Single-machine batch delivery scheduling with job release dates, due windows and earliness, tardiness, holding and delivery costs

Fardin Ahmadizar*, Soma Farhadi
Department of Industrial Engineering, University of Kurdistan, Pudaran Boulevard, Sanandaj, Iran

A R T I C L E   I N F O
Available online 30 August 2014

Keywords: Scheduling Single-machine Batch delivery Release dates Due windows Dominance properties Imperialist competitive algorithm

A B S T R A C T
This paper deals with a single-machine scheduling problem in which jobs are released in different points in time but delivered to customers in batches. A due window is associated with each job. The objective is to schedule the jobs, to form them into batches and to decide the delivery date of each batch so as to minimize the sum of earliness, tardiness, holding, and delivery costs. A mathematical model of the problem is presented, and a set of dominance properties is established. To solve this NP-hard problem efficiently, a solution method is then proposed by incorporating the dominance properties with an imperialist competitive algorithm. Unforced idleness and forming discontinuous batches are allowed in the proposed algorithm. Moreover, the delivery date of a batch may be decided to be later than the completion time of the last job in the batch. Finally, computational experiments are conducted to evaluate the proposed model and solution procedure, and results are discussed.

© 2014 Elsevier Ltd. All rights reserved.

1. Introduction

Although single-machine scheduling problems have been broadly studied with different objective functions in past decades, most of these studies have been focused on the simple form of the problem and less attention has been paid to single-machine batch delivery scheduling problems. Batch delivery is a common characteristic of many real-life systems in which jobs are ultimately delivered to customers, leading to a decrease in the total delivery cost.

Batch delivery scheduling problems were first introduced by Cheng and Kahlbacher [1]. They have studied a single-machine batch delivery scheduling problem with the objective of minimizing the sum of the total weighted earliness and delivery costs, where the earliness of a job is defined as the difference between the delivery time of the batch it belongs and the job completion time. They have shown that the problem is NP-hard in the ordinary sense but polynomially solvable for equal weights. Hermann and Lee [2] have considered a single-machine scheduling problem where all the jobs have a given restrictive common due date and the objective is to minimize the sum of earliness and tardiness penalties and delivery costs of the tardy jobs; the authors have assumed that all early jobs are delivered in one batch at the due date without any cost, thus referring to holding costs as earliness penalties. They have shown that the problem is NP-hard, and proposed a pseudo-polynomial dynamic programming algorithm to solve it. Moreover, Cheng et al. [3] have studied single-machine scheduling with batch deliveries to minimize the sum of the batch delivery and earliness penalties, where the earliness of a job is defined as in [1]. They have formulated the problem as a classical parallel-machine scheduling problem; as a result, one can straightforwardly extend the complexity results as well as algorithms for the corresponding parallel-machine problem to this problem.

Wang and Cheng [4] have considered a parallel-machine batch delivery scheduling problem with the objective of minimizing the sum of flow times and delivery cost, where the delivery cost is a non-decreasing function of the number of deliveries, and shown that the problem is strongly NP-hard. They have developed a dynamic programming algorithm for the problem and two polynomial time algorithms for the special cases where the job assignment is predetermined or the job processing times are all identical. A similar problem with a single machine and unequal job weights has been investigated by Ji et al. [5]. They have shown that the problem of minimizing the sum of the total weighted flow time and delivery cost on a single machine is strongly NP-hard, and developed a dynamic programming algorithm for the problem and two polynomial time algorithms for the special cases where the jobs have a linear precedence constraint or the weights are agreeable with the processing times. Moreover, Mazdeh et al. [6] have developed a branch-and-bound algorithm for the same problem with the assumption that the delivery cost is a linear increasing function of the number of deliveries. For the special
case in which the maximum number of batches is fixed, they have shown, through experiments carried out with problems up to 20 jobs, that their algorithm is superior to that of Ji et al. [5] in terms of efficiency.

Hall and Potts [7] have provided dynamic programming algorithms for a variety of scheduling, batching and delivery problems arising in a supply chain, where the objective is to minimize the overall scheduling and delivery cost, using several classical scheduling objectives. Single-machine scheduling with batch deliveries and job release dates has been investigated by Mazdeh et al. [8]. They have allowed forming both continuous and discontinuous batches, and proposed a branch-and-bound algorithm to minimize the sum of flow times and delivery costs under the assumptions that the release dates are agreeable with the processing times and the jobs for each customer are to be processed in the shortest processing time order. The authors have shown, through experiments carried out with problems up to 40 jobs, that their algorithm is more efficient than the dynamic programming algorithm of Hall and Potts [7] for solving the problem. Hamidinia et al. [9] have introduced a novel complex single-machine batch delivery scheduling problem with the objective of minimizing the sum of earliness, tardiness, holding, and delivery costs. They have presented a mathematical model of the problem, and proposed a genetic algorithm-based heuristic to solve it.

Furthermore, in the literature there are some references where batch delivery scheduling and due date assignment problems are considered. Shabtay [10] has studied single-machine scheduling with batch deliveries and earliness, tardiness, holding, batching and due date assignment costs, where the due dates are controllable. The author has proved that the problem is NP-hard, and presented a polynomial time algorithm for the special cases where the job processing times are all identical or the acceptable lead times are all equal to zero and the holding penalty is less than the tardiness or due date assignment penalty. Yin et al. [11] have studied a single-machine batch delivery scheduling and common due date assignment problem with the option of performing a rate-modifying activity on the machine. Considering the objective of minimizing the sum of earliness, tardiness, holding, due date and delivery costs, they have presented polynomial time algorithms for the special cases where the jobs have equal modifying rates, the processing times are all identical or the jobs and modifying rates are agreeable. Recently, Yin et al. [12] have studied single-machine scheduling with batch delivery and an assignable common due window, where the objective is to minimize the sum of earliness, tardiness, holding, window location, window size, and delivery costs. They have shown that the problem is polynomially solvable by a dynamic programming algorithm under a reasonable assumption on the relationships among the cost parameters.

Although a variety of batch delivery scheduling problems has been investigated in the above mentioned papers, the researchers have worked with two most common assumptions as follows: (1) the delivery date of a batch is equal to the completion time of the last job in the batch, and (2) there is no capacity limitation on the size of a batch and the delivery cost is a non-decreasing function of the number of batch deliveries.

This study provides an extension to that of Hamidinia et al. [9] by taking into consideration job release dates and due windows. The problem considered here is a single-machine scheduling problem where jobs are released in different points in time but delivered in batches to their respective customers or to other machines for further processing. Each job must be delivered within its due window; otherwise, a penalty is incurred. It is assumed that the delivery cost for a batch, which is different for each customer, is independent of the number of jobs in the batch. The objective is then to schedule the jobs, to form them into batches and to decide the delivery date of each batch so as to minimize the sum of earliness, tardiness, holding, and delivery costs. The major limitations of the work of Hamidinia et al. [9] concern the following implicit assumptions made: (1) continuous batch forming, and (2) non-delay scheduling. The first assumption states that the jobs forming a batch are processed continuously, that is, no job belonging to another customer is processed between them. The second assumption implies that the machine is not kept idle while a job is waiting for processing, that is, unforced idleness is prohibited. Moreover, Hamidinia et al. [9] have simply assumed that the delivery date of a batch equals to the completion time of the last job in the batch. In this paper, however, we show that, even when all jobs are released at time zero, it may be advantageous to form discontinuous batches, to have periods of unforced idleness, and to deliver a batch at a time later than the completion time of the last job in the batch. Furthermore, Hamidinia et al. [9] have proposed a mathematical model of their problem, which is clearly a special case of the problem considered here, and claimed that it cannot be solved by off-the-shelf optimizers even for small test problems. Although they have argued that this failure is due to the problem complexity, we show that the erroneous formulation is the main reason. In addition to presenting an appropriate mathematical model, we establish a set of dominance properties and propose a solution method by incorporating them with an imperialist competitive algorithm (ICA). The proposed solution procedure schedules the jobs, forms them into batches and decides the delivery date of each batch.

The rest of the paper is organized as follows. In the next section, the problem is described and formulated. Section 3 is devoted to the derivation of some dominance properties. In Section 4, the proposed solution method is described, followed by Section 5 providing computational results. Finally, Section 6 gives a summary as well as future work.

2. Problem formulation

2.1. Problem description

The problem considered in this paper is a single-machine scheduling problem in which jobs are ultimately delivered to customers in batches, leading to decrease in the total delivery cost. There are $N$ jobs belonging to $F$ customers that have to be processed by the machine; each customer $j$ has $n_j$ jobs and $N = \sum_{j=1}^{F} n_j$. The machine is continuously available and can process at most one job at any point of time. Preemption is not allowed, and there is no setup time. Associated with each job $i$ belonging to customer $j$, which is denoted by job $(i,j)$, is a release date $r_{ij}$ at which it becomes available for processing on the machine, a processing time $p_{ij}$, and a due window $[d_{ij}^l, d_{ij}^u]$, where $d_{ij}^l$ and $d_{ij}^u$ are the earliest and latest due dates, respectively; the job is on-time and thus no penalty is incurred if delivered to the customer within the due window, while it will incur an earliness penalty if delivered before $d_{ij}^l$ and a tardiness penalty if delivered after $d_{ij}^u$. Distinct due windows represent a generalization of distinct due dates and are relevant in many practical situations because of uncertainty and tolerance [13]. Moreover, associated with each job $(i,j)$ is also a unit tardiness cost $\beta_{ij}$, a unit earliness cost $\alpha_{ij}$, and a unit holding cost $h_{ij}$; the job will incur a holding penalty if not delivered to the customer at its completion time. The cost of delivering each batch to customer $j$ is denoted by $D_j$. It is assumed that this cost is independent of the number of jobs in the batch and that there is no capacity limit on each batch delivery. Although this assumption may appear restrictive, it may be reasonable under the situations where all the jobs of each customer can be delivered by one truck at a shipment, and consequently, regardless of whether a truck is fully loaded or not, the delivery cost remains
unchanged; it clearly depends on the customer site. So, the jobs of each customer may be delivered in one or more batches; each job must clearly wait for all other jobs belonging to the same batch to be completed. The total delivery cost is then a linear increasing function of the number of batch deliveries.

The objective of the problem investigated is to schedule the jobs, to form them into batches and to decide the delivery date of each batch so as to minimize the sum of earliness, tardiness, holding, and delivery costs. Several special cases of this problem, such as the single-machine total weighted tardiness scheduling problem [14], have been proved to be strongly NP-hard. Hence, the batch delivery scheduling problem under consideration, which is significantly more complicated, is strongly NP-hard too.

2.2. Mathematical model

A special case of the problem in which, for each job \((i, j), r_{ij} = 0\) and \(d_{ij} = d_{ij}^{0}\) has been investigated by Hamidinia et al. [9]. They have proposed a mathematical model and claimed that, due to the problem complexity, current applications such as Lingo are unable to implement it even for simple test problems. As shown in the next section, the work of Hamidinia et al. [9] has limitations in that unforced idleness and forming discontinuous batches are not allowed and the delivery of a batch is forbidden to occur at a time later than the completion time of the last job in the batch. However, even regardless of these limitations, we found that their mathematical model suffers from serious bugs as follows: there is no guarantee of assigning each part inside a batch to at most one job; the third constraints have to be revised such that without adding any auxiliary binary variables, each batch is assigned to at most one customer, and accordingly the last term in the objective function is redundant and should be omitted; and finally, instead of having for each part inside a batch the fourth and sixth types of constraints, the summations in these constraints have to be over all parts inside a batch as well. Therefore, in the following an appropriate mathematical model is presented which, in addition to overcoming the above-mentioned limitations, can be solved by off-the-shelf optimizers.

To formulate the problem under consideration, the following decision variables are introduced:

- \(C_{ij}\): completion time of job \((i, j)\)
- \(R_{ij}\): delivery date of job \((i, j)\)
- \(R_{ij}^{b}\): delivery date of the \(b\)th batch of customer \(j\) \((1 \leq b \leq n_j)\)
- \(x_{ijb}\): 1, if job \((i, j)\) is in the \(b\)th batch of customer \(j\); 0, otherwise
- \(y_{ij}\): 1, if the \(b\)th batch of customer \(j\) contains at least one job; 0, otherwise
- \(z_{ijf}\): 1, if job \((i, j)\) is scheduled before job \((i', j')\); 0, otherwise

Let \(M\) be a big number. The problem can then be formulated as the following mathematical model:

\[
\begin{align*}
\min \sum_j \sum_i \alpha_{ij} \max\{0, R_{ij} - d_{ij}^{0}\} &+ \sum_j \sum_i \beta_{ij} \max\{0, d_{ij}^{0} - R_{ij}\} \\
&+ \sum_i \sum_j h_{ij}(R_{ij} - C_{ij}) + \sum_b \sum_y y_{ib} D_{ib}
\end{align*}
\]  
Subject to

\[
\begin{align*}
C_{ij} &\geq r_{ij} + p_{ij}, \quad j = 1, \ldots, F, \quad i = 1, \ldots, n_j \\
R_{ij}^{b} &\geq R_{ij} + c_{ij} - M(1 - x_{ijb}), \quad j, j' = 1, \ldots, F, \quad i = 1, \ldots, n_j, \\
R_{ij}^{b} &\geq R_{ij} + Mz_{ijf}, \quad f = 1, \ldots, n_f, \quad (i, j) \neq (i', j') \\
\sum_i x_{ijb} &\geq 1, \quad j = 1, \ldots, F, \quad i = 1, \ldots, n_j
\end{align*}
\]

The objective function (1) minimizes the total cost which includes, respectively, the sum of tardiness costs, the sum of earliness costs, the sum of holding costs, and the sum of delivery costs. Constraints (2) ensure that the processing of each job cannot start before its release date. Constraints (3) make sure that the processing of each job cannot start before its release date. Constraints (4) ensure that each job is assigned only to one batch. As the initial number of batches of each customer \(j\) equals \(n_j\), some batches may remain empty, and accordingly, constraints (5) ensure that no empty batches contribute to the total delivery cost, that is, to the last term in the objective function. Constraints (6) make sure that each batch may only be delivered to the corresponding customer after the processing of the last job in the batch has been completed; that is, the batch may be held in the manufacturer’s store for some time before being delivered to its destination. Constraints (7) are used to calculate the delivery date of each job, which is clearly equal to the delivery date of the batch containing the job. Finally, constraints (8) specify that the decision variables \(x, y\) and \(z\) are binary.

As can be seen, constraints (2) and (3) are used to schedule the jobs, constraints (4) and (5) to form the jobs belonging to each customer \(j\) into one or at most \(n_j\) batches, and constraints (6) and (7) to decide the job and batch delivery dates.

The above mixed integer non-linear program can be simply converted to a mixed integer linear programming (MILP) model by linearizing the objective function as well as constraints (6) and (7). After linearization, the model would have \(10\sum_{i,j} n_{i,j}^2 + (3N^2 + 17N)/2\) constraints, \(2\sum_{i,j} n_{i,j}^2 + 5N\) continuous variables and \(\sum_{i,j} n_{i,j}^2 + (N^2 + N)/2\) binary variables. As a result, for every fixed number of jobs \(N\), the number of constraints and variables increases with increasing \(\sum_{i,j} n_{i,j}^2\).

It is easy to see that \(\sum_{i,j} n_{i,j}^2 < (\sum_{i,j} n_{i,j})^2 = N^2\) for all \(F > 1\) and \(\sum_{i,j} n_{i,j}^2 = N^2\) when \(F = 1\). Hence, if all the \(N\) jobs belong to just one customer, the resulting MILP model has the maximum number of constraints and variables and thus is harder to solve. Nevertheless, when the \(N\) jobs belong to two or more customers, the number of constraints and variables may even increase with increasing \(F\). For example, let us consider two cases, both with 10 jobs but with different number of customers: in the first case, there are two customers each with 5 jobs, while in the other case there are three customers with 7, 1, and 2 jobs; then, the model corresponding to the latter case clearly has more constraints and variables.

3. Dominance properties

In this section, three lemmas are first introduced and then a number of dominance properties providing necessary conditions for any given solution to be optimal are derived.

From the objective function stated in (1), we obtain immediately the following corollary.
Corollary 1. The objective function (1) is not necessarily non-decreasing in the job completion times and delivery dates.

Lemma 1. The optimal solution is not necessarily non-delay.

Proof. Given a solution that is non-delay, if any idle time is inserted before processing a job, the job completion time and possibly its delivery date increase; clearly, the completion time and delivery date of some other jobs may increase too. From Corollary 1, this may improve the solution, that is, it may be advantageous to keep the machine idle while a job is waiting for processing, and so, unforced idleness may occur in the optimal solution.

Lemma 2. In the optimal solution, the jobs forming a batch are not necessarily processed continuously.

Proof. Given a solution in which the jobs forming every batch are processed continuously, if a job belonging to a batch is processed later between the jobs belonging to one of the following batches (a batch of the same customer or another customer), the job completion time, its delivery date and the delivery date of all other jobs of the same batch increase; clearly, the completion time and delivery date of some other jobs may increase too. From Corollary 1, this may improve the solution and consequently, discontinuous batches may occur in the optimal solution.

Lemma 3. In the optimal solution, the delivery of a batch does not necessarily occur at the completion time of the last job in the batch.

Proof. Given a solution in which the delivery date of every batch is equal to the completion time of the last job in the batch, if the delivery of a batch is postponed, the delivery date of all its jobs clearly increase. From Corollary 1, this may improve the solution and consequently, according to the optimal solution, completed batches may be held in the manufacturer's store for a while before being delivered to their destinations.

It is obvious that the above lemmas are still valid for the special case in which \( r_j = 0 \) and \( d_j = d_j^u \) for each job \((i, j)\) (i.e., for the problem investigated by Hamidinia et al. [9]).

Property 1. Given a solution in which, after the processing of a job \((i, j)\), the machine is idle for a time period \( IT \), if \( R_j \geq C_j + IT \), where \( 0 < IT \leq IT^* \), the processing of job \((i, j)\) should be postponed to complete at \( C_j + IT^* \).

Proof. Suppose that \( S_1 \) denotes a solution in which, after the processing of job \((i, j)\), the machine is idle for a time period \( IT \) and \( R_j > C_j \). Let \( IT^* \) be the maximum period of time, but not longer than \( IT \), for which \( R_j \geq C_j + IT^* \). Assume that \( S_2 \) denotes the same solution in which the processing of job \((i, j)\) is postponed to complete at \( C_j + IT^* \); under \( S_2 \), the delivery date of job \((i, j)\) is still \( R_j \). Clearly, the two solutions differ only in the holding cost for job \((i, j)\); under \( S_1 \), this cost is equal to \( h_j \left( R_j - C_j \right) \), while under \( S_2 \) it is equal to \( h_j \left( R_j - C_j - IT^* \right) \). Thus, \( S_2 \) dominates \( S_1 \), and the proof is complete.

Since, with the exception of the last job in a batch, for the other jobs \((i, j)\) belonging to the batch we necessarily have \( R_j > C_j \), the following corollary is obtained immediately. (Recall that for the last job \((i, j)\) in a batch we may have \( R_j = C_j \).)

Corollary 2. In any optimal solution, the machine is not idle after the processing of a job, except for the last jobs in the batches.

Property 2. Given a solution in which a job \((i, j)\) is processed immediately after another job \((i, k)\) belonging to the same batch, if \( r_{ij} \leq C_{ij} - p_{ij} \) and \( h_j p_{ij} > h_k p_{kj} \), the two jobs should be interchanged.

Proof. Suppose that \( S_1 \) denotes a solution in which two adjacent jobs \((i, j)\) and \((i, j)\), with \((i, j)\) following \((i, j)\), belong to the same batch; from Property 1, the machine is not idle between the processing of these two jobs. Assume that we have \( r_{ij} \leq C_{ij} - p_{ij} \) (i.e., when the processing of job \((i, j)\) starts, job \((i, j)\) is available) and \( h_j p_{ij} > h_k p_{kj} \). Now, construct a new solution \( S_2 \) in which jobs \((i, j)\) and \((i, j)\) are interchanged and all other jobs finish at the same time as in \( S_1 \); under \( S_2 \), the delivery date of these two jobs is still \( R_j = R_j \), but their completion times change. Clearly, the two solutions differ only in the holding cost for jobs \((i', j)\) and \((i, j)\). Let \( ST \) be the point in time at which job \((i', j)\) begins in \( S_1 \) and at which job \((i, j)\) begins in \( S_2 \). In addition, let \( H(S_1) \) and \( H(S_2) \) denote the holding costs for jobs \((i', j)\) and \((i, j)\) in \( S_1 \) and \( S_2 \), respectively. Then:

\[
H(S_1) = h_j \left( R_{ij} - ST - p_{ij} \right) + h_j \left( R_j - ST - p_{ij} - p_{ij} \right),
\]

\[
H(S_2) = h_j \left( R_j - ST - p_{ij} - p_{ij} \right) + h_j \left( R_j - ST - p_{ij} - p_{ij} \right).
\]

After taking the difference of (9) and (10), it is obtained that

\[
H(S_1) - H(S_2) = h_j p_{ij} - h_j p_{ij} > 0.
\]

Thus, \( S_2 \) dominates \( S_1 \), and the proof is complete.

Property 3. Given a solution in which a job \((i, j)\) belonging to a batch \( b \) of customer \( j \) is early, if for another batch \( b' \) of the same customer \( d_j'^u < R_{ij} < d_j'^u \) and \( h_j' \left( d_j'^u - R_{ij} \right) > h_j \left( R_{ij} - R_{ij} \right) \), job \((i, j)\) should be included in batch \( b' \).

Proof. Suppose that \( S_1 \) denotes a solution in which job \((i, j)\) belonging to batch \( b \) of customer \( j \) is early, i.e., \( R_j = R_{ij} < d_j'^u \). Assume that there is another batch \( b' \) of the same customer such that \( d_j'^u = R_{ij} < d_j'^u \) (so, batch \( b' \) is delivered after batch \( b \)) and \( h_j' \left( d_j'^u - R_{ij} \right) > h_j \left( R_{ij} - R_{ij} \right) \). Now, construct a new solution \( S_2 \) in which job \((i, j)\) is included in batch \( b' \); under \( S_2 \), the job is on time and its delivery date is now equal to \( R_{ij} \). Clearly, the two solutions differ only in the earliness and holding costs for job \((i, j)\). Denoting by \( EH(S_1) \) and \( EH(S_2) \) these costs in \( S_1 \) and \( S_2 \), respectively, we have

\[
EH(S_1) = \beta_j \left( d_j'^u - R_{ij} \right) + h_j \left( R_{ij} - R_{ij} \right),
\]

\[
EH(S_2) = h_j \left( R_j - R_{ij} \right).
\]

After taking the difference of (11) and (12), it is obtained that

\[
EH(S_1) - EH(S_2) = \beta_j \left( d_j'^u - R_{ij} \right) - h_j \left( R_{ij} - R_{ij} \right) > 0.
\]

Thus, \( S_2 \) dominates \( S_1 \), and the proof is complete.

Property 4. Given a solution in which a job \((i, j)\) belonging to a batch \( b \) of customer \( j \) is tardy, if for another batch \( b' \) of the same customer \( R_j > R_{ij} \) and \( d_j'^u < R_{ij} < d_j'^u \), job \((i, j)\) should be included in batch \( b' \).

Proof. Consider a solution in which job \((i, j)\) belonging to batch \( b \) of customer \( j \) is tardy, i.e., \( R_j = R_{ij} > d_j'^u \). Assume that there is another batch \( b' \) of the same customer such that \( R_j > R_{ij} \) and \( d_j'^u < R_{ij} < d_j'^u \) (so, batch \( b' \) is delivered before batch \( b \)). Clearly, if job \((i, j)\) is included in batch \( b' \) to be delivered earlier, not only the job is on time, but also its corresponding holding cost decreases.

Property 5. Given a solution in which every job \((i, j)\) in a batch \( b \) of customer \( j \) is early or on time, if \( \sum_{i \in b} \beta_j \geq \sum_{i \in b} h_j \), where \( a_i^b \) is the set of all jobs in batch \( b \) and \( a_i^b \) is the set of its early jobs, the delivery of batch \( b \) should be postponed until \( \min \{ d_j'^u \} \).
Proof. Suppose that \( S_1 \) denotes a solution in which every job \((i,j)\) in the \( b \)-th batch of customer \( j \) is early or on time, i.e., \( R_{ij} - T_j \leq d_{ij} \), and so, \( R_{ij} \leq \min\{d_{ij}\} \). Assume that we have \( \sum_{i \in q} \beta_{ij} > \sum_{i \in q} h_{ij} \) and \( R_{ij} < \min\{d_{ij}\} \). Now, construct a new solution \( S_2 \) in which the delivery of batch \( b \) is postponed until \( R_{ij} + T_j \), where \( TP \) is the time period for which \( R_{ij} + T_j = \min\{d_{ij}\} \); under \( S_2 \), every job \((i,j)\) in the batch is still early or on time and its delivery date is now equal to \( R_{ij} + T_j \). Clearly, the two solutions differ only in the earliness and holding costs for the jobs belonging to batch \( b \). Let \( EH(S_1) \) and \( EH(S_2) \) denote these costs in \( S_1 \) and \( S_2 \), respectively. Then,

\[
EH(S_1) = \sum_{i \in q} \beta_{ij} (d_{ij} - R_{ij}) + \sum_{i \in q} h_{ij} (R_{ij} - C_j),
\]

(13)

\[
EH(S_2) = \sum_{i \in q} \beta_{ij} (d_{ij} - R_{ij} - TP) + \sum_{i \in q} h_{ij} (R_{ij} + TP - C_j).
\]

(14)

In view of the fact that under \( S_2 \), the number of early jobs belonging to batch \( b \) may decrease, \( EH(S_2) \) may be even lesser than that stated in (14). After taking the difference of (13) and (14), it is obtained that

\[
EH(S_1) - EH(S_2) = \sum_{i \in q} \beta_{ij} TP - \sum_{i \in q} h_{ij} TP.
\]

Since \( TP > 0 \) and \( \sum_{i \in q} \beta_{ij} > \sum_{i \in q} h_{ij} \), \( S_2 \) dominates \( S_1 \), and the proof is complete.

Property 6. Given a solution in which every job in a batch is tardy or on time, the delivery of the batch should occur at the completion time of its last job.

Proof. Consider a solution in which every job in a batch is tardy or on time. Clearly, each job belonging to the batch incurs holding and/or tardiness costs, which increase if the job delivery date increases. Thus, the delivery of the batch should not be postponed.

4. Proposed solution procedure

In order to efficiently solve the problem under consideration, a hybrid algorithm is proposed by incorporating the dominance properties with an ICA. Such an integrating framework has previously been applied to solve other scheduling problems (see, e.g., [15–17]). ICA, firstly introduced by Atashpaz-Gargari and Lucas in 2007 [18], is a new population-based algorithm in the evolutionary computation field mimicking the human’s socio-political evolution. The ICA was originally developed for continuous optimization problems. However, researchers have recently applied ICAs to successfully solve a range of optimization problems (see, e.g., [19–25]).

As Atashpaz-Gargari and Lucas [18] introduced, ICAs start with an initial population composed of many candidate solutions to a specific problem, each of which is called a colony. Random generation or heuristic procedures can be used to initialize the population. Some of the most powerful colonies in the initial population are chosen to be the imperialists, while the rest of the colonies form the colonies of these imperialists. The colonies are then distributed among the imperialists proportionate to their power, where the power of a colony is determined on the basis of its objective value – the lower the country cost, the higher its power. An imperialist and its colonies form an empire whose total power will then be a function of both the power of the imperialist country and that of the colonies. Over time, as the search progresses, each imperialist tries to assimilate its colonies, modeled by moving all the colonies in the empire to the imperialist; thus forcing the search towards more promising areas in the search space. In addition, a social and political revolution in a country causes changes in the country characteristics, without being assimilated by an imperialist. In an ICA, the revolution devised as a diversity preservation procedure is modeled by randomly moving some of the colonies to produce new solutions. If in an empire a colony reaches a position better than the imperialist, their positions are exchanged and the algorithm will then continue by the imperialist in the new position (i.e., the colonies of this empire will then move to the new imperialist). Moreover, all the empires try to compete for taking possession of colonies from each other, modeled by picking some (usually one) of the weakest colonies of the weakest empire and making a competition among the empires to possess them. Through the imperialistic competition, powerless empires gradually lose their power and ultimately collapse. This process can be iterated until a situation with just one empire, hopefully the optimal solution, is reached. Finally, it is noted that an ICA makes use of many stochastic components, like any other metaheuristic.

In order to improve the ICA’s search capability, the derived dominance properties are employed as a local search strategy enhancing the quality of a solution. They have an important role in generating the initial population as well as in enhancing the quality of solutions generated by the ICA. The dominance properties, which are implemented in the same order as presented in Section 3, test whether the solution satisfies the necessary optimality conditions. If a given property is satisfied, the solution is properly modified. We can easily conclude, from a computational time point of view, that it is better to apply Property 1 in backward direction (i.e., by starting from the last job in the schedule and going backward to the first one). Moreover, if two adjacent jobs \((i,j)\) and \((i,j)\), with \((i,j)\) following \((i,j)\), are exchanged through Property 2 and the job immediately before \((i,j)\) in the new solution belongs to the same batch, exchanging these two jobs via the property is then considered. Implementing the dominance properties may result in a solution better than the original one, even though the resulting solution may not be optimal. The general structure of the proposed ICA, called ICADP, is then expressed as follows (the details are presented in the remainder of this section):

Step 1 (Initialization) Generate randomly a population of feasible solutions, improve them by implementing the dominance properties, and then, initialize the empires.

Step 2 (Assimilation) Move the colonies towards their relevant imperialists.

Step 3 (Revolution) Implement the revolution process.

Step 4 (Improvement) Implement the dominance properties. If a colony is better than its imperialist, exchange their positions.

Step 5 (Imperialistic competition) Make the competition among the empires.

Step 6 (Elimination) Eliminate any empire with no colony.

Step 7 (Termination) Repeat steps 2–6 until the termination criterion is satisfied.

4.1. Initialization

4.1.1. Solution representation

The solution representation mapping solution characteristics into a string of symbols is the first step in designing an ICA for a particular problem. It sets up a bridge between the original problem space and the space being searched by the algorithm. Defining an appropriate solution representation strategy is an important issue that affects the algorithm performance.

As mentioned earlier, a solution to the problem under consideration is completely characterized by a schedule of the jobs, a...
partition of them into batches, and a choice of the job (and batch) delivery dates. Accordingly, to describe and manipulate a solution, which in the ICA terminology is named as country, a matrix with 4 rows and \( N \) columns is employed in ICADP. Assume that all the jobs are numbered from 1 to \( N \), by considering firstly the jobs of customer 1, then those of customer 2, and so on. The first row of the matrix is then a permutation of the numbers from 1 to \( N \), thus indicating a processing order of the jobs. As unforced idleness of the machine is allowed, a job sequence alone cannot represent a schedule, and consequently, the second row of the matrix, a string of real numbers, is employed to determine the job completion times. Clearly, for a solution to be feasible, the schedule represented by the first two rows must satisfy constraints (2) and (3). It should be noted here that this type of schedule representation gives the algorithm the opportunity to efficiently search the solution space: the processing order of the jobs will change through changes in the entries in the first or even the second row, while it will also be possible to complete some jobs at other points in time, without changing the order of processing on the machine, through changes in the entries in the second row. Moreover, the third row of the matrix, a string of integers (with repetition allowed), is used to assign the jobs of each customer \( j \) to one or at most \( n_j \) batches so as to satisfy constraints (4) and (5). Finally, the last row, a string of real numbers, is employed to determine the job (and batch) delivery dates; for a solution to be feasible, the delivery schedule represented by this row must satisfy constraints (6) and (7). It is noted that every element of the matrix is then a permutation of the numbers from 1 to \( N \), which in the ICA terminology is named as country, a matrix with \( |\text{jobs\, of\, customer\, } j| \) rows and \( \min\{\text{cardinal number of jobs, } N\} \), by considering \( \max\{\text{due date of job, } DF_{ij}\} \) and \( \text{DF} \times \text{Cij} \), where \( \text{DF} \) is the completion time of the last job in the batch and \( \text{DF} \) a factor to allow a completed batch to be held in the manufacturer’s store for a while before being delivered to its destination; then, the delivery date of every job belonging to this batch is set equal to the generated delivery date. Obviously, the delivery schedule satisfies constraints (6) and (7).

4.1.3. Initialization of the empires

Let \( \text{Size}_{c} \) be the size of the initial population. A number, say \( \text{Size}_{c} \), of the most powerful countries (solutions) in the population are then selected as imperialists. The rest of the countries are the colonies (with size \( \text{Size}_{c} \)) of these imperialists and distributed among them based on their power. Let \( OV_{S} \) denote the objective value of a solution \( S \); that is, the sum of earliness, tardiness, holding, and delivery costs as stated in (1). To distribute colonies among imperialist, the normalized objective value of each imperialist \( imp \) is calculated as follows:

\[
OV'_{imp} = OV_{\text{max}} - OV_{imp}
\]

where \( OV_{\text{max}} \) is the maximum objective value among all imperialists. The normalized power of imperialist \( imp \) is then estimated by

\[
POW_{imp} = OV'_{imp} / \sum_{q=1}^{\text{size}} OV'_{q}
\]

Having the normalized power of all imperialists, the initial colonies are distributed among them proportionate to their normalized power as follows:

\[
\text{NC}_{imp} = \text{round}\left(POW_{imp}\text{Size}_{c}\right)
\]

A number \( \text{NC}_{imp} \) of the colonies are randomly chosen and assigned to empire \( imp \); since the equation \( \sum_{q=1}^{\text{size}} \text{NC}_{q} = \text{Size}_{c} \) has to be satisfied, the last empire of course takes possession of the remaining colonies. As a result, empires with greater power, i.e., with a lower objective function value, have a larger number of colonies.

4.1.2. Generation of the initial population

Each solution of the initial population is generated randomly, and then, improved by implementing the dominance properties. The following procedure is used to randomly produce a feasible solution. The entries in the first row of the given matrix are first selected by generating a random permutation of the numbers from 1 to \( N \). To choose the entries in the second row, starting from the first job in the first row, the completion time of each job \((i,j)\) is generated from a uniform distribution between \( \max\{FT, r_{ij}\} + p_{ij} \) and \( \text{UIF}(\max\{FT, r_{ij}\} + p_{ij}) \), where \( FT \) is the time at which the machine is free to start job \((i,j)\) and \( \text{UIF} \) a factor to allow unforced idleness (indeed, \( \max\{FT, r_{ij}\} \) is the earliest time at which job \((i,j)\) can start its processing). Evidently, the schedule represented by the first two rows then satisfies constraints (2) and (3). The entries in the third row are set by picking, for each job \((i,j)\), a random integer from 1 to \( n_{ij} \). Considering the decoding scheme as described in the last paragraph of the previous subsection, the assignment of the jobs to batches clearly satisfies constraints (4) and (5). Furthermore, to choose the entries in the last row, the delivery date of each batch \( b \) of customer \( j \) is first generated from a uniform distribution between \( C_{bj}^{\text{last}} \) and \( \text{DF} \times C_{bj}^{\text{last}} \) where \( C_{bj}^{\text{last}} \) is the completion time of the last job in the batch and \( \text{DF} \) a factor to allow a completed batch to be held in the manufacturer’s store for a while before being delivered to its destination; then, the delivery date of every job belonging to this batch is set equal to the generated delivery date. Obviously, the delivery schedule satisfies constraints (6) and (7).

\[
\begin{array}{cccccccc}
\text{Permutation of jobs} & 3 & 5 & 6 & 1 & 7 & 2 & 4 & 8 \\
\text{Job completion times} & \text{63} & \text{151.2} & \text{184} & \text{249} & \text{338.7} & \text{359} & \text{442} & \text{531} \\
\text{Batch indices} & \text{3} & \text{2} & \text{5} & \text{3} & \text{2} & \text{1} & \text{3} & \text{5} \\
\text{Job delivery dates} & \text{25.5.6} & \text{340} & \text{194} & \text{255.6} & \text{340} & \text{380} & \text{540.2} & \text{540.2} \\
\end{array}
\]

Fig. 1. An example of the proposed representation scheme.
4.2.1. Assimilation policy 1

If the first row, i.e., the job permutation, is chosen, the assimilating operator developed by Shokrollahpour et al. [20] for a job permutation is applied. One of the jobs in the imperialist array is randomly selected. That job is found in the colony’s array and shifted until reaches to the same position as in the imperialist array; the other jobs are shifted as well. Finally, the job in the right-hand side of the selected job in the imperialist array is put in the colony’s array at the same position by a swapping process. An example of this procedure is demonstrated in Fig. 2 using the problem instance given in the previous subsection; let job 7 be the job selected at random.

After applying the above operator, the entries in the other rows of the assimilated colony are reset. For every job, its batch index in the old colony is selected, while its corresponding completion time and delivery date may no longer be feasible. Therefore, considering the new job permutation, they are chosen in the same way as in generating the initial population; in addition to resulting in a feasible solution, this may allow unforced idleness as well as the delivery of some completed batches to be postponed.

4.2.2. Assimilation policy 2

If the second row, i.e., the job completion times, is chosen, an assimilating operator derived from the original ICA suggested for continuous optimization problems is applied. Let $C_{\text{imp}}$ and $C_{\text{col}}$ be the completion times of job $(i,j)$ in the imperialist and the colony, respectively. For every job $(i,j)$, a random number $\delta_{ij}$ is picked from a uniform distribution on $[0, \tau \times \text{dis}_{ij}]$, where $\text{dis}_{ij}$ is the distance between the colony and the imperialist along dimension $ij$, that is, $\text{dis}_{ij} = |C_{\text{imp}} - C_{\text{col}}|$, and $\tau$ is a parameter greater than 1 causing more areas around the imperialist to be searched. Then, the completion time of job $(i,j)$ in the assimilated colony is set to $C_{\text{col}} + \delta_{ij}$ if $C_{\text{imp}} - C_{\text{col}} \geq 0$, and $C_{\text{col}} - \delta_{ij}$ otherwise. An example of this procedure is demonstrated in Fig. 3 using a two-dimensional optimization problem.

After applying the above operator, the jobs in the first row of the assimilated colony are rearranged in the non-decreasing order of their new completion times, where ties are broken at random; clearly, the new job permutation may be the initial one. Nevertheless, the new job completion times are not necessarily feasible, and consequently, starting from the first job in the first row, if the completion time of a job is smaller than the earliest time at which the job can be completed, it is set to that time. For every job, its batch index in the old colony is selected, while its corresponding delivery date may no longer be feasible. In the case where the job permutation in the assimilated colony is the same as in the old colony, if the delivery date of a job is smaller than the completion time of the last job in the corresponding batch, it is changed to that time; otherwise, the (old) date is selected. However, in the other case in which the processing order of the jobs has changed through the changes in the entries in the second row, considering the new job permutation, the delivery dates are all chosen in the same way as in generating the initial population.

4.2.3. Assimilation policy 3

If the third row, i.e., the batch indices, is chosen, an assimilating operator is applied as follows. One of the customers is selected at random. Let customer $j$ be the customer selected; then, for each batch of customer $j$ in the imperialist, with probability $P_{\text{rev}}$, all the jobs belonging to that batch are reassigned in the colony to one batch – this can be simply achieved by setting the batch indices of these jobs to an unused integer between 1 and $n_j$. Applying this operator only affects the delivery dates of some or all of the jobs belonging to the selected customer, and so, the corresponding entries in the last row of the assimilated colony are reset; for each newly formed batch as well as each of the other batches whose last job has been removed, the job delivery dates are chosen in the same way as in generating the initial population.

It should be noted here that an assimilating operator may not be developed to efficiently apply to the last row of the colony, i.e., the job delivery dates, since the batches in the imperialist may differ considerably from those in the colony. However, as in all the three above cases changes are made in the entries in the last row of the colony, there may be no strong need for applying such an operator to the job delivery dates.

4.3. Revolution

In a country, a social and political revolution may take place causing sudden changes in its characteristics, without being assimilated by its relevant imperialist. In ICAs, the revolution is modeled by randomly moving some of the colonies to generate new solutions in areas of yet unvisited regions of the search space in order to keep the diversity of the population.

After the assimilation process is applied to a colony, the revolution takes place with probability $P_{\text{rev}}$ referred to as the revolution rate. To ensure that good solutions are not distorted too much, the revolution rate is set as a small number. The revolution operator always generating a feasible solution is then as follows. One of the rows of the matrix is selected at random, and then, one of the following policies is implemented depending on the row selected.

4.3.1. Revolution policy 1

If the first row is chosen, one of the jobs is randomly selected and inserted into another randomly chosen position in the sequence. Then, starting from the first job whose position has been changed, the job completion times and delivery dates are chosen in the same way as in generating the initial population; obviously, the delivery date of any other job in the first positions of the sequence may change as well.
4.3.2. Revolution policy 2

If the second row is chosen, an operator derived from the genetic algorithms suggested for continuous optimization problems is applied; an amount taken from a normal distribution with zero mean and standard deviation $\sigma_{CT}$ is added to the completion time of a job, where $\sigma_{CT}$ is a parameter controlling the amount of change. Then, the jobs in the first row are rearranged in the non-decreasing order of their new completion times, where ties are broken at random; clearly, the resulting permutation may be the initial one. As the new job completion times may not be feasible, starting from the first job in the sequence, if the completion time of a job is smaller than the earliest time at which the job can be completed, it is set to that time. Besides, the job delivery dates may no longer be feasible. In the case where the job permutation has not changed, if the delivery date of a job is smaller than the completion time of the last job in the corresponding batch, it is changed to that time. However, in the other case where the job permutation has changed, the delivery dates are all chosen in the same way as in generating the initial population.

4.3.3. Revolution policy 3

If the third row is chosen, one of the jobs is randomly selected; let job $(i,j)$ belonging to batch $b$ be the job selected. Then, the batch index of the job is changed to another randomly generated integer between 1 and $n_b$. If either the job alone forms a new batch or it is included in an existing batch as its last job, its delivery date is chosen in the same way as in generating the initial population; in the later case, the delivery date of any other job belonging to the same batch is then set to the delivery date of job $(i,j)$. In the other case in which it is included in an existing batch but not as its last job, its delivery date is clearly set equal to the delivery date of the last job in the batch. Moreover, in all the three mentioned cases, if job $(i,j)$ has previously been the last job in batch $b$, the delivery date of any other job belonging to batch $b$ is chosen in the same way as in generating the initial population.

4.3.4. Revolution policy 4

If the last row is chosen, an operator similar to that of revolution policy 2 is applied; an amount taken from a normal distribution with zero mean and standard deviation $\sigma_{DD}$ is added to the delivery date of a batch, where $\sigma_{DD}$ is a parameter controlling the amount of change. If the new delivery date of a batch is smaller than the completion time of its last job, it is set to that time. Then, the delivery date of every job is set equal to the delivery date of the batch containing that job.

4.4. Imperialistic competition

The empires try to compete for taking possession of colonies from each other and increase their power. The imperialistic competition gradually results in an increase in the power of more powerful empires and a decrease in the power of weaker ones. In other words, any empire that is not able to succeed cannot increase its power and will be gradually eliminated from the competition.

In our ICA, the imperialistic competition is implemented by picking the weakest colony of the weakest empire and making a competition among all the empires to possess it. To commence the competition, the total power of an empire is measured as a function of both the power of the imperialist and that of its colonies. This is modeled by defining the total cost of each empire $imp$ as follows:

$$TC_{imp} = OAV_{imp} + \gamma AOV_{imp}$$

where $OAV_{imp}$ is the average objective function value of the colonies of empire $imp$ and $\gamma$ is a parameter between 0 and 1 determining the effect of colonies. The normalized total cost of empire $imp$ is then calculated as below:

$$TC_{imp} = TC_{max} - TC_{imp}$$

where $TC_{max}$ is the maximum total cost among all empires, i.e., the total cost of the weakest empire. Now, the possession probability of each empire $imp$ is estimated by

$$POS_{imp} = \frac{TC_{imp}}{\sum_{q=1}^{size_{imp}} TC_q}$$

For each empire $imp$, a random number $rand_{imp}$ is then picked from a uniform distribution between 0 and 1, and the freed colony is finally assigned to the empire with the highest $POS_{imp} - rand_{imp}$. As a result, each empire having more total power has a higher chance to take possession of the weakest colony of the weakest empire.

4.5. Elimination

As time passes, the colonies of a powerless empire are divided among other empires, and accordingly, it gradually loses its power and ultimately collapses. Different criteria may be applied in modeling the collapse mechanism. In this research, however, an empire will be eliminated when it loses all of its colonies.

4.6. Termination

The termination condition in the original ICA is that when there exists only one empire, it stops. However, to ensure the algorithm terminates in a reasonable amount of time, ICAPD continues until only one empire survives or the time limit $Max\_Time$ is elapsed, depending on which criterion is met first.

5. Computational results

Computational experiments are conducted in order to assess the performance of the proposed mathematical model as well as the performance of the proposed ICA hybridized with the dominance properties, called ICAPD.

A total of 100 problem instances with sizes ranging from 7 to 70 jobs are randomly generated, where there are five instances for each problem size by considering 1–5 customers. For an instance with $N$ jobs and $F > 1$ customers, the jobs are randomly distributed among the customers in such a way that each customer has at least one job. The job processing times are uniformly sampled from the range $[1,100]$. Since it may be reasonable to assume that each job is released before its latest due date, the due windows and release dates are generated in the following way. Similarly as in [13], the center of the due window associated with each job $(i,j)$ is drawn from a uniform distribution between $(1 - TAR + RDD/2) \sum \sum p_j$ and $(1 - TAR + RDD/2) \sum \sum p_j$, where TAR and RDD are the tardiness factor and the relative range of the due window, respectively, and the window size is drawn from a uniform distribution between 1 and $\sum \sum p_j/N$. Then, the release date of the job is uniformly sampled from the range $[0,du]$. For each job, TAR and RDD take random values from 0.1, 0.2, 0.3 and 0.8, 1.0, 1.2, respectively. Moreover, the unit earliness and tardiness costs are chosen from a uniform distribution between 1 and 10, and the unit holding costs from a uniform distribution between 1 and 5. Finally, the cost of delivering each batch to a given customer is drawn from a uniform distribution over the range $[100,200]$.

ICAPD has been coded in Visual C# and run on a 2.27 GHz Pentium IV processor with 3 GB of RAM. In all experiments of this paper, we set the parameters $UIF$ and $DF$ used in the initial
population generation to be 1.1, the parameters $\alpha_{CT}$ and $\alpha_{OD}$ used respectively in revolution policies 2 and 4 to be 5, and the time limit $\text{Max\ Time}$ to be 600 s. However, the other numeric parameters of ICADP have been determined empirically; in a preliminary experiment, several values for each of them have been considered. The population size $\text{Size}_p$ has been tested on 50N, 75N, 100N, and 125N, and the initial number of imperialists $\text{Size}_i$ on 0.1 $\text{Size}_p$ and 0.2 $\text{Size}_p$. The parameter $\tau$ used in assimilation policy 2 has been tested on 1.5, 2.0, and 2.5, and $P_{bi}$ used in assimilation policy 3 on 0.05, 0.1, and 0.15. The revolution rate $P_{rev}$ has been tested between 0.05 and 0.25 in increments of 0.05. Moreover, the parameter $\gamma$ determining the effect of colonies in the imperialistic competition has been tested on 0.01, 0.05, 0.1, and 0.5. To make the performance of ICADP robust, each combination of the parameter values has been tested on 10 randomly chosen instances. Experimental results have then shown that the following values can obtain satisfactory results: $\text{Size}_p = 100N$, $\text{Size}_i = 0.1 \text{Size}_p$, $\tau = 2$, $P_{bi} = 0.05$, $P_{rev} = 0.1$, and $\gamma = 0.05$.

The performance of ICADP is compared with the performance of some alternative methods, including

- MILP: the MILP model implemented in CPLEX 11; with a time limit of 7200 s;
- RODP: the revolution operator together with the dominance properties as a heuristic algorithm;
- ICA1: the proposed ICA without the revolution operator and the dominance properties;
- ICA2: the proposed ICA without the dominance properties.

Like local search algorithms, RODP starts with a random initial solution (generated by the procedure stated in Section 4.1.2) and repeatedly tries to improve the current solution. At each iteration, a new solution is generated by applying the revolution operator to the current solution and then improving the resulting solution through the application of the dominance properties; the new solution is accepted only if it is better than the current solution. However, to avoid entrapment in local optima, following a preliminary experiment RODP restarts with a random solution if 3N successive iterations are performed without finding a solution better than the current one. To make a fair comparison, RODP terminates after the maximum time spent by ICADP is arrived.

The proposed assimilating operator is the only operator exploited by ICA1 to generate new solutions, and consequently, the contribution of this operator can be observed by comparing the performance of ICA1 with MILP and RODP. The contribution of the proposed revolution operator can be clearly observed by comparing the performance of ICA2 with ICA1. Moreover, a question that may arise is whether the proposed ICA does not perform better without the dominance properties derived but with generating an additional number of solutions. Accordingly, the performance comparison of the proposed ICA with (ICADP) and without (ICA2) the dominance properties can serve to validate their effectiveness and efficiency.

Each of ICADP, RODP, ICA1 and ICA2 has been run five times for each problem instance, and the minimum, average and maximum of the objective function values obtained have been provided. The results obtained for small and large instances are, respectively, shown in Tables 1 and 2, where the solution quality is measured by the mean percentage difference from the best found solution. In both tables the third column gives, for each problem instance, the number of jobs belonging to the customers in order. The average CPU times (in seconds) spent by ICADP, RODP, ICA1 and ICA2 are also provided in the tables. In Table 1, a value indicated in boldface corresponds to a solution whose optimality has been verified by CPLEX, and an asterisk in Table 2 indicates that CPLEX has failed to identify even a feasible solution within the time limit imposed. In these tables, if the time spent by CPLEX is less than 7200 s but its solution is not optimal, it implies that CPLEX has terminated due to memory limitations.

From Tables 1 and 2, it can be observed that CPLEX is capable of achieving optimal solutions for all of the instances with up to 10 jobs and for some instances with up to 13 jobs. However, as the number of jobs increases, it fails to find optimal solutions within a time limit of 7200 s; for 14 out of 25 instances with 50 or more jobs, CPLEX has failed to identify even a feasible solution within the imposed time limit. ICA1 which exploits only the proposed assimilating operator to produce new solutions is capable of achieving optimal solutions for 13 small instances. On the whole, the performance of ICA1 is comparable with MILP in the solution quality, although ICA1 is superior to MILP in terms of CPU time. When comparing ICA1 with RODP, ICA1 is capable of finding better solutions in almost the same CPU time. ICA2 which exploits both the proposed assimilating and revolution operators to produce new solutions is capable of achieving optimal solutions for 21 small instances. When comparing ICA2 with ICA1, ICA2 is capable of finding better solutions in almost the same CPU time. ICADP is capable of achieving optimal solutions for all of the instances with up to 10 jobs as well as for those instances with up to 13 jobs whose optimality has been verified by CPLEX. ICADP has a promising performance even for the instances with a large number of jobs; besides, the differences between its best and worst results are sufficiently low. When comparing ICADP with ICA2, they perform almost the same in terms of solution quality for the instances with up to 13 jobs. However, as the number of jobs increases, ICA2 is not capable of competing with ICADP; even the worst results achieved by ICADP are better than the best results of ICA2. Moreover, in terms of CPU time, except for four small instances, ICADP despite employing the dominance properties does not require more computational effort than ICA2. The reason is, of course, that due to the lack of the dominance properties which play an important role in the intensification of the search, ICA2 converges to a situation with just one empire by generating an additional number of solutions compared to ICADP. Accordingly, it is suggested to apply the dominance properties in order to achieve better solutions in relatively shorter computation times. On the whole, the computational results suggest that ICADP is fast and robust, and provides an efficient approach to achieve optimal or near-optimal solutions with small computational requirements.

Furthermore, another experiment is conducted on the two instances reported in details in [9], denoted by simple cases $a$ and $b$. In each of these two small instances, there are 2 customers each with 4 jobs in which $r_i = 0$ and $d^*_i = d_i^*$ for every job $(i,j)$. For each test problem, Hamidinia et al. [9] have executed their proposed genetic algorithm (GA) 20 times, and reported the average objective function value. Accordingly, each of ICADP, RODP, ICA1 and ICA2 has been run 20 times for each instance, and the average results obtained have been provided. Besides, for each instance we have executed the corrected version (a MILP model, denoted as MILP1) of the mathematical model of Hamidinia et al. [9] using CPLEX 11. Recall that unlike in MILP, unforced idleness and forming discontinuous batches are not allowed in MILP1, and also the delivery of a batch is forbidden to occur at a time later than the completion time of the last job in the batch. The results acquired are then summarized in Table 3; the average CPU times are in seconds, and the optimality of the solutions of MILP and MILP1 has been verified by CPLEX.

As can be seen from Table 3, only ICA1, ICA2 and ICADP are always able to find optimal solutions for both instances on every run. Although all the algorithms perform the same in terms of solution quality for test problem $b$, even RODP significantly outperforms GA in instance $a$. In terms of CPU time, however, a proper comparison is not possible since Hamidinia et al. [9] have not reported the computer platform of their experiments. Finally, by comparing MILP against MILP1, it is observed that in instance $a$ the total cost increases.
which jobs are released in different points in time but delivered in time later than the completion time of the last job in the batch.  

6. Summary and future work

This paper deals with a single-machine scheduling problem in which jobs are released in different points in time but delivered in batches to their respective customers or to other machines for further processing. Every job must be delivered within its due date; otherwise, a penalty is incurred. The delivery cost for a batch, which is different for each customer, is assumed to be independent of the number of jobs in the batch. The aim is to schedule the jobs, to form them into batches and to decide the delivery date of each batch so as to minimize the sum of earliness, tardiness, holding, and delivery costs. A mathematical formulation is developed, and then, it is shown that it may be advantageous to tardiness, holding, and delivery costs. A mathematical formulation is developed, and then, it is shown that it may be advantageous to tardiness, holding, and delivery costs. A mathematical formulation is developed, and then, it is shown that it may be advantageous to tardiness, holding, and delivery costs. A mathematical formulation is developed, and then, it is shown that it may be advantageous to tardiness, holding, and delivery costs. A mathematical formulation is developed, and then, it is shown that it may be advantageous.

### Table 1

<table>
<thead>
<tr>
<th>Instance</th>
<th>MILP</th>
<th>RODP</th>
<th>ICA1</th>
<th>ICA2</th>
<th>ICADP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>F</td>
<td>n's</td>
<td>Quality</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Time</td>
<td></td>
</tr>
<tr>
<td>1 (7)</td>
<td>0</td>
<td>105.3</td>
<td>1.72</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>2 (4.3)</td>
<td>0</td>
<td>3.52</td>
<td>2.05</td>
<td>2.77</td>
<td>4.59</td>
</tr>
<tr>
<td>3 (2,3)</td>
<td>0</td>
<td>0.50</td>
<td>0.14</td>
<td>0.44</td>
<td>0.64</td>
</tr>
<tr>
<td>4 (2,1,1)</td>
<td>0</td>
<td>10.22</td>
<td>0.80</td>
<td>1.41</td>
<td>1.64</td>
</tr>
<tr>
<td>5 (3,1,1,1)</td>
<td>0</td>
<td>0.08</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Average</td>
<td>0</td>
<td>69.92</td>
<td>0.80</td>
<td>1.13</td>
<td>1.58</td>
</tr>
</tbody>
</table>

15.32% if unforced idleness and forming discontinuous batches are not allowed and the delivery of a batch is forbidden to occur at a time later than the completion time of the last job in the batch.

6. Summary and future work

This paper deals with a single-machine scheduling problem in which jobs are released in different points in time but delivered in batches to their respective customers or to other machines for further processing. Every job must be delivered within its due window; otherwise, a penalty is incurred. The delivery cost for a batch, which is different for each customer, is assumed to be independent of the number of jobs in the batch. The aim is to schedule the jobs, to form them into batches and to decide the delivery date of each batch so as to minimize the sum of earliness, tardiness, holding, and delivery costs. A mathematical formulation is developed, and then, it is shown that it may be advantageous to
### Table 2

Results of MILP, RODP, ICA1, ICA2 and ICADP for large instances.

<table>
<thead>
<tr>
<th>Instance</th>
<th>MILP</th>
<th>RODP</th>
<th>ICA1</th>
<th>ICA2</th>
<th>ICADP</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ave</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ave</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Min</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ave</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

have periods of unforced idleness, to form discontinuous batches, and to deliver a batch at a time later than the completion time of the last job in the batch. Moreover, a number of dominance properties providing necessary conditions for any solution to be optimal are established. In order to efficiently solve this NP-hard problem, a hybrid algorithm is then proposed by integrating the dominance properties with an ICA utilizing both assimilating and revolution operators. The dominance properties, employed as a local search strategy, have an important role in generating the initial population as well as in enhancing the quality of solutions generated by the ICA. It is demonstrated, through computational experimentation, that the proposed hybrid algorithm is able to obtain optimal or near-optimal solutions within a reasonable amount of time.

A number of possible, significant extensions to this study could be investigated in future research. It would be interesting to consider the problem with sequence-dependent setup times (or costs). Since the jobs are released in different points in time, in such a situation two versions of the problem may then be investigated: setup can be assumed to begin either when both the corresponding job and the machine are available, or when the machine is available even when the job is not released. Another extension would be to consider the case in which there are precedence constraints between the jobs belonging to the same customer, or even between all jobs. Moreover, it would be interesting to study the problem where unavailability constraints are imposed on the machine, i.e., when it is not continuously available for processing, for example, due to preventive maintenance.

Acknowledgments

The authors would like to thank the two anonymous referees for their constructive comments and suggestions, which have been very helpful in improving the presentation of this paper.

Appendix A. Supporting information

Supporting information associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.cor.2014.08.012.

Table 3

<table>
<thead>
<tr>
<th>Instance</th>
<th>GA [9]</th>
<th>MILP,</th>
<th>MILP</th>
<th>RODP</th>
<th>ICA1</th>
<th>ICA2</th>
<th>ICADP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>Time</td>
<td>Cost</td>
<td>Time</td>
<td>Cost</td>
<td>Time</td>
<td>Cost</td>
</tr>
<tr>
<td>a</td>
<td>1007</td>
<td>1.05</td>
<td>971</td>
<td>842</td>
<td>889.5</td>
<td>6.23</td>
<td>842</td>
</tr>
<tr>
<td>b</td>
<td>1378</td>
<td>1.14</td>
<td>1378</td>
<td>1378</td>
<td>1.72</td>
<td>1.03</td>
<td>1378</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

References

[10] Shabtay D. Scheduling and due date assignment to minimize earliness, tardiness, holding, due date assignment and batch delivery costs. Int J Prod Econ 2010;123:335–42.