Common-Deadline Lazy Bureaucrat Scheduling Problems

Behdad Esfahbod, Mohammad Ghodsi, and Ali Sharifi

Computer Engineering Department
Sharif University of Technology, Tehran, Iran,
{behdad,ghodsi}@sharif.edu, ali@bamdad.org

Abstract. The lazy bureaucrat scheduling is a new class of scheduling problems that was introduced in [1]. In these problems, there is one employee (or more) who should perform the assigned jobs. The objective of the employee is to minimize the amount of work he performs and to be as inefficient as possible. He is subject to a constraint, however, that he should be busy when there is some work to do.
In this paper, we focus on the cases of this problem where all jobs have the same common deadline. We show that with this constraint, the problem is still NP-hard, and prove some hardness results. We then present a tight 2-approximation algorithm for this problem under one of the defined objective functions. Moreover, we prove that this problem is weakly NP-hard under all objective functions, and present a pseudo-polynomial time algorithm for its general case.

Keywords: Scheduling Problems, Approximation Algorithms, Dynamic Programming, NP-hardness.

1 Introduction

In most scheduling problems, there is a number of jobs to be performed by some workers or employees. Studies have looked at these problems with the objective to perform the assigned jobs as efficient as possible and to maximize the number of completed jobs. This is the employer's point of view. We can also look at these problems from employee's point of view, some of whom do not have enough motivation to do their jobs efficiently. Some may even want to be as inefficient as possible while performing their duties. We call such employees *lazy*, and such scheduling problems has been classified as *Lazy Bureaucrat Scheduling Problems* (LBSP). LBSP was introduced in [1] and some results were presented there. A summary of some of them is presented here.

More specifically, we are given a set J of n jobs $j_1, \ldots, j_n$. Job j_i has *processing time* of t_i, *arrival time* of a_i , and *deadline* of d_i ($t_i \leq d_i - a_i$). It is assumed that t_i, a_i, and d_i are nonnegative integers. Jobs have hard deadlines, that is, j_i can only be executed during its allowed interval $w_i = [a_i, d_i]$, called its *window*. It is also assumed that there always exists some job which arrives at time 0. The maximum deadline time is also denoted by D.

We study the non-preemptive case of this problem and restrict ourselves to off-line scheduling in which all jobs are known to the scheduler beforehand. We also assume that there is only one processor (employee) available to do the jobs. Some results on preemptive case and also cases with multiple bureaucrats can be found in [1, 4].

Definition 1. (Executable Job) A job j_i is called *executable* at some time t, if and only if it has been arrived, its deadline has not yet passed, it is not processed yet, and if it is started now, it will be fully processed before its deadline (that is, $a_i \leq t \leq d_i - t_i$).

Definition 2. (Greedy Requirement) At any time, the bureaucrat should work on an executable job, if there is any such job.

The bureaucrat is asked to process the jobs within their windows satisfying the above greedy requirement. His goal is to be as inefficient as possible. This is captured by any of the following objective functions that is to minimize.

1.1 Objective Functions

Four objective functions have been defined for this problem [1].

1. `[min-time-spent]`: *Minimize the total amount of time spent working.* This objective naturally appeals to a *lazy* bureaucrat.
2. `[min-weighted-sum]`: *Minimize the weighted sum of completed jobs.* This objective appeals to a *spiteful* bureaucrat whose goal is to minimize the fees that the company collects on the basis of his labors, assuming that the fee is collected only for those tasks that are actually completed.
3. `[min-makespan]`: *Minimize the makespan, the maximum completion time of the jobs.* This objective appeals to an *impatient* bureaucrat, whose goal is to go home as early as possible, at the completion of the last job, when he is allowed to leave. He cares only about the number of hours spent at the office, not the number of hours spent doing work (productive or otherwise).
4. `[min-number-of-jobs]`: *Minimize the total number of completed jobs.* This can be meaningful when the overhead of performing a job is too high for the bureaucrat.

Clearly, the objective function 1 is a special case of 2, as we can define the weight of each job to be equal to its length. Objective function 4 is also a special case of 2, when the weights are all equal. Also, if all jobs have the same arrival time, the objective functions 1 and 3 are equivalent.

1.2 Previous Related Results

The main related results in [1] are summarized below:

- LBSP is strongly NP-hard [2] under all objective functions and is not approximable to within any fixed factor.

– LBSP with the same arrival times for the jobs, is weakly NP-hard, and can be solved by a pseudo-polynomial dynamic programming algorithm.
– LBSP with all the jobs having unit lengths, can be solved in polynomial time by the Latest Deadline First (LDF) scheduling policy.
– Assuming for each job i, $d_i - a_i < 2t_i$, LBSP can be solved in $O(nD \max(n, D))$ time.
– Even with a bound on δ (the ratio of the longest job to the shortest job), LBSP is strongly NP-hard. It cannot be approximated to within a factor of $\delta - \epsilon$, for any $\epsilon > 0$, unless $P = NP$.
– Given bounds on R (the maximum ratio of window length to job length) and δ, LBSP can be solved in $O(Dn^{4R \lg \delta})$.

In [3], these results have been shown:

– LBSP with all jobs having unit lengths, can be solved in polynomial time by the Latest Deadline First (LDF) scheduling policy, even with more than one bureaucrat (worker, processor).
– Assuming $d_i - a_i < 2t_i$ for each job i, LBSP can be solved in $O(nD)$ time.

Hepner and Stein have studied the preemptive case [4] where they propose pseudo-polynomial time algorithm to minimize makespan of such schedule. They have also extended this scheduling problem to the multiple-bureaucrat setting and provided pseudo-polynomial time algorithms for such problems.

1.3 Our Results

We consider a restricted case of LBSP where the deadlines for all jobs are the same (denoted by D). We call these problems common-deadline LBSP, denoted by CD-LBSP. This problem can be considered with any of the above objective functions. We denote such cases by CD-LBSP`[`*`objective-function`*`]` and CD-LBSP`[*]` is used to denote all these objective functions.

On the hardness of this problem, we first prove that CD-LBSP`[*]` is still NP-hard and show that CD-LBSP`[min-number-of-jobs]`is not approximable to within any fixed factor. But, for CD-LBSP`[min-makespan]`, we provide a tight 2-approximation algorithm. Finally, we prove that CD-LBSP`[*]` is weakly NP-hard; we provide a pseudo-polynomial time dynamic programming algorithm for the general case.

2 Hardness Results

In the following theorems we reduce the Subset Sum problem to prove that CD-LBSP`[*]` is NP-hard, but existence of approximation algorithms is not the same under different objective functions. We have found reasonable results for `[min-weighted-sum]`, `[min-makespan]`, and `[min-number-of-jobs]` objective functions, but not for `[min-time-spent]`.

Theorem 1. CD-LBSP`[*]` *is NP-hard.*

Proof. We reduce the Subset Sum problem to this problem. We are given $S = \{x_1, \ldots, x_n\}$ of n positive integers with $\Sigma_{i=1}^{n} x_i = s$, and an integer b $(0 < b < s)$. It is asked whether there is a subset T of S, satisfying $\Sigma_{x \in T}\, x = b$? Without loss of generality, we assume that $b \leq \lfloor \frac{s}{2} \rfloor$ and $x_i < b$ for all i.

We construct an instance of CD-LBSP containing $n+1$ jobs, all having deadlines $D = 2s$. For each $x_i \in S$, we define a job j_i $(1 \leq i \leq n)$ with arrival time $a_i = 0$ and processing time $t_i = 2x_i$. The last job j_{n+1} has arrival time $a_{n+1} = 2b$ and processing time $t_{n+1} = 2s - 2b - 1$.

The bureaucrat is to avoid working up to time $2s$, and to finish by $2s-1$. This can be done if and only if he starts j_{n+1} at time of $2b$ and finishes it at time $2s-1$. This is the case if and only if there is a subset of $\{j_1, \ldots, j_n\}$ with total processing time $2b$, which is equivalent to a solution for the Subset Sum problem.

This argument is clearly correct for objective functions `[min-time-spent]`, `[min-weighted-sum]`, and `[min-makespan]`. For `[min-number-of-jobs]`, the assumptions we made for the data in the Subset Sum problem is used to prove this case. □

Theorem 2. CD-LBSP `[min-number-of-jobs]` *is not approximable to within any fixed factor* $\Delta > 1$*, unless* $P = NP$*.*

Proof. As in theorem 1, we reduce the Subset Sum problem to prove the hardness. Assume by contradiction that there is an approximation algorithm with fixed factor Δ for CD-LBSP `[min-number-of-jobs]`. Using this assumption, we will show that we can solve the Subset Sum problem leading to $P = NP$.

Let $m = \lceil \Delta \rceil$ and $D = b + m(n+2)s$. We construct an instance of CD-LBSP `[min-number-of-jobs]` containing the following jobs, all with deadline D. For each $x_i \in S$, we define an *element job* j_i $(1 \leq i \leq n)$ having $a_i = 0$, and $t_i = x_i$. One *long job*, j_{n+1} with $a_{n+1} = b$ and $t_{n+1} = D - b$. We also define $m(n+2) - 1$ *extra jobs*, all having arrival times b, and processing times s.

The bureaucrat wants to do as few jobs as possible. We claim that the answer to the Subset Sum problem is 'yes', if and only if the bureaucrat performs the long job. In this case, he should be working up to time b, which leads to the answer of the Subset Sum problem and he has processed at most $n+1$ jobs (at most n element jobs and one long job), so the approximation algorithm will produce an output with at most $m(n+1)$ jobs. On the other hand, if he does not process the long job, then he has to process $m(n+2) - 1$ extra jobs (he has enough time to process all element and extra jobs), so the approximation algorithm will end with more than $m(n+1)$ jobs.

Then, the answer to the Subset Sum problem is 'yes' if and only if there are at most $m(n+1)$ jobs in the output of the algorithm, and 'no' otherwise. □

Corollary 1. CD-LBSP `[min-weighted-sum]` *is not approximable to within any fixed factor* $\Delta > 1$*, unless* $P = NP$*.*

Proof. We know that CD-LBSP `[min-number-of-jobs]` is a special case of CD-LBSP `[min-weighted-sum]`. Theorem 2 completes the proof. □

3 Approximation Algorithm

It is shown in this section that, under objective function 3 ([min-makespan]), the bureaucrat can reach a nearly optimal solution with a simple algorithm.

For a schedule σ, we say $j_i \in \sigma$ if job j_i is processed in σ. Also, we use $s_i(\sigma)$ and $f_i(\sigma)$ to denote the time when the processing of j_i is started and the time when it is finished in σ respectively ($f_i(\sigma) = s_i(\sigma) + t_i$).

Theorem 3. *The Shortest Job First (SJF) scheduling policy is a 2-approximation algorithm for* CD-LBSP [min-makespan] *and this bound is tight.*

Proof. Let σ_{OPT} be an optimal solution and σ be the schedule which the SJF policy has generated, and OPT and SJF be their makespans respectively. In the SJF policy, among the executable jobs, the one with the shortest processing time is picked first. We will show that $SJF - OPT < OPT$.

Without loss of generality, suppose that jobs $j_1 \ldots j_k$ (for some k) are processed in σ in that order. Let $j_q \in \sigma$ be the job that is being processed at time OPT in σ. That is, $s_q(\sigma) < OPT \leq f_q(\sigma)$. We know that $a_i < OPT$ for all jobs j_i. The SJF policy, therefore, forces that $t_{q+1} \leq t_{q+2} \leq \ldots \leq t_k$. Note that there is no gap between jobs $j_q, \ldots, j_k$ in σ. From the greedy requirement, we can easily conclude that $j_i \in \sigma_{OPT}$ for all $q + 1 \leq i \leq k$ (otherwise, at least one of them can be processed at time OPT).

Assume that $q < k$. The $q = k$ case is treated similarly and we omit it. We consider two different cases:

- $t_q \leq t_{q+1}$: With the same argument as above, we have $j_q \in \sigma_{OPT}$. Therefore, since $s_q(\sigma) < OPT$, we have $SJF - OPT < \Sigma_{i=q}^{k} t_i \leq OPT$.
- $t_q > t_{q+1}$: If job $j_q \in \sigma_{OPT}$, then we will have $SJF - OPT < \Sigma_{i=q}^{k} t_i \leq OPT$ and $SJF < 2OPT$; otherwise, we can show that there exists a job j_p, not shorter than j_q, such that $j_p \in \sigma_{OPT}$ and $j_p \notin \sigma$. Let $Q = \{j_i \mid q < i \leq k\}$. All jobs in Q are shorter than j_q, otherwise, there will be enough time to process j_q at time OPT, which means that σ_{OPT} is invalid. Also, from the SJF policy, we conclude that all jobs in Q have arrived after $s_q(\sigma)$.
  Let T be the set of jobs $j_i \in \sigma_{OPT}$ such that $s_i(\sigma_{OPT}) \leq s_q(\sigma)$. Obviously, T cannot be empty. Not all jobs in T can be processed in σ. To show this, consider a subset of T which has been processed continuously (without any break) and has the maximum start time among these subsets. The finish time of this subset is after time $s_q(\sigma)$. From the greedy requirement, we conclude that this set of jobs cannot be processed and finished by time $s_q(\sigma)$, and thus they cannot all be processed in σ (because job j_q has been already started at time $s_q(\sigma)$.) Thus, there is some job $j_p \notin \sigma$, with $a_p \leq s_q(\sigma)$. The SJF policy forces that $t_q \leq t_p$, and also $j_p \in \sigma_{OPT}$. Putting all together leads to

$$SJF - OPT < t_q + \Sigma_{i=q+1}^{k} t_i \leq t_p + \Sigma_{i=q+1}^{k} t_i < t_p \leq OPT.$$

To prove the tightness, for a given n and $0 < \epsilon < 1$, we construct an instance of CD-LBSP with n jobs such that the SJF policy does not do better than

$2-\epsilon$. Such instance contains n jobs with zero arrival times and deadlines D. Let $D = n-3+2L-1 = n+2L-4$ with $L = 2n/\epsilon$. The first three jobs have $t_1 = L-1$, $t_2 = L$, and $t_3 = L+1$. The next $n-3$ jobs, $j_4, \ldots, j_n$, all have $t_i = 1$.

The σ_{OPT} should process all $j_4, \ldots, j_n$ jobs and then j_3, having makespan $OPT = n-3+L+1 = n+L-2$. There is not enough time to process j_1 or j_2.

The SJF schedule processes $j_4, \ldots, j_n$, j_1, and j_2 having makespan $SJF = n-3+L-1+L = n+2L-4$. Thus, we have

$$\frac{SJF}{OPT} = \frac{n+2L-4}{n+L-2} = \frac{n\epsilon+4n-4\epsilon}{n\epsilon+2n-2\epsilon} = \frac{\epsilon+4-\frac{8}{L}}{\epsilon+2-\frac{4}{L}} > 2-\frac{\epsilon}{2(1-\frac{2}{L})} > 2-\epsilon,$$

which completes the proof. □

It looks quite likely that the same algorithm yields the same approximation bound under function `[min-time-spent]`, but we do not have a complete proof.

4 Pseudo-Polynomial Time Algorithms

We assume that the jobs are numbered in order of their arrival times (that is, $a_1 \leq a_2 \leq \ldots a_n$). Let T_i and $T_{i,k}$ denote the set of jobs $j_i, j_{i+1}, \ldots, j_n$ and $j_i, j_{i+1}, \ldots, j_k$ respectively. We will also use the following definitions:

Definition 3. The time α is called the *first rest time* of a schedule σ, if the bureaucrat has paused processing the jobs in σ for the first time at α. If there is no pause during σ, the first rest time is defined as the time when the schedule is finished.

Definition 4. For a time α, we define *critical jobs* H_α as the set of jobs $j_i \in J$ which can be processed in $[\alpha, D]$, i.e. $\max(a_i, \alpha) + t_i \leq D$.

Definition 5. For a given (T, α, U) in which $T, U \subset J$ and α is a time point, sequence E of some jobs in T is said to be a *valid sequence* if we can process these jobs in this order without any gaps in between, starting from first arrival time of the jobs in T and finishing at α such that every job in $T \cap U$ appears in E. A valid sequence E is said to be an *optimal sequence* under some objective function, if its cost is minimum among all valid sequences of (T, α, U).

Lemma 1. *For a given (T, α, U), let E be an optimal sequence and $j_m \in E$ be the job with the latest arrival time. There exists another optimal sequence F in which j_m is the last processed job.*

Proof. This can easily be done by repeated swapping of j_m with its adjacent jobs. □

Lemma 2. *There is a pseudo-polynomial time algorithm that finds the optimal sequence for any given (T_i, α, U) under any of the objective functions, if such sequence exists $(1 \leq i \leq n)$.*

Proof. Let j_f be the last job arrived before α in T_i, and $C_{x,y}$ ($i \le x \le f, a_i \le y \le \alpha$) be the cost of the optimal sequence for $(T_{i,x}, \alpha, U)$, or ∞ if no such optimal sequence exists. Our goal is to compute $C_{f,\alpha}$. We show how $C_{x,y}$ can be computed recursively from the values of $C_{x',y'}$, where $x' < x$ and $y' \le y$.

If $j_x \in U$, then it is in any valid sequence. Hence, from lemma 1, j_x can be processed last in $[y - t_x, y]$. Based on the objective function used, we can easily compute $C_{x,y}$ from $C_{x-1,y-t_x}$. For example, $C_{x,y} = C_{x-1,y-t_x} + t_x$ under `[min-time-spent]`.

On the other hand, if $j_x \notin U$, there are two options depending on whether or not it is in the optimal sequence. If j_x is processed in the optimal sequence, it can be processed last, in which case, $C_{x,y}$ can be computed from $C_{x-1,y-t_x}$ as before. Otherwise, $C_{x,y} = C_{x-1,y}$, since we can ignore j_x. The minimum of these two values is taken for $C_{x,y}$.

The running time of this algorithm is $O(nD)$, as there are at most nD values of $C_{x,y}$ to compute. □

Theorem 4. CD-LBSP`[*]` *is weakly NP-hard.*

Proof. We present a pseudo-polynomial time algorithm which can be used for any of the objective functions.

Consider T_i for some $1 \le i \le n$ and temporarily assume that the jobs in T_i are the only jobs available, and that the greedy requirement is to be satisfied on only these jobs. Let P_i be this subproblem and C_i be its optimal value. Clearly, C_1 is the desired value.

Consider an optimal schedule σ for P_i. Let α be the first rest time in σ. No job in T_i arrives at α. We know that the jobs in T_i appearing in the set of critical jobs H_α should be processed before the rest time α.

Let j_k be the first job arrived after α. Because of the pause at time α, we know that no job having arrival time less than α can be processed after α. So, we can break up the schedule σ into two subschedules: those processed before α and those processed after. These subschedules are independent. We can consider the first subschedule as a valid sequence for $(T_{i,k-1}, \alpha, H_\alpha)$. From the optimality of σ, it is clear that this sequence is optimal. Similarly, the second subschedule is an optimal schedule for P_k.

We compute C_i for every $1 \le i \le n$ from the values of C_j ($i < j \le n$) for all times α ($a_i < \alpha \le D$). Note that we only consider those α's at which time there is no job arrival. It is first checked whether there exists an optimal sequence for $(T_{i,k-1}, \alpha, H_\alpha)$. If there is no such sequence, there will be no schedule for T_i having α as the first rest time; otherwise, let C be the cost of that sequence. We know that the lowest cost of a schedule for T_i having α as the first rest time can be computed easily from the values of C and C_k and the objective function used. For example, under `[min-time-spent]` this is equal to $C + P_k$. The value of P_i is the minimum cost for different values of α.

The running time of this algorithm is $O(n^2D^2)$ because it calls the subroutine of finding optimal sequence at most $O(nD)$ times. □

5 Conclusion

In this paper, we studied a new class of the Lazy Bureaucrat Scheduling Problems (LBSP), called common-deadline LBSP, where the deadlines of all jobs are the same. We proved that this problem is still NP-hard under all four pre-defined objective functions. We also showed that this problem is not approximable to within any fixed factor in cases of `[min-weighted-sum]` and `[min-number-of-jobs]` objective functions. The problem is shown to have a tight 2-approximation algorithm under `[min-makespan]`. But, it is still open whether it is approximable under `[min-time-spent]`. In the rest of the paper, we presented pseudo-polynomial time dynamic programming algorithms for this problem under all objective functions. Further work on this problem is underway.

Acknowledgements. The authors would like to thank the anonymous referees for their useful comments.

References

1. Arkin, E. M., Bender, M. A., Mitchell, J. S. B., Skiena, S. S.: The lazy bureaucrat scheduling problem. Workshop on Algorithms and Data Structures (WADS'99), LNCS 1663, pp. 122–133, Springer-Verlag, 1999.
2. Gary, M. R., Johnson D. S.: Computers and intractability, a guide to the theory of NP-completeness. W. H. Freeman and Company, New York, 1979.
3. Farzan, A., Ghodsi, M.: New results for lazy bureaucrat scheduling problem. 7th CSI Computer Conference (CSICC'2002), Iran Telecommunication Research Center, March 3–5, 2002, pp. 66–71.
4. Hepner, C., Stein, C.: Minimizing makespan for the lazy bureaucrat problem, SWAT 2002, LNCS 2368, pp. 40–50, Springer-Verlag, 2002.