Introduction to Algorithms **Problem Set 9 (3 problems)**
CS 4820 Fall 2019 **Solutions**

(1) Knapsack Approximation. (10 points) The KNAPSACK problem discussed in class on Friday, November 22th is given by n items each with a value $v_i \geq 0$ and weight $w_i \geq 0$, and a weight limit W. The problem is to find a subset I of items maximizing the total value $\sum_{i \in I} v_i$ while obeying the weight limit $\sum_{i \in I} w_i \leq W$. You may assume that $w_i \leq W$ for all i, that is, each item individually fits in the knapsack.

This coding problem is asking you to design an efficient algorithm to approximately solve this problem. Your algorithm needs to find a solution that obeys the weight limit with total value of at least 1/2 of the maximum possible. I.e., you will receive full credits if your algorithm achieves an approximation ratio at most 2.

Solution. See the lecture notes `http://www.cs.cornell.edu/courses/cs4820/2016sp/lectures/approx-online.pdf` for a 2-approximation of Knapsack and an explanation of why some natural greedy algorithms do not work (i.e., common mistakes).

(2) Knapsack. (10 points) In class on Friday Nov 22nd we have studied the greedy approximation algorithm for the knapsack problem: the problem had n items with weight w_i and value v_i, and weight limit W. The goal is to select a subset $I \subset \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and maximizing the total value $\sum_{i \in I} v_i$. We saw an $O(n \log n)$ running time 2-approximation algorithm for this problem by taking the better of two different greedy solutions. Here we want to consider a different approximation: ensuring the total value of selected item is greater than the maximum value possible with weight limit W by allowing the total weight of the selected items to go somewhat beyond W.

(a) (5 points) Assume that no item is too big, that is $w_i \leq W/2$ for all i. Using this assumption, give a polynomial-time algorithm that finds a solution with the following guarantee: Suppose the maximum value possible with weight limit W is V^*, then the algorithm needs to output a subset I of items, such that $\sum_{i \in I} w_i \leq \frac{3}{2}W$ and $\sum_{i \in I} v_i \geq V^*$.

Solution: Sort by density v_i/w_i and add items till they overfill the knapsack. Assume items are indexed in decreasing density order. We have seen in class that if i is the first item that didn't fit, than $Opt \leq \sum_{j \leq i} v_j$. We output this solution, and notice that $\sum_{j \leq i} w_j \leq W + w_i \leq \frac{3}{2}W$.

To see why this solution has value at least the OPT of the original problem, recall that we showed in class that if the last item j can be cut and a x_j fraction fit to totally fill the knapsack with value $x_j v_j$, then the optimum is at most the resulting value. By fully taking the last item, we get more value.

(b) (5 points) Give an algorithm with the guarantee it part (a) even if the input has big items.

Solution: Only a single large item fits in the optimum solution. For each large item, add the large item to the backpack and run greedy from (a) to fill the remaining space in the backpack with the small items. Additionally, run greedy using only small items. Take the best of these at most $n + 1$ possible solutions.

To see why this has value at least the OPT, consider the optimum solution. This solution has at most one of the large items. Consider the case when we tried the same item as included in the optimum, say item i. Now the remaining knapsack has size $W - w_i$, and the OPT is the optimum for this smaller knapsack plus v_i for the first item. So the solution we get is at least the value of the solution for the reason given in part (a).

Alternate Solution: Use the algorithm from part (a). If the first item that doesn't fit is a small item, the solution and argument from part (a) applies. Suppose the first item that doesn't fit is a large item, item i_1. Put this item aside, and continue with the algorithm from part (a). Again if it ends with a small item, OK, else the last item is now i_2, put that one aside, etc. After this round, for each items put aside, try a version with this one item in the knapsack first and then put the small items in the usual order.

To see that this algorithm also works there are two cases. Case 1: if the optimum doesn't contain any of the items put aside. This case the original run will have the claimed guarantee due to part (a). If the optimum contains one of these items (and can only contain 1) than the argument from above works.

Alternate Attempt at the Solution that doesn't work: Use the algorithm from part (a). If the first item that doesn't fit is a small item, the solution and argument from part (a) applies. Suppose the first item that doesn't fit is a large item, item i. Try with starting with i and putting the rest of the items in with the usual order till then fit. Compare the two solutions obtained (by going till weight $1.5W$ and take the better one.

Unfortunately, this doesn't work. Here is a bad example. The total $W = 10$ and items in density order have (value, weight) pairs as follows $(9.2, 9), (9.1, 9), (10, 10)$, and say no small items. Now OPT is value 10, and the above solution gets is only 9.2.

(3) Vertex Cover on Hyper-graphs. (10 points) The HITTING SET problem is defined as follows: given a set V of nodes with weights w_i for $i \in V$, and a list of subsets $\mathcal{H}$ whose elements are subsets of V, and an integer k. A *hitting set* S is a subset $S \subset V$ such that $S \cap H \neq \emptyset$ for all sets $H \in \mathcal{H}$. the *minimum cost hitting set problem* is to find a hitting set S of minimum total weight $\sum_{i \in S} w_i$.

Note that when all sets $H \in \mathcal{H}$ have exactly 2 elements, then the sets in $\mathcal{H}$ form the edges in a graph G, and the hitting set problem is the vertex cover problem on the graph G. (This also shows that the HITTING SET problem is NP-hard).

(a) (5 points) For the special case of VERTEX COVER we have seen a 2-approximation algorithm in class on Monday, November 25. Show that the bound of 2 we gave comparing the true optimum and the optimum for the value of the linear program used in the approximation algorithm (on page 634) is best possible. Concretely, show that for any constant $\gamma < 2$ there is a vertex cover problem with weights w_i such that the linear programming optimum is more than a factor of γ smaller than the true optimum.

Solution: Consider a graph on n nodes with all edges included with $w_i = 1$ for all i. This graph needs $n - 1$ nodes for VERTEXCOVER, but the linear program used in class has a solution with $x_v = 1/2$ for all nodes v with a total value of $n/2$. Choose an n large enough such that $\frac{n-1}{n/2} > \gamma$.

Alternate Interpretations: Due to some confusion in some clarifications offered by the TAs, some of the solutions are interpreting the question is comparing the approximation algorithm to the true optimum. A bad example here is a cycle with n nodes, a solution to the linear program can put all $x_i = 0.5$. (In fact if the cycle is of odd length, this is the unique optimum solution to the linear program.) This solution has value $n/2$, with the true optimum almost the same: $\lceil n/2 \rceil$ nodes can give a vertex cover. However, the linear programming based approximation algorithm will take all nodes, and hence has value n.

Some people maybe just comparing the LP optimum to the Algorithm's solution. That can use either the complete graph or a cycle.

(b) (5 points) Now consider the general case of the HITTING SET, and assume that the maximum size of a set of $\mathcal{H}$ is $\max_{H \in \mathcal{H}} |H| \leq c$. Give a c-approximation algorithm for this case of the min-weight HITTING SET problem.

Solution: Consider the linear program minimizing $\sum_{i \in V} w_i x_i$ subject to the constraints $0 \leq x_i \leq 1$ for all i and $\sum_{i \in H} x_i \geq 1$ for all $H \in \mathcal{H}$. Then take the set $S = \{i : x_i \geq 1/c\}$.

Running time polynomial: solve the linear program, plus $O(n)$ for setting up and post processing.

Proving the approximation ratio: solving the linear program with integer x_i values gives the solution to the hitting set problem with $S = \{i : x_i = 1\}$. So the linear programming optimum, which we'll call OPT_{LP}, allowing fractional x values can only give a better optimum value, so we get $OPT_{LP} \leq OPT$. The solution proposed above satisfies

$$\sum_{i \in S} w_i \leq c \sum_i x_i w_i = c \cdot OPT_{LP} \leq c \cdot OPT$$

as required.

Note: An alternate option for a c approximation algorithm is to adapt the pricing method from 620-621.