COM S 783 / OR&IE 634 Approximation Algorithms **Fall Semester, 2003**

Lecture 3: Sept 8, 2003

Lecturer: Yuval Rabani *Scribe: Aleksandrs Slivkins*

1 Vertex Cover

Let G be an undirected graph. Say a set S of nodes *covers* an edge if at least one of its endpoints lies in S. Say S is a *vertex cover* if it covers every edge. The Vertex Cover problem is, given G and weights $w_i \geq 0$ on vertices, to find a vertex cover of minimum weight. Note that it is a special case of Set Cover, where each node covers the edges adjacent to it. The difference is that in Vertex Cover every element (edge) appears in exactly two sets (its endpoints). This allows for better approximation guarantees. We start with a simple 2-approximation algorithm for the unweighted case.

Algorithm 1.1 *Find a maximal matching (not necessarily maximum!), take both endpoints of each edge in the matching.*

The resulting solution S is a vertex cover by the maximality of matching: if an edge uv is not covered by S then the matching does not contain any edges incident to u or v, so uv could have been added to the matching. Clearly, every edge in the matching has to be covered by a separate vertex, which requires at least $|S|/2$ vertices. Therefore the algorithm is a 2-approximation.

In the rest of this section we'll show two 2-approximation algorithms for the weighted Vertex Cover, illustrating respectively *LP-rounding* and *primal-dual* approaches. We'll start with the former.

LP-relaxation. In the IP formulation of Vertex Cover, for each vertex i we have a variable x_i equal to 1 if i is included in the vertex cover, and 0 otherwise. As usual, in the LP-relaxation we set $x_i \in [0,1]$ which (for this LP) is equivalent to $x_i \geq 0$.

$$\begin{array}{lll} \text{VC-LP: minimize} & \sum_{i \in V} w_i x_i, & \\ \text{subject to} & x_i + x_j \geq 1 & \text{for each } ij \in E, \\ & x_i \geq 0 & \text{for each } i \in V. \end{array}$$

Call a feasible solution to LP *basic* if it cannot be expressed as a convex combination of two other feasible solutions. It is a well-known fact that there exists an optimal solution to LP which is basic. Moreover, such solution can be found in polynomial time. Call a solution *half-integral* if each variable is an integer or a half-integer.

Theorem 1.2 *Any basic feasible solution to VC-LP is half-integral.*
Proof: Consider a feasible solution x. Let

$$\begin{aligned} V_+ &= \{i \in V : \frac{1}{2} < x_i < 1\} \\ V_- &= \{i \in V : 0 < x_i < \frac{1}{2}\} \end{aligned}$$

Let x^+ be the solution where we add ε to each entry $i \in V_+$ and subtract ε from each entry $i \in V_i$. Similarly, for x^- we subtract ε from each entry of V_+ and add ε to each entry of V_i. We can choose ε so that both x^+ and x^- are feasible, namely take ε smaller than the smallest positive x_i. If x is not half-integral then x^+ and x^- are different from x, so x is not basic since $x = \frac{1}{2}(x^+ + x^-)$. □

Algorithm 1.3 [2] *Take an optimal solution to VC-LP. By the above theorem it is half-integral; pick all variables that are equal to $\frac{1}{2}$ or 1.*

This algorithm is obviously a 2-approximation. Its drawback is that it involves solving an LP, which is time-consuming (even though VC-LP is comparatively small). In the rest of this section we show a faster and simpler 2-approximation algorithm.

Algorithm 1.4 [1] *Pick an uncovered edge ij (w.l.o.g. $w_i \leq w_j$) and subtract w_i from the weights of both endpoints. Include node i in the solution. Repeat till all edges are covered.*

Note that if all weights are the same, this algorithm is equivalent to Alg. 1.1. To analyze it we need to consider the dual LP:

$$\begin{array}{lll} \text{VC-D:} & \text{maximize} & \sum_{ij \in E} y_{ij}, & \\ & \text{subject to} & \sum_{j \in N(i)} y_{ij} \leq w_i & \text{for each } i \in V, \\ & & y_{ij} \geq 0 & \text{for each } ij \in E. \end{array}$$

Here $N(i)$ denotes the neighborhood of i.

Given the dual, Alg. 1.4 can be restated as follows.

Algorithm 1.5 *Initialize all dual variables y_{ij} to zero. Pick y_{ij}, raise it till the constraint for one of its endpoints (say i) becomes tight, include i in the solution. Repeat until all edges are covered.*

Note that as a by-product, Alg. 1.5 constructs a feasible dual solution $\{y_{ij}\}$. The idea is to upper-bound the primal solution (i.e. the vertex cover) by [the multiple of] the dual solution, then use the weak LP-duality to get the approximation guarantee. This approach is known as the *primal-dual schema*.

Let U be the primal solution. Since for each $i \in U$ the dual constraint is tight,

$$w(U) = \sum_{i \in U} w_i = \sum_{i \in U} \sum_{j \in N(i)} y_{ij} \leq \sum_{i \in V} \sum_{j \in N(i)} y_{ij} = 2 \sum_{ij \in E} y_{ij} \leq 2(\text{VC-D}) \leq 2(\text{VC-LP}) \leq 2(\text{VC}),$$

so Alg. 1.4 is a 2-approximation.

2 Knapsack

The input to the Knapsack problem is n items with *sizes* $s_i \geq 0$ and *profits* $p_i \geq 0$, and the knapsack size B. The goal is to find a max-profit subset of items given the constraint that the size of the subset is at most B. W.l.o.g. the size of each item is at most B. First we show a simple 2-approximation algorithm, then a somewhat more complicated algorithm with a much better approximation guarantee.

As a first attempt, let's try a greedy algorithm.

Algorithm 2.1 *Sort items in the non-increasing order of p_i/s_i, take as many items as possible in this order.*

Unfortunately, this algorithm can perform arbitrarily bad. Suppose $n = 2$ and $(s_1, p_1, s_2, p_2) = (1, 2, B, B)$. Then the algorithm picks item 1 gaining profit 2 instead of item 2 with profit B. However, there is an easy fix.

Algorithm 2.2 *Pick the solution of Alg. 2.1 or the most profitable item, whichever is better.*

On HW1 you'll have to show that Alg. 2.2 is a 2-approximation. However, it turns out a 2-approximation is by far not the best one can do for Knapsack.

Definition 2.3 *A* polynomial-time approximation scheme *(PTAS) for a minimization problem is a set of algorithms $\{A_\varepsilon : \varepsilon > 0\}$ such that A_ε is a $(1+\varepsilon)$-approximation that runs in time polynomial in the size of the input.*[1] *If all A_ε's can be represented as a single algorithm with a running time polynomial in $\frac{1}{\varepsilon}$, we call this algorithm a* fully polynomial-time approximation scheme *(FPTAS).*

Note that a FPTAS is essentially the best we can hope for an NP-hard problem unless $P = NP$. However, only a few problems have a FPTAS or a PTAS.

Theorem 2.4 [3] *There is a FPTAS for Knapsack.*

In the next lecture we'll describe an algorithm PP (which stands for "pseudo-polynomial") which solves Knapsack exactly in time $O(n^2 P \log(SB))$, where $P = \max p_i$ and $S = \max S_i$. Note that the only non-polynomial part in the running time is P. Given PP as a black box, define the algorithm A_k as follows: given a Knapsack instance I, define the instance I_k by scaling the profits as $p'_i = \lfloor p_i/k \rfloor$. Then run PP on I_k. Note that PP runs in polynomial time if k is big enough. Finally, define the algorithm A_ε as in the definition of FPTAS as follows. Sort items in the non-increasing order by the ratio p_i/s_i, let i be the largest index such that $s_1 + \ldots + s_i \leq B$, then run A_k for

$$k = \frac{p_1 + \ldots + p_{i+1}}{2(1+\frac{1}{\varepsilon})n}.$$

In the next lecture we'll prove the following theorem.

Theorem 2.5 *The algorithm A_ε runs in time $O(\frac{1}{\varepsilon} n^3 \log(SB))$ and is a $(\frac{1}{1+\varepsilon})$-approximation.*

The running time of A_ε can actually be improved to $O(n \log \frac{1}{\varepsilon} + (\frac{1}{\varepsilon})^4)$ [4].

[1] For a maximization problem, A_ε needs to be a $(\frac{1}{1+\varepsilon})$-approximation.

References

[1] R.Bar-Yehuda and S.Even, "A linear-time approximation algorithm for the weighted vertex-cover problem," *Journal of Algorithms,* 2:198-203, 1981.

[2] D.S.Hochbaum, "Approximation algorithms for the set covering and vertex cover problems," *SIAM Journal on Computing*, 11:555-556, 1982.

[3] O.H.Ibarra and C.E.Kim, "Fast approximation algorithms for the knapsack and sum of subset problems," *Journal of the ACM*, 22:463-468, 1975.

[4] E.Lawler 1979, "Fast approximation algorithms for knapsack problems," *Mathematics of Operations Research* 4(4), 339–356, 1979.