18.415/6.854 Advanced Algorithms	November 19, 2001
Lecture 17	
Lecturer: Michel X. Goemans	*Scribe: Steven Richman, Matt Peters*

1 Refresher of Last Lecture

We continue our description of approximation algorithms and design. Last lecture we were looking at the problem of Vertex Cover.

Vertex Cover (VC):
Given undirected graph $G = (V, E)$ and nonnegative weights on each vertex $w(v) \geq 0 \quad \forall v \in V$, find a vertex cover S of minimum total weight $\sum_{v \in S} w(v)$. A vertex cover satisfies $\forall (u, v) \in E \{u, v\} \cap S \neq \emptyset$.

Last time, we introduced the two-step lower bound technique to analyze approximation algorithms. This technique can be generalized to apply to maximization problems, but in this lecture we will restrict our discussion to minimization problems. The first step in the technique is to find a lower bound LB on the optimum solution, and the second step is to show that the analyzed algorithm returns a solution at most $\alpha \times$LB.

We will use the lower bound technique to analyze our algorithm for Vertex Cover. For this, we will first formulate the vertex cover problem as an integer program (a linear program with integrality constraints). Then we will remove the integrality constraints to obtain a linear program; this is our linear programming relaxation. Since, for this linear program, we are optimizing over a set P' that contains the set P of all integer solutions to our integer program, we obtain a lower bound LB on the optimum vertex cover value.

For Vertex Cover, we can describe the optimal solution as

$$\begin{aligned} \min \quad & \sum_v w(v)x(v) \\ \text{s.t.} \quad & x(u) + x(v) \geq 1 \qquad \forall (u, v) \in E \\ & x(v) \in \{0, 1\}. \end{aligned} \tag{1}$$

Using linear relaxation, we get a lower bound:

$$\text{LB} = \qquad \begin{aligned} \min \quad & \sum_v w(v)x(v) \\ \text{s.t.} \quad & x(u) + x(v) \geq 1 \qquad \forall (u, v) \in E \\ & x(v) \geq 0. \end{aligned} \tag{2}$$

We can find a solution to Vertex Cover at most twice this lower bound using the rounding algorithm of Hochbaum (around 1981). This algorithm proceeds by first obtaining the solution $x^*(v)$ of the linear program described in (2). A 2-approximation solution to Vertex Cover is then given by

$$S = \{v \in V : x^*(v) \geq \frac{1}{2}\}. \tag{3}$$

(Proof that this is a 2-approximation solution was given in the last lecture.)

We now have an algorithm, shown using the lower bound technique to satisfy

(i)	LB	$\leq$	OPT		
(ii)	$\sum_{v \in S} w(v)$	$\leq$	2LB	$\leq$	2OPT

The drawback of this algorithm is that it requires solving a linear program. Though it is polynomial time, it may still be more costly than desired. We will start the new material in this lecture by giving an alternate approach using duality, and again analyze it using the lower bound technique.

2 *Bar-Yehuda and Even* Vertex Cover

2.1 Duality

We take the dual program of the linear program described in (2):

$$\begin{aligned} \mathrm{LB} = \quad & \max \sum_{(u,v) \in E} y(u,v) \\ & \text{s.t.} \sum_{v:(u,v) \in E} y(u,v) \leq w(u) \\ & y(u,v) \geq 0 \quad \forall (u,v) \in E. \end{aligned} \tag{4}$$

LB is equal to the solution of this program by duality. For any y feasible in the dual LP, we have

$$\sum_{(u,v) \in E} y(u,v) \leq \mathrm{LB} \leq \mathrm{OPT}, \tag{5}$$

and we need only find such a y to have a lower bound on OPT.

2.2 2-Approximation Algorithm (Bar-Yehuda and Even)

Algorithm Start with an empty S and with y initialized to 0. Find an edge (u,v) that is not covered, and increase $y(u,v)$ until one of

$$\sum_{(u,x) \in E} y(u,x) \leq w(u)$$
$$\sum_{(v,x) \in E} y(v,x) \leq w(v)$$

becomes tight. (Since we are increasing y, it will remain nonegative and thus will be feasible.) Add the vertex for which the equation becomes tight to S, and proceed, stopping when S is a vertex cover.

$y(u,v) \leftarrow 0 \quad \forall (u,v) \in E$
$S \leftarrow \emptyset$
`while` S leaves an edge (u,v) not covered `do` {
  increase $y(u,v)$
  `until` $\exists z \in \{u,v\} : \sum_{(z,x) \in E} y(z,x) = w(z)$
  $S \leftarrow S \cup \{z\}$
}

Proof of Correctness:
(i) At each step, an uncovered edge is found and covered. Thus the algorithm terminates in $O(m)$ steps.
(ii) S is a vertex cover, since the algorithm does not terminate until this is true.
(iii) y is feasible in the dual LP given in (4), since we start with a feasible $y = 0$ and at each step increase $y(u,v)$ only to the limit of feasibility.
(iv) When adding z to S in the algorithm, the satisfied equality gives us $\sum_{(z,x)\in E} y(z,x) = w(z)$. This value is never altered by the algorithm afterwards, since the algorithm only alters $y(u,v)$ for $u \notin S$ and $v \notin S$. Therefore on termination we have

$$\forall v \in S \quad \sum_{(v,x)\in E} y(v,x) = w(v). \tag{6}$$

Using this statement, we can reexpress the weight of S as

$$\begin{aligned} \sum_{v\in S} w(v) &= \sum_{v\in S} \sum_{(v,x)\in E} y(v,x) && (7) \\ &\leq 2 \sum_{(a,b)\in E} y(a,b) && (8) \\ &\leq 2\mathrm{OPT} && (9) \end{aligned}$$

(8) is true because each edge can be counted at most twice (once for $v = a$ in the outer sum and once for $v = b$). (9) was shown to be true in (5). $\square$

The above algorithm is often referred to as a *primal-dual* algorithm since it constructs both a (primal) integer solution and a dual solution for the linear programming relaxation. We will now consider a more complicated example based on this primal-dual paradigm.

3 Generalized Steiner Tree Problem

Note: Our appromixation solution of the generalized Steiner tree problem will basically be identical to the treatment of the matching problem in the approximation notes, with "generalized Steiner tree problem" substituted for "matching" (and a few other small differences).

3.1 Problem Formulation

The generalized Steiner tree problem (GSTP) is: Given a graph with edge costs and a set of pairs of vertices in the graph that you wish to connect, find a minimum cost subgraph where all pairs in the set are connected. Formally: Given a graph $G = (V,E)$, costs $c(e) \geq 0 \ \forall e \in E$, and vertex pairs $T \subseteq V \times V$, find a subgraph F of minimum cost such that $\forall (s,t) \in T : s$ and t are connected in F.

GSTP is NP-hard. The best known solution is a 2-approximation algorithm. There are two major variations of GSTP 2-approximation: one is due to Goemans and Williamson, the other to Jain. This lecture will focus mostly on the algorithm of Goemans and Williamson.

First, let's formulate GSTP as an integer program:

$$x(e) = \begin{cases} 1 & e \in F \\ 0 & e \notin F \end{cases}$$

where our objective function is:

$$\text{Min} \sum_{e\in E} c(e)x(e)$$

We can express connectivity for a vertex pair $(s,t) \in T$ as follows: Given any $s-t$ cut S, one of the edges in the coboundary (i.e., one of the edges that crosses the cut) must be in F.

$$\delta(S) = \{(u,v) \in E : |\{u,v\} \cap S| = 1\} \Rightarrow \delta(S) \cap F \neq \emptyset$$

So if we define the set of all such cuts as $\mathcal{F}$:

$$\mathcal{F} = \{S : |S \cap \{s,t\}| = 1 \text{ for some } (s,t) \in T\}$$

then we can express the connectivity constraint with one inequality for each set in $\mathcal{F}$, and our GSTP integer program becomes:

$$\text{Min} \sum_{e \in E} c(e)x(e)$$

$$\text{s.t.} \qquad \sum_{e \in \delta(S)} x(e) \geq 1 \quad S \in \mathcal{F} \tag{10}$$
$$x(e) \in \{0,1\}$$

It should be clear that this is a correct formulation of GSTP. If this program yields an infeasible solution for GSTP, then there will be some unsatisfied pair $(s,t) \in T$ that is not connected. But this means there is some $s-t$ cut for which no coboundary edge is present in F, which contradicts (10).

This integer program solves GSTP, so it is NP-hard. We can make it solvable in polynomial time, however, by applying linear relaxation.

3.2 Linear Relaxation

Relax the integrality constraint to obtain:

$$LB = \text{Min} \sum_{e \in E} c(e)x(e) \leq OPT$$

$$\text{s.t.} \qquad \sum_{e \in \delta(S)} x(e) \geq 1 \quad S \in \mathcal{F}$$
$$x(e) \geq 0$$

3.2.1 Solution with the Ellipoid Algorithm

The relaxed program seems to still be hard to solve, because there are exponentially many inequalities. However, recall that we can use the ellipsoid algorithm to solve programs with large numbers of inequalities if we can find an efficient way to do separation, i.e., if we can determine in polynomial time a violated inequality for a point, if one exists.

An efficient separation algorithm for GSTP works by computing the maximum flow between pairs: our goal is to identify violated cuts with value less than one, and we know that maximum flow is the dual problem of minimum cut. Begin by setting x as the capacity on every edge. Then, for every pair of vertices $(s,t) \in T$, compute the max flow between s and t. If the max flow is at least one, then the pair is satisfied. Otherwise, the flow gives a min cut $S \in \mathcal{F}$ that is not satisified.

So, the relaxed program can be solved in polynomial time with the ellipsoid algorithm. Jain's algorithm repeatedly solves the linear programming relaxtion to optimality. We will devote the remainder of our discussion to a different approximation algorithm (due to Goemans and Williamson) that is primal-dual and constructs simultaneously a good integer solution and a dual solution to the relaxed program.

Note that our program's constraints are formulated in terms of cuts, but alternatively we could have written our program in terms of flows. This alternative program would have a polynomial number of variables and constraints.

3.2.2 Solution with Duality

The dual of the relaxed program is:

$$\text{Max} \sum_{S \in \mathcal{F}} y_S$$

$$\text{s.t.} \qquad \sum_{S \in \mathcal{F}: e \in \delta(S)} y_S \leq c(e) \ \forall e$$

$$y_S \geq 0$$

That is, for any edge e, for every set S that e crosses, the sum of the weights placed on e by the y_S's such that e is in the coboundary of S cannot exceed e's cost.

We don't need to solve for the optimum of the dual: *any* feasible solution y is a lower bound on OPT. The algorithm we will present constructs:

1. $y \geq 0 : \sum_{S \in \mathcal{F}: e \in \delta(S)} y_s \leq c(e)$
2. a feasible set of edges F

such that:

$$\sum_{e \in F} c(e) \leq 2 \sum_{S \in \mathcal{F}} y_S \leq 2OPT$$

The first inequality will hold because of how our algorithm constructs F and y. The second inequality holds because of strong duality. We'll construct y and F simultaneously, but initially with more emphasis placed on y.

Algorithm Start with an empty F and with y intialized to zero. Consider the connected components defined by F (initially, the only connected components will be the individual vertices) that are in $\mathcal{F}$ (i.e., the components that cut one or more vertex pairs). Increase y for edges adjacent to these connected components as much as possible without violating any $\sum y_s \leq c(e)$ constraints. Add the edge that becomes tight to F, and proceed, stopping when there are no more connected components in $\mathcal{F}$.

$y_S \leftarrow 0 \quad \forall S \in \mathcal{F}$
$F \leftarrow \emptyset$
$i \leftarrow 1$
connected components $\mathcal{C} = \{\{v\} : v \in V\}$
`while` $\mathcal{C} \cap \mathcal{F} \neq \emptyset$ `do` {
  increase $y_S \quad \forall S \in (\mathcal{C} \cap \mathcal{F})$
  `until` a new edge e satisfies $c(e) = \sum_{S \in \mathcal{F}: e \in \delta(S)} y_S$
  $e_i \leftarrow e$
  $F \leftarrow F \cup \{e_i\}$
  $i \leftarrow i + 1$
  update connected components $\mathcal{C}$
}

The y computed by this algorithm is exactly the y that we seek to construct. F, however, is not quite what we need: it has too many edges. Therefore, we perform another step that considers the edges in F in the reverse order of how F was constructed. If an edge can be removed from F without violating pair connectivity, it is removed.

```
for k ← i − 1 down to 1
    if F \ {e_k} is feasible then
        F ← F \ {e_k}
```

Proof of Correctness It is clear that y is feasible because of how it is constructed: y is initially feasible, and when we increase some y_S by ε, we stop if a cost constraint becomes tight.

In order to show that F is feasible, we need a lemma.

Lemma 1 *Let $\mathcal{C}$ be the connected components of F. If $\mathcal{C} \cap \mathcal{F} = \emptyset$, then F is feasible.*

Proof of Lemma 1: Assume not. Then $\exists \mathcal{C}, \mathcal{F}$ s.t. $\mathcal{C} \cap \mathcal{F} = \emptyset$ and F is not feasible. Since F is not feasible, $\exists (s,t) \in T$ s.t. s, t not connected. Let $C_s \neq C_t$ be connected components of F containing s and t, respectively. C_s is a set that forms an $s-t$ cut (since s is in C_s and t is not), and, likewise, C_t is a set that forms a $t-s$ cut. So $C_s, C_t \in \mathcal{F} \Rightarrow \mathcal{C} \cap \mathcal{F} \neq \emptyset$. Contradiction. □

The build step of the algorithm terminates with the condition that $\mathcal{C} \cap \mathcal{F} = \emptyset$, so we know from Lemma 1 that F is feasible at this point. The edge removal step of the algorithm only removes an edge from F if doing so maintains the feasibility of F. So F must be feasible when the algorithm terminates.

All that remains to be shown is that the cost of the solution is not much more than the cost of the dual solution. If we call the F that we compute after the initial build step F and the F after excess edges have been removed F', then this can be expressed as:

$$\sum_{e \in F'} c(e) \leq 2 \sum_{S \in \mathcal{F}} y_S \tag{11}$$

We can remove costs from the inequality as follows:

$$\begin{aligned} \sum_{e \in F'} c(e) &= \sum_{e \in F'} \sum_{S \in \mathcal{F}: e \in \delta(S)} y_S && (12) \\ &= \sum_{S} |\delta(S) \cap F'| y_S && (13) \end{aligned}$$

(12) holds because each edge $e \in F'$ satsifies $c(e) = \sum_{S \in \mathcal{F}: e \in \delta(S)} y_S$ when it is first added to F, and we never change the value of y_S for a set $S : e \in \delta(S)$ later in the execution of the algorithm. To see why the value of y_S never subsequently changes, observe that the inclusion of e in F' creates a new connected component with e in its interior, and this component will now no longer be equal to any cut S for which e is a coboundary edge. (13) holds because in (12) we count each y_S for every edge that belongs to both F' and $\delta(S)$.

In the next lecture we will show that the coefficients of the y_S's, considered under some sort of averaging, are less than two. This will prove (11) and complete our correctness proof of the GSTP algorithm.