9.1 Introduction

Suppose we are trying to approximate a minimization problem (everything will hold for maximization problems too, but with the inequalities reversed). A general way to prove α-approximation is as following:

1. Prove OPT $\geq$ LB
2. Prove ALG $\leq \alpha \cdot$ LB

Then we can conclude ALG $\leq \alpha \cdot$ OPT. This is how essentially all of our analyses have worked so far. As we've discussed, a tricky step here is figuring out the lower bound LB on OPT, particularly a lower bound which can then be used to also bound the algorithm. Today we're going to start discussing a technique to approximation algorithms based on linear programming, which will make it far easier for us to find such an LB, and moreover, will give us a few ways of designing algorithms which directly take that lower bound into account.

9.2 Weighted Vertex Cover

Our running example today will be the Weighted Vertex Cover (WVC) problem. Recall that weighted vertex cover is defined as follows:

- **Input**: Graph $G = (V, E)$, and cost function $c : V \to \mathbb{R}^+$
- **Feasible solution:** $S \subseteq V$ such that $\forall \{u, v\} \in E$, either $u \in S$ or $v \in S$
- **Objective:** $\min \sum_{v \in S} c(v)$, i.e. $\min c(S)$

At the very beginning of the semester we gave a 2-approximation for the unweighted version of vertex cover. For that analysis, we used the size of a matching as LB: we proved that any vertex cover has size at least the size of any matching, and our algorithm constructed a vertex cover of size at most twice some matching. But what can we do in the weighted case? Clearly this approach fails, or at least needs to be heavily modified. What else an we use as a lower bound on OPT?

9.2.1 Integer Linear Programming

Before we present our approach to WVC, we first need to make a bunch of useful definitions. Recall the definition of an integer linear program.

- A set $\{x_1, x_2, \cdots, x_n\}$ of variables, each of which must be an integer

- m linear inequalities over variables
- Possibly a linear objective function over variables

A feasible solution is an assignment of values to variables that satisfies all of the m linear inequalities and all of the integrality constraints. In the ILP feasibility problem we are given an ILP (without an objective) and are asked to determine if there is a feasible solution. In the ILP optimization problem we are given an ILP with an objective function and are asked to find an optimal solution or determine that no feasible solution exists (an optimal solution is a feasible solution which minimizes/maximizes the objective function over all feasible solutions).

Clearly by definition, if we can solve the ILP optimization problem then we can solve the ILP feasibility problem. But it's worth noting that the converse is also true: if we can solve the ILP feasibility problem (in polynomial time), then we can solve the ILP optimization problem (in polynomial time). This is because we can turn the objective function into a constraint, and then do a binary search to find the optimal value (this is somewhat similar to the idea of "guessing OPT" from last class).

Why did we introduce all of this? In part because ILPs are a very powerful modeling tool. Let's start by writing an ILP formulation of weighted vertex cover:

$$\begin{aligned} \text{minimize:} \quad & \sum_{v \in V} c(v) \cdot x_v \\ \text{subject to:} \quad & x_u + x_v \geq 1 \qquad \text{for each edge } \{u, v\} \in E \\ & x_v \in \{0, 1\} \quad \text{for each vertex } v \in V \end{aligned}$$

This is clearly an ILP (the $x_v \in \{0, 1\}$ constraints can be replaced by standard integrality constraints $x_v \in \mathbb{Z}$ and $x_v \geq 0$ and $x_v \leq 1$). It is also a formulation of the WVC problem:

1. Every feasible solution x of the ILP corresponds to a feasible solution of the WVC instance (with the same cost), and
2. Every feasible solution S of WVC corresponds to a feasible solution of the ILP (with the same cost).

We can prove this a bit more formally, if we want:

Theorem 9.2.1 *This ILP is a formulation of WVC.*

Proof: Let x be a feasible solution to the ILP. Let $S = \{v \in V : x_v = 1\}$. Then clearly $\sum_{v \in S} c(v) = \sum_{v \in V} c(v) x_v$, so S has the same cost as x. And for all $\{u, v\} \in E$ we know that $x_v + x_u \geq 1$ and thus at least one of u, v is in S, so S is a feasible vertex cover.

Conversely, let S be a feasible vertex cover. Then for all $v \in V$, let $x_v = 1$ if $v \in S$, and let $x_v = 0$ if $v \notin S$. Then $\sum_{v \in V} c(v) x_v = \sum_{v \in S} c(v)$, so S has the same cost as x. And for all $\{u, v\} \in E$ we know that at least one of $u, v \in S$, and thus $x_u + x_v \geq 1$. Thus x is a feasible ILP solution. ■

Note that this implies that OPT(ILP) = OPT(WVC). This already implies that ILP is NP-hard (we just reduced WVC to it).

9.2.2 LP Relaxation

Since ILP is also NP-hard, what was the point of reducing to it, i.e., writing an ILP formulation? We're going to "relax" the ILP to an LP, which we can then solve in polynomial time. We can do this by just removing the integrality constraints, so that variables can take on values in $\mathbb{R}$, giving us a linear program.

$$\begin{aligned} \text{minimize:} \quad & \sum_{v \in V} c(v) \cdot x_v & & \\ \text{subject to:} \quad & x_u + x_v \geq 1 & & \text{for each edge } \{u, v\} \in E \\ & 0 \leq x_v \leq 1 & & \text{for each vertex } v \in V \end{aligned}$$

A seminal result of Khachiyan is that linear programs (ILPs without integrality constraints) can be solved in polynomial time. We'll talk more later about solving LPs efficiently, but mostly we'll use the ability to solve LPs as a black box.

Now we can make some obvious but very important points. Let x^*_{LP} denote the optimal LP solution, and let x^*_{ILP} denote the optimal ILP solution. Then clearly x^*_{ILP} is also feasible for the LP, so we get that

$$OPT(LP) = c(x^*_{LP}) \leq c(x^*_{ILP}) = OPT(ILP) = OPT(WVC).$$

So $OPT(LP)$ is a lower bound on $OPT(WVC)$, so if we can find a vertex cover of cost at most $\alpha \cdot OPT(LP)$ (or equivalently an ILP solution of cost at most $\alpha \cdot OPT(LP)$), we will have an α-approximation!

9.2.3 LP Rounding

This suggests an obvious approach: solving the LP and then "rounding" the real numbers we get back to be integers, giving us a feasible ILP solution. This type of "LP rounding" algorithm works in general as follows:

1. Write ILP
2. Relax to LP (i.e., remove the integrality constraints)
3. Solve LP, getting solution x^*
4. "Round" x^* to integer values to get a solution to ILP

Key ideas of rounding:

1. LP $\leq$ OPT, since it is a relaxation. Slightly more formally, any integral solution is obviously a solution to the LP, and hence the optimal LP value is at most the optimal ILP value, which equals OPT.

2. We design our rounding in order to guarantee that the integral solution we get back is at most $\alpha \cdot$ LP.

Putting these together, we get that ALG $\leq \alpha$ OPT. And instead of having to find some mysterious lower bound LB to compare the algorithm to, we can compare the output of the algorithm to the fractional LP solution, and only analyze how much our rounding increased the cost.

9.2.4 LP Rounding for WVC

As always, we first solve the LP to get an optimal fractional solution x^*. We can do this in polynomial time since the LP has size polynomial in the size of the instance. We then round x^* to an integral solution x' as follows: for each $v \in V$, we set $x'_v = 1$ if $x^*_v \geq \frac{1}{2}$, and set $x'_v = 0$ otherwise.

Theorem 9.2.2 *x' is a feasible solution to ILP.*

Proof: For any $\{u, v\} \in E$, we have $x^*_u + x^*_v \geq 1$, which implies that $\max(x^*_u, x^*_v) \geq \frac{1}{2}$. So either $x'_u = 1$ or $x'_v = 1$. Thus $x'_u + x'_v \geq 1$. ■

Theorem 9.2.3 $c(x') \leq 2c(x^*)$.

Proof: $c(x') = \sum_{v \in V} c(v) \cdot x'_v = \sum_{\{v \in V : x^*_v \geq 1/2\}} c(v) \leq \sum_{v \in V} c(v) \cdot 2x^*_v = 2c(x^*)$. ■

Corollary 9.2.4 *This rounding is a 2-approximation algorithm.*

9.3 Integrality Gaps

In order for LP rounding algorithms to work, we need that $LP \leq OPT$ and that $ALG \leq \alpha \cdot LP$. Since $ALG \geq OPT$, in order for this approach to work we need the LP to be not too much better than OPT. In other words: what's the best approximation ratio we can expect if we use the LP as the lower bound?

Definition 9.3.1 *The integrality gap of an LP for a minimization problem Π is*

$$\sup_{\text{instance } I \text{ of } \Pi} \left(\frac{OPT(I)}{LP(I)} \right).$$

The integrality gap of an LP measures the strength of the relaxation. If we prove that $ALG \leq \alpha \cdot OPT$ by proving that $ALG \leq \alpha \cdot LP$ (as we have been doing with rounding), then we cannot give a ratio α that is better than the integrality gap (or else on the instance I which achieves the integrality gap we would be able to round the LP solution to a value less than OPT, giving a contradiction).

9.3.1 Integrality Gap for Weighted Vertex Cover

Consider the simple instance of $G = K_n$ (the complete graph) and $c(v) = 1$ for all $v \in V$.

Theorem 9.3.2 *The integrality gap is at least $2\left(1 - \frac{1}{n}\right)$.*

Proof: OPT $= n - 1$, since if there are two nodes not in the vertex cover the edge between them will not be covered. But in the LP, if we set set $x_v = \frac{1}{2}$ for every $v \in V$ we get a valid LP solution with cost $\frac{n}{2}$. Hence $IG \geq \frac{n-1}{n/2} = 2\left(1 - \frac{1}{n}\right)$. ■

This means that we cannot really hope for a better approximation algorithm which uses the LP solution as a lower bound on OPT. This is a weaker statement than a true hardness result (since it only implies that no rounding algorithm for this LP can do better, rather than any polynomial time algorithm), but on the other hand it does not require any complexity assumptions like $P \neq NP$.

9.3.2 Integrality Gap for Max Independent Set

Let's see another quick example of an integrality gap, this time for a maximization problem: Max Independent Set. Recall that in max independent set we are given a graph $G = (V, E)$ and are trying to choose an independent set S of maximum size. So we can write the following LP relaxation:

$$\begin{aligned} \text{maximize:} \quad & \sum_{v \in V} x_v \\ \text{subject to:} \quad x_u + x_v &\leq 1 \quad \text{for all } \{u, v\} \in E \\ x_v &\leq 1 \quad \text{for all } v \in V \end{aligned}$$

Note that since this is a maximization problem, we think of the integrality gap as LP / OPT rather than OPT / LP.

Theorem 9.3.3 *The integrality gap is at least* $\frac{n}{2}$.

Proof: Consider the instance to be $G = K_n$, which is a complete graph. So OPT $= 1$ since every pair of vertices are connected. For the LP, set $x_v = \frac{1}{2}$ for every $v \in V$. Then we get the LP cost to be $\frac{n}{2}$. Hence the integrality gap is at least $\frac{n/2}{1} = \frac{n}{2}$. ■

9.4 Solving LPs

This is a bit outside the scope of this class, but we will talk a little bit about algorithms for solving LPs. This section is pretty informal, but everything can be made formal.

9.4.1 Simplex

Note that the feasible region of an LP is a polytope (by definition) and hence is convex (for every two points in the feasible region, their midpoint is also feasible). The oldest heuristic for solving LPs is the simplex algorithm. We won't talk much about this algorithm, but the high-level view is straightforward. Given a polytope, there is a natural graph associated with it where the vertices of the graph are the vertices of the polytope (points which are tight for d linearly independent constraints of the polytope, where d is the dimension) and two vertices are adjacent if in the polytope they are also adjacent (the line segment between them is tight for $d - 1$ linearly independent constraints). This algorithm starts by finding a vertex of the polytope, and then moving to a neighbor with decreased cost as long as this is possible. By linearity and convexity, once it gets stuck it has found the optimal solution.

Unfortunately simplex does not run in polynomial time – even with a polynomially-sized LP, the number of vertices of the polytope can be exponential. Simplex does well in practice, but poorly in theory.

9.4.2 Interior Point Methods

While simplex only moves along the outer faces of the polytope, there are algorithms known as "interior-point"methods which make moves *inside* the polytope. There are now known to be interior-point methods which have polynomial running time, and are also extremely fast in practice. So in both theory and in practice, we can solve LPs efficiently.

9.4.3 Ellipsoid

We will spend a bit more time talking about an algorithm known as *Ellipsoid*, which works well in theory but poorly in practice. Nevertheless, it has some properties which are extremely nice theoretically, so we will feel free to use it when designing algorithms.

As a first step, we will reduce optimization to feasibility. Suppose that we are given an algorithm which can determine whether a given LP is feasible, i.e. it returns YES if the feasible region is non-empty and NO otherwise. Now suppose that we are given an LP which we want to solve. For any value T, we can use our feasibility algorithm to see if there is an LP solution with cost at most T by adding a single new constraint (the objective is at most T). So we can do a binary search over possible values of T to find in polynomial time the smallest T for which the feasibility region is nonempty. Hence if we have such a feasibility algorithm, we can also solve LPs with objective functions.

9.4.3.1 The algorithm

The ellipsoid algorithm is an algorithm for testing feasibility. Informally, it is the following.

Algorithm 1 Using Ellipsoid to Check Feasibility Algorithm

Input: Linear constraints
Output: A feasible solution if one exists.

```
Find an ellipsoid containing the polytope.
Let x⃗ be the the center of the ellipsoid.
k = 0
while x⃗ is not feasible AND k < maxiter do
    k = k + 1
    Find a constraint violated by x⃗
    Draw a parallel hyperplane corresponding to this violated constraint through x⃗. This divides
    the ellipsoid into two parts with equal volume, one of which is on the "wrong" side of the
    hyperplane and so is entirely infeasible.
    Consider the part which is on the correct side of the violated constraint (i.e. the part in which
    the polytope must appear if it is nonempty). Find a new ellipsoid containing this part.
    Let x⃗ be the center of the new ellipsoid.
end while
return x⃗, which is a feasible solution; or no feasible solution exists.
```

It is not hard to see that this algorithm is correct, in that if the polytope is nonempty it will eventually find a feasible point and if the polytope is empty then it will never find a feasible point.

The hard part is proving that it takes only polynomial time in the worst case, i.e. that we can set maxiter to a polynomial value. While it is complicated, the key idea is that due to the geometry of ellipsoids, in each iteration the volume of the ellipsoid that we are considering decreases by a constant factor.

9.4.3.2 Separation

The ellipsoid algorithm has an extremely nice property: in order to make it work, all that we need to be able to do is find a violated constraint given some point $\vec{x}$ (or prove that $\vec{x}$ is feasible). The is is called the *separation* problem. If the LP is small then this is simple – we can just check all of the constraint. But sometimes our LPs are not small, and ellipsoid still lets us solve them! If there are an exponential number of constraints, we cannot even write down all of them in polynomial time so we certainly cannot use simplex or interior-point methods to solve the LP. But if we can separate, then ellipsoid lets us solve in polynomial time despite not even being able to write down the LP. Let's do a somewhat trivial example of this: the spanning tree polytope.

9.4.3.3 Example: Spanning Tree Polytope

Consider the minimum spanning tree problem. Obviously we know multiple simple and fast algorithms. But suppose we want to do this by an LP. While it's not immediately obvious how to write such an LP, it turns out that the following formulation is good:

$$\text{minimize:} \quad \sum_{e \in E} c(e) \cdot x_e \tag{9.4.1}$$

$$\text{subject to:} \quad \sum_{\{e \in E(S,\bar{S})\}} x_e \geq 1 \quad \text{for each cut } S \subseteq V \tag{9.4.2}$$

$$0 \leq x_e \leq 1 \quad \text{for each } e \in E \tag{9.4.3}$$

Note that the tree constraints are hidden in the optimization problem above since we can always make the objective function smaller by throwing away an edge.

Although it has exponentially many constrains, we can separate in polynomial time. Given $\vec{x}$, first run the min-cut algorithm. If the min cut is larger or equal to 1, than $\vec{x}$ is a feasible solution; otherwise the minimum cut S corresponds to a violated constraint. Hence we can separate, so by the ellipsoid algorithm we can optimize.