Computers & Operations Research 47 (2014) 11–26

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Path-relinking Tabu search for the multi-objective flexible job shop scheduling problem

Shuai Jia [a], Zhi-Hua Hu [a,b,*]

[a] Logistics Research Center, Shanghai Maritime University, Shanghai 200135, China
[b] School of Economics and Management, Tongji University, Shanghai 200092, China

ARTICLE INFO

Available online 7 February 2014

Keywords:
Flexible job shop scheduling
Multi-objective optimization
Tabu search
Path-relinking

ABSTRACT

The multi-objective flexible job shop scheduling problem is solved using a novel path-relinking algorithm based on the state-of-the-art Tabu search algorithm with back-jump tracking. A routing solution is identified by problem-specific neighborhood search, and is then further refined by the Tabu search algorithm with back-jump tracking for a sequencing decision. The resultant solution is used to maintain the medium-term memory where the best solutions are stored. A path-relinking heuristics is designed to generate diverse solutions in the most promising areas. An improved version of the algorithm is then developed by incorporating an effective dimension-oriented intensification search to find solutions that are located near extreme solutions. The proposed algorithms are tested on benchmark instances and its experimental performance is compared with that of algorithms in the literature. Comparison results show that the proposed algorithms are competitive in terms of its computation performance and solution quality.

1. Introduction

Job shop scheduling problem (JSP) is an NP-hard [1] combinatorial optimization problem. In a JSP, a set of jobs with several consecutive operations must be processed on a set of machines. Each operation should be processed by a particular machine, and each machine can only handle one operation type. The problem is to schedule operations on each machine such that a given criterion is met (e.g., minimization of makespan). The JSP has garnered tremendous attention and many algorithms have been developed [2]. As a generalization of JSP, the flexible job shop scheduling problem (FJSP) is more realistic than the JSP. The FJSP allows operations for each job to be processed on one machine out of a set of capable machines. Thus, the FJSP is more difficult to solve than the JSP because one should consider routing jobs in addition to sequencing jobs. Despite its combinatorial complexity, the FJSP is suited to practical job shops, because most machines can perform more than one task type. Moreover, in the FJSP, jobs can be transferred to other machines when a machine breaks down, thereby avoiding blocking and production interruptions [3].

Criteria such as minimization of makespan and weighted tardiness [4], and minimization of machine workload [5] and total workload are typically applied to the FJSP. A solution to the multi-objective FJSP (MOFJSP) should meet different criteria simultaneously. For a multi-objective optimization problem, the first solution is to aggregate objectives through a linear function by assigning a weight to each objective. Another approach deals with objectives via a Pareto concept (i.e., find a set of optimal solutions through a single run in which solutions are Pareto-equivalent). Apparently, Pareto-based methods make trade-offs between conflicting objectives, as well as options from which decision-makers can choose. Conversely, weighted sum-based methods merely search for a single solution that meets aggregated objectives, leaving practical implications unclear.

This work applied two versions of novel path-relinking (PR) Tabu search (TS) algorithms to solve the MOFJSP. The work contributes to the literature on the FJSP, TS, and multi-objective optimization. First, a PR combined with TS is designed for the MOFJSP; an effective dimension-oriented intensification search (IS) mechanism is developed to improve the TS algorithm; and the TS algorithm with back-jump tracking (TSAB) [6] is extended for multi-objective optimization problems.

The remainder of this paper is organized as follows. In Section 2, related studies are reviewed. In Section 3, the definition and formulation of MOFJSP are presented. Detailed frameworks of the

* Corresponding author at: Logistics Research Center, Shanghai Maritime University, Shanghai 200135, China. Tel.: +86 21 38284016; fax: 86 21 58855200.
E-mail address: zhhu@shmtu.edu.cn (Z.-H. Hu).

http://dx.doi.org/10.1016/j.cor.2014.01.010
0305-0548

proposed algorithms are given in Section 4. Empirical studies and results are presented in Section 5. Finally, conclusions and directions for future study are given in Section 6.

2. Literature review

Brucker and Schlie [7] solved an FJSP with two jobs using a polynomial algorithm. Although their algorithm obtained theoretically optimal solutions, it cannot be applied to solve large problems. Considering the complexity of the FJSP and the weak performance of exact algorithms, decomposition approaches deconstruct a problem into sub-problems and treat each sub-problem separately. Brandimarte [4] decomposed the FJSP into routing and job shop scheduling sub-problems. A proposed hierarchical TS algorithm was then applied to minimize makespan and total weighted tardiness. The TS algorithm benefits from the information flows that were established between routing and sequencing phases, accelerating convergence and improving information exploitation. Unfortunately, as one shortcoming of the algorithm, it easily gets stuck into local optima. Hurink et al. [8] and Mastrolilli and Gambardella [9] solved the FJSP using a TS algorithm. Compared with Brandimarte's TS, their algorithms were straightforward implementations of TS, and high-quality solutions were acquired with the assistance of effective neighborhood generation methods.

In the last decade, evolutionary algorithms (EAs) have garnered tremendous attention and have been adopted widely for the FJSP. Moreover, multi-objective optimization of the FJSP has become increasingly interesting. Kacem et al. [5,10] extended a single-objective FJSP to an MOFJSP by simultaneously minimizing makespan, total workload of machines and the workload of the most-loaded machines. They proposed a localization method to generate initial solutions. A hybrid fuzzy logic algorithm combined with an EA was developed to solve the MOFJSP. Although delicately designed, their algorithm cannot find best solutions for even small instances due to a lack of efficient chromosome encoding scheme and Pareto evaluation mechanism. The localization heuristics then became very popular for routing decisions. Ho and Tay [11] proposed an efficient methodology, which used a culture algorithm to guide the evolution of populations. The information of elite chromosomes was stored in a belief space of each generation and it was used to direct the selection and mutation operators for the subsequent generation. Pezzella et al. [12] proposed a GA for the MOFJSP. In their study, chromosomes were encoded in a 3-tuple form, and initial solutions were generated in a manner that resembled Kacem's localization heuristics. In addition to the 3-tuple representation, chromosomes were also encoded in two-level schemes [13–16]. The two-level scheme considered machine assignment and operation sequencing. Such an encoding method makes it possible to deal with routing and sequencing separately. Zhang et al. and Moslehi and Mahnam [17,18] applied particle swarm optimization (PSO) algorithms to solve the MOFJSP. Zhang et al. [17] aggregated various objectives into a weighted sum, while Moslehi and Mahnam [18] treated objectives in a Pareto manner. According to the computational results, the PSO algorithms can only solve small instances. Bagheri et al. [19] proposed an artificial immune algorithm for the FJSP to minimize makespan. In their algorithm, antibodies were encoded using Pezzella's 3-tuple scheme, and two mutation operators (precedence preserving shift mutation, and reordering mutation) were developed. Xia and Wu [20] developed a hybrid PSO and simulated annealing (SA) algorithm. The PSO portion of the algorithm was used to make routing decisions, while the SA portion of the algorithm was used to schedule operations on each machine. Zhang et al. [17] hybridized PSO and TS for the MOFJSP, where the PSO makes routing and sequencing decisions simultaneously, and the TS refines the sequencing decision by using neighborhood structures [9]. Except for algorithms already mentioned, other approaches adopted for the FJSP include simulation modeling [21], discrepancy search [22], shuffled frog-leaping [23,24], harmony search algorithm [25], and some parallelization techniques [26,27]. While population-based EAs are popular for solving the FJSP, TS has received relatively little attention in recent years despite its suitability and the fact that it is easily implemented for real-world problems. For Pareto optimization of the MOFJSP, Ho and Tay [28] proposed a GA with a local search mechanism. They adopted the state-of-the-art NSGA-II [29] to evaluate individuals. Additionally, a branch-and-bound algorithm was introduced to find the lower bounds to assess whether the solution by the GA were optimal. Wang et al. [30] proposed an immune algorithm, in which the fitness evaluation scheme is based on the well-known SPEA [31]. They also applied an entropy principle to maintain the diversity of individuals in order to avoid becoming trapped in local optima. Although efforts had been made, their algorithms found only small portions of the optimal solutions. The reason lies in the fact that the NSGA-II favors solutions that are located at the center of search space, while the SPEA prefers those close to extreme solutions. Li et al. [32] developed an artificial bee colony (ABC) algorithm and used the NSGA-II for individual evaluation. Moslehi and Mahnam [18] employed a hybrid PSO algorithm for Pareto

Table 1
Recent studies addressing the FJSP or MOFJSP.

Class	EA	TS	Other approaches
Makespan minimization	Ho and Tay [11] Ho et al. [13] Pezzella et al. [12] Bagheri et al. [19] Zhang et al. [15] Wang et al. [16] Yuan et al. [25]	Brandimarte [4]	Hmida et al. [35] Bożejko et al. [27] Yazdani et al. [26]
Linear weighted sum	Kacem et al. [5] Gao et al. [14] Zhang et al. [17]	Li et al. [36]	Xia and Wu [20] Xing et al. [21]
Pareto optimization	Kacem et al. [10] Ho and Tay [28] Wang et al. [30] Moslehi and Mahnam [18] Li et al. [32] Li et al. [23] Wang et al. [33] Chiang and Lin [3]		

optimization of the MOFJSP. Wang et al. [33] proposed an ABC algorithm for the MOFJSP; their algorithm was improved by an exploitation search heuristics for machine assignment and operation sequencing. Chiang and Lin [3] presented a simple EA with only two parameters, and the NSGA-II was adopted to evaluate the quality of chromosomes. To maintain the diversity of populations while utilizing the information of duplicate individuals, a mechanism that balances exploration and exploitation was developed. This mechanism was fulfilled by five effective reassignment operators.

According to the literature review, EAs are the most used methods for solving the FJSP and MOFJSP. However, complex encoding schemes as well as operators make EAs rigid in applications, especially for practical problems. Table 1 provides an overview of germane studies, which are classified according to whether multi-objective optimization is involved and how the objectives are treated. Furthermore, studies are categorized by their methodologies. As can be observed, TS received little attention, whereas EAs were preferred by researchers for solving the FJSP and MOFJSP. In particular, EA is the only methodology chosen for Pareto optimization of the MOFJSP. However, with respect to local search components, flexibility in controlling a search, and its easy-to-implement character, TS is particularly attractive for highly complex problems. Moreover, Pareto optimization by TS is rarely discussed according to [34]. Hence, Pareto optimization by TS is a new and interesting approach for the MOFJSP, and definitely deserves attention.

3. Problem statement

A typical FJSP involves a set of jobs, $J=\{1,2,\ldots,n\}$, and a set of machines, $M=\{1,2,\ldots,m\}$. A job $j\in J$ consists of a sequence of operations $O^j=(o_{j1},o_{j2},\ldots,o_{ji})$, which must be processed consecutively in a fixed order. For each operation o, a set of available machines $M^o(M^o\subset M)$ is given. For each machine $k\in M^o$ that can perform o, the processing time, τ_{ok}, is given. The MOFJSP consists of the allocation of operations for each job to machines in capable machine sets, as well as scheduling operations on each machine with fixed processing times to meet the following criteria: minimize overall finishing time (makespan); minimize total workload of all machines (in terms of processing time); and minimize the workload of the most-loaded machine.

Let $K=(K_1,K_2,\ldots,K_m)$ be a routing solution and $\pi(K)=(\pi_1(K),\pi_2(K),\ldots,\pi_m(K))$ be a sequencing solution for K, where $K_i(i\in M)$ is the operation assignment for machine i, and $\pi_i(K)$ is the order of operations on machine i. The solutions to the FJSP can then be represented by Eq. (1), where Φ represents all possible assignments and $\Pi(K)$ is all feasible sequencing solutions for a given K

$$\Omega=\{(K,\pi(K))|K\in\Phi,\pi(K)\in\Pi(K)\} \tag{1}$$

Nowicki and Smutnicki [6] established models for the JSP based on a directed graph denoted by $G(\psi)=(N,R\cup E(\psi))$, where $\psi=(K,\pi(K))(\psi\in\Omega)$ is a solution, N is a set of nodes representing operations, and $R\cup E(\psi)$ is a set of arcs, which are defined by Eqs. (2) and (3), that link nodes. Here, this work uses the graph to model FJSP as Eqs. (2) and (3), where arcs from $E(\psi)$ are the processing order of operations on machines, $\pi_k(u)$ is the uth operation processed on machine k, and O_k is the set of operations that is assigned to machine k

$$R=\ \cup_{j=1}^{n}\cup_{i=1}^{o_j-1}\{(o_{ji},o_{j(i+1)})\}(o_j=|O^j|) \tag{2}$$

$$E(\psi)=\ \cup_{k=1}^{m}\cup_{u=1}^{\varepsilon-1}\{(\pi_k(u),\pi_k(u+1))\}(\varepsilon=|O_k|) \tag{3}$$

One can observe that R is the processing order of operations for the same jobs, while $E(\psi)$ is the processing order of operations that are assigned to the same machines. Fig. 1 shows a feasible schedule and its corresponding directed graph. In the graph, conjunctive arcs represent the priorities of operations for the same job, while disjunctive arcs represent the sequence of operations on the same machine. A schedule is feasible only when the corresponding graph does not contain a cycle [6].

Each node $i\in N$ has at most two immediate predecessors and successors (shown as nodes connected by conjunctive and disjunctive arcs in the directed graph). Let $p^1(i)$ be the node (operation) that is in the same job as i and precedes i according to the priority constraint, and let $p^2(i)$ be the node that is on the same machine as i and precedes i in terms of processing time. The start time of node i can then be calculated by $t_i^s=\max(C_{p^1(i)},C_{p^2(i)})$, where $C_i=w_i+t_i^s(i>1)$ is the finish time for node i, w_i is the weight (processing time) of i. Denote the length of a path in $G(\psi)$ by $C_p(u,v)$, where u is the starting node, and v is the terminal node of path p. Therefore, $C_p(u,v)=C_v-C_u$. A critical path is the longest path in $G(\psi)$, and its length is $C_{\max}(\psi)$. The workload of each machine $W_k(\psi)(k\in M)$ can be derived by summing the weights of all nodes that are connected by the disjunctive arcs that represent k. Total workload, $W_T(\psi)$, is then the sum of $W_k(\psi)$. Given these notations, the goal in this work is to find a solution $\psi=(K,\pi(K))$ to the MOFJSP, such that $C_{\max}(\psi)$, $W_{\max}(\psi)=\max\{k\in M|W_k(\psi)\}$, and $W_T(\psi)$ are minimized.

Based on these definitions, the MOFJSP is formulated by Eqs. (4)–(11), as indicated by [17]. In the formulations, $\mu_{o_{ji},k}=1$ if operation o_{ji} is assigned to k; otherwise, zero

$$\text{Minimize}\quad f_1=C_{\max}=\max\{C_{o_{ji}}|j\in J,o_{ji}\in O^j\} \tag{4}$$

$$f_2=W_T=\sum_{k\in M}W_k \tag{5}$$

$$f_3=W_{\max}=\max\{W_k|k\in M\} \tag{6}$$

$$\text{s.t.}\quad W_k=\sum_{j\in J}\sum_{o_{ji}\in O^j}\tau_{o_{ji},k}\cdot\mu_{o_{ji},k},\quad k\in M \tag{7}$$

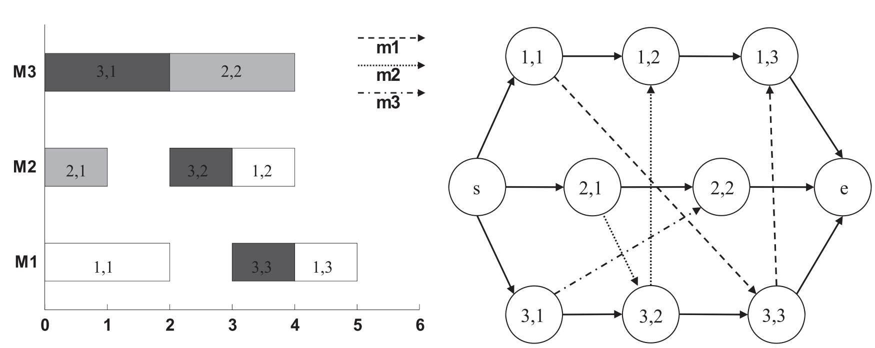


Fig. 1. A feasible schedule and its corresponding directed graph.

$$\sum_{k\in M^{o_{ji}}} \mu_{o_{ji},k}\cdot(C_{o_{ji}}-\tau_{o_{ji},k}) - \sum_{k\in M^{o_{j,i-1}}} \mu_{o_{j,i-1},k}\cdot C_{o_{j,i-1}} \ge 0 \quad \forall j\in J, \quad o_{ji}, o_{j,i-1}\in O^j, \quad i>1 \tag{8}$$

$$\begin{pmatrix} ((C_{o_{ji}} - C_{o_{pq}} - \tau_{o_{ji},k})\cdot\mu_{o_{ji},k}\cdot\mu_{o_{pq},k}\ge 0)\vee \\ ((C_{o_{pq}} - C_{o_{ji}} - \tau_{o_{pq},k})\cdot\mu_{o_{ji},k}\cdot\mu_{o_{pq},k}\ge 0) \end{pmatrix} \quad \forall j,\ p\in J,\ o_{ji}\in O^j,\ o_{pq}\in O^p,\ k\in M^{o_{ji}}\cap M^{o_{pq}} \tag{9}$$

$$\sum_{k\in M^{o_{ji}}} \mu_{o_{ji},k} = 1, \quad j\in J, \quad o_{ji}\in O^j \tag{10}$$

$$\mu_{o_{ji},k}\in\{0,1\}, \quad j\in J, \quad o_{ji}\in O^j, \quad k\in M \tag{11}$$

4. Multi-objective Tabu search with path-relinking

The TS, a local search heuristics developed and formalized for combinatorial optimization problems, takes advantage of memory structures to control the search process, which is guided by tailored neighborhood structures. Jaeggi et al. [34] proposed a multi-objective TS (MOTS) algorithm by incorporating a medium-term memory (MTM), which stores and updates optimal solutions found during the search process. Additionally, an intensification memory intensifies the search process. Although their algorithms were designed for continuous optimization problems, the algorithms implement many interesting ideas for handling multi-objective combinatorial optimization problems.

Due to the lack of mechanisms that exploit the information of good solutions, the performance of a straightforward TS might be unsatisfactory, even with delicate memory structures and good neighborhood structures. Thus, some auxiliary heuristics were incorporated into TS. This work develops a novel PR multi-objective TS (PRMOTS) for solving the MOFJSP. This algorithm begins with an initial solution generated by routing approaches and classic dispatching rules [5,15]. A variation of the well-known TSAB [6] is then applied to locate a set of local optimal solutions. The search is then diversified and intensified by problem-oriented PR and IS methods. An unbounded MTM is introduced to archive the optimal solutions found.

4.1. Solution representation

Solutions are encoded by m strings, where m is the number of machines. Each string contains operations processed on a machine and their sequences (Fig. 2(a)). Given an operation o, its start time t_o^s can be obtained by $t_o^s = \max(C_{p^1(o)}, C_{p^2(o)})$. If the end time of each operation p in $\{p^1(o), p^2(o)\}$ is unknown, t_p^s is therefore calculated using the end times of operations that precede p. Based on the recursive decoding mechanism, the strings can be decoded into a compressed schedule (Fig. 2(b)). Specially, if the current solution is infeasible, an infinite loop will be detected during decoding, indicating that the directed graph of the resultant schedule contains a cycle. Algorithm 1 presents the pseudo code of the decoding procedure. To finish the recursive decoding, three buffer sets P, K and Q are utilized, where P contains operations whose start and end times are unknown but necessary in order to get the start and end times of o. Q stores operations whose start and end times are known and can be used to calculate start times of operations in P. The time calculation is fulfilled by a function entitled as *Calculate Time*(P, Q) that uses the formulation mentioned above. K is used to determine which operation is to be included by P and Q. In lines 11 and 12, a cycle is detected if o or any operation stored in P becomes the predecessor of any operation o' selected from K. Thereby, o' is marked as an active operation. The active operation is then used to break the cycle by a check-and-repair mechanism, which is described in detail in Section 4.6. In the following sections, this work presents some novel move operators for the neighborhood search. Those moves can be performed by simply adjusting the positions of the corresponding operations in the strings.

Algorithm 1. Decoding.

Input: Coding strings S for a solution
Output: A compressed schedule
1: $O \leftarrow$ all operations in S
2: **While** O is not empty
3: $P\leftarrow\varnothing$, $K\leftarrow\varnothing$, $Q\leftarrow\varnothing$
4: $o\leftarrow$ Randomly select an operation in O, $K\leftarrow K\cup\{o\}$
5: **While** K is not empty
6: $o'\leftarrow$ Randomly select an operation in K
7: **If** $C_{o'}$ or $t_{o'}^s$ is known
8: $K\leftarrow K\setminus\{o'\}$, $O\leftarrow O\setminus\{o'\}$, $Q\leftarrow Q\cup\{o'\}$
9: **Else if** $C_{p^1(o')}$ and $C_{p^2(o')}$ are all known
10: $t_{o'}^s = \max(C_{p^1(o')}, C_{p^2(o')})$, $K\leftarrow K\setminus\{o'\}$, $O\leftarrow O\setminus\{o', p^1(o'), p^2(o')\}$, $Q\leftarrow Q\cup\{o'\}$
11: **Else if** $p=o$ or $p=p'$ (for any $p\in\{p^1(o'), p^2(o')\}$ and $p'\in P$)
12: A cycle is detected, mark o' as active operation, terminate decoding
13: **Else**
14: $K\leftarrow K\cup\{p^1(o'), p^2(o')\}$, $K\leftarrow K\setminus\{o'\}$, $P\leftarrow P\cup\{o'\}$
15: **End If**
16: **End While**
17: *Calculate Time* (P, Q), $O\leftarrow O\setminus P$
18: **End While**

4.2. Neighborhood structures

Two moves are used: *interchange moves* focus on the permutation of the positions of two adjacent operations, while *transfer moves* transform solution s into a new solution s' by transferring an operation from its current position to a new position on another machine.

Three neighborhood structures in the PRMOTS are as follows:

1. N_1: randomly pick an operation on a machine, move this operation from its current position to a random position on

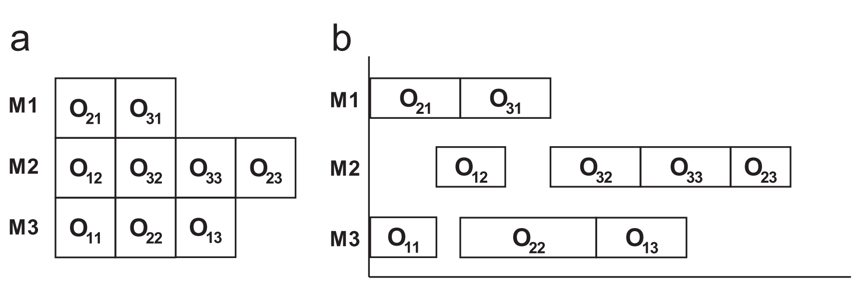


Fig. 2. Solution representation. (a) an encoded solution and (b) corresponding schedule.

another machine, such that the position does not violate the precedence constraint for the corresponding job on that machine.

2. N_2: randomly pick an operation that belongs to a critical path on a machine, move that operation from its current position to a random position that does not belong to any critical path on another machine, such that the position does not violate the precedence constraint for the corresponding job on that machine.
3. N_3: exchange two adjacent operations that are located at the beginning or end of a critical block [6].

4.3. Initial solutions

In a typical TS, initial solutions are enumerated from long-term memory, which is the entire search space. However, from a combinational optimization perspective, randomly generated solutions frequently are of poor quality and thus are too costly or impossible to refine to an optima. To be efficient in finding good solutions in reasonable computational times, a method should focus on the most promising areas and explore the search space intelligently.

The PRMOTS has four routing strategies:

1. *Random assignment*: Assign operations to capable machines randomly.
2. *Global minimum time*: Select an operation with the shortest processing time on the timetable, and assign it to the corresponding machine [5].
3. *Global permutation*: Randomly permute the order of jobs and order of machines and assign operations according to the orders on the timetable [5].
4. *Local permutation*: Permute the order of operations within the same jobs instead of permuting the order of jobs [15].

The PRMOTS also has four sequencing strategies:

1. *Random dispatching*: Operations are scheduled randomly on the machine to which they are assigned.
2. *Shortest processing time* (*SPT*): Operations are scheduled according to the duration of their processing time (i.e., operations with short processing times have high priorities for processing).
3. *Longest processing time* (*LPT*): Operations with long processing times have high priorities for processing.
4. *Most operations remained* (*MOR*): Operations are scheduled according to the number of remaining operations that belong to the same jobs, and when a job has many operations remaining, its operations have a high priority for processing.

EAs usually determine the proportions of different strategy combinations to construct initial populations [12,14]. Initial solutions by the PRMOTS are generated by randomly selecting strategic populations with equal probability.

4.4. TSAB^2 – the fundamental local search algorithm

4.4.1. Algorithm structure and neighborhood search strategy

Based on the decomposition concept, a hierarchical TS implementation is appropriate for the MOFSJP. In the upper level of the algorithm, the routing problem is solved, while in the lower level, the sequencing problem is optimized. The neighborhood search in the PRMOTS is executed hierarchically. Each time an initial solution is generated (by any strategy described in Section 4), it is first refined in the upper level. In this procedure, the algorithm first selects a neighborhood structure from N_1 and N_2. A transfer move v_t is performed based on the selected neighborhood structure. The function *Execute Move* (s_i, v_t) generates a neighbor solution s_h by applying transfer move v_t to the initial solution, s_i. Next, s_h is optimized using the function *TSAB*(s_h) in the lower level where the TSAB algorithm is used to minimize makespan. Applying TSAB to s_h yields another solution s_l. This work uses an external list L to record the non-dominated solutions located during this search procedure. Once a solution is identified, whether the solutions stored in L are non-dominant will be checked. Any solutions that are dominated by s_l are removed from L, and s_l is then added to L. Conversely, s_l is discarded if it is dominated by at least one solution in L. The function *Update Set* (s_l, L) is responsible for this operation. Thereafter, the local optimal solutions, S_t, found by TSAB^2 (which are stored in L) are returned to the MTM, which archives optimal solutions. In this stage, the function *Update Set* (S_t, MTM) updates the MTM to maintain the non-dominant solutions in L. Notably, TSAB^2 is a multi-objective implementation of TSAB for the MOFJSP that incorporates a Pareto-achieving mechanism.

4.4.2. Tabu lists and aspiration criteria

The PRMOTS is hierarchical and involves two Tabu lists: the upper level Tabu list, T_h, which stores transfer moves in the upper level, and the lower level Tabu list, T_l, which stores interchange moves in the lower level. Let $v_t = (o, m_c, m_t)$ be the transfer move that transfers operation o from its current machine m_c to a selected machine m_t, and $v_i = (j_1, j_2, m)$ be the exchange move that swaps jobs j_1 and j_2 on machine m. v_t and v_i are performed to find neighbors. Then, if the neighbors are local optima, reverse moves $\bar{v}_t = (o, m_t, m_c)$ and $\bar{v}_i = (j_2, j_1, m)$ are added to the Tabu lists by $T_h \oplus \bar{v}_t$ and $T_l \oplus \bar{v}_i$. The lengths of T_h and T_l are set by parameter *tlistLen*. In the upper level, when a move is marked Tabu, more than one candidate solution may be Tabu. The Tabu status of a move is therefore overridden when the incumbent candidate solution is good. However, with respect to the small neighborhood size of N_3, no aspiration rule is imposed in the lower level.

Algorithm 2. TSAB^2.

Input: Initial solution s_i, *higherIter*
Output: A set of local optimal solutions R
$hiter \leftarrow 0$, $T_h \leftarrow \emptyset$, $L \leftarrow \emptyset$
While $hiter < higherIter$
    $v_t \leftarrow$ Generate *Transfer Move* by N_1 or N_2
    $s_h \leftarrow$ *Execute Move*(s_i, v_t) //the upper level
    $s_l \leftarrow$ *TSAB*(s_h) //the lower level
    $L \leftarrow$ Update Set(s_l, L)
If L is updated
    $T_h \leftarrow T_h \oplus \bar{v}_t$
    $hiter \leftarrow 0$
Else
    $hiter \leftarrow hiter + 1$
End if

4.5. The path-relinking heuristics

The PR heuristics is based on the transformation of an initial solution graph into a reference solution graph. The distance between two solutions is defined by the number of operations that are located in different positions on the coding strings. Algorithm 3 is a pseudo code of the PR heuristics. After each *pathFreq* global iteration, a (s_i, s_r) pair is selected, where s_i is an initial solution from the set generated previously by TSAB^2, while s_r is a reference solution picked from the optimal solutions found so far (stored in the MTM). Function *Path Generation* (s_i, s_r) is used to find a set of intermediate solutions located between s_i and s_r. The function first randomly selects an operation $\pi_k(u)$ (the uth operation on machine k) in s_r, then its immediate predecessor, $\pi_k(u-1)$, and successor, $\pi_k(u+1)$, are also selected. An intermediate solution s_i' is therefore obtained by moving those three operations to positions $u-1$, u, and $u+1$ on machine k in s_i. If the positions are unavailable on machine k in s_i, then the operations are moved to the end of the machine's queue. Thereafter, the algorithm repeats this procedure on the pair (s_i', s_r) to locate another solution. These steps are performed iteratively until s_i is identical to s_r, or the number of intermediate solutions exceeds a given parameter *nPath*. Fig. 3 shows a sample pair (s_i, s_r) and Fig. 4 shows the transformation of s_i into s_r by the path generation method.

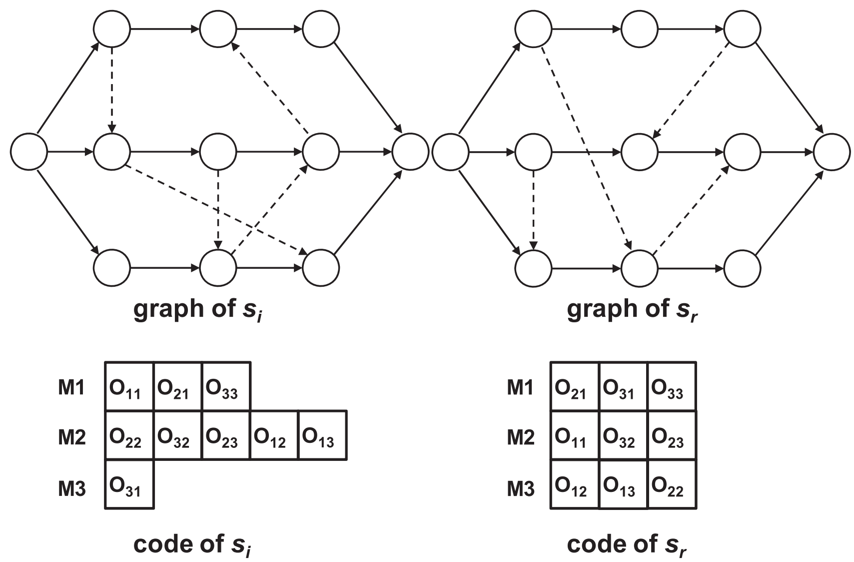


Fig. 3. An example of a (s_i, s_r) pair.

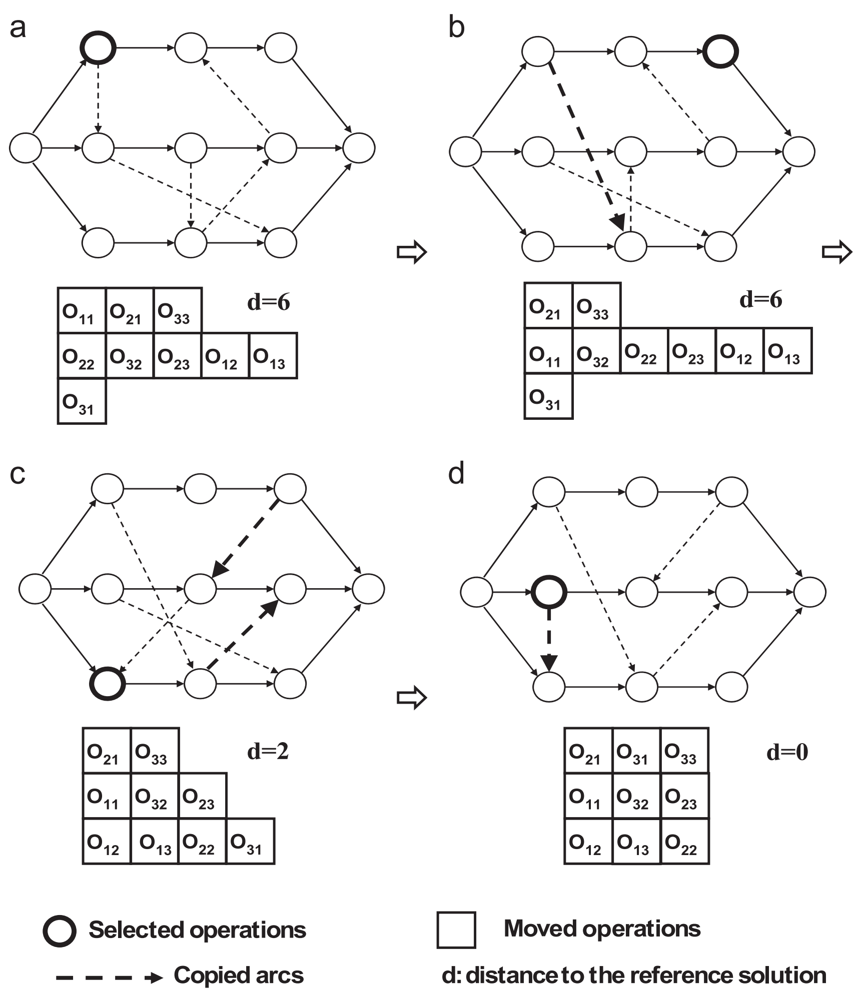


Fig. 4. Path-relinking on (s_i, s_r).

During each iteration, new arcs are “copied” from the graph of reference solution to the graph of initial solution, increasing the “similarity” of solutions. Notably, the distance between s_i and s_r is not always reduced in every iteration. However, since operations are moved to exact positions relative to the reference solution, the distance becomes zero after a few iterations.

Because the local search procedure of the TSAB^2 yields a set of local optimal solutions rather than a single solution, it may be costly in terms of time for PR to deal with all solution pairs when the solution set generated by the TSAB^2 is very large. Thus, the parameter *psetSize* is introduced to control the number of solution pairs handled by PR. A random selection scheme is applied with equal selection probability to the solutions from the initial solution set (S_i) and the reference solution set (S_r). Hence, $\min(psetSize, |S_i|) \times \min(psetSize, |S_r|)$ solution pairs will be handled by the PR procedure. To further reduce computational effort, some intermediate solutions between S_i and S_r are omitted and only the first *nPath* intermediate solutions are generated and evaluated by the proposed path generation method.

Algorithm 3. Path-relinking.

Input: Initial solution set S_i, reference solution set S_r, *psetSize*, *nPath*
Output: Best encountered solution set V

$V \leftarrow \varnothing$, $S_r' \leftarrow S_r$, $iter_1 \leftarrow 0$, $iter_2 \leftarrow 0$
While $iter_1 < \min(psetSize, |S_i|)$
  $s_i \leftarrow$ Randomly select a solution from S_i
  Remove s_i from S_i
  $iter_1 \leftarrow iter_1 + 1$
  While $iter_2 < \min(psetSize, |S_r|)$
    $s_r \leftarrow$ Randomly select a solution from S_r'
    Remove s_r from S_r'
    $G \leftarrow Path\ Generation\ (s_i, s_r)$
    $V \leftarrow Update\ Set\ (G, V)$
    $iter_2 \leftarrow iter_2 + 1$
  End While
  $S_r' \leftarrow S_r$
End While

4.6. Detecting and breaking cycles

Notably, the movement of operations by PR leads to cycles under some circumstances. Bożejko et al. [27] discussed several cases in which transfer moves lead to cycles in a directed graph. In this work, infeasible transfers are not predicted; instead, a detection and repair mechanism is devised to ensure the feasibility of intermediate solutions. After the operation movements are performed, the moved operations are then stored in a set O^m. The algorithm calculates start times of operations in O^m by decoding. If the incumbent solution is infeasible, a cycle will be detected when calculating an operation o. To break this cycle, an active operation o' is selected by the decoding algorithm (Section 4.1). Path generation is then performed on o' to yield another solution. This selection and move process are performed repeatedly until the start time of o is obtained. Afterwards, the cycle detection scheme is used again for another operation in O^m to determine whether the graph has any other cycles. Via this check-and-repair mechanism, a feasible intermediate solution can be identified. The pseudo code of the check-and-repair scheme is provided in Algorithm 4. Function *Path Generation* (o, s_i, s_r) is an overridden function of *Path Generation*(s_i, s_r), which generates an intermediate solution by moving operation o. Notably, the check-and-repair process does not lead to infinite loops because active operations are moved to absolute positions based on the reference solution, and an initial solution is therefore transferred directly into the reference solution in the worst case. In fact, the worst case seldom occurs and, according to experimental results, one can always find a set of feasible intermediate solutions using only a few check-and-repair iterations.

Fig. 5 presents how cycles are detected and broken. Applying path generation on o_{22} generates a new arc $a(o_{13}, o_{22})$, which constructs a cycle with $a(o_{23}, o_{12})$ (Fig. 5(b)). The algorithm then selects an active operation o_{23}. Path generation is therefore applied to o_{23} in the following step to remove the cycle from the graph.

Algorithm 4. Check-and-repair.

Input: Moved operations O^m, the resultant intermediate solution s_i, the reference solution s_r
Output: A feasible intermediate solution s_f

Foreach operation o in O^m
  Calculate t_o^s by decoding algorithm
  While cycle detected
    $o' \leftarrow$ select an active operation
    $O \leftarrow O \cup \{o'\}$
    $s_i \leftarrow Path\ Generation\ (o', s_i, s_r)$
    Calculate t_o^s by decoding algorithm
  End While
  $O \leftarrow O \setminus \{o\}$
End Foreach

4.7. Dimension-oriented intensification search

The primary problem associated with the TS is that it easily gets stuck in a small area in the search space when the search space is

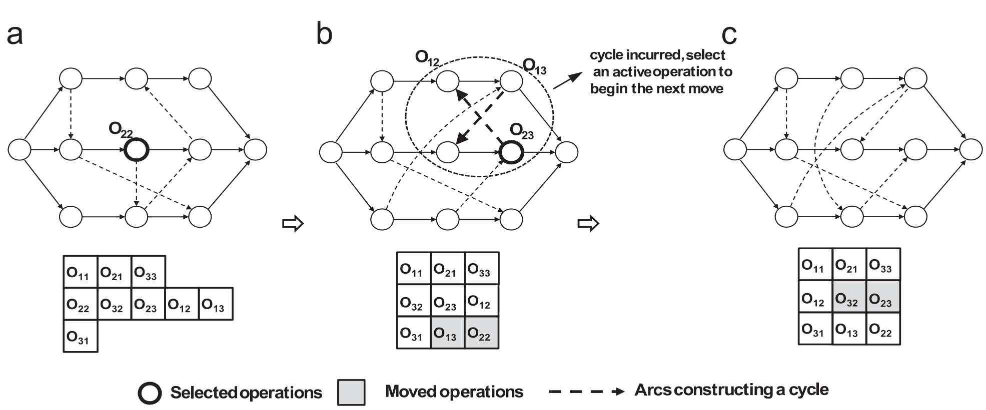


Fig. 5. Breaking cycles incurred during path relinking.

very large or has a high dimensionality. Although the PRMOTS can cover most good regions with proper neighborhood search strategies, as does the PR heuristics, according to experimental study, the acquired solutions are typically clustered within an area. To cope with this pitfall, additional operators are defined based on the concept of extreme solutions [29]. With respect to each objective in the MOFJSP, a dimension-oriented IS is performed for a chosen solution, attempting to find a new solution that is located near the extreme solution on that dimension (objective). If such a solution is found, it is used to find additional non-dominated solutions beyond the solutions found by the PRMOTS. Fig. 6 illustrates how the IS works to find extra solutions. The operators that implement the IS are as follows:

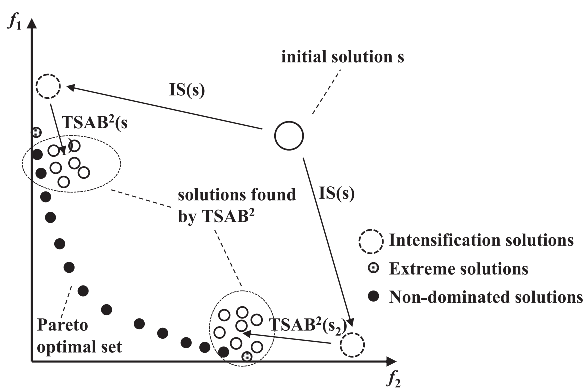


Fig. 6. An illustration of the dimension-oriented intensification search.

1. *Intensification search on makespan*: Select the last operation from a critical path, then assign it to the end of another machine. When this reassignment shortens makespan, select the last operation of another critical path and perform reassignment again. The search terminates when a critical path does not exist on which reassignment shortens makespan.
2. *Intensification search on total workload*: Randomly select an operation, and assign it to a feasible position on a target machine. A “feasible position” means that the precedence constraint holds for all jobs when the selected operation is moved there. “Target machine” is the machine on which processing time of the selected operation is the shortest. When the target machine is identical to the current machine, the reassignment is canceled. Next, another operation is selected and reassignment is performed again. The search terminates when all operations are assigned to target machines.
3. *Intensification search on maximum workload*: Randomly select an operation from the machine whose workload is largest, and assign the operation to a feasible position on a machine whose workload is smallest. Then update the workload for each machine. Thereafter, reassignment is performed again. The search terminates when any reassignment increases maximum workload.

Incorporating IS to the PRMOTS (namely, PRMOTS+IS) forms a new version of algorithm that is able to locate solutions near

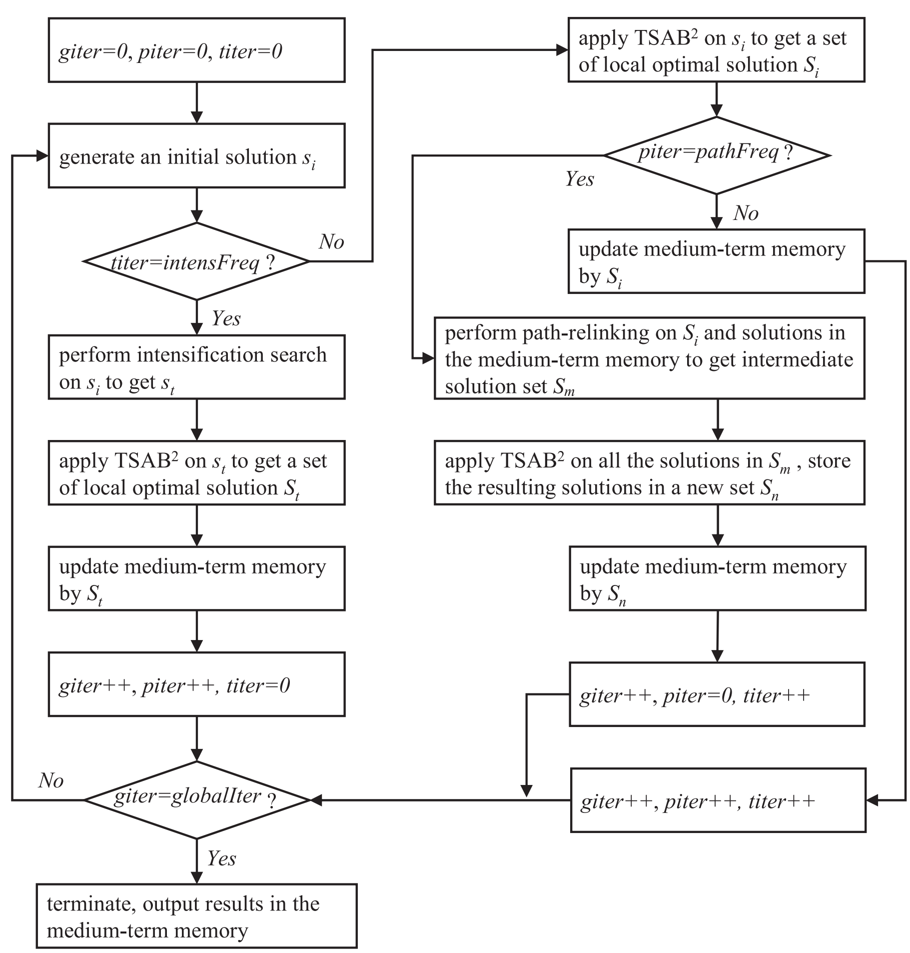


Fig. 7. Framework for the PRMOTS+IS.

extreme solutions. According to the experimental studies, PRMOTS+IS outperforms PRMOTS with only a little increased computational time incurred.

4.8. Framework of the PRMOTS+IS

The proposed PRMOTS+IS does not have long-term memory that can record visited initial solutions. The search starts at an initial solution generated by tailored strategies, and it is then stored in short-term memory (STM) where its local optimal neighbors are found by applying TSAB^2. An unbounded MTM is adopted to archive non-dominated solutions and is utilized by the PR heuristics to generate diversified solutions. When the number of global iterations reaches a user-specified threshold, the search is diversified and intensified by executing the PR heuristics and IS. Thus, PRMOTS incorporates a systematic local search with intelligent coverage of the search space. Fig. 7 presents the framework for the PRMOTS+IS.

5. Computational experiments

The proposed algorithms for the MOFJSP were implemented by using C#.NET 2010. Numerical experiments were conducted on a PC with two Intel Core TM2 T8100 @ 2.1 GHz processors and 4 GM RAM. The algorithm was tested using four sets of benchmark instances found on http://www.idsia.ch/~monaldo/fjsp.html#ProblemInstances (access time: 2014-01-20).

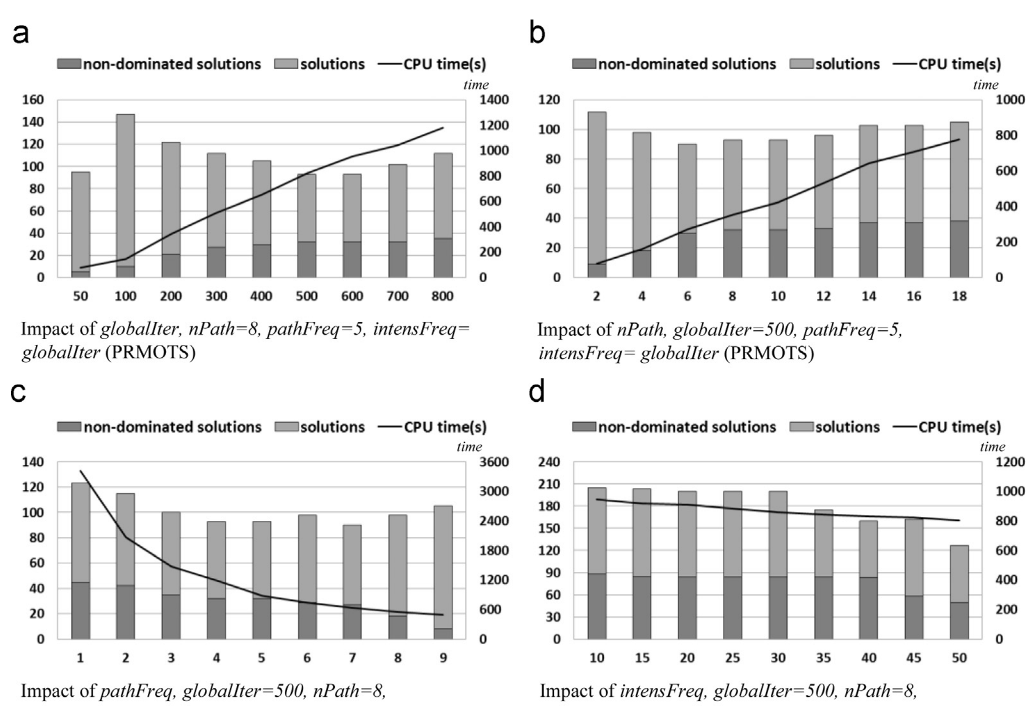
1. *KacemData*: This set of five instances was taken from [10]; the problem scale ranges from 4×5 (jobs $\times$ machines), 12 operations to 15×10, 56 operations.
2. *HUData*: This set of 15 instances was taken from [8]; the problem scale ranges from 10×5, 50 operations to 20×5, 100 operations.
3. *BRData*: This set of 10 instances was taken from [4]; the problem scale ranges from 10×6, 55 operations to 20×15, 240 operations.
4. *DPData*: This set of 18 instances was taken from [37]; the problem scale ranges from 10×5, 196 operations to 20×10, 387 operations.

5.1. Benchmark algorithms

Typical approaches and algorithms for the FJSP can be categorized into three classes based on how objectives are handled: first, minimization of makespan only; second, the objectives are considered by a linear weighted sum of all objectives; and third, the objectives are handled by Pareto optimization. The algorithms that

Table 2
Parameters of the PRMOTS+IS.

Parameter	Description	Test range	Selected value
globalIter	Total iterations of the PRMOTS+IS	100–800	500
higherIter	The maximum iterations for the upper level of TSAB^2	–	*num_ops/3*
intensFreq	Perform intensification search each *intensFreq* iterations	10–50	25
pathFreq	Perform path relinking each *pathFreg* iterations	1–9	5
psetSize	Number of solution pairs for path relinking	–	10
nPath	Intermediate solutions generated by path generation method	2–18	8
tlistLen	Length of Tabu list	–	10



a: Impact of *globalIter, nPath=8, pathFreq=5, intensFreq= globalIter* (PRMOTS)

b: Impact of *nPath, globalIter=500, pathFreq=5, intensFreq= globalIter* (PRMOTS)

c: Impact of *pathFreq, globalIter=500, nPath=8, intensFreq= globalIter* (PRMOTS)

d: Impact of *intensFreq, globalIter=500, nPath=8, pathFreq=5* (PRMOTS+IS)

Fig. 8. Impacts of *globalIter*, *pathFreq*, *nPath* and *intensFreq* on solutions for the MK10.

deal with a single objective or a weighted sum of objectives typically obtain a single solution or a small set of solutions after several runs. Such algorithms are not compared with PRMOTS and PRMOTS+IS here. Only those that belong to the Pareto-based approach are selected as benchmark algorithms in this experimental study. Six benchmark algorithms are as follows:

1. Multi-objective evolutionary algorithm and guided local search (MOEA+GLS) [28].
2. Multi-objective genetic algorithm (MOGA) [30].
3. Multi-objective particle swarm optimization and local search (MOPSO+LS) [18].
4. Artificial bee colony algorithm (ABC) [32].
5. Pareto-based artificial bee colony algorithm (EPABC) [33].
6. Simple evolutionary algorithm (SEA) [3].

5.2. Parameter tuning

The key parameters (*globalIter*, *pathFreq*, *nPath*, and *intensFreq*) are tuned to achieve best algorithmic performance. The tuning space is $D_1 \times \cdots \times D_n$, where D_i is the domain of the ith parameter. The "Test Range" column in Table 2 lists the domains of the four parameters in this tuning process. Other parameters are fixed *a priori*. Five runs are executed for each configuration and average values are used to evaluate the performance. Instance MK10 from the BRData is utilized to test the algorithm, this instance contains the most operations among the dataset and each machine can process more operations on average than that of any other instances in the dataset. Hence, this instance is more difficult to solve. To demonstrate the impact of each parameter on performance, Fig. 8 lists the results of some tuning experiments, where the light-grey bars represent the number of solutions found by the PRMOTS or PRMOTS+IS and dark-grey bars represent the number of solutions that are not dominated by the solutions found so far. Performance under different configurations is evaluated in terms of the number of non-dominated solutions and overall computational times. The following observations are based on the experimental results:

1. Computational times increase linearly as *globalIter* and *nPath* increase (Fig. 8(a) and (b)), indicating that the running scales change steadily under different parameter configurations.
2. Computational times are very affected by *pathFreq* (Fig. 8(c)). The exponential increment of computational time when *pathFreq* deceases indicates that the running scale and parameter values are not linearly related. To give an explanation, the PRMOTS performs the PR each *pathFreq* iterations. Hence, the total times PR is performed during the entire execution of algorithm is calculated by $globalIter/pathFreq$. This means that the running scale of the PRMOTS is not linearly related to *pathFreq*.
3. Increasing the number of non-dominated solutions usually decreases the number of total solutions when *globalIter* and *nPath* are small values. However, when these two values are sufficiently big, the number of non-dominated solutions hardly increases (without an IS), indicating that the PRMOTS gets stuck in a region where a portion of optimal solutions are located.
4. The number of non-dominated solutions increases markedly with just a few runs of the IS (Fig. 8(d)). Moreover, the IS does not bring much computational burden. Hence, the IS is efficient in finding optimal solutions.

According to the experimental results as well as performance evaluations, the best values of the parameters are selected (Table 2).

5.3. Performance on the KacemData and HUData

First, PRMOTS and PRMOTS+IS were tested by using the KacemData and HUData, which are relatively small instances. For the KacemData, instead of presenting all the solutions found, this work only makes a list of the number of solutions and the number of non-dominated solutions. Most algorithms found the same number of solutions for each instance (Tables 3–5). An exception was the ABC algorithm that found fewer solutions for the Kacem 4×5, Kacem 8×8 and Kacem 10×10 instances than other algorithms. The MOGA found fewer solutions for the Kacem 4×5 and Kacem8×8 instances; however, it found one more solutions for Kacem 10×15 than other algorithms. With regard to the non-dominance of the obtained solutions, both PRMOTS and PRMOTS+IS found the most non-dominated solutions, as did EPABC and SEA for all five instances. Although MOEA+GLS was tested with only three instances, it also found the best solutions

Table 3
Non-dominated solutions for the KacemData.

Instance ($n\times m$)	MOEA+GLS	MOGA	MOPSO+LS	ABC	EPABC	SEA	PRMOTS	PRMOTS+IS
Kacem 4×5	–	3/3	–	3/3	4/4	4/4	4/4	4/4
Kacem 8×8	4/4	3/3	3/5	3/3	4/4	4/4	4/4	4/4
Kacem 10×7	–	–	–	1/3	3/3	3/3	3/3	3/3
Kacem 10×10	4/4	4/4	4/4	3/3	4/4	4/4	4/4	4/4
Kacem 15×10	2/2	1/3	1/2	0/2	2/2	2/2	2/2	2/2

Note: in the format "a/b" in each cell, "b" refers to the number of solutions obtained, while "a" refers to the number of solutions that are not dominated by any solution found by other algorithms.

Table 4
Computational times for the KacemData.

Instance ($n\times m$)	MOEA+GLS	MOGA	MOPSO+LS	EPABC	SEA	PRMOTS	PRMOTS+IS
Kacem 4×5	–	5.8	–	1.2	–	9.4	9.9
Kacem 8×8	9.1	9.5	–	1.5	–	10.0	10.8
Kacem 10×7	–	–	–	4.9	–	13.2	14.5
Kacem 10×10	16.6	14.2	–	6.2	–	18.0	19.4
Kacem 15×10	24.1	87.5	–	18.5	–	26.9	28.7

Table 5
Solutions for the HUData.

Instance	Scale[a]	Bożejko et al.		PRMOTS				PRMOTS+IS			
		f_1	T	f_1	f_2	f_3	T	f_1	f_2	f_3	T
la01	(10 × 5.50)	574	16.81	572	2849	572	22.8	572	2849	572	24.6
				583	2849	570		584	2849	571	
la02	(10 × 5.50)	532	14.67	529	2643	529	19.7	530	2643	530	25.2
								558	2643	529	
la03	(10 × 5.50)	482	15.52	482	2383	480	25.6	478	2383	478	26.5
				483	2383	479		496	2383	477	
				491	2383	478					
la04	(10 × 5.50)	509	12.65	504	2507	504	31.2	502	2507	502	32.0
				508	2507	503					
				509	2507	502					
la05	(10 × 5.50)	462	5.77	459	2283	459	32.5	458	2283	458	35.0
				474	2283	457		483	2283	457	
la06	(15 × 5.75)	801	5.33	800	3992	800	29.7	799	3992	799	31.2
la07	(15 × 5.75)	751	12.22	750	3745	750	28.7	750	3745	750	30.0
la08	(15 × 5.75)	767	1.78	766	3825	766	33.3	766	3825	766	33.1
la09	(15 × 5.75)	856	9.58	853	4263	853	26.3	853	4263	853	28.5
la10	(15 × 5.75)	807	17.36	805	4020	805	29.5	805	4020	805	30.2
								829	4020	804	
la11	(20 × 5.100)	1072	19.87	1071	5351	1071	32.6	1071	5351	1071	38.9
la12	(20 × 5.100)	937	13.31	936	4676	936	37.6	936	4676	936	40.1
la13	(20 × 5.100)	1039	47.34	1038	5186	1038	31.9	1038	5186	1038	42.8
la14	(20 × 5.100)	1071	3.67	1070	5346	1070	35.4	1070	5346	1070	36.7
la15	(20 × 5.100)	1090	19.07	1090	5445	1090	37.5	1090	5445	1090	44.1

T=CPU (s).
[a] $n \times m$, the number of operations.

Table 6
Performances on the BRData.

Instance	Scale[a]	MOGA				SEA				PRMOTS				PRMOTS+IS			
		NS	N	R	T	NS	N	R	T	NS	N	R	T	NS	N	R	T
MK01	(10 × 6.55)	3/4	0	0.27	29.4	**11/11**	0	**1.00**	–	**11/11**	0	**1.00**	43.1	**11/11**	0	**1.00**	58.3
MK02	(10 × 6.58)	4/6	1	0.50	45	6/7	1	**0.75**	–	5/9	0	0.62	52.3	**6/9**	1	**0.75**	61.7
MK03	(15 × 8.150)	9/10	9	0.35	285	**17/17**	0	**0.65**	–	9/9	0	0.35	123.9	**17/17**	0	**0.65**	185.8
MK04	(15 × 8.90)	6/10	6	0.23	105.6	**19/19**	7	**0.73**	–	8/17	0	0.31	75.4	12/23	1	0.46	86.1
MK05	(15 × 4.106)	5/5	0	0.45	140.4	**10/10**	1	**0.91**	–	9/13	0	0.82	78.5	9/15	1	0.82	91.1
MK06	(10 × 15.150)	7/10	7	0.06	115.8	62/110	50	**0.55**	–	10/85	1	0.09	191.7	55/126	43	0.49	218.2
MK07	(20 × 5.100)	7/7	2	0.44	295.2	13/13	0	0.81	–	10/15	1	0.62	82.5	**14/23**	0	**0.88**	101.3
MK08	(20 × 10.225)	5/5	2	0.50	722.4	8/8	0	**0.80**	–	7/11	0	0.70	167.2	**8/9**	0	**0.80**	560.6
MK09	(20 × 10.240)	3/9	3	0.04	1168.8	**64/64**	60	**0.93**	–	1/50	0	0.01	214.7	6/93	2	0.09	794.1
MK10	(20 × 15.240)	2/18	2	0.01	1072.2	**110/138**	110	**0.56**	–	32/93	1	0.16	355.2	84/200	84	0.43	822.8

Note: MOGA [30]; SEA, http://web.ntnu.edu.tw/~tcchiang/; T=CPU (s).
[a] $n \times m$, the number of operations.

since solutions reported were all non-dominated. As for the MOGA, MOPSO+LS and ABC, only a part of the non-dominated solutions were found. Based on the analysis of the non-dominance of solutions, we conclude that PRMOTS and PRMOTS+IS are among the best class at solving small-scale problems.

For the HUData, no multi-objective optimization results are reported in the literature. The performance of the PRMOTS and PRMOTS+IS are compared with that of the parallelized TS algorithm (that minimizes makespan) by Bożejko et al. [27]. The PRMOTS and PRMOTS+IS are better than the parallelized TS at minimizing makespan for most instances. In terms of the quality of solutions for multi-objective optimization, PRMOTS+IS performs slightly better than PRMOTS.

5.4. Performance on the BRData and DPData

Next, the PRMOTS and PRMOTS+IS were tested by using medium and large instances. The BRData is the most popular set of instances tested in the literature. For comparisons, the two most successful algorithms, the MOGA and SEA, are chosen from the benchmark algorithms that obtained the most high-quality solutions for this dataset. The solutions obtained by each algorithm are not listed; instead, two additional indicators are used to assess performance. The "N" stands for the number of non-dominated solutions that are not found by other algorithms, and "R" stands for the ratio of obtained non-dominated solutions to the unified non-dominated solutions (all non-dominated solutions found so far) (Tables 6 and 8). Additionally, to show the quantity and quality of solutions as well as their distributions, solutions are presented by their objective ranges in the form of a "Min, Max" pair for each objective (Tables 7 and 9), where Min and Max are the minimum and maximum values obtained for the objective. Moreover, the objective ranges of unified solutions are also given. Readers may refer to Table A1 in Appendix A for all solutions found by the PRMOTS+IS for the BRData.

The PRMOTS+IS found a better spread of solutions than the MOGA in terms of both the number of solutions and the range of objective values. More solutions were found by the PRMOTS+IS

Table 7
Objective ranges for the BRData.

Instance	min f_1, max f_1				min f_2, max f_2				min f_3, max f_3			
	MOGA	SEA	PRMPTS+IS	Unified	MOGA	SEA	PRMPTS+IS	Unified	MOGA	SEA	PRMPTS+IS	Unified
MK01	40, 43	**40, 45**	**40, 45**	40, 45	154, 169	**153, 167**	**153, 167**	153, 169	36, 40	**36, 42**	**36, 42**	36, 42
MK02	**26, 33**	27, 33	27, 33	26, 33	140, 151	**140, 150**	**140, 151**	140, 151	**26, 33**	**26, 33**	**26, 33**	26, 33
MK03	204, 230	**204, 330**	**204, 330**	204, 330	847, 884	812, 850	812, 850	812, 884	133, 210	204, 330	204, 330	133, 330
MK04	60, 90	61, 146	63, 146	60, 146	331, 390	324, 372	324, 366	324, 390	54, 76	60, 146	60, 146	54, 146
MK05	173, 179	**173, 209**	174, 209	173, 209	682, 679	**672, 687**	**672, 687**	672, 687	173, 185	173, 209	**172, 209**	172, 209
MK06	60, 78	65, 103	63, 100	60, 103	330, 441	**330, 456**	**330, 456**	330, 456	54, 90	50, 99	**49, 100**	49, 100
MK07	139, 166	143, 217	141, 217	139, 217	657, 693	**649, 694**	**649, 694**	649, 694	138, 166	143, 217	139, 217	138, 217
MK08	**523, 587**	524, 587	**523, 587**	523, 587	2484, 2534	2484, 2519	2484, 2524	2484, 2534	497, 587	524, 587	523, 587	497, 587
MK09	310, 332	311, 454	**310, 454**	310, 454	2259, 3514	2210, 2273	**2210, 2295**	2210, 2295	299, 308	**299, 454**	**299, 454**	299, 454
MK10	214, 281	225, 290	222, 308	214, 308	1854, 2084	1847, 1986	**1847, 2014**	1847, 2014	204, 268	196, 290	**189, 290**	189, 290

Note: Bold pairs represent ranges equal to the unified ranges.

Table 8
Performances on the DPData.

Instance	Scale[a]	MOGA			PRMOTS			PRMOTS+IS		
		NS	*R*	*T*	*NS*	*R*	*T*	*NS*	*R*	*T*
01a	(10×5196)	1/2	0.25	122.5	0/1	0.00	156.0	**3/3**	**0.75**	160.4
02a	(10×5196)	2/2	0.29	153.4	1/8	0.14	177.0	**4/6**	**0.57**	185.1
03a	(10×5196)	2/2	0.25	174.0	0/5	0.00	192.5	**6/6**	**0.75**	201.8
04a	(10×5196)	3/4	0.14	124.2	13/13	0.59	342.3	**14/14**	**0.63**	351.7
05a	(10×5196)	7/10	0.05	142.4	22/146	0.16	668.0	**126/154**	**0.90**	692.1
06a	(10×5196)	2/7	0.03	185.6	**65/95**	**0.97**	523.0	40/119	0.60	576.6
07a	(15×8293)	**3/3**	**0.75**	457.8	0/1	0.00	252.5	1/1	0.25	258.2
08a	(15×8293)	2/2	0.20	496.0	0/3	0.00	342.5	**8/8**	**0.80**	363.6
09a	(15×8293)	3/3	**0.50**	609.6	1/4	0.17	339.5	**3/4**	**0.50**	364.7
10a	(15×8293)	7/8	0.06	452.8	12/111	0.10	652.8	**110/130**	**0.94**	678.9
11a	(15×8293)	4/7	0.02	608.2	125/226	0.73	821.5	**146/197**	**0.85**	803.5
12a	(15×8293)	4/10	0.04	715.4	1/145	0.01	799.0	**99/166**	**0.95**	812.2
13a	$(20 \times 10{,}387)$	1/1	0.25	1439.4	**3/3**	0.00	969.5	**3/3**	**0.75**	1007.2
14a	$(20 \times 10{,}387)$	2/2	0.40	1743.2	1/3	0.20	914.2	**3/3**	**0.60**	933.1
15a	$(20 \times 10{,}387)$	2/2	0.40	1997.1	0/3	0.00	637.0	**3/3**	**0.60**	651.2
16a	$(20 \times 10{,}387)$	8/9	0.04	1291.4	98/217	0.54	2137.1	**163/200**	**0.89**	2174.3
17a	$(20 \times 10{,}387)$	5/9	0.04	1708.0	94/176	0.80	1721.3	**104/171**	**0.88**	1903.8
18a	$(20 \times 10{,}387)$	3/6	0.03	1980.4	71/118	0.70	1854.5	**94/133**	**0.92**	2021.2

Note: MOGA=[30], *T*=CPU (s).

[a] $n \times m$, the number of operations.

Table 9
Objective ranges for the DPData.

Instance	min f_1, max f_1		min f_2, max f_2		min f_3, max f_3	
	MOGA	PRMPTS+IS	MOGA	PRMPTS+IS	MOGA	PRMOTS+IS
01a	2568, 2594	2592, 2596	11,137, 11,137	11,137, 11,173	2505, 2568	2549, 2508
02a	2289, 2313	2345, 2441	11,137, 11,137	11,137, 11,137	2238, 2263	2228, 2238
03a	2256, 2287	2317, 2351	11,137, 11,137	11,137, 11,137	2248, 2252	2228, 2245
04a	2250, 3095	2647, 2786	11,064, 11,090	11,064, 11,087	2503, 2727	2503, 2727
05a	2292, 3056	2426, 2852	10,941, 11,091	10,941, 10,988	2242, 2620	2194, 2497
06a	2250, 2967	2328, 2769	10,839, 11,009	10,809, 10,848	2219, 2840	2164, 2347
07a	2450, 2484	2522, 2522	16,485, 16,485	16,485, 16,485	2289, 2413	2187, 2187
08a	2171, 2187	2276, 2465	16,485, 16,485	16,485, 16,485	2102, 2104	2062, 2093
09a	2144, 2158	2254, 2458	16,485, 16,485	16,485, 16,485	2102, 2119	2062, 2068
10a	2461, 3064	2656, 2912	16,464, 16,547	16,464, 16,512	2265, 2734	2178, 2629
11a	2182, 2962	2416, 2954	16,247, 16,476	16,135, 16,194	2113, 2389	2021, 2328
12a	2161, 2683	2339, 2745	16,104, 16,355	15,748, 15,809	2084, 2397	1974, 2115
13a	2408, 2408	2643, 2708	21,610, 21,610	21,610, 21,610	2326, 2326	2203, 2215
14a	2334, 2340	2493, 2522	21,610, 21,610	21,610, 21,610	2251, 2258	2165, 2168
15a	2285, 2287	2549, 2595	21,610, 21,610	21,610, 21,610	2218, 2247	2164, 2173
16a	2447, 3106	2735, 3236	21,478, 21,602	21,478, 21,562	2354, 2258	2206, 2478
17a	2322, 2816	2528, 3002	21,197, 21,454	20,878, 20,972	2224, 2448	2093, 2268
18a	2267, 2545	2416, 2901	21,282, 21,483	20,566, 20,621	2206, 2310	2061, 2198

for most instances than the SEA; however, the SEA achieved better convergence performance since the number non-dominated solutions found by the PRMOTS+IS were slightly fewer than those by the SEA for some instances. An exception was the MK09 instance. The PRMOTS+IS algorithm found more solutions, but far fewer non-dominated solutions than the SEA. Although the SEA by

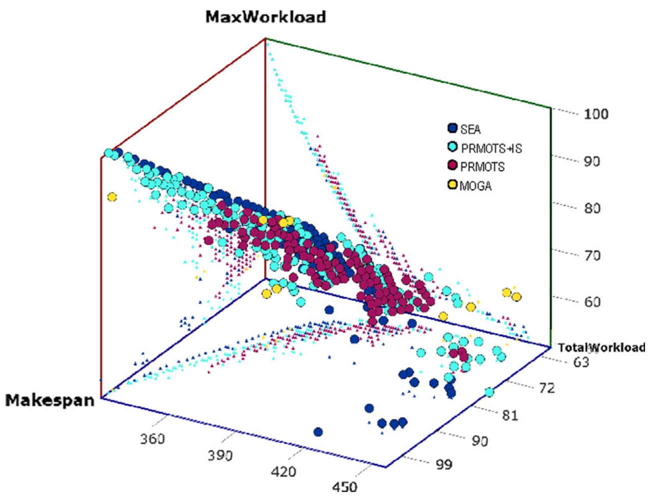


Fig. 9. Distributions of solutions found for the MK06.

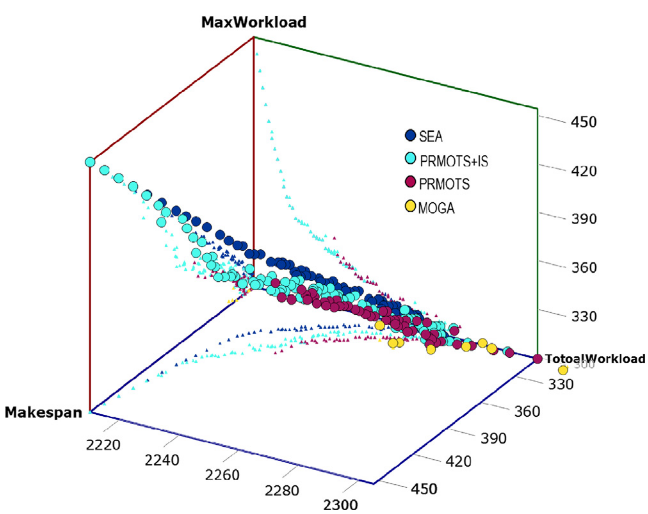


Fig. 10. Distributions of solutions found for the MK09.

Chiang and Lin [3] found more non-dominated solutions than the PRMOTS+IS, the computational times were not reported in their work. With regard to the distribution of solutions, the PRMOTS+IS found more dispersed solutions than the other two algorithms for most instances, especially for large instances. For example, the PRMOTS+IS found a range of solutions equal to the unified solutions for seven instances with respect to total workload. However, the MOGA and SEA found best ranges of solutions only for zero and three instances, respectively.

The objective ranges of the PRMOTS are not given in Table 7. The three most typical instances, MK06, MK09 and MK10, are chosen for brief comparisons of the performance of the PRMOTS and that of the MOGA, SEA and PRMOTS+IS (Figs. 9–11). The PRMOTS found a better spread of solutions than the MOGA. With IS incorporated, the PRMOTS+IS can find more solutions that are located near extreme solutions. The spread of solutions found by PRMOTS+IS is comparable to that by the SEA.

Finally, the performances on the DPData are compared (Tables 8 and 9). The solutions by the MOGA had better makespan than solutions by the PRMOTS and PRMOTS+IS. The reason is that the MOGA is makespan-oriented and solutions found are clustered where makespan is minimized. However, the PRMOTS and PRMOTS+IS performed better than the MOGA in terms of total workload and maximum workload as well as the number of non-dominated solutions. Additionally, the spread of solutions is far beyond that by the MOGA for instances la05, la06, la10, la11, 12a, 16a, 17a, and 18a. Thus, we conclude that the PRMOTS and PRMOTS+IS are more effective than the MOGA in terms of exploring a search space.

The DPData can be divided into three subsets based on the number of operations. For each subset, the difficulty of instances increases when the number of capable machines for operations increases (increasing the size of the search space for that instance). We deduce that the number of optimal solutions that can be found increases with an enlarged search space because the number of routing solutions potentially increases when capable machines are added. This can be observed from the PRMOTS+IS results. For instance, the number of non-dominated solutions increases from the instance 01a to 05a, which belong to the same subset. However, this is not always the case according to experimental results. For example, the PRMOTS+IS found the same number of solutions for instances 13a to 15a, and the number of solutions even decreases from instance 16a to 18a. Moreover, computational times increase rapidly as the number of solutions found by the PRMOTS+IS increases. This is due mainly to frequent updating of the MTM that assesses the dominance among solutions. With an increase of the number of solutions stored in the MTM, the

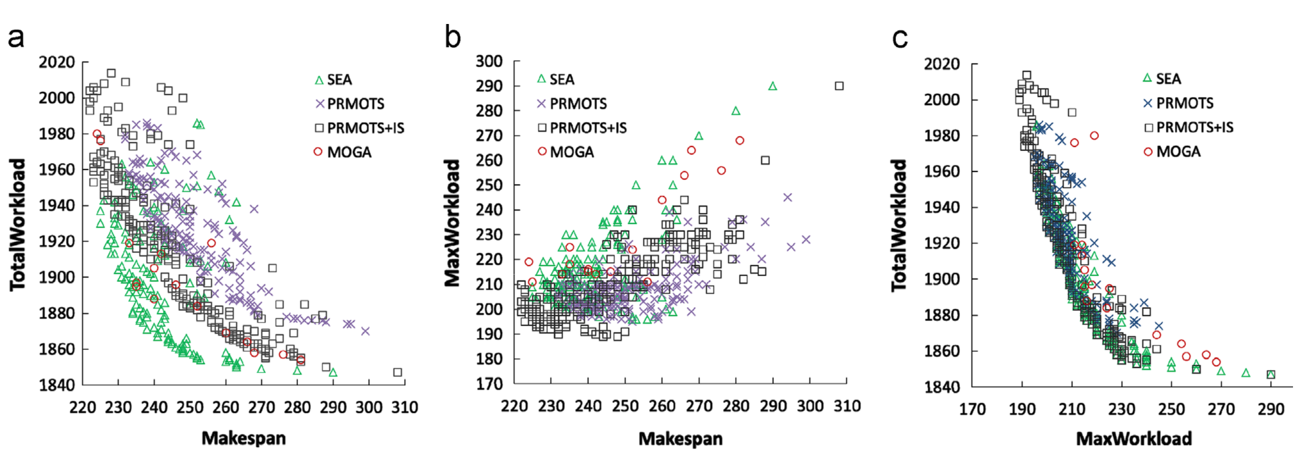


Fig. 11. Solutions found for the MK10 instance.

number of times that dominance is checked for rapidly increases. From these observations, algorithms that are able to find more accurate distributions of non-dominated solutions for the DPData within a relatively short time are interesting and deserve future endeavors.

6. Conclusions and future study

In this study, the MOFJSP is addressed in a Pareto manner and solved by two versions of multi-objective TS algorithms. Comparing related literature, the proposed algorithms have three distinct features. First, a PR heuristics is designed for exploring the search space intelligently. Second, an effective dimension-oriented IS mechanism is developed. Third, the TSAB algorithm is extended to solve multi-objective optimization problems. The proposed algorithms were tested on standard instances and the computational results were compared with those of benchmark algorithms. Comparison results show that the first version of the proposed algorithm (PRMOTS) is competitive with most benchmark algorithms. With the IS (PRMOTS+IS), the algorithm was further strengthened and was comparable to Chiang's SEA [3]. Moreover, the easy-to-implement advantage of the TS makes the proposed algorithms applicable for practical problems.

The development of local search mechanisms and their integration into the PRMOTS+IS are interesting research directions. Another possible algorithmic improvement is to incorporate a reference solution selection strategy into the proposed algorithms to perform path generation, which may potentially elevate the quality of intermediate solutions. Finally, to obtain sufficiently high-quality solutions while keeping reasonable computational performance, parallelization approaches are very promising.

Acknowledgment

The authors would like to show their gratitude to two anonymous referees whose comments greatly improved the presentation of the work. This research is partially supported by National Nature Science of China (71101088, 71171129, and 71390521), Science Foundation of Ministry of Education of China and Shanghai (20113121120002, 20123121110004, 12YZ119, 14YZ100), and Science and Technology Commission of Shanghai (12ZR1412800 and 12510501600).

Appendix A

See Table A1.

Table A1
Solutions found by the PRMOTS+IS for the BRData.

Instance	Solutions obtained						
MK01 (10 × 6)	(40, 167, 36)	(40, 164, 37)	(40, 162, 38)	(41, 163, 37)	(41, 160, 38)	(42, 165, 36)	(42, 158, 39)
	(42, 157, 40)	(43, 155, 40)	(44, 154, 40)	(45, 153, 42)			
MK02 (10 × 6)	(27, 146, 27)	(27, 151, 26)	(28, 144, 28)	(28, 145, 27)	(28, 150, 26)	(29, 143, 29)	(30, 142, 30)
	(31, 141, 31)	(33, 140, 33)					
MK03 (15 × 8)	(204, 850, 204)	(210, 848, 210)	(213, 844, 213)	(221, 842, 221)	(222, 838, 222)	(231, 834, 231)	(240, 832, 240)
	(249, 830, 249)	(258, 828, 258)	(267, 826, 267)	(276, 824, 276)	(285, 822, 285)	(294, 820, 294)	(303, 818, 303)
	(312, 816, 312)	(321, 814, 321)	(330, 812, 330)				
MK04 (15 × 8)	(63, 363, 61)	(63, 366, 60)	(64, 354, 62)	(64, 357, 61)	(64, 360, 60)	(65, 353, 62)	(65, 349, 63)
	(66, 345, 66)	(66, 348, 63)	(67, 344, 66)	(67, 347, 65)	(69, 343, 67)	(72, 340, 72)	(78, 337, 78)
	(84, 334, 84)	(90, 331, 90)	(98, 330, 98)	(106, 329, 106)	(114, 328, 114)	(122, 327, 122)	(130, 326, 130)
	(138, 325, 138)	(146, 324, 146)					
MK05 (15 × 4)	(174, 685, 173)	(175, 683, 174)	(176, 682, 176)	(177, 684, 173)	(177, 682, 175)	(178, 680, 178)	(178, 683, 173)
	(179, 679, 179)	(180, 687, 172)	(183, 677, 183)	(185, 676, 185)	(191, 675, 191)	(197, 674, 197)	(203, 673, 203)
	(209, 672, 209)						
MK06 (10 × 15)	(63, 402, 57)	(63, 403, 56)	(64, 405, 54)	(64, 388, 59)	(64, 387, 60)	(64, 389, 58)	(64, 399, 55)
	(64, 392, 56)	(65, 391, 57)	(65, 384, 61)	(65, 386, 60)	(65, 387, 59)	(65, 381, 63)	(65, 380, 64)
	(65, 398, 54)	(65, 383, 62)	(65, 394, 55)	(66, 379, 65)	(66, 378, 66)	(67, 396, 54)	(67, 446, 50)
	(67, 436, 52)	(67, 444, 51)	(67, 385, 60)	(67, 377, 67)	(67, 383, 61)	(67, 382, 62)	(67, 379, 64)
	(68, 431, 52)	(68, 374, 65)	(68, 377, 64)	(68, 379, 61)	(68, 380, 60)	(68, 378, 62)	(69, 440, 50)
	(69, 426, 53)	(69, 372, 64)	(69, 378, 60)	(69, 370, 66)	(69, 371, 65)	(69, 374, 63)	(69, 376, 61)
	(69, 375, 62)	(70, 430, 52)	(70, 373, 63)	(70, 374, 62)	(70, 369, 66)	(70, 368, 67)	(70, 375, 61)
	(70, 377, 60)	(70, 370, 65)	(71, 438, 51)	(71, 366, 68)	(71, 365, 69)	(71, 364, 70)	(71, 371, 64)
	(71, 372, 63)	(71, 367, 67)	(71, 368, 66)	(72, 428, 52)	(72, 439, 50)	(72, 373, 62)	(72, 363, 70)
	(72, 362, 72)	(72, 369, 65)	(72, 370, 64)	(72, 365, 68)	(72, 376, 60)	(73, 362, 71)	(73, 361, 72)
	(73, 366, 67)	(73, 364, 69)	(74, 437, 51)	(74, 358, 74)	(74, 362, 70)	(74, 361, 71)	(74, 360, 72)
	(74, 359, 73)	(75, 424, 53)	(75, 357, 75)	(75, 367, 66)	(75, 368, 65)	(76, 456, 49)	(76, 438, 50)
	(76, 357, 74)	(76, 358, 73)	(76, 356, 76)	(77, 356, 75)	(77, 355, 77)	(77, 364, 68)	(77, 363, 69)
	(78, 359, 72)	(78, 355, 76)	(78, 354, 77)	(79, 355, 75)	(79, 353, 78)	(80, 360, 71)	(80, 353, 77)
	(80, 352, 78)	(80, 354, 76)	(80, 351, 80)	(81, 351, 79)	(81, 350, 80)	(82, 349, 80)	(82, 351, 78)
	(82, 348, 81)	(83, 347, 82)	(83, 350, 79)	(84, 346, 83)	(84, 345, 84)	(84, 356, 74)	(85, 344, 84)
	(86, 346, 82)	(86, 343, 85)	(87, 342, 86)	(87, 341, 87)	(89, 339, 88)	(90, 338, 90)	(91, 337, 90)
	(93, 335, 93)	(93, 336, 91)	(94, 334, 94)	(96, 333, 96)	(97, 332, 97)	(99, 331, 99)	(100, 330, 100)
MK07 (20 × 5)	(141, 692, 141)	(142, 688, 142)	(143, 684, 143)	(144, 673, 144)	(145, 683, 143)	(146, 690, 141)	(146, 694, 140)
	(148, 685, 142)	(148, 689, 141)	(148, 693, 140)	(150, 669, 150)	(150, 685, 140)	(151, 667, 151)	(153, 693, 139)
	(156, 664, 156)	(157, 662, 157)	(161, 660, 161)	(162, 659, 162)	(166, 657, 166)	(175, 655, 175)	(187, 653, 187)
	(202, 651, 202)	(217, 649, 217)					
MK08 (20 × 10)	(523, 2524, 523)	(524, 2519, 524)	(533, 2514, 533)	(542, 2509, 542)	(551, 2504, 551)	(560, 2499, 560)	(569, 2494, 569)
	(578, 2489, 578)	(587, 2484, 587)					
MK09 (20 × 10)	(310, 2295, 299)	(311, 2282, 299)	(313, 2279, 299)	(313, 2278, 307)	(314, 2274, 307)	(314, 2277, 299)	(315, 2273, 307)
	(316, 2272, 307)	(316, 2275, 303)	(317, 2273, 304)	(317, 2274, 303)	(317, 2269, 305)	(317, 2268, 307)	(317, 2267, 310)
	(317, 2276, 299)	(318, 2266, 310)	(318, 2274, 299)	(318, 2271, 303)	(318, 2270, 304)	(318, 2272, 302)	(318, 2273, 301)
	(319, 2273, 299)	(319, 2266, 304)	(319, 2265, 310)	(319, 2264, 315)	(320, 2264, 310)	(321, 2260, 310)	(322, 2257, 312)
	(323, 2256, 312)	(324, 2265, 307)	(324, 2264, 309)	(324, 2255, 314)	(324, 2253, 315)	(325, 2263, 309)	(325, 2252, 316)

Table A1 (*continued*)

Instance	Solutions obtained						
	(326, 2250, 321)	(326, 2264, 308)	(328, 2249, 320)	(330, 2246, 322)	(331, 2248, 320)	(331, 2246, 321)	(331, 2245, 322)
	(332, 2244, 326)	(333, 2247, 320)	(336, 2244, 323)	(336, 2243, 327)	(337, 2242, 328)	(338, 2242, 326)	(339, 2241, 328)
	(340, 2241, 327)	(343, 2238, 334)	(344, 2239, 328)	(345, 2237, 334)	(347, 2241, 326)	(347, 2238, 331)	(348, 2237, 333)
	(348, 2240, 327)	(348, 2235, 334)	(353, 2237, 332)	(354, 2236, 333)	(355, 2236, 332)	(357, 2235, 333)	(357, 2234, 342)
	(358, 2233, 339)	(358, 2234, 334)	(359, 2231, 340)	(361, 2228, 346)	(363, 2230, 340)	(363, 2232, 339)	(364, 2231, 339)
	(370, 2229, 342)	(371, 2227, 346)	(371, 2226, 347)	(372, 2226, 346)	(375, 2225, 347)	(378, 2224, 348)	(378, 2223, 354)
	(378, 2222, 360)	(379, 2222, 355)	(380, 2221, 360)	(382, 2220, 370)	(387, 2220, 366)	(387, 2219, 375)	(390, 2218, 382)
	(391, 2217, 388)	(398, 2216, 398)	(407, 2215, 404)	(407, 2216, 394)	(415, 2214, 414)	(424, 2213, 424)	(434, 2212, 434)
	(444, 2211, 444)	(454, 2210, 454)					
MK10 (20 $\times$ 15)	(222, 1998, 203)	(222, 2004, 199)	(222, 1993, 210)	(223, 2000, 200)	(223, 2006, 195)	(223, 1959, 207)	(223, 1953, 208)
	(223, 1961, 203)	(223, 1967, 202)	(224, 2004, 198)	(224, 1964, 202)	(225, 1959, 200)	(225, 1969, 197)	(225, 1963, 199)
	(225, 1977, 195)	(226, 1989, 194)	(226, 2008, 193)	(226, 1950, 202)	(226, 1946, 204)	(226, 1952, 200)	(226, 1957, 198)
	(226, 1970, 196)	(227, 1987, 194)	(227, 1993, 193)	(227, 1943, 205)	(227, 1946, 200)	(227, 1962, 196)	(228, 2014, 192)
	(228, 1968, 195)	(229, 1943, 204)	(229, 1955, 198)	(229, 1961, 197)	(229, 1935, 210)	(229, 1965, 195)	(230, 1942, 204)
	(230, 1961, 195)	(230, 1959, 196)	(230, 1995, 192)	(230, 1939, 209)	(230, 1954, 198)	(230, 1956, 197)	(231, 1981, 194)
	(231, 1957, 196)	(231, 1934, 207)	(231, 1943, 201)	(232, 2009, 190)	(232, 1983, 192)	(232, 1942, 200)	(232, 1934, 205)
	(232, 1947, 199)	(232, 1939, 203)	(233, 1954, 196)	(233, 1928, 214)	(233, 1932, 200)	(233, 1931, 201)	(233, 1951, 197)
	(234, 1973, 194)	(234, 1979, 193)	(234, 1925, 214)	(234, 1926, 205)	(234, 1930, 204)	(234, 1945, 199)	(235, 1947, 197)
	(235, 1924, 205)	(235, 1916, 210)	(235, 1922, 207)	(236, 1929, 204)	(237, 1942, 199)	(237, 1944, 197)	(237, 1918, 209)
	(237, 1921, 205)	(237, 1929, 201)	(237, 1926, 204)	(237, 1914, 212)	(238, 1951, 196)	(239, 1977, 192)	(239, 1980, 191)
	(239, 1927, 201)	(239, 1929, 200)	(239, 1920, 205)	(239, 1910, 215)	(240, 1918, 205)	(240, 1916, 206)	(240, 1941, 199)
	(241, 1979, 191)	(241, 1926, 203)	(241, 1924, 204)	(241, 1910, 210)	(242, 2006, 190)	(242, 1923, 204)	(243, 1926, 201)
	(243, 1915, 205)	(243, 1921, 204)	(243, 1939, 199)	(243, 1900, 210)	(244, 1974, 193)	(244, 1897, 214)	(244, 1910, 208)
	(244, 2004, 189)	(244, 1914, 207)	(244, 1909, 209)	(245, 1894, 226)	(245, 1918, 204)	(245, 1923, 203)	(245, 1993, 190)
	(246, 1892, 210)	(246, 1890, 220)	(246, 1941, 198)	(246, 1911, 205)	(246, 1917, 204)	(246, 1909, 207)	(246, 1907, 208)
	(247, 1889, 230)	(248, 1891, 212)	(248, 1890, 213)	(248, 1884, 229)	(248, 2000, 189)	(248, 1899, 209)	(249, 1888, 215)
	(249, 1885, 225)	(250, 1974, 191)	(250, 1883, 230)	(250, 1884, 220)	(250, 1938, 199)	(250, 1910, 205)	(251, 1886, 218)
	(251, 1890, 212)	(252, 1887, 212)	(252, 1882, 240)	(252, 1908, 205)	(253, 1898, 209)	(253, 1884, 218)	(253, 1878, 220)
	(253, 1886, 215)	(253, 1885, 216)	(255, 1884, 215)	(255, 1897, 209)	(255, 1891, 210)	(255, 1882, 218)	(256, 1876, 220)
	(256, 1875, 227)	(256, 1881, 218)	(257, 1880, 218)	(257, 1872, 227)	(258, 1882, 215)	(258, 1872, 225)	(258, 1875, 220)
	(258, 1871, 227)	(259, 1870, 227)	(260, 1869, 227)	(261, 1881, 215)	(261, 1906, 208)	(261, 1879, 218)	(261, 1866, 234)
	(262, 1867, 230)	(262, 1865, 234)	(262, 1868, 229)	(262, 1873, 220)	(262, 1870, 225)	(263, 1865, 230)	(263, 1868, 227)
	(264, 1864, 240)	(264, 1866, 227)	(264, 1868, 225)	(265, 1863, 230)	(266, 1865, 225)	(266, 1871, 220)	(266, 1861, 244)
	(268, 1870, 224)	(268, 1863, 227)	(268, 1861, 230)	(269, 1879, 216)	(269, 1861, 227)	(269, 1863, 226)	(270, 1860, 231)
	(270, 1877, 218)	(271, 1869, 221)	(271, 1868, 224)	(271, 1855, 240)	(271, 1856, 236)	(271, 1859, 231)	(271, 1860, 230)
	(272, 1858, 230)	(273, 1905, 208)	(273, 1863, 225)	(275, 1885, 214)	(275, 1866, 224)	(275, 1875, 218)	(278, 1860, 228)
	(278, 1857, 230)	(279, 1858, 228)	(279, 1869, 220)	(279, 1856, 234)	(281, 1855, 230)	(281, 1853, 236)	(282, 1885, 212)
	(285, 1877, 216)	(287, 1879, 215)	(288, 1850, 260)	(308, 1847, 290)			

References

[1] Garey MR, Johnson DS, Sethi R. The complexity of flow shop and job shop scheduling. Math Oper Res 1976;1(2):117–29.

[2] Błażewicz J, Domschke W, Pesch E. The job shop scheduling problem: conventional and new solution techniques. Eur J Oper Res 1996;93:1–33.

[3] Chiang TC, Lin HJ. A simple and effective evolutionary algorithm for multi-objective flexible job shop scheduling. Int J Prod Econ 2013;141(1):87–98.

[4] Brandimarte P. Routing and scheduling in a flexible job shop by tabu search. Ann Oper Res 1993;41(3):157–83.

[5] Kacem I, Hammadi S, Borne P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE Trans Syst Man Cybernet, Part C 2002;32(1):408–19.

[6] Nowicki E, Smutnicki C. A fast taboo search algorithm for the job shop problem. Manage Sci 1996;42(6):797–813.

[7] Brucker P, Schlie R. Job-shop scheduling with multi-purpose machines. Computing 1990;45(4):369–75.

[8] Hurink J, Jurisch B, Thole M. Tabu search for the job-shop scheduling problem with multi-purpose machines. OR Spektrum 1994;15(4):205–15.

[9] Mastrolilli M, Gambardella LM. Effective neighborhood functions for the flexible job shop problem. J Sched 2000;3(1):3–20.

[10] Kacem I, Hammadi S, Borne P. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. Math Comput Simul 2002;60(3–5):245–76.

[11] Ho NB, Tay JC. GENACE: an efficient cultural algorithm for solving the flexible job-shop problem. In: Proceedings of the IEEE congress on evolutionary computation, vol. 2; 2004;. p. 1759–66.

[12] Pezzella F, Morganti G, Ciaschetti G. A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 2008;35(10):3202–12.

[13] Ho NB, Tay JC, Lai EM-K. An effective architecture for learning and evolving flexible job-shop schedules. Eur J Oper Res 2007;179(2):316–33.

[14] Gao J, Sun L, Gen M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput Oper Res 2008;35(9):2892–907.

[15] Zhang GH, Gao L, Shi Y. An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Syst Appl 2011;38(4):3563–73.

[16] Wang L, Wang S, Xu Y, Zhou G, Liu M. A bi-population based estimation of distribution algorithm for the flexible job-shop scheduling problem. Comput Ind Eng 2012;62(4):917–26.

[17] Zhang GH, Shao XY, Li PG, Gao L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. Comput Ind Eng 2009;56(4):1309–18.

[18] Moslehi G, Mahnam MA. Pareto approach to multi-objective flexible job-shop scheduling problem using particle swarm optimization and local search. Int J Prod Econ 2011;129(1):14–22.

[19] Bagheri A, Zandieh M, Mahdavi I, Yazdani M. An artificial immune algorithm for the flexible job-shop scheduling problem. Future Gener Comput Syst 2010;26(4):533–41.

[20] Xia WJ, Wu ZM. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 2005;48(2):409–25.

[21] Xing LN, Chen YW, Yang KW. Multi-objective flexible job shop schedule: design and evaluation by simulation modeling. Appl Soft Comput 2009;9(1):362–76.

[22] BenHmida A, Haouari M, Huguet MJ, Pierre Lopez. . Discrepancy search for the flexible job shop scheduling problem. Comput Oper Res 2010;37:2192–201.

[23] Li JQ, Pan QK, Xie SX. An effective shuffled frog-leaping algorithm for multi-objective flexible job shop scheduling problems. Appl Math Comput 2012;218 (18):9353–71.

[24] Teekeng W, Thammano AA. Combination of shuffled frog leaping and fuzzy logic for flexible job-shop scheduling problems. Proc Comput Sci 2011;6:69–75.

[25] Yuan Y, Xu H, Yang JD. A hybrid harmony search algorithm for the flexible job shop scheduling problem. Appl Soft Comput 2013;13(7):3259–72.

[26] Yazdani M, Amiri M, Zandieh M. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Expert Syst Appl 2010;37(1):678–87.

[27] Bożejko W, Uchroński M, Wodecki M. Parallel hybrid metaheuristics for the flexible job shop problem. Comput Ind Eng 2010;59(2):323–33.

[28] Ho NB, Tay JC. Solving multiple-objective flexible job shop problems by evolution and local search. IEEE Trans Syst Man Cybernet, Part C 2008;38 (5):674–85.

[29] Deb K, Pratap A, Agarwal S, TA Meyarivan. Fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evolut Comput 2002;6(2):182–97.
[30] Wang XJ, Gao L, Zhang CY, Shao XY. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling problem. Int J Adv Manufact Technol 2010;51(5–8):757–67.
[31] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE Trans Evolut Comput 1999;3(4):257–71.
[32] Li JQ, Pan QK, Gao KZ. Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. Int J Adv Manufact Technol 2011;55(9–12):1159–69.
[33] Wang L, Zhou G, Xu Y, Liu M. An enhanced Pareto-based artificial bee colony algorithm for the multi-objective flexible job-shop scheduling. Int J Adv Manufact Technol 2012;60(9–12):1111–23.
[34] Jaeggi DM, Parks GT, Kipouros T, Clarkson PJ. The development of a multi-objective Tabu search algorithm for continuous optimisation problems. Eur J Oper Res 2008;185(3):1192–212.
[35] Hmida AB, Haouari M, Huguet MJ, Pierre Lopez. Discrepancy search for the flexible job shop scheduling problem. Comput Oper Res 2010;37(12):2192–201.
[36] Li JQ, Pan QK, Liang YC. An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. Comput Ind Eng 2010;59(4):647–62.
[37] Dauzère-Pérès S, Paulli J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann Oper Res 1997;70(0):281–306.