Al Aqel *et al. Chin. J. Mech. Eng.* (2019) 32:21
https://doi.org/10.1186/s10033-019-0337-7

Chinese Journal of Mechanical Engineering

ORIGINAL ARTICLE Open Access

A Modified Iterated Greedy Algorithm for Flexible Job Shop Scheduling Problem

Ghiath Al Aqel, Xinyu Li and Liang Gao*

Abstract

The flexible job shop scheduling problem (FJSP) is considered as an important problem in the modern manufacturing system. It is known to be an NP-hard problem. Most of the algorithms used in solving FJSP problem are categorized as metaheuristic methods. Some of these methods normally consume more CPU time and some other methods are more complicated which make them difficult to code and not easy to reproduce. This paper proposes a modified iterated greedy (IG) algorithm to deal with FJSP problem in order to provide a simpler metaheuristic, which is easier to code and to reproduce than some other much more complex methods. This is done by separating the classical IG into two phases. Each phase is used to solve a sub-problem of the FJSP: sequencing and routing sub-problems. A set of dispatching rules are employed in the proposed algorithm for the sequencing and machine selection in the construction phase of the solution. To evaluate the performance of proposed algorithm, some experiments including some famous FJSP benchmarks have been conducted. By compared with other algorithms, the experimental results show that the presented algorithm is competitive and able to find global optimum for most instances. The simplicity of the proposed IG provides an effective method that is also easy to apply and consumes less CPU time in solving the FJSP problem.

Keywords: Iterated greedy, Flexible job shop scheduling problem, Dispatching rules

1 Introduction

The optimization of production scheduling can bring in considerable improvements in the manufacturing efficiency [1]. Job-shop scheduling problem (JSP) is basically an NP-complete challenge [2]. JSP considers no flexibility of any resources (such as machines and tools) for each operation [1].

Modern manufacturing systems contain many flexible machines to increase the production efficiency. These machines are capable of processing several types of the operations. This gives the permission to break the definition of JSP where an operation can be processed by a single machine [1].

As an extension of the JSP, Flexible Job-shop Scheduling Problem (FJSP), which is known to be NP-hard [3], considers the flexible machines for each operation. Furthermore, the FJSP is considered to be more complicated in comparison to the traditional JSP, as it dictates an extra decision level for the same scale in addition to the sequencing one, such as the operations routes [2].

Many approaches have been proposed to solve FJSP since first presented in 1990 [4]. The current methods for solving FJSP can be mainly categorized into; exact algorithm, dispatching rules (DR), evolutionary algorithm (EA), swarm intelligence (SI) based approaches, local search (LS) algorithms, and so on [5].

While exact algorithms tend to be inefficient with large scale FJSP, other methods, such as EA and SI, are much more expensive regarding the consumption of computation time. Many of these algorithms are also complicated and in many cases they are too difficult to reproduce. This makes it difficult to apply these methods in real-life problems.

In this research, we try to present a modified iterated greedy (IG) algorithm, a simpler metaheuristic that is easier to code and reproduce. The classical IG is separated into two phases to deal with the two sub-problems

*Correspondence: gaoliang@mail.hust.edu.cn
State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

of FJSP and it's combined with a set of dispatching rules (DRs) to solve the FJSP. The simplicity and the efficiency of IG and DRs shall result in an effective method which consumes less computation time and is easy to implement.

The remainder of this paper is arranged as follows. Literature review and problem definition are presented in Section 2. IG approach and the modified iterated greedy (MIG) is proposed in Section 3. Experimental studies are discussed in Section 4. Section 5 provides the conclusions and future work.

2 Literature Review and Problem Definition

2.1 Literature Review

Several methods have been used to deal with the FJSP. These techniques are classified into two groups; the exact methods and the approximation methods [1]. Exact algorithms include mathematical programming (MP), while the approximation algorithms include some dispatching rules (DRs) and artificial intelligence (AI) based approaches.

Brucker and Schlie [4] proposed a polynomial graphical algorithm when they first presented the FJSP with two jobs. Demir and İşleyen [6] evaluated some mathematical models of the FJSP. However, it's been proved by Pezzella et al. [7] that exact algorithms are ineffective when dealing with large scale problems of FJSP.

Baykasoğlu and Özbakır [8] analyzed the effects of several DRs on the scheduling performance of job shops with different levels of flexibility and with different sizes of the problem. They proved that the performances of these DRs were approximately similar when dealing with high machine flexibility. On the other hand, different performances were obtained for zero machine flexibility. Ingimundardottir and Runarsson [9] created an auto-selection of combined DR by converting them into measurable contribution factors in the optimizing process of scheduling problems. Huang and Süer [10] adopted GA to explore the best combination of DR and used a "Hold" strategy for multi-objective JSS.

Evolutionary algorithms, such as genetic algorithm (GA), are an effective type of meta-heuristic methods. Zhang et al. [11] proposed a bi-level GA in an attempt to keep the advantages of preceding generations and reduce the disturbance of genetic operators. Later, an improved GA were proposed by Zhang et al. [12] targeting a better initialization and faster convergence. Huang et al. [13] developed an improved GA using opposition-based learning. The method used a multi-parent precedence operation crossover and a modified neighbor search mutation with opposite inverse mutation.

Swarm intelligence (SI) algorithms mainly include ant colony optimization (ACO), particle swarm optimization (PSO) algorithm, and artificial bee colony (ABC). Xu et al. [14] used bat algorithm to solve a dual flexible job shop problems (DFJSP). That algorithm used crossover and mutation as well as an adjusted value of the inertia weight with a linear decreasing strategy to enforce the search ability of the algorithm. Wu et al. [15] proposed a hybrid algorithm based on ACO while providing a modeling method based on 3D disjunctive graph.

Local search (LS) methods have been employed widely in solving the FJSP as well. The design of the neighborhood structure contributes directly to the efficiency of the method [1]. Sobeyko and Mönch [16] developed an iterative local search approach to deal with the objective of total weighted tardiness in large-scale FJSP. The algorithm used SA acceptance criterion to avoid getting trapped in local optimum.

Many researchers attempted to combine several algorithms to create some effective hybrid algorithms (HA) for FJSP. Palacios et al. [17] also combined GA with TS and added heuristic seeding. The hybrid algorithm was used to solve the fuzzy FJSP. Gaham et al. [18] presented an operations permutation-based discrete harmony search method. That method adopted an integration of the solution harmony with a dedicated improvisation operator. In addition to an integrated modified intelligent mutation operator.

In short, exact algorithm cost less CPU time, but most of these algorithms were not able to give competitive solution quality in comparison with other methods. On the other hand, metaheuristics, such as evolutionary algorithms (EA) and swarm intelligence based algorithms (SI) have given effective solutions and better quality. However, such metaheuristics algorithms cost much more computation time.

In this research, we propose a modified iterated greedy algorithm (MIG) to reduce the cost by consuming less CPU time. MIG provides a simple and easily applicable method that can compete with more complex metaheuristics. The proposed algorithm MIG consists of two phases, each phase is derived from the classical IG to solve the two sub-problems of FJSP. Both phases use a set of dispatching rules (DRs) to solve the FJSP.

2.2 Flexible Job Shop Problem Definition

For processing n jobs on m machines, the problem is to find the best solution that achieves the minimum or maximum value for an objective function. In the FJSP, there are a set of machines $A = M_1,\ldots, M_m$, and a set of jobs, $J = J_1,\ldots, J_n$ so that each job J_i consists of a given sequence of n_i operations, $O_{i,1}$, $O_{i,2},\ldots$, $O_{i,ni}$. Each operation $O_{i,j}$ can be processed on any machine of a subset $A_{i,j} \subseteq A$ which represents the routing sub-problem. The other sub-problem is the sequencing sub-problem which is to sequence the operations on the machines. In this paper, the objective function is to minimize the makespan (maximal completion time) of all jobs.

In this research the following assumptions are considered:

1) All machines are available at time 0;
2) All jobs are released at time 0;
3) Each machine can process only one operation at a time;
4) Each operation can be processed without interruption on one of a set of available machines;
5) Recirculation occurs when a job could visit a machine more than once;
6) The order of operations for each job is predefined and cannot be modified.

The FJSP has been classified by Kacem et al. [19] into partial flexible job shop (P-FJSP) and total flexible job shop (T-FJSP). The flexibility of problems is partial when there exists a subset $A_{i,j}$ of A ($A_{i,j} \subset A$) for at least one operation $O_{i,j}$, and it is total when $A_{i,j} = A$ for all operations. For the same number of machines and jobs, although the T-FJSP has the larger solution space, the P-FJSP is more difficult to solve than the T-FJSP [19].

3 Modified Iterated Greedy

3.1 Classical Iterated Greedy

This algorithm was first proposed by Ruiz and Stützle [20] to solve traditional permutation flow shop scheduling problems. The traditional IG consists of two distinct iterative phases; destructing some a part of the solution, and reconstructing this part by some greedy techniques including local search to improve the solution [20, 21]. The original IG has adopted NEH heuristics of Nawaz et al. [22] as its greedy constructive method.

Many works have been done later with IG; Ruiz and Stützle [23] used IG to solve FSP with sequence dependent setup times, and it's been used for node placement in street networks by Toyama et al. [24], and for single machine scheduling problems by Tasgetiren et al. [25], and as a local search method for unrelated parallel machine scheduling by Fanjul-Peyro and Ruiz [21].

The simple IG has proved to be effective and able to obtain state-of-the-art outcomes for a variety of JSP with different objectives [26].

3.2 Presented Algorithm (MIG)

The classical IG algorithm has been used in a wide range of scheduling problems according to Pan and Ruiz [26]. Although the algorithm was proposed to solve flow shop problems, some researchers used IG to deal with more complex problems like the hybrid flexible flow line problem as in Refs. [27, 28] and the blocking job shop scheduling problem by Pranzo and Pacciarelli [29].

Shop scheduling problems basically differs from each other by having different types or numbers of flexibility. By definition, operation flexibility is the possibility of performing an operation on more than a machine, sequencing flexibility is the possibility of interchanging the sequence in which required manufacturing operations are performed and processing flexibility is possibility of producing the same manufacturing feature with alternative operations or sequence of operations.

According to this definition, we can consider that the main difference between flexible job shop scheduling problem and job shop scheduling problem (JSP) is that flexible job shop problem dictates operation flexibility. The FJSP could be separated into two sub-problems: routing (assigning operations to machines) and scheduling (sequencing the assigned operations on each machine) [30]. Hence, the modified iterated greedy must deal with both of the sub-problems. Basically, that is achieved by separating both of destruct and construct phases in the algorithm into two stages in which both sides of FJSP are resolved. Pan and Ruiz [31] stated that IG dictates an effective greedy reconstruction in order to outperform other algorithms. The NEH heuristics gives no contribution on decision making of machine selection, and it also has limited solution variety while studying the FJSP. Therefore, NEH heuristics is replaced with a set of dispatching rules (DRs), to help with both decisions of sequencing and machine selection in reconstruction phase.

Figure 1 shows the flowchart of the MIG. The working of the algorithm is described below:

Step 1:	Initialization, which is done randomly
Step 2:	(Phase one) Destruct part of the solution machine selection
Step 3:	(Phase one) Reconstruct machine selection
Step 4:	(Phase two) Destruct sequence and machine selection for some operations
Step 5:	(Phase two) Reconstruct the sequence and the machine selection for these operations
Step 6:	Repeat the steps 2 through 5 until a stopping criterion is met.

3.3 Destruct Phase

The first stage here is to destruct part of the schedule. This is done by dissociating some consecutive operations from their machines, without changing the sequence of these selected operations. On the other hand, the second stage is to destruct part of the sequence and the schedule.

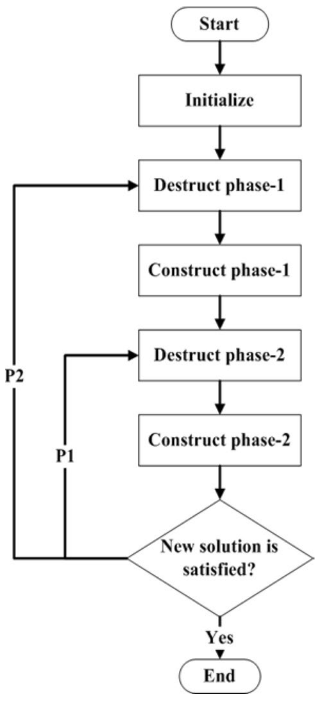


Figure 1 Flowchart of the MIG

This is done by removing some consecutive operations in from their sequence and dissociating them from their machines, later both sequence and machine selection will be reconstructed.

The destructed part is selected in one of either two ways:

1. The sequence is split into two parts and either one of these two part is destructed.
2. A number of consecutive operations are selected randomly along the sequence to be destructed.

3.4 Reconstruct Phase

In the first phase, reconstructing is to reassign the destructed operations from the existing sequence to machines using a set of dispatching rules as shown in Figure 2. This is done by comparing the quality of the sub-solution according to a randomly selected DR for all possible machines to process the operation. Then, the machine that achieves the sub-solution is selected.

Reconstructing in the second phase is to regenerate the destructed part of the sequence as in Figure 3. This is done by selecting two operations at a time. A DR is randomly selected and used to assess sequencing each of the two operations. The operation that results in a better sub-sequence is eventually selected.

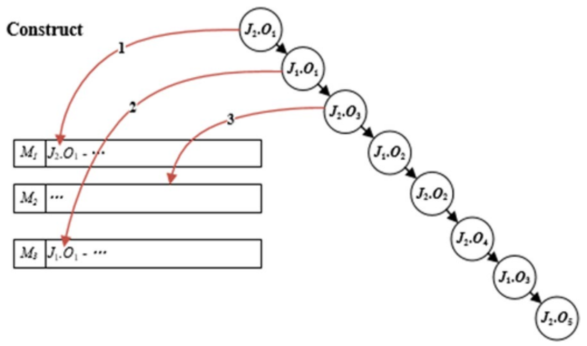


Figure 2 Construct phase–machine selection

3.5 Dispatching Rules

As a special case of priority rules, dispatching rules (DRs) are a simple scheduling heuristic, which gradually construct solutions by scheduling a single operation at a time [32, 33].

Due to their simplicity, sensitive nature, ease of use and the ability to fit a wide range of problem scale, DRs have been widely employed in solving scheduling problems [10, 33, 34]. DRs do not dictate high computational and information requirements [33]. Another important characteristic of DRs is their ability to adjust to dynamic changes [35]. These features encourage us to use DRs instead of NEH as the main heuristic to perform with IG for optimizing the FJSP.

It's been proved that a random assigning of the operations to the machines gives no better convergence. Besides that there isn't any dispatching rule that is alone capable to push forward the optimization process for any scheduling problem [36]. Thus, to make the selection procedure more intelligent, a set of dispatching rules (DRs) is used to assist machine selection procedure for each operation. Moreover, a favoring mechanism adopted by Ausaf et al. [37] is used to give more chance to more effective dispatching rules.

In general, researchers classified DRs into two main categories; online dispatching rules (dynamic DRs) and offline dispatching rules (static DRs) [38, 39].

3.5.1 Offline Dispatching Rules

According to these dispatching rules, each operation will have a fitness value based on some analysis on the problem's data. In this research, offline dispatching rules are used to optimize and construct the sequence of operation therefore the scheduling in FJSP. When two operations are selected, the fitness value is calculated according to an offline DP rule, and then the operation with the best value is placed in the sequence while these other

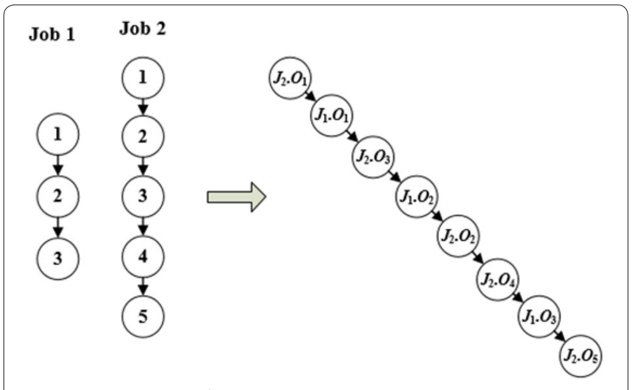


Figure 3 Construct phase–sequencing

operation is kept with ready operations. The set of ready operations is updated and another two operations are selected after that and the procedure is repeated till the construction phase of the sequence is completed. The fitness value of an offline DP rule is calculated directly from the problem's data before the optimization process. Hence, this fitness value remains constant for each pair of operation-DP rule. In this research, three offline dispatching rules are used as in Figure 4.

3.5.2 *Online Dispatching Rules*

According to these dispatching rules, each pair machine-operation will have a fitness value based on an analysis made on the current status of the machines in the sub-solution. This fitness value can be calculated only right before assigning this operation to a machine. Hence, for the same pair machine-operation, it varies along iterations due to the changes in the current sub-solution while constructing phase. In this research, 10 online dispatching rules are used (Figure 5).

The used dispatching rules can be categorized according to the use in this research as below:

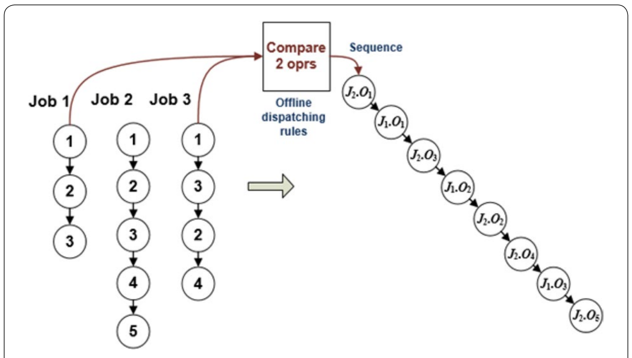


Figure 4 Using offline dispatching rules to build the sequence

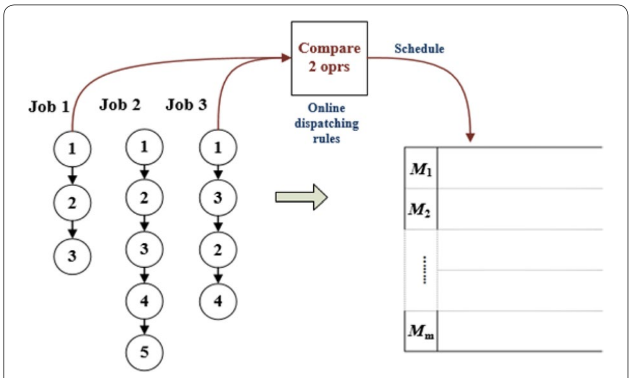


Figure 5 Using online dispatching rules for machine selection

a) Dispatching rules used to select a machine for an operation: where all the available machines are considered for each operation, and then only one machine is selected according to the following.

1. Shortest processing time (SPT), selects the machine that does the operation within the shortest processing time.
2. Earliest start (ES), selects the machine that will start with this operation earlier.
3. Earliest finish (EF), selects the machine that is able to finish the operation earlier.
4. Least utilized machine (LUM), selects the machine that currently has the minimum workload.
5. Minimum idle time (MIT), selects the machine that achieves the least idle time.
6. Earliest machine interval (EMI), selects the machine with the minimum current interval.
7. Minimum gap per job (MGJ), selects the machine on which the gap between the operation and its preceding (the time that the job will be waiting for the machine) one is the smallest.
8. Combined rules (CR), in combined dispatching rules two or more rules are performed together. The function value of the first rule is calculated for both operations. In case both operations have the same function value for this rule, another rule is used to choose the best operations.

b) Dispatching rules used to select an operation in sequencing: Two operations are selected and compared according to these rules, and then one of them is placed in the sequence. During this selection, the operations are already assigned to machines.

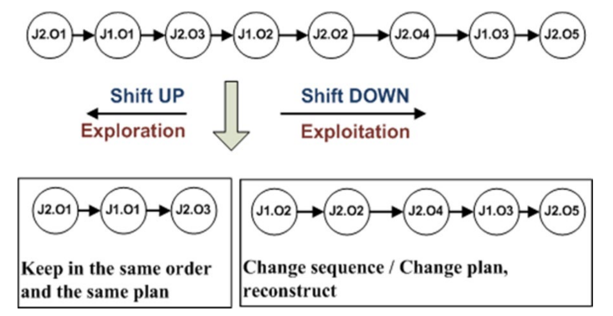
1. Shortest processing time (SPT), selects the operation with the shorter processing time.

2. Maximum processing time per job (MPJ), selects the operation which belongs to the job with the longer processing time.
3. Least utilized machine (LUM), selects the operation which is done by the machine with the greater workload.
4. Latest machine interval (LMI), selects the operation that is done by the machine with the greater processing time interval.
5. Combined rules (CR), two or more rules are combined as in (b).

The first four of these dispatching rules were adopted previously by Ausaf et al. [37]. Only four dispatching rules of the mentioned above are offline rules; SPT, MPJ, LUM (b) and LMI.

3.6 Techniques Used to Control the Algorithm

1. Adjusting the split point position
   A special function is used to adjust the split point. In case of exploitation, the split point is shifted to the right side, while in case of exploration; the split point is shifted to the left side. As is shown in Figure 6.
2. Adjusting the size of destructed part:
   Increasing the number of destructed operations will give the exploration that is needed in the algorithm, while decreasing this number will support exploitation process. The minimum and maximum sizes of the destructed part are determined initially for the instance. During the run, the algorithm starts with the minimum size, and increases gradually if the solution is not improving, when it reaches the maximum limit, the size decreases again for local search as in the previous Technique.


J2.01
J1.01
J2.03
J1.02
J2.02
J2.04
J1.03
J2.05
Shift UP
Exploration
Shift DOWN
Exploitation
Keep in the same order and the same plan
Change sequence / Change plan, reconstruct
Figure 6 Adjusting the split point position

4 Experiments and Discussion

The algorithm MIG has been coded in C++, and performed by a computer with 3.2 GHz processor and with 4.0 GB RAM memory. Three experiments of in total 35 benchmark problems are used to test the performance of the proposed algorithm. The objective considered in this paper is to minimize the makespan. The best solutions are shown in bold red font in each experiment.

4.1 Experiment 1

The data in this experiment includes 5 problems adopted from [40]. The results are compared in Table 1 with 6 other algorithms; HA, GATS, TABC, HHS, HDE-N2 and hGA proposed by Li and Gao [1], Nouri et al. [41], Gao et al. [42], Yuan et al. [43], Yuan and Xu [44], and Gao et al. [45] respectively. The proposed (MIG) obtained the global optimum for all instances.

Table 2 shows the CPU time compared with the same algorithms. It's clear that MIG consumed the lowest CPU time among other algorithms.

4.2 Experiment 2

The data includes 20 instances adopted from Fattahi et al. [46]. The first ten consecutive instances are considered as small-scale FJSP, while the remaining ten instances are categorized as medium- and large-scale FJSP. The results are compared in Table 3 with 6 algorithms; HA, EPSO, EM2, MILP and HHS proposed by Li and Gao [1], Teekeng et al. [47], Demir and İşleyen [6], Birgin et al. [48] and Yuan et al. [43] respectively. The data of AIA results are also taken from Yuan et al. [43].

The proposed algorithm (MIG) performed well on both small- and medium-scale instances. It obtains all optimum solutions for small-scale instances. For medium-scales problems, MIG outperforms HA, EPSO and HHS in two instances, and outperforms EM2 and MILP in

Table 1 Results of Kacem data (experiment 1)

$n \times m$	HA	GATS	TABC	HHS	HDE-N2	hGA	MIG
4×5	–	11	11	–	11	–	11
10×7	–	11	11	–	11	–	11
10×10	7	7	7	7	7	7	7
8×8	14	14	14	14	14	14	14
15×10	11	11	11	11	11	11	11

Table 2 CPU time comparison for instances in experiment 1

n × *m*	HA	GATS	TABC	HHS	HDE-N2	hGA	MIG
4 × 5	–	0.05	0.47	–	0.09	–	0.00
10 × 7	–	0.72	1.2	–	0.46	–	0.00
10 × 10	0.01	1.51	1.4	0.01	0.37	43.1	0.00
8 × 8	0.00	0.36	1.19	0.00	0.31	22.4	0.00
15 × 10	0.33	29.71	2.97	0.42	2.19	112.2	0.07

Table 3 Results of experiment 2

Ins.	HA	EPSO	EM2	MILP	AIA	HHS	MIG
SFJS01	66	66	66	66	66	66	66
SFJS02	107	107	107	107	107	107	107
SFJS03	221	221	221	221	221	221	221
SFJS04	355	355	355	355	355	355	355
SFJS05	119	119	119	119	119	119	119
SFJS06	320	320	320	320	320	320	320
SFJS07	397	397	397	397	397	397	397
SFJS08	253	253	253	253	253	253	253
SFJS09	210	210	210	210	210	210	210
SFJS10	516	516	516	516	516	516	516
MFJS01	468	468	468	468	468	468	462
MFJS02	446	446	446	446	448	446	446
MFJS03	466	466	466	466	468	466	450
MFJS04	554	554	564	564	554	554	554
MFJS05	514	514	514	514	527	514	514
MFJS06	634	634	634	634	635	634	634
MFJS07	879	879	928	879	879	879	881
MFJS08	884	884	–	–	884	884	889
MFJS09	1055	1059	–	–	1088	1055	1059
MFJS10	1196	1205	–	–	1267	1196	1214

3 instances. And finally, MIG outperforms AIA in 5 instances. In short, for medium-scale problems MIG outperforms all algorithms in literature in two instances (MFJS01 and MFJS03). The obtained solutions for large-scale instances were less competitive. For Large-scale problems, results for MIG are dominated by HA, EPSO and HHS. The Gantt charts for instances (MFJS01 and MFJS03) are shown in Figure 7 and Figure 8, respectively.

Table 4 shows that MIG consumed the least CPU time for all instances among the above mentioned algorithms.

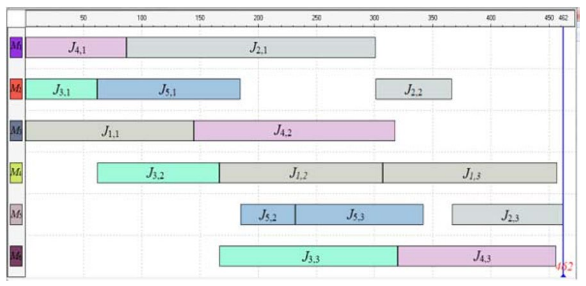


Figure 7 Gantt chart of instance MFJS01 (experiment 2)

4.3 Experiment 3

The data in this experiment is adopted from Brandimarte [49], it contains 10 instances with number of jobs ranges from 10 to 20, and the number of machines ranges from 6 to 15. We compared the results with 11 algorithms from literature; HA, HTGA, GATS, TABC, Heuristic, AMMA, HHS, hGA, TS, and IACO. These algorithm were proposed by Li and Gao [1], Chang et al. [50], Nouri et al. [41], Gao et al. [42], Ziaee [51], Zuo et al. [52] and Yuan et al. [43]. Results of AIA and TS are taken from Yuan et al. [43].

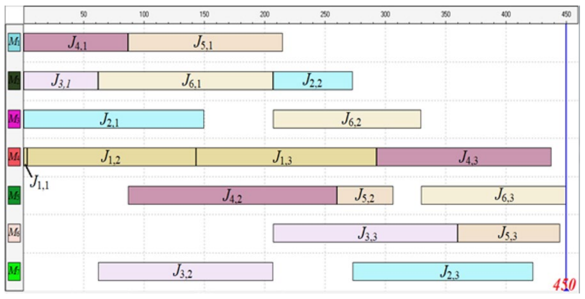


Figure 8 Gantt chart of instance MFJS03 (experiment 2)

We can see in Table 5 that the proposed MIG outperforms each of; Heuristic in 9 instances, GATS in 7 instances, HTGA and AIA in 3 instances, TS in 2 instances, and finally TABC in one instance. On the other hand, HA, AMMA, and HHS dominates MIG in 3 instances, while TS and TABC outperforms MIG in 2 instances as HTGA and AIA in one instance. Table 6 shows the CPU time for each instance compared with CPU consumed by the above mentioned algorithms. MIG consumed less CPU time than all other algorithms. Only heuristic method is competitive with MIG regarding the CPU time. Heuristic method consumed less CPU

Table 4 CPU time comparison for instances in experiment 2

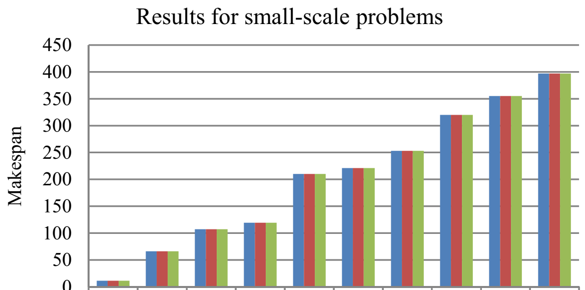
Ins.	$n \times m$	HA	EM2	MILP	AIA	HHS	MIG
SFJS01	2 × 2	0.00	0.03	0.00	0.03	0.00	0.00
SFJS02	2 × 2	0.00	0.10	0.01	0.03	0.00	0.00
SFJS03	2 × 2	0.00	0.05	0.05	0.04	0.00	0.00
SFJS04	3 × 2	0.00	0.04	0.02	0.04	0.00	0.00
SFJS05	3 × 2	0.00	0.06	0.04	0.04	0.00	0.00
SFJS06	3 × 3	0.00	0.28	0.01	0.04	0.00	0.00
SFJS07	3 × 5	0.00	0.03	0.00	0.04	0.00	0.00
SFJS08	3 × 4	0.00	0.16	0.04	0.05	0.00	0.00
SFJS09	3 × 3	0.00	1.26	0.01	0.05	0.00	0.00
SFJS10	4 × 5	0.00	0.06	0.02	0.05	0.00	0.00
MFJS01	5 × 6	0.00	0.78	0.26	9.23	0.01	0.00
MFJS02	5 × 7	0.00	49	0.87	9.35	0.01	0.00
MFJS03	6 × 7	0.02	191	1.66	10.06	0.12	0.002
MFJS04	7 × 7	0.02	1051	27.43	10.54	0.06	0.00
MFJS05	7 × 7	0.02	225	4.55	10.61	0.02	0.00
MFJS06	8 × 7	0.01	231	52.48	22.18	0.01	0.00
MFJS07	8 × 7	0.08	3600	1890	24.82	0.11	0.00
MFJS08	9 × 8	0.06	–	–	26.94	0.08	0.00
MFJS09	11 × 8	0.48	–	–	30.76	0.94	0.005
MFJS10	12 × 8	0.59	–	–	30.94	0.69	0.005

Table 5 Results of experiment 3

Ins.	$n \times m$	HA	HTGA	GATS	TABC	Heuristic	AMMA	AIA	HHS	TS	IACO	MIG
MK01	10 × 6	40	40	40	40	42	40	40	40	40	40	40
MK02	10 × 6	26	26	27	26	28	26	26	26	26	26	26
MK03	15 × 8	204	204	204	204	204	204	204	204	204	204	204
MK04	15 × 8	60	60	64	60	75	60	60	60	60	60	60
MK05	15 × 4	172	173	173	173	179	172	173	172	173	173	172
MK06	10 × 10	57	61	65	60	69	57	63	58	58	60	60
MK07	20 × 5	139	141	144	139	149	139	140	139	144	140	140
MK08	20 × 10	523	523	523	523	555	523	523	523	523	523	523
MK09	20 × 10	307	307	311	307	342	307	312	307	307	307	307
MK10	20 × 15	197	213	222	202	242	198	214	205	198	208	221

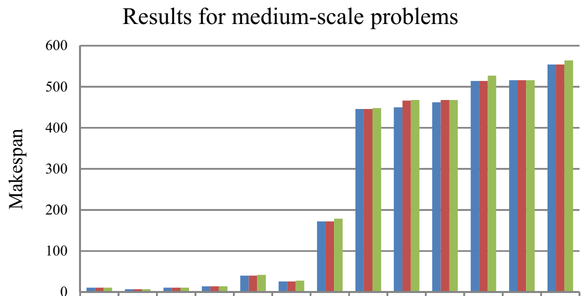
Table 6 CPU time comparison for instances in experiment 3

Ins.	*n* x *m*	HA	GATS	TABC	Heuristic	AIA	HHS	MIG
MK01	10 × 6	0.06	0.93	3.36	0.09	97.21	3.87	0.002
MK02	10 × 6	0.59	1.18	3.72	0.17	103.46	5.79	0.005
MK03	15 × 8	0.16	1.55	1.56	0.52	247.37	36.60	0.00
MK04	15 × 8	0.49	4.36	66.58	0.20	152.07	13.30	0.04
MK05	15 × 4	4.57	8.02	78.45	0.20	171.95	35.78	0.32
MK06	10 × 10	53.82	110.01	173.98	0.45	245.62	111.65	2.38
MK07	20 × 5	20.01	19.73	66.19	0.39	161.92	26.16	0.04
MK08	20 × 10	0.02	11.50	2.15	0.66	392.25	171.10	0.00
MK09	20 × 10	0.86	79.68	304.43	0.94	389.71	172.24	0.42
MK10	20 × 15	33.21	185.64	418.19	1.20	384.54	437.69	1.42



	4×5	2×2	2×2	3×2	3×3	2×2	3×4	3×3	3×2	3×5
🟦 MIG	11	66	107	119	210	221	253	320	355	397
🟥 Min.	11	66	107	119	210	221	253	320	355	397
🟩 Max.	11	66	107	119	210	221	253	320	355	397

Figure 9 Results comparison for small-scale problems



	10×7	10×10	15×10	8×8	10×6	10×6	15×4	5×7	6×7	5×6	7×7	4×5	7×7
🟦 MIG	11	7	11	14	40	26	172	446	450	462	514	516	554
🟥 Min.	11	7	11	14	40	26	172	446	466	468	514	516	554
🟩 Max.	11	7	11	14	42	28	179	448	468	468	527	516	564

Figure 10 Results comparison for medium-scale problems

time than MIG in only three instances; MK5, MK6, and MK10. On the other hand, MIG obtained better solutions for these instances than heuristic methods.

4.4 Analysis Summary

In this paper, a modified iterated greedy is proposed for solving the flexible job shop problem. In the experiments, 35 instances in total have been used from 3 different benchmarks to test MIG. We divide these instances into three categories; small-scale instances (10 instances), medium-scale instances (13 instances), and large-scales instances (12 instances).

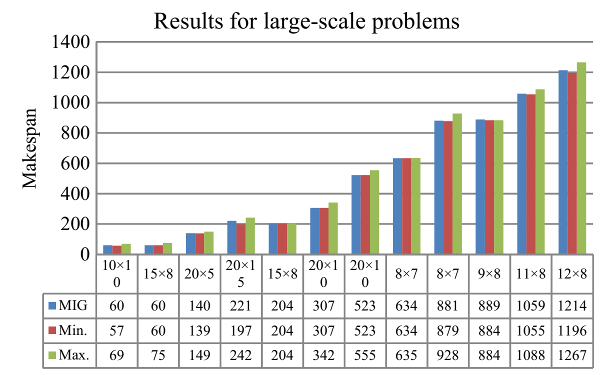
The results for small-scale instances prove that MIG is able to obtain global optimum for all instances as many previous algorithms actually did before. Figure 9 shows the results of MIG in comparison with minimum and maximum value of makespan that are obtained by other algorithms.

For medium-scale instances, MIG has obtained the best results for all instances and has outperformed all algorithms in literature for 2 instances (MFJS01 and MFJS03) of experiment 2. Figure 10 illustrates the performance of MIG in comparison with minimum and maximum makespan obtained by other algorithms.

While dealing with large-scale instances, MIG is able to obtain near-optimum solutions for many instances, but it has got trapped in local optimum for some others. Figure 11 shows the performance of MIG for 12 large-scales instances. It can be observed that the curve of MIG includes worse results for larger problems.

Experiments shows that MIG consumes less CPU time in comparison to all other algorithms included in this study. For only 3 instances of large-scale problems, only one algorithm (Heuristic) consumed less CPU time than MIG, but MIG on the other hand obtained much better solutions. This confirms that MIG is an effective method which costs the least CPU time.

The outstanding performance of MIG for small and medium-scale problems encourages us to make further development in the global search technique in future.



	10×10	15×8	20×5	20×15	15×8	20×10	20×10	8×7	8×7	9×8	11×8	12×8
🟦 MIG	60	60	140	221	204	307	523	634	881	889	1059	1214
🟥 Min.	57	60	139	197	204	307	523	634	879	884	1055	1196
🟩 Max.	69	75	149	242	204	342	555	635	928	884	1088	1267

Figure 11 Results comparison for large-scale problems

5 Conclusions and Future Work

In this paper, a modified iterated greedy is proposed for solving the flexible job shop scheduling problem. The experimental results show that the algorithm can find the global optimum solution for small and medium scale instances. MIG outperforms all other algorithms for 4 medium-scale instances. For large scale instances, the proposed algorithm has obtained optimum solutions for some instances and only near optimum solutions for some other instances. The main contribution of the proposed algorithm is to provide a simple and effective algorithm that can be easily employed in the real-life problems, and furthermore, it has insignificant CPU time cost in comparison with other metaheuristics that are widely used in this field.

In future, we will continue developing this algorithm to perform better on large-scale problems. The global search in the proposed algorithm will be developed for this purpose. Finally, multi-objective flexible job shop scheduling problem will be considered a good challenge for the developed version. Another option is to hybridize MIG with another algorithm.

Authors' Contributions

LG was in charge of the whole trial; Ghiath Al Aqel wrote the manuscript; XL assisted with the algorithm design. All authors read and approved the final manuscript.

Authors' Information

Ghiath Al Aqel, born in 1984, is currently a PhD candidate at *Huazhong University of Science and Technology, China*. He received his Master's degree on Industrial Engineering from *Huazhong University of Science and Technology, China*, in 2013.

Xinyu Li, born in 1985, is currently an associate professor at *State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, China*. He received his PhD degree on Industrial Engineering from *Huazhong University of Science and Technology, China*, in 2009.

Liang Gao, born in 1974, is currently a professor at *State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, China*. He received his PhD degree on Industrial Engineering from *Huazhong University of Science and Technology, China*, in 2002.

Competing interests

The authors declare that they have no competing interests.

Funding

Supported by National Natural Science Foundation of China (Grant Nos. 51825502, 51775216), Hubei Provincial Natural Science Foundation of China (Grant No. 2018CFA078), and Program for HUST Academic Frontier Youth Team.

Received: 22 November 2017 Accepted: 1 March 2019
Published online: 12 March 2019

References

[1] X Li, L Gao. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 2016, 174: 93-110.
[2] A Muthiah, A Rajkumar, R Rajkumar. Hybridization of artificial bee colony algorithm with particle swarm optimization algorithm for flexible job shop scheduling. *Energy Efficient Technologies for Sustainability (ICEETS), 2016 International Conference on*, 2016: 896-903.
[3] H Chen, J Ihlow, C Lehmann. A genetic algorithm for flexible job-shop scheduling. *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 1999: 1120-1125.
[4] P Brucker, R Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 1990, 45: 369-375.
[5] H E Nouri, O B Driss, K Ghédira. A classification schema for the job shop scheduling problem with transportation resources: State-of-the-art review. *Artificial Intelligence Perspectives in Intelligent Systems*, Springer, 2016: 1-11.
[6] Y Demir, S K İşleyen. Evaluation of mathematical models for flexible job-shop scheduling problems. *Applied Mathematical Modelling*, 2013, 37: 977-988.
[7] F Pezzella, G Morganti, G Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 2008, 35: 3202-3212.
[8] A Baykasoğlu, L Özbakır. Analyzing the effect of dispatching rules on the scheduling performance through grammar based flexible scheduling system. *International Journal of Production Economics*, 2010, 124: 369-381.
[9] H Ingimundardottir, T P Runarsson. Evolutionary learning of linear composite dispatching rules for scheduling. *Computational Intelligence*, Springer, 2016: 49-62.
[10] J Huang, G A Süer. A dispatching rule-based genetic algorithm for multi-objective job shop scheduling using fuzzy satisfaction levels. *Computers & Industrial Engineering*, 2015, 86: 29-42.
[11] C Zhang, Y Rao, P Li, et al. Bilevel genetic algorithm for the flexible job-shop scheduling problem. *Chinese Journal of Mechanical Engineering*, 2007, 19(4): 020.
[12] G Zhang, L Gao, P Li, et al. Improved genetic algorithm for the flexible job-shop scheduling problem. *Journal of Mechanical Engineering*, 2009, 45(7): 026 **(in Chinese)**.
[13] M Huang, W Mingxu, L Xu. An improved genetic algorithm using opposition-based learning for flexible job-shop scheduling problem. *Cloud Computing and Internet of Things (CCIOT), 2016 2nd International Conference on*, 2016: 8-15.
[14] H Xu, Z Bao, T Zhang. Solving dual flexible job-shop scheduling problem using a Bat Algorithm. *Advances in Production Engineering & Management*, 2017, 12: 5.
[15] J Wu, G Wu, J Wang. Flexible job-shop scheduling problem based on hybrid ACO algorithm. *International Journal of Simulation Modelling (IJSIMM)*, 2017, 16(3): 497-505.
[16] O Sobeyko, L Mönch. Heuristic approaches for scheduling jobs in large-scale flexible job shops. *Computers & Operations Research*, 2016, 68: 97-1096.
[17] J J Palacios, M A González, C R Vela, et al. Genetic tabu search for the fuzzy flexible job shop problem. *Computers & Operations Research*, 2015, 54: 74-89.

[18] M Gaham, B Bouzouia, N Achour. An effective operations permutation-based discrete harmony search approach for the flexible job shop scheduling problem with makespan criterion. *Applied Intelligence*, 2017: 1-19.
[19] I Kacem, S Hammadi, P Borne. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, 2002, 60: 245-276.
[20] R Ruiz, T Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 2007, 177: 2033-2049.
[21] L Fanjul-Peyro, R Ruiz. Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 2010, 207: 55-69.
[22] M Nawaz, E E Enscore, I Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 1983, 11: 91-95.
[23] R Ruiz, T Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 2008, 187: 1143-1159.
[24] F Toyama, K Shoji, J Miyamichi. An iterated greedy algorithm for the node placement problem in bidirectional manhattan street networks. *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, 2008: 579-584.
[25] M F Tasgetiren, Q K Pan, Y C Liang. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Computers & Operations Research*, 2009, 36: 1900-1915.
[26] Q K Pan, R Ruiz. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega*, 2014, 44: 41-50.
[27] T Urlings, R Ruiz. A new algorithm for multidimensional scheduling problems. *The 9th Workshop on Models and Algorithms for Planning and Scheduling Problems*, 2009: 20.
[28] T Urlings, R Ruiz, T Stützle. Shifting representation search for hybrid flexible flowline problems. *European Journal of Operational Research*, 2010, 207: 1086-1095.
[29] M Pranzo, D Pacciarelli. An iterated greedy metaheuristic for the blocking job shop scheduling problem. *Journal of Heuristics*, 2016, 22: 587-611.
[30] G Zhang, L Gao, X Li, et al. Variable neighborhood genetic algorithm for the flexible job shop scheduling problems. *International Conference on Intelligent Robotics and Applications*, 2008: 503-512.
[31] Q K Pan, R Ruiz. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 2012, 222: 31-43.
[32] R Haupt. A survey of priority rule-based scheduling. *OR Spectrum*, 1989, 11: 3-16.
[33] J Branke, S Nguyen, C W Pickardt, et al. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 2016, 20: 110-124.
[34] Y Mei, M Zhang. A comprehensive analysis on reusability of GP-evolved job shop dispatching rules. *Evolutionary Computation (CEC), 2016 IEEE Congress on*, 2016: 3590-3597.
[35] S Nguyen, M Zhang. A PSO-based hyper-heuristic for evolving dispatching rules in job shop scheduling. *Evolutionary Computation (CEC), 2017 IEEE Congress on*, 2017: 882-889.
[36] T C Chiang, L C Fu. Using dispatching rules for job shop scheduling with due date-based objectives. *International Journal of Production Research*, 2007, 45: 3245-3262.
[37] M F Ausaf, L Gao, X Li, et al. A priority-based heuristic algorithm (PBHA) for optimizing integrated process planning and scheduling problem. *Cogent Engineering*, 2015, 2: 1070494.
[38] H L Fan, H G Xiong, G Z Jiang, et al. Survey of the selection and evaluation for dispatching rules in dynamic job shop scheduling problem. *Chinese Automation Congress (CAC) 2015*, 2015: 1926-1931.
[39] K C Ying, S W Lin, C C Lu. Effective dynamic dispatching rule and constructive heuristic for solving single-machine scheduling problems with a common due window. *International Journal of Production Research*, 2017, 55: 1707-1719.
[40] I Kacem, S Hammadi, P Borne. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 2002, 32: 1-13.
[41] H E Nouri, O B Driss, K Ghédira. Genetic algorithm combined with Tabu search in a holonic multiagent model for flexible job shop scheduling problem. *ICEIS (1)*, 2015: 573-584.
[42] K Z Gao, P N Suganthan, T J Chua, et al. A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Systems with Applications*, 2015, 42: 7652-7663.
[43] Y Yuan, H Xu, J Yang. A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Applied Soft Computing*, 2013, 13: 3259-3272.
[44] Y Yuan, H Xu. Flexible job shop scheduling using hybrid differential evolution algorithms. *Computers & Industrial Engineering*, 2013, 65: 246-260.
[45] J Gao, L Sun, M Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 2008, 35: 2892-2907.
[46] P Fattahi, M S Mehrabad, F Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 2007, 18: 331.
[47] W Teekeng, A Thammano, P Unkaw, et al. A new algorithm for flexible job-shop scheduling problem based on particle swarm optimization. *Artificial Life and Robotics*, 2016, 21: 18-23.
[48] E G Birgin, P Feofiloff, C G Fernandes, et al. A MILP model for an extended version of the Flexible Job Shop Problem. *Optimization Letters*, 2014, 8: 1417-1431.
[49] P Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 1993, 41: 157-183.
[50] H C Chang, Y P Chen, T K Liu, et al. Solving the flexible job shop scheduling problem with makespan optimization by using a hybrid Taguchi-genetic algorithm. *IEEE Access*, 2015, 3: 1740-1754.
[51] M Ziaee. A heuristic algorithm for the distributed and flexible job-shop scheduling problem. *The Journal of Supercomputing*, 2014, 67: 69-83.
[52] Y Zuo, M Gong, L Jiao. Adaptive multimeme algorithm for flexible job shop scheduling problem. *Natural Computing*, 2016: 1-22.