Lecture 7

In which we show how to use linear programming to approximate the vertex cover problem.

1 Linear Programming Relaxations

An *integer linear program* (abbreviated ILP) is a linear program (abbreviated LP) with the additional constraints that the variables must take integer values. For example, the following is an ILP:

$$\begin{array}{ll} \text{maximize} & x_1 - x_2 + 2x_3 \\ \text{subject to} & \\ & x_1 - x_2 \leq 1 \\ & x_2 + x_3 \leq 2 \\ & x_1 \in \mathbb{N} \\ & x_2 \in \mathbb{N} \\ & x_3 \in \mathbb{N} \end{array} \tag{1}$$

Where $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers.

The advantage of ILPs is that they are a very expressive language to formulate optimization problems, and they can capture in a natural and direct way a large number of combinatorial optimization problems. The disadvantage of ILPs is that they are a very expressive language to formulate combinatorial optimization problems, and finding optimal solutions for ILPs is NP-hard.

If we are interested in designing a polynomial time algorithm (exact or approximate) for a combinatorial optimization problem, formulating the combinatorial optimization problem as an ILP is useful as a first step in the following methodology (the discussion assumes that we are working with a minimization problem):

- Formulate the combinatorial optimization problem as an ILP;

- Derive a LP from the ILP by removing the constraint that the variables have to take integer value. The resulting LP is called a "relaxation" of the original problem. Note that in the LP we are minimizing the same objective function over a larger set of solutions, so $opt(LP) \leq opt(ILP)$;
- Solve the LP optimally using an efficient algorithm for linear programming;
  - If the optimal LP solution has integer values, then it is a solution for the ILP of cost $opt(LP) \leq opt(ILP)$, and so we have found an optimal solution for the ILP and hence an optimal solution for our combinatorial optimization problem;
  - If the optimal LP solution x^* has fractional values, but we have a *rounding* procedure that transforms x^* into an integral solution x' such that $cost(x') \leq c \cdot cost(x^*)$ for some constant c, then we are able to find a solution to the ILP of cost $\leq c \cdot opt(LP) \leq c \cdot opt(ILP)$, and so we have a c-approximate algorithm for our combinatorial optimization problem.

In this lecture and in the next one we will see how to round fractional solutions of relaxations of the Vertex Cover and the Set Cover problem, and so we will be able to derive new approximation algorithms for Vertex Cover and Set Cover based on linear programming.

2 The Weighted Vertex Cover Problem

Recall that in the vertex cover problem we are given an undirected graph $G = (V, E)$ and we want to find a minimum-size set of vertices S that "touches" all the edges of the graph, that is, such that for every $(u, v) \in E$ at least one of u or v belongs to S.

We described the following 2-approximate algorithm:

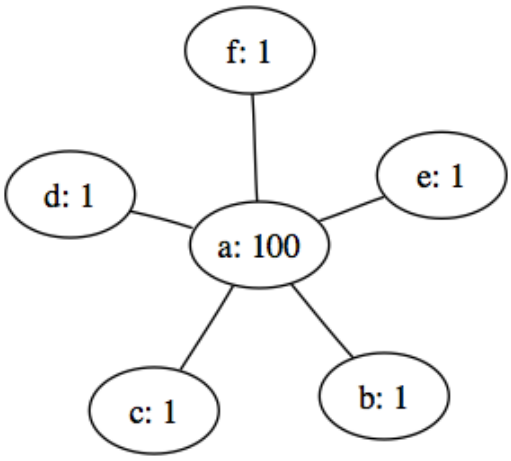
- Input: $G = (V, E)$
- $S := \emptyset$
- For each $(u, v) \in E$
  - if $u \notin S \wedge v \notin S$ then $S := S \cup \{u, v\}$
- return S

The algorithm finds a vertex cover by construction, and if the condition in the *if* step is satisfied k times, then $|S| = 2k$ and the graph contains a matching of size k,

meaning that the vertex cover optimum is at least k and so $|S|$ is at most twice the optimum.

Consider now the *weighted* vertex cover problem. In this variation of the problem, the graph $G = (V, E)$ comes with *costs* on the vertices, that is, for every vertex v we have a non-negative cost $c(v)$, and now we are not looking any more for the vertex cover with the fewest vertices, but for the vertex cover S of minimum total cost $\sum_{v \in S} c(v)$. (The original problem corresponds to the case in which every vertex has cost 1.)

Our simple algorithm can perform very badly on weighted instances. For example consider the following graph:



Then the algorithm would start from the edge (a, b), and cover it by putting a, b into S. This would suffice to cover all edges, but would have cost 101, which is much worse than the optimal solution which consists in picking the vertices $\{b, c, d, e, f\}$, with a cost of 5.

Why does the approximation analysis fail in the weighted case? In the unweighted case, every edge which is considered by the algorithm must cost at least 1 to the optimum solution to cover (because those edges form a matching), and our algorithm invests a cost of 2 to cover that edge, so we get a factor of 2 approximation. In the weighted case, an edge in which one endpoint has cost 1 and one endpoint has cost 100 tells us that the optimum solution must spend at least 1 to cover that edge, but if we want to have both endpoints in the vertex cover we are going to spend 101 and, in general, we cannot hope for any bounded approximation guarantee.

We might think of a heuristic in which we modify our algorithm so that, when it considers an uncovered edge in which one endpoint is much more expensive than the other, we only put the cheaper endpoint in S. This heuristic, unfortunately, also fails completely: imagine a "star" graph like the one above, in which there is a central vertex of cost 100, which is connected to 10,000 other vertices, each of cost 1. Then

the algorithm would consider all the 10,000 edges, and decide to cover each of them using the cheaper endpoint, finding a solution of cost 10,000 instead of the optimal solution of picking the center vertex, which has cost 100.

Indeed, it is rather tricky to approximate the weighted vertex cover problem via a combinatorial algorithm, although we will develop (helped by linear programming intuition) such an approximation algorithm by the end of the lecture.

Developing a 2-approximate algorithm for weighted vertex cover via a linear programming relaxation, however, is amazingly simple.

3 A Linear Programming Relaxation of Vertex Cover

Let us apply the methodology described in the first section. Given a graph $G = (V, E)$ and vertex costs $c(\cdot)$, we can formulate the minimum vertex cover problem for G as an ILP by using a variable x_v for each vertex v, taking the values 0 or 1, with the interpretation that $x_v = 0$ means that $v \notin S$, and $x_v = 1$ means that $v \in S$. The cost of the solution, which we want to minimize, is $\sum_{v \in V} x_v c(v)$, and we want $x_u + x_v \geq 1$ for each edge (u, v). This gives the ILP

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} c(v) x_v & \\ \text{subject to} & & \text{the vertex cover 'touches' all edges.} \\ & x_u + x_v \geq 1 & \forall (u,v) \in E \\ & x_v \leq 1 & \forall v \in V \\ & x_v \in \mathbb{N} & \forall v \in V \end{array} \tag{2}$$

Next, we relax the ILP (2) to a linear program.

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} c(v) x_v & \\ \text{subject to} & & \\ & x_u + x_v \geq 1 & \forall (u,v) \in E \\ & x_v \leq 1 & \forall v \in V \\ & x_v \geq 0 & \forall v \in V \end{array} \tag{3}$$

Let us solve the linear program in polynomial time, and suppose that $\mathbf{x}^*$ is an optimal solution to the LP (3); how do we "round" it to a 0/1 solution, that is, to a vertex cover? Let's do it in the simplest possible way: round each value to the closest integer, that is, define $x'_v = 1$ if $x^*_v \geq \frac{1}{2}$, and $x'_v = 0$ if $x^*_v < \frac{1}{2}$. Now, find the set corresponding to the integral solution $\mathbf{x}'$, that is $S := \{v : x'_v = 1\}$ and output it. We have:

- *The set S is a valid vertex cover*, because for each edge (u, v) it is true that $x^*_u + x^*_v \geq 1$, and so at least one of x^*_u or x^*_v must be at least $1/2$, and so at least one of u or v belongs to S;

- *The cost of S is at most twice the optimum*, because the cost of S is

$$\begin{aligned}
&\sum_{v \in S} c(v) \\
&= \sum_{v \in V} c(v) x'_v \\
&\leq \sum_{v \in V} c(v) \cdot 2 \cdot x^*_v \\
&= 2 \cdot opt(LP) \\
&\leq 2 \cdot opt(VC)
\end{aligned}$$

And that's all there is to it! We now have a polynomial-time 2-approximate algorithm for weighted vertex cover.

4 The Dual of the LP Relaxation

The vertex cover approximation algorithm based on linear programming is very elegant and simple, but it requires the solution of a linear program. Our previous vertex cover approximation algorithm, instead, had a very fast linear-time implementation. Can we get a fast linear-time algorithm that works in the weighted case and achieves a factor of 2 approximation? We will see how to do it, and although the algorithm will be completely combinatorial, its *analysis* will use the LP relaxation of vertex cover.

How should we get started in thinking about a combinatorial approximation algorithm for weighted vertex cover?

We have made the following point a few times already, but it is good to stress it again: in order to have any hope to design a provably good approximation algorithm for a minimization problem, we need to have a good technique to prove lower bounds for the optimum. Otherwise, we will not be able to prove that the optimum is at least a constant fraction of the cost of the solution found by our algorithms.

In the unweighted vertex cover problem, we say that if a graph has a matching of size k, then the optimum vertex cover must contain at least k vertices, and that's our lower bound technique. We have already seen examples in which reasoning about matchings is not effective in proving lower bound to the optimum of weighted instances of vertex cover.

How else can we prove lower bounds? Well, how did we establish a lower bound to the optimum in our LP-based 2-approximate algorithm? We used the fact that the optimum of the linear programming relaxation (3) is a lower bound to the minimum

vertex cover optimum. The next idea is to observe that the cost of any feasible solution to the *dual* of (3) is a lower bound to the optimum of (3), by weak duality, and hence a lower bound to the vertex cover optimum as well.

Let us construct the dual of (3). Before starting, we note that if we remove the $x_v \leq 1$ constraints we are not changing the problem, because any solution in which some variables x_v are larger than 1 can be changed to a solution in which every x_v is at most one while decreasing the objective function, and without contradicting any constraint, so that an optimal solution cannot have any x_v larger than one. Our primal is thus the LP in standard form

$$\begin{array}{lll} \text{minimize} & \sum_{v \in V} c(v) x_v & \\ \text{subject to} & & \\ & x_u + x_v \geq 1 & \forall (u,v) \in E \\ & x_v \geq 0 & \forall v \in V \end{array} \tag{4}$$

Its dual has one variable $y_{(u,v)}$ for every edge (u,v), and it is

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{(u,v) \in E} y_{(u,v)} & \\ \text{subject to} & & \\ & \displaystyle\sum_{u:(u,v) \in E} y_{(u,v)} \leq c(v) & \forall v \in V \\ & y_{(u,v)} \geq 0 & \forall (u,v) \in E \end{array} \tag{5}$$

That is, we want to assign a nonnegative "charge" $y_{(u,v)}$ to each edge, such that the total charge over all edges is as large as possible, but such that, for every vertex, the total charge of the edges incident on the vertex is at most the cost of the vertex. From weak duality and from the fact that (4) is a relaxation of vertex cover, we have that for any such system of charges, the sum of the charges is a lower bound to the cost of the minimum vertex cover in the weighted graph $G = (V, E)$ with weights $c(\cdot)$.

Example 1 (Matchings) *Suppose that we have an unweighted graph $G = (V, E)$, and that a set of edges $M \subseteq E$ is a matching. Then we can define $y_{(u,v)} := 1$ if $(u,v) \in M$ and $y_{(u,v)} := 0$ if $(u,v) \notin M$. This is a feasible solution for (5) of cost $|M|$.*

This means that any lower bound to the optimum in the unweighted case via matchings can also be reformulated as lower bounds via feasible solutions to (5). The latter approach, however, is much more powerful.

Example 2 *Consider the weighted star graph from Section 2. We can define $y_{(a,x)} = 1$ for each vertex $x = b, c, d, e, f$, and this is a feasible solution to (5). This proves that the vertex cover optimum is at least 5.*

5 Linear-Time 2-Approximation of Weighted Vertex Cover

Our algorithm will construct, in parallel, a valid vertex cover S, in the form of a valid integral solution $\mathbf{x}$ to the ILP formulation of vertex cover (2), and a feasible solution $\mathbf{y}$ to the dual (5) of the linear programming relaxation, such that the cost of $\mathbf{y}$ is at least half the cost S. Before starting, it is helpful to reformulate our old algorithms in this language

- Input: undirected, unweighted, graph $G = (V, E)$
- $\mathbf{x} = (0, \cdots, 0)$
- $\mathbf{y} = (0, \cdots, 0)$
- for each edge $(u, v) \in E$
  - if $x_u < 1$ and $x_v < 1$ then
    - $y_{(u,v)} := 1$
    - $x_u := 1$
    - $x_v := 1$
- $S := \{v : x_v = 1\}$
- return $S, \mathbf{y}$

Our goal is to modify the above algorithm so that it can deal with vertex weights, while maintaining the property that it finds an integral feasible $\mathbf{x}$ and a dual feasible $\mathbf{y}$ such that $\sum_{v \in V} c(v)x_v \leq 2 \cdot \sum_{(u,v) \in V} y_{u,v}$. The key property to maintain is that when we look at the edge (u, v), and we find it uncovered, what we are going to "spend" in order to cover it will be at most $2y_{u,v}$, where $y_{u,v}$ will be a charge that we assign to (u, v) without violating the constraints of (5).

We will get simpler formulas if we think in terms of a new set of variables p_v, which represent how much we are willing to "pay" in order to put v in the vertex cover; at the end, if $p_v = c_v$ then the vertex v is selected, and $x_v = 1$, and if $p_v < c_v$ then we are not going to use v in the vertex cover. Thus, in the integral solution, we will have $x_v = \lfloor p_v / c(v) \rfloor$, and so $c(v) \cdot x_v \leq p_v$ and so the total amount we are willing to pay, $\sum_v p_v$ is an upper bound to the cost of the integral solution $\sum_v c(v) \cdot x_v$.

Initially, we start from the all-zero dual solution $\mathbf{y} = \mathbf{0}$ and from no commitment to pay for any vertex, $\mathbf{p} = \mathbf{0}$. When we consider an edge (u, v), if $p_u = c(u)$ or $p_v = c(v)$, we have committed to pay for at least one of the endpoints of (u, v), and so the edge will be covered. If $p_u < c(u)$ and $p_v < c(v)$, we need to commit to pay for at least

one of the endpoints of the edge. We need to pay an extra $c(u) - p_u$ to make sure u is in the vertex cover, or an extra $c(v) - p_v$ to make sure that v is. We will raise, and here is the main idea of the algorithm, *both* the values of p_u and p_v by the smallest of the two values. This will guarantee that we cover (u, v) by "fully funding" one of the endpoints, but it will also put some extra "funding" into the other vertex, which might be helpful later. We also set $y_{(u,v)}$ to $\min\{c(u) - p_u, c(v) - p_v\}$.

Here is the psedocode of the algorithm:

- Input: undirected, unweighted, graph $G = (V, E)$
- $\mathbf{p} = (0, \cdots, 0)$
- $\mathbf{y} = (0, \cdots, 0)$
- for each edge $(u, v) \in E$
  - if $p_u < c(u)$ and $p_v < c(v)$ then
    - $y_{(u,v)} := \min\{c(u) - p_u, c(v) - p_v\}$
    - $p_u := p_u + \min\{c(u) - p_u, c(v) - p_v\}$
    - $p_v := p_v + \min\{c(u) - p_u, c(v) - p_v\}$
- $S := \{v : p_v \geq c(v)\}$
- return $S, \mathbf{y}$

The algorithm outputs a correct vertex cover, because for each edge (u, v), the algorithm makes sure that at least one of $p_u = c(u)$ or $p_v = c(v)$ is true, and so at least one of u or v belongs to S at the end.

Clearly, we have

$$cost(S) = \sum_{v \in S} c(v) \leq \sum_{v \in V} p_v$$

Next, we claim that the vector $\mathbf{y}$ at the end of the algorithm is a feasible solution for the dual (5). To see this, note that, for every vertex v,

$$\sum_{u:(u,v)\in E} y_{(u,v)} = p_v$$

because initially all the $y_{(u,v)}$ and all the p_v are zero, and when we assign a value to a variable $y_{(u,v)}$ we also simultaneously raise p_u and p_v by the same amount. Also, we have that, for every vertex v

$$p_v \leq c(v)$$

by construction, and so the charges $\mathbf{y}$ satisfy all the constraints

$$\sum_{u:(u,v)\in E} y_{(u,v)} = c(v)$$

and define a feasible dual solution. We then have

$$\sum_{(u,v)\in E} y_{(u,v)} \leq opt_{VC}(G)$$

by weak duality. Finally, every time we give a value to a $y_{(u,v)}$ variable, we also raise the values of p_u and p_v by the same amount, and so the sum of our payment commitments is exactly twice the sum of the charges $y_{(u,v)}$

$$\sum_{v\in V} p_v = 2 \sum_{(u,v)\in E} y_{(u,v)}$$

Putting all together we have

$$cost(S) \leq 2 \cdot opt_{VC}(G)$$

and we have another 2-approximate algorithm for weighted vertex cover!

It was much more complicated than the simple rounding scheme applied to the linear programming optimum, but it was worth it because now we have a linear-time algorithm, and we have understood the problem quite a bit better.