CS222 Algorithm Design and Analysis Homework 8

Zhou Litao 518030910407 F1803016

December 3, 2020, Fall Semester

Exercise 1 Suppose you're acting as a consultant for the Port Authority of a small Pacific Rim nation. They're currently doing a multi-billion-dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Here's a basic sort of problem they face. A ship arrives, with n containers of weight w_1, w_2, . . . , w_n. Standing on the dock is a set of trucks, each of which can hold K units of weight. (You can assume that K and each w_i is an integer.) You can stack multiple containers in each truck, subject to the weight restriction of K; the goal is to minimize the number of trucks that are needed in order to carry all the containers. This problem is NP-complete (you don't have to prove this).

A greedy algorithm you might use for this is the following. Start with an empty truck, and begin piling containers 1, 2, 3, ... into it until you get to a container that would overflow the weight limit. Now declare this truck "loaded" and send it off; then continue the process with a fresh truck. This algorithm, by considering trucks one at a time, may not achieve the most efficient way to pack the full set of containers into an available collection of trucks.

1. Give an example of a set of weights, and a value of K, where this algorithm does not use the minimum possible number of trucks.
2. Show, however, that the number of trucks used by this algorithm is within a factor of 2 of the minimum possible number, for any set of weights and any value of K.

Solution.

1. Let $K = 5$, and there are three items with weight $3, 4, 2$. The algorithm will assign three trucks since 3 and 4 are both greater than half of K. However, 2 and 3 can be in a same truck. The optimal solution should be two.
2. Let $W = \sum_i w_i$ be the total weight of goods. Assume the optimal solution use L^* trucks and our greedy solution use L trucks.

   For every L truck assigned, we group them two by two, yielding $\lceil L/2 \rceil$ groups. Based on the algorithm, every group should have a weight $\in (W/K, 2W/K]$. It follows that the optimal solution should have a solution with no less than $\lceil L/2 \rceil$ trucks. i.e. $L^* \geq \lceil L/2 \rceil \geq L/2$. The greedy algorithm is a 2-approximation.

□

Exercise 2 Consider an optimization version of the Hitting Set Problem defined as follows. We are given a set $A = \{a_1, ..., a_n\}$ and a collection $B_1, B_2, ..., B_m$ of subsets of A. Also, each element $a_i \in A$ has a weight $w_i \geqslant 0$. The problem is to find a hitting set $H \subseteq A$ such that the total weight of the elements in H, that is $\sum_{a_i \in H} W_i$, is as small as possible.(We say that H is a hitting set if $H \cap B_i$ is not empty for each i.) Let $b = max_i |B_i|$ denote the maximum size of any of the sets $B_1, B_2, .., , B_m$. Give a polynomial-time approximation algorithm for this problem that finds a hitting set whose total weight is at most b times the minimum possible.

Solution. We formulate the problem as a integer linear programming problem (which is NP complete), and then solve its linear form. We show that the rounded linear solution is a b-approximation to the problem. The linear programming problem is formulated as follows, which can be solved in polynomial time.

$$\begin{array}{ll} \min_{\mathbf{x}} & \sum_i w_i x_i \\ \text{s.t.} & 0 \le x_i \le 1, \forall i \\ & \sum_{i \in B_j} x_i \ge 1, \forall j \end{array} \tag{1}$$

As for the rounding strategy, we take all x_i such that $x_i \ge \frac{1}{b}$ as a solution (i.e. $x'_i = 1$) and others to be not selected.

The rounded solution is a solution to the problem since constraints is satisfied by the fact that the rounding results will be either 1 or 0, and that there will be at least one element in every B_j set to be selected otherwise the original linear programming will be infeasible.

Next we show that the rounded solution is a b-approximation. let x'_i denote the rounded solution set, $\hat{x}_i$ denote the linear programming solution and x_i denote the optimal solution set. S is the solution set.

$$\sum_i w_i x'_i = \sum_{i \in S} w_i \cdot 1 \le \sum_{i \in S} w_i b \hat{x}_i \le b \sum_i w_i \hat{x}_i \tag{2}$$

The above equation indicates that the rounded solution is a b-approximation to the linear programming solution. Note that the optimal solution is also a feasible solution to the linear programming problem, thus we have that

$$\sum_i w_i \hat{x}_i \le \sum_i w_i x_i \tag{3}$$

It follows that

$$\sum_i w_i x'_i \le b \sum_i w_i x_i \tag{4}$$

□

Exercise 3 Suppose you're consulting for a biotech company that runs experiments on two expensive high-throughput assay machines, each identical, which we'll label M_1 and M_2. Each day they have a number of jobs that they need to do, and each job has to be assigned to one of the two machines. The problem they need help on is how to assign the jobs to machines to keep the loads balanced each day. The problem is stated as follows.

There are n jobs, and each job j has a required processing time t_j. They need to partition the jobs into two groups A and B, where set A is assigned to M_1 and set B to M_2. The time needed to process all of the jobs on the two machines is $T_1 = \sum_{j \in A} t_j$ and $T_2 = \sum_{j \in B} t_j$. The problem is to have the two machines work roughly for the same amounts of time—that is, to minimize $|T1 - T2|$.

A previous consultant showed that the problem is NP-hard (by a reduction from Subset Sum). Now they are looking for a good local search algorithm. They propose the following. Start by assigning jobs to the two machines arbitrarily (say jobs 1, ... , $n/2$ to M_1, the rest to M_2). The local moves are to move a single job from one machine to the other, and we only move jobs if the move decreases the absolute difference in the processing times. You are hired to answer some basic questions about the performance of this algorithm.

1. The first question is: How good is the solution obtained? Assume that there is no single job that dominates all the processing time—that is, that $t_j \leqslant \frac{1}{2} \sum_{i=1}^{n} t_i$ for all jobs j. Prove that for every locally optimal solution, the times the two machines operate are roughly balanced: $\frac{1}{2} T_1 \leqslant T_2 \leqslant 2T_1$.

2. Next you worry about the running time of the algorithm: How often will jobs be moved back and forth between the two machines? You propose the following small modification in the algorithm. If, in a local move, many different jobs can move from one machine to the other, then the algorithm should always move the job j with maximum t_j. Prove that, under this variant, each job will move at most once. (Hence the local search terminates in at most n moves.)

3. Finally, they wonder if they should work on better algorithms. Give an example in which the local search algorithm above will not lead to an optimal solution.

Solution.

1. W.l.o.g, we assume in the local optimal solution, the solution set is S_1, S_2 with total time $T_1 > T_2$. Since the solution can't be more optimal, we have that $\forall t_j \in S_1, T_1 - T_2 \leq (T_2 + t_j) - (T_1 - t_j)$. i.e. $T_1 \leq T_2 + t_j$.

   With the assumption that no single job that dominates all the processing time, since $T_1 \neq T_2$, there are at least three jobs in total, and T_1 will take at least two jobs running so that it can achieve a higher total time. Let t_j be the smaller time running on M_1, we have that $t_j \leq \frac{1}{2}T_1$.

   Combining the two formulas above we have that $T_1 \leq 2T_2$. Similarly for $T_1 < T_2$, by applying the similar approach we have that $T_2 \leq 2T_1$

2. Note that there are two conditions that a job t_j can be moved, namely $t_j < |T_1 - T_2|$ and t_j is the job with the maximal length that satisfy this condition in the jobset with a greater total time. W.l.o.g, assume at a point $T_1 > T_2$, then a job $t_j \in T_1$ is selected (if exists) and moved to T_2.

   Note that as our algorithm proceed, the optimization goal $|T_1 - T_2|$ will be strictly decreasing. There will be two cases for the result of moving t_j. If $t_j < \frac{T_1 - T_2}{2}$, the time of M_1 will still be greater than M_2, thus another job in M_1 will be moved to M_2, with a smaller time value since we are always choosing the largest.

   If $t_j > \frac{T_1 - T_2}{2}$, then after the movement, the next job should be selected from M_2. Consider at this point, whether the previous jobs that consecutively moving from M_1 to M_2 will be selected again. Note that the new $T_2' - T_1' < t_j$ (otherwise the move is invalid). It follows that in the following steps, any jobs with time value greater or equal to t_j will not be selected. Thus all jobs that previously are moved from M_1 to M_2 will not be selected. The results follows directly.

3. Suppose that there are jobs with time $3, 3$ assigned to M_1 and job with time $4, 4$ assigned to M_2. Now $\Delta = 2$, but no task can be moved from one to another. However the optimal solution should be $3, 4$ assigned to both machines with $\Delta = 0$

$\square$