Hindawi Publishing Corporation
Modelling and Simulation in Engineering
Volume 2016, Article ID 5071654, 15 pages
http://dx.doi.org/10.1155/2016/5071654

Research Article

Forward VNS, Reverse VNS, and Multi-VNS Algorithms for Job-Shop Scheduling Problem

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 19 April 2016; Revised 3 July 2016; Accepted 11 August 2016

Academic Editor: Farouk Yalaoui

This paper proposes a number of forward VNS and reverse VNS algorithms for job-shop scheduling problem. The forward VNS algorithms are the variable neighborhood search algorithms applied to the original problem (i.e., the problem instance with the original precedence constraints). The reverse VNS algorithms are the variable neighborhood search algorithms applied to the reversed problem (i.e., the problem instance with the reversed precedence constraints). This paper also proposes a multi-VNS algorithm which assigns an identical initial solution-representing permutation to the selected VNS algorithms, runs these VNS algorithms, and then uses the best solution among the final solutions of all selected VNS algorithms as its final result. The aim of the multi-VNS algorithm is to utilize each single initial solution-representing permutation most efficiently and thus receive its best result in return.

1. Introduction

Job-shop scheduling problem (JSP) is a hard-to-solve scheduling problem which has commonly been found in many industries. JSP is similar to other scheduling problems in the terms that it consists of a number of jobs and a number of machines, and it requires assigning the given jobs into the given machines over time. However, JSP has some more specific constraints which make it unique and thus different from the other scheduling problems. These specific constraints are given as follows. Each job in JSP consists of a number of operations which must be processed in the specific order as the precedence constraints. Each operation must be processed on a preassigned machine by a specific processing time without preemption. In addition, each machine cannot process more than one operation simultaneously. The objective of JSP is to find a feasible schedule which completes all given jobs by the shortest makespan, that is, the time length from the starting time to the completion time of the schedule.

In order to solve JSP, this paper is interested in applying variable neighborhood search (VNS) algorithm because this algorithm is recognized as a simple, systematic, and successful metaheuristic for combinatorial problems. This paper receives insight and motivation from the previously published literature to develop the forward VNS, reverse VNS, and multi-VNS algorithms. The forward VNS algorithms are the VNS algorithms applied to the original problem (i.e., the being-considered JSP instance with the original precedence constraints). The reverse VNS algorithms are the VNS algorithms applied to the reversed problem (i.e., the being-considered JSP instance with the reversed precedence constraints); each reverse VNS algorithm has an additional step to transform its reversed problem's solution to be usable for the original problem. The proposed multi-VNS algorithm is an algorithm which assigns the same initial solution-representing permutation into a number of the specified VNS algorithms, runs these VNS algorithms, and finally uses the best solution among the final solutions of these VNS algorithms as its final result. In other words, the multi-VNS algorithm aims at utilizing each single initial solution-representing permutation most efficiently by systematically using different VNS neighborhood structures and scheduling directions on the same initial solution-representing permutation.

The remaining parts of this paper are organized as follows. Section 2 reviews the articles related to the research in this paper and then summarizes the research contributions. Section 3 proposes a generic VNS algorithm, so that this

section later proposes the forward VNS algorithms and the reverse VNS algorithms based on the given generic form. This paper proposes a multi-VNS algorithm in Section 4 and then evaluates the multi-VNS algorithm's performance in Section 5. Section 6 finally provides the conclusions of this research.

2. Literature Review and Research Contribution

Job-shop scheduling problem (JSP) starts 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 is composed of m given operations $O_{i1}, O_{i2}, \ldots, O_{im}$ which must be processed in the given order as a chain of precedence constraints. This means, for each job J_i, the operation O_{i1} must be finished before the operation O_{i2} can be started, the operation O_{i2} must be finished before the operation O_{i3} can be started, and so on. The operation O_{ij} can be processed only on a preassigned machine with a specific preassigned processing time. The preemption for each operation is not allowed; that is, after a particular machine starts processing any operation, it cannot be stopped or paused for any reasons until finishing the operation. In addition, each machine can process only one operation at a time. JSP aims at finding a schedule (i.e., an allocation of all given operations to time intervals on the given machines) which satisfies all above-given constraints so as to minimize the schedule's makespan. As mentioned, the makespan defines the time length from the starting time of the schedule (i.e., the starting time of the first started operation in the schedule) to the completion time of the schedule (i.e., the completion time of the last finished operation in the schedule). JSP is a well-known scheduling problem, so it has been mentioned frequently in the textbooks, for example, [1, 2]. The mathematical models for describing JSP have been commonly found in literature, for example, [3].

JSP is important to industry and attractive to academia, so many algorithms have been developed for solving the problem. These algorithms include tabu search algorithms [4, 5], a simulated annealing algorithm (SA) [6], a hybrid algorithm between particle swarm optimization (PSO) and VNS [7], genetic algorithms (GAs) [8–13], PSO algorithms [14–16], VNS algorithms [17–19], a hybrid algorithm between PSO and GA [20], a bee colony algorithm [21], an ant colony optimization algorithm [22], a memetic algorithm [23], and a hybrid algorithm between GA and SA [24]. Based on the literature review, the VNS algorithms are recognized as well-performing algorithms for JSP, so this paper will research more on the VNS algorithms.

VNS was first introduced in [25, 26] as a metaheuristic approach for combinatorial optimization problems. As its name implies, VNS changes its neighborhood structure from one to another systematically in the purpose of finding local optimal solutions as well as escaping from them, so VNS is highly potential to find a global optimal solution. The development of VNS is based on the three following observations [25, 27]:

(1) A local optimal solution in a neighborhood structure may not be the same as a local optimal solution in another neighborhood structure.

(2) A global optimal solution is a local optimal solution with respect to all possible neighborhood structures.

(3) In many problem instances, local optimal solutions in different neighborhood structures are relatively close to each other.

VNS generally consists of three main steps: the shaking step, the local search step, and the step of updating its best found solution. The VNS algorithm's local search step aims at finding a local optimal solution with respect to variable neighborhood structures. The shaking step aims at escaping from a local optimal solution as well as generating a new initial solution for the local search step. In published literature, the review articles about VNS are found in [25–28], the applications of VNS are found in [17–19, 28, 29], and the parallelization strategies for VNS are given in [27, 30].

The articles closely related to the research in this paper are articles [7, 17, 19, 29]. As mentioned above, a well-performing hybrid algorithm between PSO and VNS for JSP was given in [7]. Later, article [17] disassembled the VNS from the hybrid algorithm and reported that the VNS algorithm alone performs as equally well as the hybrid algorithm in terms of solution quality. After that, article [29] introduced the variants of the VNS algorithm of [17] for asymmetric traveling salesman problem, while article [19] introduced the variants of the VNS algorithm of [17] for JSP. Article [19] is most closely related to the research in this paper, because this paper aims at enhancing the performances of the VNS algorithms given in [19].

As just mentioned, the objective of the research in this paper is to enhance the performances of the VNS algorithms of [19], so the contributions of the research in this paper are given in overview as follows. A preliminary study of this research finds that, in several hard-to-solve JSP instances, the maximum iterations of the VNS algorithms of [19] should be increased in order to enhance their potentials of finding the optimal solutions; thus, this paper will find out the more proper maximum iterations for the VNS algorithms. This paper will also introduce more variants of the VNS algorithms of [19] which use different neighborhood structures from [19]. Moreover, this paper will introduce the use of the reversed problem (i.e., the being-considered JSP instance with the reversed precedence constraints) for the VNS algorithms because each hard-to-solve JSP instance may be solved easier in its corresponding reversed problem. Note that the schedule's construction using the reversed precedence constraints is called the reverse or backward scheduling, and it has often been applied for scheduling problems in many articles such as [14, 22, 31, 32]. For efficiently utilizing each initial solution-representing permutation, this paper will then propose the multi-VNS algorithm which assigns an identical initial solution-representing permutation into the selected VNS algorithms, runs these VNS algorithms, and uses the best solution found by these VNS algorithms as its final solution.

3. Proposed VNS Algorithms

This section will propose the generic VNS algorithm for JSP, so that the forward VNS algorithms and the reverse VNS algorithms will be developed based on the given generic VNS algorithm. As mentioned earlier, the forward VNS algorithms define the VNS algorithms applied to the original problem, that is, the being-considered JSP instance using the original (forward) precedence constraints. The reverse VNS algorithms define the VNS algorithms applied to the reversed problem, that is, the being-considered JSP instance using the reverse (backward) precedence constraints. In addition, each reverse VNS algorithm has one more additional step to modify its reversed problem's solution to be usable for the original problem. The terms *forward VNS* and *reverse VNS* will hereafter be abbreviated by *FVNS* and *RVNS*, respectively. Section 3.1 provides the generic VNS algorithm which is the generic form for all FVNS and RVNS algorithms proposed in this paper. Based on the generic form given, Section 3.2 presents the FVNS and RVNS algorithms using different operators to generate their solution-representing permutations. The performances of the proposed FVNS and RVNS algorithms will also be tested in Section 3.2.

3.1. Generic VNS Algorithm for JSP. This section introduces the generic VNS algorithm as the generic form of the FVNS and RVNS algorithms proposed later in Section 3.2. Note that the solutions (i.e., the JSP schedules) generated by all proposed VNS algorithms are represented by the operation-based permutations [3, 13]. Each operation-based permutation is an arrangement of the mn integers consisting of $1_1, 1_2, \ldots, 1_m, 2_1, 2_2, \ldots, 2_m, \ldots, n_1, n_2, \ldots, n_m$, where the subscripts are used to distinguish the integer of the same value. In other words, it is a sequence of the mn integers consisting of the numbers from 1 to n, where each number (from 1 to n) occurs repeatedly m times. Based on the JSP definition given in Section 2, n is the number of all given jobs, while m is the number of all given machines. Moreover, m is also equal to the number of all operations of each job, so mn is thus the number of all operations in the schedule. As an example, $P = \{2, 3, 2, 1, 1, 3\}$ is an operation-based permutation possibly generated by a particular VNS algorithm for a 3-job/2-machine JSP instance. The procedure to decode an operation-based permutation into a JSP schedule, as found in [3, 13, 17, 19], is given here in Algorithm 1. This decoding procedure transforms the operation-based permutation into an order of priorities of all given operations and then uses this order of priorities to construct a semiactive schedule. Note that a semiactive schedule is a feasible schedule such that no operations can be started earlier without altering the given order of priorities of operations.

Algorithm 1. It is a procedure to decode an operation-based permutation into a semiactive schedule.

Step 1. Let P represent the operation-based permutation which is required to be transformed into the semiactive schedule S. Let the number i ($i = 1, 2, \ldots, n$) in the jth occurrence ($j = 1, 2, \ldots, m$) from leftmost to the right of the permutation P refer to the jth operation of the job i or O_{ij}. After that, let the order of these mn operations in the permutation P from leftmost to the right define the order of priorities of the mn operations from highest to lowest. For example, $P = \{2, 3, 2, 1, 1, 3\}$ means that the order of priorities of the operations in descending order is O_{21}, O_{31}, O_{22}, O_{11}, O_{12}, and O_{32}.

Step 2. At the beginning, let the schedule S be empty, so the earliest available times of all m machines equal 0. Let $t = 0$.

Step 3. Based on the order of priorities of all operations given in Step 1, let O represent the current highest-priority operation among all as-yet-unassigned operations. Then, let τ represent the preassigned processing time of the operation O, and let M represent the preassigned machine required by the operation O.

Step 4. Assign the operation O into the schedule S by letting the starting time of the operation O equal the maximum between the earliest available time of the machine M and the completion time of the immediate-predecessor operation of the operation O. As a consequence, the completion time of the operation O equals its starting time given in this step plus its processing time τ.

Step 5. Update the earliest available time of the machine M to equal the completion time of the operation O.

Step 6. Increase t by 1. After that, if $t = mn$, stop and the schedule S is now completely constructed with the makespan equal to the maximum of the completion times of all mn operations; otherwise, repeat from Step 3.

Algorithm 2 is the procedure of the generic VNS algorithm which uses Algorithm 1 to decode operation-based permutations into JSP solutions. P_0, P_1, and P_2 in Algorithm 2 are the operation-based permutations which represent the job-shop schedules S_0, S_1, and S_2, respectively. The permutation P_0 is the current best found permutation, so the schedule S_0 is the current best found solution. As mentioned, each permutation of P_0, P_1, and P_2 is a sequence of mn integers consisting of the numbers $1, 2, 3, \ldots, n$, where each number (from 1 to n) is repeatedly m times. The solution neighborhood structures used by the generic VNS algorithm are generated based on the swap operator and the insert operator. The swap (i.e., interchange) and insert (i.e., shift) operators are commonly used in literature, for example, [7, 21, 33]. In this paper, the swap operator generates a neighbor of a specific operation-based permutation by randomly selecting two integers (of all mn integers) from two different positions in the permutation and then swapping the positions of the two selected integers. The insert operator is done by randomly selecting two integers (of all mn integers) from two different positions in the permutation, removing the first-selected integer from its old position, and then inserting it into the position in front of the second-selected integer.

Algorithm 2. It is a procedure of generic VNS algorithm.

Step 1. Let the user specify the type of this VNS algorithm to be a forward VNS algorithm or a reverse VNS algorithm and specify each of the operators A, B, C, and D to be the swap operator or the insert operator. If this VNS algorithm is specified to be a reverse VNS algorithm, then generate the reversed problem by letting the machine and the processing time of its operation O_{ij} ($i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$) be equal to the machine and the processing time of the operation O_{ik} ($i = 1, 2, \ldots, n$ and $k = m - j + 1$) of the original problem; then, replace the original problem by the reversed problem in all the following steps.

Step 2. Generate P_0 randomly as an initial current best operation-based permutation (or, as an option, P_0 can also be generated manually by the user) and then transform P_0 into the job-shop schedule S_0 by using Algorithm 1. Let the VNS algorithm's iteration $t = 0$.

Step 3. Process the shaking step by generating $P_1 = A(A(B(B(P_0))))$ and then transforming P_1 into a job-shop schedule S_1 by using Algorithm 1.

Step 4. Let the local search procedure's iteration $t_{\text{local}} = 0$. Process the local search procedure by Steps 4.1 to 4.5.

Step 4.1. Let $l_{\max} = 2$ and $l_{\text{count}} = 0$.

Step 4.2. If $l_{\text{count}} = 0$, then generate $P_2 = C(P_1)$; however, if $l_{\text{count}} = 1$, then generate $P_2 = D(P_1)$ instead. After that, transform P_2 into the job-shop schedule S_2 by using Algorithm 1.

Step 4.3. If the makespan of S_2 > the makespan of S_1, then increase l_{count} by 1. However, if the makespan of $S_2 \leq$ the makespan of S_1, then update l_{count} to equal 0, update P_1 to equal P_2, and also update S_1 to equal S_2.

Step 4.4. If $l_{\text{count}} = l_{\max}$, then go to Step 4.5; otherwise, repeat from Step 4.2.

Step 4.5. Increase t_{local} by 1. After that, if $t_{\text{local}} = mn(mn - 1)$, then go to Step 5; otherwise, repeat from Step 4.1.

Step 5. If the makespan of $S_1 \leq$ the makespan of S_0, then update P_0 to equal P_1 and also update S_0 to equal S_1.

Step 6. Increase t by 1. After that, check the conditions below:

(i) If the stopping criterion is not met, then repeat from Step 3.

(ii) If the stopping criterion is met and this VNS algorithm is specified in Step 1 to be a forward VNS algorithm, then stop and S_0 is the final solution.

(iii) If the stopping criterion is met and this VNS algorithm is specified in Step 1 to be a reverse VNS algorithm, then go to Step 7.

Step 7. Transform S_0 into S_{R0} by letting the starting time of O_{ij} ($i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$) of S_{R0} equal the makespan of S_0 minus the completion time of O_{ik} ($i = 1, 2, \ldots, n$ and $k = m - j + 1$) of S_0. As a consequence, the completion time of O_{ij} ($i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$) of S_{R0} then equals the makespan of S_0 minus the starting time of O_{ik} ($i = 1, 2, \ldots, n$ and $k = m - j + 1$) of S_0. Then, stop and S_{R0} is the final solution. Note that the makespan of S_{R0} is equal to the makespan of S_0.

The main steps of the generic VNS algorithm in Algorithm 2 are more clarified as follows. Step 1 requires the user to assign the type of the VNS algorithm which can be either a forward VNS algorithm or a reverse VNS algorithm. If the type of VNS algorithm is specified to be a reverse VNS algorithm, the precedence constraints of all operations of each job must be reversed. In Step 1, the user moreover has to specify that each of the operators A, B, C, and D is either the swap operator or the insert operator. Step 2 in Algorithm 2 generates P_0 as the initial current best operation-based permutation and then transforms the permutation P_0 into the schedule S_0 through the decoding procedure given in Algorithm 1.

Step 3 is the shaking step of the VNS algorithm, which modifies the operation-based permutation P_0 to the operation-based permutation P_1. To do so, Step 3 uses the operator B to P_0 for receiving the permutation $B(P_0)$, then uses the operator B again to the permutation $B(P_0)$ for receiving the permutation $B(B(P_0))$, then uses the operator A to the permutation $B(B(P_0))$ for receiving the permutation $A(B(B(P_0)))$, and finally uses the operator A again to the permutation $A(B(B(P_0)))$ for receiving the permutation $A(A(B(B(P_0))))$ as P_1.

Step 4 in Algorithm 2 is the VNS algorithm's local search procedure used to improve the permutation P_1; thus, at the end of Step 4, P_1 will be a local optimal solution. Later, Step 5 checks whether the permutation P_1 taken from Step 4 is better than or equal to the current best permutation P_0 or not. If so, the current best permutation P_0 will be updated to equal the permutation P_1.

Step 6 is the step to check whether the stopping criterion is satisfied or not and check whether this VNS algorithm is a forward VNS algorithm or a reverse VNS algorithm. If the stopping criterion is not satisfied, then the VNS algorithm will process its next iteration. If the stopping criterion is satisfied and this VNS algorithm is a forward VNS algorithm, then stop and the schedule S_0 is the final solution. If the stopping criterion is satisfied and this VNS algorithm is a reverse VNS algorithm, then process Step 7 in order to transform the schedule S_0 into the schedule S_{R0}, which is the reverse VNS algorithm's final solution usable for the original problem.

3.2. Proposed Forward VNS and Reverse VNS Algorithms. As previously mentioned, Algorithm 2 is the generic form for all forward VNS (FVNS) algorithms and all reverse VNS (RVNS) algorithms proposed in this section. Each of A, B, C, and D in Algorithm 2 can be specified to be either the swap operator or the insert operator, so there are a total of 32 VNS algorithms generated from Algorithm 2 including 16 FVNS

algorithms and 16 RVNS algorithms. For the identification purpose, let us name these 32 VNS algorithms in the format of *TABCD*. Let T represent the specified VNS type which can be either F = a forward VNS algorithm or R = a reverse VNS algorithm. Let A, B, C, and D here represent the specified operators for the generic operators A, B, C, and D in Algorithm 2, so each of A, B, C, and D can be either S = the swap operator or I = the insert operator. For example, FISIS refers to the forward VNS algorithm in which A is the insert operator, B is the swap operator, C is the insert operator, and D is the swap operator; RSISS means the reverse VNS algorithm in which A is the swap operator, B is the insert operator, C is the swap operator, and D is the swap operator.

Based on the format of *TABCD* above given, the 16 FVNS algorithms include FSSSS, FSSSI, FSSIS, FSSII, FSISS, FSISI, FSIIS, FSIII, FISSS, FISSI, FISIS, FISII, FIISS, FIISI, FIIIS, and FIIII, and the 16 RVNS algorithms include RSSSS, RSSSI, RSSIS, RSSII, RSISS, RSISI, RSIIS, RSIII, RISSS, RISSI, RISIS, RISII, RIISS, RIISI, RIIIS, and RIIII. Note that the eight FVNS algorithms, that is, FIIIS, FIISI, FISIS, FISSI, FSIIS, FSISI, FSSIS, and FSSSI, are slightly modified from the VNS algorithms of [19] in that their maximum iterations are extended from the 250th iteration to the 1,000th iteration. The modification just mentioned is due to the result of this paper's preliminary study which finds that the maximum iteration of 250th iteration makes each VNS algorithm stop prematurely before receiving its best returns in several hard-to-solve instances. The discussion about the proper maximum iteration will be given at the end of this section. In addition, FSISI is also found in [17] with a slightly different stopping criterion from the criterion used here; FSISI can thus be recognized as the original variant of all VNS algorithms given in this paper.

The 16 FVNS and 16 RVNS algorithms proposed in this section are compared in their performances on the 43 well-known benchmark instances, that is, ft06, ft10, and ft20 from [34] and la01–la40 from [35]. The number of all jobs (n) and the number of all machines (m) of all instances are given in parentheses in the form of *instance's name* (n, m) as follows: ft06 (6, 6), ft10 (10, 10), ft20 (20, 5), la01–la05 (10, 5), la06–la10 (15, 5), la11–la15 (20, 5), la16–la20 (10, 10), la21–la25 (15, 10), la26–la30 (20, 10), la31–la35 (30, 10), and la36–la40 (15, 15). In the experiment here, the proposed FVNS and RVNS algorithms are all coded in C# and executed on an Intel(R) Core(TM) i5 CPU processor M580 2.67 GHz. These VNS algorithms will be stopped when either the 1,000th iteration as the maximum iteration is reached or the optimal solution given by the published literature [7, 8, 24] is found. In other words, the stopping criterion in Step 6 of each VNS algorithm is either the VNS algorithm's iteration t = 1,000 or the makespan of S_0 = the optimal solution value given by the published literature. All VNS algorithms will be run once on each given instance with the same random seed number and the same initial operation-based permutation. For each instance, this paper uses the solution value deviation to evaluate the quality of the final solution given by each algorithm. For a specific instance, the *solution value deviation* is equal to 100% × (the algorithm's final solution value − the optimal solution value)/the optimal solution value. Thus, if the algorithm can reach the optimal solution value given in the published literature, the solution value deviation is then 0.000%. Note that, in this paper, a solution refers to a schedule and a solution value refers to a schedule's makespan.

Table 1 shows the solution value deviation (%) of every proposed VNS algorithm over a single run for each instance. The row *Best*, the last row in Table 1, provides the best solution value deviation found by all proposed VNS algorithms on each instance. The column *Avg*, the last column in Table 1, provides the average solution value deviation of all 43 instances of each VNS algorithm. Note that the instance will be absented from Table 1 if all 32 VNS algorithms can return the solution value deviations of 0.000% for it. This means Table 1 reports that all 32 VNS algorithms can reach the optimal solutions for the 28 instances of the total 43 instances, that is, ft06, la01–la15, la17–la19, la23, la26, la28, and la30–la35. Based on the results in Table 1, the list of the 32 VNS algorithms in ascending order of their average solution value deviations is FSSII, FISIS, FSSSI, RSSIS, RSISI, RISSI, FIIII, RIISI, RSSSI, FSIII, RIISS, FISSS, RSSSS, RISIS, RISII, RIIII, FSIIS, RIIIS, FIIIS, RSSII, RSIII, FIISI, FSSIS, RSIIS, FSSSS, FSISI, FISSI, RISSS, FIISS, FISII, RSISS, and FSISS.

The comparison results in terms of speed given in Table 2 indicate that the computational times per iteration of all proposed VNS algorithms are not significantly different. However, the total computational times of each VNS algorithm may differ from one another because the VNS algorithm will stop before reaching the maximum iteration if it can find the optimal solution. (Remember that the stopping criterion is to stop if either the optimal solution given by the published literature is found or the maximum iteration is reached.) Thus, the VNS algorithm performing better in solution quality tends to perform better in speed as well. However, if the stopping criterion is changed to be considered only on the maximum iteration, the speeds of all proposed VNS algorithms will differ very slightly. Table 2 shows the computational times and the number of iterations used by FSSII, FISIS, FSSSI, RSSIS, and RSISI which are the five best-performing VNS algorithms in Table 1. In Table 2, *Number of iters* means the number of all iterations used until the stopping criterion is met, *CPU time (sec.)* means the total computational time used until the stopping criterion is met, and the *CPU time/iter* means the computational time per iteration.

Figure 1 then reveals the proper maximum iterations for the proposed VNS algorithms. Based on the same data source used in Table 1, Figure 1 provides the average-solution-value-deviation-over-iteration plots of FSSII, FISIS, FSSSI, RSSIS, and RSISI on all 43 instances. Their average-solution-value-deviation-over-iteration plots are all formed in similar patterns which are reduced rapidly before the 500th iteration and then reduced slowly during the 500th iteration to 800th iteration. After that, the average solution value deviations of FSSII, FISIS, FSSSI, and RSSIS have not been improved after the 800th iteration, while RSISI is the only VNS algorithm which can improve its average solution value deviation until the 900th iteration. Based on this observation, this paper thus suggests that the maximum iterations of the proposed VNS algorithms should be in between the 800th iteration and the

Table 1: Solution value deviations (%) of 16 FVNS and 16 RVNS algorithms.

Algorithm	Instance															Avg
	ft10	ft20	la16	la20	la21	la22	la24	la25	la27	la29	la36	la37	la38	la39	la40	
FSSSS	0.000	0.000	0.000	0.000	0.096	0.000	0.321	0.716	0.891	1.389	0.473	0.716	0.000	0.000	0.491	0.118
FSSSI	0.000	0.000	0.000	0.000	0.096	0.000	0.321	0.000	0.243	1.128	0.000	0.716	0.000	0.000	0.164	0.062
FSSIS	0.000	0.000	0.000	0.000	0.096	0.324	0.321	0.000	0.243	1.476	0.000	1.002	0.920	0.324	0.164	0.113
FSSII	0.000	0.000	0.000	0.000	0.000	0.000	0.321	0.000	0.000	1.042	0.000	0.000	0.920	0.000	0.164	0.057
FSISS	0.860	0.000	0.000	0.000	0.669	0.324	0.321	0.000	0.891	1.736	0.000	0.000	0.920	0.000	0.491	0.144
FSISI	0.860	0.000	0.000	0.000	0.669	0.324	0.321	0.307	0.405	0.955	0.000	0.716	0.418	0.000	0.164	0.119
FSIIS	0.000	0.000	0.106	0.000	0.669	0.000	0.321	0.512	0.486	0.868	0.000	0.000	0.920	0.000	0.164	0.094
FSIII	0.000	0.000	0.000	0.000	0.765	0.000	0.321	0.512	0.000	0.868	0.000	0.000	0.920	0.000	0.164	0.083
FISSS	0.000	0.000	0.000	0.554	0.096	0.324	0.321	0.000	0.000	1.128	0.000	0.215	0.585	0.324	0.164	0.086
FISSI	0.860	0.000	0.000	0.000	0.000	0.324	0.321	0.512	0.405	2.170	0.000	0.000	0.585	0.000	0.164	0.124
FISIS	0.000	0.000	0.000	0.000	0.478	0.324	0.321	0.000	0.162	0.694	0.000	0.000	0.418	0.000	0.164	0.060
FISII	0.000	0.000	0.000	0.554	0.478	0.324	0.321	0.000	0.405	1.736	0.000	0.716	0.920	0.000	0.164	0.131
FIISS	0.753	0.000	0.000	0.000	0.000	0.000	0.321	0.205	1.134	1.910	0.000	0.000	0.000	0.973	0.164	0.127
FIISI	0.000	1.116	0.000	0.554	0.096	0.000	0.321	0.307	0.000	0.955	0.000	0.716	0.418	0.000	0.327	0.112
FIIIS	0.860	0.000	0.000	0.000	0.096	0.324	0.321	0.307	0.405	1.042	0.000	0.716	0.000	0.000	0.164	0.098
FIIII	0.000	0.000	0.000	0.000	0.574	0.000	0.321	0.307	0.486	1.042	0.000	0.000	0.418	0.000	0.164	0.077
RSSSS	0.000	0.000	0.000	0.000	0.669	0.000	0.535	0.000	0.405	2.083	0.000	0.000	0.000	0.000	0.164	0.090
RSSSI	0.000	0.000	0.000	0.000	0.382	0.324	0.321	0.307	0.000	0.955	0.079	0.000	0.920	0.000	0.164	0.080
RSSIS	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	2.257	0.000	0.000	0.000	0.000	0.573	0.066
RSSII	0.000	0.000	0.000	0.000	0.669	0.000	0.000	0.512	0.162	1.823	0.000	0.000	0.920	0.000	0.164	0.099
RSISS	1.505	0.000	0.000	0.000	0.191	0.000	0.321	0.614	0.000	2.604	0.000	0.000	0.418	0.000	0.164	0.135
RSISI	0.000	0.000	0.000	0.000	0.096	0.000	0.321	0.614	0.000	1.042	0.000	0.000	0.920	0.000	0.164	0.073
RSIIS	0.860	0.000	0.000	0.000	0.669	0.000	0.321	0.307	0.000	1.042	0.000	0.716	0.920	0.000	0.164	0.116
RSIII	0.860	0.000	0.000	0.000	1.243	0.000	0.321	0.000	0.000	1.042	0.000	0.000	0.920	0.000	0.164	0.106
RISSS	0.000	0.000	0.000	0.000	0.096	0.000	0.428	0.512	1.134	1.910	0.000	0.716	0.418	0.000	0.164	0.125
RISSI	0.000	0.000	0.000	0.000	0.574	0.000	0.321	0.512	0.000	1.215	0.000	0.000	0.418	0.000	0.164	0.074
RISIS	0.000	0.000	0.000	0.000	0.574	0.324	0.321	0.512	0.000	1.042	0.000	0.000	0.920	0.000	0.164	0.090
RISII	0.000	0.000	0.000	0.000	0.574	0.000	0.642	0.000	0.162	0.955	0.473	0.000	0.920	0.000	0.164	0.090
RIISS	0.000	0.000	0.000	0.000	0.478	0.324	0.321	0.512	0.405	1.042	0.000	0.000	0.418	0.000	0.164	0.085
RIISI	0.000	0.000	0.106	0.554	0.382	0.000	0.321	0.512	0.000	1.302	0.000	0.000	0.000	0.000	0.164	0.078
RIIIS	0.860	0.000	0.000	0.000	0.000	0.324	0.642	0.000	0.567	0.694	0.000	0.000	0.920	0.000	0.164	0.097
RIIII	0.860	0.000	0.000	0.000	0.478	0.324	0.321	0.000	0.567	1.302	0.000	0.000	0.000	0.000	0.164	0.093
Best	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.694	0.000	0.000	0.000	0.000	0.164	0.020

Remark: (i) the solution value deviations of the remaining 28 instances from the total 43 instances, which are not shown in Table 1, are 0.000% for all 32 VNS algorithms.
(ii) Avg refers to the average solution value deviation over all 43 instances for each VNS algorithm.

1,000th iteration. In case of requiring the short computational time, the maximum iteration is suggested to be the 500th iteration.

4. Proposed Multi-VNS Algorithm

In a brief explanation, the generic multi-VNS algorithm starts by randomly generating an operation-based permutation and then assigning this permutation into the multiple selected VNS algorithms as their initial current best operation-based permutations. Then, the multi-VNS algorithm runs these VNS algorithms and uses the best solution among the final solutions of these VNS algorithms as its final solution. The development of the multi-VNS algorithm proposed in this paper is motivated by four observations as follows:

(1) On the same JSP instance, two VNS algorithms with different settings of operators A, B, C, and D may not perform equally well. This property may also be true even when these two VNS algorithms both use the same initial operation-based permutation as well as the same random seed number. For example, according to Table 1, FSSII can find the optimal

Table 2: Number of iterations, CPU time, and CPU time per iteration used by each of FSSII, FISIS, FSSSI, RSSIS, and RSISI.

Instance	FSSII			FISIS			FSSSI			RSSIS			RSISI		
	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter
ft06	2	0.1	0.1	2	0.1	0.1	1	0.04	0.04	1	0.03	0.03	1	0.04	0.04
ft10	160	130	0.8	412	316	0.8	147	121	0.8	466	355	0.8	104	82	0.8
ft20	131	178	1.4	674	856	1.3	300	409	1.4	102	129	1.3	318	412	1.3
la01	5	0.6	0.1	1	0.1	0.1	1	0.1	0.1	1	0.1	0.1	3	0.4	0.1
la02	18	2	0.1	11	1	0.1	1	0.1	0.1	7	0.7	0.1	1	0.1	0.1
la03	68	8	0.1	208	22	0.1	33	4	0.1	3	0.3	0.1	186	20	0.1
la04	3	0.3	0.1	4	0.4	0.1	12	1	0.1	3	0.3	0.1	9	1	0.1
la05	1	0.2	0.2	1	0.2	0.2	1	0.2	0.2	1	0.1	0.1	1	0.2	0.2
la06	1	0.7	0.7	1	0.6	0.6	1	0.6	0.6	1	0.6	0.6	1	0.6	0.6
la07	1	0.6	0.6	1	0.5	0.5	1	0.5	0.5	1	0.5	0.5	1	0.5	0.5
la08	1	0.6	0.6	1	0.5	0.5	1	0.5	0.5	1	0.5	0.5	1	0.6	0.6
la09	1	0.7	0.7	1	0.6	0.6	1	0.6	0.6	1	0.6	0.6	1	0.6	0.6
la10	1	1	1.0	1	0.7	0.7	1	0.8	0.8	1	0.7	0.7	1	0.8	0.8
la11	1	2	2.0	1	2	2.0	1	2	2.0	1	2	2.0	1	2	2.0
la12	1	2	2.0	1	2	2.0	1	2	2.0	1	2	2.0	1	2	2.0
la13	1	2	2.0	1	2	2.0	1	2	2.0	1	2	2.0	1	2	2.0
la14	1	3	3.0	1	3	3.0	1	3	3.0	1	3	3.0	1	3	3.0
la15	1	2	2.0	1	1	1.0	1	2	2.0	1	1	1.0	1	1	1.0
la16	63	62	1.0	28	25	0.9	71	65	0.9	162	149	0.9	60	56	0.9
la17	11	10	0.9	62	52	0.8	7	6	0.9	55	46	0.8	3	3	1.0
la18	22	20	0.9	9	8	0.9	39	35	0.9	5	4	0.8	3	3	1.0
la19	19	17	0.9	122	100	0.8	7	6	0.9	36	29	0.8	26	21	0.8
la20	490	458	0.9	23	20	0.9	14	13	0.9	461	390	0.8	860	753	0.9
la21	517	1946	3.8	1000	3541	3.5	1000	3720	3.7	518	1822	3.5	1000	3611	3.6
la22	12	46	3.8	1000	3616	3.6	197	737	3.7	113	401	3.5	319	1170	3.7
la23	2	8	4.0	1	4	4.0	1	4	4.0	3	11	3.7	1	4	4.0
la24	1000	3751	3.8	1000	3550	3.6	1000	3713	3.7	721	2540	3.5	1000	3612	3.6
la25	392	5120	13.1	612	2228	3.6	419	1570	3.7	440	1583	3.6	1000	3715	3.7
la26	1	11	11.0	2	21	10.5	3	32	10.7	7	73	10.4	1	10	10.0
la27	499	5448	10.9	1000	10273	10.3	1000	10879	10.9	623	6362	10.2	601	6295	10.5
la28	26	293	11.3	53	552	10.4	66	720	10.9	95	995	10.5	53	569	10.7
la29	1000	11113	11.1	1000	10303	10.3	1000	10986	11.0	1000	10405	10.4	1000	10610	10.6
la30	1	12	12.0	3	32	10.7	3	34	11.3	1	11	11.0	1	11	11.0
la31	1	63	63.0	1	55	55.0	1	58	58.0	1	55	55.0	1	57	57.0
la32	1	58	58.0	1	52	52.0	1	54	54.0	1	51	51.0	1	53	53.0
la33	1	60	60.0	1	53	53.0	1	56	56.0	1	53	53.0	1	56	56.0
la34	1	56	56.0	1	51	51.0	1	53	53.0	1	50	50.0	1	53	53.0
la35	1	64	64.0	1	55	55.0	1	59	59.0	1	55	55.0	1	57	57.0
la36	731	9727	13.3	460	5647	12.3	635	8194	12.9	747	9083	12.2	223	2777	12.5
la37	390	4983	12.8	445	5340	12.0	1000	12637	12.6	64	765	12.0	446	5409	12.1
la38	1000	12556	12.6	1000	11802	11.8	705	8594	12.2	416	4907	11.8	1000	12026	12.0

Table 2: Continued.

Instance	FSSII			FISIS			FSSSI			RSSIS			RSISI		
	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter	Number of iters	CPU time (sec.)	CPU time/iter
la39	32	409	12.8	444	5306	12.0	226	2798	12.4	625	7446	11.9	802	9778	12.2
la40	1000	12664	12.7	1000	11964	12.0	1000	12478	12.5	1000	11825	11.8	1000	12160	12.2

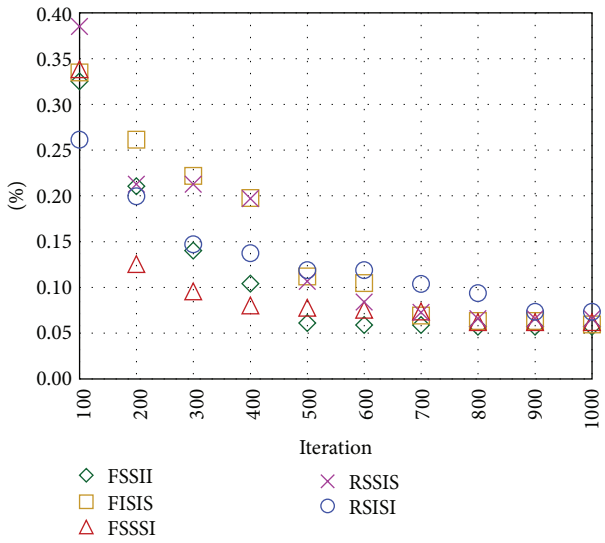

Figure 1: Average-solution-value-deviation-over-iteration plots of FSSII, FISIS, FSSSI, RSSIS, and RSISI.

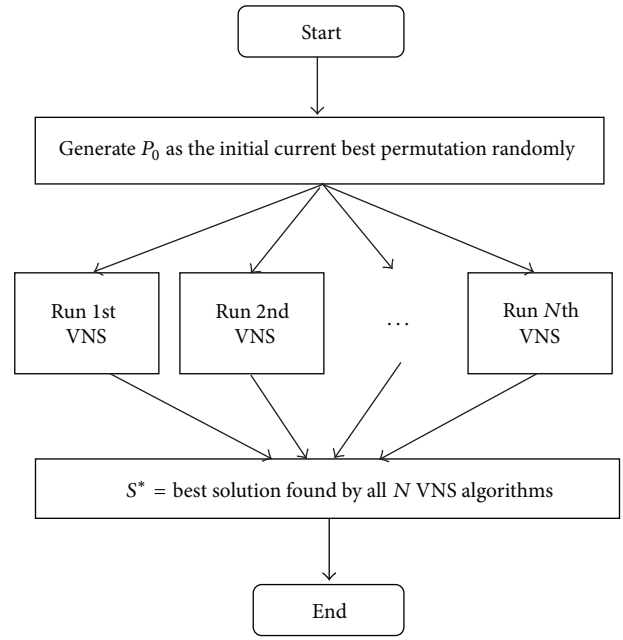

Figure 2: Flowchart of the generic multi-VNS algorithm.

solution for la37, while FSSIS cannot; FSSSS can find the optimal solution for la39, while FISSS cannot.

(2) A VNS algorithm which performs well on a particular instance may not perform as well on another instance. For example, FSSII can find the optimal solution for la37, but it cannot find the optimal solution for la29; FSISI can find the optimal solution for ft20, but it cannot find the optimal solution for ft10.

(3) A JSP instance which is hard to solve in its original form may be easier to be solved in its reversed problem, and vice versa. For example, according to Table 1, FSSII cannot find the optimal solution for la24, while RSSII can; FIIII can find the optimal solution for ft10, but RIIII cannot.

(4) It is impossible or very difficult to identify which scheduling direction (forward or reverse) is more efficient for a specific instance without experiments.

The aim of developing the multi-VNS algorithm is to handle with the four above-mentioned observations, and thus the multi-VNS algorithm should utilize each single operation-based permutation most efficiently. The generic multi-VNS algorithm, as shown in Figure 2, starts its process by generating P_0 as an identical initial operation-based permutation for the N VNS algorithms, that is, the 1st VNS, the 2nd VNS, ..., the Nth VNS. These N VNS algorithms are recommended to be different in their combinations of the A, B, C, and D operators as well as their VNS types (i.e., forward VNS or reverse VNS). The multi-VNS algorithm then runs each of these specified N VNS algorithms once on the being-considered JSP instance. After that, the best solution among the final solutions of all N given VNS algorithms is used as the final result of the multi-VNS algorithm and is abbreviated as S^*.

Although the proposed multi-VNS algorithm shown in Figure 2 can be run either in sequence or in parallel processing, this paper focuses only on the multi-VNS algorithm operated in sequence via a single stand-alone processor. Therefore, the multi-VNS algorithm in this paper runs the N VNS algorithms in order from the 1st VNS to the Nth VNS, sequentially. The procedure of the generic multi-VNS algorithm used in this paper is given in Algorithm 3.

Algorithm 3. It is a procedure of the generic multi-VNS algorithm.

Step 1. Assign the input parameter values as follows.

Step 1.1. Specify the specific 1st VNS, 2nd VNS, ..., Nth VNS algorithms for the multi-VNS algorithm differently in their combinations of the operators A, B, C, and D and the VNS types. For each VNS algorithm, each of A, B, C, and D can be either the swap operator or the insert operator, while the VNS type can be either a forward VNS algorithm or a reverse VNS algorithm.

Step 1.2. Assign the stopping criterion for each of the 1st VNS, 2nd VNS, ..., Nth VNS algorithms.

Step 1.3. Assign the stopping criterion for the multi-VNS algorithm.

Step 2. Randomly generate P_0 as an identical initial operation-based permutation for all N VNS algorithms. Let the multi-VNS algorithm's iteration $d = 1$.

Step 3. Run the dth VNS algorithm using the permutation P_0 generated in Step 2 as its initial current best operation-based permutation.

Step 4. After the dth VNS algorithm is stopped, let S_0^d be equal to S_0 of the dth VNS algorithm if the dth VNS algorithm is a forward VNS algorithm; however, let S_0^d be equal to S_{R0} of the dth VNS algorithm if the dth VNS algorithm is a reverse VNS algorithm. (Remember that S_0 is the final solution for the forward VNS algorithm, while S_{R0} is the final solution for the reverse VNS algorithm.)

Step 5. Update the best found solution of the multi-VNS algorithm or S^* using Steps 5.1 and 5.2.

Step 5.1. If $d = 1$, let S^* equal S_0^d and let the makespan of S^* equal the makespan of S_0^d.

Step 5.2. If $d \geq 2$ and the makespan of S_0^d is less than the makespan of S^*, update S^* to equal S_0^d and also update the makespan of S^* to equal the makespan of S_0^d.

Step 6. If the stopping criterion of the multi-VNS algorithm is met, then stop and let the final result of the multi-VNS algorithm equal S^*; otherwise, increase the value of d by 1 and repeat from Step 3.

Algorithm 3 is the generic multi-VNS algorithm where the user must specify the 1st VNS algorithm to the Nth VNS algorithm in their A, B, C, and D operators, VNS types, and stopping conditions. These N VNS algorithms of the multi-VNS algorithm in Algorithm 3 must be selected very carefully because more VNS algorithms used in the multi-VNS algorithm may consume more resources, especially the computational time, without any guarantees to find a better solution. Moreover, the uses of the same set of the N specific VNS algorithms in different orders may consume different computational times. This is because the stopping criterion of the multi-VNS algorithm is specified to stop when either the Nth iteration is reached ($d = N$) or the optimal solution given by the published literature is found; thus, if the optimal solution is found by the dth VNS algorithm, the multi-VNS algorithm will stop at the dth VNS algorithm and will not continue running the $N - d$ remaining VNS algorithms. Hence, to make the multi-VNS algorithm perform most efficiently in computational time, the order of the N VNS algorithms must be selected carefully as well. In this paper, the method of selecting the proper value of N and also selecting the specific 1st VNS to Nth VNS algorithms for the multi-VNS algorithm is given in Algorithm 4. Remember that, for each instance, the solution value deviation of each VNS algorithm is equal to 100% × (the VNS algorithm's final solution value − the optimal solution value)/the optimal solution value.

Algorithm 4. It is a method of selecting the value of N and selecting the specific 1st to Nth VNS algorithms for the multi-VNS algorithm.

Step 1. Let $d = 1$. Run all on-hand VNS algorithms once on all given instances and then receive the final solutions of all these on-hand VNS algorithms. Then, compute the following based on the final solutions received.

Table 3: The 1st VNS to dth VNS algorithms and $ABSVD_d$ value in each iteration of Algorithm 4 based on results from Table 1.

d	The 1st VNS to dth VNS	$ABSVD_d$
All	All 32 VNS algorithms	0.020%
1	FSSII	0.057%
2	FSSII, FISIS	0.037%
3	FSSII, FISIS, FSSSI	0.027%
4	FSSII, FISIS, FSSSI, RSSIS	0.020%

Step 1.1. For each on-hand VNS algorithm, compute the solution value deviation of every given instance and then compute the average of the solution value deviations of all given instances. Then, list all on-hand VNS algorithms in ascending order of their average solution value deviations (from left to right).

Step 1.2. For every given instance, find $BSVD_{\text{All}}$, that is, the best (lowest) solution value deviation found by all on-hand VNS algorithms. Then, compute $ABSVD_{\text{All}}$, that is, the average of the $BSVD_{\text{All}}$ values of all given instances.

Step 2. Assign the current leftmost VNS algorithm among all as-yet-unassigned on-hand VNS algorithms on the list given in Step 1.1 as the dth VNS algorithm.

Step 3. For every given instance, find $BSVD_d$, that is, the best (lowest) solution value deviation found by all the already-specified 1st VNS to dth VNS algorithms. Then, compute $ABSVD_d$, that is, the average of the $BSVD_d$ values of all given instances.

Step 4. If $ABSVD_d$ is equal to $ABSVD_{\text{All}}$, then stop and let N equal d; this means the specific 1st to Nth VNS algorithms are completely selected. On the other hand, if $ABSVD_d$ is greater than $ABSVD_{\text{All}}$, then increase d by 1 and repeat from Step 2.

As shown in Section 3.2, this paper proposes 16 FVNS and 16 RVNS algorithms, so the number of all on-hand VNS algorithms is thus 32. Based on the results from Table 1, the list of all 32 VNS algorithms in ascending order of their average solution value deviations on all 43 instances is FSSII, FISIS, FSSSI, RSSIS, RSISI, RISSI, FIIII, RIISI, RSSSI, FSIII, RIISS, FISSS, RSSSS, RISIS, RISII, RIIII, FSIIS, RIIIS, FIIIS, RSSII, RSIII, FIISI, FSSIS, RSIIS, FSSSS, FSISI, FISSI, RISSS, FIISS, FISII, RSISS, and FSISS. According to the results in Table 1, Algorithm 4 first computes $ABSVD_{\text{All}} = 0.020\%$. After that, Algorithm 4 assigns FSSII as the 1st VNS algorithm with $ABSVD_1 = 0.057\%$, FISIS as the 2nd VNS algorithm with $ABSVD_2 = 0.037\%$, FSSSI as the 3rd VNS algorithm with $ABSVD_3 = 0.027\%$, and RSSIS as the 4th VNS algorithm with $ABSVD_4 = 0.020\%$, respectively. Since $ABSVD_4$ equals $ABSVD_{\text{All}}$, Algorithm 4 stops here and thus $N = 4$. This means, based on Table 1 results, Algorithm 4 suggests that the multi-VNS algorithm should use FSSII, FISIS, FSSSI, and RSSIS as the 1st to 4th VNS algorithms, respectively. Table 3 summarizes the specified 1st VNS algorithm to the dth VNS

algorithm and the $ABSVD_d$ value given in each iteration of Algorithm 4 based on the results from Table 1.

5. Performance Evaluation for Multi-VNS

This section will provide a specific multi-VNS algorithm, which is Algorithm 3 using the N specific VNS algorithms and the other preassigned input parameter values. Later, an experiment will be conducted in order to evaluate the performance of the multi-VNS algorithm on the JSP instances. The specific VNS algorithms and input parameter values for Algorithm 3 along with the experimental conditions are given below.

(1) The multi-VNS algorithm uses the four specified VNS algorithms ($N = 4$), that is, FSSII, FISIS, FSSSI, and RSSIS as the 1st VNS, 2nd VNS, 3rd VNS, and 4th VNS algorithms, respectively. (Note that these VNS algorithms are selected by Algorithm 4 based on Table 1 results.)

(2) The stopping criterion of each specified VNS algorithm (i.e., FSSII, FISIS, FSSSI, and RSSIS) used in the multi-VNS algorithm is to stop when either the optimal solution value (i.e., the optimal makespan) given by the published literature is found or the 1,000th iteration is reached ($t = 1{,}000$).

(3) The stopping criterion of the multi-VNS algorithm is to stop when either the optimal solution value (i.e., the optimal makespan) given by the published literature is found or the multi-VNS algorithm's Nth iteration is reached.

(4) On each single run of the multi-VNS algorithm, the multi-VNS algorithm runs the specific 1st VNS, 2nd VNS, 3rd VNS, and 4th VNS algorithms with the same random seed number and also the same initial operation-based permutation.

(5) The multi-VNS algorithm is coded in C# and executed on an Intel(R) Core(TM) i5 CPU processor M580 2.67 GHz.

(6) The multi-VNS algorithm in this paper will be repeated for five runs with different random seed numbers and also different initial operation-based permutations. The multi-VNS algorithm in the 1st run uses the same random seed number and the same initial operation-based permutation as the VNS algorithms in the experiment in Table 1.

(7) The performance of the multi-VNS algorithm will be tested on the 43 benchmark JSP instances including ft06, ft10, and ft20 from [34] and la01 to la40 from [35].

Table 4 shows the final solution values from the five runs of the multi-VNS algorithm with the above-given settings. The column *Best* in Table 4 provides the best among the final solution values of the five runs for every instance. The column *Avg* provides the average of the final solution values of the five runs. The column *Avg CPU time (sec.)* provides the average computational time in second over the five runs of each instance.

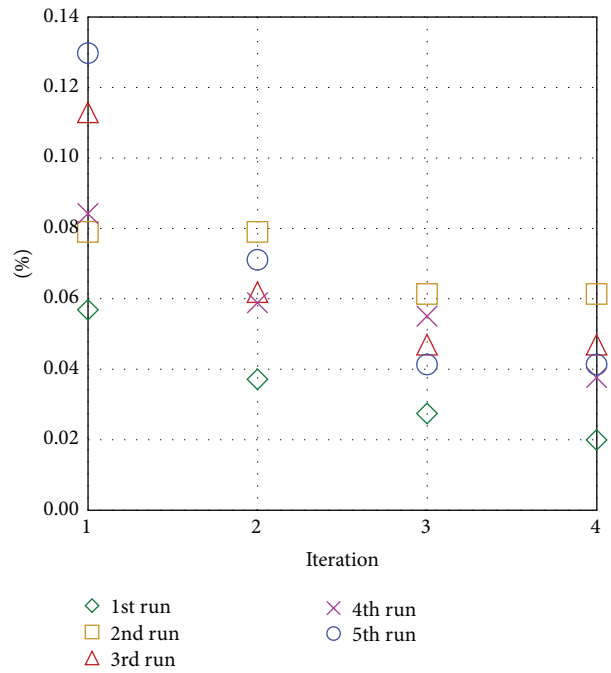

FIGURE 3: Average-solution-value-deviation-over-iteration plots for five runs of the multi-VNS algorithm.

Figure 3 shows the average solution value deviations of 43 instances over the multi-VNS algorithm's iterations (d values) of each run. The average solution value deviation in Figure 3 is, on average, reduced in high rate from the 1st iteration to the 3rd iteration and then reduced in lower rate from the 3rd iteration to the 4th iteration. Figure 3 also shows that the multi-VNS algorithm improves the average solution value deviation of its 1st VNS (i.e., FSSII here) for 0.05% on average. The rate of improvement of 0.05% may seem to be low; however, if considering only the instances where FSSII cannot find the optimal solutions in Table 1 experiment (i.e., la24, la29, la38, and la40), the rate of improvement will be 0.21% on average.

Later, the performance of the multi-VNS algorithm is then compared to the performances of the five high-performing metaheuristic algorithms in published literature, that is, the GA with the extended Akers graphical method [8], the hybrid GA [9], the two-level PSO [14], the VNS algorithm [17], the memetic algorithm [23], and also FSSII, as the best-performing VNS algorithm in Table 1. In this section, the performances of these algorithms are compared only in their best found solution values. The computational times of these algorithms will not be compared here because of the differences in their stopping criteria, programming languages, and CPU processor specifications. The best found solution values of the algorithms in [8, 9, 14, 17] are taken from their own articles, the best found solution values of the multi-VNS algorithm are taken from the column *Best* in Table 4, and the best found solution values of FSSII are taken from the results over 5 runs in an additional experiment here.

Table 4: Final solution value by each run of the multi-VNS algorithm.

Instance	1st run	2nd run	3rd run	4th run	5th run	Best	Avg	Avg CPU time (sec.)
ft06	55	55	55	55	55	55	55	0.1
ft10	930	930	930	930	930	930	930	546
ft20	1165	1165	1165	1165	1165	1165	1165	382
la01	666	666	666	666	666	666	666	0.3
la02	655	655	655	655	655	655	655	2
la03	597	597	597	597	597	597	597	37
la04	590	590	590	590	590	590	590	0.5
la05	593	593	593	593	593	593	593	0.2
la06	926	926	926	926	926	926	926	1
la07	890	890	890	890	890	890	890	1
la08	863	863	863	863	863	863	863	1
la09	951	951	951	951	951	951	951	1
la10	958	958	958	958	958	958	958	1
la11	1222	1222	1222	1222	1222	1222	1222	2
la12	1039	1039	1039	1039	1039	1039	1039	2
la13	1150	1150	1150	1150	1150	1150	1150	2
la14	1292	1292	1292	1292	1292	1292	1292	3
la15	1207	1207	1207	1207	1207	1207	1207	2
la16	945	945	945	945	945	945	945	260
la17	784	784	784	784	784	784	784	42
la18	848	848	848	848	848	848	848	16
la19	842	842	842	842	842	842	842	30
la20	902	902	902	902	902	902	902	522
la21	1046	1046	1046	1047	1047	1046	1046.4	8526
la22	927	927	927	927	927	927	927	995
la23	1032	1032	1032	1032	1032	1032	1032	11
la24	935	938	938	938	938	935	937.4	14258
la25	977	977	977	977	977	977	977	4187
la26	1218	1218	1218	1218	1218	1218	1218	26
la27	1235	1235	1237	1236	1238	1235	1236.2	14368
la28	1216	1216	1216	1216	1216	1216	1216	497
la29	1160	1172	1163	1163	1163	1160	1164.2	42644
la30	1355	1355	1355	1355	1355	1355	1355	12
la31	1784	1784	1784	1784	1784	1784	1784	61
la32	1850	1850	1850	1850	1850	1850	1850	56
la33	1719	1719	1719	1719	1719	1719	1719	59
la34	1721	1721	1721	1721	1721	1721	1721	55
la35	1888	1888	1888	1888	1888	1888	1888	62
la36	1268	1268	1268	1268	1268	1268	1268	4843
la37	1397	1397	1397	1397	1397	1397	1397	3192
la38	1196	1201	1201	1196	1196	1196	1198	40265
la39	1233	1233	1233	1233	1233	1233	1233	5941
la40	1224	1224	1224	1224	1224	1224	1224	48849

The stopping criterion of FSSII is to stop when either the optimal solution given in the published literature is found or the 1,000th iteration is met. VNS in [17] is the FSISI algorithm which uses the stopping criterion that is to stop when either the optimal solution given in the published literature is found, the 1,000th iteration is reached, or no solution improvements within 250 consecutive iterations happen. The comparison results are then given in Table 5. For each instance, Table 5 provides the best found solution value of each algorithm compared to the optimal solution value given in [7, 8, 24].

Table 5: Comparison of best found solution values of algorithms.

Instance	Optimal solution value	Multi-VNS	FSSII	GA [8]	GA [9]	PSO [14]	VNS [17]	MA [23]
ft06	55	55	55	55	55	55	n/a	55
ft10	930	930	930	930	930	930	n/a	930
ft20	1165	1165	1165	1165	1165	1165	n/a	1165
la01	666	666	666	666	666	666	666	666
la02	655	655	655	655	655	655	655	655
la03	597	597	597	597	597	597	597	597
la04	590	590	590	590	590	590	590	590
la05	593	593	593	593	593	593	593	593
la06	926	926	926	926	926	926	926	926
la07	890	890	890	890	890	890	890	890
la08	863	863	863	863	863	863	863	863
la09	951	951	951	951	951	951	951	951
la10	958	958	958	958	958	958	958	958
la11	1222	1222	1222	1222	1222	1222	1222	1222
la12	1039	1039	1039	1039	1039	1039	1039	1039
la13	1150	1150	1150	1150	1150	1150	1150	1150
la14	1292	1292	1292	1292	1292	1292	1292	1292
la15	1207	1207	1207	1207	1207	1207	1207	1207
la16	945	945	945	945	945	945	945	945
la17	784	784	784	784	784	784	784	784
la18	848	848	848	848	848	848	848	848
la19	842	842	842	842	842	842	842	842
la20	902	902	902	902	907	907	902	902
la21	1046	1046	1046	1046	1046	1046	1047	1055
la22	927	927	927	927	927	935	927	927
la23	1032	1032	1032	1032	1032	1032	1032	1032
la24	935	935	938	935	953	944	937	940
la25	977	977	977	977	986	984	977	984
la26	1218	1218	1218	1218	1218	1218	1218	1218
la27	1235	1235	1235	1235	1256	1258	1235	1261
la28	1216	1216	1216	1216	1232	1218	1216	1216
la29	1152	1160	1164	1153	1196	1184	1163	1190
la30	1355	1355	1355	1355	1355	1355	1355	1355
la31	1784	1784	1784	1784	1784	1784	1784	1784
la32	1850	1850	1850	1850	1850	1850	1850	1850
la33	1719	1719	1719	1719	1719	1719	1719	1719
la34	1721	1721	1721	1721	1721	1721	1721	1721
la35	1888	1888	1888	1888	1888	1888	1888	1888
la36	1268	1268	1268	1268	1279	1278	1268	1281
la37	1397	1397	1397	1397	1408	1410	1397	1431
la38	1196	1196	1201	1196	1219	1221	1201	1216
la39	1233	1233	1233	1233	1246	1251	1233	1241
la40	1222	1224	1224	1222	1241	1229	1224	1233
Avg deviation	0.000%	0.020%	0.045%	0.002%	0.382%	0.320%	0.046%	0.330%

For each algorithm, the row *Avg deviation* in Table 5 provides the average of the solution value deviations of all best found solutions.

According to the results in Table 5, GA [8] performs best with the average deviation of 0.002%; it can find the optimal solutions for 42 instances over all 43 instances. The multi-VNS performs as the second-best with the average deviation of 0.020%, and it can find the optimal solutions for 41 instances. The algorithms FSSII and VNS [17] perform equally well as the third and the fourth and are followed by PSO [14], GA [9], and MA [23].

6. Conclusions

This paper proposed the 16 forward VNS algorithms and the 16 reverse VNS algorithms for JSP. It has been found that, on many benchmark instances, the VNS algorithms perform unequally even when using the same initial operation-based permutation. Thus, for utilizing each initial operation-based permutation most efficiently, this paper developed the multi-VNS algorithm which assigns the same initial operation-based permutation into the selected specific VNS algorithms (i.e., FSSII, FISIS, FSSSI, and RSSIS in this paper), runs these VNS algorithms, and uses the best solution found by all these VNS algorithms as its final result. This paper then compared the multi-VNS algorithm's performance with the performances of the six other high-performing algorithms. The comparison results indicate that the multi-VNS algorithm is the second-best algorithm in terms of solution quality. Over all the 43 benchmark instances used, the multi-VNS algorithm can find the optimal solutions for the 41 instances and the very-near optimal solutions for the two instances. The further work of this research is enhancing the performances of the VNS algorithms in terms of both solution quality and computational time by applying a combination of multiple techniques.

Competing Interests

The author declares that there are no competing interests.

References

[1] K. R. Baker and D. Trietsch, *Principles of Sequencing and Scheduling*, John Wiley & Sons, Hoboken, NJ, USA, 2009.

[2] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Springer, New York, NY, USA, 4th edition, 2012.

[3] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Design*, John Wiley & Sons, New York, NY, USA, 1996.

[4] E. Nowicki and C. Smutnicki, "An advanced tabu search algorithm for the job shop problem," *Journal of Scheduling*, vol. 8, no. 2, pp. 145–159, 2005.

[5] C. Y. Zhang, P. Li, Z. Guan, and Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3229–3242, 2007.

[6] R. K. Suresh and K. M. Mohanasundaram, "Pareto archived simulated annealing for job shop scheduling with multiple objectives," *International Journal of Advanced Manufacturing Technology*, vol. 29, no. 1-2, pp. 184–196, 2006.

[7] M. F. Tasgetiren, M. Sevkli, Y.-C. Liang, and M. M. Yenisey, "A particle swarm optimization and differential evolution algorithms for job shop scheduling problem," *International Journal of Operations Research*, vol. 3, no. 2, pp. 120–135, 2006.

[8] 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.

[9] J. F. Gonçalves, J. J. Mendes, and M. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European Journal of Operational Research*, vol. 167, no. 1, pp. 77–95, 2005.

[10] M. Watanabe, K. Ida, and M. Gen, "A genetic algorithm with modified crossover operator and search area adaptation for the job-shop scheduling problem," *Computers and Industrial Engineering*, vol. 48, no. 4, pp. 743–752, 2005.

[11] M. F. N. Maghfiroh, A. Darnawan, and V. F. Yu, "Genetic algorithm for job shop scheduling problem: a case study," *International Journal of Innovation, Management and Technology*, vol. 4, no. 1, pp. 137–140, 2013.

[12] N. H. Moin, O. C. Sin, and M. Omar, "Hybrid genetic algorithm with multiparents crossover for job shop scheduling problems," *Mathematical Problems in Engineering*, vol. 2015, Article ID 210680, 12 pages, 2015.

[13] M. Gen, Y. Tsujimura, and E. Kubota, "Solving job-shop scheduling problem using genetic algorithms," in *Proceedings of the 16th International Conference on Computers and Industrial Engineering*, pp. 576–579, Ashikaga, Japan, 1994.

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

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

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

[17] P. Pongchairerks and V. Kachitvichyanukul, "A comparison between algorithms VNS with PSO and VNS without PSO for job-shop scheduling problems," *International Journal of Computational Science*, vol. 1, no. 2, pp. 179–191, 2007.

[18] M. Sevkli and M. E. Aydin, "Variable neighbourhood search for job shop scheduling problems," *Journal of Software*, vol. 1, no. 2, pp. 34–39, 2006.

[19] P. Pongchairerks, "Variable neighbourhood search algorithms applied to job-shop scheduling problems," *International Journal of Mathematics in Operational Research*, vol. 6, no. 6, pp. 752–774, 2014.

[20] L.-L. Liu, R.-S. Hu, X.-P. Hu, G.-P. Zhao, and S. Wang, "A hybrid PSO-GA algorithm for job shop scheduling in machine tool production," *International Journal of Production Research*, vol. 53, no. 19, pp. 5755–5781, 2015.

[21] J.-Q. Li, S.-X. Xie, Q.-K. Pan, and S. Wang, "A hybrid artificial bee colony algorithm for flexible job shop scheduling problems," *International Journal of Computers, Communications and Control*, vol. 6, no. 2, pp. 286–296, 2011.

[22] A. Udomsakdigool and V. Kachitvichyanukul, "Two-way scheduling approach in ant algorithm for solving job shop

problem," *Industrial Engineering and Management Systems*, vol. 5, no. 2, pp. 68–75, 2007.

[23] L. Gao, G. Zhang, L. Zhang, and X. Li, "An efficient memetic algorithm for solving the job shop scheduling problem," *Computers and Industrial Engineering*, vol. 60, no. 4, pp. 699–705, 2011.

[24] C. Zhang, P. Li, Y. Rao, and S. Li, "A new hybrid GA/SA algorithm for the job shop scheduling problem," in *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP '05)*, pp. 246–259, Lausanne, Switzerland, April 2005.

[25] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[26] P. Hansen and N. Mladenović, "Variable neighborhood search: principles and applications," *European Journal of Operational Research*, vol. 130, no. 3, pp. 449–467, 2001.

[27] J. A. Moreno-Pérez, P. Hansen, and N. Mladenović, "Parallel variable neighborhood searchs," Tech. Rep., Group of Intelligent Computing, Universidad de La Laguna, La Laguna, Spain, 2004.

[28] M. M. Gargari and M. S. F. Niasar, "A dynamic discrete berth allocation problem for container terminals," in *Proceedings of the Conference on Maritime-Port Technology*, Trondheim, Norway, 2014.

[29] I. Piriyaniti and P. Pongchairerks, "Variable neighbourhood search algorithms for asymmetric travelling salesman problems," *International Journal of Operational Research*, vol. 18, no. 2, pp. 157–170, 2013.

[30] T. Davidovic, T. G. Crainic, and T. Davidović, "Parallelization strategies for variable neighborhood search," Tech. Rep., CIRRELT, Université de Montréal, Montreal, Canada, 2013.

[31] T. Yamada and R. Nakano, "Fusion of crossover and local search," in *Proceedings of the IEEE International Conference on Industrial Technology*, pp. 426–430, Shanghai, China, December 1994.

[32] J. Alcaraz and C. Maroto, "A robust genetic algorithm for resource allocation in project scheduling," *Annals of Operations Research*, vol. 102, no. 1–4, pp. 83–109, 2001.

[33] E. J. Anderson, C. A. Glass, and C. N. Potts, "Machine scheduling," in *Local Search in Combinatorial Optimization*, E. Aarts and J. K. Lenstra, Eds., pp. 361–414, Princeton University Press, Princeton, NJ, USA, 2003.

[34] 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 Cliffs, NJ, USA, 1963.

[35] S. Lawrence, *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pa, USA, 1984.