CSE347 Lecture 24

Approximation Algorithms II: Relaxing and rounding

1. Linear programming
2. Integer linear programming formulations
3. Rounding fractional solutions

A widely used strategy for designing approximation algorithms is to use a certain kind of reduction to rich optimization problems such as "Linear Programming." Algorithms for solving linear programs are feasible to run, but not particularly fast. But, the technique is quite general, and in many cases, gives the best known approximation ratio. Thus, it is useful as either a "first cut" or as a "last resort."

Linear Programming is the following problem. We wish to find a vector $(x_1, x_2, \ldots, x_n)$ of real (rational, actually) numbers that maximize an objective function of the form $\langle \vec{c}, \vec{x} \rangle = \sum_{i=1}^{n} c_i x_i$ where $(c_1, \ldots, c_n)$ is another, given rational vector, subject to a system of linear constraints: $\langle \vec{a}_1, \vec{x} \rangle \geq b_1, \ldots, \langle \vec{a}_m, \vec{x} \rangle \geq b_m$.
We will often use the (standard) notation $A\vec{x} \geq \vec{b}$ where the jth row of the $n \times m$ rational matrix A is $\vec{a}_j$, and $\vec{b}$ is an m-dimensional column vector $(b_1, \ldots, b_m)$.
Note that we can express "$\leq$" constraints, e.g. $\langle \vec{a}_j, \vec{x} \rangle \leq b_j$ by $\langle -\vec{a}_j, \vec{x} \rangle \geq -b_j$, and thus also "$=$" const-

raints by a pair of constraints: $\langle \vec{a}_j, \vec{x} \rangle = b_j$ if and only if $\langle \vec{a}_j, \vec{x} \rangle \geq b_j$ and $\langle -\vec{a}_j, \vec{x} \rangle \geq -b_j$. So, when we write linear programs, we will freely use all three types of constraints.

Here's a silly example that may be close to home for you: you're trying to decide how much time you should spend per week studying (S), on basic life functions like eating, sleeping, and personal hygene (L), and on partying (P). Perhaps you really like partying, prefer to sleep a little more, and don't really enjoy studying, so your objective function might be $2P + 1 \cdot L - 1 \cdot S = \langle (2,1,-1), (P,L,S) \rangle$. But, if you spend < 60 hours per week studying, you won't pass your classes, which is unacceptable, and if you spend any less than 42 hours per week eating & sleeping, you can't function. You can't spend negative time partying (or on anything else, but this is redundant given the previous two constraints). There are only 168 hours in the week to spend on all three.

How can we write this as a linear program?

$S \geq 60$, $L \geq 42$, $P \geq 0$, $P + L + S \leq 168$; maximize $2P + L - S$. Or:

$c = (2,1,-1)$, $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} P \\ L \\ S \end{bmatrix} \geq \begin{bmatrix} 60 \\ 42 \\ 0 \\ -168 \end{bmatrix}$ in our standard form.

These inequalities define a region in three-dimensional space (generally, n-d), called the set of feasible

solutions. You might worry that the number of digits needed to write down an optimal feasible solution is arbitrarily large, but fortunately this doesn't happen. As long as the program has an optimal solution of finite value, it is attained at one of the vertices of the feasible region, where n of the constraints are satisfied with equality, or are said to be "tight." It then follows from our bounds on the sizes of solutions to systems of linear equations.

An algorithm for solving linear programs that is fast in practice, called the "Simplex" algorithm has been known since the 1940's, but this algorithm is known to take exponential time on some carefully constructed examples. A polynomial-time algorithm was discovered in 1979, and by the mid-80's, algorithms that are reasonably fast in practice and have good worst-case complexity were developed. But, we won't be able to discuss the details of these algorithms here.

Integer linear programming formulations

The application of linear programming to design approximation algorithms usually starts with a related problem known as integer linear programming — linear programming, with the additional constraint that the solu-

tion vector $(x_1, \ldots, x_n)$ must be integer-valued. (Recall that it was very useful to us that Ford-Fulkerson produced integer solutions to Max-Flow, analogously.) Now, using integer linear programs, we can formulate many discrete optimization problems. For example, consider weighted Vertex Cover: as before, we want a set of vertices $S \subseteq V$ such that every $e \in E$ has an endpoint in S, but now we are given weights w_v for each $v \in V$ and we wish to minimize $\sum_{v \in S} w_v$ instead of just $|S|$. How can we formulate this problem as an integer linear program? We'll use one variable x_v for each $v \in V$, with constraints $x_v \geq 0$, $x_v \leq 1$, so $x_v \in \{0,1\}$. Now, we'll interpret $x_v = 1$ as meaning "$v \in S$" — so our objective is to minimize $\sum_{v \in V} w_v x_v = \sum_{v \in S} w_v$. For each $(u,v) \in E$, we have a constraint $x_u + x_v \geq 1$, which is satisfied if and only if at least one of u or v is in S (and has x_u or $x_v = 1$). Thus, indeed, $\vec{x}$ is feasible if and only if $\{v : x_v = 1\}$ is a vertex cover.

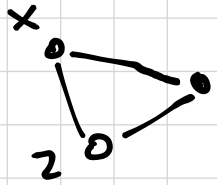
Notice furthermore that given G, it is easy to generate these constraints in $O(|E|)$ time, so we've shown Vertex Cover $\leq_P$ Integer Linear Programming; since Vertex Cover is NP-complete, it follows that there is no polynomial time algorithm for Integer Linear Programming unless P=NP. So we can't hope to solve this formulation directly.

But, what we <u>can</u> do is "relax" (remove) the constraint that the solution vector has integer coordinates. Then, we simply have an ordinary linear program that can be solved

in polynomial time to produce a "fractional solution" $\vec{x}^*$

Lemma If S^* is an optimal weighted vertex cover, $\sum_{v \in V} w_v x_v^* \leq \sum_{v \in S^*} w_v$

Proof: Observe that the integer vector $x'_v = \begin{cases} 1 & \text{if } v \in S^* \\ 0 & \text{otherwise} \end{cases}$ is a feasible solution to the (relaxed) linear program, since it was a solution to the original integer linear program. But, $\vec{x}^*$ is an optimal solution to the linear program, so $\sum_{v \in V} w_v x_v^* \leq \sum_{v \in V} w_v x'_v = \sum_{v \in S^*} w_v$ ▨

Unfortunately $w_{LP} \overset{\text{def.}}{=} \sum_{v \in V} w_v x_v^*$ can be <u>much</u> smaller than $w_{OPT} = \sum_{v \in S^*} w_v$, since the linear program can "cheat"— for example, in the triangle 

the minimum vertex cover has 2 vertices. But, in the linear program, $x = \frac{1}{2}, y = \frac{1}{2}, z = \frac{1}{2}$ satisfies all of the constraints ($x+y \geq 1, y+z \geq 1, x+z \geq 1$) and $x+y+z = \frac{3}{2}$. More generally, the "complete graph" with edges between all pairs (u,v) requires a vertex cover of size $n-1$, but the fractional solution that sets every $x_v = \frac{1}{2}$ again satisfies all of the constraints, while achieving an objective value of $\frac{n}{2}$. The ratio of these objective values—integer and fractional optima—is known as the "integrality gap" of the relaxation. In this case, the integrality gap is at least $\frac{n-1}{2/n} = 2 - \frac{2}{n} \to 2$

Rounding fractional solutions

So, the fractional solution is definitely not what we need: it is not a vertex cover, and may appear to be much better than any possible vertex cover. But, it at least <u>bounds</u> the optimum value, which is a start.

The strategy for converting such fractional to integer solutions is to round the fractional values—here, given $\vec{x}^*$, we can simply round up $x_v^* \geq 1/2$ to 1, and round down $x_v^* < \frac{1}{2}$ to 0. This gives $S = \{v : x_v^* \geq 1/2\}$.

Theorem S is a vertex cover of at most twice the optimal weight.

Proof: First observe that S is a vertex cover. Why?
Because for each edge $(u,v) \in E$, the constraint $x_u^* + x_v^* \geq 1$ was satisfied. So, at least one of x_u^* or x_v^* is $\geq 1/2$, so at least one of u or v is included in S; S is therefore a vertex cover.

Next, we observe S has weight at most $2 \cdot w_{LP}$. Indeed, for each vertex v, the rounded $\vec{x}'$ ($x'_v = 1$ if $v \in S$, $x'_v = 0$ o.w.) satisfies $x'_v \leq 2x_v^*$. Thus, $\sum_{v \in S} w_v = \sum_{v \in V} w_v x'_v \leq \sum_{v \in V} 2 w_v x_v^* = 2 w_{LP}$. But now, by our lemma, $w_{LP} \leq w_{OPT}$. Thus, $\sum_{v \in S} w_v \leq 2 \cdot w_{OPT}$ ▨

Notice, the integrality gap is a lower bound on how much we must increase the objective value by during such a rounding process. Since the integrality gap is 2 here, there is no better way of rounding this formulation. Whether or not there is a better approximation algorithm for vertex cover is not known. Specifically, unlike Set Cover or MAX 3-SAT, we don't have any theorem saying that a better than 2-approximation algorithm for vertex cover would imply faster algorithms for NP.