The Lazy Bureaucrat Problem with common arrivals and deadlines: approximation and mechanism design

Laurent Gourvès\textsuperscript{1,2}, Jérôme Monnot\textsuperscript{1,2}, and Aris T. Pagourtzis\textsuperscript{3}

\textsuperscript{1}. CNRS UMR 7243
\textsuperscript{2}. PSL Université Paris Dauphine
Place du Maréchal de Lattre de Tassigny
75775 Paris Cedex 16, France
\textsuperscript{3}. National Technical University of Athens (NTUA)
School of Electrical and Computer Engineering
Heroon Politechniou 9, GR-15780 Zographou, Greece
laurent.gourves@dauphine.fr, jerome.monnot@dauphine.fr, pagour@cs.ntua.gr

Abstract. We study the Lazy Bureaucrat scheduling problem (Arkin, Bender, Mitchell and Skiena [1]) in the case of common arrivals and deadlines. In this case the goal is to select a subset of given jobs in such a way that the total processing time is minimized and no other job can fit into the schedule. Our contribution comprises a linear time 4/3-approximation algorithm and an FPTAS, which respectively improve on a linear time 2-approximation algorithm and a PTAS given for the more general case of common deadlines [2, 3]. We then consider a selfish perspective, in which jobs are submitted by players who may falsely report larger processing times, and show a tight upper bound of 2 on the approximation ratio of strategyproof mechanisms, even randomized ones. We conclude by introducing a maximization version of the problem and a dedicated greedy algorithm.

1 Introduction

The goal of a lazy bureaucrat is to work as little time as possible in a certain day, having a good excuse for that. Such an excuse could be the fact that non accomplished tasks are too long to fit in his daily working hours. For this reason, given a set of tasks to execute in a single day he might try to choose a subset of jobs in such a way that no other task can fit into his working hours and the total duration of selected tasks is minimized. This scenario gives rise to a scheduling problem, introduced in a more general form by Arkin, Bender, Mitchell and Skiena [1]. The problem may find additional applications, for example the head of a department may want to prioritize funding requests of professors in such

\textsuperscript{*} This work is supported by French National Agency (ANR), project COCA ANR-09-JCJC-0066-01, and by the project ALGONOW of the research funding program THALIS, co-financed by the European Social Fund-ESF and Greek national funds.
a way that a minimum amount of money is spent on chosen requests, while no other request can be covered by the remaining budget.

Given a single processor and a set of jobs \([1..n]\), the unconventional objective of the lazy bureaucrat scheduling problem is to use the processor as inefficiently as possible, under a busy requirement defined below. Every job \(i\) has a processing time \(p_i\), an arrival time \(a_i\) and a hard deadline \(d_i\). These numbers are positive and satisfy \(a_i + p_i \leq d_i\). It is assumed that at least one job arrives at time 0 and the maximum deadline is denoted by \(B\). A job \(i\) is said executable at time \(t\) iff \(a_i \leq t \leq d_i - p_i\), meaning that \(i\) has arrived and it is not too late to execute it before its deadline. The busy requirement imposes that the processor cannot stay idle if an executable job exists.

The following two classical objective functions, to be minimized, are considered: (i) the completion time of the last executed job (makespan), (ii) sum of executed jobs’ processing times (time-spent). These two functions coincide when all jobs arrive at the same time because no feasible solution contains a gap [1]. Given a weight for every job, a third objective function consists in minimizing the weighted sum of completed jobs (weighted-sum).

The model can be declined in a preemptive or non-preemptive version: once a job is begun, it is either possible to interrupt it and resume it later, or the job must be completed without interruption. For the sake of conciseness, and because our contribution is not directly related to them, we deliberately skip the preemptive case and also the weighted-sum objective function. For these cases, the interested reader may consult [1, 4].

The lazy bureaucrat scheduling problem was shown strongly NP-complete and not approximable within any fixed constant for both makespan and time-spent [1]. Pseudo-polynomial algorithms based on dynamic programming can solve special cases of the problem, e.g. when all arrival times are the same [1]. The special case of unit length jobs for time-spent is solvable in polynomial time by the latest deadline first scheduling policy [1].

The common-deadline case, where the deadlines of the jobs are all equal to \(B\), has been introduced by Esfahbod, Ghodsi and Sharifi [2]. The problem remains NP-hard and the shortest job first constitutes a tight 2-approximation algorithm for both the makespan [2] and the time-spent objective functions [3]. For both makespan and time-spent, a pseudo-polynomial algorithm can solve the common-deadline case [2] and two polynomial approximation schemes (PTAS) have been given by Gai and Zhang [3].

Recently, the case where all jobs have the same arrival time and the same deadline has been studied [3]. The makespan and the time-spent objective functions coincide in this case, and the problems have been shown weakly NP-hard [3]. This article focuses on this variant.

As pointed out in [1], the lazy bureaucrat scheduling problem is not the only classical combinatorial problem that has been studied with a “reversed” objective function. One can mention the maximum traveling salesman problem, the minimum maximal matching, and more closely related, the lazy packing and covering problems [5–8].
1.1 Problem definition and contribution

We study the lazy bureaucrat scheduling problem with common arrival times and common deadlines, but we just name it the Lazy Bureaucrat Problem for the sake of simplicity. The input is a set of positive integers \( P = \{p_1, \ldots, p_n\} \), and a bound \( B \in \mathbb{Z}_+ \). A feasible solution is a set \( I \subseteq \{1, \ldots, n\} \) such that \( s(I) = \sum_{i \in I} p_i \leq B \) and \( \forall j \notin I, p_j + s(I) > B \). The objective is to minimize \( s(I) \). We always make the assumption that the jobs are named in non-decreasing order of durations: \( p_1 \leq \cdots \leq p_n \).

As an example, we are given 3 jobs of duration 1, 2, and 6 hours respectively, and that the working time is 8 hours; the lazy bureaucrat would choose to do jobs 1 and 2.

The first part of our contribution comprises approximation algorithms for the Lazy Bureaucrat Problem. We start by analyzing Greedy, a simple 2-approximation algorithm that follows the shortest-job-first scheduling policy introduced in [2]; we show a refined bound which in the worst case equals 2. We next present Approx, a more involved 4/3-approximation, that also requires only linear time by careful implementation; we last derive an FPTAS for the problem by adapting techniques proposed in [9] for the Subset-Sum problem. To obtain our FPTAS we develop an exact enumeration algorithm for the problem. This FPTAS improves on the known PTAS for the common deadline case [3].

We next proceed to study the problem in the selfish setting, in which players may falsely report larger job durations in order to get selected by the algorithm. We show that the Greedy algorithm is strategyproof and that, essentially, no better ratio is possible by any strategyproof mechanism, even randomized.

We finally define and study a maximization variation of the Lazy Bureaucrat that we call Greedy Boss Problem. Feasible solutions are minimal schedules that exceed a given bound and the goal is to find the maximum among them. We propose a longest-job-first greedy approach that is a 1/2-approximation algorithm. Some proofs are omitted due to space limitation.

2 Approximation algorithms

2.1 A simple 2-approximation algorithm

Algorithm 1 is called Greedy. It is not new since it coincides with the shortest job first scheduling policy introduced in [2] for the common deadline case. Nevertheless, Greedy is used in the FPTAS given in Subsection 2.3. Moreover, having common arrival times allows for showing a refined bound, which in the worst case equals the known bound of 2 [2].

The jobs being sorted in non-decreasing order of their durations, we have \( p_j \geq p_{j+1} \) for all \( j \in [t+1..n] \). Hence \( s([1..t]) + p_j \geq s([1..t+1]) > B \). The solution returned by Greedy is then feasible. The time complexity of Greedy is clearly \( O(n) \).

Assume that the solution \( I_G = [1..t] \) returned by Greedy is not optimal. Consider any optimal solution \( I^* \) with \( s(I^*) < s(I_G) \); hence, it must be \( I^* \neq I_G \).
Algorithm 1 Greedy

Input: \( \{p_1, \ldots, p_n\}, B \)

1: if \( s([1..n]) \leq B \) then
2: \text{return} \([1..n] \)
3: else
4: Find the smallest \( t \) such that \( s([1..t+1]) > B \)
5: \text{return} \([1..t] \)
6: end if

and \( I_G \setminus I^* \neq \emptyset \). Let \( s = \min(I_G \setminus I^*) \). That is, \( s \) is the smallest index appearing in \( I_G \) and not appearing in \( I^* \). Define \( \tau = B/p_s \).

Lemma 1. Greedy achieves an approximation ratio of \( \frac{\tau}{\tau - 1} \leq 2 \).

Proof. Let \( OPT = s(I^*) \). Since \( I^* \) is maximal it must be \( OPT + p_s > B \geq \tau \cdot p_s \Rightarrow p_s < \frac{\tau}{\tau - 1} OPT \). Therefore:

\[
s(I_G) \leq B < OPT + p_s < \frac{\tau}{\tau - 1} OPT
\]

In addition, it must be \( p_s \leq B/2 \iff \tau \geq 2 \), since \( p_s > B/2 \) would imply that \( p_s \) is the last job chosen by Greedy; this in turn would lead to \( OPT \geq s(I_G) \), a contradiction. Therefore, \( \tau/(\tau - 1) \leq 2 \).

For the tightness consider an instance consisting of \( B \) jobs of length 1 and one job of length \( 1 + \varepsilon \); then \( s(I_G) = B, OPT = (B - 1) + \varepsilon, \tau = B \).

Remark. If there are several solutions of optimum value, \( \tau \) is maximized (and the approximation ratio minimized) by the one that minimizes \( s \).

2.2 A simple 4/3-approximation algorithm

Algorithm 2 is called Approx. It returns a solution \( I_{apx} \) which is feasible for the Lazy Bureaucrat Problem because, by construction, all solutions \( I_G, I_1 \) and \( I_1' \) are feasible. Indeed, for \( I_1 \) and \( I_1' \), we respectively have \( s([1..t-1]) + p_t + p_{t+1} \geq s([1..t+1]) > B \) and \( s([1..t]) + p_t \geq s([1..t+1]) > B \), where \( I_G = [1..t] \), and because the jobs are sorted in non-decreasing order, the feasibility follows.

A rough analysis of the time-complexity of Algorithm 2 gives \( O(n^2) \), but a careful analysis yields a \( O(n) \) time. Actually when we run Greedy at step 1, for every \( i \leq t \), we store \( st_i = B - s([1..i]) \) the “saved time” using items from 1 to \( i \). For every \( i > t \), let \( f_2(i) = \max\{j \leq t : st_j \geq p_i + p_{i+1}\} \) and \( f_1(i) = \max\{j \leq t : st_j \geq p_i\} \). Clearly, the solutions produced at steps 3–5 and 8–10 respectively are given by \( [1..f_2(i)] \cup \{i, i + 1\} \) and \( [1..f_1(i)] \cup \{i\} \). Since \( f_2 \) and \( f_1 \) are decreasing mappings (the jobs are sorted in non-decreasing order), \( f_2(i + 1) \) (resp., \( f_1(i + 1) \)) can be found from \( f_2(i) \) (resp., \( f_1(i) \)) by decreasing the index \( k \) one by one until \( st_k \leq p_{k+1} + p_{k+2} \) (resp., \( st_k \leq p_{k+1} \)). Hence, Steps 2 to 11 can be performed in time \( O(n) \). The overall time-complexity is \( O(n) \).

Let \( APX \) (resp., \( OPT \)) be the value of the solution returned by Approx (resp., optimum solution \( I^* \)). Let us give some lemmas useful in the following.
Algorithm 2 Approx

Input: \( p_1, \ldots, p_n \), \( B \)
1: \( I_G := \text{Greedy}(\{p_1, \ldots, p_n\}, B) \) \hspace{1cm} \text{Comment: } I_G = [1..t]
2: for \( i = t \) to \( n \) do
3: \hspace{1cm} if \( p_i + p_{i+1} \leq B \) then
4: \hspace{2cm} \( I_i := \{i, i + 1\} \cup \text{Greedy}(\{p_1, \ldots, p_{i-1}\}, B - p_i - p_{i+1}) \)
5: \hspace{1cm} end if
6: end for
7: for \( i = t + 1 \) to \( n \) do
8: \hspace{1cm} if \( p_i \leq B \) then
9: \hspace{2cm} \( I_i' := \{i\} \cup \text{Greedy}(\{p_1, \ldots, p_i\}, B - p_i) \)
10: \hspace{1cm} end if
11: end for
12: return \( I* \), the best solution among \( I_G \), \( I_i \) and \( I_i' \)

Lemma 2. Let \( P = \{p_1, \ldots, p_n\} \), and \( B \) be an instance of the Lazy Bureaucrat Problem. The following inequalities concerning \( I_G \) hold:

(i) \( s(I_G) < \text{OPT} + p_n \), where \( s = \min(I_G \setminus I^*) \).
(ii) If \( s(I_G) > \frac{4}{3} \text{OPT} \) and \( \text{APX} \neq \text{OPT} \), then \( I^* \setminus [1..s] = \{i_1^*, i_2^*\} \) with \( s < i_1^* < i_2^* \) and \( i_2^* > t \). In particular, \( I^* = [1..s - 1] \cup \{i_1^*, i_2^*\} \).
(iii) If \( s(I_G) > \frac{4}{3} \text{OPT} \) and \( \text{APX} \neq \text{OPT} \), then \( t = s + 2 \). In particular, \( I = [1..s + 2] \).

Lemma 3. If \( s(I_G) > \frac{4}{3} \text{OPT} \) and \( \text{APX} \neq \text{OPT} \), then
\[
\text{APX} \leq 2p_{i_2^*} + s([1..s - 1])
\] (1)

Lemma 4. If \( s(I_G) > \frac{4}{3} \text{OPT} \) and \( \text{APX} \neq \text{OPT} \), then
\[
\text{APX} \leq 2p_{i_1^*} + s([1..s])
\] (2)

We are ready to give the main result of this section.

Theorem 1. Algorithm Approx is a linear-time \( 4/3 \)-approximation for the Lazy Bureaucrat Problem.

Proof. If \( s(I_G) \leq \frac{4}{3} \text{OPT} \) or \( \text{APX} = \text{OPT} \), we are done because \( \text{APX} \leq s(I_G) \).
So, assume \( s(I_G) > \frac{4}{3} \text{OPT} \) and \( \text{APX} \neq \text{OPT} \). Adding inequalities (i) of Lemma 2, (1) and (2) we obtain \( 3 \text{APX} < 3 \text{OPT} + 2p_n \). Now, using (ii) of Lemma 2, we know that \( 2p_n \leq \text{OPT} \). Hence, \( 3 \text{APX} < 4 \text{OPT} \). For the tightness, consider \( n = 7, \quad \varepsilon \in (0, 1), \quad p_1 = p_2 = p_3 = p_4 = 1, \quad p_5 = 1 + \varepsilon, \quad p_6 = 2, \quad p_7 = 2 + 2\varepsilon \) and \( B = 4 + 2\varepsilon \). We have \( \text{APX} = 4 \) given by \( I_G = \{1, 2, 3, 4\} \) while \( \text{OPT} = 3 + 3\varepsilon \) given by \( I^* = \{5, 7\} \). When \( \varepsilon \) tends to 0, we obtain \( \text{APX} = \frac{4}{3} \text{OPT} \). \( \square \)

In fact Approx consists of testing all solutions composed of a set of at most \( k = 2 \) consecutive jobs, completed by Greedy. Note that the previous instance shows that extending Approx to \( k = 3 \) yields the same approximation ratio of \( 4/3 \).
2.3 A fully polynomial approximation scheme

The following algorithms are based on the one proposed in [9, pp.1043-1049] for the SUBSET-SUM problem.

Let us first execute Greedy and let \( u = \min ([1..n] \setminus I_G) \); this is the first job that is rejected by Greedy. We are searching for solutions of value at most \( s(I_G) \).

Therefore, by the feasibility constraint, such solutions must contain \( T = \{ i \in [1..n] : p_i \leq B - s(I_G) \} \). Let \( \ell = 1 + \max T \) if \( T \neq \emptyset \), \( \ell = 1 \) otherwise; \( \ell \) is a lower bound on the index of the smallest rejected job in any solution of value at most \( s(I_G) \). Thus, \( \ell \leq u \).

We present an approximation algorithm which is based on an exact enumeration. The idea of the exact enumeration is: assuming that we know the index, say \( t^* \), of the smallest rejected job in an optimal solution, one may compute that optimal solution (or one of the same value) by taking into the solution all elements with index smaller than \( t^* \), excluding element \( p_{t^*} \), and considering all possible sums of elements with index \( \geq t^* \). The algorithm finds the smallest solution with value in \([B - p_{t^*}..B]\) (feasible solutions); this (if it exists) is the best solution with minimum excluded element \( p_{t^*} \). The algorithm tries all possible values of \( t^* \) and outputs the smallest among these solutions; note that it suffices to try only \( t^* \in [\ell..u] \).

In the following, \( \text{VAL} \) is the value of the best solution known so far and \( \text{ind} \) is the index of the smallest rejected job in that solution. \( L_i \) is a list of integers, \( L_i + x \) denotes a list that contains all the elements of \( L_i \) increased by \( x \), and \( \text{MERGE-LISTS}(L, L') \) returns the sorted list that is the merge of its two sorted input lists \( L \) and \( L' \) with duplicate values removed (the time complexity of \( L_i + x \) and \( \text{MERGE-LISTS}(L, L') \) are respectively \( \mathcal{O}(|L|) \) and \( \mathcal{O}(|L| + |L'|) \)).

The exact enumeration algorithm, called Exact, is given in Algorithm 3. Exact uses traceback, a standard procedure that outputs the set of indices \( I \) such that \( \text{VAL} = s(I) \) (the details of traceback are omitted).

Given an error \( \delta > 0 \), we say that \( z \) can represent \( y \) if \( y \leq z \leq (1 + \delta)y \). The trimming of a list \( L = (y_1, \ldots, y_m) \) of increasing elements is done as described in Algorithm 5. The trimming algorithm returns a subset of the input such that a \((1 + \delta)\)-approximate delegate \( z \) for each discarded value \( y \) is kept and \( y \leq z \leq (1 + \delta)y \). The time complexity of \( \text{trim}(L, \epsilon) \) is \( \Theta(|L|) \).

Following [9], Algorithm 4 constitutes an approximate version of Exact. It suffices to remove the trimming procedure on line 7, or set \( \epsilon \) to 0, to get Exact.

Note that a list produced by Exact is denoted by \( L^*_{t,i} \) for some parameters \( t, i \) while its counterpart produced by FPTAS is denoted by \( L^{*}_{t,i} \).

Theorem 2. Algorithm 4 is a fully polynomial time approximation scheme for the Lazy Bureaucrat problem.

Proof. Let us assume that the optimal solution, say \( I^* \), has \( t^* \) as minimum excluded element index (i.e. \([1..t^* - 1] \subseteq I^*, t^* \notin I^* \)). As explained above, it must be \( \ell \leq t^* \leq u \). Let \( \text{OPT} = s(I^*) \). Clearly, Exact finds the optimum and \( \text{OPT} \) is the smallest element of the list \( L^*_{t^*\cdot n} \).
Algorithm 3 Exact

**Input:** \( \{p_1, \ldots, p_n\}, B, I_G, u, \ell \)

1: \( VAL^* \leftarrow s(I_G) \)
2: \( ind^* \leftarrow u \)
3: for \( t \leftarrow \ell \) to \( u \) do
4: \( L_{t,t}^* \leftarrow \left\langle \sum_{j=1}^{t-1} p_j \right\rangle \) \hspace{1cm} Comment: \( \sum_{j=1}^{t-1} p_j = 0 \) when \( t = \ell = 1 \)
5: for \( i \leftarrow t + 1 \) to \( n \) do
6: \( L_{t,i}^* \leftarrow \text{merge-lists}(L_{t,i-1}^*, L_{t,i-1}^* + p_i) \)
7: Remove from \( L_{t,i}^* \) every element that is greater than \( VAL^* \)
8: end for
9: Remove from \( L_{t,n}^* \) every element that is \( \leq B - p_t \)
10: if \( L_{t,n}^* \neq \emptyset \) then
11: let \( e \) be its smallest element
12: if \( e < VAL^* \) then
13: \( VAL^* \leftarrow e \)
14: \( ind^* \leftarrow t \)
15: end if
16: end if
17: end for
18: \( I^* \leftarrow \text{traceback}(VAL^*, ind^*) \)
19: return \( I^* \)

Algorithm 4 FPTAS

**Input:** \( \{p_1, \ldots, p_n\}, B, I_G, u, \ell, \epsilon \)

1: \( VAL \leftarrow s(I_G) \)
2: \( ind \leftarrow u \)
3: for \( t \leftarrow \ell \) to \( u \) do
4: \( L_{t,t} \leftarrow \left\langle \sum_{j=1}^{t-1} p_j \right\rangle \) \hspace{1cm} Comment: \( \sum_{j=1}^{t-1} p_j = 0 \) when \( t = \ell = 1 \)
5: for \( i \leftarrow t + 1 \) to \( n \) do
6: \( L_{t,i} \leftarrow \text{merge-lists}(L_{t,i-1}, L_{t,i-1} + p_i) \)
7: \( L_{t,n} \leftarrow \text{TRM}(L_{t,n}, \epsilon/2n) \)
8: Remove from \( L_{t,i} \) every element that is greater than \( VAL \)
9: end for
10: Remove from \( L_{t,n} \) every element that is \( \leq B - p_t \)
11: if \( L_{t,n} \neq \emptyset \) then
12: let \( e \) be its smallest element
13: if \( e < VAL \) then
14: \( VAL \leftarrow e \)
15: \( ind \leftarrow t \)
16: end if
17: end if
18: end for
19: \( I_\epsilon \leftarrow \text{traceback}(VAL, ind) \)
20: return \( I_\epsilon \)
Algorithm 5 TRIM

Input: $(L, \delta)$

1: $L' \leftarrow \langle y_m \rangle$
2: last $\leftarrow y_m$
3: for $i \leftarrow m - 1$ down to 1 do
4: if last $> (1 + \delta)y_i$ then
5: insert $y_i$ at the beginning of $L'$
6: last $\leftarrow y_i$
7: end if
8: end for
9: return $L'$

We use arguments similar to those in [9]. Namely, by simple induction, one can show that $r$ applications or TRIM with an error of $\delta$ implies that a discarded element $y$ and its delegate $z$ satisfy $y \leq z \leq (1 + \delta)r y$. Comparing the list $L_{t^*, i}$ produced at line 7 of FPTAS with the list $L_{t^*, i}$ produced at line 6 of Exact, for every $y \in L_{t^*, i}$ there exists a delegate $z \in L_{t^*, i}$ satisfying

$$y \leq z \leq (1 + \epsilon)^{i - t^*} y$$

where $i \in [t^* + 1..n]$. Since $OPT \in L_{t^*, n}$, there exists a delegate $DEL \in L_{t^*, n}$ at line 7 of FPTAS when $t = t^*$ and $i = n$. Use Inequality (3) and the fact that $(1 + \epsilon/n)^{n-t^*} \leq (1 + \epsilon/n)^n \leq 1 + \epsilon$ (see [9] for a proof) to get that

$$DEL \leq (1 + \epsilon/n)^{n-t^*} OPT \leq (1 + \epsilon)OPT$$

It only remains to argue that the delegate of an optimal solution is not removed from the lists (because of line 8 or line 10 of FPTAS), or if this happens, then we are left with another solution of value at most $(1 + \epsilon)OPT$. Since approximate solutions are always at least as large as the corresponding exact ones, delegates of feasible solutions are never removed for violating the laziness requirement (i.e. for being $\leq B - p_t$). Therefore, the only case in which a delegate of a feasible solution may be removed is if it exceeds $VAL$, the currently best approximate solution. So, let us assume that this happens. Namely, the delegate $DEL$ of $OPT$ exceeds $VAL$ at some loop of the algorithm, therefore it also exceeds the final solution computed by the algorithm. In this case the returned feasible solution $I_{t}$ satisfies $s(I_{t}) \leq VAL < DEL \leq (1 + \epsilon)OPT$. Hence, $I_{t}$ is a $(1 + \epsilon)$-approximation of $OPT$.

To conclude, one has to analyze the time complexity of FPTAS. A list $L_{t,i}$ produced by Exact contains at most $B + 1$ elements. Following similar argumentation as in [9], its trimmed counterpart $L_{t,i}$ produced by FPTAS contains at most $2 + \log_{1+\epsilon/2n} B \leq 2 + \frac{4n \ln B}{\epsilon}$ elements, which is polynomial in both $n$ and $1/\epsilon$. Since FPTAS produces $O(n^2)$ lists, i.e. $L_{t,i}$ for $1 \leq t < i \leq n$, it is polynomial in $n$ and $1/\epsilon$. 

$\square$
3 Mechanisms

We consider that the lazy worker’s job selection policy, called mechanism in what follows, is publicly known. Meanwhile the jobs are submitted to the lazy worker by a pool of players who can act strategically. In concrete terms, a player may realize that his job would be rejected by the worker’s mechanism in its true form but if the job is expanded with fictitious time-consuming subtasks then it would be executed.

For example, there are three jobs of processing times $p_1 = 3$, $p_2 = 5$ and $p_3 = 7$ respectively and a budget of 8. The worker should select the third job and reject the first two. But if $p_1$ is modified to last $p_1' \in [3 + \epsilon, 5 - \epsilon]$ instead of 3 then it would be selected in order to minimize the working duration.

By doing so the players can alleviate the worker’s labor but the worker, worried by the $p_1' - p_1$ hours spent doing dummy tasks (that he cannot identify), may prefer sacrificing optimality to guarantee that the players report true jobs.

3.1 Model and notations

We assume that every player owns a unique job so we identify players to jobs. The true duration of a job $i$ is $p_i$ and this information is private to player $i$. The job that player $i$ submits requires $b_i$ time units. We assume that a player cannot cut or compress his job so $b_i \geq p_i$ for all $i$.

The mechanism is denoted by $M$. Its input is $b$, the vector containing all submitted jobs’ durations, and the budget is $B$. The output $I = M(b)$ is a subset of $[1..n]$ such that $s(I) \leq B$ and $b_i > B - s(I)$ for all $i \notin I$.

We suppose that a player has utility $u_i(b)$ equal to $x$ if $i \in M(b)$ with probability $x$. If $M$ is deterministic then $u_i(b) \in \{0, 1\}$.

3.2 Strategyproofness

A mechanism is strategyproof if a player can never benefit from reporting a false (i.e. larger) duration, regardless of the strategies of the other players.

Observation 1 Greedy is strategyproof.

Proof. Greedy takes the jobs in non-decreasing order of bids and returns the first $j$ ones, for some $j \geq 1$. The $j$ first players have utility 1 while the others have utility 0. Then a player with utility 0 keeps having utility 0 if he increases his bid. In addition a player with utility 1 (the maximum utility) can only see his utility decrease when he increases his bid. □

Note that Greedy satisfies a stronger notion of strategyproofness in which, for every group of players reporting false (larger) durations, it is not possible that all members of the group do not lose and at least one member of the group benefits.

Next result indicates that Greedy, known to be 2-approximate (see Lemma 1), yields the best approximation ratio for the class of deterministic strategyproof mechanisms. A similar proof technique has already been used in [10].
Proposition 1. Every deterministic $\rho$-approximate strategyproof mechanism on $n$ jobs satisfies $\rho \geq \frac{2}{1+h(n)}$ for every function $h : \mathbb{N} \rightarrow (0, 1)$.

Proof. Let $\mathcal{M}$ be a mechanism which is deterministic, $\rho$-approximate and strategyproof.

Consider an instance with $n$ jobs ($n \geq 3$). Each job has duration 1 and the budget is 2. A feasible solution consists of two jobs. Since the mechanism is deterministic, we assume w.l.o.g. that job 1 is rejected.

Consider a second instance which is identical to the previous one except that job 1’s duration is $1 + h(n)$ where $h(n) \in (0, 1)$. The optimal solution is to take job 1 with total duration $1 + h(n)$ while taking two jobs $\{x, y\}$ such that $x \neq 1$ and $y \neq 1$ is still a feasible solution with total duration 2. If the mechanism returns $\{1\}$ for the second instance then in the first instance, job 1 can bid $1 + h(n)$ instead of 1 and benefit. By strategyproofness, a pair of jobs $\{x, y\}$ is returned for the second instance, leading to an approximation ratio of $\frac{2}{1+h(n)}$. $\square$

For instance, by taking $h(n) = \frac{1}{2n}$, we get that $\rho \geq 2 - \frac{2}{2n+1}$ for any instance on $n$ jobs. Next result shows that randomization offers only little hope for a mechanism with a better approximation ratio.

Proposition 2. Every randomized $\rho$-approximate strategyproof mechanism satisfies $\rho \geq 2 - \frac{2}{n} - \delta$ for every $\delta \in (0, 1]$.

Proof. Let $\mathcal{M}$ be a mechanism which is randomized, $\rho$-approximate and strategyproof.

Consider an instance with $n$ jobs $\{1, \ldots, n\}$. Each job has duration 1 and the budget is 2. A feasible solution consists of a couple of jobs. For at least one job, say $i$, its probability $\text{Pr}_i$ to belong to the solution returned by $\mathcal{M}$ is at most $\frac{2}{n}$. Consider a second instance which is identical to the previous one except that job $i$’s duration is $1 + \epsilon$ for some small positive $\epsilon \in (0, 1)$. The optimal solution is $\{i\}$ with total duration $1 + \epsilon$. We suppose that $\mathcal{M}$ returns $\{i\}$ with probability $q$. Any couple of jobs in $\{1, \ldots, n\} - \{i\}$ is feasible with total duration 2. The probabilities for these suboptimal solutions add up to $1 - q$. The expected approximation ratio is then $\frac{q(1+\epsilon)+2(1-q)}{1+\epsilon} = 2 - \frac{q(1-\epsilon)}{1+\epsilon} = 2 - \delta$ where $\delta = \frac{2\epsilon}{1+\epsilon}$.

If $\text{Pr}_i < q$ then $i$ can bid $1 + \epsilon$ instead of 1 and benefit, contradiction with the strategyproofness of $\mathcal{M}$. So $q \leq \text{Pr}_i \leq \frac{2}{n}$ and the expected approximation ratio of the second instance is $2 - \frac{2}{n} - \delta$. $\square$

4 Greedy Boss vs Lazy Bureaucrat

Let us now consider the following scenario: in order to cope with lazy bureaucrats, entrepreneurs have managed to pass a law that prohibits employees from refusing a job if they have nothing to do at a given time, even if the job is to be
finished after the end of the schedule. Unfortunately, some employers are particularly greedy, and try to make use of this law in order to maximize the working time of their employees: they want to assign them jobs that exceed the schedule as much as possible, yet removing a job from the set makes the schedule unfilled, hence the employee cannot refuse to execute any job without violating the new law.

The Greedy Boss Problem is defined as follows. The input is a set of positive integers $P = \{p_1, \ldots, p_n\}$ and a bound $B \in \mathbb{Z}_+$ such that $\sum_{i=1}^n p_i \geq B$. A feasible solution is a set $I \subseteq \{1, \ldots, n\}$ such that $s(I) = \sum_{i \in I} p_i \geq B$ and $\forall j \in I, s(I) - p_j < B$. The objective is to maximize $s(I)$.

A polynomial-time reduction from Lazy Bureaucrat Problem yields the following result.

**Proposition 3.** Greedy Boss Problem is NP-complete.

As opposed to previous sections, we now make the assumption that the jobs are named in non-increasing order of durations: $p_1 \geq \cdots \geq p_n$. We propose and analyze Algorithm 6 for the Greedy Boss Problem.

**Algorithm 6 Max Greedy**

**Input:** $\{p_1, \ldots, p_n\}, B$

if $p_1 \geq B$ then
    return $\{1\}$
else
    Find the largest $t$ such that $s([1..t-1]) < B$
    return $[1..t]$
end if

The time complexity of Max Greedy is clearly $O(n)$ since we have assumed that jobs are already sorted.

**Proposition 4.** Max Greedy is a $1/2$-approximation algorithm. More precisely, if the solution $I$ returned by Max Greedy uses $t$ jobs, then $s(I) \geq \frac{t}{t+1} \text{OPT}$.

**Proof.** Let OPT denote the value of an optimum solution $I^*$, that is, $OPT = s(I^*)$. Assume that $I = [1..t]$ is the solution returned by Max Greedy and wlog., assume that $I^* \setminus I \neq \emptyset$ because otherwise we must have $I^* = I$. Let $r = \min(I^* \setminus I)$. By construction, we get: $\forall i \in I, p_i \geq p_r$. Hence,

$$p_r \leq \frac{1}{t} s(I) \tag{5}$$

Now, $OPT - p_r < B \leq s(I)$ because $I^* \setminus \{r\}$ is not feasible. Using previous inequality and inequality (5), we obtain $s(I) \geq \frac{t}{t+1} \text{OPT}$. In the worst case, $t = 1$ and the result follows. For the tightness, consider the following instance depending on $\varepsilon \in (0; 1/2)$: $n = 3$ with $p_1 = 1$, $p_2 = p_3 = 1 - \varepsilon$ and $B = 1$. 
The solution $I$ returned by Max Greedy is $I = [1]$ while the optimal solution is $I^* = [2, 3]$ because $\varepsilon < 1/2$. Hence $s(I) = 1 - 2\varepsilon$ which tends toward $1/2$ when $\varepsilon$ is very small. □

5 Future work

For the algorithmic part, it would be interesting to know if one can outperform Approx with another algorithm that runs in linear time. Another challenge is to find out whether the FPTAS given in this article can be extended to the case of heterogeneous arrival times (a PTAS is known for this case).

For the game theoretic part, we focused on mechanisms without money and have shown a tight bound of 2; we believe that it is worth studying mechanisms with payments in order to circumvent this bound for approximate strategyproof mechanisms.

Finally, a complete analysis of the Greedy Boss Problem should be conducted in terms of approximability and mechanism design.

References

Appendix A

Lemma 2. Let \( P = \{p_1, \ldots, p_n\} \), and \( B \) be an instance of Lazy Bureaucrat Problem. The following inequalities concerning \( I_G \) hold:

(i) \( s(I_G) \leq \OPT + p_s \), where \( s = \arg\min\{p_i : i \in I_G \setminus I^*\} \).

(ii) If \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \), then \( I^* \setminus [1..s] = \{i_1^*, i_2^*\} \) with \( s < i_1^* < i_2^* \) and \( i_2^* > s \). In particular, \( I^* = [1..s-1] \cup \{i_1^*, i_2^*\} \).

(iii) If \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \), then \( t = s + 2 \). In particular, \( I_G = [1..s+2] \).

Proof. If \( I_G \subseteq I^* \), then \( I_G = I^* \) by optimality of \( I^* \) and if \( I^* \subseteq I_G \), then \( I_G = I^* \) by construction of procedure Greedy. So, assume \( I^* \setminus I_G \neq \emptyset \) and \( I_G \setminus I^* \neq \emptyset \).

For (i). Let \( s = \arg\min\{p_i : i \in I_G \setminus I^*\} \). By feasibility of \( I^* \), \( \OPT + p_s > B \geq s(I_G) \).

For (ii). Assume \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \). We have \( p_s > \frac{4}{3}s(I_G) \) since otherwise (i) gives \( \frac{4}{3}\OPT > s(I_G) \). Hence, \( I^* \setminus [1..s] \subseteq \{i_1^*, i_2^*\} \), that is \( I^* \setminus [1..s] \) contains at most two jobs \( i_1^* < i_2^* \) and at least one job \( i_2^* \); moreover, \( i_2^* > s \). In this case, solution \( I^* = [1..s-1] \cup \{i_1^*, i_2^*\} \) with \( s < i_1^* < i_2^* \).

For (iii). Assume \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \). We have \( p_s > \frac{4}{3}s(I_G) \) since otherwise (i) gives \( \frac{4}{3}\OPT > s(I_G) \). So, \( t \leq s + 2 \). Using (ii), we know that \( t \geq s + 2 \) since otherwise \( t \leq s + 1 \) and then \( s(I_G) \leq \OPT \). In conclusion, \( t = s + 2 \) and \( I_G = [1..s+2] \).

Lemma 3. If \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \), then

\[
\text{APX} \leq 2p_1^* + s([1..s-1])
\]  

(6)

Proof. Assume \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \). So, \( i_2^* > t \) and we can assume that \( p_{i_2^*-1} + p_{i_2^*} \leq B \) because otherwise \( 2p_{i_2^*} \geq p_{i_2^*-1} + p_{i_2^*} > B \geq \text{APX} \) and inequality (1) is clearly satisfied. Hence \( I_{G-1} \) exists. We have \( s(I_{G-1}) \leq p_{i_2^*-1} + p_{i_2^*} + s([1..s-1]) \) because otherwise \( B \geq s(I_{G-1}) \geq p_{i_2^*-1} + p_{i_2^*} + s([1..s]) \) should not be feasible (\( I^* \) would added job \( s \)). Hence, \( \text{APX} \leq s(I_{G-1}) \leq p_{i_2^*-1} + p_{i_2^*} + s([1..s-1]) \leq 2p_1^* + s([1..s-1]). \)

Lemma 4. If \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \), then

\[
\text{APX} \leq 2p_1^* + s([1..s])
\]  

(7)

Proof. Assume \( s(I_G) > \frac{4}{3}\OPT \) and \( \text{APX} \neq \OPT \). Consider two cases: either \( i_1^* \leq t - 1 \) or \( i_1^* \geq t \). If \( i_1^* \leq t - 1 = s + 1 \), then \( i_1^* = s + 1 \) because \( s < i_1^* \) from (ii) of Lemma 2. In this case, solution \( I_{G-1}^* \) satisfies either \( s(I_{G-1}^*) \leq p_{i_1^*} + s([1..s]) \)
or \( s(I'_2) = p_{i_2^*} + s([1..s + 1]) \). In the first case, \( s(I'_2) \leq p_{i_2^*} + s([1..s]) \leq p_{i_2^*} + p_i^* + s([1..s - 1]) = \text{OPT} \) which contradicts the hypothesis \( \text{APX} \neq \text{OPT} \) while in the second case, \( B \geq s(I'_2) = p_{i_2^*} + s([1..s + 1]) = \text{OPT} + p_s \) which contradict the feasibility of \( I^* \). Hence, assume \( i_1^* \geq t \); Then, solution \( I_{i_1^* - 1} \) exists since \( p_{i_1^* - 1} + p_{i_1^*} \leq p_{i_1^*} + p_{i_2^*} \leq B \). We have \( s(I_{i_1^* - 1}) \leq p_{i_1^* - 1} + p_{i_1^*} + s([1..s]) \) because \( s + 2 = t \). In conclusion, \( \text{APX} \leq s(I_{i_1^* - 1}) \leq 2p_{i_1^*} + s([1..s]). \)

\[ \square \]

**Appendix B**

\textsc{traceback} for \textit{Exact} is a backtracking algorithm that works as follows. If \( \text{VAL}^* \) is equal to \( s(I_G) \) then \textsc{traceback} returns \( I_G \). Otherwise \( \text{VAL}^* < s(I_G) \) and \( \text{VAL}^* \) corresponds to the smallest element of the list \( L_{\text{ind}^*,n} \). One can retrieve the solution \( I^* \) with value \( \text{VAL}^* \) as follows. If \( \text{VAL}^* - p_n \in L_{\text{ind}^*,n-1} \) then \( n \in I^* \) and proceed inductively with the value \( \text{VAL}^* - p_n \) on \( L_{\text{ind}^*,n-1} \), otherwise \( \text{VAL}^* \in L_{\text{ind}^*,n-1}, n \notin I^* \) and proceed inductively with the value \( \text{VAL}^* \) on \( L_{\text{ind},n-1} \). \( I^* \) also contains the elements \( \{p_1, \ldots, p_{\text{ind}^* - 1}\} \). \textsc{traceback} for \textit{FPTAS} works similarly; just replace \( \text{VAL}^*, \text{ind}^* \) and \( I^* \) by \( \text{VAL}, \text{ind} \) and \( I_\epsilon \) respectively.