CSC373— Algorithm Design, Analysis, and Complexity — Spring 2018

Solutions for Tutorial Exercise 11: Approximation Algorithms

1. **Q5, Chp 11, Kleinberg and Tardos.** The first algorithm presented in the lecture notes for Load Balancing is an on-line algorithm, that is, the jobs can be processed as soon as they arrive. We refer to it as the **FirstArrival** algorithm. While the LPT algorithm obtains a better approximation ratio, it does not have this on-line property. Here we show that the **FirstArrival** algorithm can have an approximation ratio that is less than the worst case of 2 if the mix of jobs that it is asked to schedule is somehow restricted.

   For example, suppose that you have 10 machines, and need to schedule N jobs, where the n^{th} such job takes time t_n. In addition, you know the total time required for all the jobs is $T = \sum_{n=1}^{N} t_n = 3000$, and the time required for each individual job satisfies $1 \leq t_n \leq 50$. Show that the approximation ratio of FirstArrival for this mix of jobs is no worse than 7/6.

   Solution for Q1: In the lecture notes, we showed that the optimal makespan, L^*, satisfies

$$L^* \geq \frac{1}{m}\sum_{n=1}^{N} t_n = \frac{1}{m}T. \tag{1}$$

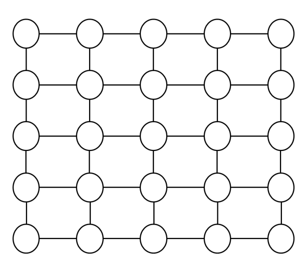
   Moreover, we showed the makespan $L(s)$ provided by the FirstArrival algorithm is the runtime for the last machine to finish, say L_i, which satisfies (slide 9):

$$(L_i - t_j) \leq \frac{1}{m}\sum_{n=1}^{N} t_n = \frac{1}{m}T. \tag{2}$$

   Therefore

$$\begin{aligned}
\frac{L(s)}{L^*} &= \frac{(L_i - t_j) + t_j}{L^*}, \\
&\leq \frac{(L_i - t_j) + t_j}{(T/m)}, \quad \text{by (1)}, \\
&\leq \frac{(T/m) + t_j}{(T/m)}, \quad \text{by (2)}, \\
&\leq \frac{(T/m) + \max(t_i)}{(T/m)}, \\
&\leq \frac{300 + 50}{300} = 7/6. \quad \text{since } \tfrac{1}{m}T = 300 \text{ and } t_i \leq 50.
\end{aligned}$$

2. **Q10, Chp 11, Kleinberg and Tardos.** Suppose you are given a weighted graph $G = (V, E, w)$ where G has the form of an $n \times n$ grid graph (see figure below). Assume the weights $w(v)$ are non-negative integers.



   Prof. Jot proposes the following greedy algorithm for obtaining an approximate solution to the maximally weighted independent set problem for this type of graph:

[S] = wIndSet(V, E, w)
Initialize $F \leftarrow (V, E)$ and $S \leftarrow \{\ \}$.
While the graph F is not empty:
    Find a vertex u in F with the largest weight $w(u)$.
    $S \leftarrow S \cup \{u\}$
    Update F by deleting the vertex u and all its neighbouring vertices v (i.e., all vertices v with an edge (u, v) still in F), and delete all the edges ending at any of these deleted vertices.
End while
return S

(a) Write a loop invariant for the above code that is useful for proving that the set S returned by `wIndSet` is an independent set for the graph G.

(b) Prove the loop invariant in part (a), and that the returned set S is an independent set for the graph G.

(c) Show that $w(S) = \sum_{v \in S} w(v)$ is at least $(1/4)w(S^*)$, where S^* is an independent set of G with the maximum possible weight $w(S^*)$.

Solution for Q2:

Soln Q2a Add the following loop invariant to the end of the loop in the above algorithm:

> **LI:** S is an independent set of G, and the updated graph F does not contain any vertex $u \in S$ nor any neighbour vertex (with respect to G) of u.

Soln Q2b This loop invariant can be proved by induction. The key is that whenever a vertex u is added to S, all remaining neighbours of u are deleted from F.

The algorithm removes at least one vertex from F each step, so it must terminate.

Upon termination, the algorithm returns S, and the LI guarantees that this is an independent set of G. We skip the details.

Soln Q2c Remember that to be a ρ-approximation we need to show the algorithm is poly-time. If, say, a heap is used to store the weights $w(v)$ (paired with v), then this algorithm runs in $O(|E| + |V| \log |V|)$ time.

To show the algorithm achieves an approximation ratio of 4, let S^* be a minimum weight independent set, and suppose S is the set produced by this algorithm. We build a "association" mapping, say $A(v)$, which maps each element $v \in S^*$ to one of its neighbours $u \in S$ as follows.

Let $v \in S^*$. If $v \in S$, define $A(v) = v$. Otherwise, $v \notin S$ and this means that, at some point, the algorithm above must have eliminated v from F when it selected one of v's neighbours, say u, to add to S. Given this $u \in S$, we define $A(v) = u$. Moreover, for the algorithm to choose u over v, it must be the case that $w(u) \geq w(v)$. Note that, in both of the above cases we have $A(v) \in S$ and the $w(A(v)) \geq w(v)$.

For each $u \in S$ let $c(u) = |\{v \mid v \in S^* \text{ and } A(v) = u\}$. That is, $c(u)$ is the number of different vertices $v \in S^*$ that are associated with u. Since S^* is an independent set, it follows that $0 \leq c(u) \leq$ degree$(u) \leq 4$ (see the above figure).

We now have

$$W^* \equiv \sum_{v \in S^*} w(v) \leq \sum_{v \in S^*} w(A(v)), \tag{3}$$

$$= \sum_{u \in S} c(u)w(u), \quad \text{by the definition of } c(u) \ , \tag{4}$$

$$\leq \sum_{u \in S} 4w(u) = 4w(S) = 4W. \tag{5}$$

Therefore, $W^*/W \leq 4$ and the above maximization algorithm is a ρ-approximation with $\rho = 4$.

3. **Modified Q3 Chp 11 Kleiberg and Tardos.** Suppose you are given a list of N integers $L = [a_1, a_2, \ldots, a_N]$, and a positive integer C. The problem is to find a subset $S \subseteq \{1, 2, \ldots, N\}$ such that

$$T(S) = \sum_{i \in S} a_i \leq C, \tag{6}$$

and $T(S)$ is as large as possible.

(a) Prof. Jot proposes the following greedy algorithm for obtaining an approximate solution to this maximization problem:

$[S]$ = maxBoundedSetSum($[a_1, \ldots, a_N]$, C)
Initialize $S \leftarrow \{\ \}$, $T = 0$
For $i = 1, 2, \ldots, N$:
    If $T + a_i \leq C$:
        $S \leftarrow S \cup \{i\}$
        $T \leftarrow T + a_i$
End for
return S

Show that Prof. Jot's algorithm is not a ρ-appoximation algorithm for any fixed value ρ. (Use the convention that $\rho > 1$.)

(b) Describe a 2-approximation algorithm for this maximization problem that runs in $O(N \log(N))$ time.

Solution for Q3:

Soln Q3a Let $\rho > 1$ be a given positive integer. Consider the input set $L = [1, \rho + 1]$, with the upper bound $C = \rho + 1$. Then the Prof. Jot's algorithm returns $S = \{1\}$, for which $T(S) = 1$, while the optimum solution is $S^* = \{2\}$, for which $T(S^*) = \rho + 1$. Therefore

$$\frac{T(S^*)}{T(S)} = \frac{\rho + 1}{1} > \rho, \tag{7}$$

so the algorithm is not a ρ-approximation for this value of ρ. Finally, since ρ was an arbitrary choice bigger than one, this is true for any $\rho > 1$. Therefore this algorithm is not a ρ-approximation.

Soln Q3b Consider the slightly modified algorithm (next page):

[S] = maxBoundedSetSum([$a_1, \ldots, a_N$], C)
Initialize $S \leftarrow \{\ \}$, $T = 0$
For $i = 1, 2, \ldots, N$:
    If $a_i \leq C$:
        If $T + a_i \leq C$:
            $S \leftarrow S \cup \{i\}$
            $T \leftarrow T + a_i$
        Else:
            If $T < C/2$:
                $S \leftarrow \{i\}$
            break
End for
return S

The algorithm runs in $O(N)$ time.

For the analysis, it is useful to first consider any item for which $a_i > C$. Such items cannot appear in any solution, and are simply discarded by the algorithm above. In the remainder of this proof we can therefore assume, without loss of generality, that $a_i \leq C$ for all i.

Given that $a_i \leq C$ for all i, there are now two general cases: 1) $\sum_{i=1}^{N} a_i = A \leq C$; and 2) $\sum_{i=1}^{N} a_i = A > C$. In the first case the alorithm above produces the optimum solution. In the second case, the algorithm above produces a set S such that $T(S) \geq C/2$. But note that, for any optimal solution S^*, it follows that $T(S^*) \leq C$. Therefore $T(S) \geq \frac{1}{2}T(S^*)$, and so the algorithm is a 2-approximation.