Hindawi
Complexity
Volume 2021, Article ID 3237342, 19 pages
https://doi.org/10.1155/2021/3237342

Research Article

A Job-Shop Scheduling Problem with Bidirectional Circular Precedence Constraints

Pisut Pongchairerks

Industrial Engineering Program, Faculty of Engineering, Thai-Nichi Institute of Technology, Bangkok 10250, Thailand

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

Received 17 June 2021; Revised 25 July 2021; Accepted 14 August 2021; Published 9 November 2021

Academic Editor: Jenq-Haur Wang

This paper introduces a job-shop scheduling problem (JSP) with bidirectional circular precedence constraints, called BCJSP. In the problem, each job can be started from any operation and continued by its remaining operations in a circular precedence-relation chain via either a clockwise or counterclockwise direction. To solve BCJSP, this paper proposes a multilevel metaheuristic consisting of top-, middle-, and bottom-level algorithms. The top- and middle-level algorithms are population-based metaheuristics, while the bottom-level algorithm is a local search algorithm. The top-level algorithm basically controls a start operation and an operation-precedence-relation direction of each job, so that BCJSP becomes a JSP instance that is a subproblem of BCJSP. Moreover, the top-level algorithm can also be used to control input parameters of the middle-level algorithm, as an optional extra function. The middle-level algorithm controls input parameters of the bottom-level algorithm, and the bottom-level algorithm then solves the BCJSP's subproblem. The middle-level algorithm evolves the bottom-level algorithm's parameter values by using feedback from the bottom-level algorithm. Likewise, the top-level algorithm evolves the start operations, the operation-precedence-relation directions, and the middle-level algorithm's parameter values by using feedback from the middle-level algorithm. Performance of two variants of the multilevel metaheuristic (i.e., with and without the mentioned extra function) was evaluated on BCJSP instances modified from well-known JSP instances. The variant with the extra function performs significantly better in number than the other. The existing JSP-solving algorithms can also solve BCJSP; however, their results on BCJSP are clearly worse than those of the two variants of the multilevel metaheuristic.

1. Introduction

The job-shop scheduling problem (JSP) [1, 2] and the open-shop scheduling problem (OSP) [3, 4] are well-known in practical applications. They are also interesting academic topics since they are NP-hard problems [5]. They both involve scheduling jobs onto machines in order to minimize makespan, i.e., the schedule's length. Each job consists of a number of operations; each operation must be processed on a predetermined machine with a given processing time. Each machine cannot process more than one operation at a time, and it cannot be stopped during processing an operation. To complete each job in JSP, all of its operations must be processed in the sequence from the first to the last operations. This sequence is called an operation-precedence-relation chain. The operation-precedence-relation chain of each job is very strict and, thus, cannot be changed. In contrast to JSP, OSP has no operation-precedence-relation chains. This means all operations of each job in OSP can be processed in any orders. The job-shop scheduling problem with bidirectional circular precedence constraints (BCJSP) introduced in this paper is an intermediate problem between JSP and OSP. It is a generalized JSP where the operation-precedence-relation chain of each job is circular and bidirectional.

BCJSP has a wide range of real applications, e.g., health check-up service, automobile repair shop, and instrument calibration service. In fact, when taking layouts and distances into account, many OSP's applications become BCJSP's applications. The health check-up service, as a

BCJSP application, starts with multiple optional check-up programs offered to hospital customers. An optional check-up program consists of specific diagnoses, each of which is provided in a different room. For customer satisfaction, the best circular route of all diagnosis rooms has been predetermined by a hospital service manager for each check-up program. The manager may assign each customer to receive any diagnosis of his chosen program as the first diagnosis. However, the manager has to assign the customer to receive his next diagnosis in a nearest predetermined room, and so on. The customer finishes his check-up activities after successfully receiving all diagnoses of his chosen program. Notice that if the diagnosis rooms are very close to each other, the manager can then assign the customer to receive all diagnoses in any orders. Then, this application fits with OSP rather than BCJSP.

There are a number of JSP and OSP's variants recently presented in the literature, e.g., [6–11]. Some of them have some partially similar properties to the BCJSP's properties. For example, the extended resource-constrained project scheduling problem [12] allows processing its activities bidirectionally, but not circularly. The flexible job-shop scheduling problem (FJSP) has more flexibility than JSP and is also defined as a generalized form of JSP. However, the flexibility of FJSP [13, 14] is due to a number of selectable machines for each operation, while the flexibility of BCJSP is due to the bidirectional circular precedence relations of operations.

To solve BCJSP, this paper introduces a multilevel metaheuristic (called MUL) based on the adaptive parameter control concept [15]. MUL consists of the top-level algorithm (called TOP), the middle-level algorithm (called MID), and the bottom-level algorithm (called BOT). In the MUL's top level, TOP controls the start operation and the operation-precedence-relation direction of every job in BCJSP. TOP is also usable to control the MID's input parameters if requested. In the middle level, MID transfers the start operation and the operation-precedence-relation direction of every job given by TOP into BOT. However, the MID's main function is to control the BOT's input parameters. In the bottom level, BOT uses the start operations and the operation-precedence-relation directions to generate a JSP instance, which is a subproblem of the BCJSP instance. BOT then acts as a local search algorithm for solving the generated JSP instance. MID evolves the BOT's input-parameter values based on feedback from BOT, while TOP evolves the operation-precedence-relation chains and the MID's input-parameter values based on feedback from MID.

The BOT combined with MID is similar to the two-level metaheuristic developed by [16]. A major difference of the MID-BOT combination from the algorithm of [16] is in the solution-decoding procedure. Once the MID-BOT combination is combined with TOP, they all together have become MUL. MUL can be defined as an adaptive multistart iterated local search algorithm for solving BCJSP. There are two variants of MUL proposed in this paper, i.e., the base-specification MUL (called MUL-B) and the top-specification MUL (called MUL-T). The only difference between the two variants is in their TOP-MID relationships. In both MUL-B and MUL-T, their TOPs control the start operation and the operation-precedence-relation direction of every job. In only MUL-T, its TOP also controls the MID's input parameters. Performance of MUL-B and MUL-T was evaluated on the BCJSP instances, modified from the JSP instances of [17, 18, 19]. On the BCJSP instances, MUL-B's and MUL-T's results were compared with each other. Because the existing JSP-solving algorithms can be used to solve BCJSP, their results were also used in the performance comparisons. Note that the existing JSP-solving algorithms mean the algorithms developed for solving JSP in the literature, e.g., [1, 2, 16, 20, 21].

The remainder of this paper is divided into six sections. Section 2 provides an overview of the relevant publications of the research topic. Section 3 describes the job-shop scheduling problem with bidirectional circular precedence constraints (BCJSP). Section 4 presents the procedure of MUL, where the procedures of BOT, MID, and TOP are described in Sections 4.1–4.3, respectively. The differences between MUL-B and MUL-T, as the two variants of MUL, are also described in Section 4. Section 5 presents the experiment's results for MUL-B's and MUL-T's performance evaluations. Section 6 then discusses the experiment's results. Finally, Section 7 concludes the research's findings.

2. Related Works

Metaheuristics can be classified into two categories based on their numbers of solutions used in each iteration, i.e., single-point-based and population-based search algorithms [22]. As its name implies, a single-point-based search algorithm starts with a single solution. Then, it moves from its current solution to another solution repeatedly. Local search is a well-known type of single-point-based search algorithms. A local search algorithm improves its solution gradually within a local region of the solution space. Although a local search algorithm aims to find just a local optimal solution, the algorithm with a good initial solution occasionally finds a global optimal solution.

Iterated local search [23] is another well-known type of single-point-based search algorithms. It can be defined as a local search algorithm that can escape a local region of the solution space. During its exploration, an iterated local search algorithm uses a neighbor operator repeatedly to find a local optimal solution. After that, it tries to escape the current local region into another local region by using a perturbation operator. In general, an iterated local search algorithm starts with a single solution; however, some recent variants, e.g., [24, 25], have multistart properties.

There are three operators, i.e., swap, insert, and inverse, commonly used as a neighbor operator and a perturbation operator [26]. To define these three operators, let h and v be two different random integers from 1 to the number of members in a solution-representing permutation. The swap operator is to swap between the two members in the h-th and the$v$-th positions. The insert operator is to remove a member from the h-th position and then insert it back at the v-th position. The inverse operator is to inverse the sequence of all members from the h-th to the$v$-th positions. Some iterated local search algorithms, e.g., [27, 28], use the swap operator or insert operator multiple times as their perturbation operators.

As mentioned, the population-based search algorithm category is the alternative of the single-point-based search algorithm category. A population-based search algorithm starts with a set (i.e., a population) of solutions instead of a single solution. At each iteration, a population-based search algorithm evolves its population by using the information from its previous iterations. Some population-based search algorithms were purposely developed for solving discrete optimization problems, such as genetic algorithm [29] and ant colony optimization [30]. In contrast, some others were intentionally developed for exploring in real-number search spaces, such as particle swarm optimization [31], differential evolution [32], and cuckoo search [33] algorithms.

A common drawback of most metaheuristics is that there is no single set of input-parameter values performing best for all problem's instances. However, several techniques can be applied for handling such a drawback. One of these techniques is adaptive parameter control [15], where an upper-level metaheuristic acts as a parameter controller for a lower-level metaheuristic. (Note that upper-level metaheuristic is commonly called *metaevolutionary algorithm* [15, 34].) In addition, an upper-level metaheuristic can be applied to control parameters of a being-considered problem for generating its simpler subproblems [35, 36]. For applying the adaptive parameter control technique, the metaheuristics usually have only two levels [4, 15, 16, 37–39]. However, for solving highly complicated problems, they may require more than two levels [35, 36].

A two-level metaheuristic of [37] was developed for solving JSP. It consists of the algorithms named UPLA and LOLA in its upper and lower levels, respectively. LOLA is a local search algorithm exploring in a solution space of parameterized-active schedules (hybrid schedules) [40–42], where each parameterized-active schedule is decoded from an operation-based permutation [29, 43]. UPLA is the population-based search algorithm intentionally developed for being a parameter controller. Its population consists of a number of value combinations of input parameters of LOLA. For updating an input-parameter-value combination, each input-parameter value is iteratively moved by a sum of two changeable opposite-direction vectors. The first vector's direction is toward the memorized best-found value, whereas the second vector's direction is away from it. The magnitudes of these two vectors are generated randomly between zeros and their given maximum values.

The two-level metaheuristic of [16] is a recent variant of [37]. In this variant, MUPLA and LOSAP are the upper-level and lower-level metaheuristics, respectively. LOSAP [16] is a local search algorithm exploring in a probabilistic-based hybrid neighborhood structure. To generate each neighbor solution, LOSAP randomly uses one of the two predetermined neighbor operators by a preassigned probability. (Other applications of randomly using one of two different operators can be found in [44, 45].) Note that while LOLA's solution space is a set of parameterized-active schedules, LOSAP's solution space is just a set of semiactive schedules. It means that the LOSAP's search ability is mainly based on its hybrid neighborhood structure, not based on a special solution space like LOLA. LOSAP has many optional operators proposed for being its perturbation and neighbor operators. In addition, LOSAP uses a different criterion from that of LOLA on accepting a new best-found solution.

MUPLA [16] is a population-based metaheuristic designed to be a parameter controller for LOSAP. Thus, its population consists of a number of value combinations of the LOSAP's input parameters. Each input-parameter-value combination contains specific values of the perturbation operator, the scheduling direction, the ordered pair of two neighbor operators, the probability of selecting a neighbor operator, and the start solution-representing permutation. A major change of MUPLA from UPLA is that each input-parameter-value combination in its population includes a specific start solution-representing permutation. Thus, the MUPLA combined with LOSAP acts as a multistart iterated local search algorithm, while the UPLA combined with LOLA is just an iterated local search algorithm.

3. Problem Definition

BCJSP is an intermediate problem between JSP and OSP; however, it can be explained simpler as a JSP's generalized variant. BCJSP aims to find a feasible schedule that minimizes makespan (i.e., a total length of the schedule). The problem comes with m given machines (i.e., M_1, M_2, . . ., M_m) and n given jobs (i.e., J_1, J_2, . . ., J_n). At the beginning (i.e., time 0), all jobs have already been arrived, and all machines have not yet been occupied. Each job J_i (where $i = 1, 2, \ldots, n$) consists of m operations (i.e., O_{i1}, O_{i2}, . . ., O_{im}). Each operation must be processed by a predetermined machine with a predetermined processing time. Each machine cannot process more than one operation at a time, and it cannot be stopped or paused during processing an operation. In other words, an operation preemption is not allowed. BCJSP differs from JSP in that an operation-precedence-relation chain of each job J_i (where $i = 1, 2, \ldots, n$) is circular and bidirectional, as shown in Figure 1.

Figure 1 shows the relationships of the operations O_{i1}, O_{i2}, . . ., O_{im} of the job J_i in BCJSP. For each job J_i, let the precedence relations of O_{i1}, O_{i2}, . . ., O_{im} be all together connected as a circular chain. It means that any operation from O_{i1} to O_{im} can be selected as the start operation of the job J_i (i.e., the operation processed first in the job J_i). Then, to complete the job J_i, the remaining operations in the circular chain must be processed one-by-one in either a clockwise or counterclockwise direction. In Figure 1, the operations connected together by green arrows present the circular operation-precedence-relation chain in clockwise direction, while those by blue arrows present the chain in counterclockwise direction.

A BCJSP instance can be divided into $(2m)^n$ JSP instances as all of its subproblems. Each subproblem is generated from the BCJSP instance by assigning a specific start operation and a specific operation-precedence-relation direction into every job. To generate a subproblem, let O_{ik} be

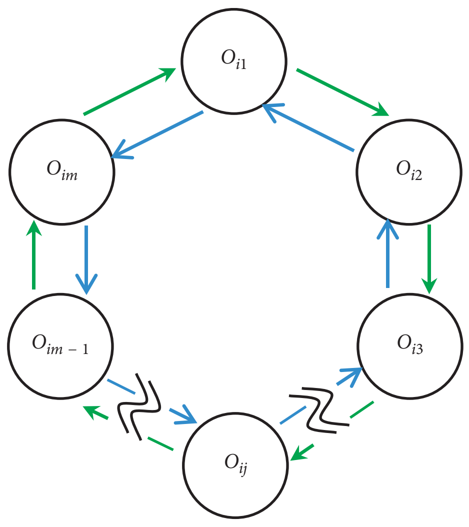


Figure 1: A diagram of the operation-precedence-relation chain of the job J_i in BCJSP.

the start operation of the job J_i, which is selected from any operation of O_{i1}, O_{i2}, ..., O_{im}. Then, let the operation-precedence-relation direction of the job J_i be selected from either clockwise or counterclockwise. If the clockwise direction is selected, let the operation-precedence-relation chain of the job J_i be $O_{ik} \Rightarrow O_{ik+1} \Rightarrow \ldots \Rightarrow O_{im} \Rightarrow O_{i1} \Rightarrow \ldots \Rightarrow O_{ik-1}$; otherwise, let it be $O_{ik} \Rightarrow O_{ik-1} \Rightarrow \ldots \Rightarrow O_{i1} \Rightarrow O_{im} \Rightarrow \ldots \Rightarrow O_{ik+1}$. In this paper, $D \Rightarrow E$ means that D must be finished before E can be started.

To clarify the above paragraph, consider a BCJSP instance that has three machines (i.e., M_1, M_2, M_3) and two jobs (i.e., J_1 and J_2). Each job J_i (where $i = 1$ and 2) then consists of three operations (i.e., O_{i1}, O_{i2}, and O_{i3}). To generate a subproblem, there are six options for the job J_1's operation-precedence-relation chain: (1) $O_{11} \Rightarrow O_{12} \Rightarrow O_{13}$, (2) $O_{12} \Rightarrow O_{13} \Rightarrow O_{11}$, (3) $O_{13} \Rightarrow O_{11} \Rightarrow O_{12}$, (4) $O_{13} \Rightarrow O_{12} \Rightarrow O_{11}$, (5) $O_{12} \Rightarrow O_{11} \Rightarrow O_{13}$, and (6) $O_{11} \Rightarrow O_{13} \Rightarrow O_{12}$. In addition, there are six options for the job J_2's operation-precedence-relation chain: (1) $O_{21} \Rightarrow O_{22} \Rightarrow O_{23}$, (2) $O_{22} \Rightarrow O_{23} \Rightarrow O_{21}$, (3) $O_{23} \Rightarrow O_{21} \Rightarrow O_{22}$, (4) $O_{23} \Rightarrow O_{22} \Rightarrow O_{21}$, (5) $O_{22} \Rightarrow O_{21} \Rightarrow O_{23}$, and (6) $O_{21} \Rightarrow O_{23} \Rightarrow O_{22}$. Of each job, the first three options are generated in clockwise direction, while the last three options are generated in counterclockwise direction. Based on the six options of each job, this BCJSP instance can be divided into 36 JSP instances as all of its subproblems.

BCJSP is a generalization of JSP and is also much more complex than JSP. Every single BCJSP instance can be divided into $(2m)^n$ JSP instances as all of its subproblems. Because JSP with $m = n = 3$ has been proven to be NP-hard [5], BCJSP with $m \geq 3$ and $n \geq 3$ thus belongs to a class of NP-hard problems. In the literature, no algorithms excepting MUL-B and MUL-T have been developed for BCJSP. Although the existing JSP-solving algorithms without modifications can be used to solve BCJSP, they may not perform well on BCJSP. This is because, with the same m and n, a solution space of BCJSP is much larger than that of JSP.

4. Methods

As mentioned, MUL represents the proposed multilevel metaheuristic for solving BCJSP. It consists of BOT, MID, and TOP algorithms in its bottom, middle, and top levels, respectively. BOT is a local search algorithm, modified from LOSAP [16], for solving subproblems of the BCJSP instance. Each subproblem is a JSP instance modified from the BCJSP instance by assigning a specific start operation and a specific operation-precedence-relation direction into every job. MID, as a variant of MUPLA [16], is a population-based metaheuristic for controlling BOT's input parameters. Another function of MID is to transfer the start operation and the operation-precedence-relation direction of every job from TOP into BOT. TOP is a population-based metaheuristic developed based on the framework of MUPLA [16]. TOP is used to control the start operation and the operation-precedence-relation direction of every job in the BCJSP instance. If requested, TOP can also control the MID's input parameters as an extra optional function.

In this paper, there are two variants of MUL, i.e., the base-specification MUL (MUL-B) and the top-specification MUL (MUL-T). MUL-B is the MUL whose TOP controls only the start operation and the operation-precedence-relation direction of every job in BCJSP. MUL-T is the MUL whose TOP controls the start operation and operation-precedence-relation direction of every job and also the MID's input parameters. The details of BOT, MID, and TOP are described in Sections 4.1–4.3, respectively.

4.1. BOT Algorithm

4.1. BOT Algorithm. BOT is a local search algorithm for solving subproblems of the being-solved BCJSP instance; each subproblem is a JSP instance. BOT generates each subproblem from the BCJSP instance by assigning a specific start operation and a specific operation-precedence-relation direction into every job. Let A_i and B_i represent the start operation and the operation-precedence-relation direction, respectively, of the job J_i (where $i = 1, 2, \ldots, n$). For each job J_i, A_i can be any operation selected from O_{i1}, O_{i2}, ..., O_{im}; in addition, B_i can be either a clockwise or counterclockwise direction. In this paper, BOT receives A_i and B_i from TOP via MID.

To illustrate how to use A_i and B_i, assume $A_1 = O_{12}$, $A_2 = O_{23}$, $A_3 = O_{31}$, $B_1 =$ counterclockwise, $B_2 =$ clockwise, and $B_3 =$ counterclockwise be assigned into a 2-machine/3-job BCJSP instance. By assigning $A_1 = O_{12}$ and $B_1 =$ counterclockwise, the job J_1's operation-precedence-relation chain becomes $O_{12} \Rightarrow O_{11} \Rightarrow O_{13}$. By assigning $A_2 = O_{23}$ and $B_2 =$ clockwise, the job J_2's operation-precedence-relation chain becomes $O_{23} \Rightarrow O_{21} \Rightarrow O_{22}$. By assigning $A_3 = O_{31}$ and $B_3 =$ counterclockwise, the job J_3's operation-precedence-relation chain becomes $O_{31} \Rightarrow O_{33} \Rightarrow O_{32}$. As a result, a subproblem of the BCJSP instance in the form of JSP has successfully been generated.

After BOT has successfully generated a JSP instance (which is a subproblem of the BCJSP specified by A_i and B_i), BOT acts as a local search algorithm for solving the JSP instance. BOT uses operation-based permutations [29, 43] to represent semiactive schedules, where an operation-based permutation is a permutation with m repetitions of the numbers 1, 2, . . ., n. However, the transformation into a schedule for the BCJSP's subproblem differs from the transformation used by [29, 43] for the classical JSP. The difference is due to the specific order of operations in the operation-precedence-relation chain assigned by A_i and B_i. For example, on a 2-machine/3-job BCJSP instance, assume a given permutation be (2, 1, 2, 3, 1, 1, 3, 3, 2). In addition, assume $A_1 = O_{12}$, $A_2 = O_{23}$, $A_3 = O_{31}$, $B_1 =$ counterclockwise, $B_2 =$ clockwise, and $B_3 =$ counterclockwise. From the given permutation, BOT constructs a semiactive schedule in the order of O_{23}, O_{12}, O_{21}, O_{31}, O_{11}, O_{13}, O_{33}, O_{32}, and O_{22}. Notice that the schedule is not constructed in the order of O_{21}, O_{11}, O_{22}, O_{31}, O_{12}, O_{13}, O_{32}, O_{33}, and O_{23} like that used for the classical JSP.

BOT, as modified from LOSAP [16], improves its solutions by using *PT* and *PN*. Let *PT* represent the perturbation operator, and let $PN \equiv (PN_f, PN_s)$ represent the ordered pair of the first neighbor operator (PN_f) and the second neighbor operator (PN_s). BOT offers five options for *PT*, i.e., n-medium swap, n-large swap, n-medium inverse, n-large insert, and n-medium insert. In addition, BOT offers four options for $PN \equiv (PN_f, PN_s)$, i.e., (1-small inverse, 1-medium insert), (1-large swap, 1-large insert), (1-medium swap, 1-medium insert), and (1-small swap, 1-small insert).

In the names of the above-mentioned operators, the number in front of the hyphen sign indicates the number of repeated uses of the operator mentioned in back of the hyphen sign. For example, the n-medium inverse operator is to use the medium inverse operator n times on a permutation. In addition, the words *small*, *medium*, and *large* are used to restrict the value of v from h (note that the uses of h and v for operators are already reviewed in Section 2). Let h and v be two different random integers within [1, mn], where mn is the number of all operations in the BCJSP instance. The words *small* and *medium* then provide additional limitations on generating v as follows: v must be generated within $[h - 4, h + 4]$ for *small*, while v must be generated within $[h - 0.2mn, h + 0.2mn]$ for *medium*. For *large*, there are no additional limitations.

The procedure of BOT is presented in Algorithm 1, and its flowchart is presented in Figure 2. The parameters and abbreviations used in Algorithm 1 and Figure 2 are defined as follows:

(i) Let m and n, respectively, be the number of all machines and the number of all jobs in BCJSP. Thus, mn is the number of all operations in BCJSP.

(ii) Let $A_i \in \{O_{i1}, O_{i2}, \ldots, O_{im}\}$ represent the start operation of the job J_i.

(iii) Let $B_i \in$ {clockwise, counterclockwise} represent the operation-precedence-relation direction of the job J_i.

(iv) Let *PT* and *P* stand for the perturbation operator and the start operation-based permutation, respectively.

(v) Let $PN \equiv (PN_f, PN_s)$ represent the ordered pair of the first neighbor operator (PN_f) and the second neighbor operator (PN_s).

(vi) Let *PR* be the probability of selecting the first neighbor operator (PN_f) of *PN*. Consequently, the probability of selecting the second neighbor operator (PN_s) is equal to unity minus *PR*.

(vii) Let P_0 (which is a permutation with m repetitions of the numbers 1, 2, . . ., n) stand for the current best-found operation-based permutation.

(viii) Let Π_0 be the permutation of mn operations decoded from P_0.

(ix) Let S_0 stand for the current best-found schedule decoded from Π_0. In addition, let $Makespan(S_0)$ represent the makespan of S_0.

(x) Let P_1 (which is a permutation with m repetitions of the numbers 1, 2, . . ., n) stand for the current neighbor operation-based permutation.

(xi) Let Π_1 be the permutation of mn operations decoded from P_1.

(xii) Let S_1 stand for the current neighbor schedule decoded from Π_1. In addition, let $Makespan(S_1)$ represent the makespan of S_1.

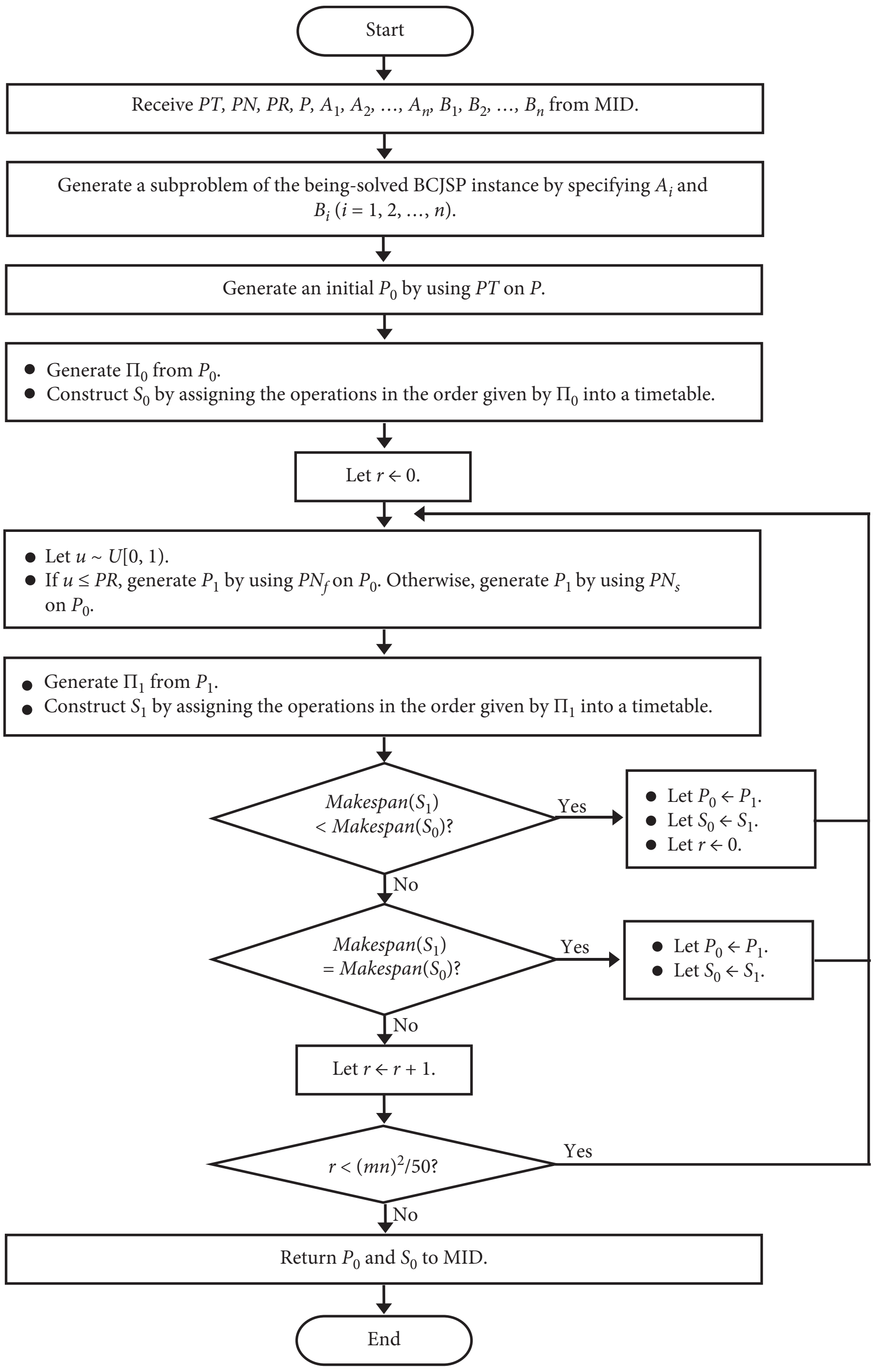
Although there are two proposed variants of MUL (i.e., MUL-B and MUL-T), the procedures of BOTs in MUL-B and MUL-T are both identical to Algorithm 1. The differences between MUL-B and MUL-T are in their MIDs and TOPs.

4.2. MID Algorithm

4.2. MID Algorithm. MID is a population-based metaheuristic modified from MUPLA [16]. It is a channel to transfer A_i and B_i (where $i = 1, 2, \ldots, n$) from TOP into BOT. However, a main function of MID is to be a parameter controller for BOT. At the t-th iteration, the MID's population contains N members, i.e., $C_1(t)$, $C_2(t)$, . . ., $C_N(t)$. For $g = 1$ to N, let $C_g(t) \equiv (pt_g(t), pn_g(t), pr_g(t), p_g(t))$ represent a value combination of the BOT's input parameters *PT*, *PN*, *PR*, and *P*, respectively. Each of $pt_g(t+1)$, $pn_g(t+1)$, and $pr_g(t+1)$ is updated from its old value via two opposite-direction vectors. The first vector's direction is toward the memorized best-found value, whereas the second vector's direction is away from it. Differently, $p_g(t+1)$ is set to the final operation-based permutation returned from the BOT with $C_g(t)$-given input-parameter values.

The procedure of MID is presented in Algorithm 2, and its flowchart is presented in Figure 3. The following list presents the definitions of parameters and abbreviations used in Algorithm 2 and Figure 3. In addition, the transformation (i.e., decoding method) of each member of $C_g(t)$ is also given:

(i) Let n be the number of all jobs in BCJSP.



Start
Receive $PT, PN, PR, P, A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n$ from MID.
Generate a subproblem of the being-solved BCJSP instance by specifying A_i and B_i ($i = 1, 2, \ldots, n$).
Generate an initial P_0 by using PT on P.
● Generate Π_0 from P_0.
● Construct S_0 by assigning the operations in the order given by Π_0 into a timetable.
Let $r \leftarrow 0$.
● Let $u \sim U[0, 1)$.
● If $u \leq PR$, generate P_1 by using PN_f on P_0. Otherwise, generate P_1 by using PN_s on P_0.
● Generate Π_1 from P_1.
● Construct S_1 by assigning the operations in the order given by Π_1 into a timetable.
$Makespan(S_1) < Makespan(S_0)$?
Yes
● Let $P_0 \leftarrow P_1$.
● Let $S_0 \leftarrow S_1$.
● Let $r \leftarrow 0$.
No
$Makespan(S_1) = Makespan(S_0)$?
Yes
● Let $P_0 \leftarrow P_1$.
● Let $S_0 \leftarrow S_1$.
No
Let $r \leftarrow r + 1$.
$r < (mn)^2/50$?
Yes
No
Return P_0 and S_0 to MID.
End

FIGURE 2: A flowchart of BOT.

(ii) Let N be the number of all members in the MID's population.

(iii) Let $C_g(t) \equiv (pt_g(t), pn_g(t), pr_g(t), p_g(t))$ represent the g-th member (where $g = 1, 2, \ldots, N$) in the MID's population at the t-th iteration. In addition, let $Score(C_g(t))$ stand for the performance score of $C_g(t)$. Note that the lower the performance score, the better the performance.

(iv) Let $pt_g(t) \in \mathbb{R}$ represent the perturbation operator (PT) of BOT. Equation (1) is used to transform $pt_g(t)$ into PT.

(v) Let $pn_g(t) \in \mathbb{R}$ represent the ordered pair of the first and second neighbor operators (PN) of BOT. Equation (2) is used to transform $pn_g(t)$ into PN.

(vi) Let $pr_g(t) \in \mathbb{R}$ represent the probability of selecting the first neighbor operator (PR) of BOT.

```
(1) Receive values of BOT's input parameters (i.e., $PT$, $PN$, $PR$, and $P$) from MID. In addition, receive $A_i$ and $B_i$ (where $i = 1, 2, \ldots, n$) from MID.
(2) To generate a subproblem of the BCJSP instance, let the start operation and the operation-precedence-relation direction be assigned to every job by using Steps 2.1 to 2.4.
    (2.1) Let $i \leftarrow 1$.
    (2.2) Let $O_{ik} \leftarrow A_i$.
    (2.3) If $B_i$ = clockwise, let the job $J_i$'s operation-precedence-relation chain be $O_{ik} \Rightarrow O_{ik+1} \Rightarrow \ldots \Rightarrow O_{im} \Rightarrow O_{i1} \Rightarrow O_{i2} \Rightarrow \ldots \Rightarrow O_{ik-1}$. Otherwise, let it be $O_{ik} \Rightarrow O_{ik-1} \Rightarrow \ldots \Rightarrow O_{i1} \Rightarrow O_{im} \Rightarrow O_{im-1} \Rightarrow \ldots \Rightarrow O_{ik+1}$.
    (2.4) If $i < n$, let $i \leftarrow i + 1$ and repeat from Step 2.2. Otherwise, go to Step 3.
(3) Generate an initial $P_0$ by using $PT$ on $P$. Then, transform $P_0$ into $S_0$ by using Steps 3.1 and 3.2.
    (3.1) Generate $\Pi_0$ by changing the $j$-th repetition of the number $i$ in $P_0$ into the operation listed in the $j$-th order of the job $J_i$'s operation-precedence-relation chain. (Note that the job $J_i$'s operation-precedence-relation chain is given in Step 2.)
    (3.2) Construct $S_0$ by assigning the operations in the order given by $\Pi_0$ (from left to right) into a timetable. In the timetable, each operation must be assigned to its predetermined machine at its earliest possible start time. (Note that the earliest possible start time of each operation is the maximum between the completion time of its immediate-predecessor operation in its job and the completion time of the current latest operation on its machine.)
(4) Find a local optimal schedule by using Steps 4.1 to 4.3.
    (4.1) Let $r \leftarrow 0$.
    (4.2) Randomly generate $u \sim U[0, 1)$. If $u \leq PR$, then generate $P_1$ by using $PN_f$ on $P_0$; otherwise, generate $P_1$ by using $PN_s$ on $P_0$. Then, transform $P_1$ into $S_1$ by using Steps 4.2.1 and 4.2.2.
        (4.2.1) Generate $\Pi_1$ by changing the $j$-th repetition of the number $i$ in $P_1$ into the operation listed in the $j$-th order of the job $J_i$'s operation-precedence-relation chain.
        (4.2.2) Construct $S_1$ by assigning the operations in the order given by $\Pi_1$ (from left to right) into a timetable. In the timetable, each operation must be assigned to its predetermined machine at its earliest possible start time.
    (4.3) Update $P_0$, $S_0$, and $r$ by using Steps 4.3.1 to 4.3.3.
        (4.3.1) If $Makespan(S_1) < Makespan(S_0)$, let $P_0 \leftarrow P_1$ and $S_0 \leftarrow S_1$, and repeat from Step 4.1.
        (4.3.2) If $Makespan(S_1) = Makespan(S_0)$, let $P_0 \leftarrow P_1$ and $S_0 \leftarrow S_1$, and repeat from Step 4.2.
        (4.3.3) If $Makespan(S_1) > Makespan(S_0)$, let $r \leftarrow r + 1$. Then, repeat from Step 4.2 if $r < (mn)^2/50$; otherwise, go to Step 5.
(5) Return $P_0$ and $S_0$ as the final (best-found) operation-based permutation and the final (best-found) schedule, respectively, to MID.
```

Algorithm 1: The procedure of BOT.

In the transformation, let $PR \leftarrow pr_g(t)$ if $0 \leq pr_g(t) \leq 1$; in addition, let $PR \leftarrow 0$ if $pr_g(t) < 0$, and let $PR \leftarrow 1$ if $pr_g(t) > 1$.

(vii) Let $p_g(t)$ represent the start operation-based permutation (P) of BOT, and let it be a member of all possible operation-based permutations. In the transformation, let $P \leftarrow p_g(t)$.

(viii) For updating $pt_g(t+1)$, let y_{pt} and w_{pt} be the controlling weights of the maximum magnitudes of the first and second vectors, respectively. However, if $pt_g(t) = pt_{\text{best}}$, let w_{pt} be the controlling weights of the maximum magnitudes of both vectors.

(ix) For updating $pn_g(t+1)$, let y_{pn} and w_{pn} be the controlling weights of the maximum magnitudes of the first and second vectors, respectively. However, if $pn_g(t) = pn_{\text{best}}$, let w_{pn} be the controlling weights of the maximum magnitudes of both vectors.

(x) For updating $pr_g(t+1)$, let y_{pr} and w_{pr} be the controlling weights of the maximum magnitudes of the first and second vectors, respectively. However, if $pr_g(t) = pr_{\text{best}}$, let w_{pr} be the controlling weights of the maximum magnitudes of both vectors.

(xi) Let P_0 and S_0, respectively, stand for the final operation-based permutation and the final schedule returned from BOT. In addition, let $Makespan(S_0)$ stand for the makespan of S_0.

(xii) Let $C_{\text{best}} \equiv (pt_{\text{best}}, pn_{\text{best}}, pr_{\text{best}}, p_{\text{best}})$ and S_{best} stand for the best $C_g(t)$ and the best S_0, respectively, ever found by the population. In addition, let $Score(C_{\text{best}})$ stand for the performance score of C_{best}.

```
(1) Receive $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and $w_{pr}$ from TOP.
(2) Let $t \leftarrow 1$ and $Score(C_{\text{best}}) \leftarrow +\infty$.
(3) Generate $C_g(t)$ by randomly generating $pt_g(t)$, $pn_g(t)$, and $pr_g(t) \sim U[0,1)$ and randomly generating $p_g(t)$ from any possible
    operation-based permutation ($g = 1, 2, \ldots, N$).
(4) Evaluate $Score(C_g(t))$ and update $p_g(t+1)$, $C_{\text{best}}$, and $S_{\text{best}}$ by using Steps 4.1 to 4.6.
    (4.1) Let $g \leftarrow 1$.
    (4.2) Transform $C_g(t)$ into the values of $PT$, $PN$, $PR$, and $P$ of BOT.
    (4.3) Execute BOT with the values of $PT$, $PN$, $PR$, and $P$ (taken from Step 4.2) and the values of $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n$ (taken
    from Step 1). This is done for receiving $P_0$ and $S_0$ from BOT.
    (4.4) Let $Score(C_g(t)) \leftarrow Makespan(S_0)$, and let $p_g(t+1) \leftarrow P_0$.
    (4.5) If $Score(C_g(t)) \le Score(C_{\text{best}})$, let $C_{\text{best}} \leftarrow C_g(t)$, $Score(C_{\text{best}}) \leftarrow Score(C_g(t))$, and $S_{\text{best}} \leftarrow S_0$.
    (4.6) If $g < N$, let $g \leftarrow g+1$ and repeat from Step 4.2. Otherwise, go to Step 5.
(5) Update $pt_g(t+1)$, $pn_g(t+1)$, and $pr_g(t+1)$, where $g = 1, 2, \ldots, N$, by using Steps 5.1 to 5.3.
    (5.1) Let $g \leftarrow 1$.
    (5.2) Generate $pt_g(t+1)$, $pn_g(t+1)$, and $pr_g(t+1)$ by below three equations, respectively. Let $u_1$ and $u_2 \sim U[0,1)$.
        $pt_g(t+1) = \begin{cases} pt_g(t) + (0.02 + 0.01y_{pt})u_1 - (0.005 + 0.01w_{pt})u_2 & \text{if } pt_g(t) < pt_{\text{best}}, \\ pt_g(t) - (0.02 + 0.01y_{pt})u_1 + (0.005 + 0.01w_{pt})u_2 & \text{if } pt_g(t) > pt_{\text{best}}, \\ pt_g(t) + (0.005 + 0.01w_{pt})u_1 - (0.005 + 0.01w_{pt})u_2 & \text{if } pt_g(t) = pt_{\text{best}}. \end{cases}$
        $pn_g(t+1) = \begin{cases} pn_g(t) + (0.02 + 0.01y_{pn})u_1 - (0.005 + 0.01w_{pn})u_2 & \text{if } pn_g(t) < pn_{\text{best}}, \\ pn_g(t) - (0.02 + 0.01y_{pn})u_1 + (0.005 + 0.01w_{pn})u_2 & \text{if } pn_g(t) > pn_{\text{best}}, \\ pn_g(t) + (0.005 + 0.01w_{pn})u_1 - (0.005 + 0.01w_{pn})u_2 & \text{if } pn_g(t) = pn_{\text{best}}. \end{cases}$
        $pr_g(t+1) = \begin{cases} pr_g(t) + (0.02 + 0.01y_{pr})u_1 - (0.005 + 0.01w_{pr})u_2 & \text{if } pr_g(t) < pr_{\text{best}}, \\ pr_g(t) - (0.02 + 0.01y_{pr})u_1 + (0.005 + 0.01w_{pr})u_2 & \text{if } pr_g(t) > pr_{\text{best}}, \\ pr_g(t) + (0.005 + 0.01w_{pr})u_1 - (0.005 + 0.01w_{pr})u_2 & \text{if } pr_g(t) = pr_{\text{best}}. \end{cases}$
    (5.3) If $g < N$, let $g \leftarrow g+1$ and repeat from Step 5.2. Otherwise, go to Step 6.
(6) If the stopping criterion is not met, let $t \leftarrow t+1$ and repeat from Step 4. Otherwise, return $S_{\text{best}}$ to TOP.
```

Algorithm 2: The procedure of MID.

$$PT = \begin{cases} n\text{-medium swap if } pt_g(t) < 0.20, \\ n\text{-large swap if } 0.20 \le pt_g(t) < 0.40, \\ n\text{-medium inverse if } 0.40 \le pt_g(t) < 0.60, \\ n\text{-large insert if } 0.60 \le pt_g(t) < 0.80, \\ n\text{-medium insert if } pt_g(t) \ge 0.80, \end{cases} \tag{1}$$

$$PN = \begin{cases} (1\text{-small inverse, } 1\text{-medium insert}) & \text{if } pn_g(t) < 0.25, \\ (1\text{-large swap, } 1\text{-large insert}) & \text{if } 0.25 \le pn_g(t) < 0.50, \\ (1\text{-medium swap, } 1\text{-medium insert}) & \text{if } 0.50 \le pn_g(t) < 0.75, \\ (1\text{-small swap, } 1\text{-small insert}) & \text{if } pn_g(t) \ge 0.75. \end{cases} \tag{2}$$

In Algorithm 2, MID starts its procedure by receiving A_i, B_i (where $i = 1, 2, \ldots, n$), and its input-parameter values (i.e., $y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and w_{pr}) from TOP. MID assigns $t \leftarrow 1$ and $Score(C_{\text{best}}) \leftarrow +\infty$; then, it generates $C_g(t)$ randomly. To solve a BCJSP subproblem specified by A_i and B_i, MID then starts a repeated loop by executing BOT N times. In the g-th execution (where $g = 1, 2, \ldots, N$), BOT is executed with $C_g(t)$-given parameter values to return P_0 and S_0; then, let $Score(C_g(t)) \leftarrow Makespan(S_0)$ and $p_g(t+1) \leftarrow P_0$. If MID finds any $C_g(t)$ better than or equal to C_{best}, then let this $C_g(t)$ and its corresponding S_0 become a new C_{best} and a new S_{best}, respectively. After that, MID completes $C_g(t+1)$ by using the two opposite-direction vectors to generate each of $pt_g(t+1)$, $pn_g(t+1)$, and $pr_g(t+1)$. If the stopping criterion is not met, MID assigns $t \leftarrow t+1$ and starts the repeated loop's next round.

As mentioned earlier, there are two variants of MUL, i.e., the base-specification MUL (MUL-B) and the top-specification MUL (MUL-T). The difference between MUL-B and MUL-T in their MIDs is given as follows. In MUL-T, the input parameters (i.e., $y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and w_{pr}) of its MID are controlled by TOP. This means MID in MUL-T is identical to Algorithm 2. Differently, these input-parameter values of MID in MUL-B are constants. The procedure of MID in MUL-B is thus modified from Algorithm 2 by removing y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}, and w_{pr} from Step 1 and

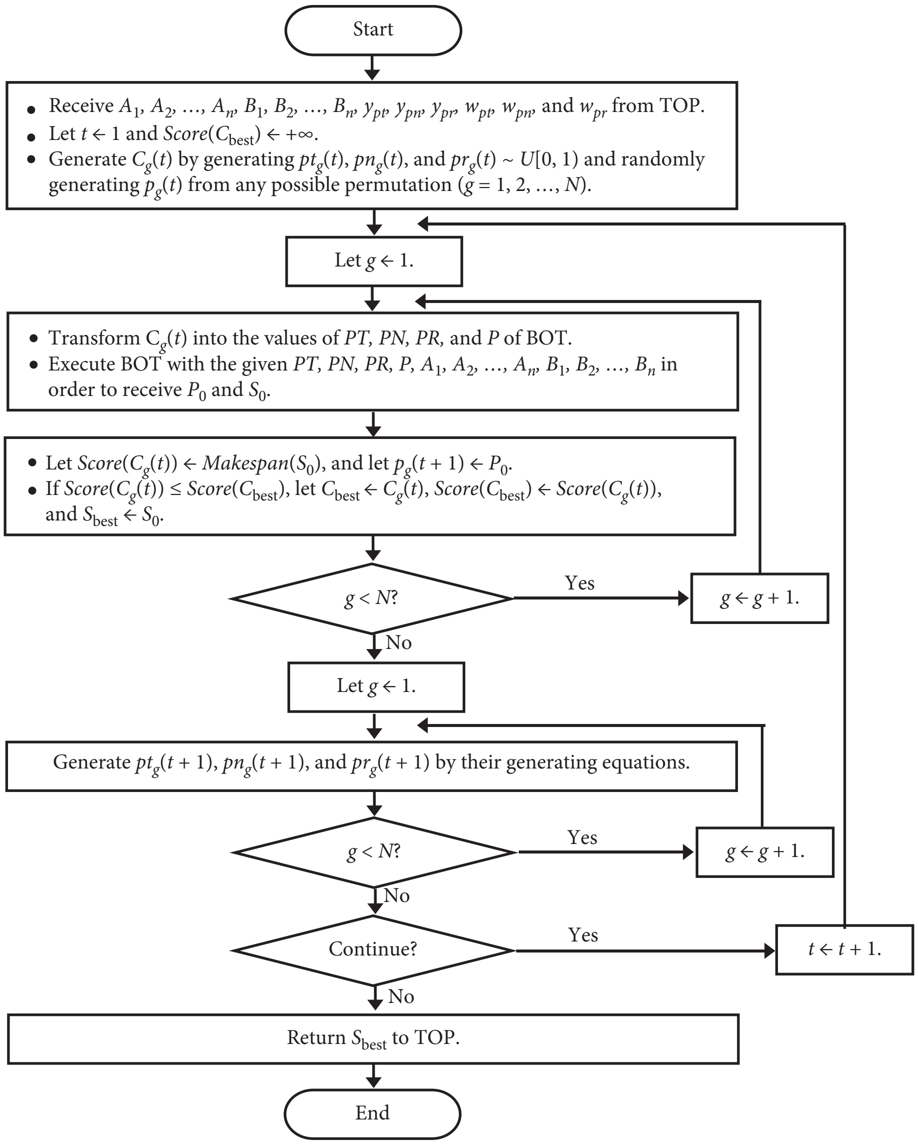

Start
Receive $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and w_{pr} from TOP.
Let $t \leftarrow 1$ and $Score(C_{best}) \leftarrow +\infty$.
Generate $C_g(t)$ by generating $pt_g(t)$, $pn_g(t)$, and $pr_g(t) \sim U[0, 1)$ and randomly generating $p_g(t)$ from any possible permutation ($g = 1, 2, \ldots, N$).
Let $g \leftarrow 1$.
Transform $C_g(t)$ into the values of PT, PN, PR, and P of BOT.
Execute BOT with the given $PT, PN, PR, P, A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n$ in order to receive P_0 and S_0.
Let $Score(C_g(t)) \leftarrow Makespan(S_0)$, and let $p_g(t+1) \leftarrow P_0$.
If $Score(C_g(t)) \le Score(C_{best})$, let $C_{best} \leftarrow C_g(t)$, $Score(C_{best}) \leftarrow Score(C_g(t))$, and $S_{best} \leftarrow S_0$.
$g < N$?
Yes
$g \leftarrow g + 1$.
No
Let $g \leftarrow 1$.
Generate $pt_g(t+1)$, $pn_g(t+1)$, and $pr_g(t+1)$ by their generating equations.
$g < N$?
Yes
$g \leftarrow g + 1$.
No
Continue?
Yes
$t \leftarrow t + 1$.
No
Return S_{best} to TOP.
End

FIGURE 3: A flowchart of MID.

changing y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}, and w_{pr} in Step 5.2 to constants. In this paper, the values of y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}, and w_{pr} of MID in MUL-B all are set to 0.5, as mentioned again in Section 5.

4.3. TOP Algorithm. Like MID, TOP is developed based on the framework of MUPLA [16]. The main function of TOP is to control the start operation and the operation-precedence-relation direction of every job in the being-solved BCJSP instance. As previously mentioned, A_i and B_i represent the start operation and the operation-precedence-relation direction, respectively, of the job J_i (where $i = 1, 2, \ldots, n$, and n is the number of all jobs in BCJSP). After assigning A_i and B_i, the BCJSP instance has become a JSP instance that is a subproblem of the BCJSP instance. In addition to the main function, TOP can control the MID's input parameters, i.e., y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}, and w_{pr}, as its optional extra function.

At the τ-th iteration, the TOP's population contains M members, i.e., $\zeta_1(\tau), \zeta_2(\tau), \ldots, \zeta_M(\tau)$. For $q = 1$ to M, let $\zeta_q(\tau) \equiv ((\alpha_{1q}(\tau), \alpha_{2q}(\tau), \ldots, \alpha_{nq}(\tau)), \beta_{1q}(\tau), \beta_{2q}(\tau), \ldots, \beta_{nq}(\tau), \gamma_{1q}(\tau), \gamma_{2q}(\tau), \ldots, \gamma_{6q}(\tau))$ represent a value combination of $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and w_{pr}, respectively. Each member of $\zeta_q(\tau+1)$, such as $\alpha_{1q}(\tau+1)$, is usually updated from its old value via two opposite-direction vectors; however, it is occasionally regenerated by a reinitialization. For the two opposite-direction vectors, the first vector's direction is toward the memorized best-found value, whereas the second vector's direction is away from it.

The procedure of TOP is presented in Algorithm 3, and its flowchart is presented in Figure 4. The following list shows the definitions of parameters and abbreviations used in Algorithm 3 and Figure 4. In addition, the transformation (i.e., decoding method) of each member of $\zeta_q(\tau)$ is also given:

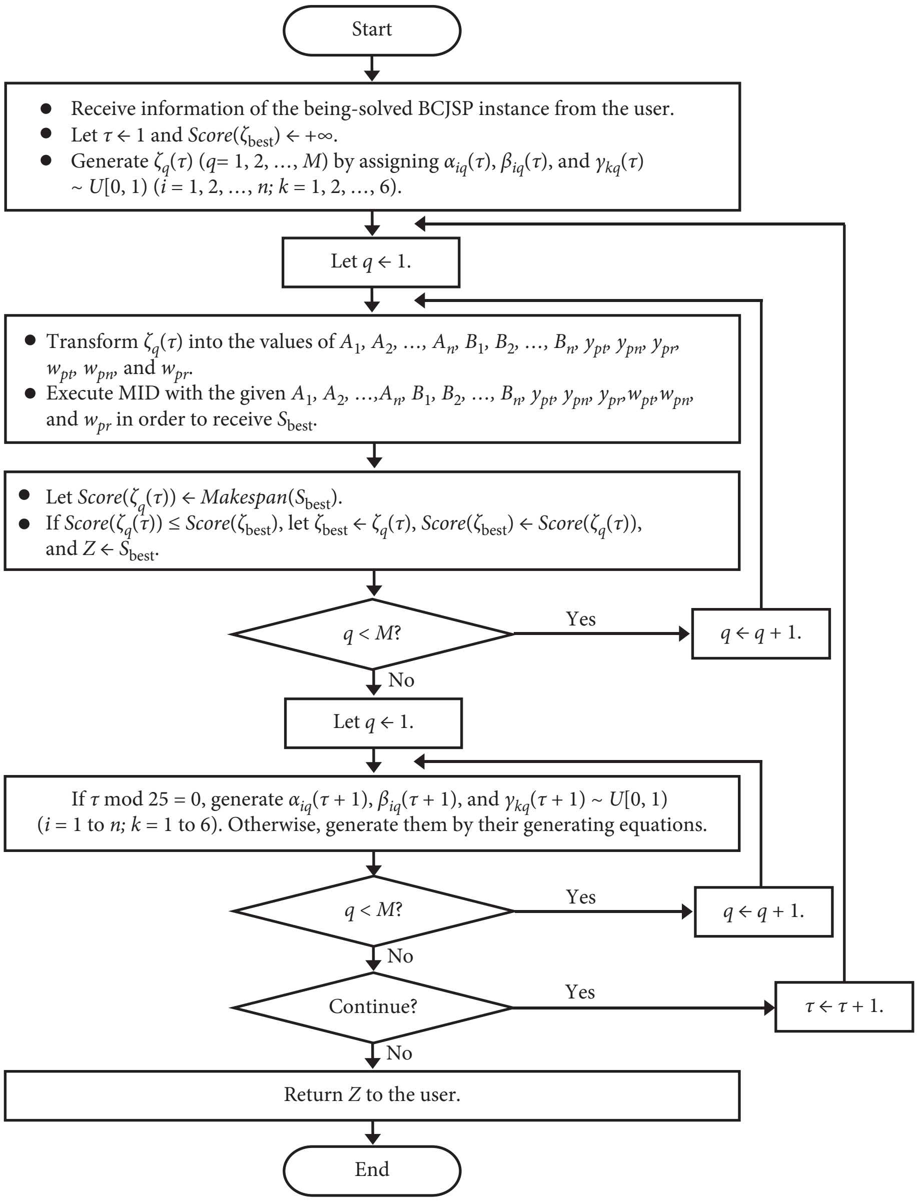

Start
Receive information of the being-solved BCJSP instance from the user.
Let $\tau \leftarrow 1$ and $Score(\zeta_{\text{best}}) \leftarrow +\infty$.
Generate $\zeta_q(\tau)$ ($q = 1, 2, \ldots, M$) by assigning $\alpha_{iq}(\tau)$, $\beta_{iq}(\tau)$, and $\gamma_{kq}(\tau) \sim U[0, 1)$ ($i = 1, 2, \ldots, n$; $k = 1, 2, \ldots, 6$).
Let $q \leftarrow 1$.
Transform $\zeta_q(\tau)$ into the values of $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and w_{pr}.
Execute MID with the given $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and w_{pr} in order to receive S_{best}.
Let $Score(\zeta_q(\tau)) \leftarrow Makespan(S_{\text{best}})$.
If $Score(\zeta_q(\tau)) \leq Score(\zeta_{\text{best}})$, let $\zeta_{\text{best}} \leftarrow \zeta_q(\tau)$, $Score(\zeta_{\text{best}}) \leftarrow Score(\zeta_q(\tau))$, and $Z \leftarrow S_{\text{best}}$.
$q < M$?
Yes
$q \leftarrow q + 1$.
No
Let $q \leftarrow 1$.
If τ mod 25 = 0, generate $\alpha_{iq}(\tau + 1)$, $\beta_{iq}(\tau + 1)$, and $\gamma_{kq}(\tau + 1) \sim U[0, 1)$ ($i = 1$ to n; $k = 1$ to 6). Otherwise, generate them by their generating equations.
$q < M$?
Yes
$q \leftarrow q + 1$.
No
Continue?
Yes
$\tau \leftarrow \tau + 1$.
No
Return Z to the user.
End

FIGURE 4: A flowchart of TOP.

(i) Let m and n, respectively, be the number of all machines and the number of all jobs in the being-solved BCJSP.

(ii) Let A_i and B_i, respectively, stand for the start operation and the operation-precedence-relation direction of the job J_i.

(iii) Let M be the number of all members in the TOP's population.

(iv) Let $\zeta_q(\tau) \equiv ((\alpha_{1q}(\tau), \alpha_{2q}(\tau), \ldots, \alpha_{nq}(\tau)), \beta_{1q}(\tau), \beta_{2q}(\tau), \ldots, \beta_{nq}(\tau), \gamma_{1q}(\tau), \gamma_{2q}(\tau), \ldots, \gamma_{6q}(\tau))$ represent the q-th member (where $q = 1, 2, \ldots, M$) in the TOP's population at the τ-th iteration. In addition, let $Score(\zeta_q(\tau))$ stand for the performance score of $\zeta_q(\tau)$. Note that the lower the performance score, the better the performance.

(v) Let $\alpha_{iq}(\tau) \in \mathbb{R}$ represent A_i (where $i = 1, 2, \ldots, n$). To transform $\alpha_{iq}(\tau)$ into A_i, let $k \leftarrow$ the integer part of $m\alpha_{iq}(\tau) + 1$. After that, reassign $k \leftarrow 1$ if $k < 1$, and reassign $k \leftarrow m$ if $k > m$. Finally, let $A_i \leftarrow O_{ik}$.

(vi) Let $\beta_{iq}(\tau) \in \mathbb{R}$ represent B_i (where $i = 1, 2, \ldots, n$). In the transformation, let $B_i \leftarrow$ clockwise if $\beta_{iq}(\tau) < 0.5$, and let $B_i \leftarrow$ counterclockwise otherwise.

(vii) Let $\gamma_{1q}(\tau)$, $\gamma_{2q}(\tau)$, $\gamma_{3q}(\tau)$, $\gamma_{4q}(\tau)$, $\gamma_{5q}(\tau)$, and $\gamma_{6q}(\tau) \in \mathbb{R}$ represent the MID's y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}, and w_{pr}, respectively. In their transformations, let $y_{pt} \leftarrow \gamma_{1q}(\tau)$, $y_{pn} \leftarrow \gamma_{2q}(\tau)$, $y_{pr} \leftarrow \gamma_{3q}(\tau)$, $w_{pt} \leftarrow \gamma_{4q}(\tau)$, $w_{pn} \leftarrow \gamma_{5q}(\tau)$, and $w_{pr} \leftarrow \gamma_{6q}(\tau)$.

(viii) Let S_{best} and $Makespan(S_{\text{best}})$ stand for the best schedule returned from MID and its makespan, respectively.

```
(1) Receive information of the being-solved BCJSP instance from the user.
(2) Let $\tau \leftarrow 1$ and $Score(\zeta_{\text{best}}) \leftarrow +\infty$.
(3) Generate $\zeta_q(\tau)$, where $q = 1, 2, \ldots, M$, by randomly generating $\alpha_{iq}(\tau)$, $\beta_{iq}(\tau)$, and $\gamma_{kq}(\tau) \sim U[0,1)$ $(i = 1, 2, \ldots, n;\ k = 1, 2, \ldots, 6)$.
(4) Evaluate $Score(\zeta_q(\tau))$ and update $\zeta_{\text{best}}$ and $Z$ by using Steps 4.1 to 4.6.
    (4.1) Let $q \leftarrow 1$.
    (4.2) Transform $\zeta_q(\tau)$ into the values of $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and $w_{pr}$.
    (4.3) Execute MID with the values of $A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}$, and $w_{pr}$ taken from Step 4.2. This is done for receiving $S_{\text{best}}$ from MID.
    (4.4) Let $Score(\zeta_q(\tau)) \leftarrow Makespan(S_{\text{best}})$.
    (4.5) If $Score(\zeta_q(\tau)) \le Score(\zeta_{\text{best}})$, then let $\zeta_{\text{best}} \leftarrow \zeta_q(\tau)$, $Score(\zeta_{\text{best}}) \leftarrow Score(\zeta_q(\tau))$, and $Z \leftarrow S_{\text{best}}$.
    (4.6) If $q < M$, then let $q \leftarrow q + 1$ and repeat from Step 4.2. Otherwise, go to Step 5.
(5) Update $\zeta_q(\tau+1)$, where $q = 1, 2, \ldots, M$, by using Steps 5.1 to 5.3.
    (5.1) Let $q \leftarrow 1$.
    (5.2) If $\tau \bmod 25 = 0$, then randomly generate $\alpha_{iq}(\tau+1)$, $\beta_{iq}(\tau+1)$, and $\gamma_{kq}(\tau+1) \sim U[0,1)$, where $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, 6$. Otherwise, generate $\alpha_{iq}(\tau+1)$, $\beta_{iq}(\tau+1)$, and $\gamma_{kq}(\tau+1)$ by below three equations, respectively $(i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, 6)$. Let $u_1$ and $u_2 \sim U[0,1)$.
        $\alpha_{iq}(\tau+1) = \begin{cases} \alpha_{iq}(\tau) + 0.025u_1 - 0.01u_2 & \text{if } \alpha_{iq}(\tau) < \alpha_{i\,\text{best}}, \\ \alpha_{iq}(\tau) - 0.025u_1 + 0.01u_2 & \text{if } \alpha_{iq}(\tau) > \alpha_{i\,\text{best}}, \\ \alpha_{iq}(\tau) + 0.01u_1 - 0.01u_2 & \text{if } \alpha_{iq}(\tau) = \alpha_{i\,\text{best}}. \end{cases}$
        $\beta_{iq}(\tau+1) = \begin{cases} \beta_{iq}(\tau) + 0.025u_1 - 0.01u_2 & \text{if } \beta_{iq}(\tau) < \beta_{i\,\text{best}}, \\ \beta_{iq}(\tau) - 0.025u_1 + 0.01u_2 & \text{if } \beta_{iq}(\tau) > \beta_{i\,\text{best}}, \\ \beta_{iq}(\tau) + 0.01u_1 - 0.01u_2 & \text{if } \beta_{iq}(\tau) = \beta_{i\,\text{best}}. \end{cases}$
        $\gamma_{kq}(\tau+1) = \begin{cases} \gamma_{kq}(\tau) + 0.025u_1 - 0.01u_2 & \text{if } \gamma_{kq}(\tau) < \gamma_{k\,\text{best}}, \\ \gamma_{kq}(\tau) - 0.025u_1 + 0.01u_2 & \text{if } \gamma_{kq}(\tau) > \gamma_{k\,\text{best}}, \\ \gamma_{kq}(\tau) + 0.01u_1 - 0.01u_2 & \text{if } \gamma_{kq}(\tau) = \gamma_{k\,\text{best}}. \end{cases}$
    (5.3) If $q < M$, then let $q \leftarrow q + 1$ and repeat from Step 5.2. Otherwise, go to Step 6.
(6) If the stopping criterion is not met, then let $\tau \leftarrow \tau + 1$ and repeat from Step 4. Otherwise, return $Z$ to the user.
```

Algorithm 3: The procedure of TOP.

(ix) Let $\zeta_{\text{best}} \equiv (\alpha_{1best}, \alpha_{2\text{best}}, \ldots, \alpha_{n\text{best}}, \beta_{1\text{best}}, \beta_{2\text{best}}, \ldots, \beta_{n\text{best}}, \gamma_{1\text{best}}, \gamma_{2\text{best}}, \ldots, \gamma_{6\text{best}})$ be the best $\zeta_q(\tau)$ ever found by the population. In addition, let $Score(\zeta_{\text{best}})$ stand for the performance score of ζ_{best}.

(x) Let Z represent the best schedule ever found by the TOP's population.

In Algorithm 3, TOP starts its procedure by receiving information of a BCJSP instance from the user. TOP assigns $\tau \leftarrow 1$ and $Score(\zeta_{\text{best}}) \leftarrow +\infty$; then, it generates $\zeta_q(\tau)$ randomly. After that, TOP starts a repeated loop by executing MID M times. In the q-th execution (where $q = 1, 2, \ldots, M$), MID is executed with $\zeta_q(\tau)$-given parameter values to return S_{best}; then, let $Score(\zeta_q(\tau)) \leftarrow Makespan(S_{\text{best}})$. If TOP finds any $\zeta_q(\tau)$ better than or equal to ζ_{best}, then let this $\zeta_q(\tau)$ and its corresponding S_{best} become a new ζ_{best} and a new Z, respectively. After that, TOP chooses to update each member of $\zeta_q(\tau+1)$ by either the two opposite-direction vectors or the reinitialization. If the stopping criterion is not met, TOP assigns $\tau \leftarrow \tau + 1$ and starts the repeated loop's next round.

As mentioned, there are two variants of MUL, i.e., MUL-B and MUL-T. The difference between MUL-B and MUL-T in their TOPs is given as follows. In MUL-T, $\zeta_q(\tau)$ of its TOP's population consists of $\alpha_{iq}(\tau)$, $\beta_{iq}(\tau)$, and $\gamma_{kq}(\tau)$. This means TOP in MUL-T is identical to Algorithm 3. In MUL-B, $\zeta_q(\tau)$ of its TOP's population consists of only $\alpha_{iq}(\tau)$ and $\beta_{iq}(\tau)$. It means that $\zeta_q(\tau)$ of TOP in MUL-B is equivalent to $(\alpha_{1q}(\tau), \alpha_{2q}(\tau), \ldots, \alpha_{nq}(\tau), \beta_{1q}(\tau), \beta_{2q}(\tau), \ldots, \beta_{nq}(\tau))$. Thus, the procedure of TOP in MUL-B is modified from Algorithm 3 by removing $\gamma_{kq}(\tau)$ from Step 3; removing y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn} and w_{pr} from Steps 4.2 and 4.3; and removing $\gamma_{kq}(\tau+1)$ and its generating equation from Step 5.2.

5. Results

The performance of the two proposed variants of MUL (i.e., MUL-B and MUL-T) was evaluated via an experiment on 53 BCJSP instances. These BCJSP instances were modified from the well-known JSP instances, i.e., FT06, FT10, and FT20 instances of [17]; LA01 to LA40 instances of [18]; and ORB01 to ORB10 instances of [19]. The modification of each instance was done by letting the operation-precedence-relation chains of all jobs be circular and bidirectional. In this paper, let BCFT06, BCFT10, and BCFT20 represent the BCJSP instances modified from FT06, FT10, and FT20, respectively. Let BCLA01 to BCLA40 represent the BCJSP instances modified from LA01 to LA40, respectively. Then, let BCORB01 to BCORB10 represent the BCJSP instances modified from ORB01 to ORB10, respectively. For each BCJSP instance, let its original JSP instance stand for the JSP instance later modified to become it. For example, FT06 is the original JSP instance of BCFT06.

Because of the BCJSP's novelty, no algorithms excepting MUL have intentionally been developed for BCJSP. Thus, to evaluate the performance of MUL-B and MUL-T, their performance was compared with each other and with the performance of the existing JSP-solving algorithms. Without modifications, the existing JSP-solving algorithms can solve BCJSP as well. However, they hardly find good solutions for BCJSP. This is because, for each BCJSP instance, the solution space of its original JSP instance is just a subset of its whole solution space. Consequently, for each BCJSP instance, the JSP-solving algorithms cannot return better solutions than the optimal solution of its original JSP instance. This is the reason that the original JSP instance's optimal solution is used as the best possible result from all JSP-solving algorithms on each BCJSP instance.

The settings of MUL-B and MUL-T for the experiment are summarized as follows:

(i) MUL-B and MUL-T were coded in C# and executed on an Intel® Core™ i5-3320M CPU processor @ 2.60 GHz with RAM of 4 GB (3.87 GB usable).

(ii) In each of MUL-B and MUL-T, let the population of MID consist of two members (i.e., $N = 2$ in Algorithm 2), and let MID be stopped after the 10-th iteration (i.e., $t = 10$ is the maximum iteration in Algorithm 2).

(iii) In only MUL-B, the constant value of 0.5 was assigned to the MID's input parameters y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}, and w_{pr}.

(iv) In each of MUL-B and MUL-T, let the population of TOP consist of two members (i.e., $M = 2$ in Algorithm 3), and let TOP be stopped if it could not find an improving solution (a better solution) within 100 consecutive iterations.

(v) Each of MUL-B and MUL-T was executed for three runs on each BCJSP instance with different random seed numbers.

Reasons for the above-mentioned parameter settings are given as follows. For limiting MUL-B's and MUL-T's computational time, the population sizes of their MIDs and TOPs were set to 2, the smallest possible size. With the same reason, their MIDs were set to stop after their 10-th iterations. However, for avoiding premature stops, their TOPs were set to stop after 100 consecutive iterations without finding a better solution. For performing as well as MUPLA [16], the parameters y_{pt}, y_{pn}, y_{pr}, w_{pt}, w_{pn}, and w_{pr} of MID in MUL-B were set to 0.5. This made the equations in Step 5.2 of Algorithm 2 become identical to those of MUPLA [16].

The experiment's results on the 53 BCJSP instances are presented in Table 1. Terms and abbreviations used in Table 1 and in its discussion are defined as follows:

(i) Let a solution and a solution value stand for a schedule and a schedule's makespan, respectively.

(ii) For each BCJSP instance, let its original JSP instance mean the JSP instance that was later modified to become it. For example, LA01 is the original JSP instance of BCLA01.

(iii) Let *Ins* column present the name of each BCJSP instance.

(iv) Let *JSP Opt* column present the optimal solution value of the original JSP instance of each BCJSP instance. The values in this column were given by published articles, e.g., [20, 21].

(v) For each of MUL-B and MUL-T, let *Best* and *Avg* stand for its best final solution value and its average final solution value, respectively, over three runs.

(vi) Let *UB* stand for the upper bound of the optimal solution value of each BCJSP instance. *UB* in this paper is a minimum among the values of *JSP Opt*, *Best* of MUL-B, and *Best* of MUL-T.

(vii) For each of MUL-B and MUL-T, let *%BD* stand for a percent deviation of its *Best* from *UB*, and let *%AD* stand for a percent deviation of its *Avg* from *UB*.

(viii) For each *JSP Opt*, let *%JD* stand for its percent deviation from *UB*.

(ix) For each of MUL-B and MUL-T, let *Avg Iters* column present the average number of iterations used until stopped on each instance. In parentheses, this column presents the average minimum number of iterations required to find the last improving solution. Note that the number of iterations used by each of MUL-B and MUL-T means the number of iterations used by its TOP.

(ix) For each of MUL-B and MUL-T, let *Avg time* column present an average computational time consumed until stopped on each instance in HH: MM: SS form. In parentheses, this column presents an average minimum computational time required to find the last improving solution.

Table 2 then summarizes the results shown in Table 1. In Table 2, the 53 BCJSP instances are classified into nine categories based on the numbers of machines (m) and the numbers of jobs (n). Terms and abbreviations used in Table 2 are defined as follows:

(i) *Category* column presents each category of the 53 BCJSP instances in the form $m \times n$, where m is the number of machines and n is the number of jobs.

(ii) *Members* column presents all instances that are members of each category. Then, *No. of Ins* column presents the number of all instances in each category.

(iii) Let *Avg %JD* stand for an average *%JD* over all instances in each category.

(iv) For each of MUL-B and MUL-T, let *Avg %BD* and *Avg %AD* stand for an average *%BD* and an average *%AD*, respectively, over all instances in each category.

(v) For each of MUL-B and MUL-T, let *Avg Iters* column present the average number of iterations used until stopped. In parentheses, this column presents the average minimum number of iterations required to

Table 1: Experiment's results.

Ins	UB	JSP Opt	%JD	MUL-B						MUL-T					
				Best	%BD	Avg	%AD	Avg iters	Avg time	Best	%BD	Avg	%AD	Avg iters	Avg time
BCFT06	47	55	17.02	48	2.13	49.3	4.89	178 (78)	0:00:10 (0:00:04)	47	0.00	48.7	3.62	217 (117)	0:00:12 (0:00:06)
BCFT10	739	930	25.85	780	5.55	786.0	6.36	150 (50)	0:07:21 (0:02:25)	739	0.00	756.0	2.30	183 (83)	0:08:56 (0:04:14)
BCFT20	1119	1165	4.11	1119	0.00	1119.0	0.00	101 (1)	0:08:18 (0:00:03)	1119	0.00	1119.0	0.00	101 (1)	0:08:15 (0:00:04)
BCLA01	666	666	0.00	666	0.00	666.0	0.00	112 (12)	0:00:29 (0:00:03)	666	0.00	666.0	0.00	112 (12)	0:00:30 (0:00:03)
BCLA02	635	655	3.15	635	0.00	635.0	0.00	114 (14)	0:00:30 (0:00:03)	635	0.00	635.0	0.00	103 (3)	0:00:27 (0:00: 00.4)
BCLA03	588	597	1.53	588	0.00	588.0	0.00	115 (15)	0:00:30 (0:00:04)	588	0.00	588.0	0.00	103 (3)	0:00:26 (0:00:01)
BCLA04	537	590	9.87	537	0.00	537.0	0.00	135 (35)	0:00:38 (0:00:10)	537	0.00	537.0	0.00	190 (90)	0:00:55 (0:00:25)
BCLA05	593	593	0.00	593	0.00	593.0	0.00	101 (1)	0:00:26 (0:00:00.3)	593	0.00	593.0	0.00	103 (3)	0:00:26 (0:00:01)
BCLA06	926	926	0.00	926	0.00	926.0	0.00	101 (1)	0:02:21 (0:00:01)	926	0.00	926.0	0.00	101 (1)	0:02:22 (0:00:01)
BCLA07	869	890	2.42	869	0.00	869.0	0.00	101 (1)	0:02:22 (0:00:01)	869	0.00	869.0	0.00	101 (1)	0:02:14 (0:00:01)
BCLA08	863	863	0.00	863	0.00	863.0	0.00	101 (1)	0:02:35 (0:00:01)	863	0.00	863.0	0.00	101 (1)	0:02:40 (0:00:01)
BCLA09	951	951	0.00	951	0.00	951.0	0.00	101 (1)	0:02:38 (0:00:02)	951	0.00	951.0	0.00	101 (1)	0:02:30 (0:00:01)
BCLA10	958	958	0.00	958	0.00	958.0	0.00	101 (1)	0:02:22 (0:00:01)	958	0.00	958.0	0.00	101 (1)	0:02:21 (0:00:01)
BCLA11	1222	1222	0.00	1222	0.00	1222.0	0.00	101 (1)	0:07:15 (0:00:05)	1222	0.00	1222.0	0.00	101 (1)	0:06:51 (0:00:03)
BCLA12	1039	1039	0.00	1039	0.00	1039.0	0.00	101 (1)	0:08:38 (0:00:06)	1039	0.00	1039.0	0.00	101 (1)	0:08:46 (0:00:03)
BCLA13	1150	1150	0.00	1150	0.00	1150.0	0.00	101 (1)	0:08:12 (0:00:05)	1150	0.00	1150.0	0.00	101 (1)	0:08:02 (0:00:05)
BCLA14	1292	1292	0.00	1292	0.00	1292.0	0.00	101 (1)	0:07:02 (0:00:03)	1292	0.00	1292.0	0.00	101 (1)	0:06:43 (0:00:04)
BCLA15	1207	1207	0.00	1207	0.00	1207.0	0.00	101 (1)	0:07:27 (0:00:05)	1207	0.00	1207.0	0.00	101 (1)	0:07:21 (0:00:03)
BCLA16	798	945	18.42	798	0.00	806.0	1.00	162 (62)	0:08:00 (0:03:11)	810	1.50	814.0	2.01	153 (53)	0:07:20 (0:02:37)
BCLA17	717	784	9.34	735	2.51	735.0	2.51	141 (41)	0:07:04 (0:02:00)	717	0.00	731.7	2.05	234 (134)	0:11:43 (0:06:47)
BCLA18	765	848	10.85	765	0.00	775.7	1.40	191 (91)	0:09:25 (0:04:20)	768	0.39	777.7	1.66	168 (68)	0:08:22 (0:03:23)
BCLA19	783	842	7.54	783	0.00	802.3	2.46	184 (84)	0:09:25 (0:04:21)	801	2.30	808.0	3.19	248 (148)	0:12:17 (0:07:18)
BCLA20	810	902	11.36	812	0.25	823.0	1.60	157 (57)	0:07:35 (0:02:48)	810	0.00	828.7	2.31	182 (82)	0:08:59 (0:04:01)
BCLA21	967	1046	8.17	981	1.45	985.7	1.93	212 (112)	1:05:44 (0:34:46)	967	0.00	979.7	1.31	181 (81)	0:55:07 (0:24:20)
BCLA22	900	927	3.00	911	1.22	925.3	2.81	172 (72)	0:54:31 (0:23:14)	900	0.00	905.0	0.56	150 (50)	0:47:03 (0:15:45)
BCLA23	1032	1032	0.00	1032	0.00	1032.0	0.00	141 (41)	0:43:54 (0:12:49)	1032	0.00	1032.0	0.00	141 (41)	0:42:37 (0:13:01)
BCLA24	932	935	0.32	938	0.64	948.3	1.75	180 (80)	0:55:37 (0:24:16)	932	0.00	950.3	1.96	174 (74)	0:54:07 (0:23:12)
BCLA25	907	977	7.72	918	1.21	935.0	3.09	226 (126)	1:09:46 (0:39:20)	907	0.00	927.0	2.21	253 (153)	1:17:00 (0:46:58)
BCLA26	1218	1218	0.00	1218	0.00	1218.0	0.00	146 (46)	2:35:48 (0:47:58)	1218	0.00	1218.0	0.00	119 (19)	2:09:40 (0:20:37)
BCLA27	1207	1235	2.32	1218	0.91	1222.0	1.24	162 (62)	2:58:45 (1:08:36)	1207	0.00	1222.7	1.30	162 (62)	3:02:29 (1:10:44)

Table 1: Continued.

Ins	UB	JSP Opt	%JD	MUL-B						MUL-T					
				Best	%BD	Avg	%AD	Avg iters	Avg time	Best	%BD	Avg	%AD	Avg iters	Avg time
BCLA28	1216	1216	0.00	1216	0.00	1217.7	0.14	223 (123)	4:03:14 (2:16:13)	1216	0.00	1216.0	0.00	177 (77)	3:13:08 (1:24:44)
BCLA29	1105	1152	4.25	1105	0.00	1105.0	0.00	152 (52)	2:46:47 (0:55:04)	1105	0.00	1115.0	0.90	188 (88)	3:26:20 (1:37:06)
BCLA30	1355	1355	0.00	1355	0.00	1355.0	0.00	101 (1)	1:43:32 (0:00:51)	1355	0.00	1355.0	0.00	101 (1)	1:39:45 (0:00:36)
BCLA31	1784	1784	0.00	1784	0.00	1784.0	0.00	101 (1)	7:50:40 (0:02:52)	1784	0.00	1784.0	0.00	101 (1)	8:55:01 (0:04:44)
BCLA32	1850	1850	0.00	1850	0.00	1850.0	0.00	102 (2)	10:21:25 (0:07:08)	1850	0.00	1850.0	0.00	102 (2)	10:30:37 (0:10:30)
BCLA33	1719	1719	0.00	1719	0.00	1719.0	0.00	101 (1)	9:13:34 (0:07:22)	1719	0.00	1719.0	0.00	101 (1)	9:35:45 (0:06:18)
BCLA34	1721	1721	0.00	1721	0.00	1721.0	0.00	101 (1)	10:45:44 (0:06:43)	1721	0.00	1721.0	0.00	101 (1)	11:06:51 (0:08:12)
BCLA35	1888	1888	0.00	1888	0.00	1888.0	0.00	101 (1)	7:31:40 (0:02:25)	1888	0.00	1888.0	0.00	101 (1)	8:07:04 (0:04:14)
BCLA36	1186	1268	6.91	1186	0.00	1210.7	2.08	196 (96)	3:59:35 (1:52:51)	1207	1.77	1218.0	2.70	177 (77)	3:40:50 (1:37:49)
BCLA37	1247	1397	12.03	1247	0.00	1271.3	1.95	220 (120)	4:36:09 (2:24:27)	1292	3.61	1295.3	3.87	171 (71)	3:38:52 (1:33:27)
BCLA38	1114	1196	7.36	1165	4.58	1173.0	5.30	198 (98)	4:11:10 (2:04:43)	1114	0.00	1145.0	2.78	195 (95)	4:15:04 (2:07:00)
BCLA39	1186	1233	3.96	1186	0.00	1196.0	0.84	195 (95)	4:06:29 (1:57:48)	1191	0.42	1192.7	0.56	214 (114)	4:26:56 (2:19:38)
BCLA40	1150	1222	6.26	1156	0.52	1182.3	2.81	166 (66)	3:24:58 (1:24:33)	1150	0.00	1179.3	2.55	145 (45)	2:58:02 (0:56:47)
BCORB01	789	1059	34.22	812	2.92	818.3	3.71	248 (148)	0:12:31 (0:07:30)	789	0.00	810.3	2.70	150 (50)	0:07:45 (0:02:35)
BCORB02	763	888	16.38	763	0.00	783.0	2.62	164 (64)	0:08:24 (0:03:19)	797	4.46	803.7	5.33	173 (73)	0:09:01 (0:03:47)
BCORB03	741	1005	35.63	741	0.00	764.3	3.14	192 (92)	0:09:29 (0:04:28)	773	4.32	780.7	5.36	333 (233)	0:16:43 (0:11:39)
BCORB04	831	1005	20.94	839	0.96	849.7	2.25	163 (63)	0:08:01 (0:03:07)	831	0.00	839.3	1.00	217 (117)	0:10:54 (0:05:48)
BCORB05	705	887	25.82	705	0.00	717.3	1.74	213 (113)	0:10:26 (0:05:42)	727	3.12	729.0	3.40	180 (80)	0:08:51 (0:03:56)
BCORB06	817	1010	23.62	822	0.61	835.3	2.24	136 (36)	0:06:40 (0:01:43)	817	0.00	832.3	1.87	187 (87)	0:09:27 (0:04:21)
BCORB07	353	397	12.46	356	0.85	358.7	1.61	201 (101)	0:09:45 (0:04:58)	353	0.00	358.0	1.42	155 (55)	0:07:35 (0:02:42)
BCORB08	671	899	33.98	675	0.60	684.3	1.98	208 (108)	0:10:14 (0:05:19)	671	0.00	683.3	1.83	186 (86)	0:09:19 (0:04:10)
BCORB09	772	934	20.98	773	0.13	792.0	2.59	154 (54)	0:07:50 (0:02:42)	772	0.00	789.3	2.24	179 (79)	0:09:15 (0:04:01)
BCORB10	780	944	21.03	812	4.10	825.7	5.86	193 (93)	0:09:30 (0:04:34)	780	0.00	788.7	1.12	266 (166)	0:13:24 (0:08:19)

find the last improving solution. Note that the number of iterations used by each of MUL-B and MUL-T means the number of iterations used by its TOP.

(vi) For each of MUL-B and MUL-T, let *Avg time* column present an average computational time consumed until stopped. In parentheses, this column presents an average minimum computational time required to find the last improving solution.

(vii) In the competition between MUL-B and MUL-T, let *W*, *D*, and *L* in *W-D-L* present the numbers of instances won, drawn, and lost, respectively. On each instance, MUL-B is judged to win MUL-T if the MUL-B's *%BD* is better than the MUL-T's *%BD*, and vice versa. In addition, they are judged to draw if their *%BDs* are equal.

Figure 5 presents the rates of solution improvements over iterations of MUL-B and MUL-T. In making the figure, *Avg-%AD-over-iteration* plots and *Avg-%BD-over-iteration* plots were plotted and compared with *Avg %JD*. Let *Avg %JD* be calculated from an average *%JD* over 53 instances. For each of MUL-B and MUL-T, let *Avg-%AD-over-iteration* plot and *Avg-%BD-over-iteration* plot present the average *%AD* over 53 instances

Table 2: A summary of experiment's results.

Category	Members	No. of Ins	Avg %JD	MUL-B Avg %BD	MUL-B *W-D-L*	MUL-B Avg %AD	MUL-B Avg iters	MUL-B Avg time	MUL-T Avg %BD	MUL-T *W-D-L*	MUL-T Avg %AD	MUL-T Avg iters	MUL-T Avg time
5×10	BCLA01–05	5	2.91	0.00	0-5-0	0.00	115 (15)	0:00:31 (0:00:04)	0.00	0-5-0	0.00	122 (22)	0:00:33 (0:00:06)
5×15	BCLA06–10	5	0.48	0.00	0-5-0	0.00	101 (1)	0:02:28 (0:00:01)	0.00	0-5-0	0.00	101 (1)	0:02:25 (0:00:01)
5×20	BCFT20, BCLA11–15	6	0.69	0.00	0-6-0	0.00	101 (1)	0:07:49 (0:00:05)	0.00	0-6-0	0.00	101 (1)	0:07:40 (0:00:04)
6×6	BCFT06	1	17.02	2.13	0-0-1	4.89	178 (78)	0:00:10 (0:00:04)	0.00	1-0-0	3.62	217 (117)	0:00:12 (0:00:06)
10×10	BCFT10, BCLA16–20, BCORB01–10	16	20.53	1.16	6-0-10	2.69	178 (78)	0:08:51 (0:03:54)	1.01	10-0-6	2.49	200 (100)	0:09:59 (0:04:45)
10×15	BCLA21–25	5	3.84	0.90	0-1-4	1.92	186 (86)	0:57:54 (0:26:53)	0.00	4-1-0	1.21	180 (80)	0:55:11 (0:24:39)
10×20	BCLA26–30	5	1.31	0.18	0-4-1	0.28	157 (57)	2:49:37 (1:01:44)	0.00	1-4-0	0.44	149 (49)	2:42:16 (0:54:45)
10×30	BCLA31–35	5	0.00	0.00	0-5-0	0.00	101 (1)	9:08:37 (0:05:18)	0.00	0-5-0	0.00	101 (1)	9:39:04 (0:06:48)
15×15	BCLA36–40	5	7.30	1.02	3-0-2	2.60	195 (95)	4:03:40 (1:56:52)	1.16	2-0-3	2.49	180 (80)	3:47:57 (1:42:56)
All	All 53 instances	53	8.09	0.59	9-26-18	1.36	149 (49)	1:40:03 (0:21:05)	0.41	18-26-9	1.21	154 (54)	1:40:49 (0:19:22)

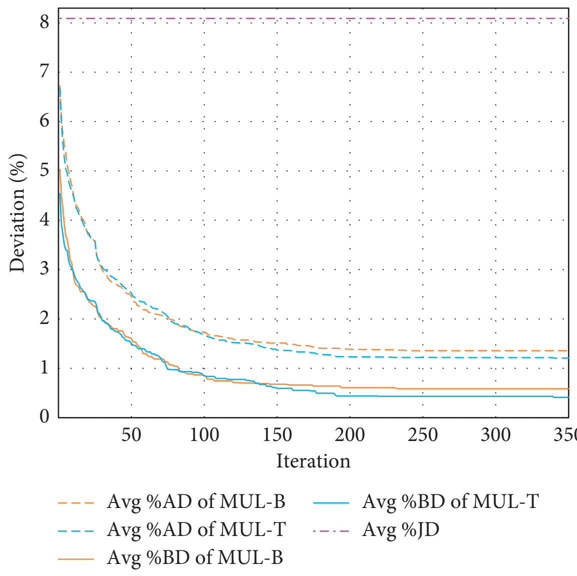


Figure 5: Avg %ADs and Avg %BDs over iterations of MUL-B and MUL-T compared with Avg %JD.

and the average *%BD* over 53 instances, respectively, from the first to the 350-th iterations. One obstacle of plotting each plot was that each algorithm was usually stopped before the 350-th iteration, as shown in Table 1. To deal with the obstacle, since the algorithms were stopped, the last values of *%AD* and *%BD* had been used in the plots until the 350-th iterations.

6. Discussion

6.1. Result Comparisons of MUL-B and MUL-T with Other Algorithms. As mentioned previously, no existing algorithms excepting MUL-B and MUL-T have been developed to solve BCJSP. This means no other existing BCJSP-solving algorithms could be used for comparison against MUL-B and MUL-T. It is the cause that, in this research, the MUL-B's and MUL-T's results were compared against *JSP Opt*. For each BCJSP instance, *JSP Opt* in Table 1 denotes the optimal solution value of its original JSP instance. (Note that, for example, FT06 is the original JSP instance of BCFT06.) Rather than comparison with many JSP-solving algorithms, let *JSP Opt* be a representative of the best possible result from all existing JSP-solving algorithms on BCJSP.

As shown in Table 1, MUL-B returns better solution values than *JSP Opt* on 32 instances, equivalent values on 20 instances, and worse values on one instance. This means MUL-B performs better than the best performance of the existing JSP-solving algorithms on 32 instances, has equivalent performance on 20 instances, and has worse performance on one instance. MUL-T returns better solution values than *JSP Opt* on 33 instances, equivalent values on 20 instances, and worse values on none of the instances. This means MUL-T performs better than the best performance of the existing JSP-solving algorithms on 33 instances, has equivalent performance on 20 instances, and has worse performance on none of the instances.

Note that, in Table 1, *%JD*, *%BD*, and *%AD* represent the percent deviations from *UB* of *JSP Opt*, *Best*, and *Avg*, respectively. One-sided paired *t* tests conclude that the population mean *%BDs* of MUL-B and MUL-T are both significantly better than the population mean *%JD* (with *p* values of 4×10^{-7} and 3×10^{-7}, respectively). Moreover, the population mean *%ADs* of MUL-B and MUL-T are also significantly better than the population mean *%JD* (with *p* values of 10×10^{-7} and 5×10^{-7}, respectively). As an interpretation, the two variants of MUL outperform the existing JSP-solving algorithms on BCJSP.

A cause for the above-mentioned results is that MUL-B and MUL-T can both explore the whole solution space of the BCJSP instance, while the existing JSP-solving algorithms can explore only the original JSP instance's solution space. For each BCJSP instance, the solution space of its original JSP instance is just a small subset of its whole solution space. Then, a BCJSP instance usually has multiple better solutions than the optimal solution of its original JSP instance. Consequently, MUL-B and MUL-T both possibly find better solutions than the original JSP instance's optimal solution, whereas the existing JSP-solving algorithms cannot.

6.2. Result Comparisons between MUL-B and MUL-T. As shown in Tables 1 and 2, MUL-T performs slightly better than MUL-B in the average solution quality. Over 53 instances, *Avg %BD* of MUL-T (i.e., 0.41%) is slightly better than *Avg %BD* of MUL-B (i.e., 0.59%). Likewise, *Avg %AD* of MUL-T (i.e., 1.21%) is slightly better than *Avg %AD* of MUL-B (i.e., 1.36%). However, based on one-sided paired t tests, no sufficient evidence suggests that the population mean *%BD* of MUL-T is better than that of MUL-B (with p value of 0.24). In addition, no enough evidence suggests that the population mean *%AD* of MUL-T is better than that of MUL-B (with p value of 0.19).

By counting the instances won in Table 1, the outperformance of MUL-T over MUL-B can be detected more clearly. On each instance, let MUL-T be judged to win MUL-B if the MUL-T's *%BD* is lower than the MUL-B's *%BD*, and vice versa. Over 53 instances, MUL-T wins MUL-B on 18 instances, draws on 26 instances, and loses on nine instances. As noticed, the number of instances won by MUL-T (i.e., 18) is twice of the number of instances won by MUL-B (i.e., 9). Although a proportion of drawn instances (i.e., 26 out of total 53) is very high, most of the drawn instances are small or easy to solve (e.g., the instances in the 5×10, 5×15, 5×20, and 10×30 categories in Table 2). To determine whether the total number of all instances won by MUL-T is greater than that by MUL-B, a one-sided binomial test was conducted at a significance level of 0.1. Let a sample size be the number of instances not drawn from the total 53 instances (i.e., 27). The test's result concludes that, from all not-drawn instances, the number of all instances won by MUL-T is significantly greater than that by MUL-B (with p value of 0.061).

As mentioned previously, the main difference between MUL-B and MUL-T is in their TOPs. The MUL-B's and MUL-T's TOPs both control the start operations and the operation-precedence-relation directions in BCJSP. However, only TOP of MUL-T additionally controls the MID's input parameters as the extra function. The outperformance of MUL-T over MUL-B abovementioned indicates that this extra function can enhance the performance of MUL. This means the MID's performance can be enhanced by using TOP to control its input parameters. Once the MID's performance has been enhanced, MID can provide better input-parameter values for BOT. Consequently, BOT can find better solutions.

6.3. Number of Iterations and Computational Time Consumed. For MUL-B and MUL-T, the numbers of used iterations directly affect computational time spent. The more the number of iterations used, the longer the computational time consumed. Table 1 presents the average number of iterations and the average computational time of the three runs of each algorithm. For each instance, *Avg Iters* and *Avg time* columns present the average number of iterations and the average computational time, respectively, used until the stopping criterion is met. (Note that the stopping criterion was to stop when TOP could not find an improving solution within 100 consecutive iterations.) In parentheses, *Avg Iters* and *Avg time* columns show the average minimum number of iterations and the average minimum computational time, respectively, required to find the last improving solution. A summary of the data from Table 1 just-mentioned can be found in Table 2.

As mentioned above, the stopping criterion for each algorithm was to stop when its TOP could not find an improving solution within 100 consecutive iterations. The purpose of using this stopping criterion is to avoid a premature stop for each algorithm. However, it probably results in an unnecessary high consumption of the number of iterations and of computational time. For example, on BCLA34, MUL-B and MUL-T both find their best-found solutions at their first iterations; however, they have to proceed until the 101-st iterations. For computational time on BCLA34, the both algorithms find their best-found solutions within 10 minutes; however, they have to stop after around 11 hours. Such cases also happen on many other instances, not only BCLA34.

Because the given stopping criterion causes a very long computational time, the user should not wait until MUL-B and MUL-T meet their stopping criteria. The user can break off the algorithms' executions at any time to receive their current best-found solutions. To choose a breaking-off time, the user may use information from the values in parentheses of *Avg Iters* and *Avg time* columns in Table 1. *Avg Iters* column indicates that, on more than 50% of all instances, the algorithms find their last improving solutions within 55 iterations; on all instances excepting BCORB03, they find their last improving solutions within 170 iterations. *Avg time* column indicates that, on more than 50% of all instances, the algorithms find their last improving solutions within 4.5 minutes; on all instances, they find their last improving solutions within 145 minutes.

As suggested from *Avg Iters* column, MUL-B and MUL-T should be broken off during the 55-th to the 170-th iterations. In addition, as suggested from *Avg time* column, they should be broken off during the 4.5-th to the 145-th minutes. Thus, as a minimum requirement, the user should break off each algorithm at the 55-th iteration or the 4.5-th minute of computational time, whichever comes first. If an optimal or near-optimal solution is required, the user is suggested to break off each algorithm at the 170-th iteration or the 145-th minute of computational time, whichever comes first. Moreover, the user may decide to break off each algorithm at any iteration index during the 55-th to the 170-th iterations or at any time during the 4.5-th to the 145-th minutes. The decision is made based on his trade-off between the computational time and the possibility of finding an improving solution. Note that the computational time in the above suggestions may be changed if the user's computer has a different specification from the computer used in this research.

6.4. Solution Improvement Rate over Iteration. Since started, *Avg-%AD-over-iteration* plots and *Avg-%BD-over-iteration* plots of MUL-B and MUL-T in Figure 5 have already been below *Avg %JD* (i.e., 8.09%). At their first iterations, *Avg %AD* of MUL-B, *Avg %AD* of MUL-T, *Avg %BD* of MUL-B, and *Avg %BD* of MUL-T are 6.72%, 6.67%, 5.03%, and 4.53%, respectively. Since then, the values of the four plots have still been reduced continuously. The four plots in the mentioned order finally become 1.36%, 1.21%, 0.59%, and 0.41%, respectively, at their final values. Obviously, all their final values are much lower than *Avg %JD*. This means MUL-B and MUL-T perform better than the JSP-solving algorithms on BCJSP because *Avg %JD* is the best possible result of the JSP-solving algorithms.

All given plots in Figure 5 behave similarly to each other in their patterns. These plots can be divided into three periods based on their similar patterns. The first period of each plot, where the value is reduced very quickly, is approximately started from the first iteration to the 55-th iteration. The second period, where the value is reduced gradually, is approximately started from the 55-th iteration to the 170-th iteration. The third period, where the value is reduced hardly, is approximately started from the 170-th iteration onwards. After the 170-the iteration, the plot has become more and more stable. The patterns of the plots emphasize that the user should break off each algorithm during the 55-th to the 170-th iterations, where the number of more iterations is a trade-off for the higher possibility of finding an improving solution.

7. Conclusions

MUL is the multilevel metaheuristic developed to solve the job-shop scheduling problem with bidirectional circular precedence constraints (BCJSP). MUL consists of TOP, MID, and BOT algorithms in its top, middle, and bottom levels, respectively. TOP is the population-based metaheuristic developed to control the start operation and the operation-precedence-relation direction of each job in BCJSP. If requested, TOP can also control the MID's input parameters. MID is the population-based metaheuristic developed to control the BOT's input parameters. BOT generates a subproblem of the BCJSP instance, in the form of JSP, by using the start operations and the operation-precedence-relation directions given by TOP. BOT then acts as a local search algorithm to solve the generated subproblem. The population in MID is evolved by the feedback from BOT, while the population in TOP is evolved by the feedback from MID. In this paper, there are two proposed variants of MUL, i.e., MUL-B and MUL-T. These two variants both use their TOPs to control the start operation and the operation-precedence-relation direction of each job. However, only MUL-T additionally uses its TOP to control the MID's input parameters. Because MUL-B and MUL-T were intentionally developed for BCJSP, they perform much better than the existing JSP-solving algorithms on BCJSP. When comparing the two MUL variants, MUL-T outperforms MUL-B slightly in the average solution quality and significantly in the number of instances won.

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 regarding the publication of this article.

Acknowledgments

The author acknowledges partial financial support for publication from the Thai-Nichi Institute of Technology, Thailand.

References

[1] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, 1996.

[2] T. Yamada and R. Nakano, "Job-shop scheduling," in *Genetic Algorithms in Engineering Systems*, A. M. S. Zalzala and P. J. Fleming, Eds., pp. 134–160, The Institution of Electrical Engineers, London, UK, 1997.

[3] C. Blum, "Beam-ACO—hybridizing ant colony optimization with beam search: an application to open shop scheduling," *Computers & Operations Research*, vol. 32, no. 6, pp. 1565–1591, 2005.

[4] 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.

[5] P. Brucker, Y. N. Sotskov, and F. Werner, "Complexity of shop-scheduling problems with fixed number of jobs: a survey," *Mathematical Methods of Operations Research*, vol. 65, pp. 461–481, 2007.

[6] M. M. Ahmadian, M. Khatami, A. Salehipour, and T. C. E. Cheng, "Four decades of research on the open-shop scheduling problem to minimize the makespan," *European Journal of Operational Research*, vol. 295, no. 2, pp. 399–426, 2021.

[7] 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, no. 4, pp. 1809–1830, 2019.

[8] 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.

[9] C. Lu, X. Li, L. Gao, W. Liao, and J. Yi, "An effective multi-objective discrete virus optimization algorithm for flexible job-shop scheduling problem with controllable processing times," *Computers & Industrial Engineering*, vol. 104, pp. 156–174, 2017.

[10] L. Yin, X. Li, L. Gao, C. Lu, and Z. Zhang, "A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem," *Sustainable Computing: Informatics and Systems*, vol. 13, pp. 15–30, 2017.

[11] Z. Adak, M. Ö. A. Akan, and S. Bulkan, "Multiprocessor open shop problem: literature review and future directions," *Journal of Combinatorial Optimization*, vol. 40, pp. 547–569, 2020.

[12] J. Kuster, D. Jannach, and G. Friedrich, "Extending the RCPSP for modeling and solving disruption management problems," *Applied Intelligence*, vol. 31, no. 3, pp. 234–253, 2009.

[13] I. A. Chaudhry and A. A. Khan, "A research survey: review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[14] X. Li and L. Gao, "Review for flexible job shop scheduling," in *Engineering Applications of Computational Methods*, vol. 2, pp. 17–45, Springer, Berlin, Germany, 2020.

[15] Á. 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.

[16] 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.

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

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

[19] 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.

[20] J. F. Gonçalves and M. G. C. Resende, "An extended akers graphical method with a biased random-key genetic algorithm for job-shop scheduling," *International Transactions in Operational Research*, vol. 21, no. 2, pp. 215–246, 2014.

[21] B. Peng, Z. Lü, and T. C. E. Cheng, "A tabu search/path relinking algorithm to solve the job shop scheduling problem," *Computers & Operations Research*, vol. 53, pp. 154–164, 2015.

[22] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[23] H. R. Lourenço, O. C. Martin, and T. Stützle, "Iterated local search," in *International Series in Operations Research and Management Science*, vol. 57, pp. 321–354, Springer, Boston, MA, USA, 2003.

[24] S. Kande, C. Prins, L. Belgacem, and B. Redon, "Multi-start iterated local search for two-echelon distribution network for perishable products," in *Proceedings of the International Conference on Operations Research and Enterprise Systems*, pp. 294–303, Lisbon, Portugal, January 2015.

[25] 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.

[26] C.-W. Chiou and M.-C. Wu, "A GA-tabu algorithm for scheduling in-line steppers in low-yield scenarios," *Expert Systems with Applications*, vol. 36, no. 9, pp. 11925–11933, 2009.

[27] T. Davidović, P. Hansen, and N. Mladenović, "Scheduling by VNS: experimental analysis," in *Proceedings of the Yugoslav Symposium on Operations Research*, pp. 319–322, Belgrade, Serbia, October 2001.

[28] M.-E. Marmion, F. Mascia, M. López-Ibáñez, and T. Stützle, "Automatic design of hybrid stochastic local search algorithms," *Hybrid Metaheuristics*, vol. 7919, pp. 144–158, 2013.

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

[30] M. Dorigo and T. Stützle, *Ant Colony Optimization*, MIT Press, Cambridge, MA, USA, 2004.

[31] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*, Morgan Kaufmann, Burlington, MA, USA, 2001.

[32] R. Storn and K. Price, "Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[33] X. S. Yang and S. Deb, "Engineering optimisation by cuckoo search," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 1, no. 4, pp. 330–343, 2010.

[34] E. K. Burke, M. Gendreau, M. Hyde et al., "Hyper-heuristic: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[35] X. Fu, F. T. S. Chan, B. Niu, N. S. H. Chung, and T. Qu, "A three-level particle swarm optimization with variable neighbourhood search algorithm for the production scheduling problem with mould maintenance," *Swarm and Evolutionary Computation*, vol. 50, Article ID 100572, 2019.

[36] A. Kattan and S. Fatima, "PSO as a meta-search for hyper-GA system to evolve optimal agendas for sequential multi-issue negotiation," in *Proceedings of the 2012 IEEE Congress on Evolutionary Computation*, Brisbane, Australia, June 2012.

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

[38] J. Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 122–128, 1986.

[39] 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.

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

[41] P. Pongchairerks, "Particle swarm optimization algorithms and their applications to scheduling problems," D. E. Dissertation, Asian Institute of Technology, Khlong Nueng, Thailand, 2008.

[42] P. Pongchairerks, "Particle swarm optimization algorithm applied to scheduling problems," *ScienceAsia*, vol. 35, no. 1, pp. 89–94, 2009.

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

[44] Q. Luo, Y. Zhou, J. Xie, M. Ma, and L. Li, "Discrete bat algorithm for optimal problem of permutation flow shop

scheduling," *Science World Journal*, vol. 2014, Article ID 630280, 15 pages, 2014.

[45] M. N. Janardhanan, Z. Li, P. Nielsen, and Q. Tang, "Artificial bee colony algorithms for two-sided assembly line worker assignment and balancing problem," in *Advances in Intelligent Systems and Computing*, vol. 620, pp. 11–18, Springer, Cham, Germany, 2018.