CSC 373 - Algorithm Design, Analysis, and Complexity **Summer 2016**

Lalla Mouatadid

Introduction to Approximation Algorithms

The focus of these last two weeks of class will be on how to deal with NP-hard problems. If we don't know how to solve them, what would be the first intuitive way to tackle them?

One well studied method is **approximation algorithms**; algorithms that run in polynomial time that can approximate the solution within a constant factor of the optimal solution. How well can we approximate these solutions? And can we approximate all NP-hard problems?

Before we try to answer any of these questions, let's formally define what we mean by *approximating a solution*. Notice first that if we're considering how "good" a solution is, then we must be dealing with **optimization** problems, instead of decision problems. Recall that when dealing with an optimization problem, the goal is minimize or maximize some objective function. For instance, maximizing the size of an independent set or minimizing the size of a vertex cover. Since these problems are NP-hard, we focus on polynomial time algorithms that give us an approximate solution. More formally:

Let P be an optimization problem.For every input x of P, let $OPT(x)$ denote the optimal value of the solution for x. Let A be an algorithm and c a constant such that $c \geq 1$.

1. Suppose P is a **minimization** problem. We say that A is a c-approximation for P if for every input x of P:

$$OPT(x) \leq A(x) \leq c \cdot OPT(x)$$

2. If P is a **maximization** problem. We say that A is a c-approximation for P if for every input x of P:

$$\frac{OPT(x)}{c} \leq A(x) \leq OPT(x)$$

In both cases, we call c the **approximation ratio** of A.

The goal of this lecture is to learn one way to compute these approximation ratios. A very common approach to do so is obtained by way of **Integer Linear Programming** (ILP)[1]. A integer linear program is exactly what you would think it is: A linear program where some constraints have to be integers. How and why would this matter? Let's look at an example.

We will formulate Minimum Vertex Cover (MVC) as an LP. To do so, we introduce a variable x_i for every vertex i such that $x_i = 1$ iff vertex i is in the vertex cover set, otherwise $x_i = 0$. The LP (or ILP?) looks then like:

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{n} x_i \\ \text{s.t. } & x_i + x_j \geq 1 \quad \forall (i,j) \in E \\ & 0 \leq x_i \leq 1 \quad \forall i \in V \\ & x_i \in \mathbb{Z} \quad \forall i \in V \end{aligned}$$

Notice that the last two constraints can be combined into one integrality constraint $x_i \in \{0, 1\}$. This LP is

[1] Often called Integer Programming, or IP.

indeed an ILP since the x_i's have to be integers. We call this special case of ILP a **0-1 Integer Program**.

How do we construct a solution to the MVC problem from the ILP? It suffices to collect the set of vertices with $x_i = 1$. Notice however that the question of whether an integer program has a feasible solution with objective value at most k (for some k) is actually NP-complete! I say "notice" because it follows exactly from what we just said.

Theorem 1. *The Decision version of Integer Programming is NP-complete.*

First, let's define the problem formally:

The Decision Version of Integer Programming
Input:An integer linear program P with m linear constraints and n variables, with integrality constraints on a subset of the variables, and an integer k.
Problem Return 1 if and only if there is an assignment to the n variables satisfying all the constraints and having objective value at most (or least) k.

Proof of Theorem 1. Clearly the problem is in NP as it suffices to give the verifier an assignment to the n variables. We then can check if all the constraints and the objective value is at most/least[2] k.
The NP-hardness is just as easy to show! In fact it follows immediately from Vertex Cover, formulated as an ILP above. □

We called this the Decision Problem, since the Optimization Problem asks to **minimize** ($\min_k$) the value of the objective function, subject to all the constraints.

Integer Programming is very powerful; there are many problems we can easily express as integer programs, and I would strongly encourage you to look at the some optimization problems of the NP-complete problems we've covered and come up with their IP formulation. As you might notice, there is a natural relationship between integer programs and linear programs. In particular, it is easy to transform an IP to LP. This is very good! Why? Because we know how to solve LPs! And since Interger Programming is NP-hard, this would suggest the ability to convert IPs to LPs, solve the LPs and if we're lucky extract a solution for our original IP.

It turns out this "operation" is one simple way to come up with approximation algorithms. **Linear Programming Relaxation** is the process of turning an IP into an LP; that is, relaxing the integrality constraints. For instance, consider the IP above for the minimum vertex cover problem. To relax it to an LP, it suffices to relax the integrality constraint $x_i \in \{0, 1\}$ to $0 \leq x_i \leq 1$:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} x_i \\ \text{s.t.} \quad & x_i + x_j \geq 1 \qquad \forall (i,j) \in E \\ & 0 \leq x_i \leq 1 \qquad \forall i \in V \end{aligned}$$

Let LP_{OPT} and IP_{OPT} denote the LP and IP optimal solutions respectively. Pick your favorite method to solve this LP in polynomial time, and let $LP_{OPT} = \{x_1^*, x_2^*, ..., x_n^*\}$ be an optimal solution. Notice that the x_i^*'s are not necessarily integers since they lie in the interval $[0, 1]$.

For instance, if $x_i^* = x_j^* = 1/2$ for some edge (i, j), think of this as if two vertices "share" the covering of the edge.

[2] depending on whether we are dealing with a minimization or maximization problem.

This LP solution is not always an IP solution, however the converse holds; so this shows that $LP_{OPT} \leq IP_{OPT}$.

But recall our goal is to compute a solution to the IP. One natural way to construct IP solutions given an LP solution, is by **rounding** the values of the LP solution to integral values. Here's how it works: We define a integer solution set for the IP $y_1, y_2, ..., y_n$ where we match every y_i to an x_i^*. Given the LP_{OPT} solution above, we will set y_i to 1 if $x_i^* \geq 1/2$, otherwise we set y_i to 0. We claim that $y_1, ..., y_n$ is a feasible solution to the MVC IP defined above!

Notice first that $y_i \in \{0, 1\}$, so the integrality constraint is satisfied. Notice next that for any edge $(i, j) \in E$, we have $x_i^* + x_j^* \geq 1$ since the x_i^* are an LP solution and thus satisfy the LP constraints. This implies that at least one of x_i^*, x_j^* is $\geq 1/2$, and so at least one of $y_i, y_j \geq 1$. Therefore for any edge $(i, j) \in E$ in the IP, we have $y_i + y_j \geq 1$, so we satisfy the inequality constraint as well.

For every i, we have $y_i \leq 2x_i^*$ (why?), so if we sum over all the y_i's in the integral solution we have:

$$\begin{aligned} \sum_{i=1}^{n} y_i &\leq 2 \sum_{i=1}^{n} x_i^* \\ &= 2 \cdot LP_{OPT} \\ &\leq 2 \cdot IP_{OPT} \end{aligned}$$

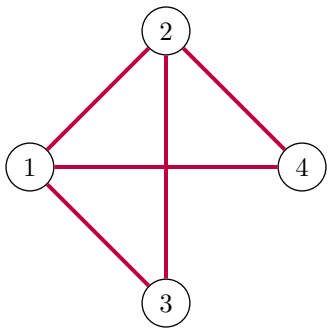
This proves that our algorithm (relaxation + rounding) gives us a 2-approximation for the Minimum Vertex Cover Problem.

Algorithm 1 Min-Vertex Cover LP-Rounding

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

1: Solve the linear programming relaxtion of the IP for MVC.
2: Let $x_1^*, x_2^*, \ldots, x_n^*$ be the values of the variables in the previous solution.
3: **for** $i = 1 \ldots n$ **do**
4: $\quad y_i = 0$;
5: $\quad$ **if** $x_i^* \geq 1/2$ **then**
6: $\quad\quad y_i = 1$;
7: $\quad$ **end if**
8: **end for**
9: **return** $\sum_{i=1}^{n} y_i$

One might ask how tight is this rounding? Consider the graph below:



A minimum vertex cover would be $S = \{1, 2\}$, but an LP could return $S = \{1, 2, 3, 4\}$ each with value

$x_1 = x_2 = x_3 = x_4 = 1/2$, for a total value of $\sum_{i=1}^{4} x_i = 2$. The rounding scheme we gave above would set all the x_i's to 1, resulting an a IP solution of total size 4 (i.e. twice the size of the optimal size). This shows that the analysis of this specific rounding scheme is tight.

Here's an exercise for you. Consider the following variant of MVC: Suppose every vertex now has a non-negative weight, and the goal is to construct a vertex cover that minimizes the sum of the weights of the vertices chosen[3]. This problem is called the *Weighted Minimum Vertex Cover Problem*, formally defined as follows:

Weighted Minimum Vertex Cover
Input:A vertex-weighted graph $G(V, E, w)$ where $w : V \to \mathbb{R}^+$.
Problem Return a subset of vertices $S \subseteq V$ such that S is a vertex cover and

$$\sum_{v \in S} w(v)$$

is minimized.

Can you modify the MVC IP formulation above to take weights into account? Does the analysis of the above rounding algorithm change?[4]

[3]This is the weighted version of MVC, similar to how we saw Interval Scheduling, and Weighted Interval Scheduling.
[4]The answer is no, but convince yourself of this, or see you at office hours!