Problem Set 9

This problem set is due at **10:00 am** on **Tuesday, May 2nd**.

Problem 9- 1: Vertex Cover

Recall that the vertex cover problem has a 2-approximation. A k-hypergraph is a graph such that each 'edge' is a set of k vertices (Since each edge so far has been between two vertices, the graphs we have seen so far have been 2-hypergraphs). A vertex cover for a k-hypergraph is a subset of vertices $C \subset V$ such that each hypergraph edge has a vertex in C. Design a k-approximation algorithm for vertex cover on a k-hypergraph. Prove the correctness of your algorithm and analyze its runtime.

Solution:

Let each edge e in the k-hypergraph be expressed as the set of vertices $\{e_1, e_2, \ldots, e_k\}$. To find a k approximation, let x_v represent whether the vertex v is in the vertex cover. The algorithm will solve for values of $x_v \in [0, 1]$, then round them to $\bar{x}_v$. To find the x_v's, solve the following LP:

$$\begin{aligned} \text{Minimize } & \sum_{v \in V} x_v & \\ \text{Subject to } & \sum_{i=1}^{k} x_{e_i} \geq 1 & \forall e \in E \\ & x_v \leq 1 & \forall v \in V \\ & x_v \geq 0 & \forall v \in V \end{aligned}$$

Then, for each $x_v \geq 1/k$, set $\bar{x}_v$ to 1; for each $x_v < 1/k$, set $\bar{x}_v$ to 0. Add v to the vertex cover iff $\bar{x}_v = 1$.

The runtime of this algorithm is $O(|V|)$ plus the runtime of solving the LP with $O(|V|)$ variables and $O(|V| + |E|)$ constraints.

Theorem 1 *The above algorithm finds a k approximation to vertex cover.*

Proof. First, we have to prove that the algorithm finds a valid vertex cover. For every edge e, there exists an constraint on the LP such that $\sum_{i=1}^{k} x_{e_i} \geq 1$. For this to be true, at least one of the x_{e_i} must be at least $1/k$, so at least one of the $\bar{x}_{e_i}$'s will be rounded to 1; therefore, every edge must have at least one adjacent vertex in the VC.

Next, we must show that the solution found is within a k factor of the optimal solution. Let x_v^* be 1 if v is in the optimal vertex cover, 0 otherwise. Notice that this assignment is in the feasible region of the LP. This means that, before rounding, $\sum_{v \in V} x_v \leq \sum_{v \in V} x_v^*$, because the assignment of x_v is optimal for the LP. After rounding, each $\bar{x}_v$ is set to at most kx_v (if it was rounded up). Therefore, $\sum_{v \in V} \bar{x}_v \leq k \sum_{v \in V} x_v \leq k \sum_{v \in V} x_v^*$, so the $\bar{x}_v$'s form a k approximation.

Problem 9- 2: Load Balancing

In class we discussed two approximation algorithms for an NP-hard load balancing problem in which we are asked to distribute n tasks, each with a positive integer weight, over m machines in a way that minimizes the maximum load of any machine. The first algorithm that we presented guarantees a maximum load within a factor of two of optimal. The second algorithm that we presented guarantees a maximum load within a factor of $\frac{3}{2}$ of optimal. These two algorithms are also presented in Section 11.1 of the text, where they are referred to as GREEDYBALANCE and SORTEDBALANCE, respectively. The purpose of the present question is to develop a tighter analysis of algorithm SORTEDBALANCE. Specifically, we will prove that this algorithm achieves an approximation factor of $\frac{4}{3}$. Fix an instance I of the load balancing problem with m machines and n tasks. Number the tasks from 1 to n, and let w_i denote the positive integer weight of task i. Assume without loss of generality that the tasks are numbered in such a way that $w_i \geq w_{i+1}$, $1 \leq i < n$. Let L^* denote the minimum possible maximum machine load. Let k denote the number of tasks with weight greater than $L^*/3$.

(a) Explain why k is at most $2m$.
Proof:
Suppose $k \geq 2m+1$, then in any assignment, there is at least one machine with at least three tasks, each of weight $> \frac{L^*}{3}$. This implies there is at least one machine with load $> L^*$. This contradicts the optimality of L^*.

(b) Let L^{**} denote the minimum possible maximum machine load when tasks 1 through k are assigned to the machines (and the rest of the tasks are not assigned to any machine). Prove that at the point in its execution when algorithm SORTEDBALANCE has processed tasks 1 through k, the maximum load of any machine is exactly L^{**}. Hints: You may find it useful to consider the cases $0 \leq k \leq m$ and $m < k \leq 2m$ separately. In each case, begin by precisely characterizing of the way that algorithm SORTEDBALANCE maps tasks 1 through k to the machines.
Solution:
Observe that the optimal solution can assign at most 2 tasks to every machine. The optimal solution now is completely determined, because $w_1 \geq w_2 \geq \cdots \geq w_k$ and looks as follows $\{w_1\}, \{w_2\}, \ldots, \{w_{m-k+1}, w_k\} \ldots, \{w_m, w_{m+1}\}$. We now need to show that SORTEDBALANCE achieves the maximum load that the optimal assignment achieves. To see this, observe that for the first m assignments, SORTEDBALANCE

matches the optimal assignment. After this point SORTEDBALANCE might place either one or two more tasks on any machine. Suppose SORTEDBALANCE places 2 more tasks on machine j and the last task to be assigned was t, this could only have happened if $w_{j-1} > w_j + w_{d-1}$. We claim that this can happen at most once. Then task $t+1$ has to be assigned to machine $j-1$ because otherwise that would mean $w_{j-1} > 3\frac{L^*}{3} > L^*$ contradicting the optimality of L^*. Similar reasoning implies that after this point every machine has at most 2 tasks assigned to it. Now, we compare the maximum load of the optimal assignment and the assignment that SORTEDBALANCE finds. The maximum load of the optimal assignment is

$$\max\{w_1, w_2, \ldots, w_{m-k}, w_{m-k+1} + w_k, \ldots, w_m + w_{m+1}\}$$

the maximum load of the assignment found by SORTEDBALANCE is given by

$$\max\{w_1, w_2, \ldots, w_{m-k-2} + w_k, \ldots w_{m-t} + w_{t+2}, w_{m-t+1} + w_t + w_{t+1}, \ldots, w_m + w_{m+1}\}$$

Where $w_{m-t+1} + w_t + w_{t+1}$ is the only three-term element of the sequence. For each element in the second sequence, we will find an element in the first sequence that is bigger than or equal to it. Observe that for machines m to $m-t-1$ the sequence are the same. $w_{m-t+1} + w_t + w_{t+1} < w_{m-t} + w_{t+1}$, because otherwise the right hand side would be chosen as the configuration. The loads for machines $m-t-1$ to 1 can now be compared directly and the load of each machine in the optimal configuration can be shown to be bigger than or equal to the load of the machine in the assignment found by SORTEDBALANCE. This implies that the SORTEDBALANCE procedure finds something at least as good as the optimal solution for the first k tasks, and hence should be optimal for the first k tasks.

(c) Prove that algorithm SORTEDBALANCE achieves an approximation ratio of $\frac{4}{3}$. Hint: Use the same overall framework as we used to establish the $\frac{3}{2}$ bound, but improve the bound by making use of the results of parts (a) and (b).
Solution:
Observe that $L^* \geq \frac{1}{m}\sum_i^n w_i$. This is because max load is greater than or equal to the average load. For tasks 1 to k we have shown in part (b) that the task assignment is optimal.

Let's consider the assignment of any arbirtary task t_i such that $i > k$. (Remember $w_i \leq \frac{L^*}{3}$) The algorithm is greedy and thus will place t_i at the machine with the smalled load L_j. Since we know that L_j has the smallest load, we know that $m \times L_j \leq \sum_i^n w_i$ and $L_j \leq \frac{1}{m}\sum_i^n w_i \leq L^*$. The new load of the machine is $L_j + w_i \leq L^* + \frac{L^*}{3} = \frac{4L^*}{3}$.