An exact algorithm for the bi-objective timing problem

Sophie Jacquin, Fanny Dufossé, Laetitia Jourdan

An exact algorithm for the bi-objective timing problem

Sophie Jacquin[1], **Fanny Dufossé**[1], **Laetitia Jourdan**[1]

Abstract The timing problem in the bi-objective just-in-time single-machine job-shop scheduling problem (JiT-JSP) is the task to schedule N jobs whose order is fixed, with each job incurring a linear earliness penalty for finishing ahead of its due date and a linear tardiness penalty for finishing after its due date. The goal is to minimize the earliness and tardiness simultaneously. We propose an exact greedy algorithm that finds the entire Pareto front in $O(N^2)$ time. This algorithm is asymptotically optimal.

Keywords Just-in-time scheduling · Bi-objective · Timing problem · Exact algorithm

1 Introduction

Just-in-time scheduling problems are studied by both researchers and practitioners for more than two decades. JiT principles improve the production process of a company under the condition of a good integration of customers and suppliers into the production process. Their purpose is to develop scheduling policies that minimize due date deviations of each job. Finishing a job before or after its due date involves direct and indirect costs [16]. Jobs finishing too early incur storage and insurance costs [19]. Likewise, a job finishing late can incur shortage costs ranging from back order and

✉ Laetitia Jourdan
laetitia.jourdan@univ-lille1.fr

Sophie Jacquin
sophie.jacquin@inria.fr

Fanny Dufossé
fanny.dufosse@inria.fr

[1] Inria Lille - Nord Europe, ORKAD, CRIStAL, University of Lille 1,
59655 Villeneuve d'Ascq, France

transportation expediting costs, to the loss of an important customer [10]. As underlined by Feldman and Biskup [7], the JiT scheduling problem has many practical applications. For instance, in the food industry, perishable goods have to be delivered to the customers shortly after they are produced. Similar constraints apply to chemical or physical mixtures where some ingredients have a short half-life period.

The JiT problem is usually modeled as a single objective optimization task that minimizes the weighted sum of total earliness and total tardiness of jobs. In most works on JiT scheduling a *branch an bound* ([8,12,16,17]) or meta-heuristic ([8,11,13]) is used to search though the jobs permutations while a *timing algorithm* is applied to compute the job execution times and thus the sequence cost [18].

In the last decades, there is a growing interest in the multi-objective version of the problem. Indeed, it could be difficult for the decision maker to design penalty values for both earliness and tardiness on the same scale. The tardiness penalty is often evaluated according to the risk of dissatisfaction of the customer, while the earliness penalty models storage costs. Therefore it is more convenient for the decision maker to evaluate the penalties of earliness and tardiness independently. This is exemplified by studies [2,20], where *just-in-time principles* are modeled by minimizing the number of tardy jobs and the weighted sum of total earliness costs. In this paper we choose to model the *just-in-time principles* by minimizing simultaneously the weighted sum of total earliness and the weighted sum of total tardiness. This choice was previously done by several authors [4,10,14,15] who propose solutions based on meta-heuristics. The advantage of this model is that it enables us to decide how important it is for a given job not to be tardy and how important it is not to be early.

In a previous work [10] we proposed a solution of the bi-objective just-in-time problem, where the order of jobs is fixed by a multi-objective evolutionary algorithm while the idle times are optimally determined by the exact Aneja–Nahir method [1]. We have demonstrated the benefits of solving the scheduling and timing problems separately, in a bi-objective context. Nevertheless, the efficiency of the method used to solve the timing problem has a major impact on this kind of approach. Therefore, in this paper, we are interested in the timing problem of the bi-objective version of the just-in-time single-machine job shop scheduling problem (JiT-JSP), where both earliness and tardiness are optimized as independent objectives. In the mono-objective version of the timing problem, there exists an $O(N \log N)$ algorithm to solve it [9], but to our knowledge there is no algorithm for the bi-objective version. The main contribution of this paper is a new exact algorithm to solve the timing problem. In Sect. 2, we define the bi-objective timing problem, its properties and introduce notations and definitions. In Sect. 3, we propose our algorithm to solve the bi-objective timing problem and prove that this algorithm is exact and asymptotically optimal. Section 4 gives the conclusion and perspective of this work.

2 The bi-objective timing problem

2.1 Problem formulation

In general JiT scheduling problems the order of jobs is not fixed. However, most methods used to solve these problems work by exploring large amounts of different

job permutations. For each fixed permutation the algorithm needs to find the best solution. This problem is called the *timing problem* and we can consider it to be a sub-problem of the general JiT scheduling problems.

The timing problem consists of scheduling the execution times of an *ordered* set of N jobs to optimize their total earliness E and total tardiness T. Each job J_i has a *processing time* p_i, a *release date* r_i, a *due date* d_i, and the *penalty factors* for earliness $\alpha_i \geq 0$ and for tardiness $\beta_i \geq 0$. A scheduling $s = (C_1, \ldots, C_N)$ is given by completion times C_i of each job.

Formally, the problem can be defined as follows:

$$E(s) = \sum_{i=1}^{N} \alpha_i \times \max(d_i - C_i, 0) \tag{1}$$

$$T(s) = \sum_{i=1}^{N} \beta_i \times \max(C_i - d_i, 0) \tag{2}$$

are to be *minimized*, subject to

$$C_i \geq C_{i-1} + p_i \quad i = 2, \ldots, N \tag{3}$$

$$C_i \geq r_i + p_i \quad i = 1, \ldots, N. \tag{4}$$

Formula for earliness (resp. tardiness) is formalized in Eq. 1 (resp. Eq. 2). Equation 3 prohibits simultaneous execution of two jobs, and Eq. 4 corresponds to the release date. If $S = (E(s), T(s))$ we write $s \mapsto S$ for short. We assume, without loss of generality, that for any job J_i, $r_i + p_i \leq d_i$.

In a given schedule, a job J_i is *early* (resp. *tardy*) if $C_i < d_i$ (resp. $C_i > d_i$) or *weakly early* (resp. *weakly tardy*) with $C_i \leq d_i$ (resp. $C_i \geq d_i$). A job J_i is *blocked by its release date* if $C_i = r_i + p_i$. In a solution, a *group* is a set of jobs that are run sequentially without idle times.

2.2 Problem properties

It is easy to see that the feasible solution set given by Eqs. 3 and 4 is convex and can be reduced to a compact set without excluding any Pareto optimal solution. Furthermore, the objective functions given by Eqs. 1 and 2 are a sum of convex linear functions, hence convex. Ehrgott et al. have demonstrated that these conditions are sufficient to prove that the Pareto front is connected and convex [5] (Theorems 3.38 and 4.1). Moreover, as the objective functions are piecewise linear, the timing problem's Pareto front is the union of finitely many semi-closed polyhedra [6]. So, in the objective space, the Pareto front of the timing problem is piecewise linear, as shown in Fig. 1.

We call each end of these linear segments an *extreme point* (including first and last points of the Pareto front). Thus, the whole Pareto front is completely determined just by the extreme points. In the following, S_j denotes the jth extreme point, sorted in ascending order by earliness. A schedule s_j which produces the jth extreme point

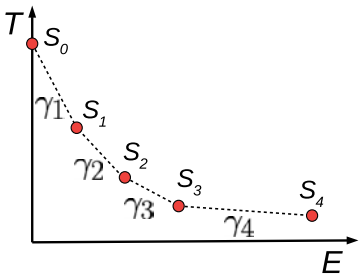

Fig. 1 Shape of the timing problem Pareto front in the objective space

Table 1 Table of notations for JiT-JSP

Notation	Definition
J_i	Job number i
r_i	Release time of job J_i
p_i	Processing time of job J_i
d_i	Deadline of job J_i
α_i	Earliness penalty of job J_i
β_i	Tardiness penalty of job J_i
C_i	Completion time of job J_i
s_j	jth extreme solution
S_j	jth extreme point
$E(s)$	Earliness of schedule s
$T(s)$	Tardiness of schedule s
γ_i	Slope of the ith line segment in the Pareto front

$s_j \mapsto S_j$ is called an *extreme solution*. There can be more than one extreme solution for each extreme point. The slope of the line segment $[S_j, S_{j+1}]$ is denoted by γ_j. These slopes are all strictly negative and form a strictly increasing sequence: $-\infty < \gamma_0 < \gamma_1 < \cdots < \gamma_N < 0$. The notation is summarized in Table 1.

The algorithm described in this paper is based on the following property:

Lemma 1 *Let S_j and S_{j+1} be two consecutive extreme points. Then for each solution $s = (C_1, \ldots, C_n) \mapsto S \in [S_j, S_{j+1}]$ there exists a solution $s' = (C'_1, \ldots, C'_n) \mapsto S_{j+1}$, such that $C'_i \leq C_i$ for $i = 1, \ldots, n$.*

In other words, we can get from any Pareto optimal solution to the next extreme solution just by *advancing* jobs. We *never* have to delay any jobs.

Proof The proof is by construction. Let us take an arbitrary $s = (C_1, \ldots, C_n) \mapsto S \in [S_j, S_{j+1}]$ and an arbitrary $s' = (C'_1, \ldots, C'_n) \mapsto S_{j+1}$. Since the choice of s' is arbitrary, there may exist some $C'_i > C_i$, i.e. some jobs have been *delayed*. Let us consider the solution that we get by simply *not* delaying the jobs $s'' = (\min(C_1, C'_1), \ldots, \min(C_n, C'_n))$. It is easy to verify that s'' satisfies Eqs. 3 and 4 and hence is a valid solution. We shall prove that $s'' \mapsto S_{j+1}$.

s' and s'' are different only because s' has a set of jobs D that have been delayed with respect to s''. Delaying these jobs has decreased earliness of s'' by $a \geq 0$ and increased its tardiness by $b \geq 0$:

$$E(s') = E(s'') - a$$
$$T(s') = T(s'') + b$$

We shall see that any other choice than $a = b = 0$ leads to a contradiction. First let us assume that $a = 0$ and $b > 0$. This means that s'' has the same earliness as s' but a smaller tardiness. This is not possible, because s' is Pareto optimal. Now assume that $a > 0$. This allows us to examine the trade-off ratio $-\frac{b}{a} = \frac{T(s')-T(s'')}{E(s')-E(s'')}$. That is the slope of the line segment between s' and s''. Surely $-\frac{b}{a} > \gamma_j$, because $s' \mapsto S_{j+1}$ which is an extreme point marking the end of the line segment with slope γ_j. But that would mean that by modifying s by delaying the jobs in D we get a solution $s''' = (\max(C_1, C'_1), \ldots, \max(C_n, C'_n))$ which is feasible, but which lies under the Pareto front. Hence $a = 0$ and $b = 0$, which means that $s'' \mapsto S_{j+1}$. □

The proof of Lemma 1 showed us that any jobs whose completion times were later in s_{j+1} than in s_j did not have any effect on either the earliness or the tardiness of s_j. This means that these jobs were either tardy with $\beta = 0$ or early with $\alpha = 0$. These jobs are a nuisance for us. They do not affect our objectives in any way, except indirectly, because they can block other more important jobs. It will help reduce our cognitive load by tidying them out of the way:

Definition 1 Let $s = (C_1, \ldots, C_N)$ be a valid solution. We call tardy jobs with $\beta = 0$ and weakly early jobs with $\alpha = 0$ *nuisance jobs*. We say that s is *normalized* if all nuisance jobs are either blocked by their release time or by the previous job.

In other words a normalized solution is the one where all the nuisance jobs have been advanced as much as possible.

3 Solving method

3.1 Motivation

Lemma 1 reveals how to discover the entire Pareto front of the timing problem. All we have to do is find the solution s_0 and then *advance* some jobs in this solution to get s_1 and so on, until we discover all the extremal solutions. The big question is "which jobs do we advance?"

The *greedy algorithm paradigm* tells us to make a move that is *locally optimal* at a given stage. In single-objective optimization this would mean a move that would yield the highest improvement in the value of the objective function. Since we have two objectives, the locally optimal step is to move a job that would give us the best *trade-off* between improving one objective and deteriorating the other. Since we are moving from a solution $s_j \mapsto S_j$ to a solution $s_{j+1} \mapsto S_{j+1}$, we are looking for the greatest possible reduction in tardiness in exchange for a unit of earliness.

Given a normalized Pareto optimal solution s, it is not possible to get to a different Pareto optimal solution just by advancing a single job. Suppose it were possible and there was such a job J. If J job is tardy it is either a nuisance job ($\beta = 0$) or a normal job ($\beta > 0$). However nuisance jobs cannot be advanced by definition and the existence of a normal job that can be advanced is in conflict with the assumption that s is Pareto optimal. Therefore in order to travel along the Pareto front we have to move blocks of consecutive jobs. Intuitively the early jobs at the beginning of the block making way for the tardy jobs at its end.

Definition 2 A moving block is a group of jobs $B = (J_i, \ldots, J_{i+k})$ preceded by idle time, where no job $J \in B$ is blocked by its release date.

Therefore a moving block is a set of jobs that can be simultaneously advanced to produce a valid solution. Advancing a moving block by some δt will increase the earliness and decrease tardiness of the solution. We can define a coefficient that will tell us what is the trade-off between the earliness gained and tardiness lost.

Definition 3 Let $B = (J_i, \ldots, J_{i+k})$ be a moving block where at least one job is weakly early with a positive earliness penalty. The coefficient of B, denoted by $c(B)$ is the ratio of tardiness coefficients of tardy jobs to the earliness coefficients of weakly early jobs:

$$c(B) = -\frac{\sum_{i \in B, C_i > d_i} \beta_i}{\sum_{i \in B, C_i \leq d_i} \alpha_i} \tag{5}$$

It is easy to see that in a normalized Pareto optimal solution each moving block contains at least one weakly early job with a positive penalty, therefore the coefficient given by Definition 3 is well defined for all moving blocks. In the following, we will explore how far can we advance a moving block.

Definition 4 Let $B = (J_i, \ldots, J_{i+k})$ be a moving block, preceded by an idle time of length d. We define:

$$a(B) = \min(d, d_i, \ldots, d_{i+k})$$

where

$$d_i = \begin{cases} C_i - p_i - r_i & \text{if } J_i \text{ is weakly early or } \alpha_i = 0 \text{ and } \beta_i = 0 \\ \min(C_i - p_i - r_i, C_i - d_i) & \text{otherwise} \end{cases}$$

We call $a(B)$ the *advance time* of B.

The following lemma shows that it makes sense to advance the moving blocks with the smallest possible coefficient by their advance time.

Lemma 2 *Let $s \mapsto S$ be a solution containing a moving block B with coefficient γ. Then by advancing B by $\epsilon \leq a(B)$ we obtain a valid solution $s' \mapsto S'$ such that the slope of line $[S, S']$ is γ. Furthermore, if B is still a moving block, then its coefficient has increased or remains the same.*

Proof The only tardy jobs that can become early are the ones with both $\alpha = 0$ and $\beta = 0$. These jobs can be ignored in the computation of earliness and tardiness change:

$$E(s) - E(s') = \sum_{i \in B, C_i \leq d_i} -\epsilon\alpha_i$$
$$T(s) - T(s') = \sum_{i \in B, C_i > d_i} \epsilon\beta_i$$

By dividing the two equations we get the slope of $[S, S']$ on the left side and $c(B)$ on the right side. It is possible that a tardy job with non-zero penalty factors has become weakly early in the process of advancing B. Substituting this into (5) shows that the coefficient of $c(B)$ has increased in this case. □

But can we travel along the Pareto front just by advancing a *single* block, or are there some situations where any infinitesimal advancement along the Pareto front would *require* the simultaneous advancing of several non-consecutive blocks? The following lemma tells us that the former is true. Furthermore, it gives us a hint at how to find such a block.

Lemma 3 *Let s be a Pareto optimal solution such that $s \mapsto S \in [S_j, S_{j+1}]$, $S \neq S_{j+1}$. Then in s there exists a moving block B with coefficient $c(B) = \gamma_j$.*

Proof From Lemma 1 we know that there exists a set of jobs in s that can be advanced to produce an extremal solution $s' \mapsto S_{j+1}$. Let us divide these jobs into groups $(G_1, \ldots, G_m)$. Two jobs belong to the same group if and only if they are consecutive both in s and s'. Let us suppose that if $i < j$ then all jobs in G_i run before jobs in G_j. It is impossible for any $c(G_k) < \gamma_j$. If such a group existed, let us choose k as the smallest index: $c(G_1) \geq \gamma_j, \ldots, c(G_{k-1}) \geq \gamma_j$. Since G_k is a group, it must be preceded by idle time either in s or in s'. If it were preceded by idle time in s, Lemma 2 tells us that we could produce a better than Pareto optimal solution just by advancing G_k. Clearly, G_k is preceded by idle time only in s'. But then there has to a group G_p, $p \in (1, \ldots, k-1)$ with $c(G_p) > \gamma_j$. Otherwise if all coefficients of these groups were equal to γ_j, then the average coefficient of the first k groups would be smaller than γ_j and by advancing *only* the first k groups by a sufficiently small period of time we would get a better than Pareto optimal solution. However, since G_k is preceded by idle time in s', it is possible to modify s' by delaying the groups $G_p, \ldots, G_{k-1}$ by a sufficiently small period of time to produce a valid solution. Since the average coefficient of these groups is greater than γ_j, delaying these groups would again produce a solution that is better than a Pareto optimal one. We now know that $c(G_1) \geq \gamma_j, \ldots, c(G_m) \geq \gamma_j$. But if any of these inequalities were strict, the average coefficient would be bigger than γ_j which would again produce a better than Pareto optimal solution. Therefore $c(G_i) = \gamma_j$ for $i = 1, \ldots, m$ and G_1 is the moving block, whose existence we were trying to prove. □

We now have all the parts needed to find the entire Pareto front. We just need to find the solution with the smallest earliness s_1 and then compute γ_1. Next we will advance all the moving blocks whose coefficient is γ_1. When there are no more such blocks,

we know we found s_2. Then we will determine γ_2 and advance blocks of jobs until we get to s_3 and so on, until we discover the entire Pareto front. The following section describes our ideas in more rigor.

3.2 Detailed description

The exact method to solve the bi-objective timing problem in JiT-JSP is described in Algorithm 1 and illustrated in Fig. 2.

The algorithm iteratively finds all normalized extreme solutions, sorted in ascending order of earliness. First, solution s_1 is found using the `find_min_earliness_solution` subroutine described in Algorithm 2. This solution is then fed into the so-called `pareto_glide` subroutine described in Algorithm 3 to produce the next normalized extreme solution s_2. s_2 is immediately used to produce s_3 and so on, until there are no more moving blocks to be advanced.

Algorithm 2 finds a solution with the smallest possible earliness. First it examines job J_1. If its earliness penalty $\alpha_i = 0$ it means that J_1 can be scheduled as early as possible in order to get out of the way of J_1. If $\alpha_i > 0$ we can schedule the job to finish exactly on its deadline, thus not causing any earliness penalty. Once J_1 is scheduled, we schedule J_2 in essentially the same way, except now we have to make sure that J_2 does not overlap with J_1. We continue until all jobs have been scheduled. Later we shall prove that the solution constructed by Algorithm 2 is a normalized extreme solution and that its earliness is zero.

The `pareto_slide` subroutine is built on ideas from Lemma 2. Its purpose is to produce solution s_{j+1} from s_j and, as the name suggests, it does so by sliding down the Pareto front. First, the slope of the Pareto front line segment (γ_j) is calculated as the smallest moving block coefficient in s_j. Then we iteratively seek and advance all

Algorithm 1: Solving method.

Require: problem parameters values
Ensure: normalized extreme solutions $s_j \mapsto S_j$.
  $j \leftarrow 1$
  $s_j \leftarrow$ find_min_earliness_solution() // Algorithm 2
  while there exists a moving block in s_j **do**
    $s_{j+1} \leftarrow$ pareto_glide(s_j) // Algorithm 3
    $j \leftarrow j + 1$
  end while

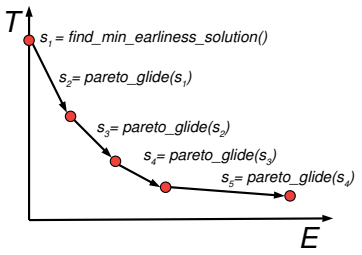

Fig. 2 Illustration of Algorithm 1

Algorithm 2: `find_min_earliness_solution`

Require: problem parameters values
Ensure: normalized extreme solution s_1
  if $\alpha_1 = 0$ **then**
    $C_1 \longleftarrow r_1 + p_1$
  else
    $C_1 \longleftarrow d_1$
  end if
  for $i = 2$ **to** N **do**
    if $\alpha_i = 0$ **then**
      $C_i \longleftarrow \max(r_i + p_i, C_{i-1} + p_i)$
    else
      $C_i \longleftarrow \max(r_i + p_i, C_{i-1} + p_i, d_i)$
    end if
  end for

Algorithm 3: `pareto_slide`

Require: problem parameters values, extreme point s_j in normalized form
Ensure: s_{j+1}
  $\gamma_j \leftarrow \min c(B)$ where B is a moving block in s_j
  $i \longleftarrow 1$
  while $i \leq N$ **do**
    Increment i until J_i is preceded by idle time.
    $i \longleftarrow k$
    while $B = (J_i, \ldots, J_k)$ is a moving block and $c(B) > \gamma_j$ **do**
      $k \longleftarrow k + 1$
    end while
    if $c(B) = \gamma_j$ **then**
      Advance B by $a(B)$.
      $k \longleftarrow k + 1$
      while $k \leq N$ and J_k is a nuisance job **do**
        Advance J_k until it is no longer a nuisance job, or until a constraint is met
        $k \longleftarrow k + 1$
      end while
      $i \longleftarrow k$
    end if
  end while

moving blocks with this coefficient. After advancing each block, we also advance any nuisance jobs that became unblocked.

3.3 Proof of correctness

First we have to prove that Algorithm 2 finds the solution with the smallest earliness.

Theorem 1 *The solution, s_1, computed by Algorithm 2 is the extreme point that minimizes the earliness criteria.*

Proof By construction, the earliness value of s_1 is 0 since the only early jobs are the ones whose earliness coefficient is 0. The only possibility to reduce the tardiness criteria is to advance the completion time of some tardy jobs. However, in s_1, each

tardy job is obstructed by the previous job or by its release date. So each tardy job is in a group that begins by a job J_i that is either completed in its due date and $\alpha_i > 0$, or blocked by its release date. So the tardiness criteria can not be reduced without increasing the earliness criteria for job J_i or violating a release constraint. So s_1 can not be Pareto dominated and thus is an extreme point. □

Theorem 2 *The solution, s_{j+1}, computed by Algorithm 3 is an extreme point. Furthermore, s_{j+1} is in normalized form.*

Proof By Lemma 2 we see that the intermediary solutions produced by advancing moving blocks with $c(B) = \gamma_j$ all lie on $[S_j, S_{j+1}]$. If the resulting solution would *not* be an extreme solution, then by Lemma 3 there would be a moving block with $c(B) = \gamma_j$. But this is not possible, since by Lemma 2 we know that if we advance B by $a(B)$, either it merges with a previous block, or its coefficient increases. Therefore when we advance B, we know that there are no moving blocks with coefficient γ_j ahead of B. This means that at the end of the loop there are no more moving blocks with $c(B) = \gamma_j$. □

3.4 Complexity

Building on our previous results, we are ready to prove the following:

Theorem 3 *The number of extreme points is linear in the number of jobs.*

Proof The number of extreme points is the number of iterations of `pareto_slide` plus one. In each such iteration at least one of these events will happen:

- M_1 : A tardy job becomes weakly early or a job gets blocked by release date.
- M_2 : A moving block merges with a previous block.

Event M_1 can happen only $2N$ times. Let us now examine M_2.

There are $N - 1$ possible gaps that can be closed by the merging of blocks. Once two blocks merge, they cannot split, unless a M_1 type event occurs in the given block. Indeed if B_k is the block resulting from advancing block B_j until it merges with B_i, then $c(B_j) \leq c(B_l)$ for all $B_l \subseteq B_i$ or $B_l \subset B_j$. Then it can be shown by simple arithmetic that for each block $B_l \subset B_k, c(B_k) \leq c(B_l)$. Therefore it can not be possible to create more than N new gaps which means that the total number of moves M_2 is bounded by $2N - 1$. Then the number of extreme points is bounded by $4N - 1$. □

Theorem 4 *The complexity of Algorithm 1 is $O(N^2)$ and this is asymptotically optimal.*

Proof Algorithm 1 invokes the `find_min_earliness_solution` subroutine only once and the `pareto_slide` at most $O(N)$ times. Since both these subroutines end in $O(N)$, the resulting complexity is $O(N^2)$. Since the length of the output is $O(N^2)$ (N completion times for each of the $O(N)$ extreme solutions), it is not possible to improve this result asymptotically. □

In comparison, the well known Aneja–Nair method [1] applied with the mono-objective timing method proposed by Bauman and Józefowska [3], which is to the best of our knowledge the most efficient method for this problem, has a complexity of $O(N^2 \log(N))$.

4 Conclusion

In this article, we proposed a new exact algorithm for the timing problem of the bi-objective version of the just-in-time single-machine jobshop scheduling problem (JiT-JSP). The next step is to incorporate this algorithm in a solving multi-objective method to improve the results on the bi-objective just-in-time single-machine jobshop scheduling problem as for example in [10].

Acknowledgements We acknowledge Dr. Martin Drozdik for proof reading the article and corrected mathematical proofs in the article during the revision phase.

References

1. Aneja, Y.P., Nair, K.P.: Bicriteria transportation problem. Manag. Sci. **25**(1), 73–78 (1979)
2. Azizoglu, M., Kondakci, S., Kksalan, M.: Single machine scheduling with maximum earliness and number tardy. Comput. Ind. Eng. **45**(2), 257–268 (2003)
3. Bauman, J., Józefowska, J.: Minimizing the earliness–tardiness costs on a single machine. Comput. Oper. Res. **33**(11), 3219–3230 (2006)
4. Dantas, J.D., Varela, L.R.: Scheduling single-machine problem based on just-in-time principles. In: 2014 Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC), pp. 164–169. IEEE (2014)
5. Ehrgott, M.: Multicriteria Optimization. Springer, Berlin (2006)
6. Fang, Y.P., Meng, K., Yang, X.Q.: Piecewise linear multicriteria programs: the continuous case and its discontinuous generalization. Oper. Res. **60**(2), 398–409 (2012)
7. Feldmann, M., Biskup, D.: Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. Comput. Ind. Eng. **44**(2), 307–323 (2003)
8. Hendel, Y., Runge, N., Sourd, F.: The one-machine just-in-time scheduling problem with preemption. Discrete Optim. **6**(1), 10–22 (2009)
9. Hendel, Y., Sourd, F.: An improved earliness–tardiness timing algorithm. Comput. Oper. Res. **34**(10), 2931–2938 (2007)
10. Jacquin, S., Allart, E., Dufossé, F., Jourdan, L.: Decoder-based evolutionary algorithm for bi-objective just-in-time single-machine job-shop. In: IEEE Symposium Series on Computational Intelligence, SSCI, pp. 1–8 (2016)
11. Liu, L., Zhou, H.: Hybridization of harmony search with variable neighborhood search for restrictive single-machine earliness/tardiness problem. Inf. Sci. **226**, 68–92 (2013)
12. Mahnam, M., Moslehi, G., Ghomi, S.M.T.F.: Single machine scheduling with unequal release times and idle insert for minimizing the sum of maximum earliness and tardiness. Math. Comput, Model. **57**(9), 2549–2563 (2013)
13. Qin, T., Peng, B., Benlic, U., Cheng, T., Wang, Y., Lü, Z.: Iterated local search based on multi-type perturbation for single-machine earliness/tardiness scheduling. COR **61**, 81–88 (2015)
14. Rahimi-Vahed, A., Dangchi, M., Rafiei, H., Salimi, E.: A novel hybrid multi-objective shuffled frog-leaping algorithm for a bi-criteria permutation flow shop scheduling problem. Int. J. Adv. Manuf. Technol. **41**(11–12), 1227–1239 (2009)
15. Rahimi-Vahed, A., Mirzaei, A.H.: Solving a bi-criteria permutation flow-shop problem using shuffled frog-leaping algorithm. Soft Comput. **12**(5), 435–452 (2008)
16. Sourd, F., Kedad-Sidhoum, S.: An efficient algorithm for the earliness–tardiness scheduling problem. Optim. Online 1205 (2005)
17. Tanaka, S., Fujikuma, S.: A dynamic-programming-based exact algorithm for general single-machine scheduling with machine idle time. J. Sched. **15**(3), 347–361 (2012)
18. Vidal, T., Crainic, T.G., Gendreau, M., Prins, C.: A unifying view on timing problems and algorithms. CIRRELT-2011-43, Montréal, QC, Canada (2011)
19. Vincent, T.: Multicriteria models for just-in-time scheduling. Int. J. Prod. Res. **49**(11), 3191–3209 (2011)
20. Wan, G., Yen, B.P.C.: Single machine scheduling to minimize total weighted earliness subject to minimal number of tardy jobs. Eur. J. Oper. Res. **195**(1), 89–97 (2009)