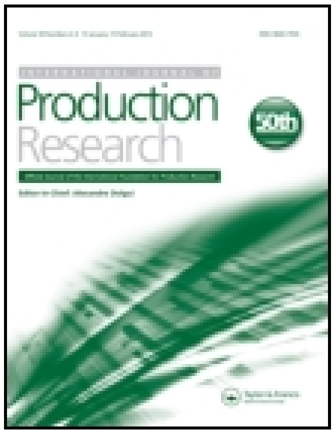

International Journal of Production Research
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/tprs20

Minimising total weighted earliness and tardiness on parallel machines using a hybrid heuristic

Rym M'Hallah [a] & Talal Al-Khamis [a]
[a] Department of Statistics and Operations Research , Kuwait University , P.O. Box 5969, Safat 13060 , Kuwait

Published online: 17 Aug 2011.

International Journal of Production Research
Vol. 50, No. 10, 15 May 2012, 2639–2664

Minimising total weighted earliness and tardiness on parallel machines using a hybrid heuristic

Rym M'Hallah and Talal Al-Khamis*

Department of Statistics and Operations Research, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait

(*Received 31 August 2010; final version received 2 February 2011*)

Scheduling jobs with different processing times and distinct known due dates on parallel machines (which may be idle) so as to minimise the total weighted earliness and tardiness is NP-hard. It occurs frequently in industrial settings with a just-in-time production environment. Here, small instances of this problem are tackled via an exact mixed-integer program, whereas larger instances are approximately solved using hybrid algorithms. The computational investigation discerns what makes a difficult instance, and demonstrates the efficiency of the approach.

Keywords: parallel machines; heuristics; genetic algorithms; scheduling; simulated annealing

1. Introduction

A major thrust of research in the scheduling field has been directed towards minimising both the tardiness and earliness penalties of scheduled jobs. This research direction is largely motivated by the adoption of the Just-In-Time (JIT) philosophy in manufacturing companies, where it is desirable to complete jobs exactly on their due dates. Baker and Scudder (1990) report that the current interest in JIT production discourages both earliness and tardiness. When completed early, a job causes an earliness penalty that reflects its holding or deterioration cost. On the other hand, when tardy, a job generates a tardiness cost that aggregates late charges, customer compensation, express delivery, lost sales or the loss of goodwill.

This paper focuses on the non-preemptive, deterministic parallel machine scheduling problem. A set $N=\{1,\dots,n\}$ of n independent jobs, available at time zero, is to be scheduled on m parallel machines. A job j is characterised by a known distinct deterministic positive integer due date d_j, a known distinct deterministic positive integer processing time p_j, a per unit cost of earliness α_j and of tardiness β_j. Job preemption is not allowed, but the machines may be kept idle. The objective is to identify a schedule of the n jobs so as to minimise the sum of weighted earliness and tardiness (WET) of the jobs, that is to minimise $\sum_{j=1}^{n}(\alpha_j E_j + \beta_j T_j)$, where E_j is the earliness of job j and T_j is its tardiness. The starting time S_j of job j, $j \in N$, is at least as large as the sum of the processing times of the subset of all jobs that precede it on the machine to which it is scheduled. In addition, its completion time C_j is the sum of its starting and its processing time: $C_j = S_j + p_j$. Job j is early if $C_j \le d_j$, and $E_j = \max\{0, d_j - C_j\}$. It is tardy if $C_j > d_j$, and $T_j = \max\{0, C_j - d_j\}$.

The parallel machine weighted earliness tardiness problem ($P|d_j|\sum\alpha_j E_j + \beta_j T_j$) is NP-hard. It is a generalisation of the parallel machine total earliness tardiness problem with a common due date (Cheng and Chen 1994), which is, in turn, an extension of the NP-hard single-machine case (Garey *et al*. 1988, Lee and Choi 1995, Lee and Kim 1995). It is also a generalisation of the NP-hard case with an unrestrictive (i.e. very large) common due date (Hall and Posner 1991), and of the NP-hard single-machine total tardiness problem (Lawler 1977). Its only known polynomially solvable case occurs when all processing times are equal and the due date is common (Cheng and Chen 1994).

$P|d_j|\sum\alpha_j E_j + \beta_j T_j$ is modelled here as a mixed-integer linear program (MIP). When solved with an off-the-shelf solver, this model yields the optimal solution for easy or small-sized instances. As expected, when instances become larger and more difficult, the commercial solver may encounter difficulties in reaching the optima or proving the optimality of the incumbent within a reasonable runtime. For these instances, applying metaheuristics is a viable alternative.

*Corresponding author. Email: alkhamis@ku.edu.kw

ISSN 0020–7543 print/ISSN 1366–588X online
© 2012 Taylor & Francis
http://dx.doi.org/10.1080/00207543.2011.578163
http://www.tandfonline.com

Metaheuristics can be grouped into two classes: neighbourhood-based such as steepest descent (SD) and simulated annealing (SA), and population-based such as genetic algorithms (GA). The first class performs a local search around a solution, whereas the second explores large neighbourhoods. These two tasks (exploitation and exploration) are commonly known as intensification and diversification. Recent research has witnessed an abundance of successful hybridised or cooperative algorithms that combine the benefits of intensification and diversification while avoiding the convergence problems encountered by their respective classes of algorithms. This paper follows this line of research and approximately solves large and/or difficult instances of $P|d_j|\sum \alpha_j E_j + \beta_j T_j$ using several algorithms that hybridise the GA with SD and SA.

The objective of this paper is fourfold. First, it gives a new exact mixed-integer program for the problem. Second, it proposes two constructive heuristics: CH1 assigns and schedules the jobs simultaneously, while CH2 hybridises a random search with integer programming. Third, it presents two neighbourhood-based metaheuristics (SD and SA), three implementations of a population-based metaheuristic (GA), and several newly designed approximate hybrid heuristics that make SD and SA collaborate with the GA via different levels and types of hybridisation and use CH1 and CH2 as decoding mechanisms. Fourth, it undertakes an extensive computational investigation to choose the best GA implementation, discerns the impact of the problem parameters and hybridisation techniques on the quality of the final solution, and assesses the overall performance of the proposed hybrid heuristic.

The paper is organised as follows. Section 2 surveys the state of the art of parallel (un)weighted earliness tardiness problems. Section 3 models the problem as a mixed-integer program (MIP) and proposes several upper bounds obtained via hybrid genetic algorithms. Section 4 presents the computational results. Finally, Section 5 summarises.

2. Literature review

$P|d_j|\sum \alpha_j E_j + \beta_j T_j$ is a generalisation of $1|d_j = d = \text{unrestrictive}|\sum E_j + T_j$. The latter was introduced by Kanet (1981). Its optimal schedule is V-shaped with one job completing at the due date d and one starting at d. All jobs scheduled prior to d are in LPT order while those scheduled after d are in SPT. The optimal schedule contains no inserted idle time between adjacent jobs. The first scheduled job may not start at time zero. When the due date is restrictive, the optimal schedule of the NP-complete problem (Hall *et al.* 1991) may contain a straddling job that starts before d and completes after d (Baker and Scudder 1990). When due dates are distinct, the optimal schedule may contain inserted idle times (Fry *et al.* 1987). In general, when idle times can be inserted, the problem is solved in two phases (Yano and Kim 1991, Davis and Kanet 1993, Kim and Yano 1994, Lee and Choi 1995, Chang 1999). The first phase identifies a good sequence of the jobs while the second phase inserts idle time between adjacent jobs to minimise the total (weighted) earliness and tardiness. Various ramifications of the two-phase approach have been investigated (Sourd and Kedad-Sidhoum 2003, Hendel and Sourd 2005, Sourd 2005).

Different extensions to the single-machine case have been considered. Their classification according to the number of machines, number of due dates, tightness of due dates, and types of penalties is given by Baker and Scudder (1990). A survey of parallel machine scheduling with common due dates and earliness tardiness penalties is given by Lauff and Werner (2004).

Cheng (1989) identifies both an optimal common due date d^* and an optimal job sequence for $P|d_j = d|\sum \alpha E_j + \beta T_j + \delta d$, where the job-independent non-negative constants α, β, and δ represent, respectively, the earliness, tardiness, and due-date assignment costs per unit time. He infers that d^* depends only implicitly on the job sequence when the number of machines $m = 1$, whereas it explicitly depends on the job sequence when $m > 1$. He exploits these results to design a simple heuristic for $m > 1$, and tests it on instances with up to 10 jobs. Biskup and Cheng (1999) show that $P|d_j = d, p_j = p|\sum \alpha E_j + \beta T_j + \delta d$ is polynomially solvable, and present a fast heuristic for the distinct processing time case. This heuristic ranks jobs in decreasing order of p_j and assigns the ranked jobs to the machine with the smallest workload. It then solves m independent single-machine problems to determine the starting time of the first job on each machine and the common due date. The heuristic has an average deviation of 4% from optimum for the tested small-sized problems ($n = 6, 8$, and $m = 2, 3$). Federgruen and Mosheiov (1996) consider $P|d_j = d|\sum f(E_j + T_j)$, where $f(E_j + T_j)$ is the cost of job $j, j \in N$. First, they address the case where $f(E_j + T_j)$ is a general non-decreasing convex function of $E_j + T_j$, d is unrestrictive, and $\alpha_j = \beta_j$. They derive a lower bound for $f(E_j + T_j)$, develop a simple 'alternating schedule' heuristic, and show that the resulting optimality gap is small for a class of instances. Second, they consider the case where $f(E_j + T_j)$ is a general asymmetric, not necessarily convex function of $E_j + T_j$, and d is restrictive. De *et al.* (1994) discern the cases when $P|d_j = d = \text{unrestrictive}|\sum w_j(E_j + T_j)$ is equivalent to its unweighted counterpart. They exploit this equivalence to prove that the two problems are NP-hard when $m = 2$ and become strongly NP-hard for $m > 2$. Subsequently, they design a dynamic programming algorithm that can be used in either case, and apply it to instances with up to 30 jobs and small integer processing times. Chen and Powell (1999) model $P|d_j = d = \text{unrestrictive}|\sum w_j(E_j + T_j)$ as a set partitioning integer program (with side constraints), and solve it using branch and bound. The bound of a node of the tree is the solution value of the corresponding linear relaxation of its set partitioning problem, whereas its value is obtained using column generation with the columns corresponding to single-machine schedules. Generating these columns is, in turn, NP-hard. It is undertaken via a pseudo-polynomial dynamic program. Monch and Unbehaun (2007) consider $P|d_j = d|\sum E_j + T_j$, where the machines are burn-in ovens. They propose three decomposition heuristics. The first applies the exact algorithm of Hall (1986) and Emmons (1987) to partition the jobs assigned to each of the parallel burn-in ovens into early and tardy job sets. The second is a genetic algorithm that assigns jobs to each single burn-in oven. The third assigns jobs to m early and m tardy job sets via a genetic algorithm. Once it has partitioned and assigned the jobs, each of the three heuristics applies sequencing rules and dynamic programming for each machine. Toksari and Guner (2009, 2010) study $P|d_j = d|\sum E_j + T_j$ under the effects of time-dependent learning and (non)linear deterioration, i.e. p_j is an increasing function of both S_j and its position in the sequence. They show that, under agreeable conditions, the optimal schedule is V-shaped, and present a mathematical model for small instances and a heuristic for large instances. Sun and Wang (2003) show that $P|d_j = d, w_j = p_j|\sum w_j(E_j + T_j)$ is NP-hard. They apply dynamic programming to solve very small instances with small integer processing times, and a heuristic based on the longest processing time first dispatching rule for other instances. They prove that the worst-case error bound of the heuristic is 25/12 when d is restrictive. Min and Cheng (2006) investigate $P|d_j = d|\sum E_j + T_j$, where d is *undetermined*. They design a hybridised GA where SA and iterative search are applied as fine-tuning operators of the GA. They test their hybrid algorithm on instances with up to 40 jobs and 10 machines. Solis and Sourd (2008) apply a local search on $P|d_j = d = \text{restrictive}|\sum \alpha_j E_j + \beta_j T_j$, and improve its solution via dynamic programming.

In contrast to the previous approaches, Mason *et al.* (2009) and Kedad-Sidhoum *et al.* (2008) consider the case where the due dates are distinct and known. Specifically, the former study the unweighed case (i.e. $\alpha_j = \beta_j = 1$), whereas the latter consider the general weighed case with the earliness and tardiness penalties not necessarily equal. In fact, Mason *et al.* (2009) extend the moving block heuristic (MBH) (Mason *et al.* 2005), initially designed for $1|d_j = d|\sum E_j + T_j$, to $P|d_j|\sum E_j + T_j$, and compare its performance with a bound obtained using a mixed-integer program. On the other hand, Kedad-Sidhoum *et al.* (2008) develop two types of lower bounds for $P|d_j|\sum \alpha_j E_j + \beta_j T_j$. The first extends existing assignment-based bounds, originally designed for the single-machine problem (Sourd and Kedad-Sidhoum 2003). The second is based on linear and Lagrangean relaxations of a time-indexed formulation with the relaxed problems solved via column generation. They assess the tightness of the lower bounds by comparing them with an upper bound based on a local search algorithm that mimics those implemented by Sourd (2005) and Solis and Sourd (2008). The search is initiated with the optimum of one of the relaxed problems or randomly. In the latter case, the algorithm partitions the jobs into m subsets, sequences them, and computes their optimal timing. In either case, the search manipulates feasible solutions via job swaps, extraction and reinsertion, and list crossover. They accelerate the search by implementing a dynamic programming (Sourd 2005) that stores partial sequences and reuses them when evaluating the neighbours' *WET*. Experimental tests of instances with up to 90 jobs and six machines yield a 1.5% average optimality gap.

Variants of the parallel machine weighed earliness and tardiness scheduling problems have been considered. Serifoglu and Ulusoy (1999) consider the case with sequence-dependent setup times, and tackle it using a GA. They further apply neighbourhood-search-based approaches to improve the GA's solution quality. Radhakrishnan and Ventura (2000) apply local search and SA to the same problem. Turkcan *et al.* (2009) investigate the online version of the problem where the processing times of the jobs are unknown variables determined during the optimisation process. They further modify the objective that minimises a convex function of processing times and *WET*. The problem is formulated as a time-indexed integer program with discrete processing time alternatives. A linear-relaxation-based algorithm assigns the jobs to the machines, and sequences them optimally. A nonlinear program finds the optimal starting and processing times of the jobs of each sequence. An approximation of the optimal solution of the nonlinear model is identified by converting it into a minimum-cost network flow model and piecewise linearising the manufacturing cost. When a stability measure is incorporated into the optimisation model, the method can be adapted to online scheduling (where events such as machine failure, arrival of new jobs, and shortage of raw material occur randomly).

This paper extends the work of Mason *et al.* (2009) since it considers the weighted case, that is it studies the same problem investigated by Kedad-Sidhoum *et al.* (2008). It presents a MIP for the weighted earliness tardiness problem (i.e. $P|d_j|\sum \alpha_j E_j + \beta_j T_j$), and proposes new upper bounds based on metaheuristics and their hybridisation. The MIP corrects the model of Mason *et al.* (2009) and extends it to the weighted case. It is precedence-based, in contrast to the time-indexed formulation of Kedad-Sidhoum *et al.* (2008). In fact, adopting a time-indexed formulation is useful when relaxation approaches are to be applied. However, when the sum of processing times is large, this approach might require a large number of variables, thus its application to directly solve the problem becomes questionable even for the single-machine case (Morton and Pentico 1993, Ventura and Kim 2003). Our preliminary computational investigation indicated that our commercial solver was not able to solve the time-indexed formulation of any of the tested small-sized instances.

3. Proposed approach

Valid upper bounds for $P|d_j|\sum \alpha_j E_j + \beta_j T_j$ are obtained using a new MIP, two new constructive heuristics, three metaheuristics, and their hybridisations.

3.1 *MIP-based upper bound*

This section discusses the limitations of the model of Mason *et al.* (2009), and presents a new modified model that overcomes them and solves $P|d_j|\sum \alpha_j E_j + \beta_j T_j$ exactly. Indeed, Mason *et al.* (2009) obtain a lower bound to $P|d_j|\sum E_j + T_j$ using a mixed-integer program whose corresponding schedule is not necessarily feasible, as elucidated by the next two examples.

Example 3.1: Consider the eight-job two-machine problem of Table 1. The model of Mason *et al.* (2009) obtains a lower bound equal to 4, with the starting and completion times given in Table 2, and the corresponding solution: $\chi_{15} = \chi_{16} = \chi_{17} = \chi_{18} = \chi_{23} = \chi_{24} = 1$, $\epsilon_1 = \epsilon_2 = 1$, where $\chi_{k\,j} = 1$ if job k precedes job j, and $\epsilon_j = 1$ if job j is the first job on the machine to which it is assigned. The corresponding Gantt diagram, displayed in Figure 1, shows that job 7

Table 1. Processing times and due dates for Example 3.1.

	j							
	1	2	3	4	5	6	7	8
p_j	80	25	45	60	65	30	60	10
d_j	165	105	145	205	255	365	80	380

Table 2. Starting and completion times obtained by the model of Mason *et al.* (2009) for Example 3.1.

	j							
	1	2	3	4	5	6	7	8
S_j	83	79	104	144	189	331	314	367
C_j	163	104	150	206	254	364	377	379

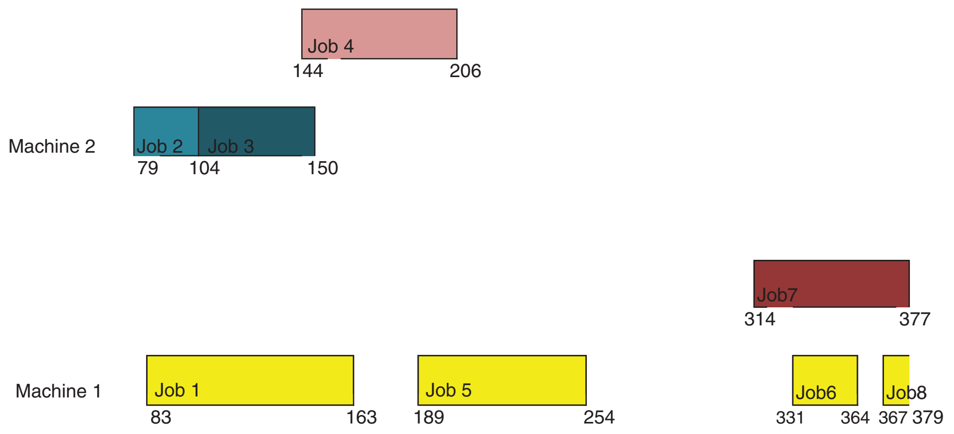

Figure 1. Infeasible schedule obtained by the model of Mason *et al.* (2009) for Example 3.1.

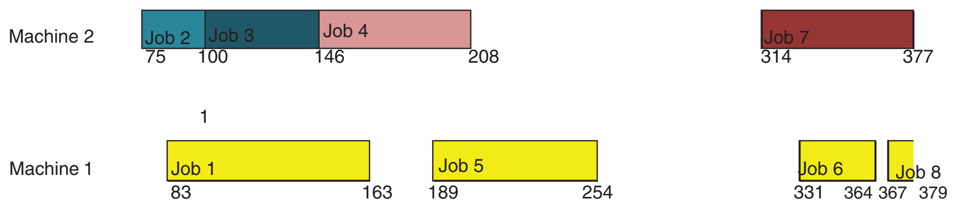

Figure 2. Optimal schedule for Example 3.1.

Table 3. Starting and completion times for an optimal feasible schedule for Example 3.1.

	j							
	1	2	3	4	5	6	7	8
S_j	83	75	100	146	189	331	314	367
C_j	163	100	146	208	254	364	377	379

Table 4. Characteristics of the infeasible schedule obtained by the model of Mason *et al.* (2009) for Example 3.2.

	j							
	1	2	3	4	5	6	7	8
p_j	80	25	45	60	65	30	60	10
d_j	165	105	145	205	255	365	380	380
S_j	85	80	105	145	190	335	320	370
C_j	165	105	150	205	255	365	380	380

cannot be assigned to machine 1 because it causes a conflict with jobs 7 and 8. In addition, job 4 cannot start on machine 2 at $t = 144$ since the machine is busy. Subsequently, the schedule is infeasible.

This schedule can be made feasible pending some changes in the model. In fact, the minimum *WET* for this instance is 6. This corresponds to the schedule described by the Gantt diagram displayed in Figure 2, with the starting and completion times given by Table 3.

As elucidated by Example 3.1, the model of Mason *et al.* (2009) provides a lower bound for the optimal solution value of *WET*. Even when the lower bound and optimal solution value coincide, the resulting schedule may still be infeasible, as Example 3.2 illustrates.

Example 3.2: Consider the eight-job two-machine instance given by rows 1–3 of Table 4. For this instance, the lower bound obtained by the model of Mason *et al.* (2009) is 5. This coincides with the optimal *WET*. However, the corresponding solution given by $\chi_{15} = \chi_{16} = \chi_{17} = \chi_{18} = \chi_{23} = \chi_{24} = 1$, $\epsilon_1 = \epsilon_2 = 1$, generates an infeasible schedule, as can be inferred from the starting and completion times, displayed in rows 4 and 5 of Table 4.

3.1.1 *Modified model*

Modifying the definition of the precedence binary variables and adding a new set of constraints to the model of Mason *et al.* (2009) resolve the aforementioned anomalies. Specifically, let x_{kj}, $k \in N$, $j \in N$, $k \neq j$, equal 1 if job k *immediately precedes* job j and *both k and j are assigned to the same machine*, and 0 otherwise. As previously defined, ϵ_j, $j \in N$, equals 1 if job j is the first job on the machine to which it is assigned, and 0 otherwise. The new MIP model follows:

$$Z_{\text{MIP}} = \min \sum_{j \in N} \alpha_j E_j + \beta_j T_j, \tag{1}$$

$$C_j - d_j = T_j - E_j, \quad j \in N, \tag{2}$$

$$C_j = S_j + p_j, \quad j \in N, \tag{3}$$

$$S_j - S_k \geq p_k x_{kj} - M(1 - x_{kj}), \quad j \in N,\ k \in N,\ k \neq j, \tag{4}$$

$$x_{kj} + x_{jk} \leq 1, \quad j \in N,\ k \in N,\ k < j, \tag{5}$$

$$\epsilon_j \geq 1 - \sum_{\substack{k \in N \\ k \neq j}}^{n} x_{kj}, \quad j \in N, \tag{6}$$

$$\sum_{j=1}^{n} \epsilon_j \leq m, \tag{7}$$

$$\sum_{j=1}^{n} x_{kj} \leq 1, \quad j \in N, \tag{8}$$

$$x_{kj} \in \{0,1\}, \quad j \in N, k \in N, k \neq j, \tag{9}$$

$$\epsilon_j \in \{0,1\}, \quad j \in N, \tag{10}$$

$$S_j,\ C_j,\ E_j,\ T_j \text{ integer}, \quad j \in N, \tag{11}$$

where M is a large positive number such that $M \to \infty$. Here, M is set to $\max_{j=1,\dots,n}\{d_j\} + \sum_{j=1}^{n} p_j$.

Equation (1) defines Z_{MIP} as the minimal sum of the weighted earliness and tardiness of all jobs. Equation (2) computes both the tardiness and earliness of job j by setting their difference to the difference between j's completion time and due date. If j is tardy, then $C_j - d_j > 0$, thus $T_j > 0$ and $E_j = 0$. On the other hand, if j is early, then $C_j - d_j < 0$, thus $T_j = 0$ and $E_j > 0$. When j is on time, $C_j - d_j = 0$, thus $T_j = 0$ and $E_j = 0$. There are n of these equations. Equation (3) sets the completion time of j to the sum of its starting and processing time. There are n of these equations. Equation (4) relates the starting times of two successive jobs on the same machine. If both k and j are scheduled on the same machine, and k is the immediate predecessor of j, then the starting time of j is greater than or equal to the sum of the starting time of k and its processing time. The starting time of j and completion time of k coincide when there is no idle time between k and j, otherwise $S_j > C_k$. The constraint does not establish any relationship between S_k and S_j when k does not immediately precede j, or j and k are not scheduled on the same machine. There are $(n-1)^2$ of these constraints.

Equation (5) reinforces the precedence relations between any pair of jobs k and j. Either k immediately precedes j or j immediately precedes k (if both are scheduled on the same machine), or neither relation holds. These $(n-1)^2/2$ constraints reduce the symmetry of the search space, but have no functional utility.

Equation (6) forces a job j with no immediate predecessor on the machine to which it is assigned to be the first job on that machine. There are n constraints of this type. Equation (7) limits the number of 'first' jobs to at most m. This set of constraints is very important because it ensures that no more than m machines are used. In conjunction with Equation (6), it provides a feasible assignment of the jobs to at most m machines, an important aspect that the model of Mason *et al.* (2009) fails to guarantee. There are n constraints of this type. Equation (8) ensures that a job k is the *immediate predecessor* of at most one job. The n constraints of this type are not equalities because the last job scheduled on any machine has no successor. Omitting this constraint may lead to situations where k is the predecessor of many jobs (as illustrated by Examples 3.1 and 3.2). In such a situation, the schedule is infeasible, and its associated cost is a lower bound to the value of the objective function.

Equations (9) and (10) declare x_{kj} and ϵ_j as binary variables. Finally, Equation (11) states that the starting time, completion time, earliness and tardiness of job j are all integer. This set of constraints can be replaced by non-negativity constraints when the processing times, due dates, and weights are all integers. In that case, there exists at least one optimal solution with integer values for all four variables.

Z_{MIP} is the solution value obtained by an off-the-shelf MIP solver (CPLEX in this case). It is optimal for small-sized instances, but is an upper bound for large or difficult instances. In fact, in the latter case, the solver fails to either converge to the optimum or prove the optimality of the incumbent within a reasonable preset run time (one hour in this case).

3.2 *Metaheuristic-based upper bounds*

As the problem size increases, the success of the MIP solver in identifying the optimum becomes less likely. A viable alternative is to opt for neighbourhood-based (such as SD and SA) and population-based metaheuristics (such as a GA) or their hybrids. This section explores this alternative. Section 3.2.1 details two new constructive heuristics used by SD, SA, and the GA to assess the *WET* of the generated solutions. The first treats the assignment and sequencing aspects of the problem simultaneously, whereas the second hybridises random search with mathematical programming. Sections 3.2.2–3.2.4 give the pseudo-codes of SD, SA, and the GA, respectively, and discuss some implementation issues. Finally, Section 3.2.5 explains how these metaheuristics are hybridised to obtain good-quality solutions within reasonable run times.

3.2.1 *Two new constructive heuristics*

Two new constructive heuristics, CH1 and CH2, are proposed. For an ordered sequence of the n jobs, CH1 chooses simultaneously the machine and the time slot on that machine to which the job is to be assigned such that the *WET* of already scheduled jobs including the current one is minimised. This may involve shifting the processing time window of jobs assigned to the same machine if such a shift reduces the current *WET*. CH2, on the other hand, is a two-stage heuristic. The first stage sequentially assigns each job to a machine, whereas the second stage searches for every job's optimal starting time given its position.

CH1: Let U be an ordered list of the n jobs of N, and let j denote the jth job of U. Initially, CH1 sets $j=1$ and $i=1$. It schedules job j on machine i such that j finishes on time, i.e. during the 'ideal' time interval $[d_j-p_j, d_j]$. If this assignment does not cause a conflict with jobs already scheduled on machine i, CH1 adopts it. Otherwise, CH1 tests the assignment of j to the next machine. Specifically, it increments i by 1 and assigns j to the same time interval $[d_j-p_j, d_j]$. It repeats this step until either the job is assigned to be on time or $i=m$ and no non-conflicting assignment has been identified. In the latter case, CH1 assigns j to the machine where j would cause the least incremental cost to the current objective function value. For each machine i, $i=1,\dots,m$, CH1 positions j in a set of possible alternative positions, and assesses the resulting cost of each alternative. Subsequently, it chooses, for each machine i the assignment that yields the least incremental cost for i. Finally, among these m assignments, CH1 opts for the one that yields the least incremental cost.

Finding the alternative positions of a job j on a machine i where scheduling j on i during $[d_j-p_j, d_j]$ causes a conflict with one or more already scheduled jobs proceeds as follows. CH1 divides the jobs already scheduled on machine i into three groups: G_1, G_2, and G_3. G_1 corresponds to jobs that finish prior to d_j-p_j, whereas G_3 corresponds to jobs that start after d_j. Finally, G_2 embeds those jobs competing with j for the same time slots, i.e. jobs scheduled during part or all of the time interval $[d_j-p_j, d_j]$. CH1 reorders the jobs of each group in ascending order of their starting times, and determines the starting time $\mathbf{S}_g$ and completion time $\mathbf{C}_g$ of G_g, $g=1,2,3$, where $\mathbf{S}_g=\min_{j'\in G_g}\{S_{j'}\}$, $\mathbf{C}_g=\max_{j'\in G_g}\{C_{j'}\}$. Next, CH1 investigates assigning j on i immediately following G_1, immediately preceding G_3, or before and after each job of G_2.

- *Scheduling j immediately after G_1*. CH1 schedules j from $\mathbf{C}_1$ to $\mathbf{C}_1+p_j$. If $\mathbf{C}_1+p_j\le\mathbf{S}_2$, then $E_j=\mathbf{C}_1+p_j-d_j$, and the incremental cost of this assignment is α_jE_j. On the other hand, if $\mathbf{C}_1+p_j>\mathbf{S}_2$, as in Figure 3, then this assignment results in a conflict with jobs of G_2 and possibly G_3. Two alternatives are considered. The one having the least incremental cost is retained. The first alternative pushes forward the starting time of G_2 and may require modifying the starting time of G_3. It sets $\mathbf{S}_2=\mathbf{C}_1+p_j$, and moves the first job of G_2 by $\delta=\mathbf{C}_1+p_j-\mathbf{S}_2$. If the idle time of the machine between the first and second jobs of G_2 is larger than or equal to δ, then no other change is needed. Otherwise, the second job is shifted to the right by an amount that is at most δ. This process is repeated until no overlap between any successive jobs occurs. Applying this alternative to the case of Figure 3 results in the Gantt diagram of Figure 4. The second alternative pushes backward the starting time of both job j and the last job of G_1 by $\delta=\mathbf{C}_1+p_j-\mathbf{S}_2$ time units. In case this shift causes a conflict with other jobs of G_1, their starting times are also pushed back by an amount that is less than or equal to δ. In case this modification results in a negative $\mathbf{S}_1$, then all jobs of G_1 are rescheduled such that the machine's starting time equals 0. If this, in turn, results in conflicts with jobs of G_2, then (i) the

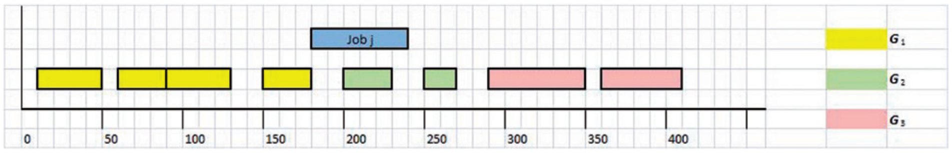

Figure 3. Scheduling j immediately after G_1 results in a conflict with jobs from G_2.

starting time of G_2 is pushed forward by $\mathbf{C}_1 + p_j - \mathbf{S}_2$, where $\mathbf{C}_1$ is the updated completion time of G_1, and (ii) all subsequent necessary changes are made on the time windows of the jobs of G_2 and G_3 so that the schedule becomes feasible. Applying this alternative to the case of Figure 3 results in the Gantt diagram displayed in Figure 5.

- *Scheduling j immediately before* G_3. CH1 schedules j from $\mathbf{S}_3 - p_j$ to $\mathbf{S}_3$. If $\mathbf{S}_3 - p_j \geq \mathbf{C}_2$, then $T_j = \mathbf{S}_3 - d_j$ and the incremental cost of this assignment is deduced. On the other hand, if $\mathbf{S}_3 - p_j < \mathbf{C}_2$, as in Figure 6, then this assignment results in a conflict with jobs of G_2 and possibly G_1. Two alternatives are considered and the one yielding the least incremental cost is adopted. The first alternative pushes forward the starting time of G_3 setting it to $\mathbf{S}_3 = \mathbf{C}_2 + p_j$, and moves the first job of G_3 by $\delta = \mathbf{C}_2 + p_j - \mathbf{S}_3$. If the idle time of the machine between the first and second jobs of G_3 is larger than or equal to the observed overlap, then no further change is needed. Otherwise, the second job is shifted forward by an amount that is less than or equal to δ. This process is repeated until no overlap between any two successive jobs occurs. Applying this alternative to our case yields the Gantt diagram of Figure 7. The second alternative pushes backward the starting time of both j and the last job of G_2 by $\delta = \mathbf{S}_3 - p_j - \mathbf{C}_2$ time units. In case this shift causes a conflict with already scheduled jobs, the starting times of the latter are also pushed back by an amount that is less than or equal to δ. If this further results in an overlap with one or more jobs of G_1, then these are also modified accordingly. In case this modification results in a negative starting time of G_1, all jobs of G_1 and G_2 are rescheduled such that the machine's starting time equals 0. This in turn may result in a conflict with jobs of G_3. This conflict is resolved by pushing forward the starting time of G_3 by $\mathbf{C}_2 + p_j - \mathbf{S}_3$, where $\mathbf{C}_2$ is the updated completion time of G_2, and making all necessary changes on the time windows of the jobs of G_3 to

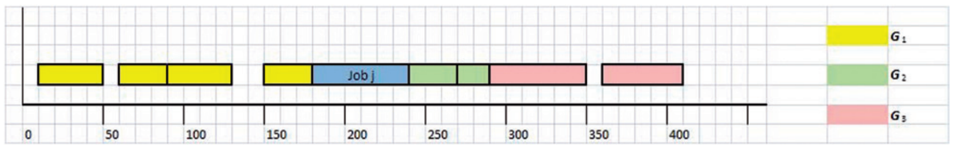

Figure 4. Restoring the feasibility of the schedule by moving the starting time of G_2 forward.

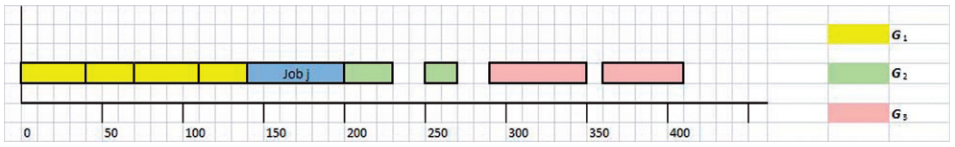

Figure 5. Restoring the feasibility of the schedule by moving the starting time of j backward.

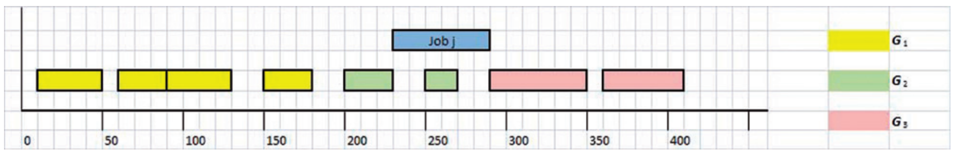

Figure 6. Scheduling j immediately before G_3 results in a conflict with jobs from G_2.

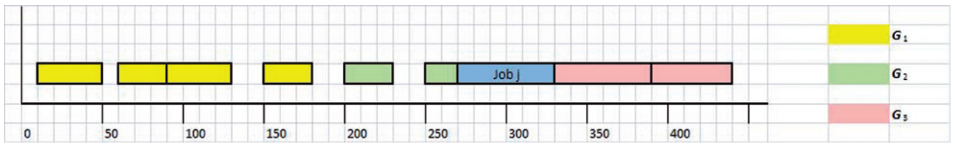

Figure 7. Pushing forward the starting time of G_3 to restore the feasibility of the schedule.

remove any overlap. Applying this alternative to the case of Figure 6 results in the Gantt diagram displayed in Figure 8.

- *Scheduling j immediately before (after) each job of* G_2. CH1 positions j immediately before and after every job $j' \in G_2$ and makes all necessary adjustments to make the schedule feasible, that is to eliminate any resulting overlap of the time windows. Specifically, when positioning j immediately before $j' \in G_2$, it locks the time window of j' by fixing its starting time to its current value $S_{j'}$, and sets the starting time of j to $S_{j'} - p_j$. This might result in an overlap with other jobs that precede j. If that is the case, the time windows of these jobs are pushed backward until no overlap occurs. If this results in a negative starting time for the machine, then all time windows are shifted rightward until the feasibility of the schedule is restored. This might in turn force moving the locked time windows of both j and j' and a few of their successors. When positioning j immediately after $j' \in G_2$, it locks the time window of j' by fixing its starting and completion time to their current respective values $S_{j'}$ and $C_{j'}$. It then sets the starting time of j to $C_{j'}$. If this results in a conflict with any of the successors of j', then the concerned jobs have their starting times moved rightward. Figure 3 shows that G_2 has two jobs. Therefore, CH1 considers four positions for j.

(1) *Before the first job of* G_2. Figure 9 shows the corresponding time window for j. This time window overlaps with that of the last job of G_1. Subsequently, the latter is pushed backward, which forces all the jobs of G_1, in turn, to be pushed backward. In this case, the resulting schedule coincides with that of Figure 5.
(2) *After the first job of* G_2. Figure 10 shows the corresponding time window of j. This time window overlaps with that of the second job of G_2. The latter is pushed forward. Subsequently, all jobs of G_3 are also pushed forward. The resulting schedule is illustrated in Figure 11.

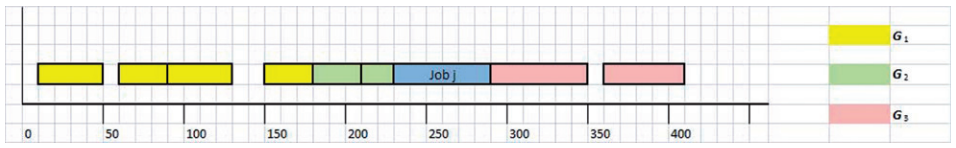

Figure 8. Restoring the feasibility of the schedule by moving the starting time of j backward.

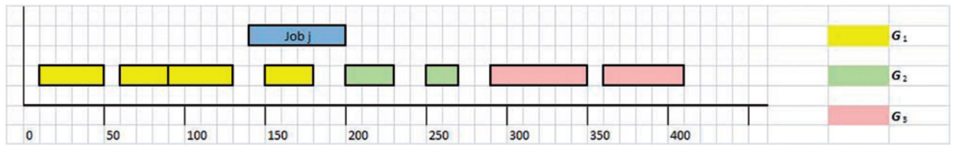

Figure 9. Time window of j when scheduled immediately before the first job of G_2.

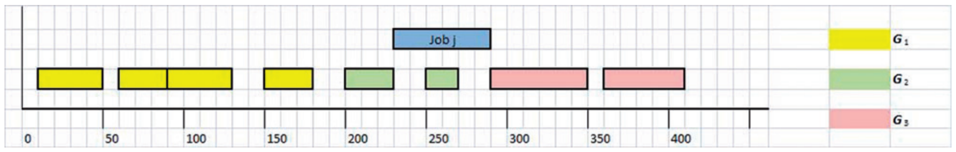

Figure 10. Time window of j when scheduled immediately after the first job of G_2.

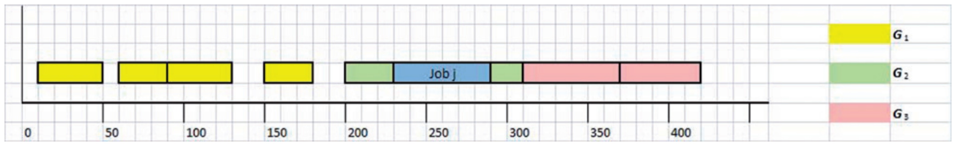

Figure 11. Restoring feasibility when j is scheduled immediately after the first job of G_2.

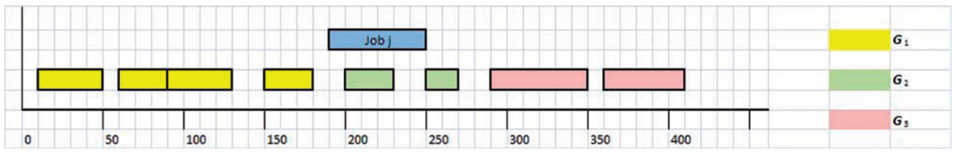

Figure 12. Time window of j when scheduled immediately before the second job of G_2.

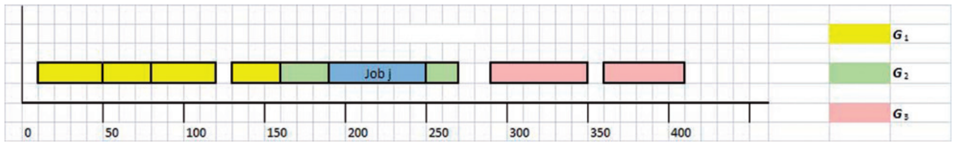

Figure 13. Restoring feasibility when j is scheduled immediately before the second job of G_2.

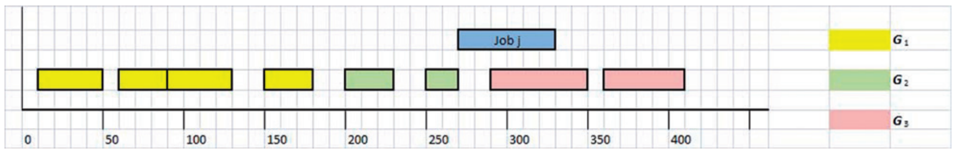

Figure 14. Time window of j when scheduled immediately after the second job of G_2.

(3) *Before the second job of* G_2. Figure 12 shows the corresponding time window of j. This time window overlaps with that of the first job of G_2. Subsequently, the latter is pushed backward, which forces three jobs of G_1 backward. The resulting schedule is given in Figure 13.

(4) *After the second job of* G_2. Figure 14 shows the corresponding time window of j. This time window overlaps with that of the first job of G_3. Subsequently, the latter is pushed forward. In this case, the resulting schedule coincides with that of Figure 4.

For each of the above three cases, CH1 maintains the resulting schedule feasible by shifting jobs left and right of their current time intervals, and computes the resulting incremental WET. Subsequently, it retains the assignment that induces the smallest change in WET.

Of the m retained alternatives, CH1 assigns j to the machine that is least affected (in terms of WET) when j is scheduled on it. This approach is applied iteratively until every job of the ordered list U is assigned a feasible time slot on a machine.

CH2: CH2 is a two-stage approach. The first stage assigns the jobs of an ordered permutation U to the m machines. It defines κ_i, the ordered set of jobs assigned to machine i, as

- κ_i, $i=1,\ldots,m-1$, is the set of jobs in positions $(i-1)[n/m]+1,\ldots,i[n/m]$ of U, and
- κ_m is the set of jobs in positions $(m-1)[n/m]+1,\ldots,n$ of the same permutation.

The second stage solves for each machine i, $i=1,\ldots,m$, an integer program that determines the starting time of each job of κ_i so as to minimise the sum of the WET of the jobs of κ_i, $i=1,\ldots,m$. The integer program is as follows:

$$\min \sum_{k=1}^{|\kappa_i|} \alpha_{[k]} E_{[k]} + \beta_{[k]} T_{[k]}, \tag{12}$$

$$C_{[k]} - d_{[k]} = T_{[k]} - E_{[k]}, \quad k = 1,\ldots,|\kappa_i|, \tag{13}$$

$$C_{[k]} = S_{[k]} + p_{[k]}, \quad k = 1,\ldots,|\kappa_i|, \tag{14}$$

$$S_{[k+1]} \geq C_{[k]}, \quad k = 1, \ldots, |\kappa_i| - 1, \tag{15}$$

$$S_{[k]},\ C_{[k]},\ E_{[k]},\ T_{[k]} \text{ integer}, \quad k = 1, \ldots, |\kappa_i|, \tag{16}$$

where $\alpha_{[k]}$, $\beta_{[k]}$, $d_{[k]}$ and $p_{[k]}$ are the earliness and tardiness weights, due date, and processing time of the kth job of κ_i, $i=1,\ldots,m$. The cardinality of the ordered set κ_i is defined as $|\kappa_i| = \lceil n/m \rceil$, $i=1,\ldots,m-1$, and $|\kappa_m| = n-(m-1)\lceil n/m \rceil$. Equation (13) relates the earliness and tardiness of a job in position k to its completion time and due date. Equation (14) defines the completion time of the job as the sum of its starting time and processing time. Equation (15) reinforces the precedence relations between successive jobs by imposing that a job cannot start unless its immediate predecessor completes. These equations are inequalities, as inserting idle time may reduce the overall objective function value defined by Equation (12). Finally, Equation (16) ensures that the variables are integer.

The performance of CH1 and CH2 depends mainly on the initial random permutation U. Therefore, a multi-start approach is necessary to enhance it. When to be used as stand-alone heuristics, CH1 and CH2 are restarted with different random permutations of N. The process stops when η non-improving solutions are obtained. The best solution is retained as a local optimum. However, in this paper, CH1 and CH2 are used in conjunction with the metaheuristics and their hybrids to decode permutations of the n jobs into good feasible schedules, and provide an estimate of their corresponding *WET*.

3.2.2 *Steepest descent*

SD moves from a current solution U to a neighbouring one $\mathcal{N}(U)$ if $Z_{\mathcal{N}(U)} \leq Z_U$, where $Z_{\mathcal{N}(U)}$ and Z_U are the *WET* of $\mathcal{N}(U)$ and U, respectively. SD stops its search if there is no improving solution in the neighbourhood of U. Here, U is an ordered permutation of the n jobs. Its value Z_U is obtained by applying either CH1 or CH2. In fact, each of the two heuristics yields, in addition to Z_U, a complete schedule that specifies the starting and completion time of each of the n jobs and the machine to which it was assigned.

The choice of the neighbourhood of U is critical to the success of the implementation of SD. For instance, obtaining a neighbour $\mathcal{N}(U)$ of U by swapping two successive jobs scheduled on the same machine, or inserting a job between two successive jobs scheduled on the same machine, or by moving a job from one machine to another is not very judicious since it is very unlikely that any of these neighbours will improve Z_U. Most of these neighbours were implicitly considered and eliminated by CH1 during the search for the time window of every job of U. A more reasonable way consists of swapping a job scheduled on machine i, $i=1,\ldots,m$, with a job scheduled on another machine i', $i'=1,\ldots,m$, and $i' \neq i$. As the schedule-building process progresses, some earlier assignments (based upon the minimal total *WET* for the subset of already scheduled jobs including the current job) may end up being suboptimal. Thus, this type of neighbourhood allows the rectification of such suboptimal earlier assignments.

Ideally, SD should not stop unless it investigates every neighbour of U, i.e. unless it tries to swap every job on every machine i, $i=1,\ldots,m$, with every job scheduled on a different machine i', $i'=1,\ldots,m$, $i \neq i'$, and none of them improves Z_U. Such a strategy would be useful for diversification purposes. However, in this case, SD is used as an exploitation mechanism to intensify the search around the current solution fed to SD rather than an exploration mechanism. In addition, many of these swaps hardly yield improving neighbours; not to mention that this type of neighbourhood search can be time consuming for large instances. Consequently, neighbouring solutions are obtained via swaps of jobs belonging to different machines but having close or overlapping 'ideal' time windows. That is, the swap of job j of machine i with job j' of machine i' is considered if and only if $d_j \in [d_{j'} - \xi p_{j'}, d_{j'} + \xi p_{j'}]$ or $d_{j'} \in [d_j - \xi p_j, d_j + \xi p_j]$, where $\xi = 1$ unless stated otherwise.

When to be used as a stand-alone heuristic, SD is replicated a preset number of times (for example, 1000 times). For every replication, it is fed with a randomly generated permutation of the n jobs. The best solution over all the replications is retained as a candidate for the optimum.

3.2.3 *Simulated annealing*

SA can be viewed as an extended SD. In addition to downhill moves, it occasionally allows uphill moves. SA may move from a current solution U to a not necessarily improving neighbour $\mathcal{N}(U)$ in the hope that it arrives at a global minimum. However, these moves become less frequent as the process ages or gains experience. Here, SA is implemented as follows.

(1) Choose an initial permutation U of N, evaluate its total weighted earliness and tardiness Z_U, and set $Z_{\text{SA}} = Z_U$, where Z_{SA} is the best solution value obtained by SA.
(2) Fix the initial temperature θ_0, the plateau size s, and set $\ell = 0$.
(3) Repeat s times.

- Obtain a neighbour $\mathcal{N}(U)$ of U, and compute its fitness $Z_{\mathcal{N}(U)}$.
- If $Z_{\mathcal{N}(U)} < Z_U$, then set $U = \mathcal{N}(U)$, $Z_U = Z_{\mathcal{N}(U)}$, and $Z_{\text{SA}} = \min\{Z_{\text{SA}}, Z_{\mathcal{N}(U)}\}$.
- Else if $\exp\,[-(Z_{\mathcal{N}(U)} - Z_U)/\theta_\ell] > r$, where r is randomly drawn from the continuous Uniform[0, 1], then set $U = \mathcal{N}(U)$, and $Z_U = Z_{\mathcal{N}(U)}$.

(4) Increment ℓ by 1, and compute the temperature θ_ℓ at plateau $\ell : \theta_\ell = 0.9\theta_{\ell-1}$.
(5) If the stopping criterion is not satisfied, go to Step 3.

In this implementation, the temperature of the annealing process is decreased geometrically. Subsequently, the probability of accepting a non-improving solution decreases as the temperature of the annealing process declines. In addition, for a fixed temperature, small uphill moves have higher probabilities of being accepted than large ones. The initial temperature θ_0 is set to 1. A neighbour $\mathcal{N}(U)$ of U is obtained by randomly permuting a subset of the jobs of U. Specifically, two integer numbers k_1 and k_2 such that $1 \le k_1 < k_2 \le n$ are randomly chosen, and all jobs of U in positions k_1 to k_2 inclusive are permutated randomly. The size s of the plateau is set to n $(n-1)$ neighbours. The algorithm stops if the best current solution does not improve for n consecutive plateaus.

3.2.4 *Genetic algorithms*

The GA was inspired from the evolutionary process of biological organisms in nature. During evolution, populations evolve according to the principles of natural selection. Individuals that are better adapted to the environment have a greater chance of proliferating; those less fit are discarded. The natural selection of the fittest spreads 'good' genes to an increasing number of individuals, favouring the production of offspring that are better adapted to their environment.

Three different genetic algorithms are considered: GP, which is permutation-based, GR, which is a random key GA, and GS, which is assignment-based. All three use solution configurations that directly map to feasible solutions.

GP: In this implementation, a chromosome is represented by an integer permutation $U = ([1][2] \ldots [n])$ of n jobs, where $[j] \in N$ represents the job in position j in the permutation. Every chromosome is subject to crossover, mutation, selection, and replacement.

Crossover produces a new chromosome, `Child`, from two parents: `Parent1` and `Parent2`. `Child` inherits part of its genes from `Parent1` and is completed according to the genes of `Parent2`. In this implementation, three crossovers were considered: partially mapped, merged, and OX. The OX consistently performed best and was retained as the crossover mechanism. Its success is most likely due to its preserving the relative positioning of the jobs. Specifically, `Child` inherits genes k_1 to k_2 in their positions from `Parent1` and has its genes 1 to $k_1 - 1$ and $k_2 + 1$ to n completed with the missing jobs according to their order of appearance in `Parent2`, where $1 \le k_1 < k_2 \le n$.

To illustrate how the OX crossover proceeds, consider an eight-job problem. Let `Parent1` = (3 2 1 5 7 8 6 4), `Parent2` = (8 7 5 4 3 2 1 6), $k_1 = 3$, and $k_2 = 5$. Then, `Child` inherits the subsequence 1 5 7 in positions 3, 4, and 5 from `Parent1` and fills its other genes with the genes of `Parent2` in their order of appearance while making sure that none of them already exists in `Child`. The resulting chromosome is `Child` = (8 4 1 5 7 3 2 6).

Mutation makes an individual better adapted to the environment. It does not greatly disturb the structure of a solution. It randomly shuffles a random subset of jobs of the chromosome.

To illustrate how mutation proceeds, consider the chromosome `Child` = (8 4 1 5 7 3 2 6), $k_1 = 4$, and $k_2 = 6$. The subset of jobs corresponding to subsequence [4] to [6] is 5 7 3. Let the random shuffling of these jobs yield 3 5 7. Then, the mutant of `Child` is = (8 4 1 3 5 7 2 6).

Parent selection defines the reproduction opportunities of every chromosome. Each chromosome initiates crossover (with a randomly selected parent) exactly once. The *replacement strategy* guarantees the survival of the fittest. In addition to the parent population, an offspring population stores each newly generated child. In every generation, the parent and offspring populations are merged and sorted in non-decreasing order of WET. The best fit individuals survive, and constitute a new generation. Each retained individual is subject to mutation. Let $Z_{\texttt{Mutant}}$ be its WET. If $Z_{\texttt{Mutant}}$ is less than or equal to the WET of the least fit individual of the population, then `Mutant` replaces it in the current population, else it is discarded.

A large population size ensures diversification of the chromosomes, while a large number of generations gives chromosomes a greater chance of strengthening their better genes. However, increasing either of them increases the runtime; yet, a good trade-off between diversification and intensification is not always possible. Here, after preliminary testing and tuning of GP, both the population size *popsize* and the number of generations n_g are set to 500.

Ideally, no duplicate chromosomes should exist in the population since having duplicate chromosomes makes the algorithm converge too early to a local optimum or stagnate. Our computational experiments showed that the algorithm reaches a stage where all the chromosomes are identical. However, checking for duplicates in the population is very time consuming, especially for large problems. Thus, a choice was made to prevent the population from having chromosomes with identical WET. Understandably, this choice may eliminate some different chromosomes, but it helps diversify the population. In addition, this is not constraining since there are several non-duplicate chromosomes that yield the same schedule (i.e. having the same starting and completion times for the jobs). Therefore, eliminating these schedules is definitely useful.

To further diversify the population, the second half of the population is discarded and replaced with randomly generated individuals if the incumbent does not improve for 50 generations. A formal statement of GP follows.

- Create an initial population with *popsize* random chromosomes, and sort them in non-decreasing order of their total weighted earliness and tardiness.
- For $generation = 1, \dots, n_g$,
  - For $h = 1, \dots, popsize$,
    - Let `Parent1` be the hth individual of the current population, and randomly select `Parent2` from the remaining $popsize - 1$ chromosomes.
    - Apply crossover to `Parent1` and `Parent2` to obtain `Child`, evaluate its total weighted earliness and tardiness, and place it in the offspring population.
  - Merge the current and the offspring populations, and sort the merged population in non-decreasing order of total weighted earliness and tardiness.
  - Set the current population to the best *popsize* chromosomes of the merged population, and set the offspring population to the empty set.
  - For $h = 1, \dots, popsize$,
    - Mutate the hth individual of the current population to obtain `Mutant`, and evaluate its total weighted earliness and tardiness $Z_{\texttt{Mutant}}$.
    - If the least fit individual of the population has a total weighted earliness and tardiness that is larger than $Z_{\texttt{Mutant}}$, then replace it with `Mutant`.

GR: GR was initially proposed by Bean (1994). A chromosome is represented by n real numbers $R = (r_1, r_2, \dots, r_n)$ such that r_j is associated with job j, and $1 \le r_j < m+1$, $j = 1, \dots, n$. The integer part of r_j refers to the machine to which j is assigned, whereas its decimal part is used to indicate its sequence number on that machine.

Consider, for example, an eight-job three-machine problem, and let

$$(2.39\ 1.87\ 1.63\ 3.21\ 3.33\ 3.01\ 1.97\ 2.22)$$

be a random key chromosome. Sorting the genes in non-decreasing order yields

$$1.63, 1.87, 1.97, 2.22, 2.39, 3.01, 3.21, 3.33,$$

corresponding to jobs

$$3\ 2\ 7\ 8\ 1\ 6\ 4\ 5,$$

indicating that job 3 precedes job 2 which precedes job 7 on machine 1, that job 8 precedes job 1 on machine 2, and that the job precedence on machine 3 is 6, 4, 5.

GR applies a parametric uniform *crossover*: `Child` inherits a gene from `Parent1` with a preset probability of 0.7 and from `Parent2` with probability 0.3. This crossover reassigns jobs to different machines or changes the precedence order of jobs on machines. For example, let

$$\texttt{Parent1} = (2.39\ 1.87\ 1.63\ 3.21\ 3.33\ 3.01\ 1.97\ 2.22),$$
$$\texttt{Parent2} = (1.89\ 1.67\ 1.93\ 3.01\ 3.83\ 2.01\ 2.71\ 3.28),$$

and a set of eight random numbers drawn from Uniform(0,1) be

$$\{0.44,\ 0.51,\ 0.79,\ 0.28,\ 0.71,\ 0.97,\ 0.43,\ 0.86\}.$$

Then the crossover of `Parent1` and `Parent2` yields

$$\texttt{Child} = (2.39\ 1.87\ 1.93\ 3.21\ 3.83\ 2.01\ 1.97\ 3.28),$$

where `Child` inherits genes 1, 2, 4, and 7 from `Parent1` and genes 3, 5, 6, and 8 from `Parent2`.

GR applies the concept of immigration. This mutation prevents premature convergence of the population and the stagnation of the algorithm in a local optimum. At each generation, the worst 1% of chromosomes of the population are replaced by randomly generated chromosomes.

GR's replacement strategy guarantees the survival of the fittest. It automatically transfers the best 20% of the current population to the next generation. In addition, it grants the remaining 80% of the population reproduction rights. In fact, for crossover, the parents are randomly selected from this part of the population, and are replaced by their offspring. The population is then ranked in non-decreasing order of *WET*, and the worst 1% of chromosomes are subject to mutation. Since GR is to be compared with GP, *popsize* and n_g are both set to 500.

A major advantage of GR is that it does not need a constructive heuristic to decode a chromosome and evaluate its *WET*. The genes of each chromosome are ordered and the corresponding ranking is used to build the schedule. Idle time is then inserted on each machine using the integer program of CH2 (i.e. given by Equations (12)–(16)). However, each chromosome is represented using n real numbers (which requires a larger storage than storing n integer numbers). A detailed algorithm of GR follows.

```
• Create an initial population with popsize random chromosomes.
• For $generation = 1, \ldots, n_g$,
    • Sort the current population in a non-decreasing order of $WET$.
    • Copy the top 20% of the current population into the offspring generation.
    • For $h = 1, \ldots, 0.79 * popsize$,
        • Randomly select Parent1 and Parent2 from the bottom 80% of the current population.
        • Apply crossover to Parent1 and Parent2 to obtain Child, evaluate its $WET$, and place it in the
          offspring population.
    • Generate 1% popsize random chromosomes, and place them in the offspring population.
    • Set the current population to the offspring population, and set the offspring population to $\emptyset$.
```

GS: GS was proposed by Monch and Unbehaun (2007). GS is a hybridisation of GR and integer programming. It decomposes the problem into an assignment problem where jobs are assigned to machines, and to m scheduling problems where each scheduling problem requires sequencing a subset of jobs on a single machine and inserting idle times between successive jobs. The assignment component is solved using a GA implementation that resembles that of GR except that it uses an integer solution configuration for the chromosomes. The scheduling component is solved using an off-the-shelf integer programming solver of the mixed-integer program given by Equations (1)–(11) for $m = 1$. Specifically, a chromosome is represented by n integer numbers $I = (i_1\ i_2\ \ldots\ i_n)$, where i_j refers to the machine to which job j is assigned, and $i_j \in \{1, \ldots, m\}$.

Consider an eight-job three-machine problem. To assess the *WET* of any chromosome requires solving three mixed-integer programs. For example, for chromosome (2 1 1 3 3 3 1 2), the first program involves jobs 2, 3 and 7, the second jobs 1 and 8, and the third jobs 4, 5, and 6.

Other than the solution configuration and fitness evaluation, GS and GR use the same crossover, mutation, and selection strategies. It is easier to handle the integer chromosomes of GS than the real ones of GR, however solving m integer mixed-integer programs per chromosome makes the application of GS unrealistic. Therefore, in this implementation, the off-the-shelf integer programming solver is stopped after 6 seconds, and both *popsize* and n_g are set to 50.

3.2.5 *Hybrid algorithms*

Our preliminary experiments showed that GP outperforms GR and GS. Therefore, the hybridisation of GA focuses only on intensifying the search around the local optima obtained by GP. Specifically, four hybrid algorithms are considered.

H1 uses an improved initial population, where each individual of the initial population is subjected to SA.
H2 uses a random initial population but applies SA to each child produced out of crossover.
H3 applies SA to each chromosome of the initial population and every child generated out of crossover.
H4 further extends H3 by applying SD to the final solution.

4. Computational results

The objective of the computational investigation is threefold: (1) to choose the most appropriate GA implementation among GP, GR, and GS; (2) to investigate the success of the different types of hybridisation for different problem types and sizes; and (3) to compare the proposed approach with the approximate approach of Kedad-Sidhoum *et al.* (2008), which is, to our knowledge, the only study that conducts a computational experiment for the weighted version of the problem.

The procedures are coded in FORTRAN and run on a Pentium IV, 2.4 GHz processor, with 2.0 Gb RAM. They are tested on seven sets of instances. Each set is characterised by the number of jobs n, number of machines m, and distributions of the data. It embeds five or 10 instances, as specified in the experiment. Each instance consists of n integer vectors (d_j, p_j, α_j, β_j) corresponding to the due date, processing time, and unit earliness and tardiness penalties of job j, $j = 1, \ldots, n$. All data are rounded off to the nearest integer. Every metaheuristic except for GR and GS is run twice: once with HC1 as the constructive heuristic and once using HC2. The reported best solution is the minimum of the two WET.

4.1 *Choice of GA*

We compare the performance of GP, GR, and GS on a set of instances as generated by Hoogeveen and van de Velde (1996) and Kedad-Sidhoum *et al.* (2008): $n = 30, 60$, $m = 2, 3, 4$, $\tau = 0.2, 0.5, 0.8$, and $\rho = 0.2, 0.5, 0.8$, where τ is the tardiness factor and ρ is the relative range of due dates. For each combination of n, m, τ, and ρ, there are five instances. The weights α_j and β_j are randomly generated from the integer uniform [1, 4] and [1, 6], respectively. The processing times are drawn from the discrete uniform [10, 100], and the due dates from discrete uniform $[\underline{d}, \overline{d}]$, where $\underline{d} = \max\{0; \tau - 0.5\rho P\}$ and $\overline{d} = \underline{d} + \rho P$, with $P = (1/m)\sum_{j=1}^{n} p_j$.

Table 5 compares the relative performance of the three GA implementations. Column 1 indicates the statistic of the measure used to assess GP, GR, and GS. The statistic is: the minimum (Min.), the median (Med.), the average (Ave.), and the maximum (Max.). A minimum value of one indicates that the GA implementation obtained the best solution value at least once. The average and the median describe the central tendency of the data. Any discrepancy between their values indicates that the results are either skewed or embed outliers. The maximum helps us discern extreme cases. Columns 2, 3, and 4 display the ratios $Z_{G\cdot}/Z_{GA}$, where $\cdot = \mathrm{P, R, S}$, respectively, and $Z_{G\cdot}$ is the best

Table 5. Relative performance of the three GA implementations.

Statistic	Z_{GP}/Z_{GA}	Z_{GR}/Z_{GA}	Z_{GS}/Z_{GA}	RT_{GP}	RT_{GR}	RT_{GS}
Min.	1.00	1.39	1.00	1.00	1.00	285.00
Med.	1.00	3.04	1.38	1.40	1.00	473.50
Ave.	1.00	4.25	1.39	1.30	1.26	1052.81
Max.	1.17	16.15	2.82	2.30	2.00	9713.00
%best	98.89	0	1.11			

solution value obtained by G$\cdot$, where $\cdot = \mathrm{P, R, S}$, and $Z_{\mathrm{GA}} = \min\{Z_{\mathrm{GP}}, Z_{\mathrm{GR}}, Z_{\mathrm{GS}}\}$. The ratios reflect the relative quality of the solution obtained by each GA implementation with respect to the best solution value. Columns 5, 6, and 7 display the run times $RT_{\mathrm{G}\cdot}$, where $\cdot = \mathrm{P, R, S}$, expressed in seconds. Finally, the last row of the table shows %best, the percentage of times $Z_{G\cdot} = Z_{\mathrm{GA}}$, $\cdot = \mathrm{P, R, S}$.

Analysis of Table 5 indicates that GP outperforms both GR and GS in terms of solution quality. GR failed to obtain the best solution for any of the tested instances, whereas GS succeeded in three out of 270 instances to obtain the best solution. The run times of GP and GR are almost equal even though, on average, GR is faster. This is mainly due to the fact that GP decodes each chromosome using CH1 and CH2. The long run times of GS are caused by the resolution of m mixed-integer integer programs to assess the fitness of each chromosome.

4.2 *Hybridisation effect*

To investigate the sensitivity of the type of hybridisation on the problem size, type of data, and neighbourhood type, we conduct five experiments. The first two, p_j, $j \in N$, are randomly picked from the integer Uniform$[1, \overline{p}]$, where $\overline{p} = 12$ for $n = 10, 15, 20, 40$ and $\overline{p} = 100$ for $n = 50, 100, 200$. Thus, very few jobs share the same processing time. The due dates d_j, $j \in N$, are chosen randomly from the integer Uniform$[\bar{p}(1 - \tau - \rho/2), \bar{p}(1 - \tau + \rho/2)]$, where the tardiness factor $\tau = 0.1$ and the relative range of due dates $\rho = 0.5$. This choice of τ and ρ is standard for this type of study (Hoogeveen and van de Velde 1996). In addition, the objective is not to study what makes a difficult instance, but rather the overall performance of the hybrid heuristics. The choice of $\tau = 0.1$ is further motivated by real-life applications, where the due dates are not very distant, not to mention that these instances are computationally harder (Kedad-Sidhoum *et al.* 2008). Finally, α_j and β_j are randomly generated from the discrete Uniform[1, 4] and [1, 6], respectively. Assuming that the data are uniformly distributed is common in the absence of relevant information regarding the distributions. In addition, it reflects a wide range of real-life applications.

4.2.1 *Small-sized instances with uniformly distributed data*

Tables 6 and 7 compare the relative performance of the metaheuristics for instances with $n = 10, 15, 20, 40$, and $m = 2, 3, 4$. Table 6 reports, for each problem size, the absolute deviation of Z_{H}, the solution value obtained by metaheuristic $\mathbf{H} \in \mathcal{H} = \{\mathrm{MIP, SA, GP, H1, H2, H3, H4}\}$, from the best known solution value $\overline{Z}$. Opting for the absolute rather than the relative deviation is motivated by the fact that many optima are zero. That is, a relative measure is not meaningful. Depending on the instance, $\overline{Z}$ may be the proven optimal solution value if the MIP solver (CPLEX) reaches the optimum when solving the problem given by Equations (1)–(11) within the allocated one hour time limit without running out of memory. Otherwise, $\overline{Z} = \min_{\mathrm{H} \in \mathcal{H}}\{Z_{\mathrm{H}}\}$. The table further shows the number of times (Nbest) each metaheuristic $\mathbf{H} \in \mathcal{H}$ obtains $\overline{Z}$. The table omits reporting the minima because they are all zero except for the minima of SA for ($m = 3$, $n = 20$) and ($m = 3$, $n = 40$), which are 2 and 4, respectively. On the other hand, Table 7 provides the statistics of RT_{H}, the run time of $\mathbf{H} \in \mathcal{H}$.

For instances with $n = 10$ and 15, the MIP model obtains the optimum for all tested instances. For $n = 20$, it matches the optima in all but two instances having $m = 3$. Its ability to obtain the optima degrades as n becomes larger. For example, $Z_{\mathrm{MIP}} = \bar{Z}$ in only two out of 10 instances when $(m, n) = (3, 40)$. In addition, its runtime becomes longer as the number of machines or jobs increases. Its run times are less than one second for $n = 10$, but increase to 233 seconds for $n = 20$, and to about 2000 seconds for $n = 40$, excluding the run times of instances where the solver aborts because the allocated run time is exceeded.

The analysis of Tables 6 and 7 highlights the advantages of the metaheuristics in comparison with the MIP model. These can be enumerated as follows.

- Despite its good results in terms of solution quality and reduced run time for very-small-sized instances, MIP is not as fast as the other proposed metaheuristics.
- SA should be used as a stand-alone heuristic only for $n \leq 10$. In fact, as n increases, its solution quality deteriorates. However, given its very short runtime, it can be used in conjunction with a multiple start strategy that launches SA with different randomly generated initial solutions, or with the best solution obtained from the previous run until no improvement is obtained for a prefixed number of iterations.

Table 6. Solution quality for small instances with uniformly distributed data.

		$Z_H - \overline{Z}$						
	H	MIP	SA	GP	H1	H2	H3	H4
$n = 10$	Med.	0	0	0	0	0	0	0
	Ave.	0	1	0	0	0	0	0
	Max.	0	5	0	0	0	0	0
	Nbest	30	25	30	30	30	30	30
$n = 15$	Med.	0	3	0	0	0	0	0
	Ave.	0	4	1	0	0	0	0
	Max.	0	19	6	4	4	4	2
	Nbest	30	13	24	24	24	24	26
$n = 20$	Med.	0	3	0	0	0	0	0
	Ave.	0	10	1	1	1	1	0
	Max.	0	42	5	5	5	5	2
	Nbest	28	2	24	22	24	24	27
$n = 40$	Med.	0	9	0	0	0	0	0
	Ave.	0	16	1	1	0	1	0
	Max.	2	58	9	9	7	6	3
	Nbest	18	4	24	21	26	23	25
$m = 2$	Med.	0	2	0	0	0	0	0
	Ave.	0.05	5.20	0.20	0.30	0.20	0.28	0.20
	Max.	2	42	5	5	5	5	3
	Nbest	38	17	37	35	37	36	36
$m = 3$	Med.	0	10	0	0	0	0	0
	Ave.	0.28	15.63	1.50	1.58	1.20	1.30	0.20
	Max.	7	58	23	25	21	25	2
	Nbest	30	10	28	27	30	29	36
$m = 4$	Med.	0	2	0	0	0	0	0
	Ave.	0.05	5.20	0.20	0.30	0.20	0.28	0.20
	Max.	2	42	5	5	5	5	3
	Nbest	38	17	37	35	37	36	36
All	Med.	0	4	0	0	0	0	0
	Ave.	3	11	3	3	3	3	3
	Max.	29	61	28	30	28	30	30
	Nbest	106	44	102	97	104	101	108

- It is difficult to discern a consistently best heuristic among GP, H1, H2, H3, and H4. Evidently, as more hybridisation is added to the metaheuristic, its search space becomes wider and its run time increases. However, there is no guarantee that this hybridisation will not lead to premature convergence.
- The performance of the metaheuristics is worst for $m = 3$. In fact, for $m = 2$, which is the most difficult case, the machines' utilisation is very high, and the jobs are competing for identical time slots on those machines. On the other hand, the machines' utilisation is very low when $m = 4$, thus jobs have a greater chance of being scheduled during their ideal time windows from $d_j - p_j$ to d_j, and conflicts are minimal. For the former case (i.e. $m = 2$), not much improvement is possible because there is almost no idle time on the machines, whereas the limited improvement in the latter case (i.e. $m = 4$) is due to the initial 'good' positioning of the jobs, which makes shifting them or swapping their positions in the schedule useless.
- The last observation also holds true for MIP. In fact, the MIP solver uses a branch-and-bound that fathoms nodes of the tree based on the computed bounds and the value of the incumbent. When $m = 2$ or $m = 4$, the incumbent is a good solution, thus its value allows the elimination of a large number of nodes, making the resolution process relatively fast. In addition, Equation (8) of the model eliminates a large number of symmetric solutions, thus further reducing the search space.

Table 7. Run times (in seconds) of the metaheuristics for $m = 3$.

		Metaheuristic H						
	n	MIP	SA	GP	H1	H2	H3	H4
Min.	10	0.0	0.0	0.0	0.0	0.0	0.0	1.0
Ave.		0.2	0.0	0.1	0.1	0.2	0.1	1.2
Med.		0.0	0.0	0.0	0.0	0.3	0.0	1.3
Max.		1.0	0.0	0.3	0.3	0.3	0.3	1.8
Min.	15	0.0	0.0	0.0	0.0	0.0	0.0	1.5
Ave.		6.1	0.0	0.3	0.3	0.4	0.4	2.0
Med.		1.0	0.0	0.3	0.3	0.3	0.3	2.0
Max.		50.0	0.0	0.7	0.7	0.7	0.7	2.5
Min.	10	2.0	0.0	0.3	0.7	0.3	0.7	1.9
Ave.		232.3	0.0	0.6	0.9	0.6	1.0	2.2
Med.		6.5	0.0	0.5	0.7	0.7	0.7	1.8
Max.		3601.0	0.0	0.7	1.3	1.0	1.3	2.4
Min.	40	322.0	0.0	1.0	2.3	1.3	2.7	1.9
Ave.		1995.6	0.0	1.4	3.2	1.9	3.7	2.2
Med.		2022.0	0.0	1.3	3.0	1.7	3.5	2.2
Max.		3268.0	0.0	2.3	5.3	3.0	6.0	5.9
All								
Min.		0.0	0.0	0.0	0.0	0.0	0.0	2.0
Ave.		838.3	0.0	0.6	1.1	0.8	1.3	6.7
Med.		6.5	0.0	0.5	0.7	0.7	0.7	5.3
Max.		3601.0	0.0	2.3	5.3	3.0	6.0	17.7

4.2.2 *Large-sized instances with uniformly distributed data*

Table 8 reflects the performance of the metaheuristics on instances with $n = 50, 100, 200$, and $m = 2, 3, 4$. It gives statistics of the ratio $Z_H/\bar{Z}$, Nbest, and RT_H for $H \in \mathcal{H}$. Its analysis reveals the following.

- The increasingly larger ratios $Z_{MIP}/\bar{Z}$ suggest that the MIP solver is incapable of reaching reasonably good solutions within the allocated one hour run time. This is most likely due to the looseness of the bounds along with the large search space. This is consistent with the behaviour of search-tree-based exact approaches when dealing with NP-hard problems.
- SA remains non-competitive (despite its very reduced run time, which is of the order of 0 seconds), yet its performance is far better than for MIP.
- The GP-based heuristics avoid premature convergence because
  - the population of any generation does not contain individuals with identical fitness,
  - the initial population is of moderately good quality, and
  - the population is diversified via injected random individuals when no improvement is recorded for a prefixed number of generations.
- The median of $Z_{H4}/\bar{Z}$ equals one for all cases. This implies that H4 obtains the best solution in more than 50% of the tested instances (regardless of the problem size). It also obtains the best averages and the smallest maxima for all tested cases. This performance is not matched by any other heuristic. This is further confirmed by the number of times each metaheuristic obtains the best solution.
- Hybridising GP by applying SA at different stages of the algorithm (as in H1, H2, and H3) yields mixed results. In many instances it results in premature convergence of the algorithm towards a non-global optimum.
- A comparison of the columns for H3 and H4 reveals that applying SD on the best solution obtained by H3 is very useful. SD intensifies the search around the best solution obtained by H3, and improves it (if this solution is not a local minimum). This is due to the design of H3 and H4, which makes $Z_{H4} \leq Z_{H3}$ for all instances, thus $Z_{H4}/\bar{Z} \leq Z_{H3}/\bar{Z}$.

Table 8. Performance of the metaheuristics for large instances with uniformly distributed data.

		$Z_H/\bar{Z}$							RT_H (seconds)					
		MIP	SA	GP	H1	H2	H3	H4	SA	GP	H1	H2	H3	H4
$n = 50$	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.16	0.48	0.20	0.52	2.12
	Ave.	5.93	1.83	1.05	1.06	1.06	1.05	1.03	0.02	0.56	1.37	0.79	1.57	4.05
	Med.	4.85	1.68	1.03	1.02	1.03	1.01	1.00	0.00	0.40	1.35	0.63	1.63	2.95
	Max.	28.53	4.50	1.50	1.50	1.50	1.50	1.50	0.50	1.50	2.50	1.50	3.00	7.50
	Nbest	5	1	13	11	11	12	23						
$n = 100$	Min.	13.28	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.52	2.80	1.12	3.36	6.80
	Ave.	229.28	1.76	1.03	1.03	1.04	1.04	1.01	0.01	1.10	5.33	2.20	6.46	9.94
	Med.	114.29	1.75	1.01	1.01	1.01	1.02	1.00	0.00	1.05	5.40	2.45	6.40	10.70
	Max.	1113.20	4.10	1.14	1.19	1.16	1.20	1.07	0.05	1.80	7.80	3.00	9.60	13.00
	Nbest	0	0	13	12	12	13	24						
$n = 200$	Min.	20.63	1.43	1.00	1.00	1.00	1.00	1.00	0.00	1.00	8.90	2.96	10.76	14.46
	Ave.	539.75	1.86	1.08	1.09	1.08	1.08	1.01	0.08	1.93	13.90	4.98	16.97	25.30
	Med.	283.12	1.78	1.05	1.05	1.06	1.07	1.00	0.05	1.32	12.00	5.50	14.36	25.63
	Max.	3526.26	3.08	1.26	1.29	1.26	1.26	1.04	0.50	3.50	20.16	6.18	24.88	37.00
	Nbest	0	0	4	5	4	2	23						
$m = 2$	Min.	2.45	1.33	1.00	1.00	1.00	1.00	1.00	0.00	0.30	1.20	0.55	1.40	2.55
	Ave.	22.38	1.69	1.10	1.10	1.10	1.09	1.00	0.04	0.91	9.24	3.05	11.42	13.30
	Med.	20.56	1.64	1.09	1.07	1.09	1.08	1.00	0.03	1.05	7.23	2.70	8.98	11.20
	Max.	62.05	2.08	1.23	1.29	1.26	1.23	1.07	0.10	1.42	20.16	6.18	24.88	27.20
	Nbest	0	0	1	1	1	1	27						
$m = 3$	Min.	3.29	1.29	1.00	1.00	1.00	1.00	1.00	0.00	0.16	0.48	0.20	0.52	2.12
	Ave.	154.24	1.77	1.03	1.03	1.04	1.03	1.01	0.02	0.71	4.91	1.76	5.96	9.01
	Med.	117.18	1.77	1.01	1.01	1.03	1.03	1.00	0.00	0.72	3.24	1.36	3.90	7.20
	Max.	600.90	2.88	1.12	1.24	1.18	1.12	1.08	0.06	1.32	12.10	4.22	14.58	18.34
	Nbest	0	0	11	9	7	8	18						
$m = 4$	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00	2.00	1.00	2.00	6.00
	Ave.	595.14	1.99	1.04	1.05	1.04	1.04	1.02	0.05	1.96	6.46	3.16	7.63	16.98
	Med.	423.29	1.82	1.00	1.00	1.00	1.00	1.00	0.00	1.60	5.40	2.60	6.40	10.90
	Max.	3526.26	4.50	1.50	1.50	1.50	1.50	1.50	0.50	3.50	12.50	6.00	15.00	37.00
	Nbest	5	1	18	18	19	18	25						
All	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.16	0.48	0.20	0.52	2.12
	Ave.	264.82	1.82	1.06	1.06	1.06	1.06	1.01	0.03	1.20	6.87	2.66	8.34	13.10
	Med.	43.00	1.76	1.03	1.03	1.03	1.04	1.00	0.00	1.05	5.40	2.45	6.40	10.70
	Max.	3526.26	4.50	1.50	1.50	1.50	1.50	1.50	0.50	3.50	20.16	6.18	24.88	37.00
	Nbest	5	1	30	28	27	27	70						

- The competitive advantage of H4 (due to hybridisation) is, however, attenuated by a larger number of machines. In fact, as m increases, the specific effect of each hybridisation mechanism becomes less pronounced and, subsequently, the choice of the best metaheuristic becomes more difficult. On average, the GP-based metaheuristics perform almost equally well when m is large.
- Different types of hybridisation induce different computational times. This time is a function of the number of solutions being evaluated. The wider the search space or the more intensified the search, the greater the computational cost.
- The computational time of H4, which is the most demanding metaheuristic in terms of run time, remains far less than the allocated run time for MIP.
- The behaviour of the run times of the metaheuristics is due to the particular implementation of the mutation operators and of the neighbourhood searches of SA and SD.
- The run time for SA is negligible for all practical purposes. Therefore, implementing it along with a multiple start search strategy as suggested above is also a valid option for large instances. In addition, using it as an intensification mechanism does not heavily impede the run time of the hybridised heuristic.

4.2.3 *Instances with normally distributed data*

A set of instances with $n=10, 15, 20, 40$, $m=2, 3, 4$, and normally distributed processing times and due dates is generated as per Mason *et al.* (2009). Specifically, p_j, $j=1, \ldots, n$, are normally distributed with mean 100 and standard deviation 10, and d_j, $j=1, \ldots, n$, follow a normal distribution with mean $\mu_d=50n/m$ and standard deviation σ_d such that the coefficient of variation of the due dates $CV_d=\sigma_d/\mu_d=0.1$ or 0.2. Thus, many of the processing times (due dates) are similar or belong to identical tight time windows. As in the experimental setup of Mason *et al.* (2009), $\alpha_j=\beta_j=1, j=1, \ldots, n$. Table 9 reports the results. It gives statistics for $Z_H/\bar{Z}$ along with Nbest for $H \in \mathcal{H}$ and $CV_d=0.1$, 0.2.

Table 9. $Z_H/\bar{Z}$ for normally distributed instances.

		$CV_d=0.1$							$CV_d=0.2$						
	H	MIP	SA	GP	H1	H2	H3	H4	MIP	SA	GP	H1	H2	H3	H4
$n=10$	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	1.00	1.00	1.00
	Ave.	1.01	1.03	1.00	1.00	1.00	1.00	1.00	1.00	1.05	1.00	1.00	1.00	1.00	1.00
	Med.	1.00	1.05	1.00	1.00	1.00	1.00	1.00	1.00	1.03	1.00	1.00	1.00	1.00	1.00
	Max.	1.06	1.11	1.07	1.07	1.07	1.07	1.00	1.02	1.13	1.01	1.01	1.01	1.01	1.00
	Nbest	18	6	28	28	28	28	30	27	9	29	29	29	29	30
$n=15$	Min.	1.00	1.01	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	1.00	1.00	1.00
	Ave.	1.04	1.05	1.00	1.00	1.00	1.00	1.00	1.05	1.08	1.01	1.01	1.02	1.01	1.00
	Med.	1.05	1.04	1.00	1.00	1.00	1.00	1.00	1.05	1.05	1.00	1.00	1.00	1.00	1.00
	Max.	1.11	1.09	1.00	1.00	1.00	1.00	1.00	1.20	1.15	1.12	1.12	1.12	1.12	1.00
	Nbest	0	0	28	28	28	29	29	10	5	25	25	23	24	29
$n=20$	Min.	1.01	1.03	1.00	1.00	1.00	1.00	1.00	1.01	1.02	1.00	1.00	1.00	1.00	1.00
	Ave.	1.05	1.06	1.01	1.01	1.01	1.01	1.00	1.08	1.06	1.01	1.01	1.01	1.01	1.00
	Med.	1.11	1.05	1.00	1.00	1.00	1.00	1.00	1.13	1.07	1.00	1.00	1.00	1.00	1.00
	Max.	1.11	1.28	1.17	1.17	1.17	1.17	1.00	1.20	1.15	1.06	1.06	1.06	1.06	1.01
	Nbest	0	0	23	22	23	26	29	10	1	14	13	13	13	22
$n=40$	Min.	1.02	1.05	1.00	1.00	1.00	1.00	1.00	1.04	1.04	1.00	1.00	1.00	1.00	1.00
	Ave.	1.12	1.07	1.00	1.00	1.00	1.00	1.00	1.25	1.07	1.01	1.01	1.01	1.01	1.00
	Med.	1.19	1.06	1.00	1.00	1.00	1.00	1.00	1.22	1.08	1.00	1.00	1.00	1.01	1.00
	Max.	1.22	1.10	1.03	1.03	1.03	1.03	1.00	1.53	1.15	1.07	1.07	1.06	1.06	1.01
	Nbest	0	0	6	2	5	0	18	8	2	14	10	13	8	17
$m=2$	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	1.00	1.00	1.00
	Ave.	1.04	1.05	1.00	1.00	1.00	1.00	1.00	1.07	1.06	1.01	1.01	1.01	1.01	1.00
	Med.	1.08	1.06	1.00	1.00	1.00	1.00	1.00	1.11	1.06	1.00	1.00	1.00	1.00	1.00
	Max.	1.15	1.09	1.02	1.02	1.02	1.03	1.00	1.37	1.15	1.07	1.07	1.06	1.06	1.01
	Nbest	4	0	25	24	27	27	36	8	0	24	22	23	20	28
$m=3$	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	1.00	1.00	1.00
	Ave.	1.07	1.05	1.00	1.00	1.00	1.00	1.00	1.10	1.07	1.01	1.01	1.01	1.01	1.00
	Med.	1.06	1.05	1.00	1.00	1.00	1.00	1.00	1.11	1.06	1.00	1.00	1.00	1.00	1.00
	Max.	1.22	1.11	1.06	1.06	1.06	1.06	1.00	1.53	1.15	1.12	1.12	1.12	1.12	1.01
	Nbest	7	1	31	28	29	29	34	9	0	21	20	18	18	34
$m=4$	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	1.00	1.00	1.00
	Ave.	1.05	1.05	1.01	1.01	1.01	1.01	1.00	1.10	1.07	1.01	1.01	1.01	1.01	1.00
	Med.	1.08	1.05	1.00	1.00	1.00	1.00	1.00	1.05	1.06	1.00	1.00	1.00	1.01	1.00
	Max.	1.17	1.28	1.17	1.17	1.17	1.17	1.00	1.16	1.13	1.12	1.12	1.12	1.12	1.01
	Nbest	7	5	29	28	28	27	36	38	17	37	35	37	36	36
All	Min.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	1.00	1.00	1.00
	Ave.	1.05	1.05	1.00	1.00	1.00	1.00	1.00	1.09	1.06	1.01	1.01	1.01	1.01	1.00
	Med.	1.04	1.05	1.00	1.00	1.00	1.00	1.00	1.05	1.06	1.00	1.00	1.00	1.00	1.00
	Max.	1.22	1.28	1.17	1.17	1.17	1.17	1.00	1.53	1.15	1.12	1.12	1.12	1.12	1.01
	Nbest	18	6	85	80	84	83	106	55	17	82	77	78	74	98

When $CV_d = 0.1$, all approaches seem to work well for small-sized instances. However, as either m or n increases, the MIP solver faces increasing difficulty in identifying $\bar{Z}$, and its performance ratio deteriorates. This is clearly reflected by the increasing values of the ratio $Z_{\text{MIP}}/\bar{Z}$. It is further clarified by $\text{Nbest}_{\text{MIP}}$, which indicates that MIP did not obtain $\bar{Z}$ in instances with more than 10 jobs. The same phenomenon is observed when analysing the results of the instances with $CV_d = 0.2$. However, the comparison of the results for $CV_d = 0.1$ and 0.2 infers that a wider range of due dates makes the problems relatively easier for the MIP solver as it offers better discrimination mechanisms among nodes, thus helping prune a larger number of them and maintaining a smaller set of nodes that could potentially lead to the optimum. This is further confirmed by the comparisons of the respective $\text{Nbest}_{\text{MIP}}$.

In contrast to the MIP solver, the lack of discrimination caused by the characteristics of the jobs does not seem to affect the solution quality of the other metaheuristics. In fact, they all perform equally well for small-sized instances in both cases ($CV_d = 0.1$ and $CV_d = 0.2$), and their deviation from the best identified solution is consistently low, much lower than MIP's. H4 dominates the other metaheuristics for $n = 10$ and $n = 15$, but not for larger instances. Yet, it retains the best average, median, and smallest maximum ratio for both cases. That is, the effect of the hybridisation used in H4 is more pronounced for small-sized instances, but which hybridisation mechanism produces consistently better results becomes questionable as the problem becomes larger. This is due to the premature convergence towards local minima that is caused by SA intensification.

4.2.4 *Relative cost of earliness and tardiness*

The objective of this experiment is to investigate the robustness of the metaheuristics to variations in the relative weights of earliness and tardiness. The tested instances are generated as in Section 4.1, with $n = 30$ and relative ratios of earliness to tardiness costs equal to 1:1, 1:2, and 2:1. Table 10 displays the statistics of $Z_{\text{H}}/\bar{Z}$ and %best (the percent of time $Z_{\text{H}} = \bar{Z}$) for $\text{H} \in \mathcal{H}$. This table confirms the previous inferences regarding the performance of the heuristics, and the non-dominance of any of them. It further shows that neither MIP nor SA succeed in obtaining a best solution in any of the tested instances. Their average performance is not dependent on the ratios of earliness to tardiness costs.

4.2.5 *Choice of the neighbourhood for SD*

The analysis of the above experiments revealed that H4 generally performs better than all the other metaheuristics of $\mathcal{H}$. This good performance is partially due to the application of SD to the best solution obtained by H3. SD exploits the neighbourhood of H3's solution in the search for a global minimum. This exploitation requires investigating a large number of neighbours provided they are not very different from the current solution. The choice of the neighbourhood of SD for this implementation was discussed in Section 3.2.2. However, it is generally claimed that

Table 10. Impact of the relative costs of earliness and tardiness on $Z_{\text{H}}/\overline{Z}$.

α:β		MIP	SA	GP	H1	H2	H3	H4
1:1	Min.	1.262	1.097	1.000	1.000	1.000	1.000	1.000
	Ave.	1.849	1.337	1.034	1.029	1.053	1.051	1.009
	Med.	1.777	1.327	1.025	1.019	1.041	1.039	1.000
	Max.	3.920	1.765	1.177	1.207	1.200	1.199	1.094
	%best	0.000	0.000	22.222	20.741	2.222	0.741	57.778
1:2	Min.	1.272	1.115	1.000	1.000	1.000	1.000	1.000
	Ave.	1.895	1.367	1.041	1.033	1.068	1.062	1.011
	Med.	1.816	1.345	1.024	1.021	1.054	1.052	1.000
	Max.	3.896	1.792	1.304	1.253	1.347	1.337	1.125
	%best	0.000	0.000	17.037	20.741	3.704	0.741	61.481
2:1	Min.	1.295	1.100	1.000	1.000	1.000	1.000	1.000
	Ave.	1.807	1.325	1.029	1.025	1.049	1.047	1.010
	Med.	1.774	1.313	1.016	1.018	1.037	1.042	1.000
	Max.	3.378	1.720	1.224	1.117	1.201	1.136	1.090
	%best	0.000	0.000	21.481	22.222	5.926	2.222	53.333

the larger the neighbourhood considered during the exploitation stage, the more successful the implementation of SD in identifying a better quality solution. To elucidate this claim, the following experiment considers different sizes of neighbourhoods. This is achieved by varying the parameter ξ. That is, a neighbouring solution is obtained via swaps of two jobs j and j' scheduled, respectively, on machines i and i' with $i \neq i'$ and $d_j \in [d_{j'} - \xi p_{j'}, d_{j'} + \xi p_{j'}]$ or $d_{j'} \in [d_j - \xi p_j, d_j + \xi p_j]$, where $\xi > 0$.

This experiment involves two sets of instances. Set 1 is generated as in Section 4.1 with $n = 30$. Set 2 is generated as in Mason *et al.* (2009) with $n = 50$ and $m = 2, 3, 4$. The processing times are normally distributed with mean 100 and standard deviation 10, whereas the due dates follow a normal distribution with $\mu_d = 50n/m$ and $\sigma_d = 0.1\,\mu_d$. Finally, $\alpha_j = \beta_j = 1$, $j = 1, \ldots, n$.

Each instance is run using GP, H3, and H4$_\xi$, where $\xi = 0.5, 0.75, 1.0, 1.5, 2.0, 2.5$. Table 11 provides statistics on the ratios $Z_H/\overline{Z}$ together with %best for $\mathbf{H} = $ GP, H3, H4$_\xi$, $\xi = 0.5$, 0.75, 1.0, 1.5, 2.0, 2.5.

Table 11 shows that enlarging the neighbourhood search for the exploitation step of H4 is beneficial in most instances, however there is no guarantee that it yields a better quality solution. For example, for Set 1, H4 obtains more best solutions when $\xi = 0.75$ than with $\xi = 1$, while its average performance is better with $\xi = 1$ than with $\xi = 0.75$. In fact, there is no clear rule of thumb on how to set ξ since some of the swaps may lead to premature convergence of SD to a local optimum.

Comparison of the improvement factors for both sets shows that the diversification effect is more pronounced for Set 1 than for Set 2. This is due to the distribution of the processing times and due dates, which make more jobs compete for the same time slot in instances of Set 2 than in instances of Set 1.

Any improvement of the average performance of H4 (thanks to SD's expansion of its neighbourhood search) occurs, of course, at the expense of additional runtime. This is substantiated by Table 12, which provides the harmonic average slow down factor (caused by the intensification strategies of H3 and H4) measured via the ratio RT_H/RT_{GP} for H = GP, H3, H4$_\xi$, $\xi = 0.5$, 0.75, 1.0, 1.5, 2.0, 2.5. Subsequently, a judicious choice of ξ should balance the potential average improvement of the solutions' quality with the incremental computational cost. A plausible approach would vary the neighbourhood size of the best solution obtained by H3, and select for each instance the solution having the smallest objective function value. Another diversification strategy would apply SD, for each neighbour, with ξ randomly picked from [0.5, 3.0].

Table 11. Effect of enlarging the neighbourhood search of SD on $Z_H/\bar{Z}$.

				H4					
Set		GP	H3	$\xi = 0.5$	$\xi = 0.75$	$\xi = 1.0$	$\xi = 1.5$	$\xi = 2.0$	$\xi = 2.5$
1	Min.	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
	Ave.	1.035	1.051	1.017	1.015	1.009	1.008	1.007	1.006
	Med.	1.025	1.039	1.010	1.007	1.001	1.000	1.000	1.000
	Max.	1.184	1.202	1.136	1.126	1.075	1.075	1.075	1.084
	%best	0	23	66	33	24	46	53	59
2	Min.	1.086	1.034	1.000	1.000	1.000	1.000	1.000	1.000
	Ave.	1.330	1.198	1.130	1.123	1.011	1.082	1.051	1.026
	Med.	1.299	1.153	1.095	1.118	1.002	1.048	1.015	1.005
	Max.	1.648	1.436	1.367	1.336	1.075	1.266	1.244	1.124
	%best	0	3	14	24	34	13	43	54

Table 12. Effect of enlarging the neighbourhood search of SD on RT_H/RT_{GP}.

		H4					
m	H3	$\xi = 0.5$	$\xi = 0.75$	$\xi = 1.0$	$\xi = 1.5$	$\xi = 2.0$	$\xi = 2.5$
2	1.31	1.83	1.99	2.14	2.44	2.98	3.98
3	1.28	3.12	3.88	4.19	5.47	7.35	10.68
4	1.23	4.94	6.32	7.12	9.59	13.03	18.61

4.3 *Performance of H4 on the instances of Kedad-Sidhoum* et al. *(2008)*

The set of instances of Kedad-Sidhoum *et al.* (2008) has known lower and upper bounds. It is characterised by $n = 30, 60, 90$, $m = 2, 3, 4$, $\tau' = 0.2, 0.5, 0.8$, $\tau' = 1 - \tau$, and $\rho = 0.2, 0.5, 0.8$. For each combination of n, m, τ', and ρ, there are five instances. For comparison purposes, two performance measures are computed:

- $\Delta_{\text{H4}}^{\text{LB}}$, the percent deviation of Z_{H4} from the known lower bound, LB, defined as

$$\Delta_{\text{H4}}^{\text{LB}} = \frac{Z_{\text{H4}} - \text{LB}}{\text{LB}} 100\%$$

and

- $\Delta_{\text{H4}}^{\text{UB}}$, the percent deviation of Z_{H4} from the known upper bound, UB, defined as

$$\Delta_{\text{H4}}^{\text{UB}} = \frac{Z_{\text{H4}} - \text{UB}}{\text{UB}} 100\%.$$

A negative $\Delta_{\text{H4}}^{\text{UB}}$ indicates that H4 improves the known upper bound. A zero $\Delta_{\text{H4}}^{\text{LB}}$ proves that H4 obtains the optimum and LB is tight. However, a non-zero $\Delta_{\text{H4}}^{\text{LB}}$ is not necessarily a reflection of the performance of H4 because LB may be loose. In fact, $\Delta_{\text{H4}}^{\text{LB}}$ is the optimality gap.

Tables 13 and 14 provide a summary of the percent deviations $\Delta_{\text{H4}}^{\text{LB}}$ and $\Delta_{\text{H4}}^{\text{UB}}$ by problem size (i.e. as a function of m and n) and type (i.e. as a function of ρ and τ'), respectively. They show that H4 improves the known upper bound in many instances with a very small median deviation from both the lower and upper bounds, thus H4 reduces the optimality gap reported by Kedad-Sidhoum *et al.* (2008). This advantage of H4 is further accentuated as n, m, τ', or ρ decrease.

To compare the effect of the constructive heuristics on H4, we record the number of times H4 obtains its best solution value for each of the tested instances using CH1 and CH2. The corresponding percentages are reported for

Table 13. Summary of the performance of H4 on the instances of Kedad-Sidhoum *et al.* (2008) for different problem sizes.

		$\Delta_{\text{H4}}^{\text{LB}}$			$\Delta_{\text{H4}}^{\text{UB}}$		
Statistics	n	$m=2$	$m=4$	$m=6$	$m=2$	$m=4$	$m=6$
Minimum	$n=30$	0.00	0.01	0.00	−1.47	0.00	−0.09
Median		1.94	1.37	1.52	0.23	0.99	0.87
Minimum	$n=60$	0.08	0.40	0.22	−0.59	−0.25	−0.20
Median		3.67	4.25	3.77	1.16	2.48	2.10
Minimum	$n=90$	0.14	0.18	0.22	−0.52	−0.09	0.05
Median		4.09	6.03	6.09	0.81	2.01	4.41

Table 14. Summary of the performance of H4 on the instances of Kedad-Sidhoum *et al.* (2008) for different problem types.

		$\Delta_{\text{H4}}^{\text{LB}}$			$\Delta_{\text{H4}}^{\text{UB}}$		
		$\rho=0.2$	$\rho=0.5$	$\rho=0.8$	$\rho=0.2$	$\rho=0.5$	$\rho=0.8$
Minimum	$\tau'=0.2$	0.00	0.35	0.12	−0.09	−0.25	−0.59
Median		0.27	1.12	5.95	0.08	0.36	2.48
Minimum	$\tau'=0.5$	0.09	0.09	0.24	−0.23	−0.19	−1.47
Median		0.75	3.18	8.10	0.42	1.21	3.60
Minimum	$\tau'=0.8$	0.02	0.17	0.19	0.00	−0.19	0.19
Median		3.67	8.43	11.25	3.54	7.21	7.92

Table 15. Percent of times CH1 is better than CH2 for different problem sizes.

	$m=2$	$m=4$	$m=6$
$n=30$	46.67	53.33	66.67
$n=60$	35.56	42.22	44.44
$n=90$	33.33	33.33	33.33

Table 16. Percent of times CH1 is better than CH2 for different problem types.

	$\rho=0.2$	$\rho=0.5$	$\rho=0.8$
$\tau'=0.2$	0.00	4.44	20.00
$\tau'=0.5$	8.89	13.33	42.22
$\tau'=0.8$	100.00	100.00	100.00

different problem sizes and types in Tables 15 and 16, respectively. The analysis of these two tables reveals that CH1 should be adopted as the decoding mechanism of H4 when $\tau'=0.8$, whereas CH2 should be the constructive heuristic to be applied when both τ' and ρ belong to $\{0.2, 0.5\}$. For other instances, it seems that CH2 dominates CH1 for $n=90$ and $n=60$, but there is no conclusive evidence for the dominance of the two heuristics for $n=30$. In summary, CH2 performs well for relatively easy instances, whereas CH1 is better adapted to difficult instances where several jobs compete for the same time slots. For $\tau'=0.8$ or $\rho=0.8$, the run times of H4 with CH1 as the decoding mechanism are approximately three times longer than those of H4 using CH2. However, these run times are almost equal for all other instances.

5. Conclusion

This paper addresses the minimum weighted earliness tardiness parallel machine scheduling problem with distinct deterministic known due dates. The problem is modelled as a mixed-integer program that provides the exact solution for small or relatively easy instances. For difficult large instances, the problem is approximately solved via two constructive approaches, three metaheuristics and several hybrid heuristics. The first constructive heuristic deals with the assignment and scheduling aspects of the problem simultaneously, whereas the second hybridises a random search with mathematical programming. The hybrid heuristics hybridise the different stages of genetic algorithms (which is a population-based metaheuristic) using various diversification and intensification mechanisms. The extensive computational investigation shows that the hybrid heuristic combining low and high levels of hybridisation (i.e. H4) is the most effective. It reaches the best solution in most instances within a reduced run time, and improves many existing upper bounds. The proposed hybrid heuristics can be extended to more complex earliness tardiness problems with different job environments. Possible extensions include, but are not limited to, the parallel machine problem with release dates, the hybrid flow shop problem, the job shop problem, and their stochastic versions.

Acknowledgements

The authors thank Dr. S. Kedad-Sidhoum and F. Sourd for providing the instances of Kedad-Sidhoum *et al.* (2008) along with their bounds. The authors also thank the two anonymous referees for their constructive comments.

References

Baker, K.R. and Scudder, G.D., 1990. Sequencing with earliness and tardiness penalties: a review. *Operations Research*, 38 (1), 22–36.

Bean, J.C., 1994. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6 (2), 154–160.

Biskup, D. and Cheng, T.C.E., 1999. Multiple machine scheduling with earliness, tardiness and completion time penalties. *Computers & Operations Research*, 26 (1), 45–57.

Chang, P.C., 1999. A branch and bound approach for single machine scheduling with earliness and tardiness penalties. *Computers and Mathematics with Applications*, 37 (10), 133–144.

Chen, Z.L. and Lee, C.Y., 2002. Parallel machine scheduling with a common due date window. *European Journal of Operational Research*, 136 (3), 512–527.

Chen, Z.L. and Powell, W.B., 1999. A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European Journal of Operational Research*, 116 (1), 220–232.

Cheng, T.C.E., 1989. A heuristic for common due date assignement and job scheduling on parallel machines. *Journal of the Operational Reasearch Society*, 40 (12), 1129–1135.

Cheng, T.C.E. and Chen, Z.L., 1994. Parallel-machine scheduling problems with earliness and tardiness penalties. *Journal of the Operational Research Society*, 45 (6), 685–695.

Davis, J.S. and Kanet, J.J., 1993. Single machine scheduling with early and tardy completion costs. *Naval Research Logistics*, 40 (1), 85–101.

De, P., Ghosh, J.B., and Wells, C.E., 1994. Due-date assignment and early/tardy scheduling on identical parallel machines. *Naval Research Logistics*, 41 (1), 17–32.

Emmons, H., 1987. Scheduling to a common due date on parallel uniform processors. *Naval Research Logistics*, 34 (6), 803–810.

Federgruen, A. and Mosheiov, G., 1996. Heuristics for multimachine scheduling problems with earliness and tardiness costs. *Management Science*, 42 (11), 1544–1555.

Fry, T.D., Armstrong, R., and Blackstone, J., 1987. Minimizing weighted absolute deviation in single machine scheduling. *IIE Transactions*, 19 (4), 445–450.

Garey, M.R., Tarjan, R.E., and Wilfong, G.T., 1988. One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research*, 13 (2), 330–348.

Hall, N.G., 1986. Single and multiple processor models for minimizing completion time variance. *Naval Research Logistics Quarterly*, 33 (1), 49–54.

Hall, N.G. and Posner, M.E., 1991. Earliness-tardiness scheduling problems, I: weighted deviation of completion times about a common due date. *Operations Research*, 39 (5), 836–846.

Hall, N.G., Kubiak, W., and Sethi, S.P., 1991. Earliness tardiness scheduling problems, II: deviation of completion times about a restrictive common due date. *Operations Research*, 39 (5), 847–856.

Hendel, Y. and Sourd, F., 2005. Efficient neighborhood search for the one-machine earliness-tardiness scheduling problem. *European Journal of Operational Research*, 173 (1), 108–119.

Hoogeveen, J.A. and van de Velde, S.L., 1996. A branch-and-bound algorithm for single-machine earliness-tardiness scheduling with idle time. *INFORMS Journal on Computing*, 8 (4), 402–412.

Kanet, J.J., 1981. Minimizing the average deviation of job completion times about a common due date. *Naval Research Logistics Quarterly*, 28 (4), 643–651.

Kedad-Sidhoum, S., Solis, R.Y., and Sourd, F., 2008. Lower bounds for the earliness-tardiness scheduling problem on parallel machines with distinct due dates. *European Journal of Operational Research*, 189 (3), 1305–1316.

Kim, Y. and Yano, C.A., 1994. Minimizing mean tardiness and earliness in single machine scheduling with unequal due dates. *Naval Research Logistics*, 41 (7), 913–933.

Lauff, V. and Werner, F., 2004. Scheduling with common due date, earliness and tardiness penalties for multimachine problems: A survey. *Mathematical and Computer Modeling*, 40 (5–6), 637–655.

Lawler, E.L., 1977. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1, 331–342.

Lee, C.Y. and Choi, J.Y., 1995. A genetic algorithm for job sequencing problems with distinct due dates and general early-tardy penalty weights. *Computers & Operations Research*, 22 (8), 857–869.

Lee, C.Y. and Kim, S.J., 1995. Parallel genetic algorithms for the earliness-tardiness job scheduling problem with general penalty weights. *Computers & Industrial Engineering*, 28 (2), 231–243.

Mason, S.J., Jin, S., and Jampani, J., 2005. A moving block heuristic for minimizing earliness and tardiness on a single machine with unrestrictive common due dates. *Journal of Manufacturing Systems*, 24 (4), 328–338.

Mason, S.J., Jin, S., and Jampani, J., 2009. A moving block heuristic to minimise earliness and tardiness costs on parallel machines. *International Journal of Production Research*, 47 (19), 5377–5390.

Min, L. and Cheng, W., 2006. Genetic algorithms for the optimal common due date assignment and the optimal scheduling policy in parallel machine earliness/tardiness scheduling problems. *Robotics and Computer Integrated Manufacturing*, 22 (4), 279–287.

Monch, L. and Unbehaun, R., 2007. Decomposition heuristics for minimizing earliness-tardiness on parallel burn-in ovens with a common due date. *Computers & Operations Research*, 34 (11), 3380–3396.

Morton, T.E. and Pentico, D.W., 1993. *Heuristic scheduling systems*. 8th ed. New York: Wiley.

Radhakrishnan, S. and Ventura, J.A., 2000. Simulated annealing for parallel machine scheduling with earliness and tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, 38 (10), 2233–2252.

Serifoglu, S.F. and Ulusoy, G., 1999. Parallel machine scheduling with earliness and tardiness penalties. *Computers & Operations Research*, 26 (8), 773–787.

Solis, Y.R. and Sourd, F., 2008. Exponential neighborhood search for a parallel machine scheduling problem. *Computers & Operations Research*, 35 (5), 1697–1712.

Sourd, F., 2005. Optimal timing of a sequence of tasks with general completion costs. *European Journal of Operational Research*, 165 (1), 82–96.

Sourd, F. and Kedad-Sidhoum, S., 2003. The one machine problem with earliness and tardiness penalties. *Journal of Scheduling*, 6 (6), 533–549.

Sun, H. and Wang, G., 2003. Parallel machine earliness and tardiness scheduling with proportional weights. *Computers & Operations Research*, 30 (5), 801–808.

Toksari, M.D. and Guner, E., 2009. Parallel machine earliness/tardiness scheduling problem under the effects of position based learning and linear/nonlinear deterioration. *Computers & Operations Research*, 36 (8), 2394–2417.

Toksari, M.D. and Guner, E., 2010. The common due-date early/tardy scheduling problem on a parallel machine under the effects of time-dependent learning and linear and nonlinear deterioration. *Expert System with Applications*, 37 (1), 92–112.

Turkcan, A., Akturk, M.S., and Storer, R.H., 2009. Predictive/reactive scheduling with controllable processing times and earliness-tardiness penalties. *IIE Transactions*, 41 (12), 1080–1095.

Ventura, J.M.S. and Kim, D., 2003. Parallel machine scheduling with earliness tardiness penalties and additional resource constraints. *Computers & Operations Research*, 30 (13), 1945–1958.

Yano, C.A. and Kim, Y.D., 1991. Algorithms for a class of single machine weighted tardiness and earliness scheduling problems. *European Journal of Operational Research*, 52 (2), 167–178.