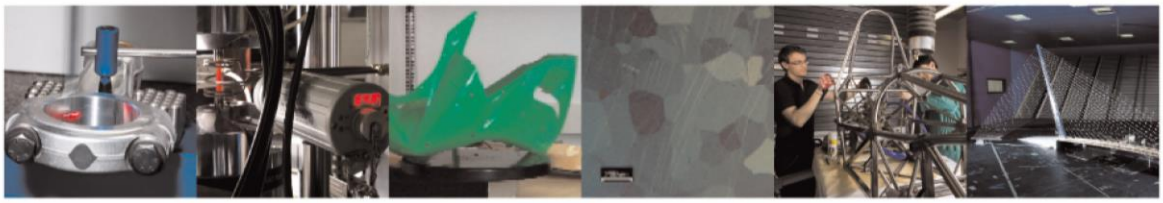


Multi-objective scheduling in hybrid flow shop: Evolutionary algorithms using multi-decoding framework

Chunlong Yu, Pietro Andreotti, Quirico Semeraro

This is a post-peer-review, pre-copyedit version of an article published in Computers & Industrial Engineering. The final authenticated version is available online at: http://dx.doi.org/10.1016/j.cie.2020.106570

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

Multi-objective scheduling in hybrid flow shop: Evolutionary algorithms using multi-decoding framework

Chunlong Yu

Département de mathématiques et génie industriel, École Polytechnique de Montréal, Montréal, Canada

Quirico Semeraro, Pietro Andreotti

Dipartimento di Meccanica, Politecnico di Milano, Milano, Italy

Abstract

Hybrid flow shops are common manufacturing environments applied in many industrial fields. This paper tackles the scheduling problem in hybrid flow shop with unrelated machines, machine eligibility and sequence-dependent setup times (SDST) to minimize the bi-criteria of total tardiness and total setup time. Evolutionary algorithms (EAs) are adopted to solve the problem. Firstly, four efficient decoding algorithms using different machine selection rules are developed for constructing a schedule from a job permutation. These decoding algorithms are able to map the job permutation space to distinct regions in the objective space. Then, we propose a multi-decoding framework (MDF) for taking advantage of multiple decoding algorithms along one evolution path. The hybridization of MDF and EAs leads to the hyper-heuristic approach. The proposed MDF is coupled with a genetic algorithm to solve the problem in "a priori" approach, that is, to optimize a convex combination of the objectives given user preference information. The framework is also embedded to a multi-objective genetic algorithm, known as NSGA-II, to solve the problem in "a posteriori" approach, which aims at approximating the Pareto-optimal set for the user to make posterior decisions. The efficiency of the proposed methods is validated by numerical results. More specifically, when "a priori" approach is used, the proposed MDF helps EAs adjusting the adopted decoding scheme and generating solution aligned to the user preference; when "a posteriori" approach is applied, the MDF extends the search space and improves the solution quality.

Keywords: Scheduling; multi-objective optimization; hybrid flow shop; hyper-heuristic; genetic algorithm; evolutionary algorithm.

1. Introduction

Hybrid flow shops (HFS) are common manufacturing environments in which a set of n jobs are to be processed in a series of m stages [1]. The HFS scheduling problem has attracted significant research attention due to its wide application in a variety of industrial fields such as the electronics, paper, textile, pharmaceutical and sheet metal industry. The scheduling problem aims at the creation of a production Gantt chart to optimize certain objective functions. To this end, two types of decisions are to be made jointly: assigning jobs to machines, and sequencing jobs on

*Corresponding author
Email address: chunlong.yu@polimi.it (Chunlong Yu)

the assigned machine. This is not a trivial task. In fact, even for a simple two-stage HFS, the scheduling problem is already NP-hard [2].

In the literature, the majority of researches on HFS scheduling consider only one objective function [1]. However, in manufacturing, important criteria such as throughput, due-date performance and production costs are closely associated with the production schedule and should be considered simultaneously. For this reason, the trend of considering multiple objectives in the HFS scheduling is increasing. Dugardin et al. [3] studied a HFS with reentrant jobs to minimize makespan and maximize the bottleneck machine utilization. The same objective functions were also considered in Wang and Liu [4]. Karimi et al. [5] tackled the scheduling problem with makespan and total tardiness as objectives. In Lu et al. [6], the authors studied a scheduling problem in a real-world welding industry to minimize the makespan, machine load and instability simultaneously. Recently, with the growing research interest in green manufacturing, researchers are trying to save energy consumption via proper scheduling. Luo et al [7] optimized the makespan as well as the electricity consumption during the machining process in a HFS with uniform parallel machines. Zeng et al [8] investigated the manufacturing process in a real-world tissue paper mill company, and proposed an efficient solution for reducing electricity consumption, material wastes and makespan.

In this work, we consider a HFS with sequence-dependent setup times (SDST), unrelated machines and machine eligibility. The objective is to minimize the total tardiness and total setup time. Unrelated machines assumption indicates that the parallel machines in a stage could be different in terms of, e.g., processing technology, brand, model and condition. Thus, the processing time of a job is not necessary equal on the parallel machines. Machine eligibility represents the fact that not all parallel machines are eligible to process a job because of certain technological constraint. These two characteristics correspond to the most general situation of the parallel machines in a stage [9]. For the objective functions, the total tardiness concerns the due-date performance and acts as a critical factor for customer satisfaction, especially in Make-to-order environment. On the other hand, setups are non-value-added activities during manufacturing and are usually associated with certain costs. Setups could be the activities like cleaning the dyeing tank before dyeing a yarn in the textile manufacturing, switching the tool set of the punching machine in the sheet metal manufacturing, etc. The setup time is usually sequence-dependent, i.e., it depends on the similarity of the previously processed job and the current job. It is therefore important to decide properly the job processing sequence to avoid unnecessary setups. The importance of considering setups in scheduling is emphasized by Allahverdi [10], and it has been shown in Trovinger and Bohn [11] that by reducing setup time, a direct saving of 1.8 million per year was obtained in a printed circuit board assembly plant. Considering the total tardiness and total setup time simultaneously allows the manufacturing companies to respect due-dates meanwhile reducing non-value-added activities. As far as we know, this problem has not been tackled in the literature.

We propose evolutionary algorithms (EAs) to solve the scheduling problem in both *a priori* and *a posteriori* fashion. A priori approach refers to the optimization of a scalarized objective function with user preference input; *a posteriori* approach pursues the generation of a set of Pareto non-dominated solutions, from which the user selects the final solution. The contributions of this research are:

- We develop several decoding algorithms for the HFS with sequence-dependent setup times. These build the complete schedule from a given job permutation using different machine selection rules in the simulation. Compared to the conventional decoding method, they are able to map the design space to different portions of the objective space which are more favorable to the total setup time objective, and thus provides better options for companies focusing on setup reduction.

- When more than one objective functions are considered, the superiority of certain decoding algorithm is dependent on the user preference. For this reason, we develop a multi-decoding framework (MDF) that allows using multiple decoding algorithms in EAs. The hybridization of MDF and EAs leads to the hyper-heuristic paradigm. For *a priori* approach, EAs using the proposed MDF is able to adjust the adopted decoding algorithm during the evolution, and provides efficient solution aligned to the user preference; whilst when *a posteriori* approach is adopted, the MDF extends the search region in the objective space and helps EAs to generate a better non-dominated solution set.

The main body of the paper is organized as follows. In Section 2, we provide a problem description and a mix-integer linear programming model for the problem. In Section 3, a literature review is given. In Section 4, we present the proposed methods. Firstly, four decoding algorithms are developed, then we report the multi-decoding framework. In Section 5, the proposed decoding algorithms and the MDF are coupled with EAs to solve the multi-objective HFS scheduling problem. We validate the efficiency of the proposed methods by numerical experiments. Finally, Section 6 draws the conclusions.

2. Problem description

2.1. The problem

There are n jobs to be processed at m production stages sequentially from stage 1 to stage m. The i-th stage consist of h_i unrelated machines, and for at least one stage $h_i > 1$. Each job j consists of a sequence of m operations. The execution of the i-th operation of job j requires one machine out of an eligible machine set $E_{ij} \subseteq \{1, \ldots, h_i\}$ at stage i. Parameter p_{ilj} is the processing time of job j on the l-th machine at stage i. S_{iljk} is the machine-based sequence-dependent setup time on the l-th machine at stage i when processing job k, after having processed job j. The setup time is anticipatory, i.e., the setup might be done before the job is released at the previous stage. Any job, say j, has a release date $r_j = 0$ and a nonzero due date d_j. Let C_{ij} be the completion time of job j at stage i, the tardiness of job j is given by $T_j = \max\{C_{mj} - d_j, 0\}$. The objective is to minimize the following two objectives:

- f_1 total tardiness
- f_2 total setup time

Other assumptions are as below:

- the processing of the jobs cannot be preempted;

- machines are reliable and no machine failures can happen;
- each machine has an upstream buffer of unlimited capacity;
- each machine can process only one job at a time, and each job can be processed on only one machine at a time;
- job transportation time between machines is neglected.

The scheduling problem can be denoted using a triplet $\alpha|\beta|\gamma$ notation as in Ruiz and Vázquez-Rodríguez[1]. The described scheduling problem is denoted as:

$$FHm, ((RM^{(i)})_{i=1}^{m})|M_j, S_{sd}|\sum T_j, TST.$$

Here, FHm indicates a HFS with m stages; $((RM^{(i)})_{i=1}^{m})$ represents that each stage consists of multiple unrelated machines; M_j represents machine eligibility constraint, S_{sd} stands for sequence-dependent setup times; $\sum T_j$ indicates the total tardiness and TST is total setup time objective.

2.2. Mathematical model

In this section, we provide a mix-integer linear programming model of the defined problem. Let $J = \{1, \ldots, n\}$ be the set of jobs, $I = \{1, \ldots, m\}$ be the set of stages, $M_i = \{1, \ldots, h_i\}$ be the set of machines at stage i, $E_{ij} \subseteq M_i$ be the eligible machine set of job j at stage i. Below are the other notations of the model:

Parameters:

w: weight of the total tardiness objective function

p_{ilj}: processing time of job j on machine l at stage i

S_{iljk}: sequence-dependent setup time on machine l at stage i after switching from job j to job k

d_j: due-date of job j

V: a very large number

Decision variables:

x_{iljk}: equal to 1 if job j precedes immediately job k on machine l at stage i, 0 otherwise.

Auxiliary variables:

T_j: tardiness of job j,

C_{ij}: completion time of job j at stage i.

The objective is to

$$\min w\sum_{j} T_j + (1-w)\sum_{i\in I}\sum_{l\in M_i}\sum_{j\in\{J,0\}}\sum_{k\in J, k\neq j} x_{iljk}S_{iljk} \tag{1}$$

s.t.

$$\sum_{j\in\{J,0\}, j\neq k}\sum_{l\in M_i} x_{iljk} = 1, \forall i \in I, k \in J \tag{2}$$

$$\sum_{j\in J, j\neq k} \sum_{l\in M_i} x_{ilkj} \leq 1, \forall i \in I, k \in J \tag{3}$$

$$\sum_{j\in J} x_{il0j} \leq 1, \forall i \in I, l \in M_i \tag{4}$$

$$\sum_{s\in\{J,0\}, s\neq j, s\neq k} x_{ilsj} \geq x_{iljk}, \forall i \in I, l \in M_i, j, k \in J, j \neq k \tag{5}$$

$$\sum_{l\in M_i} (x_{iljk} + x_{ilkj}) \leq 1, \forall i \in I, j \in J, k = j+1, \ldots, n \tag{6}$$

$$\sum_{j\in\{J,0\}, j\neq k} \sum_{l\in M_i \setminus E_{ik}} x_{iljk} = 0, \forall i \in I, k \in J \tag{7}$$

$$\sum_{j\in J, j\neq k} \sum_{l\in M_i \setminus E_{ik}} x_{ilkj} = 0, \forall i \in I, k \in J \tag{8}$$

$$C_{i0} = 0, \forall i \in I \tag{9}$$

$$C_{ik} + V(1 - x_{iljk}) \geq C_{i-1,k} + p_{ilk}, \forall k \in J, j \in \{J,0\}, j \neq k, i = 2, \ldots, m, l \in E_{ik} \tag{10}$$

$$C_{ik} + V(1 - x_{iljk}) \geq C_{ij} + S_{iljk} + p_{ilk}, \forall k \in J, j \in \{J,0\}, j \neq k, i \in I, l \in E_{ik} \tag{11}$$

$$T_j \geq C_{mj} - d_j, \forall j \in J \tag{12}$$

$$x_{iljk} \in \{0,1\}, \forall i \in I, l \in M_i, j \in \{J,0\}, k \in J, j \neq k \tag{13}$$

$$C_{ij} \geq 0, \forall i \in I, j \in \{J,0\} \tag{14}$$

$$T_j \geq 0, \forall j \in J \tag{15}$$

The objective is to minimize the weighted sum of the total tardiness and the total setup time. Constraints (2) ensure that each job is preceded exactly by one job on only one machine at every

stage. A job is scheduled as the first job on a machine if it is preceded by the dummy job 0. Constraints (3) ensure that each job is succeeded by at most one job on each machine at every stage. Constraints (4) impose that the dummy job can be succeeded by at most one job on each machine. Constraints (5) impose that if a job precedes any other job on a machine, it must has a predecessor on the same machine. This is a way to realize the assignment consistency. Constraints (6) prevent the cross-precedence. Constraints (7) and (8) impose the machine eligibility constraint. Constraints (9) set the completion time of the dummy job at each stage. Constraints (10) impose the job precedence constraint. A job cannot start to process at the current stage until the operation at the previous stage is finished. Constraints (11) impose the machine capacity constraint. A machine can process only one job at a time and therefore, a job can be started only after the completion of its predecessor and the machine setup. Constraints (12) calculate the job tardiness. Constraints (13) - (15) define the decision variables.

3. Literature review

The classical mean of solving the multi-objective optimization (MOO) problem is to optimize the scalarized objective function. By pre-multiplying the objectives with a user-defined weight vector $\underline{w} = [w_1, w_2, \ldots]$, the multi-objective problem is converted to a single objective problem, i.e., $\mathscr{P} : \min_{x \in \mathcal{X}} \underline{w} F(x)$. This approach is known as the *a priori* approach [12]. This approach has also been adopted in many multi-objective HFS scheduling problems [13, 14, 15, 16, 17, 18]. Jungwattanakit et al. [18] optimized a weighted sum of the makespan and the number of tardy jobs in a HFS with unrelated machines and SDST. The authors have performed a thorough study on the comparison of different methods including constructive heuristics and metaheuristics such as simulated annealing (SA), tabu search (TS) and genetic algorithms (GA). One interesting insight is that the user preference has impact on the ranking of the algorithms under comparison. For *a priori* approach, the main difficulty resides in the correct specification of the weight vector due to the vagueness nature of human decision-making, especially when the user is not provided any information on the possible trading-off behavior of different objective functions. Another disadvantage is that this approach cannot be used to find Pareto-optimal solutions which lie on the non-convex portion of the Pareto-optimal front [12].

Another stream of methods, known as the *a posteriori* approach, try to find a set of non-dominated solutions or Pareto-optimal set for the posterior decision-making procedure. For this purpose, various methods were proposed, including the classical weighted-sum approach, ϵ-constraint method [19] and multi-objective evolutionary algorithms. EAs are stochastic search methods that simulate the process of evolution, incorporating ideas such as reproduction, mutation and the Darwinian principle of "survival of the fittest". VEGA is the first practical EA for MOO problem. Schaffer [20] extended the simple GA for MOO by modifying the selection phase. The idea is to select individuals according to each objective function in turn. However, the main problem of VEGA is its bias toward some Pareto-optimal solutions, making it difficult to find a complete Pareto front. To overcome this weakness, Goldberg [21] suggested the use of a niching technique in the selection procedure. Later, Fonseca and Fleming [22], Horn et al. [23] and Srinivas and

Deb [24] implemented that suggestion and succeeded by applying the resulting algorithms, i.e., MOGA, NPGA and NSGA, to some problems. Zizler and Thiele [25] compared four different EAs with a 0/1 knapsack problem and showed that NSGA provided the best performance. Then, they proposed the SPEA, which uses an external elite archive and applies a fitness assignment method based on the so-call individual strength. The authors showed that SPEA outperformed the other four algorithms. The idea of using elite archive is also found in PAES (Knowles and Corne [26]), but instead of population-based search, the author suggested a simple (1+1) evolution strategy. Deb et al. [27] summarized the main criticisms received by NSGA and proposed the NSGA-II. The three main improvements are: a faster nondominated sorting approach, the use of elitism, and a parameter-free crowding-comparison approach. Compared to PAES and SPEA, NSGA-II obtains better spread of solutions in all the test problems, and converges closer to the true Pareto set in most of the problems. Zitzler and Thiele [28] improved their SPEA algorithm to obained SPEA2. By comparison, the authors showed that SPEA2 and NSGA-II have similar performance in most problems but in higher dimensional objective spaces, SPEA2 has advantages. We refer to [12] for more information on multi-objective EAs. Among these algorithms, NSGA-II is considered as one of the most efficient method for Pareto-optimal set approximation.

EA-based methods are adopted in many multi-objective HFS problems. In Ebrahimi [29], the NSGA-II has been adopted to optimize the makespan and the total weighted tardiness of a HFS with SDST. Results showed the superiority of NSGA-II comparing to a multi-objective genetic algorithm and a multi-phase genetic algorithm. Dugarding et al. [3] addressed the scheduling problem in a HFS with reentrant flow. Based on the NSGA-II framework, the authors proposed a new algorithm called L-NSGA which replaces the Pareto dominance with the Lorenz dominance relationship aiming at improving the search intensification. The proposed method has been shown better than the NSGA-II and the SPEA2. Abyaneh et al. [30] considered the minimization of the makespan and the total tardiness in a HFS with finite buffer capacity and SDST. They have adapted the NSGA-II and a sub-population genetic algorithm for the problems and reported that the solution quality can be improved by integrating a local search procedure. The hybridization of multi-objective EA with local search procedure was also adopted in Behnamian et al. [31], Asefi et al. [32] and Zandieh et al. [33]. Li et al. [34] developed a hybrid algorithm which employs two exploitation phases and two exploration phases for a balance in global and local search abilities, and have applied it to solve the HFS scheduling problem with setup energy consumption. Recently, Zeng et al. [8] investigated the scheduling problem in a real-world paper mill manufacturing environment to optimize the makespan, electricity consumption and material wastage. The authors hybridized the NSGA-II with a tabu search as well as a job merging strategy which reduces the production switches. The hybrid algorithm outperforms the NSGA-II by a large amount. The job merging strategy contributes much more than the embedded tabu search. This is an example of using problem structure knowledge to promote the optimization performance.

Besides EAs, there are researches extending other metaheuristics, such as ant colony optimization, tabu search and iterated local search, to MOO and have obtained state-of-art results on some multi-objective HFS scheduling problems. Luo et al. [7] proposed a multi-objective ant colony optimization algorithm for optimizing the makespan and the electric power consumption. The

proposed method has been shown to outperform the NSGA-II and SPEA2 in terms of solution quality even it is slower. Wang and Liu [4] considered the scheduling problem in a two-stage HFS with preventive maintenance. A multi-objective tabu search which employs several parallel searching paths has been proposed and is shown superior than the NSGA-II. In a recent research, Schulz et al. [35] developed a multiphase iterated local search algorithm to approximate a three-dimensional Pareto front regarding three objectives: makespan, total energy costs and peak load.

However, the scheduling problem is not just about the searching scheme. A metaheuristic-based scheduling algorithm usually employs an optimization model for searching the best solution in the design space and a simulation model for fitness evaluation. In the optimization model, a schedule is encoded as a string of decision variables. Such string is decoded into a complete schedule in the simulation model for performance evaluation. Because the solution space of HFS scheduling problems is large, in most of the researches, not all the decision variables are encoded but just a subset of them. So, it raises the problem of how much detailed the encoding scheme should be. Apparently, the smaller the encoded space, the easier for the optimizer to find the best therein but the risk of losing the global optimal is greater. Fernandez-Viagas et al. [36] investigated the potential optimum reachable by using different encoding schemes and concluded that a job permutation representation, while maintaining the search efficiency, can lead to high quality solutions. Such scheme is also adopted in the majority of HFS researches. On the other hand, the decoding algorithms, or saying, the methods constructing a complete schedule from a job permutation, also play an important role in the solution quality. For job permutation representation, common decoding algorithms are the list scheduling (LS) [16] and the permutation scheduling (PS). In Ruiz and Maroto [37], the typical PS was adapted for HFS with SDST by modifying the machine selection rule from first-available-machine to earliest-finish-machine. Li et al. [34] developed three different decoding algorithms based on the LS framework. These use distinct machine selection rules taking the machine setup time and setup energy into consideration. Yu et al. [38] proposed a dynamic scheduling (DS) for optimizing the total tardiness objective and reported its advantage over LS and PS on the testing problems. Yet, they considered the machine setup time is sequence independent and can be included in the job processing time, which is not practical in many scenes.

To the best of our knowledge, the problem defined in this paper has not been tackled in the literature. Also, no ad-hoc decoding algorithm has been proposed considering simultaneously the total tardiness and total setup time. In this research, instead of developing a complicated hybridization of search mechanisms, we focus on the development of efficient decoding algorithms and the systematic use of them to solve the defined problem.

4. Proposed algorithm

4.1. Decoding algorithms

In this research, we use the permutation-based encoding scheme which is widely applied in the literature. A solution is represented by a job permutation $\pi = \{1, 2, \ldots, n\}$.

Table 1: The original and the proposed DS versions

Decoding algorithm	Machine selection indicator $\phi(t)$	Comments
DS	BTPT + PT + MTTI	Aim at balancing the machine workload
DS2	BTPT+PT+MTTI+MDST	Major workload + few setup information
DS3	PT + MDST	A balance between machine workload and setup information
DS4	MTST	Only total setup time
DS5	MDST	Only difference on the setup time

For decoding algorithm, we develop four variants of the dynamic scheduling (DS) for our problem. DS is basically a simulation procedure with embedded machine selection rule and job sequencing rule. The jobs are assigned to the machine selected according to the least workload rule, and then they are sequenced in the machine buffer according to their priorities indicated by the encoded job permutation. Whenever a machine becomes idle, it takes the first job waiting in its upstream buffer and starts the process; whenever a job is released by a machine, it is assigned to the next stage. This simulation continues when all jobs are finished. We refer the readers to Algorithm 1 in Yu et al. [38] for more technical details of DS. More specifically, under the framework of DS, we modify the machine selection rule in order to allow the SDST information being taken into account. Details of the modifications are as follows.

In DS, the machine selection is based on an indicator. When assigning a job j^* to one of the eligible machines at simulation time t, the machine k^* with the lowest indicator $\phi(t)$ is selected. This indicator $\phi(t)$ is calculated as $\phi(t) = \sum_{\omega \in \Omega} \omega(t)$, where $\omega(t)$ is called machine selection metric, and Ω is the set of metrics to be considered for machine selection. In the original DS, the machine selection considers only metrics related to machine workload. However, when SDST exist, the machine with the lowest workload may require a long setup time, leading to inefficiency. For this reason, we develop four DS versions denoted by DS2, DS3, DS4 and DS5, which use different combinations of SDST-related metrics. The $\phi(t)$ of each version is reported in Table 1, where the candidate metrics composing Ω are: Processing time (PT), Buffer Total Processing Time (BTPT), Machine Time to Idle (MTTI), Machine Total Setup Times (MTST) and Machine Differential Setup Times (MDST).

Figure 1 shows an example of calculating those machine selection metrics when assigning the job j^* to a machine. Here, colorful circles stand for jobs, gray rectangles represent setup operations between jobs, the numbers represent the time length. PT is the processing time of job j^*, i.e., 4; BTPT is the sum of processing time of jobs already in the buffer at this moment, i.e., 8+ 6+5 = 19; MTTI is the remaining processing time of the current job in the machine, i.e., 1. MTST is the total setup time after assigning job j^* to the machine and sorting the jobs according to their encoded priorities, i.e., 3+1+3+1 = 8; MDST is the increment of setup time after the assignment, i.e., (3+1+3+1) - (3+2+1) = 2. The mathematical formulations for these metrics are provided in Appendix A.

Apparently, different machine selection rules result in different dynamics in the simulation and may favor the two objective functions, i.e., total tardiness and total setup time, to different extents.

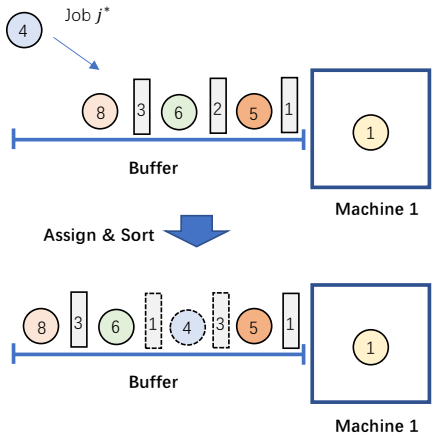


Figure 1: An example for machine selection metrics calculation

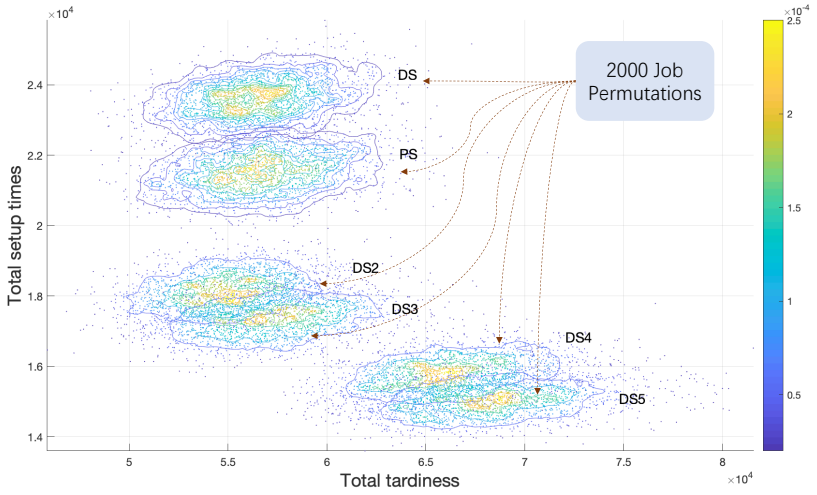


Figure 2: Solution distribution of different decoding algorithms

To show this, we have performed a preliminary experiment. For a scheduling problem with 50 jobs and 10 stages, two thousand randomly generated job permutations are decoded by different versions of DS, as well as by the modified PS [37]. The performance of each job permutation is plotted in the objective space in Figure 2. In the figure, the Voronoi cells are used to calculate data density and the contours of densisty are provided. It shows that different decoding algorithms map the same set of job permutations to different regions. By considering SDST information, the new DS versions perform better than the original DS in terms of total setup time. From DS2 to DS5, the tendency of favoring the setup time objective is increasing. Similar results are observed in other experiments, where we increase the number of random generated job permutation to 20000 and summarize the results from 90 different instances. Details are provided in Appendix B.

This phenomenon, however, raises questions for the multi-objective scheduling. For *a priori* approach, the best decoding algorithm no doubt depends on the user preference. How to select the most proper decoding algorithm given a preference vector? For *a posteriori* approach, using only one decoding algorithm is insufficient for generating a representative non-dominated front. How to embed several decoding algorithms together in EAs? To answer these questions, we present a multi-decoding framework in the next section.

4.2. The proposed multi-decoding framework

The decoding algorithm is a heuristic for schedule construction. In the combinatorial optimization literature, using several heuristics simultaneously on a problem is not new. This branch of research is known as hyper-heuristic. More specifically, a hyper-heuristic is an automated methodology for selecting heuristics to solve hard computational search problems [39]. To do so, a hyper-heuristic uses some feedback from the search process and learns the adaptability or utility of the low-level heuristics on the problem. Meta-heuristics have been adopted as hyper-heuristics in many researches. These include evolutionary algorithms [3, 40, 41], tabu search [42, 43] and single-point local search [44].

In this section, we propose a simple framework for using a set of candidate decoding algorithms for scheduling based on EAs. Under this framework, EAs work as high-level hyper-heuristics to search the joint space of low-level heuristic (decoding algorithm) and job sequence. The basic idea is to treat the decoding algorithm type as one of the decision variables. This is made by adding a digit in the typical job-permutation-based chromosome to represent the decoding algorithm used to decode the job sequence. In this way, the fitness of the chromosome depends not only on the quality of the job sequence but also on the performance of the decoding algorithm. The set of chromosomes using the same decoding algorithm can be considered as a *tribe*. In the initial population, several tribes exist. During the evolution, the tribe better fitting the environment will be more likely to survive. In this way, the proper decoding algorithm is chosen by the evolution process itself.

4.2.1. Hyperchromosome

Denote the job sequence as π and the decoding algorithm as $\mathscr{D} : \pi \to X$, where X are the decision variables for constructing a complete schedule. In our framework, an individual is encoded as a tuple $chrom = (\mathscr{D}, \pi)$. To distinguish it from the convention, *chrom* is named hyperchromosome. The decoding procedure of *chrom* is performed by $X = decode(chrom) = \mathscr{D}(\pi)$.

4.2.2. Crossover and mutation

In EAs, crossover and mutation are the two main genetic operators for reproduction. In our framework, individuals from different tribes are allowed to mate and produce offspring. To this end, the hypercrossover operator $\otimes_H$ for two hyperchromosomes $chrom_1 = (\mathscr{D}_1, \pi_1)$ and $chrom_2 = (\mathscr{D}_2, \pi_2)$ is defined as below:

$$\otimes_H(chrom_1, chrom_2) = (\mathscr{D}_1, \otimes(\pi_1, \pi_2)) \cup (\mathscr{D}_2, \otimes(\pi_1, \pi_2)) \tag{16}$$

where $\otimes$ stands for the common crossover operator applicable to two job permutations, such as one-point crossover, partial mapped crossover and order-based crossover [45]. $\otimes(\pi_1, \pi_2)$ represents the child given by π_1 and π_2 through common crossover. In case π_1, π_2 generate multiple children $c_1, c_2, \ldots, c_r$, $(\mathscr{D}, \otimes(\pi_1, \pi_2))$ is the union set of $(\mathscr{D}, c_1), (\mathscr{D}, c_2), \ldots, (\mathscr{D}, c_r)$.

Remark 1. *The hypercrossover operator is designed considering the principle of equality. It generates equal number of children for each of the parents' tribe. This can be treated as a sampling*

scheme that generates a full-factorial design with two factors: decoding algorithm $= \{\mathscr{D}_1, \mathscr{D}_2\}$ *and job permutation* $= \{c_1, c_2, \ldots, c_r\}$.

Here we give an example of the hypercrossover operator. Let the parents be $chrom_1 = (PS, \pi_1)$ and $chrom_2 = (DS, \pi_2)$, where $\pi_1 = \{5, 2, 3, 4, 1\}$ and $\pi_2 = \{4, 2, 1, 5, 3\}$. Let $\otimes$ be the order-based crossover [45]. With a binary vector of, say, $\{1, 1, 0, 0, 1\}$, $\otimes(\pi_1, \pi_2)$ generates two offsprings, $c_1 = \{5, 2, 4, 3, 1\}$ and $c_2 = \{4, 2, 5, 1, 3\}$. Then, according to equation 16, $\otimes_H(chrom_1, chrom_2)$ generates four offsprings, (PS, c_1), (DS, c_1), (PS, c_2) and (DS, c_2).

The hypermutation operator $\odot_H$ is defined as below. Let $\mathbb{D}$ be the set of candidate decoding algorithms, we have:

$$\odot_H((\mathscr{D}, \pi)) = \{(\mathscr{D}, \odot(\pi)), \forall \mathscr{D} \in \mathbb{D}\} \tag{17}$$

where $\odot$ stands for the common mutation operator applicable to a job permutation π, such as insert, swap and reverse.

Remark 2. *The hypermutation operator aims at introducing diversity into the population in terms of both job sequence and decoding algorithm. It aims at preventing the premature convergence of the population due to the advantage of certain decoding algorithm over the others in early search phase.*

4.2.3. Generational scheme

In EAs, the generational scheme defines how the new generation of the population is created. In our framework, a preservation strategy is used. Indeed, the quality of certain tribe may be poor in the initial search phase but improves only after several generations. For this reason, we preserve for each tribe some promising individuals for the next generation. Actually, specie preservation is an important strategy for the multi-modal optimization [46]. The generational scheme is implemented as Algorithm 1.

Algorithm 1: Generational scheme

Input: Current population $\mathbb{P}$, offsprings generated by crossover and mutation $\mathbb{C}$, perservation percentage δ

Output: New population $\mathbb{P}_{new}$

1 Initialization: $\mathbb{P}_{new} = \emptyset$, $\mathbb{P}_{all} = \mathbb{P} \cup \mathbb{C}$;
2 Sort $\mathbb{P}_{all}$ according to the non-decreasing order of the fitness;
3 **for** $\mathscr{D} \in \mathbb{D}$ **do**
4 abstract from $\mathbb{P}_{all}$ the best $\delta * |\mathbb{P}|$ individuals of that tribe and put them into $\mathbb{P}_{new}$;
  %Parameter δ should satisfy $\delta * |\mathbb{D}| < 1$, i.e., the total preservation number should be less than the population size.
5 **end**
6 Complete $\mathbb{P}_{new}$ by filling the best $|\mathbb{P}| - |\mathbb{P}_{new}|$ chromosomes abstracted from $\mathbb{P}_{all}$;

It should be noted that this generational scheme can not only be applied to canonical single objective EAs, but also to multi-objective EAs, such as NSGA-II [27] by sorting in step 2 using the non-dominated soring technique proposed therein.

4.3. The complete procedure

Basically, the MDF is implemented by replacing several key operators existed in the EA framework. For this reason, it can be easily applied to many EAs such as the canonical genetic algorithm, multi-objective genetic algorithms NSGA-II [27] and many others. Figure 3 shows a general integration of the MDF with EAs. The procedure starts with the creation of an initial population of hyperchromosomes. Then, a selection procedure, e.g., tournament selection or roulette wheel selection, is used to choose the individuals to form a mating pool. The parents in the mating pool generate offsprings via hyper genetic operators. Finally, a new population is obtained by the generational scheme. These steps repeat until a termination condition is met.

In this research, the MDF is incorporated to a simplified version of the GA presented in [38], as well as to the NSGA-II. Both are implemented with the common framework described in Algorithm 2.

Algorithm 2: Evolutionary algorithm with multi-decoding framework

Input: population size P_{size}, crossover probability p_c, mutation probability p_m, candidate decoding set $\mathbb{D}$, tribe preservation percentage δ

Output: Non-dominated solutions

Initialization: Create an initial population $\mathbb{P}$ with P_{size} hyperchromosomes;
while *Not terminate* **do**
  Selection: Create a mating pool $\mathbb{M}$ with P_{size} parents using the *selection* procedure;
  Reproduction: set the offspring set $\mathbb{C} \leftarrow \emptyset$, counter $i \leftarrow 1$;
  while $i < P_{size}$ **do**
    Pick two parents $\mathbb{M}(i)$, $\mathbb{M}(i+1)$ from the mating pool;
    if $rand < p_c$ **then**
      Perform hypercrossover $\mathscr{C} \leftarrow \otimes_H(\mathbb{M}(i), \mathbb{M}(i+1))$;
    else
      $\mathscr{C} \leftarrow \{\mathbb{M}(i), \mathbb{M}(i+1)\}$;
    end
    for $child \in \mathscr{C}$ **do**
      if $rand < p_m$ **then**
        Perform hypermutation $child \leftarrow \odot_H(child)$
      end
      Update $\mathbb{C} \leftarrow \mathbb{C} \cup child$;
    end
    Increment $i \leftarrow i + 2$;
  end
  Decode and evaluate the solutions in $\mathbb{C}$;
  Perform the generational scheme $\mathbb{P} \leftarrow$ Generational_scheme$(\mathbb{P}, \mathbb{C}, \delta)$;
end

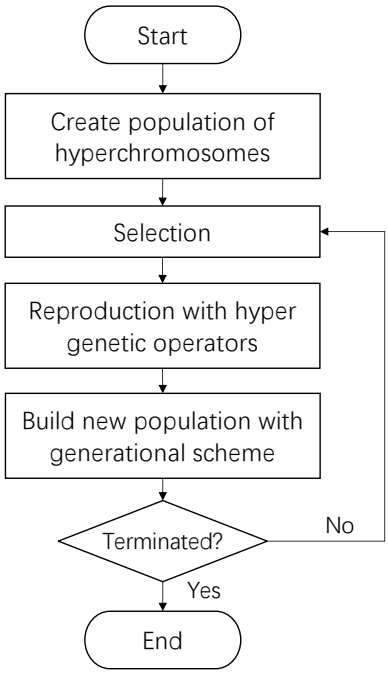


Figure 3: EA with the proposed multi-decoding framework

5. Numerical results

To validate the usefulness of the proposed decoding algorithms and the MDF, we apply them with EAs for solving the HFS scheduling problem defined in section 2 using *a priori* and *a posteriori* approach, respectively. In the first case, the user preference is given as a weight vector $[w, 1-w]^T$, and a single-objective GA is implemented to optimize a weighted-sum of the total tardiness and total setup time. In the second case, the problem is solved by approximating the Pareto optimal set with the NSGA-II [27]. In the first case, we compare the performance of the GA with and without using the proposed MDF (Experiment A); in the second case, besides the comparison between different versions of EAs (Experiment B), the proposed NSGA-II is compared with two state-of-art multi-objective algorithms of different search mechanisms (Experiment C).

5.1. Test Instances

The test instances are generated as follows. An instance is featured by three factors, number of jobs n, number of stages m and maximum machine setup time S_{max}. The number of parallel machines in each stage is randomly sampled from [2,3,4]. For a given job, not all of these parallel machines are eligible. The probability of eligibility is set as 80%, under the condition that at least one machine is eligible for a job. The job processing times are random integers sampled from [1, 100]. The sequence-dependent setup times are integers sampled from $[1, S_{max}]$. The setup time for the first job on a machine is set as zero. The job due date is generated using the same method and parameters given in [38]. We have created 3 instance groups with $S_{max} = 25, 100$ and 200, respectively, to simulate short, medium and long setup scenarios. Each group contains 9 instance sets with different combinations of number of jobs $n = \{20, 50, 100\}$ and number of stages $m = \{5, 10, 20\}$. Each set has 10 randomly generated instances. In total 3*3*3*10 = 270 instances

Table 2: Algorithm parameters

Algorithm	Parameter	Value
GA	Population size P_{size}	150
	Crossover rate p_c	1
	Mutation rate p_m	0.01
NSGA2	Population size P_{size}	150
	Crossover rate p_c	1
	Mutation rate p_m	0.01
MOSA	Epoch duration	100
	Number of temperatures N	150
	Initial temperature T_0	1
	Final temperature T_f	0
	Cooling function	Hyperbolic cooling rate [47]
	Perturb operator	Swap
RIPG	Destruction size k	5
	Local search radius n_neigh	5
	Unchanged iterations to restart	50

are created. An instance is denoted with its parameters as below, e.g., SSD25_N20M5_P1 indicates the first instance with $S_{max} = 25$, $n = 20$ and $m = 5$.

5.2. Performance comparison: a priori approach

The settings of Experiment A are as follows. The GA we implemented is a version simplified from [38] with the following building blocks: Roulette wheel selection, order-based crossover (OBX) [45], and insert mutation. Initial population is created as following: we preserve equal number of chromosomes for each tribe; the job permutations for each tribe are randomly created except two of them are given by the earliest-due-date rule and the minimal slackness rule as in [38]. Based on some preliminary tests, the parameters of the GA are set, as shown in Table 2.

By coupling with different decoding algorithms, we have obtained several GA versions: GA_PS, GA_DS2, GA_DS3, GA_DS4, GA_DS5. The GA coupled with the MDF is denoted as GA_mix. The candidate decoding algorithm set $\mathbb{D} = \{PS, DS2, DS3, \ldots, DS5\}$. The preservation percentage δ is set as 5%. The objective is to minimize $f = w\sum T_j + (1-w)TST$. These GAs are compared on the instances of group SSD100. The CPU time allocated for these algorithms are identical and increase with problem size, as shown in Table 4. They are calculated following the formula $10000 + 0.5\tau mn^2$[ms], where τ is set as 5 in this case. The comparison is implemented as a full factorial design with factors and levels indicated in Table 3. Each test runs for 5 replications. These result in a total $6 \times 6 \times 3 \times 3 \times 10 \times 5 = 16200$ runs. To easily compare the performance of algorithms across different instances, two indexes are often used. The relative deviation index (RDI) [38] is calculated as

$$RDI = \frac{Alg_{sol} - Min_{sol}}{Max_{sol} - Min_{sol}} * 100, \tag{18}$$

and the Relative percentage increase (RPI) index [37] is given by

$$RPI = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} * 100, \tag{19}$$

Table 3: Design of experiments

Experiment	Factor	Levels
A	Algorithm	GA_PS, GA_DS2, GA_DS3, GA_DS4, GA_DS5, GA_mix
	Weight value w	0, 0.2, 0.4, 0.6, 0.8, 1
	Number of stages m	5, 10, 20
	Number of jobs n	20, 50, 100
	Instance	1,2,. . . ,10
B	Algorithm	NSGA2_PS, NSGA2_DS2, NSGA2_DS3, NSGA2_DS4, NSGA2_DS5, NSGA2_mix
	Number of stages m	5, 10, 20
	Number of jobs n	20, 50, 100
	Instance	1,2,. . . ,10
C	Algorithm	MOSA, RIPG, NSGA2_mix
	Number of stages m	5, 10, 20
	Number of jobs n	20, 50, 100
	Max setup time S_max	25, 100, 200
	Instance	1,2,. . . ,10

Table 4: CPU time allocated for algorithms on different test instances [Seconds]

	N20M5	N20M10	N20M20	N50M5	N50M10	N50M20	N100M5	N100M10	N100M20
Experiment A	15	20	30	41.25	72.5	135	135	260	510
Experiment B	60	70	90	112.5	175	300	300	550	1050

where Alg_{sol} is the objective value of the current algorithm on the given instance, Max_{sol} and Min_{sol} are the worst and the best objective value obtained by any of the algorithms in the comparison, respectively. RDI ranges in [0,100] and is suitable for measuring the performance ranking among the competitors, whilst RPI is good for revealing the difference in objective values. We would choose to use the proper index according to the context in the later analysis. All optimization algorithms are coded in Matlab 2016a on a PC with Intel XEON E5-2699 v4 CPU (22 cores, 2.2 GHZ) and 256 GB of RAM. Decoding algorithms are implemented in C++ and called in the Matlab environment.

Table 5: Tukey test result of Experiment A

Group	Algorithm ranking and RDI values					
$w = 0$	GA_DS5	GA_mix	GA_DS4	GA_DS3	GA_DS2	GA_PS
	10.41 (A)	11.44 (A)	22.47 (B)	44.91 (C)	57.49(D)	85.74 (E)
$w = 0.2$	GA_mix	GA_DS2	GA_DS3	GA_DS5	GA_DS4	GA_PS
	33.49 (A)	35.10 (A)	35.95 (A)	44.76 (B)	45.38 (B)	69.85 (C)
$w = 0.4$	GA_DS2	GA_mix	GA_DS3	GA_PS	GA_DS4	GA_DS5
	23.20 (A)	24.52 (A)	35.60 (B)	36.46 (B)	73.46 (C)	80.33 (D)
$w = 0.6$	GA_PS	GA_DS2	GA_mix	GA_DS3	GA_DS4	GA_DS5
	18.20 (A)	18.77 (A)	19.74 (A)	32.29 (B)	76.26 (C)	85.04 (D)
$w = 0.8$	GA_PS	GA_mix	GA_DS2	GA_DS3	GA_DS4	GA_DS5
	13.25 (A)	19.17 (B)	20.18 (B)	33.028 (C)	77.96 (D)	86.86 (E)
$w = 1$	GA_PS	GA_mix	GA_DS2	GA_DS3	GA_DS4	GA_DS5
	12.08 (A)	16.84 (B)	21.61 (C)	35.46 (D)	77.81 (E)	88.82 (F)

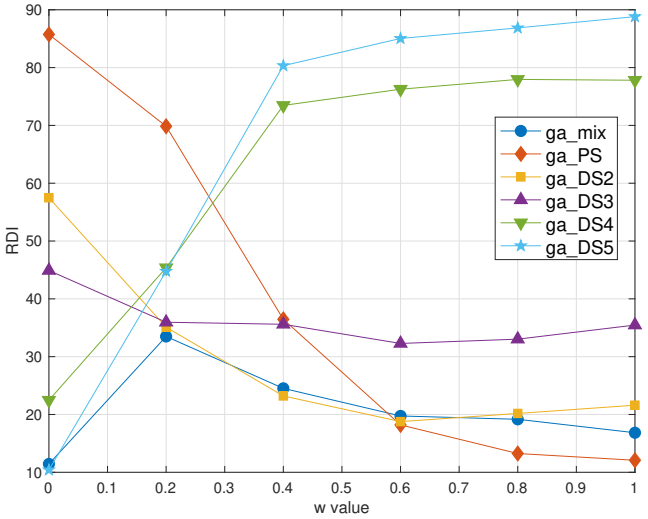
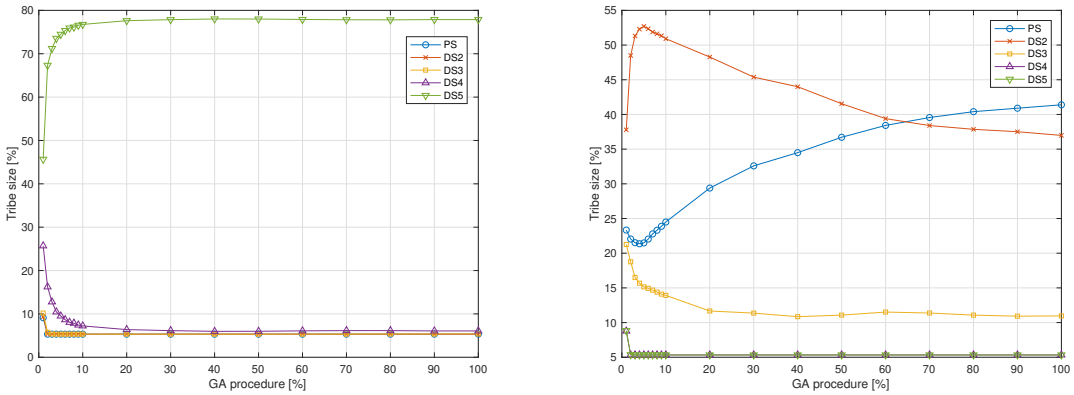


Figure 4: Comparison of GAs under various scenarios of user preference

The comparison results are summarized in Figure 4. As seen, the ranking of algorithms varies with the w value. Indeed, GAs integrated with different decoding algorithms favor different user preferences. When w is large, which means the total tardiness is more important for the user, the conventional GA_PS is the best. Yet, when w is smaller, the advantage of GA_PS vanishes, and the GAs using decoding algorithms favoring setup objective (like DS4 and DS5) outperform the others. In summary, when using single decoding algorithm, the GA cannot cover all situations in terms of user preference.

On the other hand, the performance of GA coupled with the MDF does not vary too much with the user preference. Actually, its performance approaches to the best GA version in that specific w level. To provide statistical evidence, an ANOVA analysis is performed using factors of n, m, w, *algorithm*, as well as their second-order interactions. According to the ANOVA table (Figure C.13), *algorithm* and *algorithm**w are the most influential factors, both with p-value less than 0.001. This consolidates the finding in [18] that the algorithm ranking is affected by user preference. Then, the Tukey test, which is a multiple comparison method, is adopted to compare the levels of the second-order interaction *algorithm**w. In Table 5 is reported the mean RDI value for each algorithm and the results of the Tukey test. Algorithms with different letters in the parenthesis are statistically different in terms of the performance (p-value $< 5\%$). Details of the ANOVA and Tukey test can be found in Appendix C. As shown, the performance of GA_mix is not statistically worse than the corresponding best GA version except when $w = 0.8$ and 1. This shows that using the MDF one can generate a solution almost as good as the best single decoding GA version aligning to the user preference.

Figure 5 shows the varying tribe size when GA_mix is running. Each point represents the mean value from 90 (instances)*5(replications) = 450 runs. The tribe size is represented by the percentage of chromosomes in the population. It shows how a tribe adapts to the environment. As seen in (a), when $w = 0$, the "fittest" tribe is DS5, its size grows quickly whereas the other tribes keep shrinking as the GA proceeds. When $w = 1$, the fittest tribe is PS, yet its size decreases in



(a) $w = 0$ (b) $w = 1$

Figure 5: Tribe size variation during the running of GA_mix

the initial phase then returns to an increasing trend. This is the case where the most promising tribe requires certain "warm up" period. This shows the necessity of the preservation strategy. However, due to such "warm up", the PS tribe is not able to dominate the population. Thus, a considerable amount of the search budget is actually shared by the second most promising tribe DS2 and others, leading to a result a bit worse than using only PS in the GA. This explains the performance gap between GA_mix and GA_PS when $w = 1$ (Figure 4). As a conclusion, the MDF shows the ability to select the suitable decoding algorithm according to user preference during the GA procedure.

5.3. Performance comparison: a posteriori approach

5.3.1. Experiment setup

We conduct two comparison experiments, B and C. In Experiment B, the performance of the MDF is shown by comparing different NSGA-II versions; in Experiment C, the NSGA-II coupled with the MDF is compared to two benchmark algorithms on different groups of instances.

The settings of Experiment B are as follows. We implement the NSGA-II as in Deb et al. [27]. To adopt it for the HFS problem, tournament selection, order-based crossover and insert mutation are used. The algorithm parameters are given in Table 2. Six different versions of NSGA-II are created using different decoding algorithms: NSGA2_PS, NSGA2_DS2, NSGA2_DS3, NSGA2_DS4, NSGA2_DS5 and NSGA2_mix. For NSGA2_mix, the candidate decoding algorithm set $\mathbb{D} = \{PS, DS2, DS3, \ldots, DS5\}$, and the preservation percentage δ is set as 5%. Initial population is built as described in Experiment A. Given that NSGA-II is more complex than the canonical GA, and searching for a Pareto front is more difficult than a single optimal solution, the allocated CPU time is extended to $50000 + 0.5\tau mn^2$[ms], with $\tau = 10$, whose values are reported in Table 4. The comparison is made on the instance group SSD100, and is implemented as a full factorial design with factors and levels given in Table 3. Each test runs for 5 replications.

In Experiment C, besides NSGA2_mix, we implement the Multi-objective Simulated Annealing (MOSA) proposed in Smith et al. [48], and the Restarted Iterated Pareto Greedy (RIPG) given in

Ciavotta et al. [49] as two benchmarks. Simulated annealing is a single-point local search method, it once shattered the world of combinatorial optimization by its convergence property to optimality in single objective optimization. Smith et al. [48] extended the method to multi-objective problems by introducing the dominance-based energy function to guide the move acceptance, and a Pareto archive to store efficient solutions. The authors showed the MOSA "consistently generates archives closer to the true front than NSGA-II." The RIPG in [49] is basically an iterated local search which explores different regions of the current non-dominated front using a destruction-construction strategy. In the destruction phase, a block of jobs are removed from the string, then they are re-inserted into the string in the construction phase using the well-known NEH approach. The RIPG is shown quite efficient for searching permutation solution space, and outperforms several multi-objective algorithms on the flowshop scheduling problem. To apply the MOSA and RIPG on our HFS scheduling problem, the solution is encoded as a job permutation, and Ruiz's modified PS [37] is adopted as the decoding method for solution evaluation. The algorithm parameters are given in Table 2, which are set based on the proposals of the original papers and some preliminary tests. The algorithms are compared on the instance groups of SSD25, SSD100 and SSD200. The experiment is implemented as a full factorial design with factors and levels indicated in Table 3. Each algorithm runs for 5 replications on the same instance. To mitigate the influence of different algorithm implementations/mechanisms, all algorithms in Experiment C are budgeted with the same number of objective function evaluations (decodings). The budget increases with the problem size. More specifically, the budget is 30000, 40000 and 50000 when number of jobs equals 20, 50 and 100, respectively.

5.3.2. Performance indicators

The NSGA-II, as well as other multi-objective algorithms, outputs a set of non-dominated solutions. When evaluating the quality of a non-dominated set, different aspects should be considered: convergence, spread and distribution. Various performance metrics have been proposed to this aim [50]. To evaluate the quality of a non-dominated set, we use the hyper volume (HV) [25] and the Modified Inverted generational distance (IGD^+) [51]. The HV measures the volume of the area dominated by the front to a given nadir point. The larger the HV, the better the front. The IGD^+ measures the distance of the front to the true Pareto front. Front with smaller IGD^+ is considered better. In our case, since the true Pareto front for an instance is unknown, we gather the non-dominated solutions obtained by all competing algorithms in all replications and use them as the reference front.

An important property for the quality metric is Pareto-compliant. A quality metric is Pareto-compliant if and only if the ranking it establishes over approximated fronts does not contradict Pareto optimality. In the literature, HV is the only metric known as Pareto-compliant. IGD^+ is weakly Pareto-compliant, it is a modification of the most applied Inverted generational distance (IGD) which is, however, not Pareto-compliant. Both HV and IGD^+ consider the three criteria (convergence, spread and distribution) simultaneously.

5.3.3. Comparison results

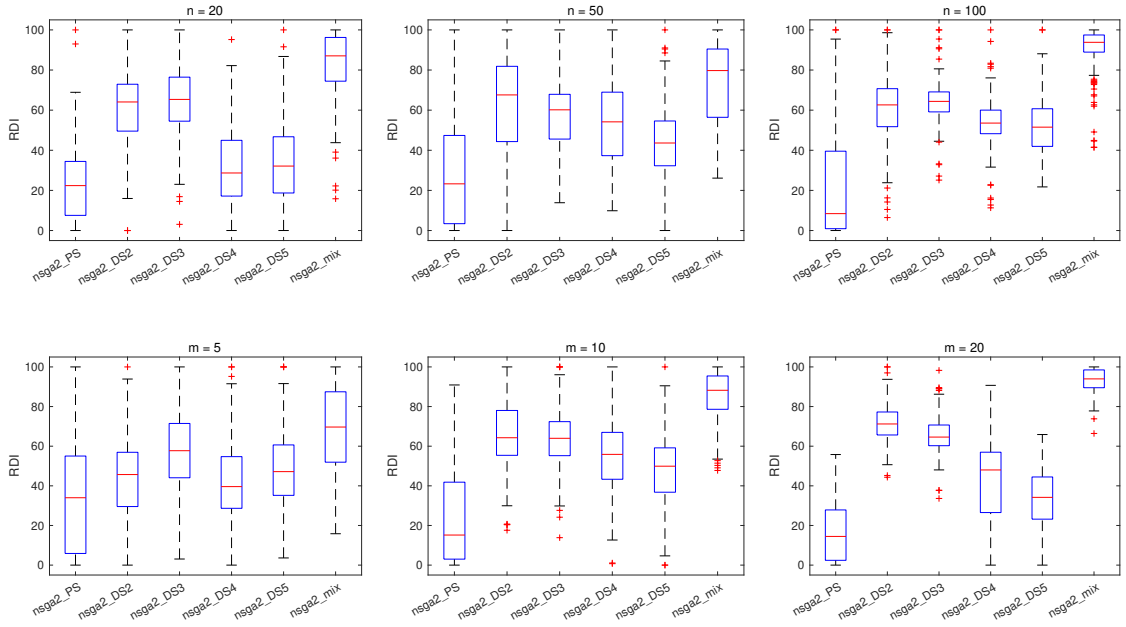


Figure 6: NSGA-II performance comparison with HV

The results of Experiment B are as follows. In Figure 6 are reported the boxplot of RDI values calculated using HV. The comparison is made on instance groups with different number of jobs $n = \{20, 50, 100\}$ and number of stages $m = \{5, 10, 20\}$. Each box contains 3 (stages/jobs)*10 (instances)*5 (replications) = 150 data. As shown, NSGA-II coupled with the proposed DS versions tends to result in higher median of HV than that with the PS. When using the MDF, the front output by the NSGA-II covers more area than those using only one decoding algorithm and obtains higher HV. Such advantage is more obvious when the instance is large ($n = 100$ and $m = 20$). The comparison based on IGD$^+$ is reported in Figure 7. The advantage of NSGA2_mix is clearly observed in the groups of $n = 100$ and $m = 20$. When $n = 50$ and $m = 5$, the performance of NSGA2_mix is still considered competitive. To provide statistical evidence, we perform the ANOVA analysis on both responses of RDI_HV and RDI_IGD$^+$. For both responses, *algorithm* is the most influential factor, and the interaction terms *algorithm***n* and *algorithm***m* are also significant. This means that the performance of different algorithms are not statistically equal, and, their rankings depend on the instance features of number of jobs and stages. We perform Tukey test to obtain the ranking of algorithms under different n and m, for both RDI_HV and RDI_IGD$^+$. The results of Tukey test on RDI_HV and RDI_IGD$^+$ are reported in Table 6 and Table 7, respectively. Algorithms with different letters in the parenthesis are statistically different (p-value $< 5\%$). Details of the statistical analysis are given in Appendix C. Results show that the NSGA2_mix statistically outperforms the other algorithms in most of the cases.

In Figure 8 are depicted the approximated Pareto fronts output by different NSGA-II versions for both a small and a large instances. We have some observations:

1. With the MDF, the NSGA-II outputs a front spreading in a larger range of the objective space. We found that such front is actually composed by individuals of different tribes with

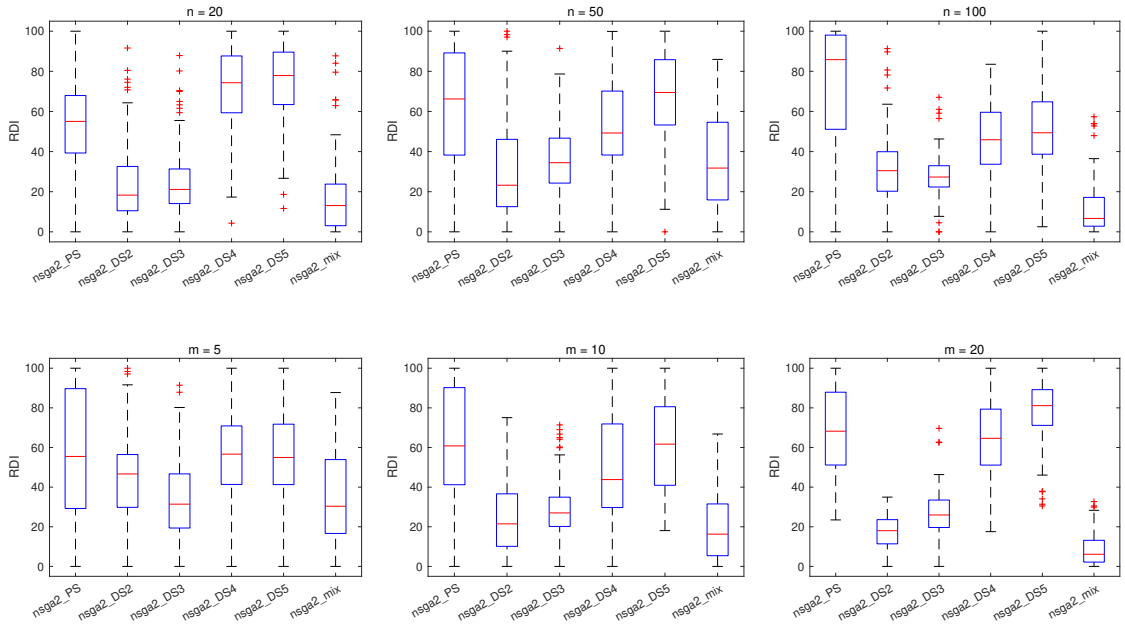


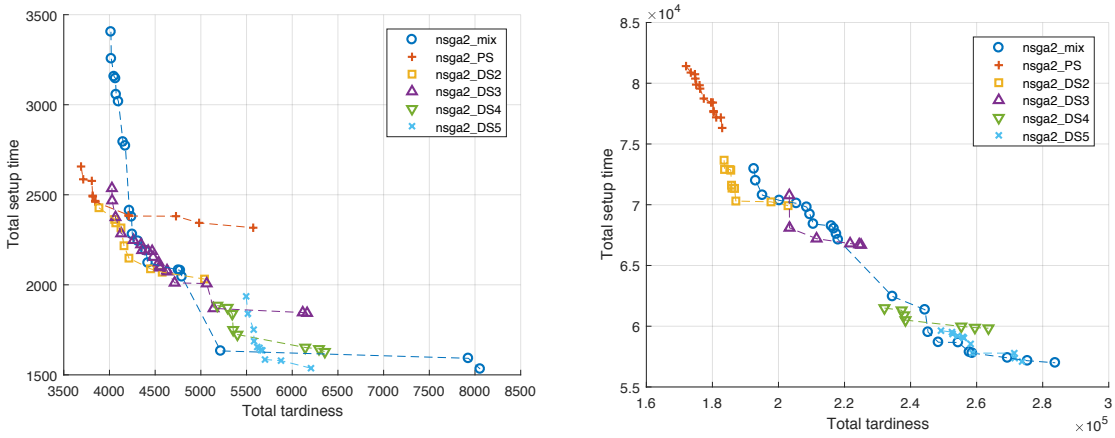
Figure 7: NSGA-II performance comparison with IGD^+

Table 6: Tukey test result of Experiment B (HV)

Group	Algorithm ranking and RDI values					
$n = 20$	NSGA2_mix	NSGA2_DS3	NSGA2_DS2	NSGA2_DS5	NSGA2_DS4	NSGA2_PS
	82.78 (A)	64.07 (B)	60.28 (B)	33.39 (C)	31.81 (C)	23.21 (D)
$n = 50$	NSGA2_mix	NSGA2_DS2	NSGA2_DS3	NSGA2_DS4	NSGA2_DS5	NSGA2_PS
	73.51 (A)	62.08 (B)	58.65 (BC)	53.96 (C)	44.4 (D)	28.56 (E)
$n = 100$	NSGA2_mix	NSGA2_DS3	NSGA2_DS2	NSGA2_DS4	NSGA2_DS5	NSGA2_PS
	89.72 (A)	64.38 (B)	59.59 (BC)	53.36 (CD)	51.41 (D)	22.21 (E)
$m = 5$	NSGA2_mix	NSGA2_DS3	NSGA2_DS5	NSGA2_DS2	NSGA2_DS4	NSGA2_PS
	68.16 (A)	58.27 (B)	47.69 (C)	44.85 (C)	42.17 (C)	34.15 (D)
$m = 10$	NSGA2_mix	NSGA2_DS2	NSGA2_DS3	NSGA2_DS4	NSGA2_DS5	NSGA2_PS
	84.91(A)	65.51(B)	63.13 (B)	54.68 (C)	47.79 (C)	23.5 (D)
$m = 20$	NSGA2_mix	NSGA2_DS2	NSGA2_DS3	NSGA2_DS4	NSGA2_DS5	NSGA2_PS
	92.92 (A)	71.58 (B)	65.69 (C)	42.27 (D)	33.72 (E)	16.32 (F)

Table 7: Tukey test result of Experiment B (IGD^+)

Group	Algorithm ranking and RDI values					
$n = 20$	NSGA2_mix	NSGA2_DS2	NSGA2_DS3	NSGA2_PS	NSGA2_DS4	NSGA2_DS5
	17.22 (A)	24.37 (A)	24.37 (A)	56.02 (B)	71.86 (C)	75.01 (C)
$n = 50$	NSGA2_DS2	NSGA2_mix	NSGA2_DS3	NSGA2_DS4	NSGA2_PS	NSGA2_DS5
	29.98 (A)	35.42 (A)	35.47 (A)	51.88 (B)	61.56 (C)	67.63 (C)
$n = 100$	NSGA2_mix	NSGA2_DS3	NSGA2_DS2	NSGA2_DS4	NSGA2_DS5	NSGA2_PS
	11.18 (A)	27.98 (B)	31.46 (B)	45.9 (C)	52.73 (C)	72.17 (D)
$m = 5$	NSGA2_DS3	NSGA2_mix	NSGA2_DS2	NSGA2_DS4	NSGA2_DS5	NSGA2_PS
	32.94 (A)	34.87 (A)	44.8 (B)	56.29 (C)	56.37 (C)	57.04 (C)
$m = 10$	NSGA2_mix	NSGA2_DS2	NSGA2_DS3	NSGA2_DS4	NSGA2_DS5	NSGA2_PS
	20.26 (A)	23.84 (AB)	28.37 (B)	48.9 (C)	61.01 (D)	63.46 (D)
$m = 20$	NSGA2_mix	NSGA2_DS2	NSGA2_DS3	NSGA2_DS4	NSGA2_PS	NSGA2_DS5
	8.69 (A)	17.17 (B)	26.51 (C)	64.44 (D)	69.25 (D)	78 (E)



(a) Small case (SSD100_N20M5_P3)

(b) Large case (SSD100_N100M20_P3)

Figure 8: Comparison of fronts obtained by different NSGA-II versions

obvious clustering features in the objective space, this will be discussed in the next section with details. Indeed, by employing several decoding algorithms, the MDF extends the search dimension of the NSGA-II in the objective space. As a result, the user is provided with more options to align the schedule with the company's actual needs.

2. Although wide spreading, the front by NSGA2_mix does not dominate any other front given by single-decoding versions, and in some specific region it is worse than them. For example in the small case (Figure 8 (a)), if the user can accept only solutions with a total tardiness less than 4100, then the solutions given by NSGA2_PS is better than that by NSGA2_mix. This shows that the front given by PS is actually "deeper" than the front of MDF. Indeed, coupled with the MDF, the NSGA-II is able to explore different portions of the objective space. Yet, given a fixed computational budget, the trade-off between exploration and exploitation, or saying, "width" and "depth", always exits. This explains the phenomenon. It is however reasonable to believe that as the computational budget increases, such gap would be decreased to an acceptable range.
3. In the large case (Figure 8 (b)), the front given by NSGA2_mix seems fail to cover the upper left region where the front of NSGA2_PS locates. The cause will be discussed in the next subsection.

The result of Experiment C is summarized in Table 8. Each row contains the comparison result between NSGA2_mix and the two benchmark algorithms on 10 instances with the same parameters. The value in each cell is the mean value of 50 data (10 instances*5 replications). To provide statistical evidence, besides the mean value, we report in the parenthesis the standard deviation σ, as well as the p-value of the Mann-Whitney U test comparing the incumbent algorithm to the one with the best mean. A p-value less than 0.05 indicates that the algorithm is statistical different from the best. Note that the HV and IGD^+ values are calculated after normalizing the fronts using their Ideal and Nadir points. As shown, the proposed NSGA2_mix outperforms statistically the two benchmark algorithms in most of the instances, in terms of both HV and IGD^+. There are some cases where the RIPG performs the best, like in SSD100_N100M5, SSD200_N50M5 and SSD200_N100M5. We notice that all these cases correspond to a short shop environment with number of stages $m = 5$. Such advantage of RIPG may come from its decoding algorithm, PS, which is shown efficient when m is small [38]. Although identical evaluation budget is given, the algorithms have different CPU times. This is mainly due to the adopted decoding algorithms are different. Indeed, the DSs have higher time complexity than PS. More specifically, the time complexity of DSs is $O(mn(n+h))$, while for PS is $O(mnh)$ [38]. This makes the CPU time of NSGA2_mix higher. In Figure 9 we plot the fronts obtained by different algorithms in medium-length shop. As we can see, given the same evaluation budget, the fronts obtained by different search mechanisms are not similar. As a single-point search algorithm, MOSA tends to focus its search on a relative small area and sometimes leads to much "deeper" fronts than the others, like in SSD200_N50M10_P4. Whilst for NSGA2_mix, by taking advantage of different decoding algorithms, the obtained front is generally more wide-spreading than RIPG and MOSA. In the cases where the machine setup times are short, clear advantage can be seen in the front given

Table 8: Performance comparison between NSGA2_mix and two benchmark algorithms

Instance group	n	m	HV ($\sigma; p$)			IGD^+ ($\sigma; p$)			CPU Time [sec]		
			NSGA2_mix	RIPG	MOSA	NSGA2_mix	RIPG	MOSA	NSGA2_mix	RIPG	MOSA
SSD25	20	5	**0.713 (0.055; -)**	0.564 (0.056; 6.2e-16)	0.451 (0.076; 4.5e-18)	**0.0515 (0.023; -)**	0.147 (0.042; 3e-16)	0.232 (0.065; 5.4e-18)	40.2	20.2	28.6
SSD25	20	10	**0.669 (0.052; -)**	0.498 (0.05; 9.2e-18)	0.414 (0.058; 3.5e-18)	**0.0454 (0.019; -)**	0.178 (0.043; 4e-18)	0.243 (0.061; 3.5e-18)	59.8	34.6	41.4
SSD25	20	20	**0.601 (0.032; -)**	0.413 (0.046; 3.5e-18)	0.332 (0.051; 3.5e-18)	**0.0272 (0.01; -)**	0.241 (0.042; 3.5e-18)	0.306 (0.052; 3.5e-18)	97.5	58.4	65.7
SSD25	50	5	**0.625 (0.072; -)**	0.483 (0.12; 1.5e-09)	0.492 (0.087; 6.6e-12)	**0.116 (0.049; -)**	0.187 (0.077; 1.7e-06)	0.167 (0.055; 4.2e-06)	94.1	49.5	57.9
SSD25	50	10	**0.617 (0.048; -)**	0.411 (0.051; 4.5e-18)	0.411 (0.06; 8.7e-18)	**0.0751 (0.024; -)**	0.197 (0.06; 6.4e-18)	0.183 (0.056; 3.6e-17)	181.0	103.0	116.0
SSD25	50	20	**0.54 (0.041; -)**	0.331 (0.041; 3.5e-18)	0.301 (0.048; 3.5e-18)	**0.0387 (0.018; -)**	0.236 (0.082; 3.5e-18)	0.255 (0.073; 3.5e-18)	358.0	216.0	230.0
SSD25	100	5	**0.594 (0.052; -)**	0.531 (0.085; 1.6e-05)	0.406 (0.075; 1.3e-16)	**0.119 (0.036; 0.088)**	**0.111 (0.048; -)**	0.178 (0.054; 1.6e-08)	287.0	152.0	164.0
SSD25	100	10	**0.573 (0.065; -)**	0.389 (0.079; 5.4e-17)	0.302 (0.044; 3.5e-18)	**0.0814 (0.028; -)**	0.126 (0.056; 7.8e-06)	0.177 (0.036; 6.4e-17)	559.0	319.0	334.0
SSD25	100	20	**0.54 (0.045; -)**	0.314 (0.045; 3.5e-18)	0.246 (0.041; 3.5e-18)	**0.0414 (0.015; -)**	0.186 (0.043; 3.5e-18)	0.233 (0.045; 3.5e-18)	1130.0	685.0	673.0
SSD100	20	5	**0.708 (0.074; -)**	0.572 (0.099; 7.4e-11)	0.384 (0.11; 1.3e-17)	**0.0877 (0.045; -)**	0.166 (0.065; 1.4e-09)	0.315 (0.097; 5.4e-17)	32.8	16.7	22.1
SSD100	20	10	**0.727 (0.062; -)**	0.522 (0.072; 2.8e-17)	0.353 (0.08; 3.5e-18)	**0.0668 (0.031; -)**	0.194 (0.06; 3.8e-17)	0.326 (0.077; 3.8e-18)	51.8	30.0	36.2
SSD100	20	20	**0.731 (0.032; -)**	0.434 (0.032; 3.5e-18)	0.333 (0.05; 3.5e-18)	**0.0452 (0.014; -)**	0.23 (0.043; 3.5e-18)	0.312 (0.058; 3.5e-18)	123.0	78.4	86.9
SSD100	50	5	0.527 (0.13; 0.32)	**0.551 (0.16; -)**	0.453 (0.15; 0.0026)	0.23 (0.11; 0.0045)	**0.174 (0.078; -)**	0.249 (0.11; 0.0003)	126.0	66.5	79.7
SSD100	50	10	**0.638 (0.071; -)**	0.409 (0.057; 1.2e-17)	0.345 (0.068; 4.5e-18)	**0.107 (0.055; -)**	0.207 (0.072; 1.1e-11)	0.262 (0.1; 2.5e-14)	216.0	123.0	131.0
SSD100	50	20	**0.642 (0.041; -)**	0.342 (0.046; 3.5e-18)	0.325 (0.064; 3.5e-18)	**0.0773 (0.025; -)**	0.226 (0.054; 3.5e-18)	0.238 (0.055; 3.5e-18)	394.0	240.0	252.0
SSD100	100	5	0.463 (0.08; 0.00058)	**0.558 (0.15; -)**	0.378 (0.15; 1.2e-08)	0.265 (0.11; 3.1e-11)	**0.115 (0.064; -)**	0.238 (0.099; 1.7e-10)	320.0	168.0	187.0
SSD100	100	10	**0.587 (0.062; -)**	0.4 (0.11; 3.9e-13)	0.254 (0.09; 4.8e-17)	**0.109 (0.056; -)**	0.167 (0.085; 0.00088)	0.285 (0.095; 4e-16)	576.0	331.0	344.0
SSD100	100	20	**0.6 (0.031; -)**	0.279 (0.033; 3.5e-18)	0.19 (0.054; 3.5e-18)	**0.0806 (0.021; -)**	0.207 (0.041; 3.5e-18)	0.277 (0.075; 3.5e-18)	1110.0	656.0	672.0
SSD200	20	5	**0.662 (0.11; -)**	0.642 (0.1; 0.18)	0.37 (0.16; 1.1e-13)	**0.157 (0.08; 0.38)**	**0.15 (0.07; -)**	0.387 (0.17; 9.5e-13)	33.3	16.2	21.2
SSD200	20	10	**0.748 (0.082; -)**	0.579 (0.1; 2.5e-12)	0.374 (0.1; 3.5e-18)	**0.0935 (0.048; -)**	0.171 (0.074; 2.6e-08)	0.334 (0.1; 7.2e-18)	50.7	29.2	34.7
SSD200	20	20	**0.702 (0.068; -)**	0.463 (0.069; 1.2e-17)	0.329 (0.074; 3.5e-18)	**0.08 (0.031; -)**	0.222 (0.058; 1.4e-17)	0.334 (0.06; 3.5e-18)	92.3	58.8	67.2
SSD200	50	5	0.481 (0.15; 1.2e-05)	**0.634 (0.17; -)**	0.484 (0.18; 0.0001)	0.342 (0.17; 6.2e-07)	**0.183 (0.15; -)**	0.3 (0.19; 0.00014)	99.2	50.0	59.3
SSD200	50	10	**0.574 (0.078; -)**	0.535 (0.089; 0.017)	0.399 (0.15; 1.2e-08)	**0.168 (0.058; -)**	0.187 (0.064; 0.11)	0.295 (0.11; 6.3e-09)	184.0	101.0	113.0
SSD200	50	20	**0.624 (0.054; -)**	0.41 (0.044; 3.5e-18)	0.352 (0.059; 3.5e-18)	**0.138 (0.03; -)**	0.234 (0.037; 1.7e-16)	0.275 (0.052; 3.5e-18)	353.0	226.0	237.0
SSD200	100	5	0.335 (0.083; 6.4e-18)	**0.723 (0.1; -)**	0.418 (0.17; 1.6e-14)	0.431 (0.098; 4.5e-18)	**0.0818 (0.057; -)**	0.289 (0.17; 2e-13)	296.0	151.0	170.0
SSD200	100	10	**0.538 (0.073; -)**	0.453 (0.053; 2.3e-09)	0.282 (0.084; 6.4e-18)	0.2 (0.051; 1.3e-05)	**0.149 (0.06; -)**	0.276 (0.083; 6.3e-12)	649.0	376.0	372.0
SSD200	100	20	**0.584 (0.046; -)**	0.358 (0.041; 3.5e-18)	0.248 (0.057; 3.5e-18)	**0.143 (0.035; -)**	0.216 (0.082; 2.1e-06)	0.294 (0.073; 3.8e-17)	1240.0	749.0	755.0
Average			**0.605 (0.12; -)**	0.474 (0.14; 9e-140)	0.357 (0.12; 8.9e-316)	**0.127 (0.11; -)**	0.181 (0.078; 1e-95)	0.269 (0.11; 5.4e-233)	324.0	189.0	198.0

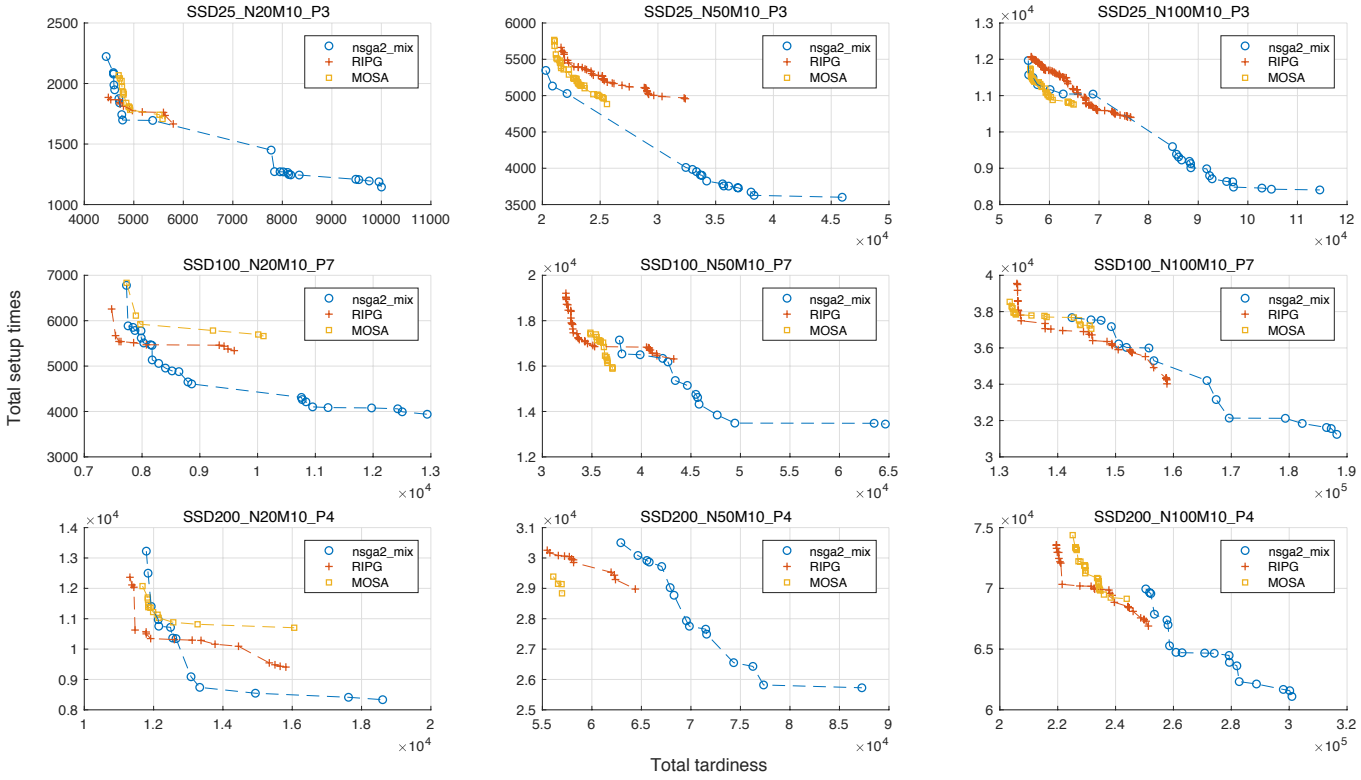


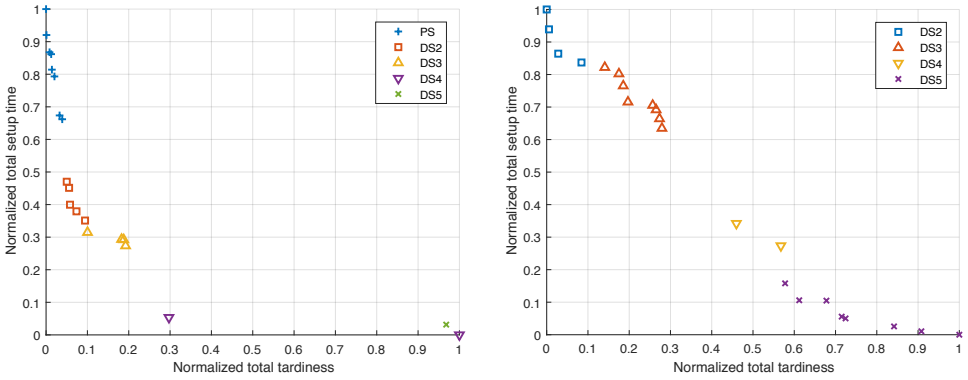
Figure 9: Comparison of fronts obtained by NSGA2_mix and two benchmark algorithms

by NSGA2_mix; yet when long setup times are incurred, for example in SSD200_N100M10_P4, NSGA2_mix focuses more on reducing the total setup time whilst losses the coverage of the upper left region. This looks quite similar to the problem aforementioned in Figure 8 (b).

5.3.4. Issue and discussion

We investigate how the tribe members are distributed in the front given by NSGA2_mix. As shown in Figure 10, the distribution of the tribes in the NSGA2_mix front is consistent with the previous analysis on the tribe fitness both in small case (a) and large case(b). From PS to DS5, the fitness bias from the total tardiness to the total setup time objective. However, we also notice that the PS tribe is missing from the final front of the large case (b). Actually, this is not occasional but it occurs in almost all media ($n = 50$) and large instances ($n = 100$) in the instance group of SSD100. This also explains the fact that in the large case (Figure 8 (b)) the NSGA2_mix fails to cover the upper left region. This issue we call the loss of potentially promising tribes.

For better understanding this phenomenon, we plot the tribe size, averaged tribe objective values during the evolution procedure in Figure 11. We summarize the experiment data for the large instances ($n = 100$) of group SSD100 in Figure 11 (b)(d)(f). Each point is the mean value from 30 (instances) * 5 (replications) = 150 runs. As shown in (b), the averaged tribe size of PS decreases dramatically just after the GA starts (the initial percentage is about 20%) and maintains at the preservation percentage (5%). From (d) and (f) it is observed that the tribe PS has a low total tardiness value but the highest total setup time makes it less competitive. Indeed, it is observed from previous experiment (Figure 4) that the PS is the fittest tribe for the total tardiness objective, and ideally, the PS individuals should be able to built up some part of the final



(a) Small case (SSD100_N20M5_P3) (b) Large case (SSD100_N100M20_P3)

Figure 10: Tribe distribution in the non-dominated front output by NSGA2_mix

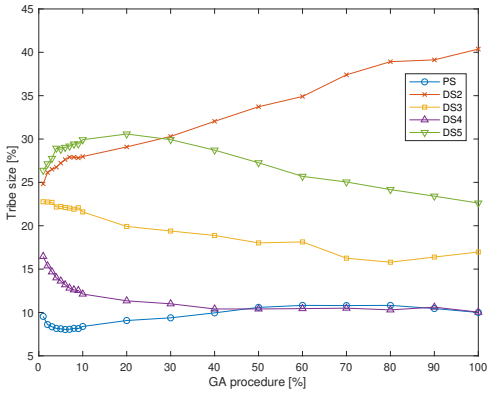
front. Yet, the dramatic decrease of the tribe size in the initial phase heavily prevents the future allocation of computing resource to search the space represented by the tribe PS. This suppresses the development of tribe PS.

Obviously, it requires a better tribe size management strategy to avoid suppressing the potentially promising tribe. For example, the preservation percentage δ could follow a certain pattern like the cooling temperature curve in simulated annealing, instead of a simple constant. This problem, in its nature, relates to the topic of how to efficiently allocate the computing budget to search the different regions of the objective space and how to improves the budget allocation through learning during the optimization procedure.

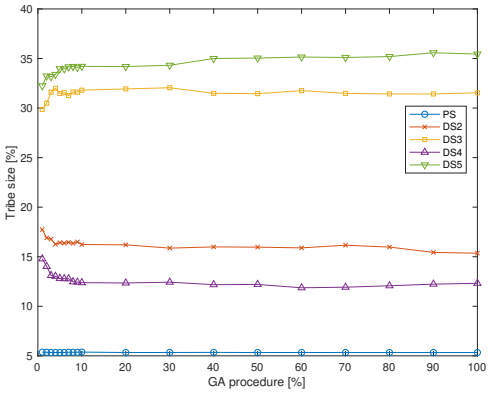
Another issue is observed in the comparison between Figure 11(e) and (f). As seen, the overall reduction of the total setup time by the GA procedure is quite obvious in the small instances; whilst in the large instances, such reduction is almost negligible. More specifically, the RPI reduction of the total setup time of the tribe DS5 is about 22, 8 and 4 for the small, medium ($n = 50$) and large instances, respectively. This implies that the more jobs there are, the more difficult for the algorithm to reduce the total setup time. This is caused by the job-permutation encoding scheme. Indeed, the setup time incurred by a given job sequence is different in the stages. Keeping the same job sequence or priority for each stage limits the possibility to minimize the overall setup time in the system, and the difficulty of reducing the total setup time increases with the number of jobs. To solve this, one may adopt the operation-based encoding scheme, which, however, increases dramatically the cardinality of the search space. For this reason, the shift-representation approach proposed in Urlings et al. [52] is a possible solution. This approach first adopts the indirect solution representation, like job-permutation, for an efficient initial search, then at a specific time, switches to full representation for a more thorough search.

6. Conclusion

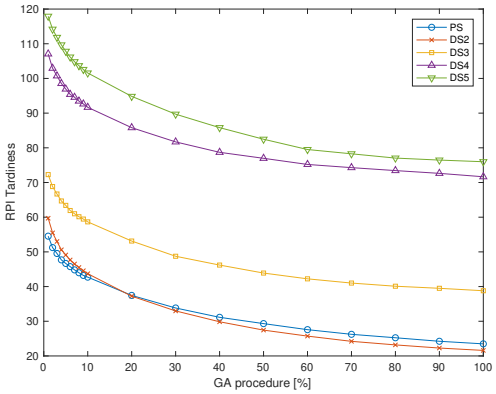
In this research, we consider a HFS scheduling problem with sequence-dependent setup times (SDST) to minimize the total tardiness and total setup time. We have extended the schedule construction algorithm proposed in Yu et al. [38] to HFS with SDST, obtaining several versions



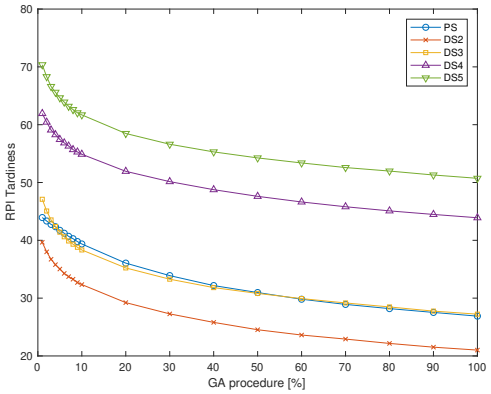
(a) Tribe size, $n = 20$



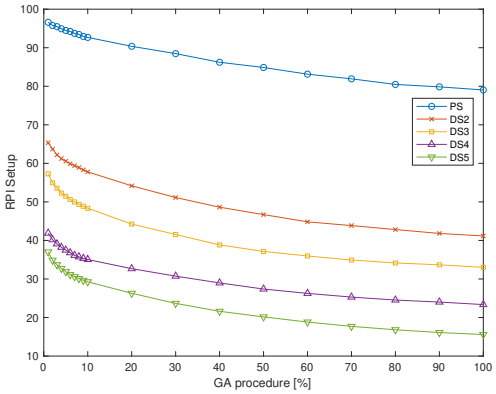
(b) Tribe size, $n = 100$



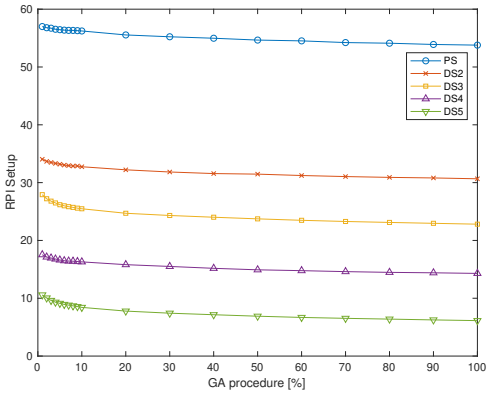
(c) Total tardiness, $n = 20$



(d) Total tardiness, $n = 100$



(e) Total setup time, $n = 20$



(f) Total setup time, $n = 100$

Figure 11: Tribe size, total tardiness and total setup time variation during the running of NSGA2_mix

which favor the objective functions to different extent. Compared to the conventional decoding algorithm, these are shown performing better when the user pursue more on reducing the machine setup time in the system. Then, we propose a Multi-Decoding Framework (MDF) for taking advantage of different decoding algorithms. After hybridizing with the MDF, EAs function as high-level hyper-heuristics that search in the joint space of low-level heuristic and encoded solution. The usefulness of the MDF has been shown by integrating it to both a genetic algorithm and a multi-objective evolutionary algorithm, NSGA-II, to solve the HFS problem with *a priori* and *a posteriori* approach, respectively. For *a priori* approach, the genetic algorithm using the proposed MDF is able to adjust the adopted decoding algorithm during the evolution, and provides efficient solution aligned to user preference. For *a posteriori* approach, compared to the NSGA-II versions with single decoding algorithm, the NSGA-II using the proposed MDF is shown to generate a non-dominated front with better quality in terms of both hyper volume and distance to the reference front. This provides more alternatives for the posterior decision-making procedure. Finally, the NSGA-II coupled with the proposed MDF is shown superior than two benchmark algorithms on most of the test instances.

To the best of our knowledge, this is the first report to apply EAs with multiple decoding algorithms to the multi-objective scheduling problem in HFS with SDST. The necessity of employing multiple decoding algorithms rises from the fact that they map the design space to different regions of the multi-dimensional objective space and favor different user preferences. The proposed multi-decoding framework is simple and useful, yet some drawbacks are observed from the numerical experiments when solving the problem in *a posteriori* approach: first, the use of MDF extends the search space, yet, the front given by the framework may be worse than some single-decoding method in some specific region of the objective space. This is due to the trade-off between the search "width" and "depth" given limited computing resources. One idea to tackle this is to incorporate the user preference information into the framework, which allows it to narrow down the search space and focus on the specific region interests the user. Secondly, the key problem of MDF is how to allocate the computing budget to search the subspace represented by different heuristics (decoding algorithms). It is observed that EAs, when employed as hyper-heuristics, tend to allocate more computing budget to the subspace where improvement is more readily at the initial search phase. Whilst for the subspace which contains high-quality solutions but requires more efforts to find, due to the little allocated budget, the search becomes quite difficult after losing the competition against others at the initial search phase. As a consequence, it would lose the opportunity to discover the non-dominated solutions in that subspace. To tackle this problem, it requires a mechanism which allocates the search budget in a more balanced way in terms of exploitation and exploration. We will look for the answer from the reinforcement learning literature as a future work.

Acknowledgements

The authors thank the referee and AE for their comments and suggestions that have improved the quality of the paper. The authors also thank Dr. Michele Ciavotta for his help on

the benchmark algorithm. This work was supported, in part, by the project HPM-CLUSTER (CTN01_00163_216758) of MIUR Italy.

Appendix A. Calculation of machine selection metrics

Denote j^* as the job to assign at system clock t, i as the index of stage, l as the index for the machine in stage i, the machine selection metrics of the candidate machine l are calculated as below:

- Buffer Total Processing Time (BTPT): Let $\mathscr{B}_l$ be the upstream machine buffer of machine l. BTPT is the summation of processing times of all jobs waiting in $\mathscr{B}_l$ at time t, i.e., BTPT $= \sum_{j \in \mathscr{B}_l} p_{ilj}$.
- Processing time (PT): The processing time of job j^*, i.e., PT $= p_{ilj^*}$.
- Machine Time to Idle (MTTI): The duration before the machine returns to idle. Let a_l be the expected release time of the current processing job, MTTI $= a_l - t$.
- Machine Total Setup Times (MTST): Let $\mathscr{B}_l$ be the upstream buffer of machine l, job (0) be the current processing job. Let $\mathscr{B}'_l \leftarrow \mathscr{B}_l \cup \{j^*\}$ be the updated buffer after j^* is assigned. Sort the jobs in $\mathscr{B}'_l$ by job priorities and we get $\{(1), (2), \ldots, (|\mathscr{B}'_l|)\}$. Then, MTST $= \sum_{j=0}^{|\mathscr{B}'_l|-1} S_{il(j)(j+1)}$. If machine l is idle at time t, job (0) refers to the latest processed job.
- Machine Differential Setup Times (MDST): The difference of total setup time on machine l after assigning job j^* to its buffer. Let $\mathscr{B}_l$ and $\mathscr{B}'_l$ be the current buffer content and that after assigning job j^*, MDST $= \sum_{j=0}^{|\mathscr{B}'_l|-1} S_{il(j)(j+1)} - \sum_{j=0}^{|\mathscr{B}_l|-1} S_{il(j)(j+1)}$. If machine l is idle at time t, the second term of the formula is 0.

Appendix B. Empirical comparison of decoding algorithms

In this section, the details of the empirical comparison between decoding algorithms are given. For each instance in the group SSD100, 20000 random job sequences are created. These job sequences are decoded by the six decoding algorithms, i.e., PS, DS, DS2, DS3, DS4, DS5, respectively. Then, the obtained performance metrics, i.e., total tardiness and total setup time, are converted to the RPI values using Equation 19. For each decoding algorithm, the RPI values on all 90 instances are collected, we have in total 90*20000 = 1800000 values for each objective function. Based on these values, an empirical cumulative distribution function (CDF) can be plotted. This CDF shows the general performance of the decoding algorithm on the tested instances.

The empirical CDFs of the decoding algorithms are plotted in Figure B.12. As shown, different decoding algorithms have different performance. For total tardiness, DS2 seems to be the best one, because its CDF increases the fastest. This means the solutions decoded by DS2 tends to have smaller RPI values than others. In this way, the performance of decoding algorithms in terms of total tardiness can be ranked as DS2, DS, PS, DS3, DS4, DS5; whilst for total setup time the ranking is DS5, DS4, DS3, DS2, PS, DS. It should be noted that the aim of this empirical comparison is to provide general information on the decoding algorithms' performance. The ranking given by

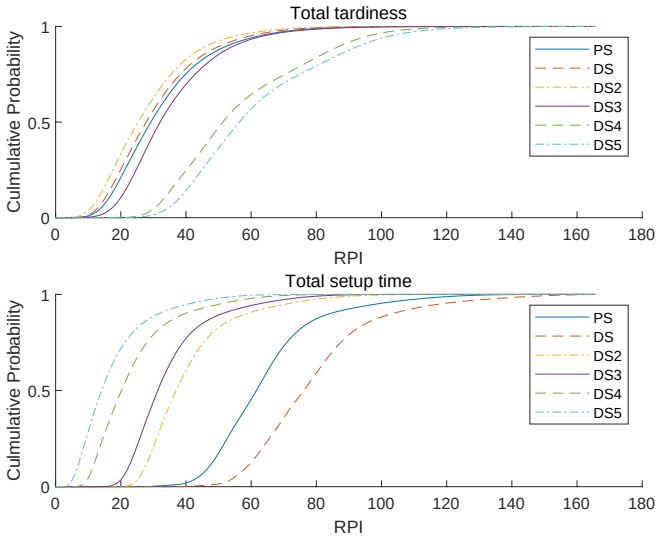


Figure B.12: Empirical cumulative distribution function of the RPI values of decoding algorithms

this empirical comparison is not exact, because the sample size (20000) is still much smaller than the cardinality of the corresponding permutation space.

Appendix C. Details of statistical tests

In this section, we provide the details of statistical analysis on the results of Experiment A, B and C. For Experiment A, the ANOVA is performed by adopting a reduced model using w, *algorithm*, n, m and all their second-order interactions as factors. Higher order interactions are not included, this is to prevent overfitting, and to preserve the model generality to some extent. Note that we averaged the responses from replications, and used responses from different *instances* as source of variability. The ANOVA table is reported in Figure C.13(a). As shown, the model fits the data well with a high R-sq(adj) value of 88%. Although model assumptions (equal variance, normality) are slightly violated due to the enormous amount of data, the results of the F-test are still considered robust. As shown, all factors are significant, *algorithm* and w*$algorithm$ appear to be the most influential factors due to their large F-Value. This means that the algorithm rankings are mainly affected by the user preference whilst less influenced by the problem size n and the shop length m.

To analyze the algorithm rankings under different user preference, a pairwise comparison has to be made. Many pairwise comparison methods are available, including T-Bonferroni, Scheffé, and Tukey test. Here, we choose the Tukey test for its ability to control the overall error rate [53]. The details of Tukey test is as follows. For the second-order interaction w*$algorithm$, there are in total $6 * 6 = 36$ levels. For any two levels, say i-th and l-th, the following relationship holds:

$$\text{Prob}\{\frac{|\hat{D}_{il} - D_{il}|}{\sqrt{MS_E/ns}} \leq q_\alpha(a, df_E)\} \geq 1 - \alpha,$$

where $\hat{D}_{il}$ and D_{il} are the difference between the sample means and true means of level i and l, respectively. MS_E is the mean of squared error term from the ANOVA table, and ns is the number of samples of the level, $q_\alpha(a, df_E)$ is the $1 - \alpha$ quantile of the studentized range distribution, a is

the number of levels, df_E is the degree of freedom of errors. It can be shown that two levels are significantly different if the absolute value of their sample differences exceeds

$$T_\alpha = q_\alpha(a, df_E)\sqrt{\frac{MS_E}{ns}}.$$

See Montgomery [53] for more details of Tukey test. In the case of Experiment A, $MS_E = 88, ns = 90, a = 36, df_E = 3156$, $q_{0.05}(36, 3156) = 5.42$, so $T_{0.05} = 5.36$. As a result, any levels in Table 5 with difference greater than 5.36 is statistically different. For example, when $w = 0$, GA_mix is not statistically different from GA_DS5 because their difference $|11.44 - 10.41| \leq 5.36$. We mark these two with the same letter A; whilst GA_mix is statistically better than the remaining algorithms, which are marked with letters different than A.

For Experiment B, the ANOVA model uses *algorithm*, n, m and their second-order interactions as factors. The ANOVA tables of RDI_HV and RDI_IGD$^+$ are reported in Figure C.13(b) and (c), respectively. As shown, for both indicators, the terms *algorithm**n and *algorithm**m are significant, showing that the algorithm rankings are different under distinct problem size and shop length. To investigate this, four Tukey tests are performed (two for each indicator). With the aforementioned approach, we obtain the significant gap as $T_{0.05} = 9.025$ for the levels of factor *algorithm**n and *algorithm**m in terms of RDI_HV, and $T_{0.05} = 11.9$ for RDI_IGD$^+$. These lead to the results summarized in Table 6 and Table 7.

In Experiment C, the proposed NSGA2_mix shows advantage in most of the instance groups by a better mean. To confirm such advantage with statistical evidence, we implement a simple approach, i.e., in each group we compare the algorithm to the one with the best sample mean using hypothesis test. For each instance group in Table 8, an algorithm produces 10(instances) * 5(replications) = 50 results. To compared two data groups, *t-test* is the common method. However, it requires the group data are normally distributed. This is not necessary true in our case, because the result variability from different instances may be greater than that from replications. To overcome this, we choose the *Mann-Whitney U test*. This test has greater efficiency than *t-test* on non-normal distributions and, in most of the cases, it is nearly as efficient as the *t-test* on normal distributions. Let us consider the HV indicator, and let Algorithm B be the one with the better mean. The test for an Algorithm, say A, works with the null hypothesis H_0 : {Algorithm A has equal HV with B}, and alternative hypothesis H_1 : {Algorithm A has worse HV than B}. Let $A_1, A_2, \ldots, A_m$ and $B_1, B_2, \ldots, B_n$ be the results by the two algorithm, and define $D_{ij} = \mathbf{1}_{(B_j < A_i)}$, where $\mathbf{1}_{(\mathscr{S})}$ is the indicator function equal to 1 if statement $\mathscr{S}$ is true, otherwise 0. The Mann-whitney U statistic is given by

$$U = \sum_{i=1}^{m}\sum_{j=1}^{n} D_{ij}.$$

When n and m are large enough, the following large-sample test statistic

$$Z = \frac{U - mn/2}{\sqrt{mn(n+m+1)/12}}$$

approximately follows the standard normal distribution $\mathscr{N}(0, 1)$. See Gibbons and Chakraborti

(a) Experiment A

Source	DF	Adj SS	Adj MS	F-Value	P-Value
w	5	14801	2960	33.77	0.000
algorithm	5	881727	176345	2011.67	0.000
n	2	2293	1147	13.08	0.000
m	2	22462	11231	128.12	0.000
w*algorithm	25	1263959	50558	576.75	0.000
w*n	10	2527	253	2.88	0.001
w*m	10	15218	1522	17.36	0.000
algorithm*n	10	5068	507	5.78	0.000
algorithm*m	10	33744	3374	38.49	0.000
n*m	4	919	230	2.62	0.033
Error	3156	276659	88		
Total	3239	2519376			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
9.36275	89.02%	88.73%	88.43%

(b) Experiment B (RDI_HV)

Source	DF	Adj SS	Adj MS	F-Value	P-Value
algorithm	5	95603	19120.6	190.32	0.000
n	2	5147	2573.6	25.62	0.000
m	2	5138	2569.1	25.57	0.000
algorithm*n	10	10472	1047.2	10.42	0.000
algorithm*m	10	16760	1676.0	16.68	0.000
n*m	4	1227	306.7	3.05	0.017
Error	506	50835	100.5		
Total	539	185183			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
10.0232	72.55%	70.76%	68.74%

(c) Experiment B (RDI_IGD+)

Source	DF	Adj SS	Adj MS	F-Value	P-Value
algorithm	5	175007	35001.4	200.31	0.000
n	2	4278	2138.9	12.24	0.000
m	2	3326	1663.1	9.52	0.000
algorithm*n	10	30903	3090.3	17.69	0.000
algorithm*m	10	33777	3377.7	19.33	0.000
n*m	4	1526	381.5	2.18	0.070
Error	506	88416	174.7		
Total	539	337233			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
13.2187	73.78%	72.07%	70.14%

Figure C.13: ANOVA tables

[54] for details. Let $\Phi(x)$ be the CDF of the standard normal distribution, then the p-value of the test is obtained by $p = 1 - \Phi(Z)$. A p-value less than 0.05 leads to the rejection of H_0.

References

[1] R. Ruiz, J. A. Vázquez-Rodríguez, The hybrid flow shop scheduling problem, European journal of operational research 205 (1) (2010) 1–18.

[2] J. N. Gupta, Two-stage, hybrid flowshop scheduling problem, Journal of the operational Research Society 39 (4) (1988) 359–364.

[3] F. Dugardin, F. Yalaoui, L. Amodeo, New multi-objective method to solve reentrant hybrid flow shop scheduling problem, European Journal of Operational Research 203 (1) (2010) 22–31.

[4] S. Wang, M. Liu, Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method, International Journal of Production Research 52 (5) (2014) 1495–1508.

[5] N. Karimi, M. Zandieh, H. Karamooz, Bi-objective group scheduling in hybrid flexible flowshop: a multi-phase approach, Expert Systems with Applications 37 (6) (2010) 4024–4032.

[6] C. Lu, L. Gao, X. Li, S. Xiao, A hybrid multi-objective grey wolf optimizer for dynamic scheduling in a real-world welding industry, Engineering Applications of Artificial Intelligence 57 (2017) 61–79.

[7] H. Luo, B. Du, G. Q. Huang, H. Chen, X. Li, Hybrid flow shop scheduling considering machine electricity consumption cost, International Journal of Production Economics 146 (2) (2013) 423–439.

[8] Z. Zeng, M. Hong, Y. Man, J. Li, Y. Zhang, H. Liu, Multi-object optimization of flexible flow shop scheduling with batch process?consideration total electricity consumption and material wastage, Journal of cleaner production 183 (2018) 925–939.

[9] M. Pinedo, Scheduling, Vol. 29, Springer, 2012.

[10] A. Allahverdi, The third comprehensive survey on scheduling problems with setup times/costs, European Journal of Operational Research 246 (2) (2015) 345–378.

[11] S. C. Trovinger, R. E. Bohn, Setup time reduction for electronics assembly: Combining simple (smed) and it-based methods, Production and operations management 14 (2) (2005) 205–217.

[12] E. K. Burke, G. Kendall, et al., Search methodologies, Springer, 2005.

[13] M. K. Marichelvam, T. Prabaharan, X. S. Yang, A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems, IEEE transactions on evolutionary computation 18 (2) (2013) 301–305.

[14] S. Mousavi, M. Zandieh, M. Yazdani, A simulated annealing/local search to minimize the makespan and total tardiness on a hybrid flowshop, The International Journal of Advanced Manufacturing Technology 64 (1-4) (2013) 369–388.

[15] F. Pargar, M. Zandieh, Bi-criteria sdst hybrid flow shop scheduling with learning effect of setup times: water flow-like algorithm approach, International Journal of Production Research 50 (10) (2012) 2609–2623.

[16] Q.-K. Pan, R. Ruiz, P. Alfaro-Fernández, Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows, Computers & Operations Research 80 (2017) 50–60.

[17] O. Shahvari, R. Logendran, Hybrid flow shop batching and scheduling with a bi-criteria objective, International Journal of Production Economics 179 (2016) 239–258.

[18] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, F. Werner, A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria, Computers & Operations Research 36 (2) (2009) 358–378.

[19] Y. H. YV, L. S. Lasdon, D. DA WISMER, On a bicriterion formation of the problems of integrated system identification and system optimization, IEEE Transactions on Systems, Man and Cybernetics (3) (1971) 296–297.

[20] J. D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, in: Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 1985, Lawrence Erlbaum Associates. Inc., Publishers, 1985.

[21] D. E. Goldberg, Messy genetic algorithms: Motivation analysis, and first results, Complex systems 4 (1989) 415–444.

[22] C. M. Fonseca, P. J. Fleming, et al., Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization., in: Icga, Vol. 93, 1993, pp. 416–423.

[23] J. rey Horn, N. Nafpliotis, D. E. Goldberg, A niched pareto genetic algorithm for multiobjective optimization, in: Proceedings of the first IEEE conference on evolutionary computation, IEEE world congress on computational intelligence, Vol. 1, Citeseer, 1994, pp. 82–87.

[24] N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary computation 2 (3) (1994) 221–248.

[25] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach, IEEE transactions on Evolutionary Computation 3 (4) (1999) 257–271.

[26] J. D. Knowles, D. W. Corne, Approximating the nondominated front using the pareto archived evolution strategy, Evolutionary computation 8 (2) (2000) 149–172.

[27] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: Nsga-ii, IEEE transactions on evolutionary computation 6 (2) (2002) 182–197.

[28] E. Zitzler, M. Laumanns, L. Thiele, Spea2: Improving the strength pareto evolutionary algorithm, TIK-report 103 (2001).

[29] M. Ebrahimi, S. F. Ghomi, B. Karimi, Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates, Applied Mathematical Modelling 38 (9-10) (2014) 2490–2504.

[30] S. H. Abyaneh, M. Zandieh, Bi-objective hybrid flow shop scheduling with sequence-dependent setup times and limited buffers, The International Journal of Advanced Manufacturing Technology 58 (1-4) (2012) 309–325.

[31] J. Behnamian, S. F. Ghomi, M. Zandieh, A multi-phase covering pareto-optimal front method to multi-objective scheduling in a realistic hybrid flowshop using a hybrid metaheuristic, Expert Systems with Applications 36 (8) (2009) 11057–11069.

[32] H. Asefi, F. Jolai, M. Rabiee, M. T. Araghi, A hybrid nsga-ii and vns for solving a bi-objective no-wait flexible flowshop scheduling problem, The International Journal of Advanced Manufacturing Technology 75 (5-8) (2014) 1017–1033.

[33] M. Zandieh, S. M. Sajadi, R. Behnoud, Integrated production scheduling and maintenance planning in a hybrid flow shop system: a multi-objective approach, International Journal of System Assurance Engineering and Management 8 (2) (2017) 1630–1642.

[34] J.-q. Li, H.-y. Sang, Y.-y. Han, C.-g. Wang, K.-z. Gao, Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions, Journal of Cleaner Production 181 (2018) 584–598.

[35] S. Schulz, J. S. Neufeld, U. Buscher, A multi-objective iterated local search algorithm for comprehensive energy-aware hybrid flow shop scheduling, Journal of Cleaner Production 224 (2019) 421–434.

[36] V. Fernandez-Viagas, P. Perez-Gonzalez, J. M. Framinan, Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective, Computers & Operations Research 109 (2019) 77–88.

[37] R. Ruiz, C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility, European Journal of Operational Research 169 (3) (2006) 781–800.

[38] C. Yu, Q. Semeraro, A. Matta, A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility, Computers & Operations Research 100 (2018) 211–229.

[39] M. Gendreau, J.-Y. Potvin, et al., Handbook of metaheuristics, Vol. 2, Springer.

[40] H.-L. F. P. Ross, D. Corne, A promising hybrid ga/heuristic approach for open-shop scheduling problems, in: Proc. 11th European Conference on Artificial Intelligence, 1994, pp. 590–594.

[41] J. V. Rodríguez, S. Petrovic, A. Salhi, A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines, in: Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications. MISTA: Paris, France, 2007, pp. 506–513.

[42] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper-heuristic for educational timetabling problems, European Journal of Operational Research 176 (1) (2007) 177–192.

[43] K. A. Dowsland, E. Soubeiga, E. Burke, A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation, European Journal of Operational Research 179 (3) (2007) 759–774.

[44] R. Qu, E. K. Burke, Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems, Journal of the Operational Research Society 60 (9) (2009) 1273–1285.

[45] V. Yaurima, L. Burtseva, A. Tchernykh, Hybrid flowshop with unrelated machines, sequence-dependent setup time, availability constraints and limited buffers, Computers & Industrial Engineering 56 (4) (2009) 1452–1463.

[46] J.-P. Li, M. E. Balazs, G. T. Parks, P. J. Clarkson, A species conserving genetic algorithm for multimodal function optimization, Evolutionary computation 10 (3) (2002) 207–234.

[47] B. Naderi, M. Zandieh, A. K. G. Balagh, V. Roshanaei, 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 36 (6) (2009) 9625–9633.

[48] K. I. Smith, R. M. Everson, J. E. Fieldsend, C. Murphy, R. Misra, Dominance-based multiobjective simulated annealing, IEEE Transactions on Evolutionary Computation 12 (3) (2008) 323–342.

[49] M. Ciavotta, G. Minella, R. Ruiz, Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study, European Journal of Operational Research 227 (2) (2013) 301–313.

[50] T. Okabe, Y. Jin, B. Sendhoff, A critical survey of performance indices for multi-objective optimisation, in: The 2003 Congress on Evolutionary Computation, 2003. CEC'03., Vol. 2, IEEE, 2003, pp. 878–885.

[51] H. Ishibuchi, H. Masuda, Y. Tanigaki, Y. Nojima, Modified distance calculation in generational distance and inverted generational distance, in: International Conference on Evolutionary Multi-Criterion Optimization, Springer, 2015, pp. 110–125.

[52] T. Urlings, R. Ruiz, T. Stützle, Shifting representation search for hybrid flexible flowline problems, European Journal of Operational Research 207 (2) (2010) 1086–1095.

[53] D. C. Montgomery, Design and analysis of experiments, John wiley & sons, 2017.

[54] J. D. Gibbons, S. Chakraborti, Nonparametric Statistical Inference: Revised and Expanded, CRC press, 2014.