18.415/6.854 Advanced Algorithms December 10, 2001

Problem Set Solution 5

Lecturer: Michel X. Goemans

1. **Consider the linear programming relaxation of the vertex cover problem seen in class.**

$$\textbf{Min} \quad \sum_{i \in V} w_i x_i$$

subject to:

$$\begin{aligned} x_i + x_j &\geq 1 \qquad && (i,j) \in E \\ x_i &\geq 0 && i \in V \end{aligned}$$

(a) **Argue that any basic feasible solution x of the above linear program must satisfy $x_i \in \{0, \frac{1}{2}, 1\}$ for all vertices $i \in V$.**
Hint: given a bfs x, consider the vector y defined by $y_i = x_i$ if $x_i \in \{0,1\}$, and $y_i = 0.5$ otherwise.

Let x be a basic feasible solution. We follow the hint and let y be defined by $y_i = x_i$ if $x_i \in \{0,1\}$, and $y_i = 0.5$ otherwise. Let A be the matrix of coefficients of the constraints $x_i + x_j \geq 1$. From problem 1 on the linear programming problem set we know that we can find a matrix A^I_J such that,

i. A^I_J is a square, invertible submatrix of A.
ii. If J is the set of indices of non-zero components of x then $A^I_J x_J = b^I$ where b^I is a vector of all 1's.

By construction J is the set of indices of non-zero components of y. An equation contained in $A^I_J x_J = b^I$ is either of the form $x_i + x_j = 1$ or else it is of the form $x_i = 1$. In the first case since $i, j \in J$ we have $x_i > 0$ and $x_j > 0$. This implies that both x_i and x_j are strictly less than 1 and so $y_i = y_j = 0.5$. Hence $y_i + y_j = 1$. In the second case we have $y_i = 1$ since $x_i = 1$. We have therefore shown that $A^I_J y_J = b^I$. But A^I_J is an invertible matrix and so $x_J = y_J \Rightarrow x = y$. i.e., $x_i \in \{0, \frac{1}{2}, 1\}$ for all vertices $i \in V$.

(b) **To solve the linear program to optimality, we can therefore restrict our attention to solutions satisfying $x_i \in \{0, 0.5, 1\}$. For this purpose, consider the bipartite graph obtained by introducing two vertices say a_i and b_i for every vertex i, both of weight w_i, and having edges (a_i, b_j) and (a_j, b_i) for every edge (i,j) of the original graph. Show that the minimum weight of any vertex cover in this bipartite graph is exactly equal to twice the value of the above linear program. Also, how can you extract the solution of the LP from the vertex cover in the bipartite graph and vice versa?**

Let x^* be an optimal solution to the linear program satisfying $x^*_i \in \{0, \frac{1}{2}, 1\}$. We form a subset S of the nodes in the bipartite graph as follows. If $x^*_i = 1$ then both a_i and b_i are put into S. If $x^*_i = \frac{1}{2}$ then a_i but not b_i is put into S. If $x^*_i = 0$ then neither a_i nor b_i is put into S. The set S is a vertex cover of the bipartite graph since,

i. $(i,j) \in E \Rightarrow x^*_i + x^*_j \geq 1$.
ii. $x^*_i = 1 \Rightarrow a_i, b_i \in S$.

iii. $x_i^* = x_j^* = \frac{1}{2} \Rightarrow a_i, a_j \in S$.

Conversely, given a vertex cover T of the bipartite graph we can obtain a feasible solution y to the linear program by setting $y_i = 0$ if neither a_i nor b_i is in T, $y_i = \frac{1}{2}$ if exactly one of them is in T and $y_i = 1$ if both are in T. By construction the weight of S is $\sum_{i \in V} w_i(2x_i^*)$ and the weight of T is $\sum_{i \in V} w_i(2y_i)$. This implies that S is a minimum weight vertex cover since $\sum_{i \in V} w_i x_i^* \le \sum_{i \in V} w_i y_i$. Hence the minimum weight of any vertex cover in the bipartite graph is exactly equal to twice the value of the linear program. The above constructions show how to obtain the solution of the LP from the vertex cover in the bipartite graph and vice versa.

(c) **Show that the problem of finding a minimum weight vertex cover in a bipartite graph can be solved by a minimum cut computation or a maximum flow computation in a related graph.**

Let $V = A \cup B$ be the partition of vertices associated with the bipartite graph. We first of all direct all edges from A to B. We also add a source node s which we connect to each node in A and a sink node t which we add to each node in B. If $i \in A$ then the edge (s, i) is given capacity w_i. If $j \in B$ then the edge (j, t) is given capacity w_j. All of the original edges are given infinite capacity. Suppose that S is a vertex cover in the bipartite graph. Consider the set $C = (A - S) \cup (B \cap S) \cup \{s\}$ and its associated cut $(C, \bar{C})$. An edge (i, j) of infinite capacity cannot be a member of $\delta(C)$ since that would mean that $i \notin S$ and $j \notin S$ contradicting the fact that S is a vertex cover. An edge (s, i) is in $\delta(C)$ if and only if $i \in S$ and an edge (j, t) is in $\delta(C)$ if and only if $j \in S$ and so the value of the cut is the same as that of the vertex cover. Conversely, suppose that $(C, \bar{C})$ is a cut of finite weight. Let $S = (A - S) \cup (B \cap S)$. Suppose that (i, j) is an edge in the bipartite graph such that $i, j \notin S$. Then $i \in C$ and $j \in \bar{C}$ which means that $(i, j) \in \delta(C)$. This contradicts the fact that $(C, \bar{C})$ is a cut of finite weight. Hence S is a vertex cover of the bipartite graph. It can easily be seen that the weight of the vertex cover is the same as that of the cut. Hence we can find a minimum weight vertex cover in a bipartite graph by finding a minimum cut $(C, \bar{C})$ and setting $S = (A - S) \cup (B \cap S)$. (By the max-flow min-cut theorem we could find the vertex cover by first solving a maximum flow problem.)

2. **Consider the 2-approximation algorithm seen in class for the generalized Steiner tree problem (we are given a set T of pairs of vertices and cost on the edges of a graph, and the goal is to find a subgraph (a forest) of minimum cost in which every pair of vertices in T is connected).**

   (a) **Argue that this problem is a generalization of the minimum spanning tree problem.**

   Let T be the set of all pairs of the vertices of the graph. It is obvious that any minimal subgraph that connects every pair of vertices in T is a minimum spanning tree.

   i. **Does the algorithm seen in class produce a minimum spanning tree in that case? If so, prove it; if not, give a counterexample.**

   The algorithm from class does produce a minimum spanning tree in this case. We first prove the following claim about the order in which the algorithm adds edges to the subgraph.

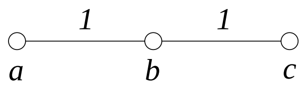
   Claim 1 *In each step, the algorithm adds the edge of least cost that connects any two connected components.*

Proof: By the definition of T, throughout the algorithm, every connected component $C \in \mathcal{C}$ is in $\mathcal{F}$. Therefore, in each step we increase the value of $d(i)$ for *every* vertex in the graph by the same amount. Thus, all vertices have the same value of $d(i)$ at any time during the execution of the algorithm. Now, notice that in each step, we pick an edge ij with $i \in C_p, j \in C_q$, $p \neq q$, that minimizes $(c_{ij} - d(i) - d(j))/2$, or equivalently, minimizes c_{ij}. $\square$

By the above claim, our algorithm adds the edges to the subgraph in the same order as Kruskal's minimum spanning tree algorithm. Therefore, by the optimality of Kruskal's algorithm (see, for example, *Introduction to Algorithms* by Cormen, Leiserson, and Rivest), and considering the fact that by the definition of T the second phase of the algorithm doesn't change the subgraph, the algorithm outputs a minimum spanning tree of the graph.

ii. Is the value ($\sum_S y_S$) of the dual solution y constructed equal to the minimum spanning tree value? If so, prove it; if not, give a counterexample.

Consider the graph G in the following figure.



It is easy to see that the dual solution y constructed by the algorithm has $y_{\{a\}} = y_{\{b\}} = y_{\{c\}} = 1/2$, and $y_S = 0$ for every $S \neq \{a\}, \{b\}, \{c\}$. Therefore, the value of the dual solution is $3/2$, whereas the minimum spanning tree has cost 2.

(b) **Argue that this problem is a generalization of the shortest s-t path problem (in an undirected graph with nonnegative edge weights).**

It is easy to see that the shortest s-t path problem is equivalent to the generalized Steiner tree problem with $T = \{(s,t)\}$.

i. Does the algorithm seen in class produce a shortest s-t in that case? If so, prove it; if not, give a counterexample.

ii. Is the value ($\sum_S y_S$) of the dual solution y constructed equal to the shortest path value? If so, prove it; if not, give a counterexample.

We prove that the cost of the path F' that the algorithm produces is equal to the value ($\sum_S y_S$) of the dual solution y that it constructs. Since the value of y, and the cost of F' are lower and upper bounds for the value of the shortest s-t path, we conclude that the answer to both of the above questions is positive.

It just remains to prove $\sum_{S \in \mathcal{F}} y_S = \sum_{e \in F'} c_e$. We proved in the class that

$$\sum_{e \in F'} c_e = \sum_{S \in \mathcal{F}} |F' \cap \delta(S)|.y_S. \tag{1}$$

It is clear from the description of the algorithm that throughout the algorithm (before the end of the *while* loop), $\mathcal{C}$ contains only two connected components, one containing s and the other one containing t. Both of these connected components C must satisfy $|F' \cap \delta(C)| = 1$, for otherwise there must be a cycle in F. This implies that for every set

S with $y_S \neq 0$, we have $|F' \cap \delta(S)| = 1$. This fact, together with Equation (1) completes the proof.

3. **We would like to design an approximation algorithm for the following problem. We are given an undirected graph G with cost c_e for every edge e, 2 disjoint sets A and B of vertices of the same size, and we would like to find a minimum cost set H of edges such that in every connected component of H, we have the same number of vertices of A and B (so for example a matching between A and B would be one possible solution, and a spanning tree would be another).**

   Show that the approach used to design the 2-approximation algorithm seen in class for the generalized Steiner problem can be applied here to get a 2-approximation algorithm as well. Do not reprove everything, but state and prove everything (in the algorithm and/or in the proof) that differs from the case of the generalized Steiner tree problem seen in class.

   To solve this problem, we can use the algorithm for generalized Steiner tree problem. The only difference is that we define $\mathcal{F} := \{S \subset V : |S \cap A| \neq |S \cap B|\}$. Our algorithm guarantees $\mathcal{C} \cap \mathcal{F}$ is empty at the end, so the correctness follows. The analysis is also the same except for the proof of the following lemma.

   Lemma 2 *Consider the set $\mathcal{C}$ of the connected components at an arbitrary time during the execution of the algorithm and let $E := \mathcal{C} \cap \bar{\mathcal{F}}$, $O := \mathcal{C} \cap \mathcal{F}$. Let H be the graph with the vertex set $\mathcal{C}$ that has an edge between C_p and C_q iff there is an edge between C_p and C_q in the final subgraph F'. Then, there does not exist a vertex C in E with $d_C = 1$.*

   Proof: Suppose there is a $C \in E$ with $d_C = 1$. Let e be the edge in F' that crosses C. As we remove edges in reverse order in the second phase of the algorithm, when we consider removing e, C is still a connected component. Therefore, since $d_C = 1$, removing e creates two new connected components C and C'. Since $C \notin \mathcal{F}$, $|A \cap C| = |B \cap C|$. Also, since the subgraph is feasible before removing e, we have $|A \cap (C \cup C')| = |B \cap (C \cup C')|$ and therefore $|A \cap C'| = |B \cap C'|$. So, removing e leaves the subgraph feasible. This means that e should be removed in the second phase of the algorithm and therefore $e \notin F'$, which is a contradiction. □

4. **Consider the maximum weight matching problem in a (non-bipartite) graph $G = (V, E)$. More precisely, given a non-negative weight w_{ij} for each edge $(i,j) \in E$, the problem is to find a (not necessarily perfect) matching of maximum total weight. Consider the following greedy algorithm: start from an empty matching and repeatedly add an edge of maximum weight among all edges which do not meet any of the edges chosen previously. Stop as soon as the matching is maximal (i.e. no other edge can be added). Let M_G denote the greedy matching and Z_G its cost. You are asked to show that the greedy algorithm is a 2-approximation algorithm.**

   Show that the following linear program gives an upper bound Z_{LP} on the optimal value Z_M of the maximum weight matching problem.

$$\textbf{Min} \quad \sum_{i \in V} u_i$$

subject to:

$$\begin{aligned} u_i + u_j &\geq w_{ij} & \quad (i,j) \in E \\ u_i &\geq 0 & \quad i \in V. \end{aligned}$$

The dual to the above linear program is:

$$Z_{max} = max \sum_{(i,j)\in E} x_{ij} w_{ij}$$

subject to

$$\begin{aligned} \sum_{j:(i,j)\in E} x_{ij} &\leq 1 & \forall i \in V \\ x_{ij} &\geq 0 & \text{for } (i,j) \in E \end{aligned}$$

Since this is the dual, $Z_M \geq Z_{max}$. Consider a matching in G. If we set $x_{ij} = 1$ if (i,j) is in the matching and 0 otherwise, then we obtain a feasible solution to the LP. But we are then maximizing over the weight of the matching, so $Z_{max} \geq Z_{OPT}$, the weight of the max cost matching.

From the greedy matching M_G, construct a feasible solution u to the above linear program and show that its value is $2Z_G$. Conclude that $Z_G \geq \frac{1}{2}Z_M$.

Given the matching M_G we set $u_i = u_j = w_{ij}$. To see that this is feasible, we need only show that $u_i + u_j \geq c_{ij}$. But we know that each edge in the graph has a vertex which is connected to an edge in M_G. If this were not the case, the matching would not be maximal. Suppose i is this vertex, and the edge in M_G is (i,k). Then $u_i = w_{ik} \geq c_{ij}$ because otherwise edge (i,j) would have been placed into M_G rather than (i,k). So the solution is feasible. $\sum_i u_i$ counts the weight of each edge in M_G twice (once for each of the two vertices) so $\sum_i u_i = 2Z_G$. This yields $Z_G \geq \frac{1}{2}Z_M$ directly.