is as follows: first formulate the problem as an IP problem, relax it to an LP problem, and finally

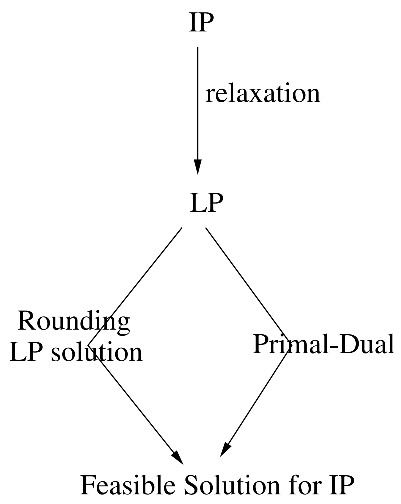

Figure 3: the key idea of approximation algorithms by LP method

either solve it by "rounding" the fractional solution or use a primal dual method to guide the construction of a feasible integral solution. This is illustrated in the following figure.

For the Set Cover problem, an IP can be formulated as follows:

$$\begin{aligned} \text{Min } & \sum_{s \in S} c_s x_s \\ \text{s.t. } & \sum_{s: e \in s} x_s \geq 1, \text{ for all } e \\ & x_s \in \{0, 1\} \end{aligned}$$

An IP formulation for the VC problem is

$$\begin{aligned} \text{Min } & \sum_{i=1}^{n} c_i x_i \\ \text{s.t. } & x_u + x_v \geq 1, \text{ for all } (u, v) \in E \\ & x_i \in \{0, 1\}, \text{ for } i\{1, ..., n\} \end{aligned}$$

The IP formulation will not be unique. For example, for the VC problem, we can add the following constraints:

$$x_u + x_v + x_z \geq 2, \text{ for all triangles } (u, v, z) \text{ in the graph}$$

Then a linear programming problem is obtained by relaxing the integral constraints. For the Set Cover Problem, $x_S \in \{0, 1\}$ is relaxed to to $x_S \geq 0$, where we note that $x_S \leq 1$ is redundant.

By using some polynomial LP algorithm, a fractional solution for the linear programming problem can be obtained. Then a **rounding** strategy is applied: For the set covering LP above, if in the fractional solution, $x_S \geq \frac{1}{f}$, where $f = max\ \ frequency$, then x_S is assigned 1, which means the corresponding set is selected; otherwise, x_S is assigned to 0, which means the set s is not selected.

For the above algorithm, the first thing we need to prove is that it is a feasible solution for the set cover problem. Then we will prove that the above approximation algorithm has an f-approximation ratio.

Different rounding methods might result in different approximation ratios. But for this problem, this is the best ratio that can be obtained by this method, because there is a inherent integrality gap between the corresponding IP and LP formulation.

References

[1] Allan Borodin, Morten N. Nielsen, Charles Rackoff: (Incremental) priority algorithms. SODA 2002: 752-761

[2] Vijar V.Vazirani, Approximation Algorithm, pp17-19, Springer,2001