Hindawi
Complexity
Volume 2022, Article ID 3487355, 17 pages
https://doi.org/10.1155/2022/3487355

WILEY | Hindawi

Research Article

A Two-Level Metaheuristic for the Job-Shop Scheduling Problem with Multipurpose Machines

Pisut Pongchairerks

Thai-Nichi International College, Thai-Nichi Institute of Technology, Bangkok 10250, Thailand

Correspondence should be addressed to Pisut Pongchairerks; pisut@tni.ac.th

Received 21 November 2021; Revised 1 January 2022; Accepted 18 January 2022; Published 1 June 2022

Academic Editor: Yu Zhou

This paper proposes a two-level metaheuristic consisting of lower- and upper-level algorithms for the job-shop scheduling problem with multipurpose machines. The lower-level algorithm is a local search algorithm used for finding an optimal solution. The upper-level algorithm is a population-based metaheuristic used to control the lower-level algorithm's input parameters. With the upper-level algorithm, the lower-level algorithm can reach its best performance on every problem instance. Most changes of the proposed two-level metaheuristic from its original variants are in the lower-level algorithm. A main purpose of these changes is to increase diversity into solution neighborhood structures. One of the changes is that the neighbor operators of the proposed lower-level algorithm are developed to be more adjustable. Another change is that the roulette-wheel technique is applied for selecting a neighbor operator and for generating a perturbation operator. In addition, the proposed lower-level algorithm uses an adjustable delay-time limit to select an optional machine for each operation. The performance of the proposed two-level metaheuristic was evaluated on well-known benchmark instances. The evaluation's results indicated that the proposed two-level metaheuristic performs well on most benchmark instances.

1. Introduction

The job-shop scheduling problem (JSP) is a well-known NP-hard optimization problem [1–3]. JSP involves scheduling jobs onto machines in order to minimize *makespan*, i.e., the schedule's length. Each job consists of a number of operations, where each operation must be processed on a predetermined machine with a predetermined processing time. To complete each job, all of its operations must be processed in the sequence from the first to the last operations. JSP has many variants and related problems, such as [4–7]. One of the well-known variants of JSP is the job-shop scheduling problem with multipurpose machines (MPMJSP) [8–10]. MPMJSP is defined as a generalized variant of JSP. An only difference between them is that each operation in JSP has only one predetermined machine, while each operation in MPMJSP may have more than one optional machine. This difference makes MPMJSP be closer to the real-world applications in modern factories than JSP because, nowadays, most machines have been developed for multiple tasks.

In this paper, the research's objective is to develop a high-performing algorithm for MPMJSP. To do so, this paper proposes a two-level metaheuristic, based on the framework of [5, 11, 12], consisting of upper- and lower-level algorithms. The upper-level algorithm is a population-based metaheuristic that acts as a parameter controller for the lower-level algorithm. The upper-level algorithm's population consists of the parameter-value combinations of the lower-level algorithm. In a parameter-value combination, each parameter's value is iteratively changed by a sum of two changeable opposite-direction vectors. The directions of the first and the second vectors are toward and away from, respectively, the memorized best-found value. The lower-level algorithm is a local search algorithm searching for an optimal solution of the being-solved MPMJSP problem. Like other metaheuristics, the lower-level algorithm cannot perform its best on all instances with a single combination of

input-parameter values. This drawback can be overcome when its input parameters are controlled by the upper-level algorithm.

The proposed two-level metaheuristic is modified from its original variant [12], where the main differences are in their lower-level algorithms. Their lower-level algorithms both search for an optimal solution in hybrid neighborhood structures via their optional operators. Their optional neighbor operators are similarly modified from the traditional operators (i.e., swap, insert, and reverse) by limiting a distance between positions of two members selected in a solution-representing permutation. However, while the lower-level algorithm of [12] has only three levels of the distance limit, the distance limit in this paper is adjustable to any possible range. Another main difference is in their methods of generating their hybrid neighborhood structures. To generate each neighbor solution-representing permutation, the lower-level algorithm of [12] uses a given probability to select one of two neighbor operators. Instead, the proposed lower-level algorithm uses the roulette-wheel method [13] to select one from three neighbor operators. In this paper, the roulette-wheel method is also applied to select multiple optional operators for generating a perturbation operator. A purpose of using the roulette-wheel method is to diversify more on the hybridization of the neighborhood structure.

As mentioned, each operation in MPMJSP has one or more optional machines. The lower-level algorithm proposed in this paper uses the delay-time limit (δ), as its input parameter, to make a criterion of selecting a machine for each operation. This use of δ is different from the uses of δ in the other researches, e.g., [10, 11, 14–17]. While the proposed lower-level algorithm uses δ to select a machine for each operation, the other researches use δ to select an operation into a timetable. A method of generating a schedule of the proposed lower-level algorithm is briefly presented as follows: First, an appearance order of all operations in a solution-representing permutation is used as a priority order for all operations. Then, every operation is assigned one-by-one into a schedule by the given priority order. When being assigned, each operation must be processed on the machine that satisfies δ, and it must be started as early as the machine can.

The remainder of this paper is divided into five sections. Section 2 describes MPMJSP and reviews the previous researches relevant to the proposed two-level metaheuristic. Section 3 describes the proposed two-level metaheuristic in detail. Section 4 shows experiment's results of evaluating the proposed two-level metaheuristic's performance. Section 5 then analyzes and discusses the experiment's results. Finally, Section 6 concludes the research's findings.

2. Preliminaries

In this section, the description of MPMJSP is given in Section 2.1, and the review on the previous researches relevant to the proposed two-level metaheuristic is given in Section 2.2.

2.1. Description of MPMJSP. The job-shop scheduling problem with multipurpose machines (MPMJSP) is classified as a generalization of the job-shop scheduling problem (JSP). An only difference between JSP and MPMJSP is their numbers of optional machines of each operation. That is, while each operation in JSP has only one predetermined machine, each operation in MPMJSP has one or more optional machines. MPMJSP thus becomes JSP if each of its operations has only one optional machine. A similar variant of MPMJSP is the flexible job-shop scheduling problem (FJSP) [18, 19]. MPMJSP and FJSP both are the JSP's variants where each operation may have more than one optional machine. However, the processing time of an operation in FJSP may change when changing its selected optional machine, while the processing time of each operation in MPMJSP is fixed for all of its optional machines. This means MPMJSP is a specific FJSP where, for each operation, all optional machines have the same processing time.

Notation used to describe MPMJSP in this paper is defined below:

(i) Let n denote the number of all jobs in MPMJSP.

(ii) Let m denote the number of all machines in MPMJSP.

(iii) Let J_i denote the i-th job in MPMJSP, where $i = 1, 2, \ldots, n$.

(iv) Let M_j denote the j-th machine in MPMJSP, where $j = 1, 2, \ldots, m$.

(v) Let n_i denote the number of all operations of J_i.

(vi) Let O_{ik} denote the k-th operation of J_i, where $k = 1, 2, \ldots, n_i$.

(vii) Let τ_{ik} denote the processing time of O_{ik}.

(viii) Let m_{ik} denote the number of all optional machines of O_{ik}, where $i = 1, 2, \ldots, n$, and $k = 1, 2, \ldots, n_i$.

(ix) Let $E_{ikl} \in$ all optional machines of O_{ik} that appear in $\{M_1, M_2, \ldots, M_m\}$ and denote the l-th optional machine of O_{ik}, where $l = 1, 2, \ldots, m_{ik}$. In other words, E_{ikl} is equivalent to the M_j that owns the l-th lowest value of j among all optional machines of O_{ik}. For example, suppose O_{12} has three optional machines, i.e., M_2, M_4, and M_5. Thus, $E_{121} = M_2$, $E_{122} = M_4$, and $E_{123} = M_5$.

MPMJSP comes with n given jobs $(J_1, J_2, \ldots, J_n)$ and m given machines $(M_1, M_2, \ldots, M_m)$. Each job J_i consists of a sequence of n_i given operations $(O_{i1}, O_{i2}, \ldots, O_{in_i})$ as a chain of precedence constraints. To complete each job J_i, O_{i1} must be finished before O_{i2} can start; O_{i2} must be finished before O_{i3} can start, and so on. Each O_{ik} must be processed by one of $E_{ik1}, E_{ik2}, \ldots, E_{ikm_{ik}}$ with the processing time of τ_{ik}. Each machine cannot process more than one operation at a time, and it cannot be stopped during processing an operation. At the beginning (i.e., time 0), all jobs have already arrived, and all machines have not been occupied. An optimal schedule is a feasible schedule that minimizes makespan (i.e., the schedule's length).

MPMJSP was first introduced by Brucker and Schlie [8]. They also proposed a polynomial-time algorithm for MPMJSP with two jobs. MPMJSP with three jobs belongs to NP-hard problem even if the number of all machines is two [9]. Tabu-search algorithms were developed by [9] for solving three sets of MPMJSP benchmark instances, i.e., Edata, Rdata, and Vdata. Since then, these three instance sets have been commonly used for comparing results of different algorithms on MPMJSP. To date, many algorithms have been developed for solving MPMJSP and its closely related problems [10, 18–21].

2.2. Previous Relevant Researches. Iterated local search is traditionally defined as a single-solution-based metaheuristic that can search for a global optimal solution. During an exploration, it uses a neighbor operator repeatedly to find a local optimum and then uses a perturbation operator to escape the found local optimum. Note that a perturbation operator stands for an operator that generates a new initial solution by largely modifying a found local optimal solution [22]. Some untraditional iterated local search algorithms are enhanced in their performance by using multiple initial solutions [23, 24]. The iterated local search algorithms have been successful on many optimization problems, including MPMJSP and FJSP [25, 26].

In iterated local search and related algorithms, there are three operators usually used as neighbor operators and perturbation operators. These three operators are the traditional swap, insert, and reverse operators [27]. To explain the mentioned operators, let u and v be two different integers randomly generated from 1 to D, where D represents the number of all members in a solution-representing permutation. The swap operator is to swap between the two members in the u-th and the v-th positions of the permutation. The insert operator is to remove a member from the u-th position of the permutation and then insert it back at the v-th position. The reverse operator is to reverse the sequence of all members from the u-th to the v-th positions of the permutation.

A common drawback of most metaheuristics is that their performance is dependent on their parameter-value settings. To overcome such a drawback, many applications use upper-level algorithms to control parameters of their solution-searching algorithms [11, 12, 14, 15, 28–31]. Some of them, e.g., [5], require more than two levels of algorithms for very complicated problems; however, most of them require only two levels. For solving JSP, there are two two-level metaheuristics acting as adaptive iterated local search algorithms [11, 12]. In each of the two-level metaheuristics, the upper-level algorithm controls the lower-level algorithm's input parameters, while the lower-level algorithm is a local search algorithm searching for an optimal job-shop schedule.

UPLA and MUPLA are the upper-level algorithms in [11, 12], respectively; they both are population-based algorithms searching in real-number search spaces. In each of them, the population is a number of the parameter-value combinations of the lower-level algorithm. In a parameter-value combination, each parameter's value is iteratively changed by a sum of two changeable opposite-direction vectors. The first vector's and the second vector's directions are toward and away from, respectively, the memorized best-found value. In only MUPLA, each parameter-value combination includes a different start operation-based permutation; thus, the two-level metaheuristic of [12] has a multistart property.

LOLA, the lower-level algorithm in [11], is a local search algorithm exploring in a solution space of parameterized-active schedules (i.e., hybrid schedules [16]). Its input parameters (i.e., a delay-time limit, a scheduling direction, a perturbation operator, and a neighbor operator) are controlled by UPLA, its upper-level algorithm. Because the delay-time limit (δ) is one of the input parameters controlled, UPLA then can control the solution space's size of parameterized-active schedules. Such a control of δ follows in the successes of the two-level PSOs of [14, 15]. Other techniques of controlling δ can also be found in literature. For example, the value of δ in [10] is dependent on the number of jobs and the number of machines, while the value of δ in [17] is dependent on the algorithm's iteration index. In addition, the PSO in [32] controls the value of δ by using the concept of self-adaptive parameter control [28].

LOSAP, the lower-level algorithm in [12], is a local search algorithm searching in a probabilistic-based hybrid neighborhood structure. By a given probability, LOSAP randomly uses one from two predetermined operators to generate a neighbor solution-representing permutation. In other words, based on the given probability, LOSAP can switch between the two given operators anytime during its exploration. While the search performance of LOLA [11] is mainly based on its special solution space, that of LOSAP is mainly based on its hybrid neighborhood structure. LOSAP has multiple optional operators for its perturbation and neighbor operators. These optional operators are modified from the traditional operators by limiting the distance of v from u. For generating v in LOSAP, there are three optional distance-limit levels: $[u-4, u+4]$, $[u-(D/5), u+(D/5)]$, and $[1, D]$.

3. Proposed Two-Level Metaheuristic

For solving MPMJSP, this paper proposes the two-level metaheuristic consisting of the lower- and upper-level algorithms. In this section, MPM-LOLA and MPM-UPLA represent the lower-level algorithm and the upper-level algorithm, respectively. The description of MPM-LOLA is given in Section 3.1, and the description of MPM-UPLA is given in Section 3.2.

3.1. MPM-LOLA. MPM-LOLA, as a variant of LOLA [11] and LOSAP [12], is a local search algorithm exploring in a hybrid neighborhood structure. Similar to LOLA and LOSAP, MPM-LOLA generates its neighborhood structure by using multiple optional operators. However, there are many changes of MPM-LOLA from its older variants. Although MPM-LOLA uses the delay-time limit (δ) like LOLA does, it uses δ in different way and purpose. While LOLA

uses δ to select an operation, MPM-LOLA uses δ to select an optional machine for each operation. In the remaining other parts, MPM-LOLA is more similar to LOSAP than LOLA. The major changes of MPM-LOLA from LOSAP are summarized below:

(i) While the LOSAP's solution-decoding method generates JSP's solutions, the MPM-LOLA's solution-decoding method generates MPMJSP's solutions.

(ii) MPM-LOLA is similar to LOSAP in that its optional operators are modified from the traditional operators by limiting the distance of v from u. However, while LOSAP has only three levels of distance limit, the distance limit of MPM-LOLA is adjustable to any possible range.

(iii) Unlike LOSAP, MPM-LOLA applies the roulette-wheel method to select a neighbor operator. It also applies the roulette-wheel method to select optional operators for generating a perturbation operator.

For the purpose of clarification, the description of MPM-LOLA is divided into two parts: a description of its solution-decoding method and a description of its overall procedure. The solution-decoding method is described in Section 3.1.1, and the overall procedure is described in Section 3.1.2.

3.1.1. Solution-Decoding Method. MPM-LOLA decodes a solution-representing permutation into a schedule by using the delay-time limit (δ) and the tiebreak criterion (TB). Note that TB is used only if there is more than one optional machine that satisfies δ. In this paper, every solution-representing permutation is in a form of the operation-based permutation [33, 34]. An operation-based permutation is a permutation of numbers $1, 2, \ldots, n$ where the number i ($i = 1, 2, \ldots, n$) appears n_i times. Remind that n and n_i denote the number of all jobs and the number of all operations of the job J_i, respectively. In the permutation, the number i in its k-th appearance represents the operation O_{ik}. Then, a schedule is constructed by scheduling all operations one-by-one in the order given by the permutation. Each operation must be processed by its optional machine that satisfies δ and TB, and it must be started as early as this machine can. It is noticed that the use of δ in this paper is different from those in the other researches, e.g., [10, 11, 14–16]. While MPM-LOLA uses δ to select an optional machine for each operation, the other researches use δ to select an operation into the timetable.

As mentioned above, $\delta \in [0, 1)$ and $TB \in \{\text{lowest, highest}\}$ are used to select an optional machine for each operation. If $\delta = 0$, each operation must be processed on its optional machine that can start processing earliest. When the value of δ is assigned larger, the maximum delay-time allowed for each operation is then longer; consequently, it may increase the number of optional machines that satisfy δ for each operation. If there is more than one optional machine that satisfies δ, then TB is required as a tie breaker. If TB is selected to be lowest, the lowest-indexed optional machine that satisfies δ is selected; otherwise, the highest-indexed optional machine that satisfies δ is selected.

Algorithm 1 presents the solution-decoding method used by MPM-LOLA. The algorithm uses δ and TB, as its input parameters, to transform an operation-based permutation into an MPMJSP's solution. Note that Algorithm 1 may return a different schedule from the same operation-based permutation if the values of δ and TB are changed. Notation used in Algorithm 1 is defined below:

(i) Let D denote the number of all operations in the being-solved MPMJSP instance. Thus, $D = n_1 + n_2 + \ldots + n_n$, where n_i is the number of all operations of the job J_i ($i = 1, 2, \ldots, n$).

(ii) Let Π denote the sequence of operations transformed from the operation-based permutation.

(iii) Let Φ denote the schedule transformed from Π.

(iv) Let $O_{i'k'}$ denote the k'-th operation of the job $J_{i'}$, and it represents the as-yet-unscheduled operation that is currently in its turn to be scheduled.

(v) Let $m_{i'k'}$ denote the number of all optional machines of $O_{i'k'}$.

(vi) Let $E_{i'k'l}$ denote the l-th optional machine of $O_{i'k'}$, where $l = 1, 2, \ldots, m_{i'k'}$.

(vii) Let E denote the chosen machine for processing $O_{i'k'}$. This machine must be chosen from all $E_{i'k'l}$ ($l = 1, 2, \ldots, m_{i'k'}$).

(viii) Let δ be a real number within $[0, 1)$ and denote the delay-time limit.

(ix) Let $TB \in \{\text{lowest, highest}\}$ denote the tiebreak criterion for selecting an optional machine to process $O_{i'k'}$. It is used only if there is more than one optional machine that satisfies δ. If $TB = \text{lowest}$, let E be the machine with the lowest l from all $E_{i'k'l}$ (where $l = 1, 2, \ldots, m_{i'k'}$) that satisfy δ. If $TB = \text{highest}$, let E be the machine with the highest l from all $E_{i'k'l}$ (where $l = 1, 2, \ldots, m_{i'k'}$) that satisfy δ.

(x) Let $\tau_{i'k'}$ denote the processing time of $O_{i'k'}$.

(xi) Let σ_M denote the minimum of the earliest available times of all optional machines of $O_{i'k'}$.

(xii) Let σ_J denote the earliest possible start time of $O_{i'k'}$ in the job $J_{i'}$. This means σ_J is equal to the finished time of $O_{i'k'-1}$. If $O_{i'k'}$ has no immediate-preceding operation, then σ_J is equal to 0.

(xiii) Let σ denote the earliest possible start time of $O_{i'k'}$. It is equal to the maximum between σ_M and σ_J.

3.1.2. Procedure of MPM-LOLA. MPM-LOLA generates its neighborhood structure based on multiple optional operators. These optional operators consist of d-swap, d-insert, d-reverse, D-swap, D-insert, and D-reverse operators. Note that D-swap, D-insert, and D-reverse are identical to the traditional swap, insert, and reverse operators, respectively. Notation and terminologies used to define the mentioned operators are given below:

Step 1. Receive a δ's value, a *TB*'s value, and an operation-based permutation needed to be transformed from MPM-LOLA (Algorithm 2).
Step 2. Transform the operation-based permutation taken from Step 1 into Π by changing the number i in its k-th appearance into the operation O_{ik} ($i = 1, 2, \ldots, n$; $k = 1, 2, \ldots, n_i$). For example, the operation-based permutation (3, 2, 3, 1, 1, 2) is transformed into $\Pi = (O_{31}, O_{21}, O_{32}, O_{11}, O_{12}, O_{22})$.
Step 3. Transform Π into Φ by using Steps 3.1 to 3.6.
Step 3.1. Let $\Phi \longleftarrow$ an empty schedule, and let $t \longleftarrow 1$.
Step 3.2. Let $O_{i'k'} \longleftarrow$ the leftmost as-yet-unscheduled operation in Π.
Step 3.3. Find σ_M, σ_j, and σ of $O_{i'k'}$.
Step 3.4. Let $E \longleftarrow$ the machine, chosen from all $E_{i'k'l}$ ($l = 1, 2, \ldots, m_{i'k'}$), that can start processing not-later-than $\sigma + \delta\tau_{i'k'}$. If there is more than one machine that can be chosen as E, then choose one of them that has the lowest l if TB = lowest; otherwise, choose one of them that has the highest l.
Step 3.5. Modify Φ by assigning E to process $O_{i'k'}$. In the schedule, let E start processing $O_{i'k'}$ as early as possible.
Step 3.6. If $t < D$, then $t \longleftarrow t + 1$ and repeat from Step 3.2. Otherwise, go to Step 4.
Step 4. Return Φ as the complete schedule to MPM-LOLA (Algorithm 2).

Algorithm 1: Solution-decoding method.

(i) Let u and v be integers used to point at the two member positions in an operation-based permutation.

(ii) Let d denote a distance limit of v from u. It is used for specifying d-swap, d-insert, and d-reverse.

(iii) Let D denote the number of all members in the solution-representing permutation, which equals the number of all operations in the MPMJSP instance. Thus, $D = n_1 + n_2 + \ldots + n_n$, where n_i is the number of all operations of the job J_i.

(iv) d-swap is to swap two members in the u-th and the v-th positions in the permutation. Let u be randomly selected from 1 to D. Then, let v be randomly selected from $\max(1, u - d)$ to $\min(u + d, D)$ except u.

(v) d-insert is to remove a member from the u-th position in the permutation and then insert it into the v-th position. Let u be randomly selected from 1 to D. Then, let v be randomly selected from $\max(1, u - d)$ to $\min(u + d, D)$ except u.

(vi) d-reverse is to reverse the positions of all members from the u-th to the v-th positions in the permutation. Let u be randomly selected from 1 to D. Then, let v be randomly selected from $\max(1, u - d)$ to $\min(u + d, D)$ except u.

(vii) D-swap is to swap two members in the u-th and the v-th positions in the permutation. Let u and v be two different integers randomly selected from 1 to D.

(viii) D-insert is to remove a member from the u-th position in the permutation and then insert it into the v-th position. Let u and v be two different integers randomly selected from 1 to D.

(ix) D-reverse is to reverse the positions of all members from the u-th to the v-th positions in the permutation. Let u and v be two different integers randomly selected from 1 to D.

Before an execution, eight MPM-LOLA's input parameters must be assigned values. The first input parameter, denoted by P, is the start operation-based permutation. The remaining seven input parameters consist of two parameters for specifying a perturbation operator, three parameters for generating its neighbor operators, and two parameters for selecting an optional machine. In MPM-LOLA, the perturbation operator is to randomly use one from D-swap, D-insert, and D-reverse on the start operation-based permutation n times (remind that n denotes the number of all jobs). In each of these n random selections, the roulette-wheel technique is applied to select one from D-swap, D-insert, and D-reverse. The probabilities of selecting D-swap and D-insert in the roulette wheel are the second and the third input parameters of MPM-LOLA, respectively. Consequently, the probability of selecting D-reverse is unity subtracted by the sum of the probabilities of selecting D-swap and D-insert.

The fourth to the sixth input parameters are used to generate MPM-LOLA's neighbor operators (i.e., d-swap, d-insert, and d-reverse). The fourth input parameter, denoted by d, is the distance limit of v from u for specifying d-swap, d-insert, and d-reverse. Then, MPM-LOLA uses the roulette-wheel technique to randomly select one from d-swap, d-insert, and d-reverse to generate a neighbor solution-representing permutation. In the roulette wheel, the probabilities of selecting d-swap and d-insert are the fifth and the sixth input parameters, respectively. The probability of selecting d-reverse is thus unity subtracted by the sum of the probabilities of selecting d-swap and d-insert.

The delay-time limit (δ) and the tiebreak criterion (TB) are the MPM-LOLA's seventh and eighth input parameters, respectively. These two input parameters are used to select an optional machine for each operation in constructing a schedule. Thus, instead of using δ and TB by MPM-LOLA itself, MPM-LOLA transfers the values of δ and TB into its solution-decoding method (Algorithm 1).

The overall procedure of MPM-LOLA is given in Algorithm 2. Notation used in Algorithm 2 is defined below:

Step 1. Receive values of its eight input parameters (i.e., P, ρ_S, ρ_I, d, ρ_{NS}, ρ_{NI}, δ, and TB) from MPM-UPLA (Algorithm 3).
Step 2. Generate P_0 from P by using Steps 2.1 to 2.4.
  Step 2.1. Let $r \longleftarrow 1$.
  Step 2.2. Randomly generate $p \sim U[0, 1)$.
  Step 2.3. Modify P by using D-swap if $p < \rho_S$, D-insert if $\rho_S \leq p < \rho_S + \rho_I$, and D-reverse otherwise.
  Step 2.4. If $r < n$, let $r \longleftarrow r + 1$ and repeat from Step 2.2. Otherwise, let $P_0 \longleftarrow P$ and go to Step 3.
Step 3. Execute Algorithm 1, with the taken values of δ and TB, for transforming P_0 into S_0.
Step 4. Find a local optimal schedule by using Steps 4.1 to 4.5.
  Step 4.1. Let $t_L \longleftarrow 0$.
  Step 4.2. Randomly generate $p \sim U[0, 1)$.
  Step 4.3. Generate P_1 from P_0 by using d-swap if $p < \rho_{NS}$, d-insert if $\rho_{NS} \leq p < \rho_{NS} + \rho_{NI}$, and d-reverse otherwise.
  Step 4.4. Execute Algorithm 1, with the taken values of δ and TB, for transforming P_1 into S_1.
  Step 4.5. Update P_0, S_0, and t_L by using Steps 4.5.1 to 4.5.3.
    Step 4.5.1. If $Makespan(S_1) < Makespan(S_0)$, then let $P_0 \longleftarrow P_1$ and $S_0 \longleftarrow S_1$, and repeat from Step 4.1.
    Step 4.5.2. If $Makespan(S_1) = Makespan(S_0)$, then let $P_0 \longleftarrow P_1$ and $S_0 \longleftarrow S_1$, and repeat from Step 4.2.
    Step 4.5.3. If $Makespan(S_1) > Makespan(S_0)$, then let $t_L \longleftarrow t_L + 1$. If $t_L < D^2$, repeat from Step 4.2; otherwise, go to Step 5.
Step 5. Return P_0 and S_0 as the final (best-found) operation-based permutation and the final (best-found) schedule, respectively, to MPM-UPLA (Algorithm 3).

Algorithm 2: MPM-LOLA's procedure.

(i) Let D denote the number of all operations in the being-solved MPMJSP instance. Thus, $D = n_1 + n_2 + \ldots + n_n$, where n_i is the number of all operations of the job J_i.

(ii) Let d denote the distance limit of v from u in the operation-based permutation for specifying d-swap, d-insert, and d-reverse.

(iii) Let ρ_S and ρ_I, which are real numbers within [0, 1), denote the probabilities of selecting D-swap and D-insert, respectively. Thus, the probability of selecting D-reverse is $1 - \rho_S - \rho_I$.

(iv) Let ρ_{NS} and ρ_{NI}, which are real numbers within [0, 1), denote the probabilities of selecting d-swap and d-insert, respectively. Thus, the probability of selecting d-reverse is $1 - \rho_{NS} - \rho_{NI}$.

(v) Let δ, which is a real number within [0, 1), denote the delay-time limit for selecting an optional machine for each operation.

(vi) Let $TB \in \{\text{lowest, highest}\}$ denote the tiebreak criterion for selecting an optional machine for each operation.

(vii) Let P denote the start operation-based permutation.

(viii) Let P_0 denote the current best-found operation-based permutation. An initial P_0 is generated from P via the perturbation operator.

(ix) Let S_0, which is decoded from P_0, denote the current best-found schedule. In addition, $Makespan(S_0)$ stands for the makespan of S_0.

(x) Let P_1 denote the current neighbor operation-based permutation.

(xi) Let S_1, which is decoded from P_1, denote the current neighbor schedule. In addition, $Makespan(S_1)$ stands for the makespan of S_1.

3.2. MPM-UPLA. MPM-UPLA is an upper-level algorithm of the proposed two-level metaheuristic. It uses the same framework of the upper-level algorithms of [11, 12]. MPM-UPLA is thus a population-based search algorithm that acts as a parameter controller. It evolves the MPM-LOLA's input-parameter values, so that MPM-LOLA can return its best performance on every single MPMJSP instance. At the t-th iteration, the MPM-UPLA's population consists of N combinations of the MPM-LOLA's input-parameter values, i.e., $C_1(t)$, $C_2(t)$, ..., $C_N(t)$. In short, let a *parameter-value combination* stand for a combination of the MPM-LOLA's input-parameter values. Let $C_g(t) \equiv (c_{1g}(t), c_{2g}(t), \ldots, c_{9g}(t))$ denote the g-th parameter-value combination (where $g = 1, 2, \ldots, N$) in the population at the t-th iteration. It represents the value combination of the MPM-LOLA's input parameters, i.e., P, ρ_S, ρ_I, d, ρ_{NS}, ρ_{NI}, and TB.

The delay-time limit, δ, is an important MPM-LOLA's input parameter controlled by MPM-UPLA. However, the value of δ is not assigned as a member in $C_g(t)$, but it is controlled by the MPM-UPLA's iteration index, t. At the first MPM-UPLA's iteration ($t = 1$), the value of δ is set to be 0.0 for MPM-LOLA. For every next 50 MPM-UPLA's iterations, the value of δ is increased by 0.2 for MPM-LOLA. This setting of δ was based on the result of a preliminary study of this research. It found that the control of δ using the MPM-UPLA's iteration index usually performs better than the control of δ using $C_g(t)$.

The transformations from the $C_g(t)$'s members into MPM-LOLA's parameter values are described below:

(i) Let $c_{1g}(t)$ represent P. In other words, P is directly equal to $c_{1g}(t)$ in the transformation.

(ii) Let $c_{2g}(t)$, $c_{3g}(t)$, and $c_{4g}(t) \in \mathbb{R}$ be used together to determine the values of ρ_S and ρ_I. Their transformations are given in (1) and (2).

$$\rho_S = \frac{\max\left(0, c_{2g}(t)\right)}{\max\left(0, c_{2g}(t)\right) + \max\left(0, c_{3g}(t)\right) + \max\left(0, c_{4g}(t)\right)}, \quad (1)$$

$$\rho_I = \frac{\max\left(0, c_{3g}(t)\right)}{\max\left(0, c_{2g}(t)\right) + \max\left(0, c_{3g}(t)\right) + \max\left(0, c_{4g}(t)\right)}. \quad (2)$$

(iii) Let $c_{5g}(t) \in \mathbb{R}$ be used to determine the value of d. In its transformation, let d be equal to the rounded integer from $1 + Dc_{5g}(t)$, where D is the number of all operations in the MPMJSP instance. After that, reassign $d = 1$ if $d < 1$, and reassign $d = D$ if $d > D$.

(iv) Let $c_{6g}(t)$, $c_{7g}(t)$, and $c_{8g}(t) \in \mathbb{R}$ be used together to determine the values of ρ_{NS} and ρ_{NI}. Their transformations are given in (3) and (4).

$$\rho_{NS} = \frac{\max\left(0, c_{6g}(t)\right)}{\max\left(0, c_{6g}(t)\right) + \max\left(0, c_{7g}(t)\right) + \max\left(0, c_{8g}(t)\right)}, \quad (3)$$

$$\rho_{NI} = \frac{\max\left(0, c_{7g}(t)\right)}{\max\left(0, c_{6g}(t)\right) + \max\left(0, c_{7g}(t)\right) + \max\left(0, c_{8g}(t)\right)}. \quad (4)$$

(v) Let $c_{9g}(t) \in \mathbb{R}$ be used to determine the value of TB. In its transformation, let TB = lowest if $c_{9g}(t) < 0.5$, and let TB = highest if $c_{9g}(t) \geq 0.5$.

The overall procedure of MPM-UPLA is given in Algorithm 3. Notation used in Algorithm 3 is defined below:

(i) Let N denote the number of all parameter-value combinations in the MPM-UPLA's population.

(ii) Let $C_g(t) \equiv (c_{1g}(t), c_{2g}(t), \ldots, c_{9g}(t))$ denote the g-th parameter-value combination (where $g = 1, 2, \ldots, N$) in the MPM-UPLA's population at the t-th iteration.

(iii) Let $Score(C_g(t))$ denote the performance score of $C_g(t)$. Note that the lower the performance score, the better the performance.

(iv) After executing MPM-LOLA with the parameter values given by $C_g(t)$, let its final (best-found) operation-based permutation and final (best-found) schedule be denoted by $P_{fg}(t)$ and $S_{fg}(t)$, respectively.

(v) Let $Makespan(S_{fg}(t))$ denote the makespan of $S_{fg}(t)$.

(vi) Let $C_{\text{best}} \equiv (c_{1\text{best}}, c_{2\text{best}}, \ldots, c_{9\text{best}})$ denote the best parameter-value combination ever found by the population. In addition, let $Score(C_{\text{best}})$ denote the performance score of C_{best}.

(vii) Let S_{best} denote the best schedule ever found by the population.

4. Experiment's Results

In this paper, an experiment was conducted to compare MPM-UPLA's results with those of TS, PSO, and CP. Let TS, PSO, and CP represent the tabu-search algorithm [9], the particle swarm optimization algorithm [10], and the ILOG constraint programming optimizer [20], respectively. These three algorithms were chosen because they perform well on the same benchmark instance sets used in this paper's experiment. In Sections 4 and 5, let MPM-UPLA stand for the whole two-level metaheuristic, i.e., MPM-UPLA combined with MPM-LOLA. The reason is that MPM-UPLA uses MPM-LOLA as its component when solving MPMJSP.

In the comparison, this paper's experiment used three benchmark instance sets, i.e., Edata, Rdata, and Vdata, taken from [9, 20, 21]. Each instance set consists of 66 instances, modified from the well-known JSP benchmark instances [35–39]. The difference among the three instance sets is the number of optional machines of each operation in their instances. In Edata, the average number of optional machines of each operation is 1.15, and the maximum number of optional machines of each operation is 2 or 3. In Rdata, the average number and the maximum number of optional machines of each operation are 2 and 3, respectively. In Vdata, the average number and the maximum number of optional machines of each operation are $0.5m$ and $0.8m$, respectively (where m = the number of all machines in an instance).

The parameter settings of MPM-UPLA used in the experiment are shown below:

(i) The population of MPM-UPLA consisted of three parameter-value combinations (i.e., $N = 3$ in Algorithm 3).

(ii) The stopping criterion of MPM-UPLA was to stop when any of the below conditions was satisfied:

(a) The 1,000-th iteration (i.e., $t = 1{,}000$ in Algorithm 3) was reached.

(b) The 150-th minute of computational time was reached.

(c) The known optimal solution was found. If the optimal solution has been yet unknown, its lower bound [20, 21] was used instead.

(iii) MPM-UPLA was coded in C# and executed on an Intel® Core™ i7-3520M @ 2.90 GHz with RAM of 4 GB (3.87 GB useable).

(iv) For each instance, MPM-UPLA was executed five trials with different random-seed numbers.

With the above settings, the experiment's results on Edata, Rdata, and Vdata are presented in Tables 1 to 3, respectively. These tables first show the name, size, and best-known solution value of each instance. The best-known solution value, given by literature, stands for the upper bound of the optimal solution value. For each instance, each table then shows the best-found solution values of TS [9], PSO [10], CP [20], and MPM-UPLA. The best-found solution values of TS, PSO, and CP are their best solution values taken from their original articles [9, 10] and [20], respectively. For MPM-UPLA, its best-found solution value on each instance is a minimum of the best-found solution values from its five trials in this experiment. Each table also shows an average of the best-found solution values, the average number of used iterations, and the average computational time from the MPM-UPLA's five trials on each instance.

Step 1. Receive a value of N and a stopping criterion from a user. Let $t \longleftarrow 1$, $\delta \longleftarrow 0.0$, and $Score(C_{\text{best}}) \longleftarrow +\infty$.
Step 2. Generate $C_g(t) \equiv (c_{1g}(t), c_{2g}(t), \ldots, c_{9g}(t))$, where $g = 1, 2, \ldots, N$, by using Steps 2.1 to 2.4.
Step 2.1. Let $g \longleftarrow 1$.
Step 2.2. Randomly generate $c_{1g}(t)$ from any possible operation-based permutation.
Step 2.3. Randomly generate $c_{2g}(t), c_{3g}(t), \ldots, c_{9g}(t) \sim U[0, 1)$.
Step 2.4. If $g < N$, let $g \longleftarrow g + 1$ and repeat from Step 2.2. Otherwise, go to Step 3.
Step 3. Evaluate $Score(C_g(t))$, and update C_{best} and S_{best} by using Steps 3.1 to 3.6.
Step 3.1. Let $g \longleftarrow 1$.
Step 3.2. Transform $C_g(t)$ into the values of P, ρ_S, ρ_I, d, ρ_{NS}, ρ_{NI}, and TB.
Step 3.3. Execute MPM-LOLA (Algorithm 2) with the last-updated values of P, ρ_S, ρ_I, d, ρ_{NS}, ρ_{NI}, TB, and δ in order to receive $P_{fg}(t)$ and $S_{fg}(t)$.
Step 3.4. Let $Score(C_g(t)) \longleftarrow Makespan(S_{fg}(t))$.
Step 3.5. If $Score(C_g(t)) \leq Score(C_{\text{best}})$, then let $C_{\text{best}} \longleftarrow C_g(t)$, $Score(C_{\text{best}}) \longleftarrow Score(C_g(t))$, and $S_{\text{best}} \longleftarrow S_{fg}(t)$.
Step 3.6. If $g < N$, let $g \longleftarrow g + 1$ and repeat from Step 3.2. Otherwise, go to Step 4.
Step 4. Update $C_g(t+1)$, where $g = 1, 2, \ldots, N$, by using Steps 4.1 to 4.5.
Step 4.1. Let $g \longleftarrow 1$.
Step 4.2. Let $c_{1g}(t+1) \longleftarrow P_{fg}(t)$.
Step 4.3. If $(t+1) \bmod 50 = 0$, let $\delta \longleftarrow \delta + 0.2$. After that, reassign $\delta \longleftarrow 0.999$ if $\delta \geq 1.0$.
Step 4.4. If $(t+1) \bmod 50 = 0$, randomly generate $c_{2g}(t+1), c_{3g}(t+1), \ldots, c_{9g}(t+1) \sim U[0, 1)$. Otherwise, generate $c_{qg}(t+1)$, where $q = 2, 3, \ldots, 9$, by using the below equation. Let u_1 and $u_2 \sim U[0, 1)$.

$$c_{qg}(t+1) = \begin{cases} c_{qg}(t) + 0.05u_1 - 0.01u_2 & \text{if } c_{qg}(t) < c_{q\text{best}}, \\ c_{qg}(t) - 0.05u_1 + 0.01u_2 & \text{if } c_{qg}(t) > c_{q\text{best}}, \\ c_{qg}(t) + 0.01u_1 - 0.01u_2 & \text{if } c_{qg}(t) = c_{q\text{best}}. \end{cases}$$

Step 4.5. If $g < N$, then let $g \longleftarrow g + 1$ and repeat from Step 4.2. Otherwise, go to Step 5.
Step 5. If the stopping criterion is not met, then let $t \longleftarrow t + 1$ and repeat from Step 2. Otherwise, return S_{best} as the final result to the user.

Algorithm 3: MPM-UPLA's procedure.

Notation and terminologies used for each instance in Tables 1 to 3 are given as follows:

(i) Let *Instance* column present the name of each instance.

(ii) Let n and m denote the number of all jobs and the number of all machines, respectively, in the instance.

(iii) Let a *solution* denote an MPMJSP's schedule, and let a *solution value* denote a makespan of an MPMJSP's schedule.

(iv) Let *BKS* column present the best-known solution value given by literature, e.g., [20, 21]. If the best-known solution value has been proven to be an optimal solution value, it is presented without parentheses. Otherwise, it is an upper bound of the optimal solution value and is presented within parentheses.

(v) If the best-found solution value of MPM-UPLA is better than the best-known solution value given by literature, let the best-found solution value of MPM-UPLA become the new best-known solution value. When such a case occurs, an arrow symbol ($\longrightarrow$) is given in front of the value.

(vi) Let *Best* represent the best-found solution value of each algorithm. *Best* of MPM-UPLA was taken from the five trials in this experiment. *Bests* of TS, PSO, and CP were taken from [9, 10] and [20], respectively. N/A means the best-found solution value does not appear in the original article.

(vii) Let *Avg* represent the average of the best-found solution values of the five trials of MPM-UPLA.

(viii) Let *No of Iters* and *Time* stand for the average number of iterations and the average computational time (in HH:MM:SS format), respectively, until the MPM-UPLA's stopping criterion is met.

For each instance set, Table 4 shows *Avg %BD* of each algorithm on each instance category. Of each algorithm, *%BD* of each instance denotes a percent deviation of the best-found solution value from the best-known solution value. Then, *Avg %BD* denotes an average of *%BDs* of all instances in their category. In Table 4, each instance is classified into one of 13 instance categories, based on its source and size. The details of these 13 categories are given below:

(i) M6-20 consists of three instances, i.e., M6, M10, and M20.

(ii) LA1-5 consists of five 10-job/5-machine instances, i.e., LA1, LA2, ..., LA5.

(iii) LA6-10 consists of five 15-job/5-machine instances, i.e., LA06, LA07, ..., LA10.

(iv) LA11-15 consists of five 20-job/5-machine instances, i.e., LA11, LA12, ..., LA15.

(v) LA16-20 consists of five 10-job/10-machine instances, i.e., LA16, LA17, ..., LA20.

Table 1: Results on Edata.

Instance	n, m	BKS	Best of TS [9]	Best of PSO [10]	Best of CP [20]	MPM-UPLA			
						Best	Avg	No of Iters	Time
M6	6, 6	55	57	55	55	55	55.0	1	< 1 sec.
M10	10, 10	871	917	892	877	873	875.0	1000	0:26:10
M20	20, 5	1088	1109	1116	1088	1090	1090.2	1000	0:31:11
LA1	10, 5	609	611	609	609	609	609.0	1	< 1 sec.
LA2	10, 5	655	655	655	655	655	655.0	3	< 1 sec.
LA3	10, 5	550	573	567	567	550	550.0	89	0:00:19
LA4	10, 5	568	578	582	568	568	568.0	25	0:00:05
LA5	10, 5	503	503	503	503	503	503.0	1	< 1 sec.
LA6	15, 5	833	833	833	833	833	833.0	1	< 1 sec.
LA7	15, 5	762	765	765	765	762	762.0	377	0:04:02
LA8	15, 5	845	845	845	845	845	845.0	3	0:00:01
LA9	15, 5	878	878	878	878	878	878.0	1	< 1 sec.
LA10	15, 5	866	866	866	866	866	866.0	1	< 1 sec.
LA11	20, 5	1103	1106	1103	1106	1103	1103.0	246	0:05:09
LA12	20, 5	960	960	960	960	960	960.0	1	0:00:02
LA13	20, 5	1053	1053	1053	1053	1053	1053.0	1	0:00:01
LA14	20, 5	1123	1151	1123	1123	1123	1123.0	2	0:00:02
LA15	20, 5	1111	1111	1111	1111	1111	1111.0	2	0:00:03
LA16	10, 10	892	924	893	904	892	892.0	4	0:00:07
LA17	10, 10	707	757	707	707	707	707.0	3	0:00:04
LA18	10, 10	842	864	847	843	842	842.0	250	0:06:00
LA19	10, 10	796	850	820	799	796	796.0	161	0:04:20
LA20	10, 10	857	919	859	857	857	857.0	6	0:00:11
LA21	15, 10	1009	1066	1057	1044	1021	1025.0	1000	1:42:08
LA22	15, 10	880	919	912	887	882	883.2	1000	1:39:29
LA23	15, 10	950	980	994	950	950	950.0	68	0:06:38
LA24	15, 10	908	952	939	913	909	909.2	1000	1:39:33
LA25	15, 10	936	970	974	955	945	945.2	1000	1:31:55
LA26	20, 10	1106	1169	1173	1143	1115	1123.0	543	2:30:09
LA27	20, 10	1181	1230	1247	1188	1182	1188.8	552	2:30:12
LA28	20, 10	1142	1204	1195	1153	1149	1149.0	566	2:30:08
LA29	20, 10	1107	1210	1175	1128	1124	1130.2	555	2:30:08
LA30	20, 10	(1193)	1253	1262	1241	1220	1223.4	551	2:30:19
LA31	30, 10	(1532)	1596	1620	1552	1541	1541.0	195	2:30:31
LA32	30, 10	1698	1769	1743	1698	1698	1698.0	2	0:01:19
LA33	30, 10	1547	1575	1578	1560	1547	1547.0	2	0:01:22
LA34	30, 10	1599	1627	1662	1609	1608	1608.8	259	2:30:09
LA35	30, 10	1736	1736	1736	1736	1736	1736.0	1	0:01:06
LA36	15, 15	1160	1247	1202	1160	1160	1161.8	307	1:49:22
LA37	15, 15	1397	1453	1425	1397	1397	1397.0	41	0:11:14
LA38	15, 15	1141	1185	1209	1146	1143	1148.2	424	2:30:18
LA39	15, 15	1184	1226	1220	1184	1186	1187.4	423	2:30:14
LA40	15, 15	1144	1214	1197	1174	1150	1153.6	391	2:30:05
ABZ5	10, 10	1167	N/A	N/A	1176	1167	1167.0	235	0:06:08
ABZ6	10, 10	925	N/A	N/A	925	925	925.0	35	0:00:49
ABZ7	20, 15	(610)	N/A	N/A	638	619	621.0	150	2:30:44
ABZ8	20, 15	(637)	N/A	N/A	654	648	650.0	152	2:30:37
ABZ9	20, 15	644	N/A	N/A	668	655	656.6	155	2:30:26
CAR1	11, 5	6176	N/A	N/A	6176	6176	6182.0	723	0:03:37
CAR2	13, 4	6327	N/A	N/A	6455	6432	6433.6	1000	0:05:05
CAR3	12, 5	6856	N/A	N/A	6856	6856	6875.8	780	0:04:25
CAR4	14, 4	7789	N/A	N/A	7789	7789	7789.0	1	< 1 sec.
CAR5	10, 6	7229	N/A	N/A	7229	7229	7229.0	41	0:00:15
CAR6	8, 9	7990	N/A	N/A	8478	7990	7990.0	62	0:00:33
CAR7	7, 7	6123	N/A	N/A	6123	6123	6123.0	6	0:00:01
CAR8	8, 8	7689	N/A	N/A	7689	7689	7689.0	9	0:00:04
ORB1	10, 10	977	N/A	N/A	988	977	977.0	223	0:06:25
ORB2	10, 10	865	N/A	N/A	870	865	865.0	53	0:01:21
ORB3	10, 10	951	N/A	N/A	960	952	952.0	1000	0:26:46

Table 1: Continued.

Instance	n, m	BKS	Best of TS [9]	Best of PSO [10]	Best of CP [20]	MPM-UPLA			
						Best	Avg	No of Iters	Time
ORB4	10, 10	984	N/A	N/A	1016	984	984.0	255	0:06:47
ORB5	10, 10	842	N/A	N/A	865	842	842.0	215	0:05:27
ORB6	10, 10	958	N/A	N/A	1004	958	958.0	93	0:02:48
ORB7	10, 10	387	N/A	N/A	387	389	389.0	1000	0:26:27
ORB8	10, 10	894	N/A	N/A	894	894	894.0	15	0:00:23
ORB9	10, 10	933	N/A	N/A	933	933	933.0	121	0:03:14
ORB10	10, 10	933	N/A	N/A	937	933	933.0	226	0:06:02

Table 2: Results on Rdata.

Instance	n, m	BKS	Best of TS [9]	Best of PSO [10]	Best of CP [20]	MPM-UPLA			
						Best	Avg	No of Iters	Time
M6	6, 6	47	47	47	47	47	47.0	1	< 1 sec.
M10	10, 10	686	737	724	686	686	686.0	29	0:01:10
M20	20, 5	1022	1028	1025	1024	1022	1022.0	22	0:00:46
LA1	10, 5	(571)	574	574	573	571	571.0	1000	0:04:50
LA2	10, 5	529	535	534	534	530	530.8	1000	0:04:51
LA3	10, 5	477	481	480	478	478	478.0	1000	0:05:43
LA4	10, 5	502	509	506	504	502	502.8	878	0:04:31
LA5	10, 5	457	460	459	458	457	457.6	648	0:03:03
LA6	15, 5	799	801	800	799	799	799.0	15	0:00:13
LA7	15, 5	749	752	750	750	749	749.0	158	0:02:38
LA8	15, 5	765	767	767	766	765	765.0	183	0:02:54
LA9	15, 5	853	859	854	854	853	853.0	100	0:01:29
LA10	15, 5	804	806	806	805	804	804.0	274	0:03:55
LA11	20, 5	1071	1073	1072	1072	1071	1071.0	3	0:00:07
LA12	20, 5	936	937	936	936	936	936.0	2	0:00:04
LA13	20, 5	1038	1039	1039	1038	1038	1038.0	2	0:00:03
LA14	20, 5	1070	1071	1070	1071	1070	1070.0	4	0:00:07
LA15	20, 5	1089	1093	1090	1091	1089	1089.0	67	0:02:21
LA16	10, 10	717	717	732	717	717	717.0	2	0:00:05
LA17	10, 10	646	646	654	646	646	646.0	1	0:00:02
LA18	10, 10	666	674	694	666	666	666.0	72	0:02:49
LA19	10, 10	700	725	730	703	700	701.2	534	0:22:48
LA20	10, 10	756	756	756	757	756	756.0	1	0:00:03
LA21	15, 10	(829)	861	916	845	844	846.6	617	2:30:04
LA22	15, 10	(753)	790	839	775	772	774.6	754	2:30:05
LA23	15, 10	(832)	884	892	857	850	856.2	665	2:30:04
LA24	15, 10	(801)	825	870	818	821	823.0	723	2:30:06
LA25	15, 10	(782)	823	858	805	802	804.0	785	2:30:04
LA26	20, 10	(1059)	1086	1114	1074	1067	1068.2	218	2:30:31
LA27	20, 10	(1087)	1109	1141	1101	1095	1098.4	249	2:30:15
LA28	20, 10	(1077)	1097	1135	1084	1083	1086.0	227	2:30:07
LA29	20, 10	(996)	1016	1046	1006	1003	1004.6	258	2:30:08
LA30	20, 10	(1072)	1105	1148	1087	1087	1090.2	257	2:30:15
LA31	30, 10	⟶1520	1532	1549	1525	1520	1523.2	61	2:16:01
LA32	30, 10	(1658)	1668	1691	1664	1659	1660.4	65	2:30:56
LA33	30, 10	(1498)	1511	1530	1502	1498	1500.2	70	2:30:47
LA34	30, 10	(1536)	1542	1556	1542	1536	1537.2	70	2:31:01
LA35	30, 10	(1550)	1559	1577	1556	1551	1551.8	73	2:31:01
LA36	15, 15	1023	1054	1119	1034	1026	1033.0	213	2:30:17
LA37	15, 15	1062	1122	1190	1084	1084	1086.0	219	2:30:21
LA38	15, 15	954	1004	1063	973	976	976.0	249	2:30:18
LA39	15, 15	1011	1041	1131	1018	1024	1025.2	258	2:30:26
LA40	15, 15	955	1009	1057	984	977	984.0	169	2:30:33
ABZ5	10, 10	954	N/A	N/A	962	959	960.0	1000	0:48:07
ABZ6	10, 10	807	N/A	N/A	807	807	807.0	4	0:00:08
ABZ7	20, 15	(527)	N/A	N/A	544	547	548.8	57	2:32:24

Table 2: Continued.

Instance	n, m	BKS	Best of TS [9]	Best of PSO [10]	Best of CP [20]	MPM-UPLA			
						Best	Avg	No of Iters	Time
ABZ8	20, 15	(540)	N/A	N/A	555	561	565.2	67	2:31:02
ABZ9	20, 15	(539)	N/A	N/A	562	555	565.2	68	2:31:18
CAR1	11, 5	(5035)	N/A	N/A	5057	5050	5053.4	1000	0:07:21
CAR2	13, 4	(5986)	N/A	N/A	5987	5986	5986.0	1000	0:05:45
CAR3	12, 5	(5623)	N/A	N/A	5626	5625	5629.2	1000	0:10:00
CAR4	14, 4	(6515)	N/A	N/A	6518	6515	6515.2	1000	0:07:25
CAR5	10, 6	5615	N/A	N/A	5764	5680	5691.4	1000	0:09:24
CAR6	8, 9	6147	N/A	N/A	6147	6147	6147.0	17	0:00:12
CAR7	7, 7	4425	N/A	N/A	4432	4425	4430.6	803	0:04:07
CAR8	8, 8	5692	N/A	N/A	5692	5692	5692.0	99	0:00:54
ORB1	10, 10	746	N/A	N/A	763	746	746.0	20	0:00:40
ORB2	10, 10	696	N/A	N/A	703	696	698.4	807	0:32:37
ORB3	10, 10	712	N/A	N/A	720	715	716.0	1000	0:44:29
ORB4	10, 10	753	N/A	N/A	753	753	753.0	8	0:00:22
ORB5	10, 10	639	N/A	N/A	643	639	639.0	325	0:15:04
ORB6	10, 10	754	N/A	N/A	766	754	754.0	145	0:05:35
ORB7	10, 10	302	N/A	N/A	302	302	303.4	866	0:38:49
ORB8	10, 10	639	N/A	N/A	651	641	641.0	1000	0:42:24
ORB9	10, 10	694	N/A	N/A	694	694	694.0	33	0:01:27
ORB10	10, 10	742	N/A	N/A	750	742	743.4	847	0:41:27

Table 3: Results on Vdata.

Instance	n, m	BKS	Best of TS [9]	Best of PSO [10]	Best of CP [20]	MPM-UPLA			
						Best	Avg	No of Iters	Time
M6	6, 6	47	47	47	47	47	47.0	1	< 1 sec.
M10	10, 10	655	655	655	655	655	655.0	1	0:00:01
M20	20, 5	1022	1023	1024	1023	1022	1022.0	4	0:00:10
LA1	10, 5	570	573	571	570	570	570.2	431	0:02:30
LA2	10, 5	529	531	530	529	529	529.0	134	0:00:43
LA3	10, 5	477	482	479	478	477	477.8	875	0:04:58
LA4	10, 5	502	504	504	502	502	502.0	345	0:01:48
LA5	10, 5	457	464	460	458	457	457.4	527	0:02:43
LA6	15, 5	799	802	799	799	799	799.0	6	0:00:05
LA7	15, 5	749	751	750	750	749	749.0	103	0:01:47
LA8	15, 5	765	766	766	766	765	765.0	96	0:01:39
LA9	15, 5	853	854	855	854	853	853.0	28	0:00:28
LA10	15, 5	804	805	805	804	804	804.0	124	0:02:04
LA11	20, 5	1071	1073	1071	1071	1071	1071.0	1	0:00:03
LA12	20, 5	936	940	936	936	936	936.0	1	0:00:03
LA13	20, 5	1038	1040	1038	1038	1038	1038.0	4	0:00:08
LA14	20, 5	1070	1071	1070	1070	1070	1070.0	2	0:00:03
LA15	20, 5	1089	1091	1090	1090	1089	1089.0	36	0:01:21
LA16	10, 10	717	717	717	717	717	717.0	1	0:00:02
LA17	10, 10	646	646	646	646	646	646.0	1	0:00:02
LA18	10, 10	663	663	663	663	663	663.0	1	0:00:02
LA19	10, 10	617	617	619	617	617	617.0	3	0:00:10
LA20	10, 10	756	756	756	756	756	756.0	1	0:00:02
LA21	15, 10	(802)	826	819	804	817	822.2	501	2:30:08
LA22	15, 10	(735)	745	755	736	755	756.4	536	2:30:11
LA23	15, 10	(812)	826	828	815	826	831.2	533	2:30:06
LA24	15, 10	(775)	796	790	775	793	797.4	589	2:30:06
LA25	15, 10	(753)	770	775	756	768	770.2	562	2:30:08
LA26	20, 10	(1053)	1058	1058	1054	1073	1075.8	267	2:30:24
LA27	20, 10	1084	1088	1091	1084	1106	1110.2	260	2:30:20
LA28	20, 10	1069	1073	1076	1070	1091	1094.6	253	2:30:20
LA29	20, 10	(994)	995	1003	995	1010	1013.6	241	2:30:18
LA30	20, 10	(1069)	1071	1078	1072	1085	1093.8	222	2:30:17

Table 3: Continued.

Instance	n, m	BKS	Best of TS [9]	Best of PSO [10]	Best of CP [20]	MPM-UPLA			
						Best	Avg	No of Iters	Time
LA31	30, 10	1520	1521	1524	1522	1538	1540.6	82	2:31:00
LA32	30, 10	(1658)	1658	1664	1661	1681	1686.6	79	2:31:10
LA33	30, 10	(1498)	1498	1503	1500	1511	1519.2	79	2:31:12
LA34	30, 10	1535	1536	1541	1537	1557	1561.4	84	2:30:49
LA35	30, 10	1549	1553	1555	1551	1563	1570.8	74	2:31:05
LA36	15, 15	948	948	955	948	948	948.0	4	0:03:21
LA37	15, 15	986	986	993	986	986	986.0	54	0:49:27
LA38	15, 15	943	943	943	943	943	943.0	1	0:00:38
LA39	15, 15	922	922	945	922	922	922.0	27	0:24:02
LA40	15, 15	955	955	955	955	955	955.0	1	0:00:41
ABZ5	10, 10	859	N/A	N/A	860	859	859.0	12	0:00:53
ABZ6	10, 10	742	N/A	N/A	742	742	742.0	1	0:00:04
ABZ7	20, 15	(495)	N/A	N/A	495	535	536.0	86	2:30:44
ABZ8	20, 15	(509)	N/A	N/A	509	554	554.8	84	2:30:58
ABZ9	20, 15	(499)	N/A	N/A	500	540	541.8	85	2:31:11
CAR1	11, 5	5005	N/A	N/A	5013	5007	5007.0	1000	0:08:30
CAR2	13, 4	5929	N/A	N/A	5930	5929	5929.0	255	0:01:22
CAR3	12, 5	(5598)	N/A	N/A	5600	5601	5601.4	1000	0:10:41
CAR4	14, 4	6514	N/A	N/A	6517	6514	6514.0	199	0:01:22
CAR5	10, 6	(4913)	N/A	N/A	4932	4935	4941.0	1000	0:11:21
CAR6	8, 9	5486	N/A	N/A	5486	5486	5486.0	1	0:00:01
CAR7	7, 7	4281	N/A	N/A	4281	4281	4281.0	1	0:00:00
CAR8	8, 8	4613	N/A	N/A	4613	4613	4613.0	2	0:00:02
ORB1	10, 10	695	N/A	N/A	695	695	695.0	1	0:00:01
ORB2	10, 10	620	N/A	N/A	620	620	620.0	1	0:00:04
ORB3	10, 10	648	N/A	N/A	648	648	648.0	1	0:00:02
ORB4	10, 10	753	N/A	N/A	753	753	753.0	1	0:00:02
ORB5	10, 10	584	N/A	N/A	584	584	584.0	1	0:00:04
ORB6	10, 10	715	N/A	N/A	715	715	715.0	1	0:00:01
ORB7	10, 10	275	N/A	N/A	275	275	275.0	10	0:00:32
ORB8	10, 10	573	N/A	N/A	573	573	573.0	1	0:00:02
ORB9	10, 10	659	N/A	N/A	659	659	659.0	1	0:00:03
ORB10	10, 10	681	N/A	N/A	681	681	681.0	1	0:00:03

(vi) LA21-25 consists of five 15-job/10-machine instances, i.e., LA21, LA22, . . ., LA25.

(vii) LA26-30 consists of five 20-job/10-machine instances, i.e., LA26, LA27, . . ., LA30.

(viii) LA31-35 consists of five 30-job/10-machine instances, i.e., LA31, LA32, . . ., LA35.

(ix) LA36-40 consists of five 15-job/15-machine instances, i.e., LA36, LA37, . . ., LA40.

(x) ABZ5-6 consists of two 10-job/10-machine instances, i.e., ABZ5 and ABZ6.

(xi) ABZ7-9 consists of three 20-job/15-machine instances, i.e., ABZ7, ABZ8, and ABZ9.

(xii) CAR1-8 consists of eight instances, i.e., CAR1, CAR2, . . ., CAR8.

(xiii) ORB1-10 consists of ten 10-job/10-machine instances, i.e., ORB1, ORB2, . . ., ORB10.

In addition to the 13 categories, Table 4 includes four more instance categories, i.e., M6-LA40, SM-M6-LA40, M6-ORB10, and SM-M6-ORB10. These additional categories were used for comparing performance of the algorithms. SM in SM-M6-LA40 and SM-M6-ORB10 indicates that these categories contain only small-to-medium instances. Let instances be defined as small-to-medium instances if their $nm < 150$ and large instances otherwise (where $n =$ the number of jobs and $m =$ the number of machines). The details of these four additional categories are given below:

(i) M6-LA40 consists of the first 43 instances of all 66 instances, starting from M6 to LA40. These 43 instances were used by TS [9] and PSO [10] in their original articles.

(ii) SM-M6-LA40 consists of all 23 small-to-medium instances from M6-LA40.

(iii) M6-ORB10 consists of all 66 instances, starting from M6 to ORB10. These 66 instances were used by CP [20] in its original article.

(iv) SM-M6-ORB10 consists of all 43 small-to-medium instances from M6-ORB10.

5. Result Analysis and Discussion

This section analyzes and discusses the results shown in Section 4. Like Section 4, MPM-UPLA in this section stands for the whole two-level metaheuristic, i.e., MPM-UPLA

Table 4: Avg %BDs.

Category	No. of instances	n, m	Edata				Rdata				Vdata			
			TS	PSO	CP	MPM-UPLA	TS	PSO	CP	MPM-UPLA	TS	PSO	CP	MPM-UPLA
M6-20	3	Vary	3.62	1.66	0.23	0.14	2.67	1.94	0.07	0.00	0.03	0.07	0.03	0.00
LA01-05	5	10, 5	1.25	1.11	0.62	0.00	0.91	0.67	0.42	0.08	0.78	0.37	0.09	0.00
LA06-10	5	15, 5	0.08	0.08	0.08	0.00	0.37	0.18	0.10	0.00	0.20	0.12	0.08	0.00
LA11-15	5	20, 5	0.55	0.00	0.05	0.00	0.17	0.06	0.07	0.00	0.22	0.02	0.02	0.00
LA16-20	5	10, 10	5.46	0.79	0.37	0.00	0.95	2.36	0.11	0.00	0.00	0.06	0.00	0.00
LA21-25	5	15, 10	4.34	4.10	1.37	0.50	4.65	9.49	2.58	2.31	2.21	2.33	0.23	2.13
LA26-30	5	20, 10	5.92	5.64	2.16	1.06	2.30	5.53	1.15	0.83	0.30	0.70	0.11	1.82
LA31-35	5	30, 10	2.38	2.87	0.55	0.23	0.65	1.82	0.35	0.02	0.08	0.35	0.14	1.16
LA36-40	5	15, 15	5.01	3.85	0.61	0.17	4.51	11.08	1.77	1.65	0.00	0.79	0.00	0.00
ABZ5-6	2	10, 10	N/A	N/A	0.39	0.00	N/A	N/A	0.42	0.26	N/A	N/A	0.06	0.00
ABZ7-9	3	20, 15	N/A	N/A	3.66	1.64	N/A	N/A	3.42	3.55	N/A	N/A	0.07	8.38
CAR1-8	8	Vary	N/A	N/A	1.02	0.21	N/A	N/A	0.42	0.19	N/A	N/A	0.08	0.07
ORB1-10	10	10, 10	N/A	N/A	1.39	0.06	N/A	N/A	0.96	0.07	N/A	N/A	0.00	0.00
M6-LA40	43	Vary	3.16	2.26	0.69	0.24	1.87	3.76	0.77	0.57	0.44	0.56	0.08	0.59
SM-M6-LA40	23	$nm <$ 150	2.07	0.65	0.27	0.02	0.87	0.96	0.16	0.02	0.26	0.13	0.04	0.00
M6-ORB10	66	Vary	N/A	N/A	0.96	0.26	N/A	N/A	0.87	0.57	N/A	N/A	0.07	0.78
SM- M6-ORB10	43	$nm <$ 150	N/A	N/A	0.67	0.07	N/A	N/A	0.55	0.24	N/A	N/A	0.04	0.01

combined with MPM-LOLA. The performance of MPM-UPLA was compared with the performance of TS [9], PSO [10], and CP [20] via three performance indicators. These indicators are the number of instances achieved in finding the best-known solutions, the number of instances won by an algorithm against another, and the average percent deviation of the algorithm's best-found solution value from the best-known solution value (*Avg %BD*). Of each instance, the best-known solution value means the best solution value found by the published literature. An only exception is in LA31 of Rdata, where its best-known solution value was taken from the best-found solution value of MPM-UPLA. The reason is that, in LA31 of Rdata, MPM-UPLA found the better solution than the previously published best-known solution.

For each instance set, this section separates analyses on the first 43 instances from those on all 66 instances. The reason is that the results of TS and PSO were given on only the 43 instances in their original articles [9, 10], while the results of CP were given on the 66 instances in its original article [20]. In addition, this section separates analyses on small-to-medium instances from those on all given instances. Note that all instances with $nm < 150$ are defined as small-to-medium instances (where n = the number of jobs and m = the number of machines). Sections 5.1 to 5.3 show the analyses and discussions via the three given indicators. Then, Section 5.4 provides an overall summary from Sections 5.1 to 5.3.

5.1. The Number of Instances Achieved in Finding the Best-Known Solutions. Of each instance set, this section first compares the number of instances achieved in finding the best-known solutions by MPM-UPLA with those by TS, PSO, and CP on the first 43 instances. The numbers of instances achieved by each algorithm in Edata, Rdata, and Vdata were counted from Tables 1, 2, and 3, respectively. For the first 43 instances, MPM-UPLA obviously outperforms the three other algorithms on all three instance sets, especially Rdata. Of each instance set, the comparison results are given below:

(i) For the first 43 instances of Edata, the algorithms TS, PSO, CP, and MPM-UPLA reach the best-known solutions on 10, 15, 22, and 27 instances, respectively.

(ii) For the first 43 instances of Rdata, the algorithms TS, PSO, CP, and MPM-UPLA reach the best-known solutions on 4, 4, 8, and 24 instances, respectively. MPM-UPLA also found a new best-known solution value (i.e., 1520) for LA31 of Rdata. This value is defined as the optimal solution value because it equals the optimal solution value's lower bound given in [20].

(iii) For the first 43 instances of Vdata, the algorithms TS, PSO, CP, and MPM-UPLA reach the best-known solutions on 14, 13, 23, 28 instances, respectively.

Of each instance set, this section then compares the number of instances achieved by MPM-UPLA with that by CP on all 66 instances. For all 66 instances, MPM-UPLA outperforms CP on all three instance sets, especially Rdata. Of each instance set, the comparison results are given below:

(i) For all 66 instances of Edata, CP and MPM-UPLA reach the best-known solutions on 32 and 44 instances, respectively.

(ii) For all 66 instances of Rdata, CP and MPM-UPLA reach the best-known solutions on 14 and 38 instances, respectively.

(iii) For all 66 instances of Vdata, CP and MPM-UPLA reach the best-known solutions on 39 and 45, respectively.

Thus, as a conclusion, MPM-UPLA outperforms TS, PSO, and CP in finding the best-known solutions on all three instance sets, especially Rdata. On Rdata, the number of instances achieved by MPM-UPLA is more than double the number of instances achieved by each of the others. Moreover, MPM-UPLA also found the new best-known solution value on LA31 of Rdata.

5.2. The Number of Instances Won. This section first compares the number of instances won by MPM-UPLA with those by TS, PSO, and CP on the first 43 instances of each instance set. Note that in the first 43 instances, there are 23 small-to-medium instances included. The numbers of instances won in Edata, Rdata, and Vdata were counted from Tables 1, 2, and 3, respectively. For the first 43 instances, MPM-UPLA obviously outperforms the three other algorithms on Edata and Rdata; MPM-UPLA outperforms TS and PSO but underperforms CP on Vdata. However, when considering only the 23 small-to-medium instances, MPM-UPLA outperforms CP on Vdata. Of each instance set, the comparison results are detailed below:

(i) Out of the first 43 instances of Edata, MPM-UPLA has 33 wins, 10 draws, and 0 losses against TS; it has 28 wins, 15 draws, and 0 losses against PSO. In addition, it has 21 wins, 20 draws, and 2 losses against CP.

(ii) Out of the first 43 instances of Rdata, MPM-UPLA has 39 wins, 4 draws, and 0 losses against each of TS and PSO. In addition, it has 29 wins, 11 draws, and 3 losses against CP.

(iii) Out of the first 43 instances of Vdata, MPM-UPLA has 19 wins, 13 draws, and 11 losses against TS; it has 18 wins, 14 draws, and 11 losses against PSO. In addition, it has 7 wins, 21 draws, and 15 losses against CP. However, when considering only the 23 small-to-medium instances, MPM-UPLA has 7 wins, 16 draws, and 0 losses against CP.

Then, this section compares the number of instances won by MPM-UPLA with that by CP on all 66 instances of each instance set. Note that in the 66 instances, there are 43 small-to-medium instances included. For the 66 instances, MPM-UPLA outperforms CP on Edata and Rdata, but it underperforms CP on Vdata. However, when considering only the 43 small-to-medium instances, MPM-UPLA obviously outperforms CP on all three instance sets. For each instance set, the comparison results are detailed below:

(i) Out of all 66 instances of Edata, MPM-UPLA has 34 wins, 29 draws, and 3 losses against CP. Out of the 43 small-to-medium instances of Edata, MPM-UPLA has 17 wins, 24 draws, and 2 losses against CP.

(ii) Out of all 66 instances of Rdata, MPM-UPLA has 44 wins, 17 draws, and 5 losses against CP. Out of the 43 small-to-medium instances of Rdata, MPM-UPLA has 28 wins, 15 draws, and 0 losses against CP.

(iii) Out of all 66 instances of Vdata, MPM-UPLA has 11 wins, 35 draws, and 20 losses against CP. Out of the 43 small-to-medium instances of Vdata, MPM-UPLA has 11 wins, 30 draws, and 2 losses against CP.

As a conclusion, in terms of the number of instances won, MPM-UPLA outperforms the three other algorithms on Edata and Rdata. For Vdata, MPM-UPLA outperforms TS and PSO but underperforms CP. However, when considering only small-to-medium instances, MPM-UPLA outperforms CP on Vdata.

5.3. Avg %BD. This section analyzes *Avg %BDs* in Table 4. To do so, it first analyzes *Avg %BDs* of the first 43 instances of each instance set. Then, it analyzes those of the 23 small-to-medium instances of the first 43 instances. In Table 4, the rows M6-LA40 and SM-M6-LA40 provide *Avg % BDs* of the first 43 instances and those of the 23 small-to-medium instances, respectively. For *Avg %BDs* of the first 43 instances, MPM-UPLA outperforms the three other algorithms on Edata and Rdata, but it underperforms the three other algorithms on Vdata. When considering only the 23 small-to-medium instances, MPM-UPLA obviously outperforms the three other algorithms on all three instance sets. Of each instance set, the analysis results are detailed below:

(i) For the first 43 instances of Edata, MPM-UPLA's *Avg %BD* (i.e., 0.24%) is much better than those of TS, PSO, and CP (i.e., 3.16%, 2.26%, and 0.69%, respectively). Based on these 43 instances, paired t tests concluded that the mean *%BD* of MPM-UPLA is significantly better than those of TS, PSO, and CP (with p values of 3×10^{-10}, 1×10^{-8}, and 0.0002, respectively). When considering only the 23 small-to-medium instances, MPM-UPLA's *Avg %BD* (i.e., 0.02%) is also much better than those of TS, PSO, and CP (i.e., 2.07%, 0.65%, and 0.27%, respectively).

(ii) For the first 43 instances of Rdata, MPM-UPLA's *Avg %BD* (i.e., 0.57%) is much better than those of TS, PSO, and CP (i.e., 1.87%, 3.76%, and 0.77%, respectively). Based on these 43 instances, paired t tests concluded that the mean *%BD* of MPM-UPLA is significantly better than those of TS, PSO, and CP (with p values of 4×10^{-7}, 1×10^{-7}, and 0.00004, respectively). When considering only the 23 small-to-medium instances, MPM-UPLA's *Avg %BD* (i.e., 0.02%) is also much better than those of TS, PSO, and CP (i.e., 0.87%, 0.96%, and 0.16%, respectively).

(iii) For the first 43 instances of Vdata, MPM-UPLA's *Avg %BD* (i.e., 0.59%) is worse than those of TS, PSO, and CP (i.e., 0.44%, 0.56%, and 0.08%, respectively). However, when considering only the 23 small-to-medium instances, MPM-UPLA's *Avg %BD* (i.e., 0.00%) is better than those of TS, PSO,

and CP (i.e., 0.26%, 0.13%, and 0.04%, respectively). Based on the 23 small-to- medium instances, paired t tests concluded that the mean *% BD* of small-to-medium instances of MPM-UPLA is significantly better than those of TS, PSO, and CP (with p values of 0.001, 0.0007, and 0.004, respectively).

Of each instance set, this section then compares *Avg % BDs* of MPM-UPLA and CP from all 66 instances and from their 43 small-to-medium instances. In Table 4, the rows M6-ORB10 and SM-M6-ORB10 provide *Avg %BDs* of all 66 instances and those of the 43 small-to-medium instances, respectively. For all 66 instances, MPM-UPLA outperforms CP on Edata and Rdata, but it underperforms CP on Vdata. However, when considering only the 43 small-to-medium instances, MPM-UPLA obviously outperforms CP on all three instance sets. Of each instance set, the comparison results are detailed below:

(i) For all 66 instances of Edata, MPM-UPLA's *Avg % BD* (i.e., 0.26%) is better than CP's *Avg %BD* (i.e., 0.96%). A paired t-test concluded that the mean *% BD* of MPM-UPLA is significantly better than the mean *%BD* of CP (with p value of 0.00001). When considering only the 43 small-to-medium instances, MPM-UPLA's *Avg %BD* (i.e., 0.07%) is also better than CP's *Avg %BD* (i.e., 0.67%).

(ii) For all 66 instances of Rdata, MPM-UPLA's *Avg % BD* (i.e., 0.57%) is better than CP's *Avg %BD* (i.e., 0.87%). A paired t-test concluded that the mean *% BD* of MPM-UPLA is significantly better than the mean *%BD* of CP (with p value of 0.00002). When considering only the 43 small-to-medium instances, MPM-UPLA's *Avg %BD* (i.e., 0.24%) is also better than CP's *Avg %BD* (i.e., 0.55%).

(iii) For all 66 instances of Vdata, MPM-UPLA's *Avg % BD* (i.e., 0.78%) is worse than CP's *Avg %BD* (i.e., 0.07%). However, when considering only the 43 small-to-medium instances, MPM-UPLA's *Avg % BD* (i.e., 0.01%) is better than CP's *Avg %BD* (i.e., 0.04%). Based on the 43 small-to-medium instances, a paired t-test concluded that the mean *%BD* of small-to-medium instances of MPM-UPLA is significantly better than that of CP (with p value of 0.002).

As a conclusion, based on *Avg %BDs*, MPM-UPLA obviously outperforms the three other algorithms on Edata and Rdata, but it underperforms the three other algorithms on Vdata. When considering only the small-to-medium instances, MPM-UPLA outperforms the three other algorithms on all three instance sets.

5.4. Overall Summary. In the number of instances achieved in finding the best-known solutions, MPM-UPLA outperforms the three other algorithms on all three sets of instances. In the number of instances won, MPM-UPLA outperforms the three other algorithms on Edata and Rdata; MPM-UPLA outperforms TS and PSO but underperforms CP on Vdata. However, when considering only small-to-medium instances, MPM-UPLA outperforms CP on Vdata in the number of instances won. In *Avg %BD*, MPM-UPLA outperforms the three other algorithms on Edata and Rdata, but it underperforms the three other algorithms on Vdata. However, when considering only small-to-medium instances, MPM-UPLA outperforms the three other algorithms on Vdata. As a conclusion, MPM-UPLA usually performs very well on the MPMJSP instances where each operation has less than four optional machines (e.g., the instances in Edata and Rdata). When each operation has many optional machines (i.e., $\geq 0.5m$ optional machines), MPM-UPLA usually performs well on only small-to-medium instances (i.e., the instances of $nm < 150$).

6. Conclusion

In this paper, a two-level metaheuristic was proposed for solving MPMJSP. The two-level metaheuristic consists of MPM-UPLA and MPM-LOLA as its upper- and lower-level algorithms, respectively. MPM-UPLA, a population-based algorithm, acts as the MPM-LOLA's parameter controller. MPM-LOLA is a local search algorithm, searching for an MPMJSP's optimal solution. MPM-LOLA has many changes from its older variants, such as perturbation and neighbor operators. It also uses a unique method to select an optional machine for each operation. The MPM-UPLA's function is to evolve the MPM-LOLA's input-parameter values, so that MPM-LOLA can perform its best for every single instance. In this paper's experiment, the performance of the two-level metaheuristic was evaluated on the three instance sets, i.e., Edata, Rdata, and Vdata. The experiment's results indicated that the two-level metaheuristic performs very well on Edata and Rdata. For Vdata, the two-level metaheuristic usually performs well on only the category of small-to-medium instances. Thus, a future research should be focused to enhance the two-level metaheuristic's performance, especially on large instances of Vdata.

Data Availability

The data used to support the findings of this study are available from the author upon request.

Conflicts of Interest

The author declares that there are no conflicts of interest.

References

[1] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.

[2] B. Çaliş and S. Bulkan, "A research survey: review of AI solution strategies of job shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 26, no. 5, pp. 961–973, 2015.

[3] C. Cebi, E. Atac, and O. K. Sahingoz, "Job shop scheduling problem and solution algorithms: a review," in *Proceedings of the 11th International Conference on Computing,*

Communication and Networking Technologies, Kharagpur, India, July 2020.

[4] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under Industry 4.0," *Journal of Intelligent Manufacturing*, vol. 30, pp. 1809–1830, 2019.

[5] P. Pongchairerks, "A job-shop scheduling problem with bi-directional circular precedence constraints," *Complexity*, vol. 2021, Article ID 3237342, 19 pages, 2021.

[6] F. Zhao, X. He, and L. Wang, "A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem," *IEEE Transactions on Cybernetics*, vol. 51, no. 11, pp. 5291–5303, 2021.

[7] F. Zhao, L. Zhao, L. Wang, and H. Song, "An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion," *Expert Systems with Applications*, vol. 160, Article ID 113678, 2020.

[8] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, pp. 369–375, 1990.

[9] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *OR Spektrum*, vol. 15, pp. 205–215, 1994.

[10] P. Pongchairerks and V. Kachitvichyanukul, "A particle swarm optimization algorithm on job-shop scheduling problems with multi-purpose machines," *Asia-Pacific Journal of Operational Research*, vol. 26, no. 2, pp. 161–184, 2009.

[11] P. Pongchairerks, "A two-level metaheuristic algorithm for the job-shop scheduling problem," *Complexity*, vol. 2019, Article ID 8683472, 11 pages, 2019.

[12] P. Pongchairerks, "An enhanced two-level metaheuristic algorithm with adaptive hybrid neighborhood structures for the job-Shop scheduling problem," *Complexity*, vol. 2020, Article ID 3489209, 15 pages, 2020.

[13] J. Grefenstette, "Proportional selection and sampling algorithms," in *In Evolutionary Computation 1: Basic Algorithms and Operators*, T. Bäck, D. B. Fogel, and Z. Michalewicz, Eds., Taylor & Francis Group, New York, NY, USA, 2000.

[14] P. Pongchairerks and V. Kachitvichyanukul, "A two-level particle swarm optimisation algorithm for open-shop scheduling problem," *International Journal of Computing Science and Mathematics*, vol. 7, no. 6, pp. 575–585, 2016.

[15] P. Pongchairerks and V. Kachitvichyanukul, "A two-level particle swarm optimisation algorithm on job-shop scheduling problems," *International Journal of Operational Research*, vol. 4, no. 4, pp. 390–411, 2009.

[16] C. Bierwirth and D. C. Mattfeld, "Production scheduling and rescheduling with genetic algorithms," *Evolutionary Computation*, vol. 7, no. 1, pp. 1–17, 1999.

[17] P. Pongchairerks, "Efficient local search algorithms for job-shop scheduling problems," *International Journal of Mathematics in Operational Research*, vol. 9, no. 2, pp. 258–277, 2016.

[18] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, "Review on flexible job shop scheduling," *IET Collaborative Intelligent Manufacturing*, vol. 1, no. 3, pp. 67–107, 2019.

[19] X. Li and L. Gao, "Review for flexible job shop scheduling," in *Engineering Applications of Computational Methods*, vol. 2, Berlin, Germany, Springer, 2020.

[20] D. Behnke and M. J. Geiger, "Test instances for the flexible job shop scheduling problem with work centers," Logistics Management Department, Hamburg, Germany, RR-12-01-01, 2012.

[21] G. A. Kasapidis, D. C. Paraskevopoulos, P. P. Repoussis, and C. D. Tarantilis, "Flexible job shop scheduling problems with arbitrary precedence graphs," *Production and Operations Management*, vol. 30, no. 11, pp. 4044–4068, 2021.

[22] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *International Series in Operations Research and Management Science*, vol. 57, Springer, Boston, MA, USA, 2003.

[23] M. Avci and S. Topaloglu, "A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem," *Computers & Operations Research*, vol. 83, pp. 54–65, 2017.

[24] P. Venkatesh, A. Singh, and R. Mallipeddi, "A multi-start iterated local search algorithm for the maximum scatter traveling salesman problem," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1390–1397, Wellington, New Zealand, June 2019.

[25] A. Ishigaki and S. Takaki, "Iterated local search algorithm for flexible job shop scheduling," in *Proceedings of the 6th IIAI International Congress on Advanced Applied Informatics*, pp. 947–952, Hamamatsu, Japan, July 2017.

[26] D. de C. Bissoli and A. R. S. Amaral, "A hybrid iterated local search metaheuristic for the flexible job shop scheduling problem," in *Proceedings of the XLIV Latin American Computer Conference (CLEI)*, pp. 149–157, Sao Paulo, Brazil, October 2018.

[27] P. Guo, W. Cheng, and Y. Wang, "Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm," *Engineering Optimization*, vol. 47, no. 11, pp. 1564–1585, 2015.

[28] A. E. Eiben, R. Hinterding, and Z. Michalewicz, "Parameter control in evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 124–141, 1999.

[29] S.-J. Wu and P.-T. Chow, "Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization," *Engineering Optimization*, vol. 24, no. 2, pp. 137–159, 1995.

[30] T. Brys, M. M. Drugan, and A. Nowé, "Meta-evolutionary algorithms and recombination operators for satisfiability solving in fuzzy logics," in *Proceedings of the 2013 IEEE Congress on Evolutionary Computation*, pp. 1060–1067, IEEE, Cancun, Mexico, June 2013.

[31] P. Cortez, M. Rocha, and J. Neves, "A meta-genetic algorithm for time series forecasting," in *Proceedings of the Workshop on Artificial Intelligence Techniques for Financial Time Series Analysis, 10th Portuguese Conference on Artificial Intelligence (EPIA 2001)*, pp. 21–31, Porto, Portugal, December 2001.

[32] P. Pongchairerks, "A self-tuning PSO for job-shop scheduling problems," *International Journal of Operational Research*, vol. 19, no. 1, pp. 96–113, 2014.

[33] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, New York, NY, USA, 1997.

[34] C. Bierwirth, "A generalized permutation approach to job shop scheduling with genetic algorithms," *OR Spektrum*, vol. 17, no. 2-3, pp. 87–92, 1995.

[35] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *In Industrial Scheduling*, J. F. Muth and G. L. Thompson, Eds., pp. 225–251, Prentice-Hall, Englewood, NJ, USA, 1963.

[36] S. Lawrence, *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)*, Carnegie Mellon University, Pittsburgh, PA, USA, 1984.

[37] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149–156, 1991.
[38] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, vol. 34, no. 3, pp. 391–401, 1988.
[39] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Management Science*, vol. 35, no. 2, pp. 164–176, 1989.