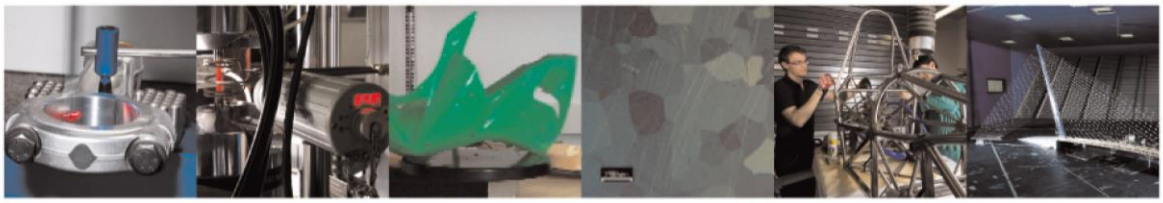


A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility

Yu, Chunlong; Semeraro, Quirico; Matta, Andrea

This is a post-peer-review, pre-copyedit version of an article published in COMPUTERS & OPERATIONS RESEARCH. The final authenticated version is available online at: http://dx.doi.org/10.1016/j.cor.2018.07.025

DIPARTIMENTO DI MECCANICA ■ POLITECNICO DI MILANO
via G. La Masa, 1 ■ 20156 Milano ■ EMAIL (PEC): pecmecc@cert.polimi.it
http://www.mecc.polimi.it
Rev. 0

A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility

Chunlong Yu[a*], Quirico Semeraro[b] and Andrea Matta[c]

[a,b,c]*Dipartimento di Meccanica, Politecnico di Milano, Milan, Italy*

(*Received 00 Month 20XX; accepted 00 Month 20XX*)

This paper presents a genetic algorithm to solve the hybrid flow shop scheduling problem to minimize the total tardiness. Practical assumptions as unrelated machines and machine eligibility are considered. The proposed algorithm incorporates a new decoding method developed for total tardiness objective, which is able to obtain tight schedule meanwhile guarantee the influence of the chromosome on the schedule. The proposed algorithm has been calibrated with a full factorial design of experiment, and compared to several calibrated state-of-art algorithms on 450 instances with different size and correlation patterns of operation processing time. The results validate the effectiveness of the proposed algorithm.

Keywords: Scheduling; Hybrid flow shop; Genetic algorithm.

1. Introduction

Flow shops are manufacturing environments in which a set of jobs are to be processed in a series of stages. Hybrid flow shop (HFS) is a generalization of flow shop where a stage can have two or more parallel machines (Pinedo, 2012). This duplication of machines can introduce flexibility, increase capacities and avoid bottleneck if some operations are more time-consuming. Thus, the HFS is a more common production system and it is applied in many industrial fields such as the electronics, paper, textile, pharmaceutical, and sheet metal industry.

The scheduling problem in HFS is a decision-making process which concerns how to allocate available production resources to tasks over given time periods, aiming at optimizing one or more objectives, such as makespan and due-date related performance. Such problem is not easy to solve. The two-stage HFS problem is already NP-hard on minimizing the makespan (Gupta, 1988), multi-stage HFS scheduling problems with additional system assumptions are NP-hard in strong sense. Though difficult, the HFS scheduling problem still attracts many attentions due to its strong engineering background and practical relevance to industry. Recent and comprehensive reviews on the HFS scheduling problems are available in Ruiz and Vázquez-Rodríguez (2010) and Ribas et al. (2010).

In this paper, we are interested in HFS with unrelated parallel machines and machine eligibility constraints. More specifically, unrelated machine assumption indicates that the parallel machines in a stage are not identical but could be different in terms of processing speed or manufacturing technologies applied; machine eligibility constraint indicates that not all machines in a stage are able to process any job visiting the stage due to certain limitations. These two characteristics are of great importance in modern industry, but less considered by the literature (Ruiz and Vázquez-Rodríguez, 2010). In Figure 1 it is shown an example of 4-stage HFS in sheet metal manufacturing environment with unrelated machines and machine eligibility constraint. The system is composed

*Corresponding author: chunlong.yu@polimi.it

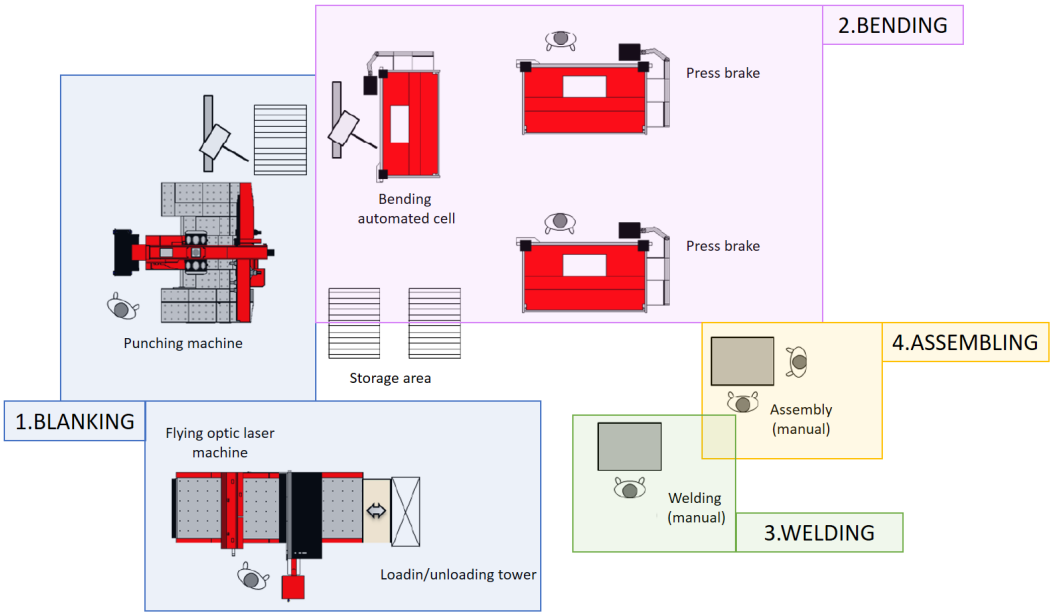


Figure 1. An example of hybrid flow shop with unrelated machines and machine eligibility constraint

of 4 stages: blanking, bending, welding and assembling. Metal sheets enter the blanking stage as raw material and are cut in the laser cutting machine or punching machine into 2D parts. Then, they are delivered to the bending stage to be bent into specific 3D parts, after which welding and assembling process are performed to obtain the final products. In the blanking stage, cutting with the punching machine is usually faster but it can handle only jobs of simple shapes; laser cutting machine is able to cut complex shapes but not adaptable for all materials. In the bending stage, machines are unrelated because of different toolsets, work range, etc. All these realistic features render a HFS model with identical parallel machines not adaptable here.

On the other hand, most HFS studies focus on problems with throughput related measures, such as minimizing makespan or mean flow time (Choi et al., 2005; Ruiz and Vázquez-Rodríguez, 2010). However, in practical cases throughput may not be the most important objective. In manufacturing environments such as make-to-order, a late order implies a penalty in the form of loss of goodwill and the magnitude of the penalty depends on the importance of the order and the tardiness of the delivery (Pinedo, 2012). Clearly, in such circumstance managing the on-time delivery of the orders has more importance than simply improving the throughput of the system. As a matter of fact, optimizing due-date related metrics in schedules such as number of tardy orders, total tardiness and maximum lateness is quite important for the manufacturing companies.

To contribute to the gap between scheduling literature and practical applications, in this paper we consider the scheduling problem in a HFS with unrelated machines and machine eligibility constraints, and the objective is to minimize the total tardiness. A genetic algorithm (GA) with efficient and innovative components is proposed to solve the problem. The rest of this paper is organized as follows: Section 2 briefly describes the problem and gives a standard notation for the problem. Section 3 provides a literature review. Then, the GA components and the proposed GA procedure are described in Section 4. In section 5 numerical experiments are used to calibrate the GA procedure, and a comparison between the proposed GA and the best-performing algorithms in the literature is reported. Section 6 provides conclusion and discuss future research works.

2. Problem description

There are n jobs to be processed at m production stages sequentially from stage 1 to stage m. The i-th stage consists of m_i unrelated machines $M_{i1}, M_{i2}, \ldots, M_{im_i}$, and for at least one stage $m_i > 1$. Each job j consists of a sequence operations O_{ij}, $i = 1, 2, \ldots, m$. The execution of O_{ij} requires one machine out of an eligible machine set E_{ij} at stage i. p_{ilj} is the processing time of O_{ij} on machine M_{il}, and C_{ij} is the completion time of O_{ij}. Any job, say j, has a release date $r_j = 0$ and a nonzero due date d_j. The objective is to minimize the total job tardiness $\sum T_j$, which is calculated by:

$$\sum T_j = \sum_{j=1}^{n} \max\{C_{mj} - d_j, 0\} \tag{1}$$

Followings are other assumptions we use:

- Machines are reliable and no machine failures can happen
- Buffers of machines have unlimited capacities
- Each machine can process only one job at a time, and each job can be processed on only one machine at a time
- Transportation times of job between machines are neglected
- Machine setup times are considered as sequence-independent and included in the processing time.

The scheduling problem can be denoted using a triplet $\alpha|\beta|\gamma$ notation derived from Vignier et al. (1999) and Ruiz and Rodriguez (2010). In this notation, α defines the shop configuration, β describes the constraints and assumptions and γ indicates the objective function. Consequently, the described scheduling problem is denoted as:

$$FHm, ((RM^{(k)})_{k=1}^{m})|M_j|\sum T_j$$

Here, FHm indicates a HFS with m stages; $(RM^{(k)})_{k=1}^{m}$ represents that each stage consists of multiple unrelated machines; M_j represents machine eligibility; $\sum T_j$ indicates the total tardiness objective.

3. Literature review

The literature review can be divided into three parts. First, we review different methods solving HFS scheduling problems with a focus on genetic algorithm. Second, we review papers tackling HFS problems with due-date related objectives. Last, research considering unrelated machines assumption and machine eligibility constraint is reviewed.

Methods for HFS scheduling problem. In literature, methods for HFS scheduling problem can be categorized as exact and heuristic. Exact methods, including mathematical programming and branch & bound, solve the problem to optimality. However, due to the lack of efficient lower bounds, branch & bound approach is limited to simple shop configurations; also, exact methods require long time for solving large instances. Both facts limit the industrial application of exact methods. A practical idea is to search for quasi-optimal solution in a reasonable time. For this reason, the trend of solving HFS scheduling problems with heuristic, especially metaheuristic, is increasing.

In the past decade, genetic algorithm (GA) has gained the widest applications. Xiao et al.(2000) applied a basic GA to minimize makespan in a basic m-stage HFS. Afterwards, research efforts were made to improve the GA building blocks (like representation, crossover and mutation) to

enhance the performance on dedicated HFS problems. Kurz and Askin (2004) implemented a different representation, named random keys representation, in the proposed GA. Urlings and Ruiz (2010) compared the performances of GAs when different representation schemes are used. They showed that compact, indirect representations outperformed verbose representations. Oğuz and Ercan (2005) proposed a new crossover operator in GA and showed the superiority over state-of-art crossover operators. Engin et al. (2011) implemented a new mutation operation able to make use of critical job information to obtain high quality neighborhoods. In some studies, like Tavakkoli et al.(2009) and Chamnanlor et al. (2015), different search techniques are incorporated into the GA framework to improve the algorithm performance.

Besides GA, various types of metaheuristics were proposed for HFS. These include well-known algorithms like simulated annealing (Naderi et al., 2009), tabu search (Wang and Tang, 2009), particle swarm optimization (Liao et al., 2012), ant colony system (Ying and Lin, 2006), immune evolutionary algorithm (Zandieh et al., 2006), iterated greedy algorithm (Urlings et al., 2010), artificial bee colony (Cui and Gu, 2015; Li et al., 2016). In recent years, there emerges a trend to apply new metaheuristics like water flow-like algorithm (Pargar and Zandieh, 2012), firefly algorithm (Marichelvam et al., 2014a), cuckoo search algorithm (Marichelvam et al., 2014b) to solve HFS problems. Indeed, different metaheuristics represent different search patterns in the solution space. However, due to the existence of the No-Free Lunch (NFL) Theorem (Wolpert and Macready, 1997), it is more important on how to make use of problem structure information to improve the search procedure than just applying new general purpose optimization methods to HFS problems.

HFS problems with due-date related objective. Compared to the abundant researches on makespan, due-date related objective receives less attention. Dispatching rules are heuristics prioritizing the job based on static or dynamic information. Due to their simplicity and robustness, dispatching rules were widely applied in early practice and preferred when tackling large size problems. As early as in Paul (1979), several ad-hoc dispatching rules were implemented for minimizing the average tardiness and number of tardy jobs for a two-stage HFS in glass container industry. The shortest processing time based rule was shown to be the most efficient. Adler et al.(1993) developed a scheduling system based on dispatching rules for supporting paper bag factories. Hunsucker and Shan (1992) compared six simple dispatching rules in a HFS with identical machines, and concluded the superiority of the first-in-first-out rule for the mean tardiness criterion. Voß and Witt (2007) considered a large size scheduling problem in a German steel manufacturer, and proposed dispatching rules to minimize the weighted tardiness.

Heuristics with higher complexities were proposed as well, these organize a deterministic and structural path to construct a schedule, which are known as *constructive heuristic*. Botta (2000) developed six heuristics for minimizing maximum lateness in a HFS with job precedence constraints and time lags. Lee et al. (2004) proposed a heuristic for minimizing total tardiness in m-stage HFS with identical machines. The algorithm constructs the schedule for a bottleneck workstation first, based on which the schedule of other workstations is constructed. Later, by extending the famous NEH approach (Nawaz et al., 1983), Choi et al. (2005) proposed several heuristics for minimizing total tardiness in a HFS with reentrant lots. Chen and Chen (2009) considered a HFS with unrelated machines and a bottleneck stage, they proposed two bottleneck-based heuristics with three machine selection rules to minimize total tardiness.

Metaheuristics are proposed to get better solution than deterministic heuristics by randomizing the search routine. Jungwattanakit et al. (2009) implemented and compared several methods including dispatching rule, constructive heuristic and metaheuristics for minimizing the weighted sum of makespan and number of tardy jobs in a HFS with unrelated machines. They showed that, first, metaheuristics generally outperform constructive heuristics, which outperform dispatching rules; second, longest processing time (LPT), NEH and simulated annealing (SA) are the best methods in their categories for the tested cases. The efficiency of SA is shown also in Naderi et al. (2009). The authors proposed a SA for HFS with sequence-dependent setup and transportation times to

minimize total completion time and total tardiness, respectively. Two neighborhood structures as well as a simple local search were implemented in the proposed SA. The proposed SA was reported to be superior to several algorithms including the aforementioned GA in Kurz and Askin (2004) and the immune evolutionary algorithm in Zandieh et al (2006). On the other hand, Li et al. (2015) proposed a GA to address the HFS scheduling problem with batch processing machines to minimize makespan and total weighted tardiness, respectively. Unlike conventional approach, the proposed GA searches for the best combination of dispatching rules used for schedule construction. The proposed approach was compared to the CPLEX solver on small cases and showed better performance under limited computational time (6 hours), but a sound comparison to state-of-art algorithms was not provided. Another metaheuristic, named Iterated greedy algorithm (IG), has been successful applied to many different scheduling problems like permutation flow shop (Ruiz and Stützle, 2007) and unrelated parallel machines (Fanjul-Peyro and Ruiz, 2010). Recently, Pan et al. (2017) proposed four IG-based methods to minimize a convex combination of job tardiness and earliness and obtained state-of-art results.

Unrelated machines and Machine eligibility. Actually, in most of the works tackling realistic problems, unrelated machines and machine eligibility constraint are always taken into consideration. These are like the container handling systems in Chen et al. (2007), the cardboard boxes production system in Alfieri (2009) and the printed circuit-board assembly lines in Yaurima et al. (2009). However, from a large perspective of HFS research, the assumption of unrelated machines is considered in merely 11 % of the more than 200 papers reviewed in Ruiz and Vázquez-Rodríguez (2010).

The scheduling problem in unrelated parallel machines environment, which could be considered as a special case of HFS with only one stage, is already very hard. Generally, the unrelated parallel machine scheduling problems can be decomposed into a partitioning or machine selection problem and a transportation or scheduling sub-problem on each machine. To give an example, we refer to a recent research of Şen and Bülbül (2015), in which the problem was solved by an approximation approach using preemptive relaxation and Benders decomposition. While in many HFS studies, the partitioning and scheduling sub-problem are tackled in a less exact but simpler way, with mainly two methodologies: (i) The partitioning problem is solved by certain machine selection rule, while the decision variables of the scheduling sub-problem are optimized in a higher level search technique. For example, in Ruiz and Maroto (2006), the *earliest finishing time* rule is used to assign jobs to machines and the job sequence on machine is derived from a job permutation which is encoded and searched in the proposed GA. Similar strategy was adopted in Yaurima et al. (2009) and Rashidi et al.(2010). (ii) The decision variables of both the partitioning and scheduling problems are encoded and optimized in search technique, as in Chen et al. (2007).

In summary, to the best of our knowledge, the problem considered in this paper is seldom tackled by GA in the literature.

4. Genetic algorithm

We proposed a genetic algorithm to solve the problem defined in Section 2. Genetic algorithm (Holland, 1992) is a well-known search technique used for combinatorial optimization problems. The genetic algorithm mimics the genetic evolution procedure which results in strong individuals adaptive to the environment. It works with a set of encoded solutions to the problem, called *population*. Each solution is represented by a string of genes, called a *chromosome*. The evaluated objective value of a solution is named the *fitness*. To evolve, the current population generate the *offspring* through some genetic operators. First, a *selection* mechanism picks some chromosomes from the current population to be the parents. The basic idea is that stronger chromosome, i.e., with better fitness, should be selected with higher probability to propagate their genes. Then, parents generate the offspring via a *crossover* process, in which the genes of two parents are exchanged and

Table 1. A summary of encoding & decoding methods used in related papers

Literature	Encoding method	Decoding method	Machine selection rule	Objective
Oguz and Ercan, 2005	job permutation	list scheduling	first available machine	$C_{\max}$
Rashidi, 2010	random key representation	list scheduling	earliest finishing machine	$C_{\max}$ and $T_{\max}$
Pan et al, 2017	job permutation	list scheduling	first available machine	$\sum w_j(T_j + E_j)$
Ruiz and Maroto, 2006	job permutation	permutation scheduling	earliest finishing machine	$C_{\max}$
Naderi, 2009	random key representation	permutation scheduling	earliest finishing machine	$\sum C_j$, $\sum T_j$

inherited by the offspring. After, a *mutation* operator may occur to introduce genetic variations into the offspring. Finally, the current population is replaced by the offspring with certain generational scheme. The evolution continues generation by generation, until a stopping criteria is met.

4.1. *Encoding and decoding*

4.1.1. Common methods

Encoding is to represent a schedule by a string of decision variables, or saying, chromosome. A schedule, from the most general perspective, can be defined by setting the start and finish times for each operation on the machine to which it is assigned. This allows an infinite solution space in making a schedule. Since we are optimizing regular objectives like makespan and tardiness, all operations are expected to be started as early as possible. This makes the schedule a semi-active schedule (Pinedo, 2012), in which no operation can be completed earlier without changing the processing order on any of the machines. In such schedules, the decision variables are reduced to the machine assignment decision of each operation, and the sequence of operations on each machine, with a solution cardinality of $\prod_{i=1}^{m} \frac{(n+m_i-1)!}{(m_i-1)!}$ (Urlings et al., 2010). Obviously, a direct encoding scheme involving such large solution space may render an inefficient searching procedure. Actually, Urlings and Ruiz (2010) studied the GA with different encoding schemes and demonstrated that the more detailed the encoding, the worse the results. Indeed, indirect encoding employing surrogate heuristics in the decoding procedures for completing the solution is usually much efficient than a direct encoding. For this reason, most of the researches use the following indirect encoding scheme: a solution is encoded as a job permutation $\pi = \{1, 2, \ldots, n\}$.

Decoding is to derive a schedule from the encoded chromosome. It is notable that the encoding scheme described above does not contain all required decision variables for constructing a HFS schedule. These missing variables, e.g, machine selection decisions, are actually determined by some heuristics during the decoding procedure, for this reason, decoding method acts an important role for the solution quality. List scheduling (LS) is a decoding method adopted in many researches (as shown in Table 1), it applies as follows: (1) In the first stage, create a job list $L_1 = \pi$, then pick out jobs from L_1 sequentially and schedule them as early as possible on the machine selected by a machine assignment rule, eg., the first available machine. (2) in the remaining stages, the procedure is the same as stage 1 except that $L_i (i > 1)$ is created by sorting the jobs non-decreasingly by their completion time in the precedent stage $i - 1$, in other words, the jobs are scheduled by the First-come-first-served (FCFS) rule. Another widely used decoding method is the Permutation scheduling (PS), as adopted in Ruiz and Maroto (2006). PS is similar to LS except that the job lists in each remaining stage are equal to π as well, i.e., $L_i = \pi, \forall i$. Table 1 summarizes the decoding methods used in related papers.

Though widely applied, both LS and PS have drawbacks. In scheduling, one may want to handle hot jobs, which are the urgent orders from important clients and should be finished as soon as possible. When constructing the schedule with LS, the only way to handle the hot jobs is to arrange them in the head (left part) of the chromosome so that they can get scheduled earlier. Yet this is only for stage 1 but makes no guarantees for the subsequent stages where jobs are queued

Table 2. A simple 2-stage scheduling problem

	Stage 1		Stage 2		
Job	Eligible machines	Processing time	Eligible machines	Processing time	Due-date
1	$\{M_{11}, M_{12}\}$	$\{2, 2\}$	$\{M_{21}\}$	$\{4\}$	9
2	$\{M_{11}, M_{12}\}$	$\{2, 2\}$	$\{M_{21}\}$	$\{3\}$	12
3	$\{M_{11}, M_{12}\}$	$\{5, 5\}$	$\{M_{21}\}$	$\{2\}$	8

by the FCFS rule. Indeed, with the propagation characteristic of LS, one can hardly control the schedule property by manipulating the chromosome, which leads to, from the small aspects, the difficulty to handle urgent jobs, and from a larger perspective, the loss of opportunities to reach some promising zones in the solution space. Such we call the *controllability* problem. In contrast, with PS we have no such problem because we schedule the jobs in each stage by the same sequence π. Yet, this leads to another problem: unnecessary machine idleness. More specifically, when the sequence that jobs exiting from the stage $i-1$ is different from π, to schedule the jobs at the stage i by sequence π we have to delay the starting time of some jobs, which may lead to unnecessary machine idleness. Indeed, ignoring the dynamic properties during the schedule construction results in less tightness of the schedule. This we call the *tightness* problem.

To show controllability and tightness problem, a simple 2-stage problem is given as example in Table 2. We first obtain a solution (chromosome) $= \{3, 1, 2\}$ using the Earliest due date heuristic, which is known for its simplicity and effectiveness for due-date related problem. This algorithm sorts the jobs non-decreasingly by their due-dates. Then, we decode this solution with different decoding methods. The case of LS is shown in Figure 2(a). Due to different job finishing times in stage 1, the job processing sequence changes to $\{1, 2, 3\}$ in the second stage. The priority given to the urgent job 3 is lost, and for this reason job 3 is late. The case of PS is shown in Figure 2(b). To schedule in stage 2 by the sequence π, the starting time of job 1 and job 2 are delayed, and machine M_{21} remains unnecessarily idle in the time period of [2,5], which results in the tardiness of job 1 and job 2, as well as a large makespan.

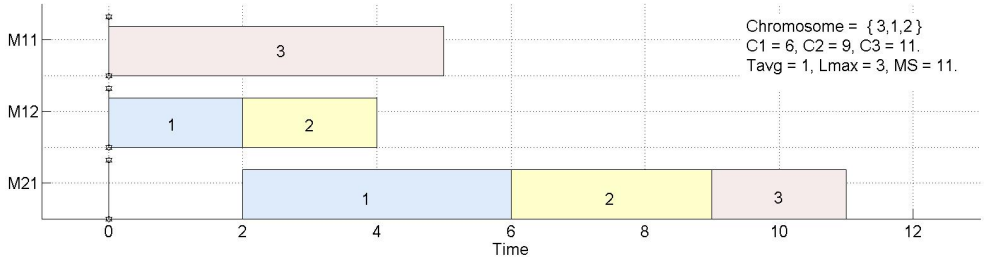
4.1.2. Proposed method

In this section we propose a decoding method for total tardiness objective. To mitigate the controllability and tightness problem, first, the proposed decoding method builds the job sequence in each stage respecting to the chromosome information and meanwhile, the decoding method avoids the introduction of unnecessary machine idleness for a tight schedule. The proposed decoding method is implemented as follows.

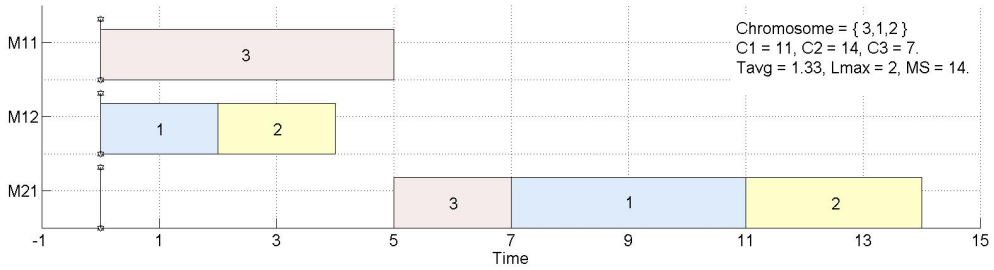
A solution is represented by a job permutation as $\pi = \{1, 2, \ldots, n\}$. The decoding method is based on simulation. For each iteration g we have a system clock t_g. An operation finishing event is denoted by $\mathscr{E} = \{t, M_{il}\}$, in which t is the operation finishing time on M_{il}. A list $\mathscr{L}$ is used to store all operation finishing events. Each job j has a priority value derived from π, as $\rho_j = \{j' : \pi(j') = j\}$. Each machine M_{il} has an infinite buffer $\mathscr{B}_{il}$, a variable a_{il} indicating the next machine available time, i.e., the time machine turns from busy to idle (if the machine is idle at t_g then $a_{il} = t_g$), and a binary variable b_{il} indicating the machine is busy (1) or idle (0). For better explanation, three functions are defined below, each of them is composed of several basic operators. These functions will be called iteratively in the decoding procedure.

- Job_assign$(j, i) \rightarrow M_{il^*}$: Assign job j to the buffer of M_{il^*} in stage i. M_{il^*} is the eligible machine selected by the *least expected workload* rule. In mathematical form that is, $\mathscr{B}_{il^*} := \mathscr{B}_{il^*} + \{j\}$, where

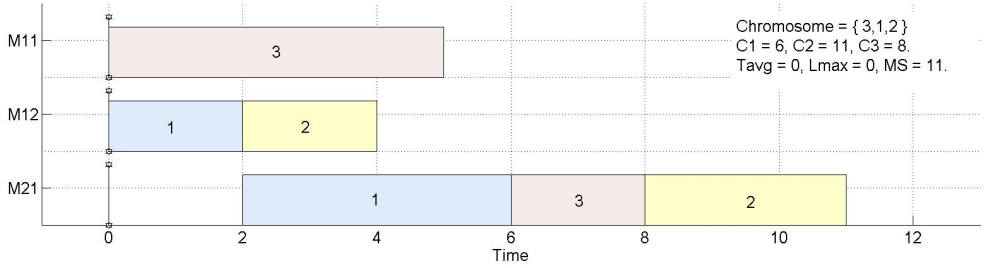
$$l^* = argmin_{l \in E_{ij}} \{ \sum_{j' \in \mathscr{B}_{il}} p_{ilj'} + p_{ilj} + (a_{il} - t_g) \},$$



(a) List-scheduling



(b) Permutation scheduling



(c) Dynamic scheduling

Figure 2. Schedules constructed by different decoding methods using the same chromosome given by EDD heuristic. Notations: Tavg, Lmax and MS are the average tardiness, maximum lateness and makespan of the schedule, respectively.

the first term is a summation of processing time of the jobs already waiting in the machine buffer before assigning job j, second term is the processing time of the assigned job, and third term is the remaining time before the machine becomes available.

- Machine_seize$(M_{il}) \rightarrow [j^*, C_{ij^*}]$: Machine M_{il} seizes the job j^* from the buffer to process. j^* is selected according to priority. That is, $\mathscr{B}_{il} := \mathscr{B}_{il} - \{j^*\}$, where

$$j^* = argmin_{j \in \mathscr{B}_{il}}\{\rho_j\}.$$

  Record the expected completion time of job j^* as $C_{ij^*} := t_g + p_{ilj^*}$, and update $a_{il} := C_{ij^*}$, $b_{il} := 1$.
- Machine_release$(M_{il}) \rightarrow j$: machine M_{il} releases the job j under processing and returns to idle state. That is, $b_{il} := 0$ and $a_{il} := t_g$.

The pseudo codes of the decoding method is as in Algorithm 1. The main characteristics of the proposed decoding method can be summarized as: 1, whenever a job becomes available for being processed, assign it to the machine buffer according to a machine selection rule; 2, whenever a machine becomes available for processing a job, process a job in its buffer selected by their priorities. Moreover, the adopted machine selection rule takes into consideration not only the factor

Algorithm 1 Proposed decoding method

Input: chromosome π;
Output: T_{tot}
Set $g := 1$, $t_g := 0$, $\mathscr{B}_{il} := \emptyset, \forall i, l$, $\mathscr{L} := \emptyset$; Initialize $\rho_j := \{j' : \pi(j') = j\}, \forall j$;
for $k = 1 : n$ **do**
    perform Job_assign$(\pi(k), 1) \rightarrow M_{1l^*}$;
    if $b_{1l^*} = 0$ **then**
        perform Machine_seize$(M_{1l^*}) \rightarrow [j^*, C_{1j^*}]$; create and add $\mathscr{E} = \{C_{1j^*}, M_{1l^*}\}$ to $\mathscr{L}$;
while not all jobs are completed **do**
    Find in $\mathscr{L}$ the event $\mathscr{E}^* = \{t, M_{il}\}$ with the earliest occur time, ties are broken by selecting the event whose job has the highest priority. Set the clock $t_g := t$ and remove $\mathscr{E}^*$ from $\mathscr{L}$;
    perform Machine_release$(M_{il}) \rightarrow j$;
    if $i < m$ **then**
        perform Job_assign$(j, i + 1) \rightarrow M_{i+1,l^*}$;
        if $b_{i+1,l^*} = 0$ **then**
            perform Machine_seize$(M_{i+1,l^*}) \rightarrow [j, C_{i+1,j}]$; add $\mathscr{E} = \{C_{i+1,j}, M_{i+1,l^*}\}$ to $\mathscr{L}$;
    if $\mathscr{B}_{il} \neq \emptyset$ **then**
        perform Machine_seize$(M_{il}) \rightarrow [j^*, C_{ij^*}]$; add $\mathscr{E} = \{C_{ij^*}, M_{il}\}$ to $\mathscr{L}$;
    $g := g + 1$;
Calculate T_{tot} with Eq.(1);

of unrelated machines but also the dynamic status of the eligible machines, aiming at balancing the machine workload meanwhile obtaining a small upper bound for the finishing time of the assigned job.

On one hand, the proposed decoding method allows the chromosome influencing the job sequence in every stage. On the other hand, unnecessary machines idleness is prevented thanks to the machine seize mechanism. In a word, the proposed method has better controllability than LS, and better tightness than PS. However, it should be noted that the time complexity of the proposed decoding method is higher than LS and PS. Given the problem size parameters n, m and h which represent the number of jobs, number of stages and maximum number of parallel machines in a stage respectively, the time complexity of PS, LS and the proposed decoding algorithm are $O(mnh)$, $O(mn(\log(n)+h))$ and $O(mn(n+h))$, respectively. Featured by its dynamic construction procedure, the proposed decoding method is named as *dynamic scheduling* (DS).

We apply the DS to decode the EDD solution for the simple 2-stage problem described in Table 2. As shown in Figure 2(c), the idle period [2,5] of machine M_{21} is avoided, meanwhile job 3 precedes job 2 in stage 2 due to a higher priority indicated by the chromosome. No job is tardy in this case, and a tighter schedule is obtained.

4.2. *Initial population*

Initial population plays an important role in the solution quality of the GA. In many studies, the NEH algorithm (Nawaz et al,1983) is used to generate initial solution. However, when using NEH to handle total tardiness, selection difficulties arise as the sub-schedules are giving identical objective values (Choi et al., 2005). Furthermore, the computation efforts increases rapidly as the number of jobs. Therefore, it is not always sure allocating computational resource to the NEH approach is worthy.

For simplicity, we avoid NEH but use two simple heuristics. The initial population is constructed by $Psize$ individuals as $Pop = \{\pi_1, \pi_2, \ldots, \pi_{Psize}\}$. All individuals are randomly generated except π_1 and π_2 are by earliest due-date heuristic (EDD) and minimal slack heuristic, i.e., sorting the

jobs in ascending order of their due-dates and slacks, respectively;

4.3. *Selection, crossover and mutation*

The aim of selection procedure is to select from the population a set of parents for generating the offsprings. This set is also known as the mating pool. Two most used selection schemes in GA are the roulette wheel selection (Oguz and Ercan, 2005), and the tournament selection (Ruiz and Maroto, 2006). These two selection procedure will be tested in the calibration section and the best one will be chosen.

The goal of crossover is to obtain better chromosomes by exchanging information contained in two parents. Many crossover operators were proposed for permutation encoded chromosomes. According to result of Ruiz and Maroto (2006), the Similar Block 2-Point Order Crossover (SB2OX) and the conventional Partial Mapped Crossover (PMX ,Goldberg and Lingle, 1985) are efficient for permutation solution space. The one-point order crossover (OPX, Michalewicz, 1996) and two-point order crossover (TPX, Michalewicz, 1996) are also widely used. Another promising operator is the order-based crossover (OBX, Yaurima et al., 2009), by homogeneously sampling order information from both parents, this operator compromises between the exploitation and exploration abilities. More specifically, it is based on a randomly generated binary mask of the same length of the chromosome. When the values of the mask are equal to one, the corresponding genes in parent_1 (parent_2) are copied to child_1 (child_2) by maintaining their same position. After, the lacking genes in child_1 (child_2) are filled with those in parent_2 (parent_1) by maintaining their relative orders.

Mutation introduces genetic variability into the population to increase the diversity of population and to avoid early convergence to local optimum. Common mutation operators used for job permutations are:

- *insert*: A randomly selected job is picked out and inserted to another randomly selected position.
- *interchange*: Two randomly selected positions are chosen and the corresponding jobs are interchanged.
- *swap*: It is a specific case of interchange, where two consecutive positions are selected and the jobs are swapped

The effects of different crossover and mutation operators will be tested later.

4.4. *Local search*

Local search procedures are widely used in genetic algorithms as well as in other metaheuristics to improve solutions. We implement a simple local search based on the insert operator as follows:

- Step1: Import the current solution π. Set maximum evaluation number N_{eval}, and initialize the evaluation counter $n_{eval} := 0$;
- Step2: Generate a new solution with the insert operator: $\pi' = insert\{\pi\}$. If the new solution is better, replace π with π'.
- Step3: Update the evaluation count $n_{eval} = n_{eval}+1$. If $n_{eval} \geq N_{eval}$, terminate the procedure and output the current solution π; otherwise go back to Step2.

The local search procedure will be used to improve the best solution in the population. It will be triggered once in every G_l generations. N_{eval} values the computational resource allocated to the local search. Generally, the longer the job permutation, the greater it should be. Thus we set $N_{eval} = int_{ls} * n$ where n is the number of jobs and int_{ls} is named the intensity factor of local search.

4.5. *Restart*

A restart procedure is used to avoid premature convergence in the population and search for further improvements. The restart procedure is based on the one used in Ruiz et al (2006), we add a new mutation operator to increase the gene diversity of the new population. At each generation the best objective function of the population is stored. When the best objective value has remained not improved for G_r consecutive generations, the restart procedure is triggered:

- Step1: Sort the P_{size} chromosomes in the population in ascend order of their objective values.
- Step2: Skip the first 20% chromosomes, i.e., chromosome $1, 2, \ldots, \lceil 0.2 \cdot P_{size} \rceil$.
- Step3: The remaining 80% of chromosomes are discarded and re-generated in the following way:
  - The chromosomes $\lceil 0.2 \cdot P_{size} \rceil + 1, \ldots, \lceil 0.4 \cdot P_{size} \rceil$ are generated by copying a randomly chosen chromosome from the first 20% chromosome. This copied chromosome is mutated once with the insert operator.
  - The chromosomes $\lceil 0.4 \cdot P_{size} \rceil + 1, \ldots, \lceil 0.6 \cdot P_{size} \rceil$ are generated by copying a randomly chosen chromosome from the first 20% chromosomes. This copied chromosome is mutated with a new operator as follows. Randomly select half of genes from the chromosome, then, re-arrange these genes randomly while keeping unchanged the position and order of those genes which are not selected. For example, given a chromosome $\{3, 4, 6, 2, 1, 5\}$, let the selected genes be $\{4, 6, 1\}$, re-arrange them and we have a new order $\{6, 1, 4\}$, so the chromosome mutated becomes $\{3, 6, 1, 2, 4, 5\}$.
  - The remaining $\lceil 0.6 \cdot P_{size} \rceil + 1, \ldots, P_{size}$ chromosomes are generated randomly.

4.6. *Termination criteria*

The termination criteria applied for the proposed GA approach is the maximum CPU time. When the specified CPU time have elapsed, the GA stops.

4.7. *The GA procedure*

With the components described before, the GA procedure is applied in steps:

- Step1: Set GA parameters: population size P_{size}, crossover probability p_c, mutation probability p_m, local search frequency G_l, local search intensity factor int_{ls}, no-improvement generations for triggering restart procedure G_r.
- Step2: Create an initial population Pop with P_{size} chromosomes.
- Step3: Decode each chromosome in the population and evaluate the objective function value.
- Step4: Create a mating pool with P_{size} chromosomes using the selection procedure.
- Step5: Reproduction. Set $i := 1$. While $i < P_{size}$ do:
  - Select two parents π_i, π_{i+1} in the mating pool. Generate a random variable $rand$ in $[0, 1]$, if $rand < p_c$: generate two children with $[\text{child}_1, \text{child}_2] = \text{CrossoverFunction}(\pi_i, \pi_{i+1})$; otherwise let $\text{child}_1 = \pi_i$ and $\text{child}_2 = \pi_{i+1}$.
  - Generate $rand$ in $[0, 1]$, if $rand < p_m$: let $\text{child}_1 = \text{MutationFunction}(\text{child}_1)$. Repeat the same procedure for child_2.
  - Decode and evaluate child_1 and child_2.
  - Let π_{worst} be the chromosome which has the worst fitness value in Pop. If child_1 has better fitness value than π_{worst}, and child_1 is not included in Pop, replace π_{worst} with child_1. Repeat the same procedure for child_2.
  - Update $i := i + 2$.
- Step6: Check local search condition. If the condition is met, trigger the local search procedure to improve the best solution in the population Pop.

- Step7: Check restart condition. If the condition is met, perform the restart procedure. Evaluate the new constructed population.
- Step8: Check termination condition. If the condition is met, end the procedure and output the best solution in the population; otherwise go back to step 4.

5. Numerical results

5.1. *Test instance*

Since no benchmark problems are found in the literature for the problem considered in this paper, to calibrate the proposed algorithm and to compare its performance to some state-of-art algorithms, we generate the test instance as follows.

An instance is defined by 3 parameters as $\mathscr{P} = \{n, m, I\}$. n and m indicate the number of jobs and number of stage, respectively. I indicates the pattern used for generating operation processing times. The common pattern is to generate the processing time for each operation independently using a uniform distribution. However, this does not fit for the practical situation where correlations may exist between the operation processing times. We consider two types of correlations: machine-based and job-based. On one hand, the machine characteristics and conditions may have impact on the job processing time, hence jobs visiting the same machine may have certain level of correlation in their processing times. On the other hand, processing time of operations of the same job could be correlated due to some common attributes like size of the order, quality requirements. As shown in Semeraro (1983), in cases of different machine-based and job-based correlation levels, the performance ranking of scheduling heuristics could be different. For a sound comparison, we define 5 different patterns. When $I = 1$, no correlation is involved. The processing time of each operation is given as $DU(1,\ 99)$, where $DU(x, y)$ returns a randomly integer in range $[x, y]$; when $I = 2$ or $I = 3$, machine-based correlation is involved. Here, $p_{ilj} = c_{mac} \cdot p^*_{il} + (1 - c_{mac}) \cdot DU(1,\ 99), \forall i, j, l \in E_{ij}$, where $p^*_{il} \in [1\ 99]$ is the machine-based processing time common for all jobs processed on machine M_{il}, and c_{mac} is the machine-based correlation factor set as 0.25 and 0.75 in these two cases, respectively; when $I = 4$ or $I = 5$, job-based correlation is involved. Here, $p_{ilj} = c_{job} \cdot p^*_j + (1 - c_{job}) \cdot DU(1,\ 99), \forall i, j, l \in E_{ij}$, where $p^*_j \in [1\ 99]$ is the job-based processing time common for all operations of job j, and c_{job} is the machine-based correlation factor set as 0.25 and 0.75 in these two cases, respectively.

For any instance, the job release dates are set to 0, and the due dates are generated using the method of Choi et al.(2005). Given two parameters TF and DR called tardiness factor and due date range respectively, the due date is calculated by: $DU(P(1-TF-DR/2), P(1-TF+DR/2))$, where P is a lower bound on the makespan calculated by

$$P = \sum_{i=1}^{m}\{\min_j\{\sum_{k=1}^{i-1}\min_{l\in E_{kj}} p_{klj}\} + (1/m_i)\sum_{j=1}^{n}\min_{l\in E_{ij}} p_{ilj} + \min_j\{\sum_{k=i+1}^{m}\min_{l\in E_{kj}} p_{klj}\}\}/m.$$

We set $TF = 0.1, DR = 0.8$ for a proper simulation of real situations. In each stage the number of parallel machines is $DU(2,\ 4)$. For each job, not all but at least one machine in a stage is eligible to process it. The probability that a machine is not eligible to process a job is set as 20%.

5.2. *Algorithm Calibration*

It is well recognized that the components and parameter values significantly affect the algorithm performance. In this section we aim at calibrating the components and parameters of the proposed

algorithm. The algorithm involves in total 9 parameters to be calibrated, which are listed below with their levels:

- Selection: two levels (Tournament selection, Roulette wheel selection)
- Crossover: five levels (OBX, PMX, SB2OX, OPX, TPX)
- Mutation: three levels (Insert, Interchange, Swap)
- Population size (P_{size}): four levels (30, 70, 110, 150)
- Crossover probability (p_c): three levels (0.8, 0.95 and 1)
- Mutation probability (p_m): three levels (0.00, 0.03 and 0.05)
- Restart (G_r): three levels (30, 50 and ∞)
- Local search frequency (G_l): three levels (5, 10 and ∞)
- Local search intensity factor (int_{ls}): two levels (1 and 5)

To simplify the calibration experiment we separate it into two phases: first, we calibrate and select the best components (selection, crossover, mutation) for the GA; then, the rest of parameters are calibrated.

5.2.1. Calibration phase 1: GA components

With a full factorial experimental design,the three factors of GA components results in $2*5*3 = 30$ algorithms. Each algorithm is evaluated by fixing other GA parameters as follows. We set $p_c = 0.95$, $p_m = 0.05$ to guarantee the triggering of crossover and mutation operators, a moderate $P_{size} = 50$ is used; since we are focusing on the performance of genetic operators here, the restart and local search are not applied by imposing $G_r = \infty$ and $G_l = \infty$. Each algorithm is tested with a small set of 48 instances created by the method of section 5.1 with parameters as follows. $I = 1$, and the number of jobs and machines are chosen randomly from the sets $n \in \{15, 30, 45, 60, 75\}$ and $m \in \{4, 8, 12\}$, respectively.

The termination criterion is the maximum elapsed CPU time $t = 10000+\tau\times n^2\times(m/2)$ ms, where $\tau = 5$. It is composed by a basic term plus another term increasing with the problem size for a better search of the enlarged solution space. Each algorithm is evaluated by 5 independent replications on each test instance. To compare different algorithms, the common measure for makespan criteria is the relative percentage increase (RPI), as in Pan and Ruiz (2012). Yet for tardiness criteria, RPI is no longer adaptable because it may provide a division by zero when the schedule has no tardy jobs (Naderi et al., 2009). For this reason we use the relative deviation index (RDI) as the response variable, which is defined as follow:

$$\text{RDI} = \frac{Alg_{sol} - Min_{sol}}{Max_{sol} - Min_{sol}} \cdot 100$$

where Alg_{sol} is the objective value of the current algorithm on the given instance, Max_{sol} and Min_{sol} are the worst and best objective value obtained by any of the algorithms in the comparison, respectively. Specially, in the case that the Max_{sol} and Min_{sol} are equal to each other, the RDI will be 0 for all the algorithms. The experiments are implemented in Matlab 2016a on a PC with Intel XEON E5-2699 v4 CPU (22 cores, 2.2 GHZ) and 256 GB of RAM. To increase algorithm running speed, all decoding methods (which accounts for more than 90% of algorithm running time) are converted to C++ codes and called in Matlab environment.

The results of calibration experiments are analyzed by ANOVA. The ANOVA table in Figure 3 shows that all GA components have significant effects on the performance with significance level 0.05. According to the F-values, crossover and mutation are much influential than the selection scheme, which can be seen in the main effect plot in Figure 4(a) as well. The interactions between the GA components are less important due to a much smaller F-value. To select the best levels, we use Tukey test on each factor with a familywise significant level equal to 0.05. As in Figure 4(b),

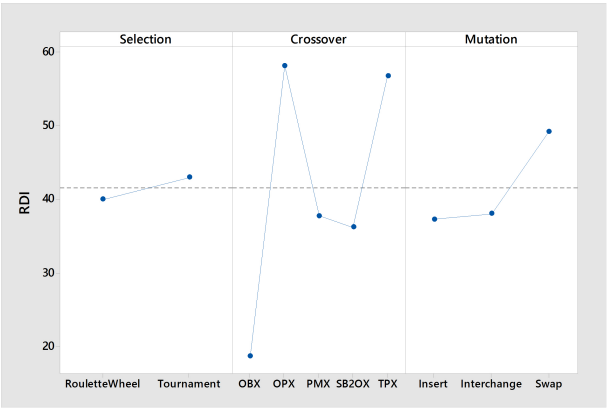
Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Inst	47	108953	2318.1	55.15	0.000
Selection	1	3189	3189.3	75.87	0.000
Crossover	4	308753	77188.3	1836.21	0.000
Mutation	2	42875	21437.4	509.97	0.000
Inst*Selection	47	1687	35.9	0.85	0.747
Inst*Crossover	188	39412	209.6	4.99	0.000
Inst*Mutation	94	9627	102.4	2.44	0.000
Selection*Crossover	4	762	190.6	4.53	0.001
Selection*Mutation	2	521	260.4	6.19	0.002
Crossover*Mutation	8	6224	778.0	18.51	0.000
Error	1042	43802	42.0		
Total	1439	565806			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
6.48358	92.26%	89.31%	85.22%

Figure 3. ANOVA table of the calibration of GA components. In the model we include the test instance as a factor to reduce the noise and to increase the R-sq value. The normality assumption is accepted with a Anderson-Darling test of 0.01 significant level.



(a) Main effect plot

Tukey Pairwise Comparisons: Response = RDI

Selection	N	Mean	Grouping
Tournament	720	43.0342	A
RouletteWheel	720	40.0577	B

Crossover	N	Mean	Grouping
OPX	288	58.2075	A
TPX	288	56.7444	A
PMX	288	37.8126	B
SB2OX	288	36.2567	C
OBX	288	18.7085	D

Mutation	N	Mean	Grouping
Swap	480	49.2513	A
Interchange	480	38.0557	B
Insert	480	37.3308	B

(b) Tukey test table. Means that do not share a letter are significantly different.

Figure 4. Calibration results for the GA components

RouletteWheel and OBX are statistically better than the other levels in their group; for mutation, Insert and Interchange are not statistically different but both outperform Swap. As a consequence, we choose RouletteWheel, OBX and Insert as the GA components.

5.2.2. Calibration phase 2: GA parameters

The factors of P_{size}, p_c, p_m, G_r, G_l and int_{ls} result in a total of $4 \times 3 \times 3 \times 3 \times 3 \times 2 = 648$ different configurations for the proposed GA. The same approach in phase 1 is used to calibrate these parameters.

As shown in the ANOVA table (Figure 5), all GA parameters have significant effects. Although the normality assumption of ANOVA is violated (Figure 6(b)), which may due to the large amount of data or the effect of RDI measure, the F-test is still considered robust. Among the GA parameters,

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
P_size	3	1216772	405591	5143.12	0.000
p_c	2	40876	20438	259.17	0.000
p_m	2	39063	19532	247.67	0.000
G_r	2	4993	2496	31.65	0.000
G_l	2	12531	6265	79.45	0.000
int_ls	1	5573	5573	70.67	0.000
P_size*p_c	6	3739	623	7.90	0.000
P_size*p_m	6	15212	2535	32.15	0.000
P_size*G_r	6	4507	751	9.53	0.000
P_size*G_l	6	36310	6052	76.74	0.000
P_size*int_ls	3	1188	396	5.02	0.002
p_c*p_m	4	580	145	1.84	0.118
p_c*G_r	4	6	1	0.02	0.999
p_c*G_l	4	191	48	0.61	0.658
p_c*int_ls	2	138	69	0.87	0.418
p_m*G_r	4	207	52	0.66	0.622
p_m*G_l	4	10250	2563	32.49	0.000
p_m*int_ls	2	233	117	1.48	0.228
G_r*G_l	4	1330	333	4.22	0.002
G_r*int_ls	2	53	26	0.33	0.717
G_l*int_ls	2	4291	2145	27.21	0.000
Error	31032	2447208	79		
Lack-of-Fit	576	28773	50	0.63	1.000
Pure Error	30456	2418435	79		
Total	31103	3845252			

Model Summary

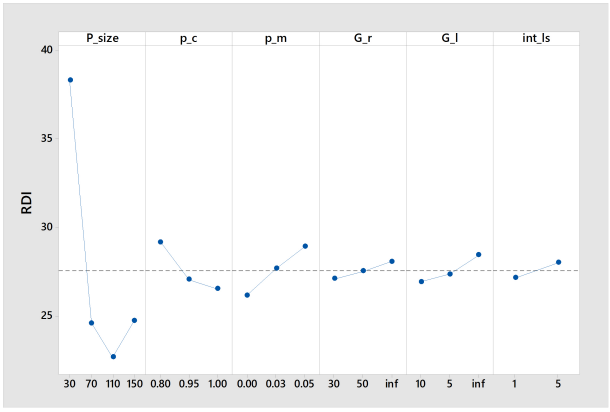
S	R-sq	R-sq(adj)	R-sq(pred)
8.88036	36.36%	36.21%	36.06%

Figure 5. ANOVA table of the calibration of GA parameters. The normality assumption is rejected by a Anderson-darling test of 0.01 significant level.

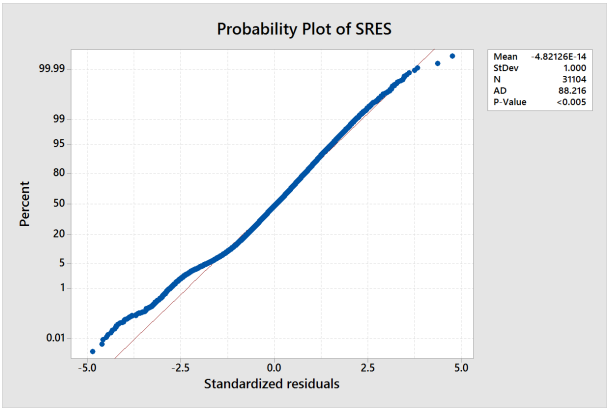
P_{size}, p_c and p_m are the three most influential factors according to the F-values. This is also seen in the main effect plot (Figure 6(a)). Some significant interactions between factors are observed. To analyze the most important ones, we focus on those with F-values larger than 50 (only $P_{size} * G_l$ satisfies this condition) and report the interaction plot in Figure 6(c). As shown, the interaction effect between P_{size} and G_l is obvious at $P_{size} = 30$ whilst becomes negligible when we select more promising levels of P_{size} such as $P_{size} = 110$.

As shown in Figure 6(a), the optimum value of P_{size} is obtained at 110. Indeed, increasing P_{size} introduces more diversity in the population which benefits the evolution, yet trade-off exits between the population size and number of generations when the computational budget is bounded, this explains the "bath-curve". In terms of crossover and mutation, higher p_c yields better performance, while the application of mutation deteriorates the performance.

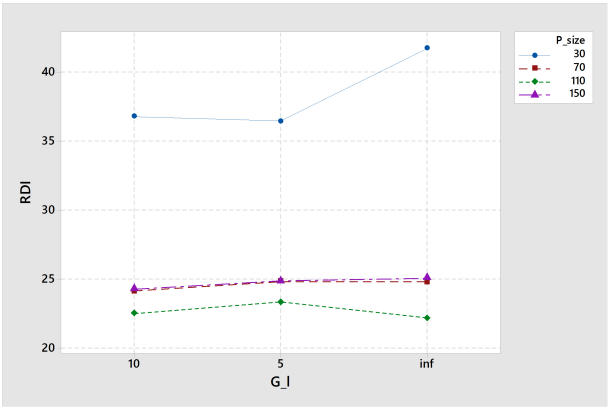
The effects of restart and local search are less obvious. Removing both of them from the GA framework does not seem to deteriorate the performance too much. However, keeping the restart as a diversification mechanism and the local search as a intensification mechanism maintains the potential effectiveness of the algorithm to handle larger or more complex instances beyond the calibration. This is shown in the extra experiments where we compared the versions of GA with/without restart and local search. The main effect plot suggests a moderate restart and local search frequency, as well as a low local search intensity.



(a) Main effect plot



(b) Standardized residual plot of the Anderson-Darling test for Normality



(c) Interaction plot between P_{size} and G_l

Tukey Pairwise Comparisons: Response = RDI

P_size	N	Mean	Grouping
30	7776	38.3299	A
150	7776	24.7512	B
70	7776	24.6035	B
110	7776	22.6752	C

p_c	N	Mean	Grouping
0.80	10368	29.1823	A
0.95	10368	27.0574	B
1.00	10368	26.5301	C

p_m	N	Mean	Grouping
0.05	10368	28.9178	A
0.03	10368	27.6752	B
0.00	10368	26.1767	C

G_r	N	Mean	Grouping
inf	10368	28.0973	A
50	10368	27.5547	B
30	10368	27.1178	C

G_l	N	Mean	Grouping
inf	10368	28.4494	A
5	10368	27.3845	B
10	10368	26.9359	C

int_ls	N	Mean	Grouping
5	15552	28.0132	A
1	15552	27.1666	B

(d) Tukey test table. Means that do not share a letter are significantly different.

Figure 6. Calibration results for the GA parameters

The previous qualitative analysis yields the following candidate levels: $P_{size} = 110$, $p_c = 1$, $p_m = 0$, $G_r = 30$, $G_l = 10$ and $int_{ls} = 1$. To provide quantitative supports, we use Tukey test on each factor with familywise significant level equal to 0.05 (Figure 6(d)). As shown, for any factor, the candidate level is significantly better than the other levels. It should be noted that applying Tukey test requires normality assumption. In our case, although the normality test fails, the violation is not too much severe as seen in Figure 6(b). For this reason, the results of Tukey

Table 3. Calibrated factors of the state-of-art algorithms

Algorithm	Parameters			
GA_Ruiz	$P_{size} = 20$	$P_c = 0.5$	$P_m = 0.02$	$G_r = 25$
SA_Naderi*	C = 150	D = 30	E = 10	
ABC_Cui	MR = 0.8	CR = 0.2	IT = 3	$d = 3$
GA_Li	$ps = 48$	$pc = 0.9$	$pm = 0.18$	

*In SA_Naderi, factor C represents number of temperatures between the initial and final temperature; D represents number of neighborhood search in each temperature; E represents the initial temperature

test are not exact but still make sense to certain extent. In summary, we set $P_{size} = 110$, $p_c = 1$, $p_m = 0$, $G_r = 30$, $G_l = 10$ and $int_{ls} = 1$.

5.3. *Performance comparison*

5.3.1. Adaptation and calibration of state-of-art algorithms

After calibration, the performance of the proposed GA algorithm is to be verified by comparing it to some state-of-the-art algorithms. However, as we have seen in section 3, no algorithms are found for the considered problem and thus direct comparison cannot be carried out. To overcome this difficulty, we have first selected several novel algorithms for HFS scheduling problem, then we have applied some adaptations on the algorithms for using them on the considered problems.

Five state-of-art algorithms are selected, including one deterministic heuristic NEH_EDD (Ruiz et al., 2008); two efficient metaheuristics, i.e., GA by Ruiz and Maroto (2006), denoted as GA_Ruiz, and SA by Naderi et al.(2009), denoted as SA_Naderi; and two recent proposed metaheuristics, i.e., ABC by Cui et al. (2015), denoted as ABC_Cui, and GA by Li et al.(2015), denoted as GA_Li. The adaptations are as follows. First, GA_Ruiz and ABC_Cui were proposed for makespan objective, we modify their objective function to total tardiness. Then, NEH_EDD was proposed for standard flowshop, we modify it to be adoptable to hybrid flow shop by replacing its decoding method with the one used in GA_Ruiz, i.e., PS. For this reason, the algorithm is actually NEH_EDD(PS). Besides, in the decoding method used in ABC_Cui, jobs are assigned to the first-available machine that can process them. Yet according to Ruiz et al.(2006), this rule is not efficient in the environment where unrelated machines exist. For a fair comparison, we modified the machine selection rule of ABC_Cui by applying the one used in GA_Ruiz and SA_Naderi, i.e., jobs are assigned to the machine which is expected to finish them at the earliest time. At last, the random key representation adopted in SA_Naderi was found not adaptable when machine eligibility constraint exists, we replace it with the common job-based representation.

To decide the parameters for these state-of-art algorithms, we perform the calibration procedure used in section 5.2 for each of them except NEH_EDD which is parameter free. More specifically, for each algorithm we generate a full factorial DOE using the factors and levels taken from its original paper, then we evaluate each configuration of the algorithm with the same instance set used for calibrating GA_Yu. Note that we calibrate only the factors with numerical levels, whilst the settings of factor indicating the algorithm components are set as in the original paper, such as the factors related to the selection of crossover operator and encoding method. The calibrated factors of each algorithm are reported in Table 3.

5.3.2. Comparison with state-of-art algorithms

The performance of the algorithms are compared on 5 groups of instances created using the method of section 5.1. Each instance group corresponds to a different value of I, i.e., in group 1 $I = 1$, group 2 $I = 2$ and so on. In each group, instances are generated using all combinations of $n \in \{20, 50, 100\}$ and $m \in \{5, 10, 20\}$. For each combination of n and m, 10 instances are generated. So in a group

there are $3 \cdot 3 \cdot 10 = 90$ instances, and in total $90 \cdot 5 = 450$ instances are created. Each algorithm runs 5 replications on each instance. For fair comparison, all algorithms stop after a CPU time of $t = 10000 + \tau \times n^2 \times (m/2)$ ms, where τ is set as 10, except the deterministic algorithm NEH_EDD(PS).

The comparison results on each instance group can be seen in Tables 4-8. Besides the 5 state-of-art algorithms we include several variants of GA_Yu, which will be introduced and discussed in the next section. Each cell in the table contains a mean value and a standard deviation (in the parenthesis) of the RDI value. Note that the mean value is the average of the 5 replicates for each one of the 10 instances on each $n \times m$ combination, so 50 values are averaged at each cell, and the standard deviation is calculated using these 50 values. Underlined value is the minimum value among GA_Yu and the 5 state-of-art algorithms, while bold value is the minimum value of all 9 algorithms. Given that RDI reveals only the relative rank of the algorithms, we provide also the results of RPI in Table 9-13, where RPI $= \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} \cdot 100$. We remove all instances where $Min_{sol} = 0$ (only found in one instance), and cap the RPI values at 1000 to avoid outliers.

Table 4 shows that when no correlation exists among the operation processing time, GA_Yu provides the best result in all $n \times m$ combinations cases. Moreover, GA_Yu has the minimum averaged standard deviation of RDI, this shows the stability of the GA_Yu performance. As for the other algorithms, SA_Naderi is shown to be the second best algorithm, followed by ABC_Cui,GA_Ruiz which have similar averaged RDI, whereas the rankings of these three algorithms deviate with the problem size. Since GA_Ruiz and SA_Naderi both use NEH_EDD for initialization, it is of no surprise that they outperform NEH_EDD, yet the difference between them decreases with the increase of problem size. The results of GA_Li are consistently worse in almost all cases. Indeed, GA_Li is implemented without any efficient initialization scheme. Moreover, the strategy of this algorithm is to find out the best combination of heuristics to construct a schedule rather than the detail decision variables in the schedule. Such strategy shows no advantages in the cases at hand, but could be more efficient for larger problem with hundreds or thousands of jobs.

From Tables 5-8 it can be seen that, with the presence of different type and extent of correlation effects, the proposed GA_Yu is still the best algorithm in all cases. In contrast, GA_Li performs the worst in almost all cases. As in Figure 7, the performance of GA_Yu and GA_Li are quite stable regardless of the correlation patterns. For other algorithms, their performances fluctuate but their rankings maintain. It is noticed that gap of averaged RDI between GA_Yu and the second best state-of-art algorithm SA_Naderi is more than 30 when I = 5. This gap seems large but may only due to the effect of RDI. As seen in Table 13, the gap between averaged RPI of GA_Yu and SA_Naderi is only 12.33.

Not only the correlation pattern, problem size is also an influential factor of the algorithm performance. As shown in Figure 8 and 9, the performance of ABC_Cui deteriorates as number of jobs n increases whereas NEH_EDD has the opposite trend. The performance of other algorithms are less affected by n. On the other hand, as the number of stages increases, ABC_Cui and NEH_EDD gives relatively better performance. In summary, GA_Yu has stable performance in terms of both n and m.

Table 4. Performance comparision results on instance group 1 (I= 1, random): mean and standard deviation of the RDI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	23.49 (12.19)	43.21 (22.52)	17.26 (10.03)	86.84 (17.79)	70.47 (22.69)	**8.54** (7.79)	16.34 (11.51)	10.31 (7)	9.08 (7.16)
20x10	29.94 (8.41)	45.73 (15.39)	23.36 (13.24)	71.08 (21.55)	74.03 (19.27)	**7.09** (5.8)	21.9 (7.12)	12.81 (10.43)	8.65 (7.4)
20x20	50.52 (13.89)	57.53 (16.47)	19.62 (9.56)	78.93 (20.88)	78.91 (15.89)	11.25 (6.91)	41.92 (13.84)	15.36 (7.55)	**9.53** (6.55)
50x5	27.43 (7.04)	37.02 (13.08)	48.58 (12.4)	63.41 (13.38)	83.43 (16.37)	**8.37** (5.8)	13.58 (6.26)	15.71 (7.98)	10.74 (7.05)
50x10	30.72 (6.17)	34.76 (6.22)	36.8 (7.55)	41.95 (5.37)	81.2 (14.82)	**6.74** (5.54)	17.73 (6.36)	12.32 (5.95)	6.84 (4.98)
50x20	39.31 (6.38)	39.82 (6.92)	37.26 (6.5)	47.3 (7.92)	85.88 (12.85)	**7.02** (5)	26.78 (8.62)	9.52 (5.64)	7.67 (6.03)
100x5	14.36 (8.71)	24.45 (11.04)	46.91 (21.27)	36.37 (15.14)	72.61 (18.88)	10.05 (5.68)	**7.31** (4.67)	10.9 (7.66)	8.77 (6.39)
100x10	25.36 (7.32)	32.08 (10.64)	39.63 (9.44)	40.05 (13.68)	85.47 (11.93)	7.39 (5.24)	11.92 (6.11)	9.46 (6.85)	**7.08** (4.83)
100x20	27.79 (5.16)	28.18 (6.99)	37.49 (5.36)	30.61 (6.77)	86.36 (10.32)	**6.22** (4.62)	16.77 (5.88)	9.62 (4.57)	6.96 (4.98)
Average	29.88 (8.36)	38.09 (12.14)	34.1 (10.59)	55.17 (13.61)	79.82 (15.89)	**8.07** (5.82)	19.36 (7.82)	11.78 (7.07)	8.37 (6.15)

Table 5. Performance comparision results on instance group 2 (I= 2, Machine-based correlation 0.25): mean and standard deviation of the RDI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	18.03 (7.46)	37.13 (11.38)	25.48 (12.29)	65.43 (22.94)	72.92 (21.64)	**8.44** (7.49)	11.08 (8.59)	19.41 (14.73)	11.35 (9.55)
20x10	23.87 (8.76)	35.48 (19.17)	24.02 (10.36)	64.2 (21.38)	76.9 (16.8)	10.64 (8.26)	13.43 (8.99)	15.15 (11.29)	**9.17** (7.3)
20x20	28.8 (12.81)	42.56 (12.37)	17.47 (5.91)	59.27 (14.47)	85.19 (12.41)	7.47 (4.72)	19.05 (12.82)	11.25 (5.97)	**7.3** (5.36)
50x5	26.45 (12.06)	42.85 (18.94)	44.64 (18.15)	68.12 (21.86)	80.8 (16.52)	**9.6** (5.78)	10.94 (10.63)	12.89 (9.34)	12.15 (9.43)
50x10	32.07 (6.64)	37.36 (8.09)	36.98 (8.89)	46.09 (12.01)	84.28 (12.71)	**8.03** (6.07)	17.33 (9.56)	11.12 (8.28)	8.08 (5.59)
50x20	33.43 (5.79)	38.28 (7.02)	34.92 (5.74)	44.46 (8.55)	84.9 (11.15)	**6.51** (4.31)	20.92 (6.33)	7.82 (5.1)	8.1 (5.84)
100x5	24.63 (13.24)	35.77 (16.98)	55.69 (23.72)	67.05 (26.75)	79.64 (18.24)	12.78 (8.17)	10.42 (8)	12.47 (8.1)	**10.01** (6.71)
100x10	33.06 (11.63)	40.37 (11.41)	52.14 (8.07)	48.96 (11.35)	88.34 (10.01)	**8.21** (6.39)	11.12 (8)	11.12 (6.27)	9.75 (6.11)
100x20	31.42 (4.53)	34.47 (7.8)	38.35 (7.56)	37.08 (7.02)	89.38 (8.63)	**6.04** (4.45)	19.08 (5.97)	7.98 (4.93)	8.64 (4.52)
Average	27.97 (9.21)	38.25 (12.57)	36.63 (11.19)	55.63 (16.26)	82.48 (14.23)	**8.64** (6.18)	14.82 (8.77)	12.13 (8.22)	9.39 (6.71)

Table 6. Performance comparision results on instances group 3 (I= 3, Machine-based correlation 0.75): mean and standard deviation of the RDI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	19.25 (10.76)	32 (16.3)	24.23 (15.02)	74.48 (26.19)	66.04 (26.84)	13.98 (13.63)	**7.4** (6.48)	17.33 (15.8)	14.31 (11.75)
20x10	20.31 (11.93)	38.04 (15)	23.61 (11.01)	74.47 (19.02)	77.8 (15.13)	10.23 (7.31)	14.61 (12.37)	13.67 (7.04)	**9.4** (6.37)
20x20	22.66 (10.49)	36 (16.06)	17.81 (8.25)	58.54 (18.93)	83.15 (15.6)	7.49 (6.01)	16.76 (11.17)	10.32 (4.9)	**7.09** (4.91)
50x5	46.22 (18.12)	50.08 (14.82)	51.94 (15.71)	82.17 (15.44)	84.03 (15.45)	16.27 (9.22)	18.87 (12.06)	18.78 (10.35)	**15.2** (8.88)
50x10	26.83 (11.86)	47.02 (10.32)	40.87 (11.96)	62.65 (16.93)	79.81 (13.67)	8.02 (5.99)	14.2 (8.19)	10.59 (6.68)	**7.53** (5.39)
50x20	25.23 (6)	38.92 (11.39)	27.98 (6.69)	45.96 (13.94)	81.97 (12.53)	**5.63** (4.17)	17.07 (7.22)	8.14 (5.43)	7.91 (6.88)
100x5	35.81 (15.4)	52.73 (15.51)	52.33 (11.2)	84.38 (17.8)	77.95 (16.87)	23.88 (16.19)	24.02 (16.23)	**19.9** (14.05)	22.38 (14.35)
100x10	32.57 (9.04)	56.46 (14.64)	51.59 (13)	75.49 (16.51)	85.36 (11.14)	15.33 (8.89)	18.27 (8.14)	16.02 (10.16)	**12.76** (7.62)
100x20	31.18 (13.86)	51.55 (19.03)	39.15 (10.61)	62.76 (19.68)	86.45 (10.72)	11.94 (8.51)	21.75 (13.77)	11.92 (9.01)	**11.55** (9.26)
Average	28.9 (11.94)	44.76 (14.79)	36.61 (11.49)	68.99 (18.27)	80.28 (15.33)	12.53 (8.88)	16.99 (10.63)	14.07 (9.27)	**12.01** (8.38)

Table 7. Performance comparision results on instance group 4 (I= 4, Job-based correlation 0.25): mean and standard deviation of the RDI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	17.26 (7.26)	38.32 (13.71)	22.25 (11.47)	80.62 (20.86)	71.65 (16.55)	10.99 (8.71)	13.36 (7.62)	16.84 (9.88)	**8.82** (7.87)
20x10	38.73 (11.45)	45.15 (18.77)	18.17 (8.33)	71.64 (24.93)	81.39 (13.67)	**9.85** (6.76)	31.4 (10.42)	14.15 (7.05)	10.6 (6.8)
20x20	39.19 (11.23)	42.51 (13.64)	19.76 (9.95)	65.11 (15.1)	79.72 (16.02)	8.64 (5.16)	32.06 (12.7)	10.16 (5.97)	**8.03** (4.76)
50x5	27.92 (12.11)	38.94 (16.51)	43.4 (16.97)	67.59 (14.28)	83.01 (14.74)	10.53 (6.33)	10.11 (7.17)	12.61 (7.49)	**8.81** (6.14)
50x10	36.41 (9.44)	44.81 (11.35)	40.06 (8.68)	57.54 (12.27)	83.71 (14.24)	7.69 (4.89)	17.87 (8.07)	11.3 (5.79)	**7.53** (5.89)
50x20	35.85 (7.14)	39.51 (8.24)	35.28 (5.84)	47.44 (11.78)	84.44 (12.84)	**6.63** (4.53)	23.24 (6.95)	8.52 (4.51)	7.75 (4.97)
100x5	24.61 (9.85)	27.3 (9.56)	45.68 (21.61)	48.54 (19.88)	79.48 (16.07)	8.65 (7.34)	**7.6**(6.32)	10.93 (8.56)	8.78 (6.77)
100x10	35.13 (11.71)	43.5 (12.72)	57.4 (11.91)	51.97 (10.06)	85.88 (11.01)	10(5.72)	12.21 (6.63)	11.24 (6.96)	**9**(7)
100x20	36.46 (4.61)	41.28 (4.34)	44.79 (7.02)	44.29 (3.83)	91.09 (9.27)	**7.71** (5.25)	20.04 (5.13)	12 (6.1)	9.62 (5.83)
Average	32.4 (9.42)	40.15 (12.09)	36.31 (11.31)	59.42 (14.78)	82.26 (13.82)	8.97 (6.08)	18.65 (7.89)	11.97 (6.92)	**8.77** (6.23)

Table 8. Performance comparision results on instance group 5 (I = 5, Job-based correlation 0.75): mean and standard deviation of the RDI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	25.72 (11.25)	40.67 (15.49)	13.08 (9.49)	91.3 (14.41)	65.74 (20.79)	**7.95** (5.62)	18.36 (9.85)	8.1 (6.49)	8.99 (7.04)
20x10	24.72 (10.93)	32.01 (15.62)	18.26 (11.06)	90.6 (17.43)	65.47 (23.5)	6.89 (5.79)	14.77 (11.99)	**6.67** (5.85)	8.02 (7.6)
20x20	32.36 (13.82)	32.45 (14.9)	15.65 (7.49)	62.42 (20.77)	76.73 (17.89)	7.08 (5.56)	24.09 (13.21)	6.66 (4.75)	**6.34** (4.74)
50x5	37.84 (11.34)	45.49 (20.52)	41.17 (12.81)	75.34 (21.6)	81.42 (16.99)	7.97 (5.62)	13.85 (6.68)	**7.61** (6.38)	8.85 (6.06)
50x10	41.91 (9.27)	58.11 (18.58)	43.72 (11.75)	98.01 (4.54)	76.77 (15.81)	9.51 (5.55)	19.63 (7.27)	9.45 (5.05)	**7.15**(5.02)
50x20	49.15 (10.9)	47.36 (12.47)	54.29 (10.55)	75.86 (14.89)	84.86 (11.3)	9.48 (5.46)	20.26 (9.89)	8.65 (5.17)	**7.63** (6.1)
100x5	42.04 (19.21)	44 (20.28)	64.79 (20.02)	72.56 (22.12)	83.36 (12.59)	15.39 (9.22)	15.44 (10.52)	14.03 (9.99)	**11.35** (6.42)
100x10	48.84 (12.96)	61.29 (17.14)	48.78 (9.77)	80.09 (18.18)	79.75 (17.37)	8.38 (5.27)	19.76 (7.82)	8.36 (5.79)	**7.53** (5.22)
100x20	49.71 (3.62)	57.34 (14.15)	61.35 (13.15)	74.1 (15.98)	91.96 (6.64)	7.96(4.52)	23.74 (6.57)	7.69 (6.26)	**7.7** (5.71)
Average	39.14 (11.48)	46.52 (16.57)	40.12 (11.79)	80.03 (16.66)	78.45 (15.88)	8.96 (5.85)	18.88 (9.31)	8.58 (6.19)	**8.17** (5.99)

To provide statistical evidence for the conclusion of the comparison, we have performed a full-factorial ANOVA where I,n,m, instance and the type of algorithm are considered as factors. Yet, the normality assumption fails with a p-value < 0.005. For this reason, nonparametric methods are used as follows. The goal is to test whether GA_Yu is the best algorithm in comparison with the five state-of-art algorithms in each combination of $n \times m$ of each instance group. Given the RDI values of algorithms obtained in certain $n \times m$ combinations and correlation pattern, pairwise comparison is performed between GA_Yu and another algorithm, denoted as Alg, with the following hypothesis: $H_0 : \mu_{\text{RDI(GA\_YU)}} = \mu_{\text{RDI(Alg)}}$ and $H_1 : \mu_{\text{RDI(GA\_YU)}} < \mu_{\text{RDI(Alg)}}$, where μ stands for the true mean. Considering the non-normality of the RDI values, non-parametric method *Mann-Whitney U test* is used. If the obtained p-value is lower than the significance level, then H_0 is rejected and one can conclude GA_Yu is better then Alg in the case. Since we need to test if GA_Yu is better than

Table 9. Performance comparision results on instance group 1 (I= 1, random): mean and standard deviation of the RPI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	9 (4.35)	17.18 (10.47)	8.04 (5.99)	38.42 (21.23)	29.66 (14.3)	4.12 (4.49)	6.49 (5.05)	4.64 (3.99)	**4.11** (3.83)
20x10	9.04 (3.28)	13.57 (5.01)	6.75 (3.58)	21.04 (7.44)	22.5 (8.36)	**2.12** (1.75)	6.64 (2.63)	3.71 (2.93)	2.6 (2.36)
20x20	12.01 (2.92)	13.65 (3.41)	4.68 (2.26)	18.75 (4.35)	19.13 (5.06)	2.73 (1.67)	10.03 (3.26)	3.67 (1.76)	**2.29** (1.57)
50x5	39.58 (24.29)	56.06 (41.13)	64.51 (27.33)	87.82 (44.48)	119.76 (78.25)	**12.38** (10.93)	18.9 (11.8)	20.96 (12.19)	14 (10.99)
50x10	28.14 (9.76)	32.06 (12.48)	33.34 (11)	38.31 (13.23)	73.78 (24.23)	6.35 (6.15)	16.34 (7.81)	11.44 (6.85)	**6.11** (5.18)
50x20	21.37 (4.13)	21.88 (5.54)	20.21 (3.9)	25.83 (5.71)	46.62 (8.53)	**3.81** (2.7)	14.54 (4.87)	5.21 (3.07)	4.12 (3.24)
100x5	31.54 (32.33)	39.97 (33.1)	61.21 (32.97)	53.79 (33.89)	144.95 (159.61)	13.63 (11.86)	14.61 (17.48)	12.88 (9.73)	**11.89** (11.79)
100x10	23.89 (10.72)	28.8 (9.83)	37.04 (14.49)	35.2 (9.49)	78.72 (23.43)	6.96 (5.77)	10.94 (6.77)	9.09 (7.49)	**6.68** (5.51)
100x20	23.75 (4.54)	24.01 (5.83)	32.07 (5.22)	26.13 (5.7)	74.35 (13.73)	**5.4** (4.3)	14.35 (5.27)	8.07 (3.73)	5.96 (4.14)
Average	22.04 (10.7)	27.46 (14.09)	29.76 (11.86)	38.37 (16.17)	67.72 (37.28)	**6.39** (5.51)	12.54 (7.22)	8.85 (5.75)	6.42 (5.4)

Table 10. Performance comparision results on instance group 2 (I= 2, Machine-based correlation 0.25): mean and standard deviation of the RPI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	10.07 (5.82)	19.83 (9.96)	13.48 (7.01)	33.32 (12.87)	38.89 (20.29)	**4.14** (3.61)	6.22 (5.49)	9.64 (7.21)	5.81 (4.82)
20x10	8.44 (3.8)	12.35 (7.14)	8.43 (4.24)	21.85 (7.31)	27.33 (9.67)	3.59 (2.89)	4.78 (3.3)	5.23 (4.16)	**3.14** (2.57)
20x20	8.46 (4.22)	12.79 (5.32)	5.01 (1.62)	17.62 (6.19)	25.18 (6.2)	2.25 (1.68)	5.66 (4.16)	3.22 (1.58)	**2.18** (1.74)
50x5	24.26 (18.02)	36.49 (25.13)	39.27 (30.66)	58.55 (35.96)	70.24 (38.55)	**8.69** (7.44)	9.1 (8.83)	11.1 (9.69)	10.96 (13.53)
50x10	25.23 (8.2)	28.62 (6.42)	29.24 (11.13)	34.99 (7.52)	66.45 (20.28)	**6.36** (5.02)	14.24 (10.25)	8.95 (7.55)	6.51 (5.01)
50x20	20.07 (3.99)	22.96 (4.98)	21.26 (5.26)	26.68 (6.14)	51.41 (10.58)	**3.96** (2.79)	12.57 (3.98)	4.87 (3.29)	4.91 (3.86)
100x5	74.72 (168.37)	76.35 (155.62)	89.05 (148.95)	105.85 (182.57)	150.39 (287.64)	23.93 (49.34)	21.37 (46.72)	24.16 (48.28)	**17.14** (35.06)
100x10	31.57 (19.22)	37.57 (20.65)	46.38 (19.3)	45.23 (24.06)	78.3 (29.91)	**7.13** (5.61)	10.13 (8.55)	9.81 (6.89)	8.26 (6.2)
100x20	26.52 (7.57)	28.57 (7.33)	31.62 (6.35)	30.79 (7.16)	75.22 (18.12)	**5.09** (4.35)	16.35 (7.04)	6.65 (4.39)	7.08 (3.81)
Average	25.48 (26.58)	30.61 (26.95)	31.53 (26.06)	41.65 (32.2)	64.82 (49.03)	**7.24** (9.19)	11.16 (10.92)	9.29 (10.34)	7.33 (8.51)

Table 11. Performance comparision results on instances group 3 (I= 3, Machine-based correlation 0.75): mean and standard deviation of the RPI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	5.45 (3.74)	8.44 (4.06)	7.14 (5.41)	20.33 (8.07)	18.61 (9.71)	4.38 (4.75)	**1.98** (1.88)	5.53 (5.9)	4.36 (4.33)
20x10	5.1 (4.03)	8.55 (4.39)	5.29 (3.18)	16.48 (6.01)	17.55 (6.68)	2.26 (1.83)	3.5 (3.48)	3.2 (2.28)	**2.14** (1.69)
20x20	4.88 (3.09)	7.68 (4.89)	3.53 (1.41)	12 (5.13)	17.6 (5.88)	**1.37** (0.92)	3.71 (3.18)	2.08 (0.99)	1.39 (0.94)
50x5	19.96 (14.05)	21.58 (14.33)	21.38 (12.31)	33.52 (17.69)	34.72 (18.36)	6.68 (4.93)	7.91 (6.93)	8.04 (6.46)	**6.09** (4.77)
50x10	12.21 (9.83)	17.18 (8.85)	14.05 (6.08)	21.81 (9.82)	30.85 (18.33)	2.85 (2.73)	5.99 (5.23)	3.58 (2.77)	**2.72** (2.26)
50x20	8.52 (2.34)	12.7 (2.48)	9.27 (1.91)	15.04 (3.36)	27.92 (7.46)	**1.82** (1.27)	5.82 (2.59)	2.76 (1.96)	2.69 (2.38)
100x5	9.35 (5.76)	14.11 (11.09)	14.48 (10.71)	23.27 (17.91)	21.66 (16.66)	6.74 (7.57)	6.61 (7.91)	**6.01** (7.63)	6.19 (7.83)
100x10	8.65 (5.36)	13.47 (4.9)	12.89 (6.11)	18.9 (9.02)	22.39 (12.44)	3.66 (2.32)	4.61 (3.3)	3.86 (2.94)	**3.43** (2.89)
100x20	7.16 (4.14)	11.14 (4.42)	8.43 (2.33)	13.52 (4.44)	20 (8.89)	2.42 (1.49)	5.13 (4)	2.47 (1.7)	**2.34** (1.55)
Average	9.03 (5.82)	12.76 (6.6)	10.72 (5.49)	19.43 (9.05)	23.48 (11.6)	3.58 (3.09)	5.03 (4.28)	4.17 (3.63)	**3.48** (3.18)

Table 12. Performance comparision results on instance group 4 (I= 4, Job-based correlation 0.25): mean and standard deviation of the RPI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	8.07 (4.3)	17.39 (7.02)	10.18 (5.89)	39.42 (21.12)	34.1 (13.66)	4.52 (3.26)	6.62 (4.64)	7.23 (3.79)	**3.84** (3.28)
20x10	10.22 (3.39)	11.43 (4.13)	5.16 (2.82)	18.88 (7.13)	22.59 (8.97)	**2.77** (2.19)	8.17 (2.68)	3.89 (2.38)	2.94 (2.1)
20x20	7.84 (2.55)	8.47 (3.16)	3.78 (1.69)	13.03 (3.85)	15.79 (4.12)	1.69 (1)	6.46 (2.7)	1.96 (1.06)	**1.56** (0.91)
50x5	39.81 (35.15)	44.63 (30.15)	58.44 (48.31)	77.78 (44.74)	99.81 (63.82)	14.44 (14.23)	13.14 (15.09)	17.54 (17.64)	**10.44** (10.78)
50x10	25.74 (7.05)	31.79 (8.96)	28.3 (6.73)	40.59 (9.39)	59.3 (12.78)	5.39 (3.39)	12.84 (6.3)	8.01 (4.15)	**5.36** (4.41)
50x20	16.02 (2.54)	17.64 (3.16)	16.05 (3.62)	20.92 (3.13)	38.61 (9.24)	**3.01** (2.14)	10.33 (2.64)	3.94 (2.12)	3.52 (2.44)
100x5	150.43 (305.24)	131.94 (259.44)	167.79 (293.58)	169.09 (298.65)	221.19 (288.92)	**37.21** (79.5)	49.9 (112.47)	51.95 (101.93)	45.76 (100.5)
100x10	36.58 (19.84)	44.01 (20.85)	53.88 (16.95)	51.15 (20.62)	83.79 (31.82)	10.02 (7.43)	11.95 (8.25)	10.93 (8.53)	**8.55** (7.87)
100x20	28.29 (6.44)	31.73 (5.56)	34.32 (6.52)	34.02 (5.43)	70.3 (13.8)	**6.09** (4.51)	15.57 (4.87)	9.19 (4.69)	7.57 (5.21)
Average	35.89 (42.94)	37.67 (38.05)	41.99 (42.9)	51.65 (46.01)	71.72 (49.68)	**9.46** (13.07)	15 (17.74)	12.74 (16.25)	9.95 (15.28)

all other 5 algorithms, this is actually a multiple testing problem. To control the *familywise error*, the Holm's familywise p-value (Holm, 1978), denoted as ξ, is calculated. If ξ is smaller than the significance level (set as 0.05), we can conclude that GA_Yu is the best algorithm in the case. We have calculated the ξ value in the cases of all $n \times m$ combinations of each instance group, as shown in Table 14. As we found, GA_Yu is statistically better than all the other 5 algorithms in all tested cases.

5.3.3. Comparison with variants of the proposed algorithm

In GA_Yu, the proposed DS decoding method is the main contribution. To study its effectiveness, one idea is to see how the algorithm performs without it but with other existing approaches. Meanwhile, as discussed before, the NEH approach is used as initialization technique in many other

Table 13. Performance comparision results on instance group 5 (I = 5, Job-based correlation 0.75): mean and standard deviation of the RPI indicator

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	6.1 (2.72)	9.68 (3.87)	3.01 (1.98)	22.12 (6.43)	15.3 (4.57)	**1.85** (1.27)	4.37 (2.44)	1.9 (1.58)	2.11 (1.54)
20x10	3.06 (1.17)	4.04 (1.81)	2.28 (1.19)	13.58 (7.85)	8.27 (1.96)	**0.88** (0.68)	1.84 (1.24)	0.88 (0.71)	0.99 (0.85)
20x20	1.93 (0.88)	1.95 (1.02)	0.97 (0.55)	3.68 (1.35)	4.66 (1.53)	0.42 (0.34)	1.42 (0.83)	0.39 (0.27)	**0.38** (0.31)
50x5	29.32 (13.36)	35.53 (18.86)	32.12 (14.29)	57.98 (23.83)	59.72 (17.01)	6.43 (5.21)	10.86 (6.51)	**6.22** (6.01)	7.05 (5.25)
50x10	16.42 (5.85)	22.27 (7.45)	17.16 (7.03)	38.93 (11.69)	29.38 (7.39)	3.77 (2.62)	7.76 (3.55)	3.74 (2.3)	**2.95** (2.37)
50x20	7.13 (1.44)	6.89 (1.74)	7.97 (1.81)	11.03 (2.05)	12.41 (1.98)	1.36 (0.76)	2.91 (1.34)	1.29 (0.81)	**1.1** (0.86)
100x5	32.46 (25.93)	33.72 (22.84)	46.11 (23.1)	54.01 (31.22)	57.81 (27.63)	11.78 (10.06)	11.77 (11.3)	10.1 (10.59)	**8.23** (6.7)
100x10	34.14 (11.46)	42.31 (12.96)	33.79 (8.91)	55.34 (14.77)	55.11 (13.9)	5.96 (4.08)	13.91 (6.32)	5.99 (4.61)	**5.18** (3.74)
100x20	15.32 (2.66)	17.75 (5.25)	18.84 (4.47)	22.87 (6.61)	28.42 (5.33)	2.44 (1.47)	7.3 (2.43)	2.4 (2.03)	**2.34** (1.78)
Average	16.21 (7.27)	19.35 (8.42)	18.03 (7.04)	31.06 (11.76)	30.12 (9.03)	3.88 (2.94)	6.9 (4)	3.66 (3.21)	**3.37** (2.6)

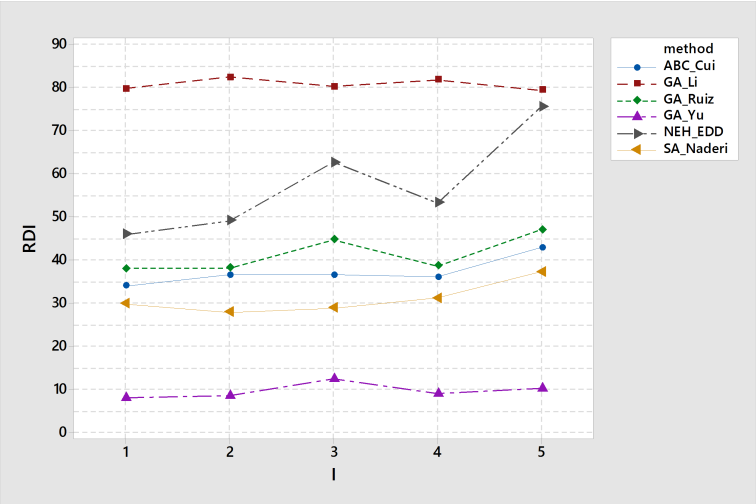


Figure 7. Algorithm comparison on instances with different operation processing correlation pattern. Each point in the figure represents the averaged RDI values of the algorithm over 450 experiments for the given I (3 number of jobs · 3 number of stages · 10 instances · 5 replicates = 450 experiments)

Table 14. Holm's familywise p-value ξ calculated for each case with the assumption that GA_Yu is better than all the state-of-art algorithms under comparison with time factor $\tau = 10$

	I				
n x m	1	2	3	4	5
20x5	7.79E-06	1.74E-08	3.40E-03	1.11E-04	1.29E-04
20x10	8.36E-12	2.36E-09	7.59E-06	7.33E-07	2.20E-10
20x20	3.48E-06	1.40E-12	2.30E-10	6.39E-10	8.94E-12
50x5	7.97E-17	2.33E-13	2.38E-14	1.26E-13	3.13E-17
50x10	1.68E-17	1.68E-17	1.26E-13	1.69E-17	1.41E-17
50x20	1.68E-17	1.69E-17	1.69E-17	1.68E-17	1.73E-17
100x5	4.72E-03	1.19E-06	8.25E-05	6.91E-12	1.20E-08
100x10	1.13E-16	9.68E-16	1.34E-12	5.30E-15	1.40E-17
100x20	2.01E-17	1.70E-17	4.72E-12	1.68E-17	1.69E-17

researches. It is therefore interesting to study if GA_Yu can perform better with NEH approach. To this end, several variations of GA_Yu created as follows:

(1) GA_Yu(PS): Replacing the DS decoding method in GA_Yu with the PS used by GA_Ruiz
(2) GA_Yu(LS): Replace the DS decoding method in GA_Yu with the LS used by ABC_Cui
(3) GA_Yu(INEH_EDD): Add the solution given by an improved version NEH_EDD(DS) to the initial population. We improve the NEH_EDD to tackle the selection difficulties of sub-schedules as following: we use the total lateness as a secondary criterion to choose the sub-schedules when they have identical total tardiness.

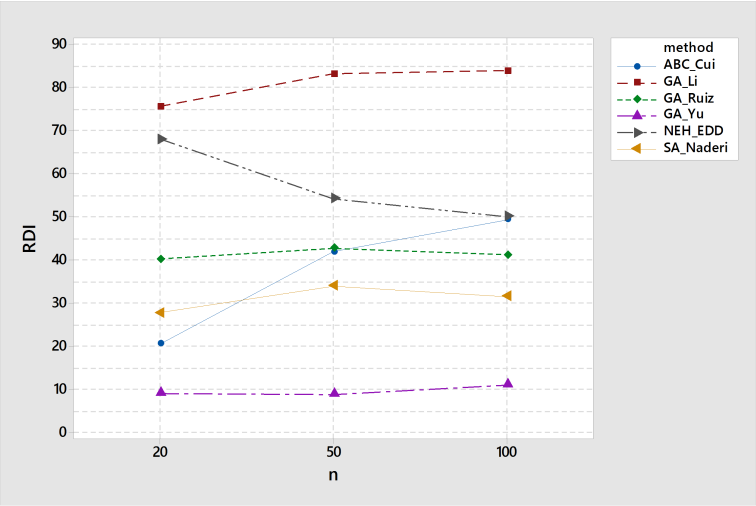


Figure 8. Algorithm comparison on instances with different number of jobs. Each point in the figure represents the averaged RDI values of the algorithm over 750 experiments for the given n (3 number of stages · 5 correlation patterns · 10 instances · 5 replicates = 750 experiments)

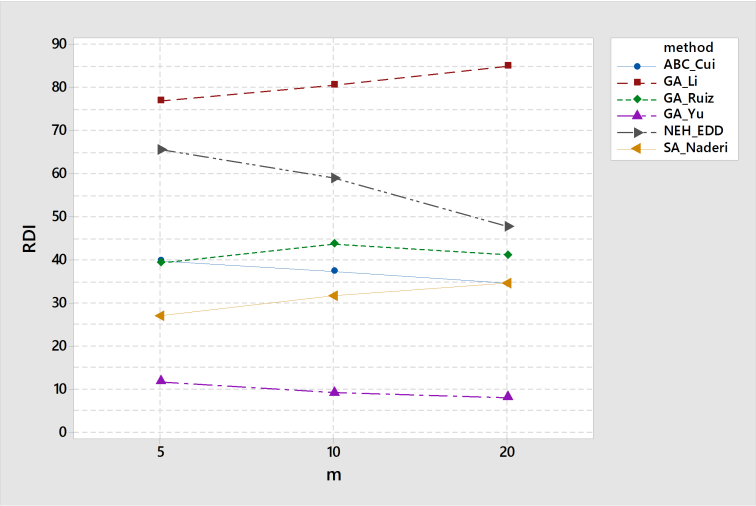


Figure 9. Algorithm comparison on instances with different number of stages. Each point in the figure represents the averaged RDI values of the algorithm over 750 experiments for the given m (3 number of jobs · 5 correlation patterns · 10 instances · 5 replicates = 750 experiments)

All these variants of GA_Yu use the same parameters and termination condition as GA_Yu.

It can be seen from Table 9-13 that:(1) when using different decoding methods, the performance of GA_Yu are different. Generally, the proposed DS leads to the best performance, followed by LS, and the worst one is PS. However, there are cases where LS outperforms the other two (see Table 13) when high job-based correlation exists; (2) when using the improved NEH_EDD to generate initial solution, the performance of GA_Yu deteriorates in many cases; while in the cases of improvement, the increment of is not so significant. Indeed, given fixed computational resources, when the converging speed of GA_Yu is fast enough, allocating computational resource to the NEH_EDD approach could be of no advantage and sometimes of disadvantage.

In Figure 10-12 are plot the averaged RPI values of the GA_Yu with different decodings in various cases of I, n, m, respectively. Some observations are as follows:

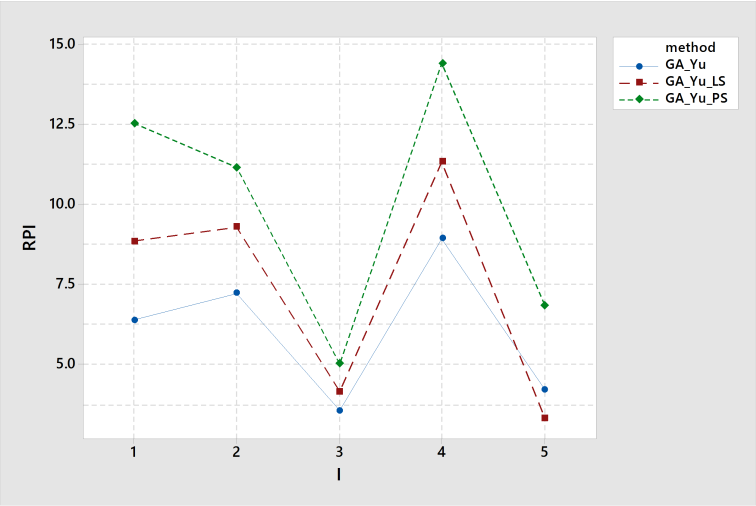


Figure 10. Comparison between GA_Yu and its decoding variants on instances with different operation processing correlation pattern (RPI)

- Figure 10 shows the advantage of DS over PS are less in the case of machine-based correlation processing times (I=3). The explanation could be as below. The advantage of DS comes from the elimination of unnecessary machine idleness. Such idleness is introduced in PS when a high-prioritized job j arrives at the stage, say i, later than a low-prioritized job k, which we call the "*Overturn*" phenomenon. *Overturn* happens when job j and k are assigned to different machines in stage $i-1$, and the processing time of j is much longer than that of k. If machine-based correlation exists, generally, the higher this correlation, the less the difference between the processing time of j and k on the same machines. Also, under a greedy machine assignment rule, jobs have higher probability to be assigned to the same machine which offers a low processing time basis for all the jobs. As a consequence, the occurring of *Overturn* is reduced, and hence PS may introduce less unnecessary idleness and get closer performance to DS.
- Figure 10 shows the LS outperforms DS as high job-based correlation exists. Note that the advantage of DS over LS derives from a better controllability by re-sequencing the jobs waiting in the machine buffers using chromosome information. For this phenomenon one speculation is that due to the job-based correlation feature, the LS is able to obtain as high as controllability with DS even without the re-sequencing mechanism, and due to a lower time complexity which allows GA to have more evaluations on the chromosomes, LS outperforms DS. Yet this requires future studies.
- Figure 12 shows that DS and LS are more efficient comparing to PS when the number of stages is large. Indeed, when PS is used to construct the schedule, machine idleness are introduced during the coupling of job sequence between any pair of consecutive stages, and consequently, the longer the shop is, the more machine idleness are introduced, and the more inefficient PS is.

5.3.4. Comparison with different computing budget

It is well-known that the performance of metaheuristics are related to the computing budget. In Figure 13 are reported the total tardiness values obtained by different algorithms on two specific instances as the computing budget increases. Denote t_{neh} as the computation time of NEH_EDD uses for a case. For the algorithms which use NEH_EDD in the initialization phase, before t_{neh},

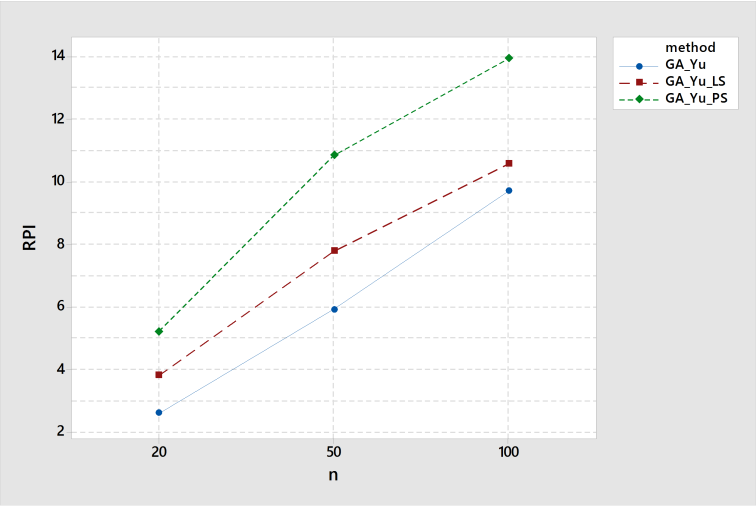


Figure 11. Comparison between GA_Yu and its decoding variants on instances with different number of jobs (RPI)

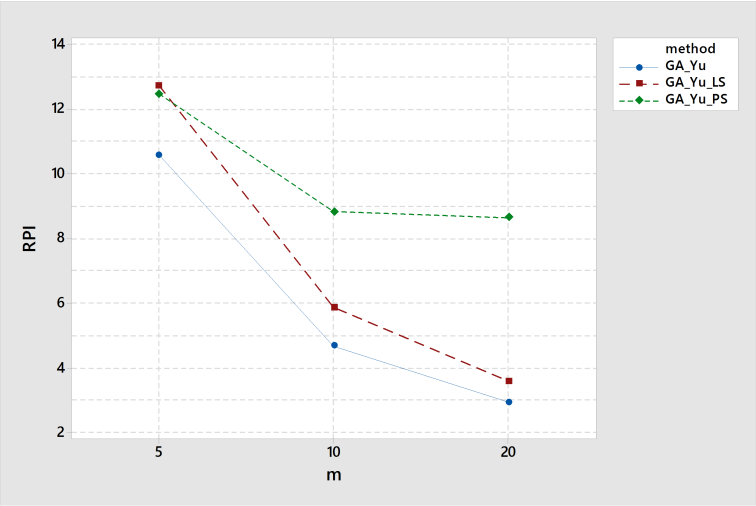
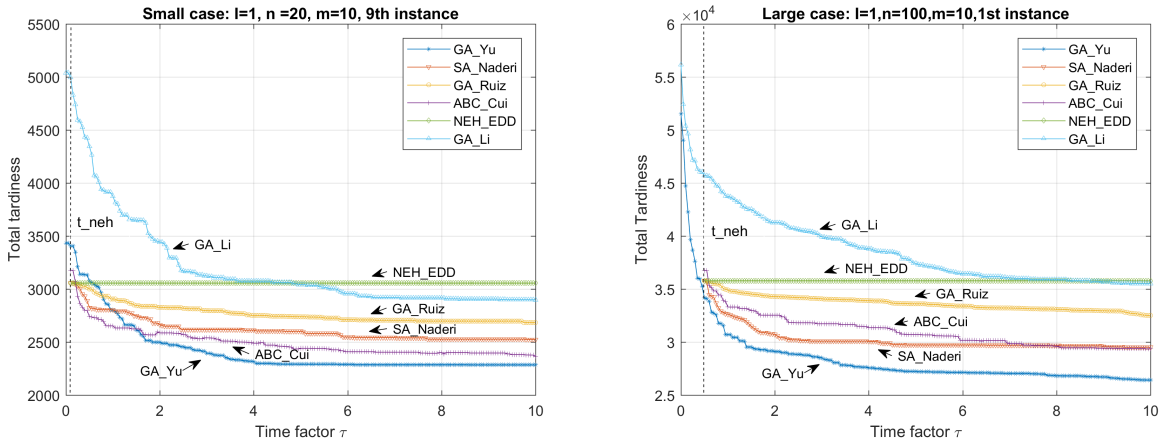


Figure 12. Comparison between GA_Yu and its decoding variants on instances with different number of stages (RPI)

the total tardiness is null. As shown, in the small case, GA_Yu starts at a higher value than those using NEH_EDD for initialization, but converges quite fast and exceeds the other algorithms before $\tau = 2$; whilst in the large case, GA_Yu is able to converge to a better result than the NEH_EDD in a time shorter than t_{neh}.

In Table 15 and 16 are reported the RPI values of comparisons on group $I = 1$ with time factor $\tau = 2$ and $\tau = 5$, respectively. In both cases, GA_Yu is the best on average. It is observed that as the computing budget increases, the gap between GA_Yu and the state-of-art algorithms becomes larger, which implies GA_Yu converges faster. To show this trend better, in Figure 14 is plotted the averaged RPI values of all algorithms on group 1 instances, calculated by gathering all data with $\tau = 1, 2, \ldots, 10$.



(a) Small case (b) Large case

Figure 13. Convergence curve comparison. In horizontal axis: the time factor τ is used to represent the CPU time consumed by the algorithms, where $t_{CPU} = \tau \times n^2 \times (m/2)$ ms. In vertical axis: the total tardiness value is an average from 5 replications obtained by the algorithm at the given CPU time

Table 15. Performance comparision results on instance group 1 (I= 1, random): mean and standard deviation of the RPI indicator (time factor $\tau = 2$)

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	9.99 (5.63)	17.5 (10.69)	10.92 (7.54)	37.79 (21.29)	32.67 (15.48)	4.14 (4.68)	6.64 (4.91)	4.31 (4.16)	**4.03** (3.82)
20x10	10.31 (3.71)	14.39 (4.36)	8.38 (3.64)	20.21 (7.3)	28.17 (9.09)	**2.33** (1.64)	7.58 (2.91)	3.58 (2.86)	2.74 (2.17)
20x20	11.19 (2.73)	12.85 (2.86)	5.42 (2.6)	16.3 (3.52)	24.29 (7.35)	**2.57** (2.06)	11.37 (3.49)	3.56 (2.28)	3.1 (1.94)
50x5	16.39 (12.26)	32.64 (20.9)	41.72 (12.7)	45 (19.65)	96.52 (60.17)	**10.73** (7.66)	20.45 (11.32)	16.46 (6.92)	13.44 (7.73)
50x10	9.58 (9.75)	11.88 (10.45)	20.46 (9.8)	14.15 (11.1)	58.43 (17.4)	**6.22** (4.88)	16.58 (9.39)	7.66 (6.59)	6.67 (6.08)
50x20	10.96 (3.04)	10.91 (3.16)	15.1 (3.57)	12.3 (3.36)	47.18 (8.69)	**4.18** (2.42)	16.35 (3.82)	4.03 (2.61)	5.45 (3.2)
100x5	**11.93** (10.73)	18.91 (14.86)	42.39 (11.16)	23.34 (14.35)	114.09 (99.92)	18.72 (9.06)	19.95 (11.53)	15.21 (9.26)	18.41 (11.98)
100x10	8.41 (5.81)	11.79 (8.7)	22.29 (8.16)	13.58 (8.82)	61.46 (11.38)	6.05 (4.05)	13.01 (5.45)	**5.89** (4.06)	10.71 (5.66)
100x20	7.42 (3.64)	7.53 (3.9)	18.94 (3.88)	8.03 (3.91)	59.54 (8.99)	**3.53** (2.51)	14.97 (4.25)	3.74 (2.42)	7.52 (3.42)
Average	10.69 (6.37)	15.38 (8.88)	20.62 (7.01)	21.19 (10.37)	58.04 (26.5)	**6.5** (4.33)	14.1 (6.34)	7.16 (4.57)	8.01 (5.11)

Table 16. Performance comparision results on instance group 1 (I= 1, random): mean and standard deviation of the RPI indicator (time factor $\tau = 5$)

Instance	SA_Naderi	GA_Ruiz	ABC_Cui	NEH_EDD	GA_Li	GA_Yu	GA_Yu(PS)	GA_Yu(LS)	GA_Yu(INEH)
20x5	9.72 (5.51)	17.31 (10.79)	9.68 (7.48)	38.08 (21.39)	31.62 (15.18)	4.04 (4.42)	6.49 (5.01)	4.46 (4.2)	**3.92** (3.75)
20x10	9.85 (3.58)	14.02 (4.29)	7.35 (3.66)	20.68 (7.26)	25.33 (9)	**2.19** (1.69)	7.04 (2.99)	3.51 (2.89)	2.54 (2.42)
20x20	12.23 (2.89)	13.68 (3.23)	5.47 (2.66)	18.02 (4.25)	21.43 (6.04)	2.56 (1.7)	10.2 (3.2)	3.46 (1.86)	**2.48** (1.83)
50x5	26.68 (16.84)	42.71 (31.06)	51.32 (19.04)	63.8 (30.73)	107.67 (75.8)	**7.85** (7.41)	15.8 (10.36)	16.61 (8.6)	10.76 (7.37)
50x10	19.73 (8.13)	23.41 (12.28)	27.98 (9.99)	27.54 (13.15)	66.53 (20.58)	**6.26** (4.85)	15.57 (8.2)	10.37 (6.91)	6.9 (4.9)
50x20	17.17 (4.27)	17.57 (5.85)	18.36 (4.82)	20.28 (5.62)	46.72 (7.84)	4.94 (3.36)	15.2 (5.13)	**4.05** (3.14)	5.23 (2.4)
100x5	20.81 (19.68)	30.56 (26.74)	53.11 (20.26)	38.65 (25.81)	123.24 (124.67)	15.31 (9.09)	16.35 (12.49)	**12.76** (8.76)	14.2 (7.17)
100x10	16.06 (6.33)	20.83 (7.59)	31 (11.19)	24.59 (7.51)	69.68 (16.21)	**5.71** (3.87)	11.49 (5.64)	6.79 (4.28)	6.14 (4.6)
100x20	15.18 (2.77)	15.22 (3.53)	25.76 (3.81)	16.48 (3.54)	65.93 (11)	**3.58** (2.18)	12.9 (4.69)	4.19 (2.7)	4.7 (2.78)
Average	16.38 (7.78)	21.7 (11.71)	25.56 (9.21)	29.79 (13.25)	62.02 (31.81)	**5.83** (4.29)	12.34 (6.41)	7.36 (4.82)	6.32 (4.14)

6. Conclusion

In this study we have proposed a genetic algorithm to solve the hybrid flow shop scheduling problem with unrelated machines and machine eligibility constraint to minimize the total tardiness. The main contribution of this research is a new decoding method aiming at improving the schedule construction procedure from a given job permutation. More specifically, by analyzing two widely used decoding methods, permutation scheduling and list scheduling, we discover disadvantages when they are applied to due-date related objective. Then, we propose the dynamic scheduling which uses a machine seize mechanism to avoid tightness problem and job re-sequencing strategy in machine buffers to mitigate the controllability problem. The proposed decoding method is incorporated into a genetic algorithm. The proposed GA has been calibrated and compared to five

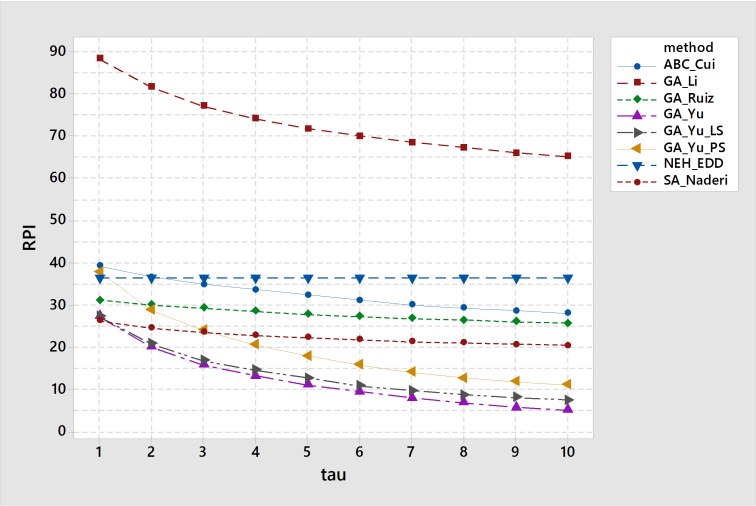


Figure 14. Comparison of the averaged RPI values on group 1 instances with different time factor τ

calibrated state-of-art algorithms on a set of 450 instances with different problem size and operation processing time correlation patterns. The computational results show that the proposed GA outperforms the state-of-art algorithms in almost all the cases. Finally, we have verified the superiority of the proposed decoding method by comparing the performance of the genetic algorithm when coupled with different decoding methods. Also, we have shown that the use of NEH_EDD approach as initialization would not benefit the proposed GA on the cases at hand.

The proposed GA can be extended to solve hybrid flow shop scheduling problem with more realistic assumptions such as sequence-dependent setups, job release date, finite intermediate buffer capacity, and to handle multi-objective problems.

References

Alfieri A. Workload simulation and optimisation in multi-criteria hybrid flowshop scheduling: a case study. International Journal of Production Research, 2009, 47(18): 5129-5145.

Adler L, Fraiman N, Kobacker E, et al. BPSS: a scheduling support system for the packaging industry. Operations Research, 1993, 41(4): 641-648.

Botta-Genoulaz V. Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. International Journal of Production Economics, 2000, 64(1): 101-111.

Chamnanlor C, Sethanan K, Gen M, et al. Embedding ant system in genetic algorithm for re-entrant hybrid flow shop scheduling problems with time window constraints. Journal of Intelligent Manufacturing, 2015: 1-17.

Chen C L, Chen C L. Bottleneck-based heuristics to minimize total tardiness for the flexible flow line with unrelated parallel machines. Computers & Industrial Engineering, 2009, 56(4): 1393-1401.

Chen L, Bostel N, Dejax P, et al. A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. European Journal of Operational Research, 2007, 181(1): 40-58.

Choi S W, Kim* Y D, Lee G C. Minimizing total tardiness of orders with reentrant lots in a hybrid flowshop. International Journal of Production Research, 2005, 43(11): 2149-2167.

Cui Z, Gu X. An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. Neurocomputing, 2015, 148: 248-259.

Engin O, Ceran G, Yilmaz M K. An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Applied Soft Computing, 2011, 11(3): 3056-3065.

Fanjul-Peyro L, Ruiz R. Iterated greedy local search methods for unrelated parallel machine scheduling. European Journal of Operational Research, 2010, 207(1): 55-69.

Gupta J N D. Two-stage, hybrid flowshop scheduling problem. Journal of the Operational Research Society, 1988, 39(4): 359-364.
Goldberg D E, Lingle R. Alleles, loci, and the traveling salesman problem. Proceedings of an international conference on genetic algorithms and their applications. Lawrence Erlbaum, Hillsdale, NJ, 1985, 154: 154-159.
Hunsucker J L, Shah J R. Performance of priority rules in a due date flow shop. Omega, 1992, 20(1): 73-89.
Holland J H. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, 1992.
Holm S. A simple sequentially rejective multiple test procedure. Scandinavian journal of statistics, 1979: 65-70.
Jungwattanakit J, Reodecha M, Chaovalitwongse P, et al. A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. Computers & Operations Research, 2009, 36(2): 358-378.
Kurz M E, Askin R G. Scheduling flexible flow lines with sequence-dependent setup times. European Journal of Operational Research, 2004, 159(1): 66-82.
Lee G C, Kim Y D, Choi S W. Bottleneck-focused scheduling for a hybrid flowshop. International Journal of Production Research, 2004, 42(1): 165-181.
Li D, Meng X, Liang Q, et al. A heuristic-search genetic algorithm for multi-stage hybrid flow shop scheduling with single processing machines and batch processing machines. Journal of Intelligent Manufacturing, 2015, 26(5): 873-890.
Li J, Pan Q, Duan P. An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping. IEEE transactions on cybernetics, 2016, 46(6): 1311-1324.
Liao C J, Tjandradjaja E, Chung T P. An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. Applied Soft Computing, 2012, 12(6): 1755-1764.
Michalewicz Z, Hartley S J. Genetic algorithms+ data structures= evolution programs. Mathematical Intelligencer, 1996, 18(3): 71.
Marichelvam M K, Prabaharan T, Yang X S. A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. IEEE transactions on evolutionary computation, 2014, 18(2): 301-305.
Marichelvam M K, Prabaharan T, Yang X S. Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. Applied Soft Computing, 2014, 19: 93-101.
Naderi B, Zandieh M, Balagh A K G, et al. An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. Expert systems with Applications, 2009, 36(6): 9625-9633.
Nawaz M, Enscore E E, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 1983, 11(1): 91-95.
Oĝuz C, Ercan M F. A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. Journal of Scheduling, 2005, 8(4): 323-351.
Pan Q K, Ruiz R. An estimation of distribution algorithm for lot-streaming flow shop problems with setup times. Omega, 2012, 40(2): 166-180.
Pan Q K, Ruiz R, Alfaro-Fernández P. Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. Computers & Operations Research, 2017, 80: 50-60.
Paul R J. A production scheduling problem in the glass-container industry. Operations Research, 1979, 27(2): 290-302.
Pargar F, Zandieh M. Bi-criteria SDST hybrid flow shop scheduling with learning effect of setup times: water flow-like algorithm approach. International Journal of Production Research, 2012, 50(10): 2609-2623.
Pinedo M. Scheduling. New York: Springer, 2012.
Rashidi E, Jahandar M, Zandieh M. An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. The International Journal of Advanced Manufacturing Technology, 2010, 49(9): 1129-1139.
Ribas I, Leisten R, Framian J M. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers & Operations Research, 2010, 37(8): 1439-1454.
Ruiz R, Maroto C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research, 2006, 169(3): 781-800.
Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling

problem. European Journal of Operational Research, 2007, 177(3): 2033-2049.
Ruiz R, Stützle T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. European Journal of Operational Research, 2008, 187(3): 1143-1159.
Ruiz R, Vázquez-Rodríguez J A. The hybrid flow shop scheduling problem. European Journal of Operational Research, 2010, 205(1): 1-18.
Semeraro Q. Heuristic scheduling in a flow shop type manufacturing cell. Manufacturing systems, 1983, 12(3): 197-208.
Şen H, Bülbül K. A strong preemptive relaxation for weighted tardiness and earliness/tardiness problems on unrelated parallel machines. INFORMS Journal on Computing, 2015, 27(1): 135-150.
Tavakkoli-Moghaddam R, Safaei N, Sassani F. A memetic algorithm for the flexible flow line scheduling problem with processor blocking. Computers & Operations Research, 2009, 36(2): 402-414.
Urlings T, Ruiz R, Serifoglu F S. Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. International Journal of Metaheuristics, 2010, 1(1): 30-54.
Urlings T, Ruiz R, Sttzle T. Shifting representation search for hybrid flexible flowline problems. European Journal of Operational Research, 2010, 207(2): 1086-1095.
Voß S, Witt A. Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. International journal of production economics, 2007, 105(2): 445-458.
Vignier A, Billaut J C, Proust C. Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art. RAIRO-Operations Research, 1999, 33(2): 117-183.
Wang X, Tang L. A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. Computers & Operations Research, 2009, 36(3): 907-918.
Wolpert D H, Macready W G. No free lunch theorems for optimization. IEEE transactions on evolutionary computation, 1997, 1(1): 67-82.
Xiao W, Hao P, Zhang S, et al. Hybrid flow shop scheduling using genetic algorithms. Intelligent Control and Automation, 2000. Proceedings of the 3rd World Congress on. IEEE, 2000, 1: 537-541.
Yaurima V, Burtseva L, Tchernykh A. Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers. Computers & Industrial Engineering, 2009, 56(4): 1452-1463.
Ying K C, Lin S W. Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach. International Journal of Production Research, 2006, 44(16): 3161-3177.
Zandieh M, Ghomi S M T F, Husseini S M M. An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. Applied Mathematics and Computation, 2006, 180(1): 111-127.