COMPSCI 330: Design and Analysis of Algorithms	December 2, 2014
Lecture # 26	
Lecturer: Debmalya Panigrahi	*Scribe: Nat Kell*

1 Overview

In this lecture, we will examine a technique known as *LP rounding*. We will first describe the technique at a high level and then give examples of LP rounding algorithms for vertex cover and set cover.

2 The Technique

Recall that the goal of approximation is (typically) to design an algorithm for an NP-hard optimization problem that runs in polynomial time[1]. It is futile to try to devise an algorithm that outputs the optimal solution for every instance (or else this would show P = NP), but hopefully we can guarantee that the solutions produced by the algorithm are always within some multiplicative factor of the optimal solution (if this factor is α, we say the algorithm is α-approximate). As we have seen, the key to proving the approximation factor of an algorithm is lower bounding the cost of the optimal solution (or upper bounding the cost for maximization problems; to keep things simple, we will assume we are dealing with minimization problems for the rest of the notes).

Linear programming provides us with a means of obtaining both a polynomial time algorithm and a lower bound: There are many algorithms that can solve linear programs in weakly-polynomial time. Furthermore, the optimal solution to the fractional relaxation of an integer program is a lower bound on the optimal integral solution (since an integer solution is also a feasible solution to the factional relaxation, the solution can only improve when we remove the requirement of integrality). Therefore, if we can take an optimal solution to an LP relaxation x^*, specify a method for rounding the fractional values in x^* to integer values, and then show that this rounding procedure can only increase the cost of x^* by a factor of α, then this gives an α-approximate polynomial time algorithm (as long as the rounding procedure runs in polynomial time).

The recipe for designing an LP rounding algorithm is shown in Figure 2. Note that ALGO 1 is the always the same: we use some black-box linear programming solver to obtain a fractional solution. The the intricacies (or magic) of every LP rounding algorithm happens in ALGO 2—designing a procedure that rounds the fractional solution to an integer solution.

3 Vertex cover: Threshold rounding

Recall the vertex cover problem: given a graph $G = (V, E)$, find a subset of vertices $V' \subset V$ of minimum size such that for each $(u, v) \in E$, $u \in$ or $v \in V$. In Lecture 24, we gave a greedy algorithm for vertex cover and showed it is a 2-approximation. We will now see another 2-approximation algorithm that uses LP rounding.

[1]Note that there are approximation algorithms for problems that have polynomial-time exact algorithms. For example, one might try to design a $(1+\varepsilon)$-approximate algorithm that runs in $O(n)$ time for a problem whose best exact solution runs in $O(n^2)$ time

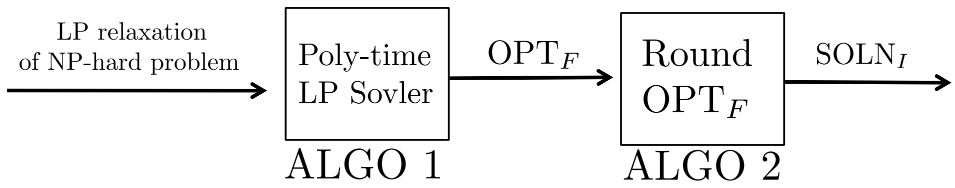

Figure 1: Outline for designing an LP rounding algorithm. We start with an integer programming formulation of an NP-hard problem, which we then relax to a linear programming formulation. We then solve this relaxed LP for optimal fractional solution OPT_F (ALGO 1). Next, we feed OPT_F into ALGO 2, which rounds the fractional values in OPT_F to obtain an integer solution SOLN_I. To do our analysis, we prove that $\text{Cost}(\text{SOLN}_I)) \leq \alpha \cdot \text{Cost}(\text{OPT}_F)$ to show the entire algorithm is an α approximation (since OPT_F is a lower bound on the optimal solution to our NP-hard problem).

Vertex cover can be formulated as the following integer program:

$$\min \sum_{v \in V} x_v$$

$$\text{s.t. } x_u + x_v \geq 1 \; \forall \; (u,v) \in E \tag{1}$$

$$x_v \in \{0,1\} \forall \; v \in V \tag{2}$$

To formulate the LP relaxation, we simply replace our "$x_v \in \{0,1\}$" constraints with $0 \leq x_v \leq 1$ (although note that the LP can gains no advantage by setting a variable to be larger than 1; therefore, really all we need is the constraint $x_v \geq 0$). As the first step in our rounding algorithm, we will solve the LP relaxation to obtain an optimal fractional solution x^* (which is our OPT_F is the Figure 2).

Next, we round x^* in ALGO 2. For our rounding procedure, we will do the simplest thing one can think of: if $x_v^* \geq 1/2$, round x_v^* up to 1; otherwise, round x_v down to 0. Since we are picking a fixed threshold for the rounding boundary that is the same no matter what x^* we are given, this technique is called *threshold rounding*. We then return this rounded solution $\tilde{x}$ as our integer solution (this is SOLN_I in Figure 2).

We first need establish that integer solution is feasible, i.e., every edge is still covered in $\tilde{x}$. This follows directly from that fact that x^* is a feasible solution: If there existed an edge $(u,v) \in E$ such that both $\tilde{x}_v = 0$ and $\tilde{x}_u = 0$, then, based on the rounding algorithm, we would have $x_v^* < 1/2$ and $x_u^* < 1/2$. This implies that $x_v^* + x_u^* < 1$, which is a contradiction since constraint (1) ensures $x*_v + x_u^* \geq 1$.

Finally, we argue that this algorithm is indeed a 2-approximation by showing $\sum_{v \in V} \tilde{x}_v \leq 2 \sum_{v \in V} x_v^*$. Again, this follows the directly from the rounding procedure. In the worst case, every $x_v^* = 1/2$ for all $v \in V$, implying we round all variables up and double their cost. Therefore, the cost of the $\tilde{x}$ can be at most double the cost of x^*.

4 Set cover: Randomized rounding

(Under construction)