601.435 / 601.635 Approximation Algorithms **Spring 2026**
Homework #5 **Due:** April 16, 2026, 9:00am

Reminder: you may work in groups of up to three people, but must write up solutions entirely on your own. Collaboration is limited to discussing the problems – you may not look at, compare, reuse, etc. any text from anyone else in the class. Please include your list of collaborators on the first page of your submission. Many of these problems have solutions which can be found on the internet – please don't look. You can of course use the internet (including the links provided on the course webpage) as a learning tool, but don't go looking for solutions.

Please include proofs with all of your answers, unless stated otherwise.

1 Multiway Cut (50 points)

Consider the following two permutations π_1 and π_2 of $[k]$, where $\pi_1(1) = 1, \pi_1(2) = 2, \ldots, \pi_1(k) = k$ and $\pi_2(1) = k, \pi_2(2) = k-1, \ldots, \pi_2(k) = 1$.

(a) (25 points) Consider a modification of the 3/2 approximation for Multiway Cut from Lecture 17: instead of choosing π uniformly at random from all permutations of $[k]$, we choose $\pi = \pi_1$ with probability 1/2 and choose $\pi = \pi_2$ with probability 1/2. Prove that this modified algorithm is still a 3/2-approximation for Multiway Cut.

Solution: Lemma 17.2.2 still implies that the algorithm is a 3/2-approximation, so we just need to prove that Lemma 17.2.2 is still true for the modified algorithm. So consider so $\{u, v\} \in E$. Lemma 17.2.3 is independent of the permutation, and the event that terminal i cuts $\{u, v\}$ (the event X_i) is also independent of the permutation, so it is still true that $\Pr[X_i = 1] \leq |x_u^i - x_v^i|$. Thus it is still true that for the special terminal ℓ, $\Pr[X_\ell = S_\ell = 1] \leq |x_u^\ell - x_v^\ell|$. For some terminal $i \neq \ell$, as before, a necessary condition for i to settle $\{u, v\}$ conditioned on the fact that it also cuts $\{u, v\}$ is for it to appear before ℓ in the permutation. With our new algorithm, this probability is still 1/2, since of the two permutations we are choosing from, i is before ℓ in exactly one of them. Hence we still have that $\Pr[S_i = X_i = 1] \leq \frac{1}{2}|x_u^i - x_v^i|$. The rest of the analysis from class goes through without change.

(b) (25 points) Using the previous part, design a *deterministic* 3/2-approximation for Multiway Cut. As always, prove the approximation ratio and polynomial running time.

Solution: Let π be one of the two permutations π_1 or π_2. Then if we force the algorithm to use π, there are at most nk possible cuts corresponding to different choices of r (this is because if we order all distances between terminals and nonterminals from smallest to largest there are nk different distances, and two value of r that lie between the same two distances will give the same cut). Hence our algorithm can just do the following for both π_1 and π_2: order the terminal-nonterminal distances from smallest to largest, and for each of the resulting nk intervals compute the associated cut if we use the given permutation and a value of r from that interval. This gives us at most $2nk$ different cuts, so we choose the best of them.

This algorithm runs in polynomial time since we only have to try $2nk$ different combinations of π and r, and for each resulting cut we can determine its cost in polynomial time. Hence the total running time is polynomial.

For the approximation ratio, we know from part a that if choose r uniformly at random from $(0, 2)$ and choose π uniformly at random from $\{\pi_1, \pi_2\}$, then the expected approximation ratio is at most 3/2. This implies that there exists some cut that the algorithm *might* generate which has approximation ratio at most 3/2. Since our algorithm tries all cuts that the randomized algorithm might generate and then returns the best one, this implies that our algorithm is a 3/2-approximation.

2 Multicut in Trees (50 points)

Consider the *multicut problem in trees*. In this problem, we are given a tree $T = (V, E)$, k pairs (s_i, t_i) of vertices, and edge costs $c : E \to \mathbb{R}^+$. A feasible solution is a set $F \subseteq E$ such that for all $i \in [k]$, s_i and t_i are in different connected components of $T - F$. The objective is to minimize the total edge cost $\sum_{e \in F} c(e)$.

Let P_i be the unique path between s_i and t_i in T. Then we can write an integer linear programming formulation of this problem:

$$\begin{aligned} \min \quad & \sum_{e \in E} c(e) x_e \\ \text{subject to} \quad & \sum_{e \in P_i} x_e \geq 1 \quad \forall i \in [k] \\ & x_e \in \{0, 1\} \quad \forall e \in E \end{aligned}$$

(a) (25 points) Write the dual of the LP relaxation of the above ILP (note: we did this in class for multicut!)

Solution:

$$\begin{aligned} \max \quad & \sum_{i=1}^{k} y_i \\ \text{subject to} \quad & \sum_{i : e \in P_i} y_i \leq c(e) \quad \forall e \in E \\ & y_i \geq 0 \qquad\qquad \forall i \in [k] \end{aligned}$$

Suppose that we root the tree at an arbitrary vertex r. Let $depth(v)$ be the number of edges on the path from v to r. Let $lca(s_i, t_i)$ be the vertex v on the path from s_i to t_i whose depth is minimum. Suppose that we use the primal-dual method to solve this problem, where the dual variable that we increase in each iteration corresponds to the violated (primal) constraint that maximized $depth(lca(s_i, t_i))$. After all primal constraints are satisfied, we do a "reverse cleanup" stage like in Steiner Forest, where we look at the edges we added in reverse order and remove them if we can do so while still having a feasible solution.

(b) (25 points) Prove that this is a 2-approximation. Hint: consider a path P_i where the dual variable is nonzero. How many edges in the final solution can be on the path from s_i to $lca(s_i, t_i)$, and how many can be on the path from t_i to $lca(s_i, t_i)$?

Solution: Obviously this algorithm terminates in polynomial time and returns a feasible solution, so we just need to argue about the approximation factor.

Let F' denote the set of edges added by the algorithm, and let $F \subseteq F$ be the set returned by the algorithm (i.e., that survive the reverse cleanup procedure). First, it is easy to see that the y is always a feasible dual solution throughout the algorithm. This is by induction. It is obviously true at the beginning of the algorithm. So suppose that it is true at the beginning of some iteration. Note that when the dual constraint for an edge e is tight we add e to F', and so any i for which $e \in P_i$ becomes satisfied and so will not be raised in the future. Hence whenever we raise a dual variable it does not appear in the constraint of any tight edge e, and so we never violate any dual constraints. Thus y is feasible at the end of the iteration.

We now prove a useful lemma:

Lemma 1 *If $y_i > 0$, then $|F \cap P_i| \leq 2$.*

Proof: Let $\ell_i = lca(s_i, t_i)$, and let P denote the subpath of P_i from s_i to ℓ_i. We will prove that $|F \cap P| \leq 1$. Since a symmetric argument will imply the same thing for the subpath of P_i from t_i to ℓ_i, this will imply the lemma.

Suppose for contradiction that there are two edge from F in P. Let's call them e_a and e_b, and say that e_a was added in the iteration where we were raising y_a and e_b was added in the iteration where we were raising y_b (note that a or b could equal i). Without loss of generality, we will assume that e_a is lower (further form the root) than e_b (we can always just change which we call a and which we call b). Since $y_i > 0$, when we were raising y_i we must not yet have added e_a or e_b to F', or else i would not have been a violated primal constraint (it would have been satisfied already). Hence by our ordering, we know that $lca(s_a, b_a)$ and $lca(s_b, t_b)$ have depth at most the depth of ℓ (are closer to the root). But this means that P_a includes at least the part of P from e_a to ℓ, and in particular include e_b. So y_a must have been raised before y_b, since otherwise a would have been a satisfied constraint (not a violated constraint) and so y_a would not have been raised at all.

Now consider the reverse cleanup step. Since y_a was raised before y_b we know that in the reverse cleanup step e_b is considered before e_a. We also know that e_b survives, since $e_b \in F$. Consider what happens when we analyze e_a in the reverse cleanup step. Since $e_b \in F$, we know that e_a is not necessary for demand a, or for any demand with one endpoint a descendent of e_b and with lca an ancestor of a_b. But for any demand m with lca in the part of P between e_a and e_b, the depth of its lca is larger than the depth of ℓ, and so it was considered before i (and so before a and b) by the algorithm, and so e_a cannot be necessary for it to be satisfied. Hence e_a is not actually necessary for *any* demand. But this is a contradiction, since then the reverse cleanup step would remove e_a. ■

Now that we have this lemma, we can analyze the cost of the algorithm. We have

$$
\begin{aligned}
c(F) &= \sum_{e \in F} c(e) \\
&= \sum_{e \in F} \sum_{i : e \in P_i} y_i && \text{(algorithm only adds edges with tight dual constraint)} \\
&= \sum_{i \in [k]} \sum_{e \in F \cap P_i} y_i && \text{(switch order of summations)} \\
&= \sum_{i \in [k]} |F \cap P_i| y_i \\
&\leq \sum_{i \in [k]} 2 y_i && \text{(Lemma 1)} \\
&\leq 2 \cdot OPT && \text{(weak duality)}
\end{aligned}
$$