

# CS599 (Spring 2010) — Takehome Final

Due in my office by Tuesday, 05/11, 5pm

No late submission will be granted, so you may want to start early. Contrary to the policy on homeworks, you cannot discuss problems, ideas, or solutions to this final with anyone (in or outside of class) except myself. Also, as a reminder, you are not allowed to seek solutions to these problems online or elsewhere. If you are stuck on a problem and need hints, or you need help understanding a problem, you are welcome to e-mail me, or talk to me in person. If I am not in the office when you submit, feel free to slide your exam under the door.

## Problem 1

Here is a variant of the graph multiway cut problem. As before, each edge has a non-negative cost  $c_e$ . You are to divide the graph into  $k$  partitions  $i = 1, \dots, k$ . For each node  $v$ , there is a cost  $b(v, i)$  for putting node  $v$  in partition  $i$ . Your goal is to find a partition  $S_1, \dots, S_k$  of the nodes minimizing

$$\sum_i \sum_{v \in S_i} b(v, i) + \sum_i \sum_{j \neq i} \sum_{e \in (S_i, S_j)} c_e,$$

i.e., the total cost of the assignments of nodes to their partitions and the cut edges. Notice that if we designate  $k$  sources  $s_i$  with  $b(s_i, i) = 0$  and  $b(s_i, j) = \infty$  for  $j \neq i$ , and set  $b(v, i) = 0$  for all other  $v$  and  $i$ , then this is just the standard Multiway Cut problem. Here, you are to derive a randomized 2-approximation algorithm for this problem.

- [2 points] Write an Integer Program and its LP relaxation for this problem. (Hint, use variables  $x_{i,v}$  capturing whether node  $v$  is assigned to partition  $i$ .)
- [2 points] Prove that if you assign  $v$  to a partition  $i$  with probability  $x_{v,i}$ , and simply do that independently, the approximation guarantee can be arbitrarily bad.
- [6 points] Derive a randomized 2-approximation algorithm for this problem. (Hint: Repeat the following until all nodes are assigned: pick a partition  $i$  and a threshold  $\rho \in [0, 1]$  independently and uniformly at random, and include in  $S_i$  all nodes  $v$  with  $x_{i,v} \geq \rho$ .)

## Problem 2

Consider again the single-swap Local Search algorithm for  $k$ -median. Instead of comparing our own solution to the optimum, let us compare it to a more restricted optimum: the optimum solution with only  $\frac{2}{3} \cdot k$  facilities. (You may assume that  $\frac{2}{3} \cdot k$  is an integer). If  $C$  is our total cost with  $k$  facilities, and  $C_{2/3}^*$  is the optimum cost with  $\frac{2}{3} \cdot k$  facilities, show that  $C \leq 3C_{2/3}^*$ . Thus, by comparing ourselves against a more restricted optimum, we improve the approximation guarantee from a factor 5 to a factor 3.

## Problem 3

For many of our approximation algorithms in the past, we had used explicit upper or lower bounds on OPT that we computed along the way, and for which we showed that we were within the desired approximation factor. We didn't do that for the continuous greedy algorithm for maximizing a submodular function over matroids. Let's catch up and do that now.

The notation is exactly as in class or in the notes. Define  $\hat{V} = \min_t (F(\vec{y}(t)) + \hat{v}(\vec{y}(t)) \cdot \nabla F(\vec{y}(t)))$ .

- [3 points] Show that  $\text{OPT} \leq \hat{V}$ .
- [7 points] Show that the final answer of the algorithm satisfies  $F(\vec{y}(1)) \geq (1 - 1/e) \cdot \hat{V}$ .

#### Problem 4

Here is another variant of SAT: as before, you are given a collection of 3-clauses  $C_j$ , where each clause consists of (exactly) 3 literals. However, instead of the usual definition of satisfying a clause, we say that  $C_j$  is satisfied if the values of the literals in the clause are not all equal. Thus, as usual, if all three literals are false, then  $C_j$  is not satisfied. But  $C_j$  is also not satisfied if all three literals are true. If there is at least one false and at least one true literal, then  $C_j$  is satisfied. As before, our goal is to maximize the number of satisfied clauses (under the new definition).

- (a) [3 points] Give and analyze a  $3/4$  approximation algorithm for this problem (Hint: Johnson's Algorithm).
- (b) [7 points] Prove that there is a constant  $\gamma > 0$  such that no polynomial-time algorithm can be better than a  $1 - \gamma$  approximation for this problem.