COMP 5711: Advanced Algorithm
2014 Fall Semester
Written Assignment # 2

Chapter 11

Problem 11 (20 pts) Consider the following variant of the knapsack problem. There are n items, each with a weight w_i and a value v_i. Now, you're told that there is a subset O whose total weight is $\sum_{i \in O} w_i \leq W$, where W is the weight limit, and whose total value is $\sum_{i \in O} v_i = V$ for some V. For a given fixed $\epsilon > 0$, design a polynomial-time algorithm that finds a subset of items $\mathcal{A}$ such that $\sum_{i \in \mathcal{A}} w_i \leq (1+\epsilon)W$ and $\sum_{i \in \mathcal{A}} v_i \geq V$, i.e., your algorithm does at least as good as O in terms of value, but it may exceed the weight limit slightly. Note that your algorithm is given the value of W but not V.

Problem 12 (40 pts) *Facility Location Problem*: (Note that this question is slightly different from the one in the textbook.) There is a set U of n nodes, which you can think of as users. You are also given a set S of n possible sites, and you want to place servers at some of these sites to serve the users. For each site $s \in S$, there is a fee $f_s \geq 0$ for placing a server at that location. Every user $u \in U$ can be served from any server, but there is an associated cost d_{us} for serving user u from a server at site s. Given the sets U and S, and costs f and d, you need to select a subset $A \subseteq S$ at which to place servers, and assign each user u to a server. Give an $O(\log n)$-approximation algorithm to minimize the overall cost $\sum_{s \in A} f_s + \sum_{u \in U} \min_{s \in A} d_{us}$.

There are two ways to achieve an $O(\log n)$-approximation. Below we provide the framework, and you need to complete the details.

(a) You can mimic the greedy algorithm for the set cover problem. We use R to denote the set of users not yet served, and T is the set of sites already selected. Initially, $R = U$ and $T = \emptyset$. In every step, we try to serve one or more users in R with the lowest average cost. There are two cases: (1) We can serve a new user $u \in R$ with an existing server $s \in T$ such that d_{us} is minimized; or (2) we can place a new server at some site $s \notin T$ and a set of new users $V \subseteq R$ such that $(f_s + \sum_{u \in V} d_{us})/|V|$ is minimized. Clearly, the best (s, u) in case (1) can be found easily. Show that the best (s, V) in case (2) can also be found in polynomial time.

We will then compare the costs of the two cases. If case (1) is better, we simply remove u from R. If case (2) is better, we add s to T, and remove V from R. The process continues until $R = \emptyset$.

Show that this greedy algorithm yields an $O(\log n)$-approximation.

(b) You can also use LP relaxation and randomized rounding. More precisely, we use $x_s = 1$ to denote that we put a server at site s and $x_s = 0$ otherwise. We use y_{us} to denote that we serve u from site s.

Write down the integer linear program for this problem. Then we can solve the fractional version of the linear program. Show how we can use randomized rounding to find an $O(\log n)$-approximate optimal solution to the original problem with at least constant probability.

Chapter 12

Problem 2 (20 pts) Consider the following gradient ascent algorithm for finding a matching in a bipartite graph. Recall that a matching in a graph is a set of edges in the graph such that no two edges share a common endpoint.

As long as there is an edge whose endpoints are unmatched, add it to the current matching. When there is no longer such an edge, terminate with a locally optimal matching.

(a) Give an example of a bipartite graph G for which this gradient ascent algorithm does not return the maximum matching.

(b) Let M and M' be matchings in a bipartite graph G. Suppose that $|M'| > 2|M|$. Show that there is an edge $e' \in M'$ such that $M \cup \{e'\}$ is a matching in G.

(c) Use (b) to conclude that any locally optimal matching returned by the gradient ascent algorithm in a bipartite graph G is at least half as large as a maximum matching in G. (Note that finding the maximum matching is *not* an NP-hard problem.)

Problem 4 (20 pts) (Note that this problem is slightly different from the one in the textbook.) Consider the following local search algorithm for the Load Balancing Problem.

We start with an arbitrary assignment of jobs to machines, and then try to make local improvement by "swaps". Let $A(i)$ and $A(j)$ be the jobs assigned to machines M_i and M_j, respectively. To perform a swap on M_i and M_j, we choose subsets of jobs $B(i) \subseteq A(j)$ and $B(j) \subseteq A(j)$, and "swap" these jobs between the two machines, i.e., update $A(i)$ to be $A(i) \cup B(j) - B(i)$ and update $A(j)$ to be $A(j) \cup B(i) - B(j)$. It is allowed to have $B(i) = A(i)$, or to have $B(i) = \emptyset$; and so for $B(j)$.

Suppose the loads on M_i and M_j before the swap are T_i and T_j, respectively, and the loads after the swap are T_i' and T_j'. We say that the swap is improving if $\max(T_i', T_j') < \max(T_i, T_j)$—in other words, the larger of the two loads involved has strictly decreased. The local search algorithm will keep making improving swaps until reaching a local optimal, i.e., an assignment for which no improving swaps exist.

(a) Show that this local search algorithm will always terminate.

(b) Give an example on which the local search algorithm may make exponentially many steps before reaching a local optimum.

(c) Show that any locally optimal assignment has a makespan that is within a factor of 2 of the globally optimal makespan.