University of Toronto at Scarborough
CSCC73 - Algorithm Design and Analysis

Assignment #8 Approximation Algorithms

Due: April 11, 2023 at 11.59pm This exercise is worth 5% of your final grade.

Warning: Your electronic submission on MarkUs affirms that this exercise is your own work and no one else's, and is in accordance with the University of Toronto Code of Behaviour on Academic Matters, the Code of Student Conduct, and the guidelines for avoiding plagiarism in CSCC73. Late assignments will not be accepted. If you are working with a partner your partners' name must be listed on your assignment and you must sign up as a "group" on MarkUs. Recall you must not consult **any outside sources except your partner, textbook, TAs and instructor**.

1. A trucking company needs to load trucks with n boxes to send to a distribution center. We'll say that the size of a truck is 1 and that the i^{th} box has size $0 \leq s_i \leq 1$. It doesn't matter which boxes go into which trucks. Each truck can hold a subset of the boxes that does not exceed 1 unit. Consider the heuristic that takes each box in turn and places it into the first truck that can accommodate it. Let $S = \sum_{i=1}^{n} s_i$.

   (a) Determine the minimum number of trucks required by the optimal solution. Justify your answer.
   SOLUTION:

   The minimum number of trucks required is at least $\lceil S \rceil$. Why? we need enough trucks that can haul the load if they were packed perfectly.

   (b) Prove an upper bound on the number of trucks that this heuristic leaves less than half full.
   SOLUTION:

   There can be at most one truck less than half full. Suppose otherwise. Then there exist at least 2 trucks A and B that are at most half full. Consider the point when the 2nd truck, B, recieves it's first box of value ≤ 0.5. Since there exists truck A that has load $\leq .5$, the algorithm would have put this box into A and not started a new truck B.

   (c) Prove an upper bound on the number of trucks used by the heuristic.
   SOLUTIONS:

   The heuristic can never use more than $\lceil 2S \rceil$ trucks. To see this, recall that the total load of the boxes is S. We can have at most one truck that is at most half full.
   Suppose that T_H is the number of trucks used by the heuristic. Then $T_H - 1$ trucks have load > 0.5. Let t represent the load on the truck t with load ≤ 0.5. Since the sum of the truck loads equals the sum of the box weights, we can say,

$$S \geq \sum_{i=1}^{T_H - 1} \frac{1}{2} + t \Rightarrow S \geq \frac{1}{2}(T_H - 1) + t \Rightarrow 2S \geq T_H - 1 + 2t$$

   So $2S \geq T_H$ since $2t - 1 \leq 0$. Now since we are dealing with whole trucks, we need to take the ceiling of $2S$, or $\lceil 2S \rceil$.

(d) Determine α and prove an approximation ratio of α for this heuristic.

SOLUTIONS:

Let $\alpha = 2$. Then notice that the optimal solution is at least $\lceil S \rceil$ from part (a), so if T^* is the optimal, we have $T^* \geq \lceil S \rceil$. Therefore, $2S \leq \lceil 2S \rceil \leq 2T^*$. By part (c) we have that the heuristic is bounded by $\lceil 2S \rceil$ to give $T_H \leq \lceil 2S \rceil \leq 2T^*$.

(e) Give an efficient implementation for this algorithm and analyze it's running time.

SOLUTIONS:

The simplest implementation can simply use a linked list for the trucks. As a new truck is required, it is added as a new link at the end of the list. Each link will need a value to store the current weight on the truck. Notice that in the worst case, we need 1 truck for each box, this means that the i^{th} box will take i steps to load. So in the worst case, this is $\sum_{i=1}^{n} i$ where n is the number of boxes. This sum give us a complexity of $\mathcal{O}(n^2)$.

A nicer solution, which doesn't exactly represent the given algorithm, would be to use either a heap for storing the trucks with n items in the heap (where the priority is the current weight of the truck) or using a BST like data structure where the keys are the weights of the trucks. In either case, we can find the first truck (weight-wise) that can accept the current box in $\mathcal{O}(\log n)$ time. We can also update either data structure in $\mathcal{O}(\log n)$ time giving a final complexity of $\mathcal{O}(n \log n)$ time.

2. Consider the load balancing problem with just two machines M_1 and M_2. The goal is to keep the loads balanced. In particular, there are n jobs, and each job j has a required processing time t_j. The jobs need to be partitioned into two groups A and B, where set A is assigned to M_1 and B is assigned 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 $|T_1 - T_2|$. This problem is known to be $NP - hard$ which is why a good local search algorithm may be a good option.

Consider the following proposal. Start by assigning jobs to the two machines arbitrarily (say jobs $1, \ldots, n/2$ 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. We want 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, $t_j \leq \frac{1}{2} \sum_{i=1}^{n} t_i$ for all jobs j. Prove that for every locally optimal solution, the times the two machines operate are roughly balanced: $\frac{1}{2} T_1 \leq T_2 \leq 2T_1$.

SOLUTIONS:

We can assume by symmetry that the first machine M_1 has the higher load. we need to prove that $T_1 \leq 2T_2$. Notice that our assumption that no single job takes more than half of the processing time implies the machine with higher load must have at least two jobs. Let job j be the smallest job on machine M_1. Clearly, $T_1 \geq 2t_j$. The local search algorithm terminated, so moving job j from machine M_1 to machine M_2 does not decrease the difference between the processing times. This implies that $t_j \geq T_1 - T_2$. We get that $T_1 \leq t_j + T_2 \leq \frac{1}{2} T_1 + T_2$, and multiplying by two we get that $2T_1 \leq T_1 + 2T_2$ or $T_1 \leq 2T_2$.

(b) Next we consider the running time of the algorithm: *How often will jobs be moved back and forth between the two machines?* Consider the following modification to 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.

SOLUTIONS:

We observed above that if a job j satifies $t_j \geq |T_1 - T_2|$ it cannot move. Further the difference $|T_1 - T_2|$ decreases throughout the algorithm, so once this condition holds, job j will never move.

Now consider a job j, and we aim to prove that j wiol move at most once. Assume that job j starts on machine M_1. So, the first time it moves it will move from machine M_1 to machine M_2. We always move the largest job so at this point all remaining jobs on machine M_1 will have processing time at most t_j. Consider the sequence of consequtive moves all from machine M_1 to M_2 and let $t_{j'}$ be the last job that moves in this direction (possibly $j = j'$). At this time $T_2 \geq T_1$ and $T_2 - T_1 \leq t_{j'}$ as before moving job j', machine M_1 had more work. Now we have that $t_j \geq t_{j'} \geq |T_1 - T_2|$, so by the observation above job j will not move again.

(c) Show that the local search algorithm above, will not always lead to an optimal solution by giving an example.

SOLUTIONS:

As an example of a bad local optimum, let machine M_1 have two jobs with processing time 3 each, and machine M_2 has two jobs with processing time 2 each. Now the difference in loads is 2, no single job can move, but swapping a pair of jobs yields a solution with identical loads.

(d) Express this problem as a linear programming problem. In other words, define an objective function which needs to be minimized/maximized and constraints on the variables. You may find it helpful to define indicator variables x_i and y_i that indicate whether job i is on machine M_1 and M_2 respectively.

SOLUTION

Let the objective function be:

$$\sum_{i \in A} t_i - \sum_{j \in B} t_j = \sum_{i=1}^{n} t_i(x_i - y_i)$$

where x_i and y_i are indicator variables. If $x_i = 1$ then job i is assigned to machine A and similalry if $y_i = 1$ then job i is assigned to machine B. We require that the sum of x_i and y_i is 1 so that only one of x_i or y_i is 1. This ensures that a job can only be assigned to one machine.

Furthermore, we need to ensure that the sum is minimized and positive*. If we allow it to be negative, then the optimal solution would assign all jobs to machine B. Thus, our final set of constraints and objective function are:*

$$\textbf{Minimize} \sum_{i=1}^{n} t_i(x_i - y_i)$$

subject to,

$$\begin{aligned} \sum_{i=1}^{n} t_i(x_i - y_i) &\geq 0 \\ x_i + y_i &= 1, 1 \leq i \leq n \\ x_i, y_i &\in \{0, 1\} \end{aligned}$$