CS222 Homework 8

Algorithm Analysis & Deadline: 2020-12-05 Saturday 24:00

Exercises for Algorithm Design and Analysis by Li Jiang, 2020 Autumn Semester

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 $w_1, w_2, \ldots, w_n$. 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.

   (a) 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.

   (b) 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.

(a) $n = 3, (w_1, w_2, w_3) = (3, 5, 2), K = 5$. This algorithm use 3 trucks, however the minimum possible number of trucks is 2.

(b) Denote the solution of this algorithm as $f(w_{\leq n}, K)$, and denote the optimal solution as $f^*(w_{\leq n}, K)$.

We prove $f(w_{\leq n}, K) \leq 2f^*(w_{\leq n}, K)$ by induction:

- **Base Step**: $f(w_1, K) = 1 \leq 2^*(w_1, K) = 2$.
- **Inductive Step**: We have $f(w_{\leq n}, K) \leq 2f^*(w_{\leq n}, K)$, and need to show $f(w_{\leq n+1}, K) \leq 2f^*(w_{\leq n+1}, K)$.

  We claim that $f(w_{\leq n+1}, K) = f(w_{\leq n}, K) \vee f(w_{\leq n}, K) < 2f^*(w_{\leq n}, K) \vee f^*(w_{\leq n}, K) < f^*(w_{\leq n+1}, K)$.

  Proof:

  Otherwise, $f(w_{\leq n+1}, K) = f(w_{\leq n}, K)+1 \wedge f(w_{\leq n}, K) = 2f^*(w_{\leq n}, K) \wedge f^*(w_{\leq n}, K) = f^*(w_{\leq n+1}, K)$.

  We say $f^*(w_{\leq n}, K) = t$. Then we have $\sum_{i=1}^{n+1} w_i \leq K \times f^*(w_{n+1}, K) = Kt$ (1). Assume in the greedy algorithm, the p_i-th container is the first one to be added to the i-th truck. Then $\forall 1 \leq i \leq 2t, \sum_{j=p_i}^{p_{i+1}} w_j > K$. Sum over i, we have $\sum_{i=1}^{2t} \sum_{j=p_i}^{p_i+1} w_j = \sum_{j=1}^{n+1} w_j + \sum_{i=1}^{2t} w_{p_{i+1}} > 2Kt$. However, by (1), $\sum_{i=1}^{2t} w_{p_{i+1}} \leq \sum_{j=1}^{n+1} w_j \leq Kt$. Contradictive! ■

  So from the claim above:

  * $f(w_{\leq n+1}, K) = f(w_{\leq n}, K) \Rightarrow f(w_{\leq n+1}, K) = f(w_{\leq n}, K) \leq 2f^*(w_{\leq n}, K) \leq 2f^*(w_{\leq n+1}, K)$.
  * $f(w_{\leq n}, K) < 2f^*(w_{\leq n}, K) \Rightarrow f(w_{\leq n+1}, K) \leq f(w_{\leq n}, K) + 1 \leq 2f^*(w_{\leq n}, K) \leq 2f^*(w_{\leq n+1}, K)$.
  * $f^*(w_{\leq n}, K) < f^*(w_{\leq n+1}, K) \Rightarrow f(w_{\leq n+1}, K) \leq f(w_{\leq n}, K)+1 \leq 2f^*(w_{\leq n}, K)+1 < 2f^*(w_{\leq n+1}, K)$.

So by induction we prove that given a fixed K, $\forall n \in \mathbb{N}_+, f(w_{\leq n}, K) \leq 2f^*(w_{\leq n}, K)$.

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.

This problem is equivalent to a $0-1$ LP problem:

$$\min \sum_i w_i \quad s.t. \begin{cases} w_i \in \{0,1\}, \ \forall 1 \leq i \leq n \\ \sum_{i:a_i \in B_k} w_i \geq 1, \ \forall 1 \leq k \leq m \end{cases}$$

An approximate real number LP problem which can be solve in polynomial time:

$$\min \sum_i x_i \quad s.t. \begin{cases} 0 \leq x_i \leq 1, \ \forall 1 \leq i \leq n \\ \sum_{i:a_i \in B_k} x_i \geq 1, \ \forall 1 \leq k \leq m \end{cases}$$

Find the optimal solution $x^* = (x_1^*, x_2^*, \cdots, x_n^*)$ of the real number LP problem above. Construct a feasible solution for the $0-1$ LP problem: $w_i \leftarrow 1\{x_i^* \geq \frac{1}{b}\}$. (Because the $\max_{i:a_i \in B_k} x_i^* \geq \frac{1}{|B_k|} \geq \frac{1}{b}$, the solution $w = (w_1, w_2, \cdots, w_n)$ is guaranteed feasible.) Assume the optimal solution for the $0-1$ LP problem is $w^* = (w_1^*, w_2^*, \cdots, w_n^*)$, then we have $\sum_i w_i \leq \sum_i bx_i^* \leq \sum_i bw_i^* = b\sum_i w_i^*$.

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.

   (a) 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: $\frac{1}{2}T_1 \leqslant T_2 \leqslant 2T_1$.

   (b) 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.)

(c) 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.

(a) Assume $T_2 > 2T_1$. There are at least two jobs in B, for $t_j \leq \frac{1}{2}\sum_{i=1}^{n} t_i$. Assume the job with the least processing time in B is job j, $t_j \leq \frac{T_2}{2} < T_2 - T_1$. After moving job j from B to A, $T_1 - T_2 < T_2' - T_1' = T_2 - T_1 - 2t_j < T_2 - T_1$. So $|T_2' - T_1'| < |T_2 - T_1|$, conflictive with the termination of the algorithm. So $T_2 \leq 2T_1$. And similarly, $T_1 \leq 2T_2$. Therefore, $\frac{1}{2}T_1 \leq T_2 \leq 2T_1$.

(b) Assume the job j being moved more than once. The first movement, say A $\rightarrow$ B, occured at time a, and the second movement, say B $\rightarrow$ A, occured at time b. We have $T_1^a - T_2^a > t_j$ and $T_2^b - T_1^b > t_j$. There must be a time c, $a \leq c < c+1 \leq b$, s.t. $T_1^c - T_2^c > 0$ and $T_2^{c+1} - T_1^{c+1} > 0$. The absolute difference is decreasing, so $T_1^c - T_2^c > T_2^{c+1} - T_1^{c+1} \geq T_2^b - T_1^b > t_j$. Assume the job moved from A to B at time c has a processing time t_c, $t_c < T_1^c - T_2^c \leq T_1^a - T_2^a$. Then $2t_c = (T_1^c - T_2^c) - (T_1^{c+1} - T_2^{c+1}) > 2t_j$, i.e. $T_1^a - T_2^a > t_c > t_j$. So at time a, we should move job c not job j. Contradictive! So each job will move at most once.

(c) This approximate algorithm executes in a certain fixed policy (select the largest one or select the smallest one or select according to an other policy). If the algorithm move the largest possible job each turn, I can Construct an example, where the jobs are given in such an order $t_1 = 5, t_2 = 4, t_3 = 3, t_4 = 2$. At the beginning, A $= (t_1 = 5, t_2 = 4)$ and B $= (t_3 = 3, t_4 = 2)$, which is also the final state of this algorithm. However the optimal solution is A $= (t_1 = 5, t_4 = 2)$, B $= (t_2 = 4, t_3 = 3)$. How about selecting a random valid job to move in each turn? It may be quite difficult to give a counter example.

Remark: You need to upload your .pdf file.