Oliver Tonnesen
V00885732

Assignment 4

CSC 425
December 4, 2020

1

Let $j_1, \ldots, j_n$ be a set of jobs ordered by increasing time t_i, and let L and L^* be the LPT and optimal makespans, respectively.

Assume without loss of generality that job j_n finishes last. If j_n starts at time t, then

$$L = t + t_n \leq \frac{1}{m} \sum_{i \neq n} t_i + t_n \leq L^* + t_n.$$

If $t_n \leq \frac{1}{3}L^*$, then $L \leq L^* + \frac{1}{3}L^* = \frac{4}{3}L^*$, and we're done. If $t_n > \frac{1}{3}L^*$, then we claim the LPT rule gives an optimal solution.

Assume for a contradiction that it does not, and that job j_r is the first job to finish after time L^*. All jobs take more than $\frac{1}{3}L^*$ to complete, so before job j_r, no machine can be scheduled more than two jobs. Let machines $M_1, \ldots, M_i$ all have one job, and machines $M_{i+1}, \ldots, M_m$ have two. Job j_r is not compatible (without going past time L^*) with any of the single jobs scheduled on machines M_1 to M_i, since the algorithm using the LPT rule would simply assign it there without surpassing time L^* if it could. Job j_r is also not compatible with any two of the jobs assigned to machines M_{i+1} to M_m, since any set of three jobs would surpass L^*. Thus L^* is not actually an attainable makespan. This contradicts the assumption that L^* is the optimal makespan, and thus in this case, the LPT rule gives an optimal solution.

Thus either $L \leq \frac{4}{3}L^*$ if $t_n \leq \frac{1}{3}L^*$, or $L = L^*$ if $t_n > \frac{1}{3}L^*$, and so LPT is indeed a $\frac{4}{3}$-approximation.

2

---

function MAX-SUM-APPROXIMATION$(a_1, \ldots, a_n, B)$

return the k largest elements whose sum is as large as possible and does not surpass B.

This algorithm takes $O(n \log n)$ since it must first sort the n numbers.

We claim that the sum of the returned elements is at least half that of an optimal solution.

Let S be the sum of the returned elements, and let S^* be the sum in an optimal solution. Assume for a contradiction that $S < \frac{1}{2}S^*$. Let a_r be the largest element not returned by the algorithm. Since it was not chosen, $S + a_r > B$. But $S < \frac{1}{2}S^* \leq \frac{1}{2}B$, so $a_r > \frac{1}{2}B$. Any number already chosen is at least as large as a_r which in turn is at least $\frac{1}{2}B$, so S must be at least $\frac{1}{2}B \geq \frac{1}{2}S^*$, a contradiction. Thus our algorithm returns a set summing to at least $\frac{1}{2}$ of an optimal solution, as desired.

function 3-MATCHING-APPROXIMATION(X, Y, Z, T)
return any maximal matching.

3

A maximal matching can trivially be constructed in linear time, and we claim that any maximal matching has size at least $\frac{1}{3}$ of a maximum matching.

Let M and M^* be maximal and maximum matchings, respectively, and assume for a contradiction that $|M| < \frac{1}{3}|M^*|$.

M contains $|M|$ triples, so it uses exactly $3|M|$ elements ($|M|$ from each of X, Y, and Z). This means that M^* cannot share more than $3|M|$ elements with M. In the extreme case, M^* would share all $3|M|$ of these elements with M, each contained in a separate triple in M^*. Since $|M^*| > 3|M|$, M^* must have an edge that shares no elements with any triple in M, contradicting M's maximality. Thus we can conclude that any maximal matching must have size at least $\frac{1}{3}$ of a maximum matching, as desired.

4

We define $x_i = \begin{cases} 1 & \text{if } S_i \text{ is chosen} \\ 0 & \text{otherwise} \end{cases}$.

Our goal is to minimize $\sum_{i=1}^{m} x_i$ subject to $\sum_{i:u \in S_i} x_i \geq 1$. We relax this integer linear program by allowing x_i to take values in $[0, 1]$ instead of $\{0, 1\}$.

Let x_i^* be the values given to each x_i in the linear program relaxation. If u is contained in f sets, then any set might be assigned a value as low as $\frac{1}{f}$, so to guarantee that each element in U is covered, we must round all x_i^* with values at least $\frac{1}{f}$ to 1, and those with values less than $\frac{1}{f}$ to 0.

Consider an arbitrary element $u \in U$. In the extreme case, an optimal solution uses exactly one of the f (or fewer) sets containing u. In this case, our linear program relaxed approximation algorithm uses all f sets, giving an f-approximation, as desired.