Optimization Letters
https://doi.org/10.1007/s11590-024-02138-5

ORIGINAL PAPER

Minimizing the number of tardy jobs with generalized due-dates and position-dependent processing times

Enrique Gerstl[1] · Gur Mosheiov[1]

Received: 10 July 2023 / Accepted: 2 July 2024

Abstract
We study a single machine scheduling problem with generalized due-dates and general position-dependent job processing times. The objective function is minimum number of tardy jobs. The problem is proved to be NP-hard in the strong sense. We introduce an efficient algorithm that solves medium size problems in reasonable running time. A simple and efficient heuristic is also introduced, which obtained the optimal solution in the vast majority of our tests.

Keywords Scheduling · Single machine · Generalized due-dates · Position-dependent processing times · Minimum number of tardy jobs

1 Introduction

The single machine model studied in this paper combines two popular features in scheduling theory: (1) Generalized due-dates (*gdd*), and (2) Position-dependent job processing times. In a *gdd* setting, the due-dates are not job-specific, but the input contains a sorted set of numbers, such that the j-th number is the due-date assigned to the job processed in the j-th position. When position-dependent processing times are considered, the processing time of a given job varies in the most general way as a function of its position in the sequence. The objective function considered is minimum number of tardy jobs.

Hall [1] introduced the concept of generalized due-dates, and Hall et al. [2] provided the first set of complexity results for a number of problems considering classical scheduling measures. Some of the recently published papers dealing with various extensions of these *gdd* models are: Gerstl and Mosheiov [3], Choi and Park [4], Park et al. [5], Choi et al. [6], Gerstl and Mosheiov [7], Choi et al. [8], Li and Chen [9], Park et al. [10], Mor et al. [11], Choi et al. [12] and Mosheiov et al. [13, 14]. These papers contain numerous applications of *gdd* scheduling. An interesting

✉ Enrique Gerstl
enrique.gerstl@mail.huji.ac.il

1 Present Address: School of Business Administration, The Hebrew University, Jerusalem, Israel

example from the petrochemical industry was described in the original paper of Hall [1]. In this setting, a number of interchangeable heat exchangers must be maintained up to a certain date. The identity of the heat exchangers is immaterial, implying that the problem can be formulated as a *gdd* scheduling problem.

Scheduling with position-dependent job processing times is also a wide area which has been studied by many researchers in the last three decades. Most of the researchers considered either learning see e.g., Biskup [15, 16], or job deterioration see e.g., Yin et al. [17] and Huang and Wang [18]. These two settings assume monotonicity: when a learning effect is considered, the job processing times are non-increasing as a function of their position, in the case of deterioration, the job processing times are assumed to be non-decreasing as a function of the job-position. In this paper we assume position-dependent processing time in the most general way, i.e., they do not follow any given function, and in particular no monotonicity is assumed. This general form is justified e.g., in settings consisting of the following two effects: (1) a learning process (of the scheduler/producer) is valid and as a result the processing times decrease for jobs performed later in the sequence, and (2) the entire system deteriorates, which leads to larger processing times of late processed jobs. In such systems, the impact of the job position on its processing time may change, and in particular the resulting sequence is not necessarily monotone. Some examples of studies of scheduling with general position-dependent job processing times are: Mosheiov [19], Gerstl and Mosheiov [20], Yu et al. [21], Agnetis and Mosheiov [22], Gerstl et al. [23], Pei et al. [24], Fiszman and Mosheiov [25], Kovalyov et al. [26], Yang and Lu [27], Mosheiov et al. [28], Montoya-Torres et al. [29], and Przybylski [30].

As mentioned, the scheduling measure considered here is minimum number of tardy jobs. The classical single-machine problem of minimizing the number of tardy jobs is solved in $O(nlogn)$ time (where n is the number of jobs); see Moore [31]. Numerous extensions of this problem have been studied since. Some prominent examples are: Ho and Chang [32], Lann and Mosheiov [33], Lodree et al. [34], Mosheiov and Sidney [35], Mosheiov and Oron [36], Adamu and Adewumi [37], Allahverdi et al. [38], Aydilek et al. [39], Mor et al. [40], He et al. [41] and Hermelin et al. [42].

Our paper studies for the first time a setting combining all these features: (1) Generalized due-dates, (2) Position-dependent job processing times, and (3) The objective of minimum number of tardy jobs. The single-machine problem combining features (1) and (3), i.e., Minimizing the number of tardy jobs with generalized due-dates, is easily shown to be solved in polynomial time as well. Specifically, it is solved by the Shortest Processing Times first (SPT) policy, i.e., in $O(nlogn)$ time as well. On the other hand, the complexity status of the single-machine problem combining all three features was unknown. In this paper we prove that the problem is strongly NP-hard. Consequently, we introduce first an efficient heuristic of a greedy nature. Then, an exact solution algorithm is introduced. This algorithm finds the optimal schedule significantly faster than a standard full-enumeration procedure.

A numerical study is performed in order to measure (1) The running time required by the (exact) algorithm, and (2) The percentage of the instances solved to optimality by the heuristic.

The paper is organized as follows: Sect. 2 contains the notation and the formulation. Section 3 presents the NP-hardness proof. In Sect. 4 we introduce the exact solution algorithm and the heuristic. The results of our numerical tests are reported in Sect. 5. Conclusions and some ideas for future research are provided in the last section.

2 Formulation

We study a single machine n-job scheduling problem. p_{jr} denotes the processing time of job j if assigned to position $r;j, r = 1, \dots, n$. d_r denotes the r-th generalized due-date, i.e., the due-date of the job assigned to position $r, r = 1, \dots, n$.

For a given schedule of the jobs, C_r denotes the completion time of the job in position r, $r = 1, \dots, n$. The tardiness of the job in position r is denoted by $T_r = \max\{C_r - d_r, 0\}, r = 1, \dots, n$. U_r is the tardiness indicator, i.e., $U_r = 1$ if $T_r > 0$, and $U_r = 0$ otherwise ($T_r = 0$). The objective function is minimum number of tardy jobs: $\sum_{r=1}^{n} U_r$. Hence, the problem studied here is:

$$1\left|p_{jr}, gdd\right| \sum U_r$$

3 NP-hardness proof

In this section we study the complexity status of the problem.

Theorem 1 : *Problem* $1\left|p_{jr}, gdd\right| \sum U_r$ *is strongly NP-Hard.*

Proof The proof is by reduction from 3-Partition.

3-Partition: Consider a set A of positive rational numbers, $A = \{a_1, a_2, \dots, a_{3t}\}$, where $\sum_{i=1}^{3t} a_i = t$ and $\frac{1}{4} < a_i < \frac{1}{2}, i = 1,2, \dots, 3t$. Can the set A be partitioned into t disjointed triplets, $A_1, A_2, \dots, A_t$, such as $\sum_{i \in A_j} a_i = 1, j = 1,2, \dots, t$?

We construct the following instance of the scheduling problem from 3-partition:

There are $n = 3t$ jobs. The position-dependent processing time of job j if assigned to position r ($p_{jr}, j, r = 1, \dots, n$) is the following:

$$
\begin{aligned}
&p_{j1} = p_{j2} = p_{j3} = a_j,\\
&p_{j4} = p_{j5} = p_{j6} = 2a_j,\\
&p_{j7} = p_{j8} = p_{j9} = 3a_j,\\
&\dots\\
&p_{j(3k-2)} = p_{j(3k-1)} = p_{j3k} = ka_j,\\
&\dots\\
&p_{j(3t-2)} = p_{j(3t-1)} = p_{j3t} = ta_j.
\end{aligned}
$$

Let the generalized due-dates be:

$$
\begin{aligned}
&d_1 = d_2 = d_3 = 1,\\
&d_4 = d_5 = d_6 = 3,\\
&d_7 = d_8 = d_9 = 6,\\
&\dots\\
&d_{3k-2} = d_{3k-1} = d_{3k} = \sum_{i=1}^{k} i,\\
&\dots\\
&d_{3t-2} = d_{3t-1} = d_{3t} = \sum_{i=1}^{t} i.
\end{aligned}
$$

The scheduling measure is minimum number of tardy jobs.

The recognition version (RV) of this scheduling problem: Is there a schedule with no tardy jobs?

We prove in the following that there is a YES answer to 3-Partition if and only if there is a YES answer to RV.

($\Rightarrow$) Assume first that there is a YES solution to 3-partition. Then, schedule the jobs of the first triplet A_1 in the first 3 positions, the jobs of the second triplet A_2 in the next 3 positions, etc. It follows that the third job is completed at time $d_3 = 1$, the sixth job is completed exactly at time $d_6 = 3$, and so on; see Fig. 1. The resulting sequence contains no tardy jobs.

($\Leftarrow$) Assume now that there is NO solution to 3-partition. It follows that any allocation to triplets contains some with total load strictly smaller than 1, and some with total load strictly larger than 1. Consider the case that there is a single triplet of each type: triplet A_k in which the sum of its elements is strictly smaller than 1 (say, $1 - \epsilon$), and triplet A_l, in which the sum of its elements is strictly larger than 1 ($1 + \epsilon$).

We create a schedule based on this allocation to triplets. Consider first a schedule in which $l < k$, i.e., the triplet of jobs A_l is processed before the triplet A_k. In this case, the completion time of job $3l$ (the last job in this triplet) is $1 + 2 + \dots l - 1 + l(1 + \epsilon) = \sum_{i=1}^{l} i + l\varepsilon > \sum_{i=1}^{l} i = d_{3l}$. Hence, this schedule contains at least one tardy job, implying a NO solution to 3-partition (Fig. 2). Assume now that the triplet of jobs A_l is processed after the triplet A_k. Assume that there are x triplets between them (i.e.,

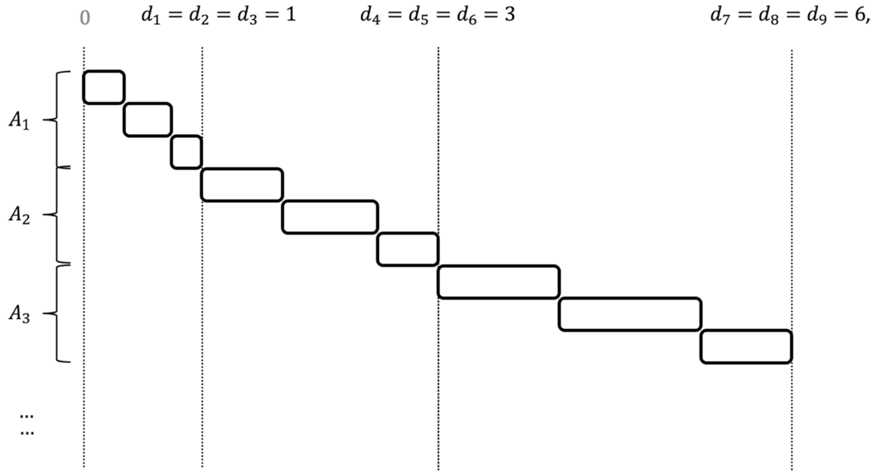

Fig. 1 A feasible schedule of $3t$ jobs with no tardy jobs

$l = k + x$). Then, the completion time of job $3l$ (the last job in triplet A_l) is: $1 + 2 + \ldots k - 1 + k(1 - \epsilon) + (k + 1) + (k + 2)+,,,+(k + x - 1) + (k + x)$ $(1 + \epsilon) = \sum_{i=1}^{l} i - k\epsilon + (k + x)\epsilon = \sum_{i=1}^{l} i + x\epsilon > \sum_{i=1}^{l} i = d_{3l}$. Hence, as above, this schedule contains at least one tardy job, implying a NO solution to 3-Partition. ■

4 A heuristic and an exact solution algorithm

Since the problem $1\left|p_{jr}, gdd\right| \sum U_r$ is NP-hard in the strong sense, we focus in this section on the introduction of a simple heuristic and of an efficient exact solution algorithm. The heuristic (denoted by *H*) is based on assigning the jobs sequentially from position 1 to position *n*. It is of a greedy nature - at each iteration (position), the shortest possible job is selected. Let *G* denote the set of unscheduled jobs so far. Initially, $G = \{j = 1, \ldots, n\}$. Let $\vec{S}$ denote the resulting job sequence. Clearly, the initial sequence is empty. In the first iteration, we schedule the shortest job (in the first position: $i = 1$). The index of this job is denoted by *minJobIndex*: $minJobIndex = argmin_{j \in G}\{p_{j1}\}, j = 1, \ldots, n$. This job is removed from *G*: $G = \{G \backslash minJobIndex\}$ and added to the job sequence $\vec{S}$ in position $i = 1$. We repeat the same job selection procedure (from the unscheduled jobs) for the remaining positions: positions 2 to *n*. The following pseudo code introduces in detail:

Algorithm $\boldsymbol{H}$ ($p_{jr}, d_i;\ j, i, r = 1, \dots n$):

```
    $Define\ G = \{j = 1, \dots, n\}$;                  // The group of unscheduled jobs.
    $Define\ \vec{S} = (\emptyset, \emptyset, \dots, \emptyset)$           // The optimal job sequence (initially empty).
    $Define\ minProcessing$;                 // The minimum processing time.
    $Define\ minJobIndex$;                   // The index of the job with minimum processing time.
    $Define\ numOfTardyJobs$;                // Number of tardy jobs.
    $numOfTardyJobs = 0$;                    // Initialize the number of Tardy jobs to zero.

    $for\ i = 1\ to\ i = n$                  // Iterate on $n$ positions.
        $minProcessing = \infty$;            // Initialize the minimum processing time.
        $for\ j = 1\ to\ j = n$              // For each position all non-scheduled jobs are
                                             // considered.
            // We check if the processing time of job $j$ in position $i$ is the smallest so far
            $if\ (j \in G)\ and\ (p_{jr} < minProcessing)$
                $minProcessing = p_{ji}$;
                $minJobIndex = j$;
                $\vec{S}[i] = j$;           // The job with minimum processing time is assigned
        to
                                             //  position $i$.
            $G = \{G \backslash minJobIndex\}$;   //Remove $minJobIndex$ from $G$.
        $C = C + minProcessing$;             // Update the total completion time.

        $if\ (C > d_i)$                      // Check if the job is tardy.
            $numOfTardyJobs = numOfTardyJobs + 1$;

$return\ numOfTardyJobs, \vec{S}$;
```

Running Time: We consider all n positions. For each position, $O(n)$ jobs are checked. The calculation of the completion time of the selected job requires constant time. It follows that the total running time of H is $O(n^2)$.

We now introduce an exact non-polynomial algorithm (denoted F) that solves to optimality problem $1\left|p_{jr}, gdd\right| \sum U_r$. The algorithm starts by running H and obtaining a feasible schedule and an upper bound on the optimal number of tardy jobs. The procedure is based on building partial schedules at each iteration such that a job is added to the sequence and the number of tardy jobs (so far) and the upper bound are updated. However, we avoid the evaluation of all $n!$ schedules by discarding partial schedules when the number of tardy jobs (*minNumOfTardyJobs*) exceeds the current upper bound. (It should be noted that in the worst case, the algorithm evaluates the assignment of all jobs to all positions.) After obtaining the initial upper bound, Algorithm F calls function L that evaluates all options of adding the unscheduled jobs to the remaining positions.

In the following we introduce Algorithm F and the function L.

Algorithm F $\boldsymbol{F}(p_{jr}, d_r)$

```
    $Define\ G = \{j = 1, \ldots, n\}$;                  // The group of unscheduled jobs.
    $Define\ minNumOfTardyJobs$;                       // Upper bound on the number of tardy jobs.
    $Define\ \overrightarrow{S_{opt}} = (\emptyset, \emptyset, \ldots, \emptyset)$   // The optimal job sequence (initially empty).

    $minNumOfTardyJobs, \overrightarrow{S_{opt}} = H(p_{jr}, d_r;\ j, r = 1, \ldots n)$;// Obtain the initial upper bound and sequence.

    $for\ j = 1 \rightarrow n$                              // Evaluates all possible sequences.
        $j \in G$;                                         // Set all jobs as unscheduled.
    Call $\boldsymbol{L}(minNumOfTardyJobs, 1, 0, 0, G, \overrightarrow{S_{opt}})$;   // Evaluate all jobs.
    Return $minNumOfTardyJobs, \overrightarrow{S_{opt}}$;          // Return the number of tardy jobs.


Function $\boldsymbol{L}(minNumOfTardyJobs, i, C, tardyJobs, G, \vec{S})$
        $If\ i = n$                                         // Check if all jobs are scheduled.
            $if\ tardyJobs < minNumOfTardyJobs$         // Check if the number of tardy jobs in the
                                                        // sequence is smaller than the upper bound.
                $minNumOfTardyJobs = tardyJobs$; // If this is the case update the upper
bound.
                $\overrightarrow{S_{opt}} = \vec{S}$;                            // Update the optimal job-sequence.
            $return\ minNumOfTardyJobs, \overrightarrow{S_{opt}}$;

        $If\ (tardyJobs \geq minNumOfTardyJobs)$        // Check if the number of tardy jobs exceeds the
                                                           // upper bound.
            $return\ minNumOfTardyJobs, \overrightarrow{S_{opt}}$;     // Discard the partial schedule and return the
                                                        // current upper bound and optimal sequence.

        $for\ j = 1 \rightarrow n$
            $if\ j \notin G$                                     // If $j$ is scheduled continue to the next
job.
                $continue\ to\ next\ iteration$;
            $else$
                $G = \{G \backslash j\}$;                       // Remove $j$ from $G$.
                $\vec{S}[i] = j$;                          // Add job $j$ to position $i$
                  $if\ C + p_{ji} > d_i$                          // Check if job $j$ is tardy.
                        $L(minNumOfTardyJobs, i + 1, C + p_{ji}, tardyJobs + 1, G, \vec{S})$;
                  $else$                             // Job $j$ is not tardy.
                        $L(minNumOfTardyJobs, i + 1, C + p_{ji}, tardyJobs, G, \vec{S})$;
            $return$;
```

Special Case 1: Position-independent processing times ($1|gdd|\sum U_r$).

The special case of position-independent processing times (i.e., the case that $p_{jr} = p_j; j, r = 1, \ldots, n$) was shown to be solved by the SPT (Shortest Processing Time first) policy; see Hall [1].

Special Case 2: A common due-date ($1\left|p_{jr}, d_j = d\right|\sum U_r$).

A special case of general learning curves with a common due-date was studied by Mosheiov and Sidney [35], who proved that an optimal solution is obtained in polynomial time. A similar idea can be used for solving the setting of general position-dependent processing times.

We start by fixing a number $k\,(1 \leq k \leq n)$. Then we solve the problem of minimizing the makespan of k jobs (out of the original set of n jobs). This problem can be formulated as a Linear Assignment Problem (LAP), where n jobs need to assigned to k positions. Thus, the input matrix is of size $n \times k$, and each entry contains the processing time of job i if assigned to the j-th position ($p_{ij}, i = 1, \ldots, n, j = 1, \ldots, k$). The LAP is the following:

$$\text{MIN} \sum_{i=1}^{n} \sum_{j=1}^{k} X_{ij} p_{ij}$$

$$\text{S.T.}$$

$$\sum_{j=1}^{k} X_{ij} \le 1, i = 1, \dots, n$$

$$\sum_{i=1}^{n} X_{ij} = 1, j = 1, \dots, k$$

$$X_{ij} binary, i = 1, \dots, n, j = 1, \dots, k.$$

The result of this LAP, denoted $C_{max}(k)$, is the minimal makespan value of k jobs. This value is compared to d, and if $C_{max}(k) \le d$, then k jobs can be completed on time (leading to $n-k$ tardy jobs). This procedure is repeated for all k values, and the largest k for which $C_{max}(k) \le d$ is the optimal solution. Running time: Each LAP is solved in $O(n^3)$, the procedure is repeated $O(n)$, leading to total running time of $O(n^4)$.

5 Numerical study

We tested numericaly the performance of the exact algorithm F and the heuristic H. In all our numerical experiments, the job-position processing times were generated uniformly in the interval $[1, p_{max} = 100]$. The generalized due dates were generated uniformly in the interval $[1, d_{max} = \alpha P]$, where $P = np_{max}$ (n is the number of jobs), and α is the tightness factor. We assumed: $\alpha = 0.25, 0.5, \dots, 1.5$. For each

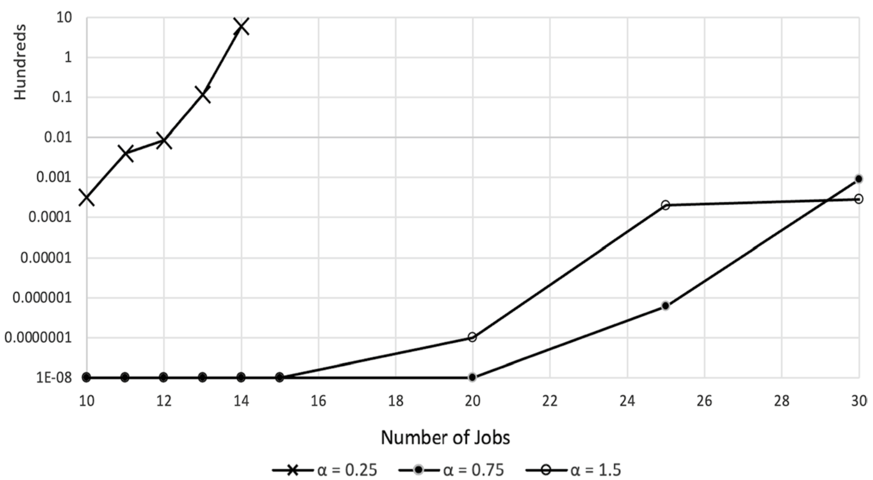

Fig. 2 Average running time of algorithm F as a function of the number of jobs for different tightness factors (logarithmic scale)

Table 1 Average and worst-case run times (sec) for Algorithm *F* with the smallest tightness factor ($\alpha = 0.25$)

Number of Jobs	$\alpha = 0.25$	
	Average Time	Worse Time
10	0.032	0.219
11	0.403	2.550
12	0.844	6.120
13	11.738	216.229
14	611.624	6705.725

combination of n and α, 10 instances were generated and solved. [All algorithms were coded in C, and executed on a Macintosh 2.7 GHz Intel Core i7 and 16 Gb RAM.]

We first evaluated Algorithm F assuming a small tightness factor: $\alpha = 0.25$. In this part of the numerical study, only small instances were solved (up to 14 jobs), and all problems were solved to optimality. Table 1 reports the average and worst-case running time for instances of $n = 10,11,\ldots,14$. Table 1 is limited to instances of this size because the running time for $n = 14$ increased significantly and reached an average of 611 s. Next, the running time of Algorithm F was measured for larger due-dates densities: $\alpha = 0.5, 0.75, \ldots, 1.5$. For these densities, we were able to solve instances of larger numbers of jobs: $n = 10,15,\ldots,30$. The running times, reported in Table 2, indicate that instances of up to 25 jobs were solved in few milliseconds. However, for $n = 30$ and $\alpha = 1$, a single instance was solved in more than 2.7 s. A logarithmic scale graph of the average running time as a function of the number of jobs for different tightness factors is provided in Fig. 2. In the last part of the numerical study, the proposed heuristic H was evaluated**.** The results of H were compared to those obtained by Algorithm F given an upper bound of 60 s on its running time. We report that H reached an optimal schedule for a given instance, if the solution of algorithm H is identical to that obtained by the modified algorithm F (in less than 60 s). As above, the tightness factors were: $\alpha = 0.5, 0.75, \ldots, 1.5$, and larger instances were solved: $n = 10,20,\ldots,50$. Again, for each combination of n and α, 10 problems were generated and solved. Table 3 and Fig. 3 provide the number of times that algorithm H reached an optimal schedule (as a function of the number of jobs and the tightness factor). Note that H performs well for small- and medium-size problems: for instances of up to 30 jobs, only in one case (out of 300), the optimum was not obtained by the heuristic. Also, for (relatively) large tightness factor of $\alpha = 1.5$, the heuristic missed the optimum in one case only (out of 250). Based on these promising results, we believe that Heuristic H can be used in most practical settings.

Table 2 Average and worst-case run times (sec) for Algorithm F with different tightness factors ($\alpha = 0.5, 0.75, 1, 1.25, 1.5$)

Number of Jobs	$\alpha = 0.5$		$\alpha = 0.75$		$\alpha = 1$		$\alpha = 1.25$		$\alpha = 1.5$	
	Average time	Worse time	Average time	Worse time	Average time	Worse time	Average time	Worse time	Average time	Worse time
10	0.000	0.009	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
15	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
20	0.001	0.013	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
25	0.000	0.002	0.000	0.002	0.000	0.000	0.000	0.002	0.029	0.860
30	0.000	0.004	0.089	2.679	0.004	0.107	0.000	0.004	0.000	0.000

Table 3 Number of times that algorithm H reached the optimum with different tightness factors ($\alpha = 0.5, 0.75, 1, 1.25, 1.5$)

Number of Jobs	Number of Optimal results				
	$T = 0.5$	$T = 0.75$	$T = 1$	$T = 1.25$	$T = 1.5$
10	10	10	10	10	10
20	10	10	10	10	10
30	10	9	10	10	10
40	10	9	10	7	10
50	6	7	7	9	9

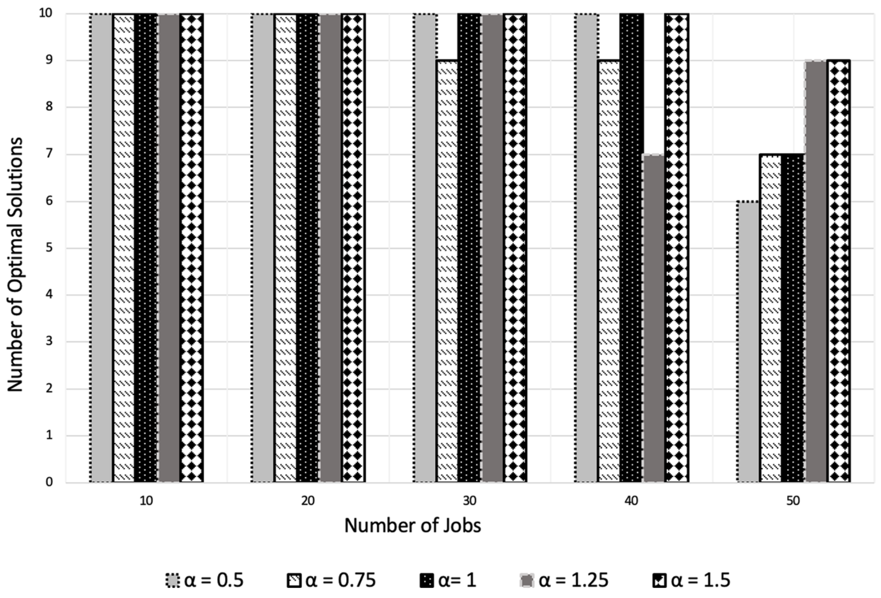

Fig. 3 Number of optimal schedules obtained by algorithm H as a function of the number of jobs for different tightness factors

6 Conclusion

We studied a single machine scheduling problem to minimize the number of tardy jobs. The special features considered are: generalized due-dates and general position-dependent job processing times. We first proved that the problem is NP-hard in the strong sense. Hence, a simple greedy heuristic and an exact algorithm were introduced. Both procedures were tested numerically. The exact algorithm was shown to be able to handle medium size instances, and the heuristic reached the optimum in the vast majority of (the larger) instances in our tests.

Since no studies considering both generalized due-dates and general position-dependent processing times have been published, future research may focus on other settings combining these two features. Challenging options are either other machine settings (multi-machines or shops), or other scheduling measures. In addition, we note that the complexity of the problem of minimizing the number of tardy jobs with general position-dependent processing times (and standard job-dependent due-dates) is still unknown, and is clearly another possible topic for future research.

Acknowledgements The second author was supported by The Deutsche Forschungsgemeinschaft (DFG, German Research Foundation—Grant Number 452470135).

Funding Open access funding provided by Hebrew University of Jerusalem.

Data availability All data used in our simulations was generated randomly according to the procedures described in the numerical study section and are available upon request.

References

1. Hall, N.G.: Scheduling problems with generalized due dates. IIE Trans. **18**, 220–222 (1986)
2. Hall, N.G., Sethi, S.P., Sriskandarajah, C.: On the complexity of generalized due date scheduling problems. Eur. J. Oper. Res.Oper. Res. **51**, 100–109 (1991)
3. Gerstl, E., Mosheiov, G.: Single machine scheduling problems with generalized due-dates and job-rejection. Int. J. Prod. Res. **55**, 3164–3172 (2017)
4. Choi, B.C., Park, M.J.: Just-in-time scheduling with generalized due dates and identical due date intervals. Asia-Pacific J. Oper. Res.Oper. Res. **35**, 1850046 (2018)
5. Park, M.J., Min, Y.H., Choi, B.C.: Two-agent scheduling with generalized due dates. Asian J Shipp Logist **34**, 345–350 (2018)
6. Choi, B.C., Min, Y., Park, M.J.: Strong NP-hardness of minimizing total deviation with generalized and periodic due dates. Oper. Res. Lett. Res. Lett. **47**, 433–437 (2019)
7. Gerstl, E., Mosheiov, G.: Single machine to maximize the number of on-time jobs with generalized due-dates. J. Sched. **23**, 289–299 (2020)

8. Choi, B. C., Park, M. J., and Min, Y.: A Just-in-time Scheduling Problem with Generalized Due Dates and Controllable Processing Times. 한국 *SCM* 학회지. **20**(1), 17–23 (2020)
9. Li, S.S., Chen, R.X.: Scheduling with common due date assignment to minimize generalized weighted earliness–tardiness penalties. Optim. Lett. **14**, 1681–1699 (2020)
10. Park, M.J., Choi, B.C., Min, Y., Kim, K.M.: Two-machine ordered flow shop scheduling with generalized due dates. Asia-Pacific J. Oper. Res.Oper. Res. **37**, 1950032 (2020)
11. Mor, B., Mosheiov, G., Shabtay, D.: Minimizing the total tardiness and job rejection cost in a proportionate flow shop with generalized due dates. J. Sched. **24**, 553–567 (2021)
12. Choi, B.C., Kim, K.M., Min, Y., Park, M.J.: Scheduling with generalized and periodic due dates under single-and two-machine environments. Optim. Lett. **16**, 623–633 (2022)
13. Mosheiov, G., Oron, D., Shabtay, D.: Minimizing total late work on a single machine with generalized due-dates. Eur. J. Oper. Res.Oper. Res. **293**, 837–846 (2021)
14 Mosheiov, G., Oron, D., Shabtay, D.: On the tractability of hard scheduling problems with generalized due-dates with respect to the number of different due-dates. J. Sched. **25**(5), 577–587 (2022). https://doi.org/10.1007/s10951-022-00743-9
15. Biskup, D.: Single-machine scheduling with learning considerations. Eur. J. Oper. Res.Oper. Res. **115**, 173–178 (1999)
16. Biskup, D.: A state-of-the-art review on scheduling with learning effects. Eur. J. Oper. Res.Oper. Res. **188**, 315–329 (2008)
17. Yin, Y., Wu, W.H., Cheng, T.C.E., Wu, C.C.: Due-date assignment and single-machine scheduling with generalised position-dependent deteriorating jobs and deteriorating multi-maintenance activities. Int. J. Prod. Res. **52**, 2311–2326 (2014)
18. Huang, X., Wang, J.J.: Machine scheduling problems with a position-dependent deterioration. Appl. Math. Model. **39**, 2897–2908 (2015)
19. Mosheiov, G.: Proportionate flowshops with general position-dependent processing times. Inf. Process. Lett. **111**, 174–177 (2011)
20. Gerstl, E., Mosheiov, G.: Scheduling on parallel identical machines with job-rejection and position-dependent processing times. Inf. Process. Lett. **112**, 743–747 (2012)
21. Yu, X., Zhang, Y., Huang, K.: Multi-machine scheduling with general position-based deterioration to minimize total load revisited. Inf. Process. Lett. **114**, 399–404 (2014)
22. Agnetis, A., Mosheiov, G.: Scheduling with job-rejection and position-dependent processing times on proportionate flowshops. Optim. Lett. **11**, 885–892 (2017)
23. Gerstl, E., Mor, B., Mosheiov, G.: Minmax scheduling with acceptable lead-times: extensions to position-dependent processing times, due-window and job rejection. Comput. Oper. Res. Oper. Res. **83**, 150–156 (2017)
24. Pei, J., Liu, X., Pardalos, P.M., Li, K., Fan, W., Migdalas, A.: Single-machine serial-batching scheduling with a machine availability constraint, position-dependent processing time, and time-dependent set-up time. Optim. Lett. **11**, 1257–1271 (2017)
25. Fiszman, S., Mosheiov, G.: Minimizing total load on a proportionate flowshop with position-dependent processing times and job-rejection. Inf. Process. Lett. **132**, 39–43 (2018)
26. Kovalyov, M.Y., Mosheiov, G., Šešok, D.: Comments on "proportionate flowshops with general position dependent processing times" [Inf. Process. Lett. 111 (2011) 174–177] and "minimizing total load on a proportionate flowshop with position-dependent processing times and job-rejection" [Inf. process. Lett. 132 (2018) 39–43]. Inf. Process. Lett. **147**, 1–2 (2019)
27. Yang, L., Lu, X.: Two-agent scheduling problems with the general position-dependent processing time. Theoret. Comput. Sci.. Comput. Sci. **796**, 90–98 (2019)
28 Mosheiov, G., Sarig, A., Strusevich, V.: Minmax scheduling and due-window assignment with position-dependent processing times and job rejection. 4OR **18**(4), 439–456 (2020). https://doi.org/10.1007/s10288-019-00418-w
29. Montoya-Torres, J. R., Botta-Genoulaz, V., Materzok, N., Gíslason, Þ. P. and Mendiela, S.: Modeling the parallel machine scheduling problem with worker-and position-dependent processing times. In: IFIP International Conference on Advances in Production Management Systems. pp. 351–359. Springer, Cham (2021)
30. Przybylski, B.: Parallel-machine scheduling of jobs with mixed job-, machine-and position-dependent processing times. J. Comb. Optim.Optim. **44**, 207–222 (2022)
31. Moore, J.M.: An n job, one machine sequencing algorithm for minimizing the number of late jobs. Manag. Sci. **15**, 102–109 (1968)

32. Ho, J.C., Chang, Y.L.: Minimizing the number of tardy jobs for m parallel machines. Eur. J. Oper. Res.Oper. Res. **84**, 343–355 (1995)
33. Lann, A., Mosheiov, G.: Single machine scheduling to minimize the number of early and tardy jobs. Comput. Oper. Res.. Oper. Res. **23**, 769–778 (1996)
34. Lodree, E., Jr., Jang, W., Klein, C.M.: A new rule for minimizing the number of tardy jobs in dynamic flow shops. Eur. J. Oper. Res.Oper. Res. **159**, 258–263 (2004)
35. Mosheiov, G., Sidney, J.B.: Note on scheduling with general learning curves to minimize the number of tardy jobs. J Operational Res Soc **56**, 110–112 (2005)
36. Mosheiov, G., Oron, D.: Minimizing the number of tardy jobs on a proportionate flowshop with general position-dependent processing times. Comput. Oper. Res.. Oper. Res. **39**, 1601–1604 (2012)
37. Adamu, M.O., Adewumi, A.O.: A survey of single machine scheduling to minimize weighted number of tardy jobs. J Ind manag optim **10**(1), 219–241 (2013)
38. Allahverdi, A., Aydilek, A., Aydilek, H.: Minimizing the number of tardy jobs on a two-stage assembly flowshop. J. Ind. Prod. Eng. **33**, 391–403 (2016)
39. Aydilek, A., Aydilek, H., Allahverdi, A.: Algorithms for minimizing the number of tardy jobs for reducing production cost with uncertain processing times. Appl. Math. Model. **45**, 982–996 (2017)
40. Mor, B., Mosheiov, G., Shapira, D.: Lot scheduling on a single machine to minimize the (weighted) number of tardy orders. Inf. Process. Lett. **164**, 106009 (2020)
41. He, R., Yuan, J., Ng, C.T., Cheng, T.C.E.: Two-agent preemptive Pareto-scheduling to minimize the number of tardy jobs and total late work. J. Comb. Optim.Optim. **41**, 504–525 (2021)
42. Hermelin, D., Karhi, S., Pinedo, M., Shabtay, D.: New algorithms for minimizing the weighted number of tardy jobs on a single machine. Ann. Oper. Res.Oper. Res. **298**, 271–287 (2021)