Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 3586731, 8 pages
https://doi.org/10.1155/2018/3586731

Research Article

An Order Effect of Neighborhood Structures in Variable Neighborhood Search Algorithm for Minimizing the Makespan in an Identical Parallel Machine Scheduling

Ibrahim Alharkan,[1] Khaled Bamatraf,[1] Mohammed A. Noman,[1] Husam Kaid,[1] Emad S. Abouel Nasr,[1,2] and Abdulaziz M. El-Tamimi[1]

[1] *Industrial Engineering Department, College of Engineering, King Saud University, P.O. Box 800, Riyadh 11421, Saudi Arabia*
[2] *Faculty of Engineering, Mechanical Engineering Department, Helwan University, Cairo 11732, Egypt*

Correspondence should be addressed to Mohammed A. Noman; mmohammed1@ksu.edu.sa

Received 11 September 2017; Revised 6 February 2018; Accepted 27 February 2018; Published 22 April 2018

Academic Editor: Ton D. Do

Variable neighborhood search (VNS) algorithm is proposed for scheduling identical parallel machine. The objective is to study the effect of adding a new neighborhood structure and changing the order of the neighborhood structures on minimizing the makespan. To enhance the quality of the final solution, a machine based encoding method and five neighborhood structures are used in VNS. Two initial solution methods which were used in two versions of improved VNS (IVNS) are employed, namely, longest processing time (LPT) initial solution, denoted as HIVNS, and random initial solution, denoted as RIVNS. The proposed versions are compared with LPT, simulated annealing (SA), genetic algorithm (GA), modified variable neighborhood search (MVNS), and improved variable neighborhood search (IVNS) algorithms from the literature. Computational results show that changing the order of neighborhood structures and adding a new neighborhood structure can yield a better solution in terms of average makespan.

1. Introduction

Identical parallel machine scheduling (IPMS) with the objective of minimizing the makespan is one of the combinational optimization problems. It is known to be NP-hard by Garey and Johnson [1] since it does not have a polynomial time algorithm. Exact algorithms such as branch and bound [2] and cutting plane algorithms [3] solve this type of IPM and find optimal solution for small size instances. As the problem size increases, the exact algorithms are inefficient and take much time to get a solution.

That disadvantages bring a need for heuristics and metaheuristics that give optimal or near optimal solution within a reasonable amount of time. Longest Processing Time Rule (LPT) proposed by Mokotoff [4] is the first heuristic applied in IPMS which has a tight worst case performance of bound of 4/3–1/3m, where m is the number of parallel machines. LPT is based on distributing jobs on machines according to maximum processing time and the remaining jobs go one by one to the least loaded machine until assigning all the jobs to the machines. The LPT heuristic performs well for makespan criteria but the solution obtained is often local optima. Later, Coffman et al. [5] proposed MULTIFIT algorithm that is based on techniques from bin-packing. Blackstone Jr. and Phillips [6] proposed a simple heuristic for improving LPT sequence by exchange jobs between processors to reduce makespan. Lee and Massey [7] combine two heuristics, LPT and MULTIFIT, to form a new one. The heuristic uses LPT heuristic as an initial solution for the MULTIFIT heuristic. The performance of the combined heuristic is better than LPT and the error bound is not worse than the MULTIFIT. Yue [8] proved the bound for MULTIFIT to be 13/11. Lee and Massey [9] extend the MULTIFIT algorithm and show that the error bound of implementing the algorithm is only 1/10. Garey and Johnson [1] proposed that 3-phase composite heuristic consists of constructive phase and two improvement phases with no preliminary sort of processing times. They showed that their proposed heuristic is quicker than LPT. Ho and

Wong [10] introduce Two-Machine Optimal Scheduling which uses lexicographic search. Their method performs better that LPT, MULTIFIT, and MULTIFIT extension algorithm and it takes less amount of CPU time than MULTIFIT and MULTIFIT extension algorithms.

Riera et al. [11] proposed two approximate algorithms that use LPT as an initial solution and compare them with dynamic programming and MULTIFIT algorithms. Algorithm 1 uses exchange between two jobs to improve the makespan. Algorithm 2 schedules a job such that the completion time and process time of the selected job are near the bound. Their second algorithm is compared with MULTIFIT algorithms and results showed similarity to the MULTIFIT algorithm, but their algorithm reduces CPU time with respect to MULTIFIT heuristic. Cheng and Gen [12] applied memetic algorithm to minimize the maximum weighted absolute lateness on PMS and showed that it outperforms genetic algorithm and the conventional heuristics. Ghomi and Ghazvini [13] proposed a pairwise interchange algorithm, and it gave near optimal solution in a short period of time. Min and Cheng [14] proposed a genetic algorithm GA using machine code. They showed that GA outperforms LPT and SA and is suitable for large scale IPMS problems. Gupta and Ruiz-Torres [15] proposed a LISTFIT heuristic based on bin-backing and list scheduling. The LISTFIT generate an optimal or near optimal solution and outperforms LPT, MULTIFIT, and COMBINE heuristics. Costa et al. [16] proposed algorithm inspired by the immune systems of vertebrate animals. Lee et al. [17] proposed a simulated annealing (SA) approach for makespan minimization on IPMS. It chooses LPT as an initial solution. Computational results showed that the SA heuristic outperforms the LISTFIT and pairwise interchange (PI) algorithms. Moreover, it is efficient for large scale problems. Tang and Luo [18] propose a new ILS algorithm combining with a variable number of cyclic exchanges. Experiments show that the algorithm is efficient for $Pm \parallel C_{\max}$. Akyol and Bayhan [19] proposed a dynamical neural network that employs parameters of time varying penalty. The simulation results showed that the proposed algorithm generated feasible solutions and it found better makespan when compared to LPT. Kashan and Karimi [20] presented discrete particle swarm optimization (DPSO) algorithm for makespan minimization. Computational results showed that hybridized DPSO (HDPSO) algorithm outperforms both SA and DPSO algorithms. Sevkli and Uysal [21] proposed modified variable neighborhood search (MVNS) which is based on exchange and move neighborhood structures. Computational results demonstrated that the proposed algorithm outperforms both GA and LPT algorithms. Min and Cheng [14] proposed a harmony search (HS) algorithm with dynamic subpopulation (DSHS). Results show that DSHS algorithm outperforms SA and HDPSO for many instances. Moreover, the execution time is less than 1 sec. for all computations. Chen et al. [22] proposed discrete harmony search (DHS) algorithm that uses discrete encoding scheme to initialize the harmony memory (HM), then the improvisation scheme for generating new harmony is redefined for suitability for solving the combinational optimization problem. In addition, the study made hybridizing a local search method with DHS to increase the speed of local search. Computational results show that the DHS algorithm is very competitive when compared with other heuristics in the literature. Jing and Jun-qing [23] proposed efficient variable neighborhood search that uses four neighborhood structures and has two versions. One version uses LPT sequence as an initial solution. The other version uses random sequence as an initial solution. A computational result demonstrates that EVNS is efficient in searching global or near global optimum. M. Sevkli and A. Z. Sevkli [24] proposed stochastically perturbed particle swarm optimization algorithm (SPPSO). The algorithm compared two recent PSO algorithms. It is concluded that SPPSO algorithm has produced better results than DPSO and PSOspv in terms of the optimal solutions number. Laha [25] proposed an improved simulated annealing (SA) heuristic. Computational results show that the proposed heuristic is better than that produced by the best-known heuristic in the literature. Other advantages of it are the ease of implementation. In this paper, the proposed algorithm of Jing and Jun-qing [23] in their paper "efficient variable neighborhood search for identical parallel machines scheduling" is used with some changes on it. One of the changes is changing in the order of the neighborhood structures and the other change is adding another neighborhood structure to get five neighborhood structures in our proposed algorithm.

The remaining sections of this paper are organized as follows. In Section 2, a brief description of IPMS problem is mentioned. In Section 3, the steps of proposed algorithm are described in detail and the neighborhood structures of this proposed algorithm are explained. In Section 4, computational results are discussed. Conclusion is made in Section 5.

2. Problem Description

The identical parallel machine scheduling (IPMS) problem can be described as follows.

A set n of an independent jobs $J = \{J_1, J_2, \ldots, J_n\}$ to be processed on m identical parallel machines $M = \{M_1, M_2, \ldots, M_m\}$ with the processing time of job i on any identical machine is given by p_i.

A job can only be processed on one machine simultaneously and a machine cannot process more than one job at a time. Priority and precedence constraints are not allowed. There is no job cancellation and a job completes its processing on a machine without interruption.

The objective is to minimize the total completion time "the makespan" of scheduling jobs on the machines.

This scheduling problem can be described by a triple $\alpha \mid \beta \mid \gamma$ as follows:

$$Pm \parallel C_{\max}, \tag{1}$$

where P indicates parallel machine environment, m indicates number of machines, β indicates no constraints in this problem, and $C_{\max}$ indicates that the objective is to minimize the makespan.

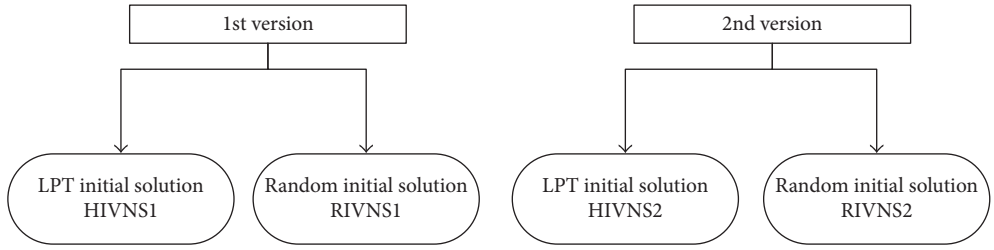

Figure 1: The two versions of the proposed algorithms.

This problem is interesting because minimizing the makespan has the effect of balancing the load over the various machines, which is an important goal in practice.

3. Development of the Proposed (IVNS) Algorithm

3.1. Basic VNS. Variable neighborhood search (VNS) is a metaheuristic proposed by Mladenović and Hansen [26] to enhance the solution quality by systematic neighborhoods changes. The main VNS algorithm steps can be summarized as follows: Initialization: choose the neighborhood structures set (NK′), $k = 1, 2, \ldots, k'_{\max}$, obtain an initial solution, and select a stopping condition. Repeat the next steps until the stopping condition is satisfied:

(1) Set $k \leftarrow 1$.

(2) Repeat the following steps until $k = k_{\max}$:

(a) Shaking: generate a point x' at random from the kth neighborhood of x ($x' \in N_k(x)$).

(b) Local search: apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum.

(c) Move or not: if the local optimum x'' is better than the incumbent x, move there ($x \leftarrow x''$), and continue the search with $N1$ ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$, improved variable neighborhood search (IVNS) algorithm.

As we mentioned earlier, the proposed algorithm is an addition of the proposed algorithm of Jing and Jun-qing [23].

The proposed algorithms have two versions and two types for each version as shown in Figure 1. In the first version, a new neighborhood structure was added to the four neighborhood structures which are proposed by Jing and Jun-qing [23] while in the second version the order of these neighborhood structures was changed. Both versions use LPT [20] and random initial solutions and are referred to as "HIVNS" and "RIVNS," respectively. All these versions of the proposed algorithm use the same five neighborhood structures. These neighborhood structures will be discussed in the following section.

3.2. Neighborhood Structures. Determining the neighborhood structures is critical in the VNS algorithm. To enhance the local searching abilities, five different kinds of neighborhoods are utilized to find better solutions on a given schedule in the proposed algorithm, which are designed based on such an idea that a given solution can be improved by moving or swapping jobs between the problem machines (the machines with their finished time equal to the makespan of the solution) and any other nonproblem machines (the machines with their finished time less than the makespan of the solution).

The five neighborhood structures are illustrated as follows:

(1) Move: move a job J_i from M_p to M_{np} if condition ($C_{Mp} - C_{Mnp} > pi$) is satisfied.

(2) Exchange 1: exchange a job J_i selected from Mp with another job J_j selected from M_{np} if ($p_i - pj > 0$) and ($C_{Mp} - C_{Mnp} > p_i - p_j$).

(3) Exchange 2: exchange two jobs, J_i and J_j, selected from M_p with one job J_k selected from M_{np} if ($p_i + p_j - p_k > 0$) and ($C_{Mp} - C_{Mnp} > p_i + p_j - p_k$).

(4) Exchange 3: exchange two jobs, J_i and J_j, selected from M_p with two jobs, J_k and J_t, selected from M_{np} if ($p_i + p_j - (p_k + p_t) > 0$) and ($C_{Mp} - C_{Mnp} > p_i + p_j - (p_k + p_t)$).

(5) Exchange 4: exchange one job J_i, selected from M_p with two jobs, J_i and J_k, selected from M_{np} if ($p_i - (p_j + p_k) > 0$) and ($C_{Mp} - C_{Mnp} > p_i - (p_j + p_k)$).

The orders assigned to the types proposed of the algorithm are as follows:

(1) The order of "HIVNS1" and "RIVNS1" is "move, exchange 1, exchange 2, exchange 3, and exchange 4."

(2) The order of "HIVNS2" and "RIVNS2" is "exchange 3, exchange 1, move, exchange 2, and exchange 4". Improved VNS (IVNS) flow chart is shown as Figure 2.

3.3. Steps of IVNS. The steps of IVNS for "HIVNS1" and "RIVNS1" are shown as follows.

Step 1. Generate initial schedule X (generated randomly or obtain from the LPT rule), MaxIterNum = 100, $i = 0$, and $k_{\max} = 5$.

Step 2. Compute lower bound: $\mathrm{LB}(C_{\max}) = \max\{[(1/m)\sum_{i=1}^{n} pi]; \max_i\{pi\}\}$.

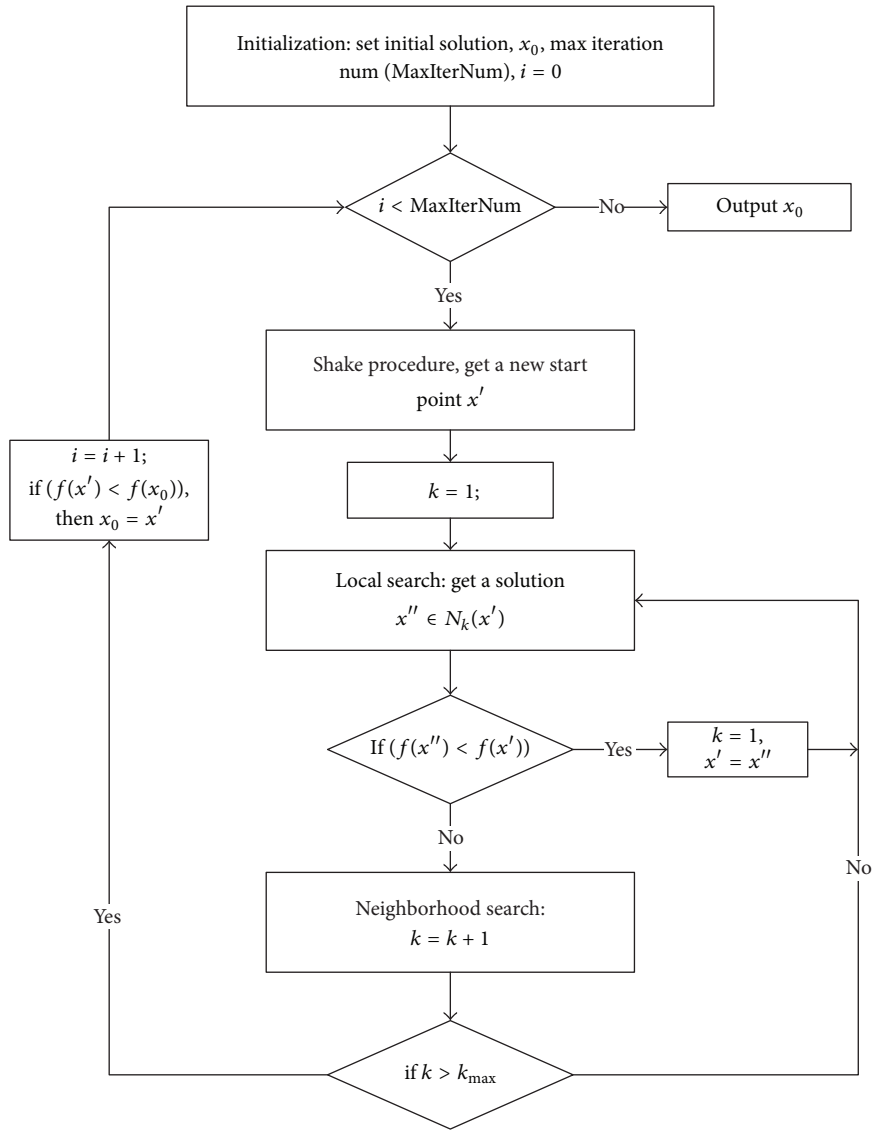

Figure 2: Flow chart of the basic VNS algorithm.

Step 3. Repeat until $f(X)$ (makespan of X) is equal to LB or $i >$ MaxIterNum.

Step 3.1. For schedule X, distinguish the problem machine set (S_{pm}) and the nonproblem machine set (S_{npm}).

Step 3.2. For each machine M_p in S_{pm} do.

Step 3.2.1. For each machine M_{np} in S_{npm} do.

Step 3.2.1.1. Set $K = 1$, finish = false.

Step 3.2.1.2

Repeat

Switch (K) {

$K = 1$: $X' = \text{Move}(M_p, M_{np}, X)$; break;

$K = 2$: $X' = \text{Exchange } 1(M_p, M_{np}, X)$; break;

$K = 3$: $X' = \text{Exchange } 2(M_p, M_{np}, X)$; beak;

$K = 4$: $X' = \text{Exchange } 3(M_p, M_{np}, X)$; break;

$K = 5$: $X' = \text{Exchange } 4(M_p, M_{np}, X)$;

}

if $(f(X') < f(X))$ then set $X = X'$, finish = true, go to Step 3; otherwise, set $k = k + 1$, until $k = k_{\max}$.

Step 3.3. If finish = false then $i = i + 1$.

Step 4. Output the best solution X found so far.

The steps of IVNS for "HIVNS2" and "RIVNS2" are shown as follows.

Step 1. Generate initial schedule X (generated randomly or obtain from the LPT rule), MaxIterNum = 100, $i = 0$, and $k_{\max} = 5$.

Table 1: Number of machines and jobs.

Number of machines	2	5	10	20
Number of jobs	20, 50, 100, 200	20, 50, 100, 200	20, 50, 100, 200	50, 100, 200

Step 2. Compute lower bound: $\mathrm{LB}(C_{\max}) = \max\{[(1/m)\sum_{i=1}^{n} pi]; \max_i\{pi\}\}$.

Step 3. Repeat until $f(X)$ (makespan of X) is equal to LB or $i >$ MaxIterNum.

Step 3.1. For schedule X, distinguish the problem machine set (S_{pm}) and the nonproblem machine set (S_{npm}).

Step 3.2. For each machine M_p in S_{pm} do.

Step 3.2.1. For each machine M_{np} in S_{npm} do.

Step 3.2.1.1. Set $K = 1$, finish = false.

Step 3.2.1.2

Repeat

Switch (K) {

$K = 1$: $X' =$ Exchange $3(M_p, M_{np}, X)$; break;

$K = 2$: $X' =$ Exchange $1(M_p, M_{np}, X)$; break;

$K = 3$: $X' = \mathrm{Move}(M_p, M_{np}, X)$; break;

$K = 4$: $X' =$ Exchange $2(M_p, M_{np}, X)$; beak;

$K = 5$: $X' =$ Exchange $4(M_p, M_{np}, X)$;

}

if $(f(X') < f(X))$ then set $X = X'$, finish = true, go to Step 3; otherwise, set $k = k + 1$,

until $k = k_{\max}$.

Step 3.3. If finish = false then $i = i + 1$.

Step 4. Output the best solution X found so far.

4. Computational Results and Comparison

In this section, the results of two versions of the proposed algorithm were compared with LPT [1], SA [17], GA [14], MVNS [21], and IVNS [23] algorithms from the literature. The two versions of the improved variable neighborhood search algorithms "HIVNS1 and RIVNS1" and "HIVNS2 and RIVNS2" were coded in MATLAB R2012a and executed on i5 CPU 5 GHz with 6 GB of RAM. All of them were stopped after getting the lower bound or running for 100 iterations for RIVNS1 and RIVNS2. The number of machines and number of jobs are shown in Table 1.

The processing time of the jobs is the same as Jing and Jun-qing [23]. As a result, 15 instances were conducted and each instance was conducted with 10 generations of different processing times. The total is 150 instances. The

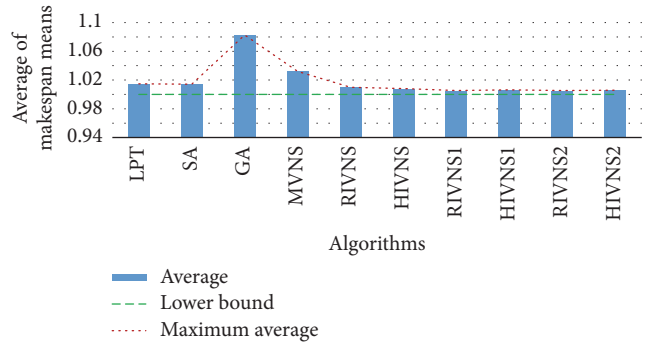

Figure 3: Histogram of averages of makespan means.

performance of the algorithms is measured with respect to the average makespan (mean) and average CPU time (Avg. time in second). The "mean" performance is a relative quality measure of solutions computed by C/LB, where C is the average makespan obtained for each instance with 10 generations given by the algorithm and LB is the lower bound of the instance, calculated in equation that mentioned in Section 3.3, Step 2. The "Avg. time" refers to the total time it takes for the algorithm to finish the solution. Table 2 presents the results of the previous algorithms from the literature while Table 3 presents the results from the proposed algorithms. By comparing the results of the average means of the makespan. It is obvious that the proposed algorithms outperform all the algorithms in Table 2 from the literature. It is worth noting that for each instance the proposed algorithms obtain no worse than algorithms in Table 2 except at 10 machines and 20 jobs instance in only HIVNS algorithm and that is due to the difficulty facing the proposed algorithms to get a lower bound when the difference between the number of machines and the number of jobs is relatively small. In addition, by comparing the two proposed algorithms, we can see that the versions have the same average means of the makespan in case of random initial solution while as the 2nd version outperforms the 1st version in case of LPT initial solution in both average means of makespan and average CPU time.

Figure 3 shows the averages of makespan mean the maximum averages for each algorithm and the lower bound. It can be observed that the two proposed versions algorithms have makespan means averages, which closed to the lower bound, especially in RIVNS1 and RIVNS2 that have the same average (1.0054).

Figure 4 shows the averages of Avg. time means for all algorithms. We can see that RIVNS1 and RIVNS2 have the smallest Avg. time, which are 0.008 and 0.007, respectively. Moreover, it can be observed that the Avg. time of HIVNS1 and HIVNS2 is much higher than HIVNS, because in this paper, Matlab is used to construct the code of HIVNS1 and

Table 2: The makespan results before the changing order of the neighborhood structures [23].

Instance number	m	n	LPT	SA		GA		MVNS		RIVNS		HIVNS	
				Mean	Avg. time	Mean	Avg. time	Mean	Avg. time	Mean	Avg. time	Mean	Avg. time
1	2	20	1.0033	1.0006	1.0149	1.0000	0.2215	1.0000	0.0106	1.0000	0.0009	1.0000	0.0006
2		50	1.0001	1.0000	1.0738	1.0000	0.5131	1.0000	0.0226	1.0000	0.0016	1.0000	0.0003
3		100	1.0000	1.0000	1.9382	1.0000	2.7948	1.0000	0.0445	1.0000	0.0031	1.0000	0.0005
4		200	1.0000	1.0000	0.9218	1.0000	2.3145	1.0000	0.0905	1.0000	0.0059	1.0000	0.0003
5	5	20	1.0315	1.0264	1.3614	1.0356	0.3412	1.0127	0.0124	1.0045	0.0083	1.0043	0.0073
6		50	1.0053	1.0045	2.5502	1.0312	0.6935	1.0050	0.0283	1.0012	0.0027	1.0009	0.0022
7		100	1.0005	1.0005	4.2271	1.0242	1.3862	1.0052	0.0534	1.0002	0.0042	1.0000	0.0015
8		200	1.0003	1.0003	14.4302	1.0168	2.6560	1.0084	0.1064	1.0001	0.0075	1.0000	0.0035
9	10	20	1.0794	1.0792	1.9328	1.1336	0.3575	1.0987	0.0131	1.0799	0.0402	1.0590	0.0384
10		50	1.0207	1.0207	4.1428	1.1169	1.6303	1.0315	0.0315	1.0078	0.0075	1.0036	0.0067
11		100	1.0110	1.0110	20.1301	1.1421	1.4547	1.0181	0.0647	1.0031	0.0061	1.0011	0.0043
12		200	1.0007	1.0007	1.0165	1.0611	3.3761	1.0127	0.1561	1.0004	0.0117	1.0002	0.0034
13	20	50	1.0510	1.0510	2.4803	1.3167	0.9728	1.1671	0.0441	1.0312	0.1389	1.0304	0.1392
14		100	1.0123	1.0123	4.2799	1.2270	1.8111	1.0857	0.0905	1.0087	0.0179	1.0046	0.0087
15		200	1.0063	1.0063	14.6684	1.1353	3.4492	1.0415	0.1840	1.0039	0.0163	1.0024	0.0113
	Average		**1.0148**	**1.0142**	**5.0779**	**1.0827**	**1.5982**	**1.0319**	**0.0635**	**1.0095**	**0.0182**	**1.0071**	**0.0152**

Table 3: The makespan results after the effect of the changing order of the neighborhood structures.

Instance number	m	n	RIVNS1		HIVNS1		RIVNS2		HIVNS2	
			Mean	Avg. time	Mean	Avg. time	Mean	Avg. time	Mean	Avg. time
1	2	20	1.0000	0.0005	1.0000	0.0004	1.0000	0.0008	1.0000	0.0005
2		50	1.0000	0.0002	1.0000	0.0002	1.0000	0.0002	1.0000	0.0002
3		100	1.0000	0.0004	1.0000	0.0005	1.0000	0.0004	1.0000	0.0083
4		200	1.0000	0.0002	1.0000	0.0004	1.0000	0.0002	1.0000	0.7030
5	5	20	1.0005	0.0014	1.0000	0.0025	1.0005	0.0014	1.0000	0.0016
6		50	1.0000	0.0005	1.0000	0.0028	1.0000	0.0008	1.0000	0.0175
7		100	1.0000	0.0235	1.0000	0.1104	1.0000	0.0258	1.0000	0.1705
8		200	1.0000	0.0002	1.0000	2.5874	1.0000	0.0002	1.0000	2.5057
9	10	20	1.0625	0.0016	1.0682	0.0040	1.0625	0.0015	1.0602	0.0040
10		50	1.0004	0.0103	1.0000	0.0156	1.0000	0.0064	1.0000	0.0580
11		100	1.0000	0.0052	1.0000	0.1122	1.0000	0.0054	1.0000	0.0713
12		200	1.0000	0.0238	1.0000	1.8051	1.0000	0.0207	1.0000	1.556
13	20	50	1.0175	0.0158	1.0256	0.0542	1.0175	0.0153	1.0263	0.0672
14		100	1.0003	0.0288	1.0000	0.2312	1.0003	0.0270	1.0000	0.0222
15		200	1.0000	0.0004	1.0000	1.0538	1.0000	0.0004	1.0000	0.7414
	Average		**1.0054**	**0.0075**	**1.0063**	**0.3987**	**1.0054**	**0.0071**	**1.0058**	**0.3951**

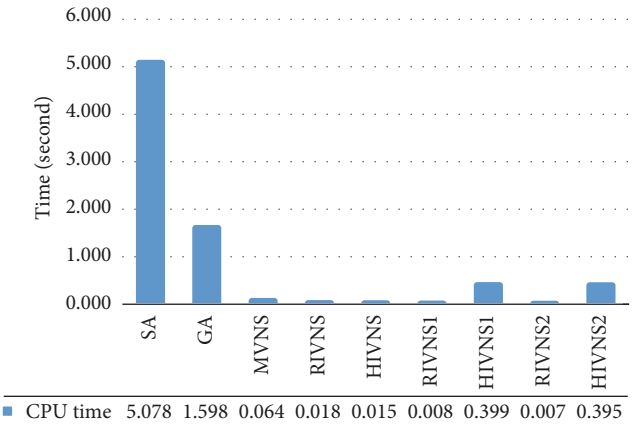

Figure 4: Histogram of averages of Avg. time means.

HIVNS2, and the authors of [23] used C++ to construct HIVNS. Thus, the benefits of Avg. time offered by C++ far outweigh the simplicity of Matlab. Avg. time is especially important when dealing with algorithms, since many calculations involve immense optimization with complex equations and algorithms or calculations with a large number of iterations. As the amount of data increases, the computation time (Avg. time) for Matlab code increases significantly; therefore, Matlab code takes more time for those calculations; for example, in Table 3 the cases of $m = 2, 5, 10, 20$, $n = 200$ are need to large number of iterations. C++ is the way to go for algorithms calculations because of its speed and versatility.

5. Conclusion

In this paper, two versions of improved variable neighborhood search (IVNS) algorithms are proposed for scheduling identical parallel machines IPM with the objective of studying the effect of adding a new neighborhood structure and changing the order of neighborhood structures on minimizing the makespan C_{max}. In the proposed algorithms, a machine based encoding method and five neighborhood structures are used to enhance the quality of the final solution. Computational results showed that the proposed algorithms outperform all the algorithms in the literature and obtain no worse than algorithms except when the number of machines and the number of jobs are relatively small which is due to the difficulty facing the proposed algorithms to get a lower bound in that case. In addition, we concluded that the second version outperforms the first version in case of LPT initial solution and therefore changing the order of neighborhood structures has an effect on minimizing the makespan. Further research is to implement the proposed algorithms in scheduling of unrelated parallel machines.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The authors extend their appreciation to the Deanship of Scientific Research at King Saud University for funding this work through Research Group no. RG-1439-009.

References

[1] M. R. Gary and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, WH Freeman and Company, New York, NY, USA, 1979.

[2] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM Journal on Applied Mathematics*, vol. 17, pp. 416–429, 1969.

[3] S. L. van de Velde, "Duality-based algorithms for scheduling unrelated parallel machines," *ORSA Journal on Computing*, vol. 5, no. 2, pp. 192–205, 1993.

[4] E. Mokotoff, "An exact algorithm for the identical parallel machine scheduling problem," *European Journal of Operational Research*, vol. 152, no. 3, pp. 758–769, 2004.

[5] J. Coffman, M. R. Garey, and D. S. Johnson, "An application of bin-packing to multiprocessor scheduling," *SIAM Journal on Computing*, vol. 7, no. 1, pp. 1–17, 1978.

[6] J. H. Blackstone Jr. and D. T. Phillips, "An improved heuristic for minimizing makespan among m identical parallel processors," *Computers & Industrial Engineering*, vol. 5, no. 4, pp. 279–287, 1981.

[7] C.-Y. Lee and J. D. Massey, "Multiprocessor scheduling: combining LPT and MULTIFIT," *Discrete Applied Mathematics*, vol. 20, no. 3, pp. 233–242, 1988.

[8] M. Y. Yue, "On the exact upper bound for the multifit processor scheduling algorithm," *Annals of Operations Research*, vol. 24, no. 1–4, pp. 233–259, 1990.

[9] C.-Y. Lee and J. D. Massey, "Multiprocessor scheduling: An extension of the MULTIFIT algorithm," *Journal of Manufacturing Systems*, vol. 7, no. 1, pp. 25–32, 1988.

[10] J. C. Ho and J. S. Wong, "Makespan minimization for m parallel identical processors," *Naval Research Logistics (NRL)*, vol. 42, no. 6, pp. 935–948, 1995.

[11] J. Riera, D. Alcaide, and J. Sicilia, "Approximate algorithms for the $P \parallel C_{\max}$ problem," *TOP*, vol. 4, no. 2, pp. 345–359, 1996.

[12] R. Cheng and M. Gen, "Parallel machine scheduling problems using memetic algorithms," *Computers & Industrial Engineering*, vol. 33, no. 3-4, pp. 761–764, 1997.

[13] S. F. Ghomi and F. J. Ghazvini, "A pairwise interchange algorithm for parallel machine scheduling," *Production Planning & Control*, vol. 9, pp. 685–689, 1998.

[14] L. Min and W. Cheng, "A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines," *Artificial Intelligence in Engineering*, vol. 13, no. 4, pp. 399–403, 1999.

[15] J. N. D. Gupta and J. Ruiz-Torres, "A LISTFIT heuristic for minimizing makespan on identical parallel machines," *Production Planning and Control*, vol. 12, no. 1, pp. 28–36, 2001.

[16] A. M. Costa, P. A. Vargas, F. J. Von Zuben, and P. M. Franca, "Makespan minimization on parallel processors: An immune-based approach," in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02)*, pp. 920–925, IEEE, May 2002.

[17] W.-C. Lee, C.-C. Wu, and P. Chen, "A simulated annealing approach to makespan minimization on identical parallel machines," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 3-4, pp. 328–334, 2006.

[18] L. Tang and J. Luo, "A new ILS algorithm for parallel machine scheduling problems," *Journal of Intelligent Manufacturing*, vol. 17, no. 5, pp. 609–619, 2006.

[19] D. E. Akyol and G. M. Bayhan, "Minimizing makespan on identical parallel machines using neural networks," in *Proceedings of the International Conference on Neural Information Processing*, vol. 4234 of *Lecture Notes in Computer Science*, pp. 553–562, Springer, 2006.

[20] A. H. Kashan and B. Karimi, "A discrete particle swarm optimization algorithm for scheduling parallel machines," *Computers & Industrial Engineering*, vol. 56, no. 1, pp. 216–223, 2009.

[21] M. Sevkli and H. Uysal, "A modified variable neighborhood search for minimizing the makespan onidentical parallel machines," in *Proceedings of the 2009 International Conference on Computers and Industrial Engineering (CIE '09)*, pp. 108–111, IEEE, July 2009.

[22] J. Chen, Q.-K. Pan, and H. Li, "Harmony search algorithm with dynamic subpopulations for scheduling identical parallel machines," in *Proceedings of the 2010 6th International Conference on Natural Computation (ICNC '10)*, pp. 2369–2373, IEEE, August 2010.

[23] C. Jing and L. Jun-qing, "Efficient variable neighborhood search for identical parallel machines scheduling," in *Proceedings of the Control Conference (CCC '12)*, pp. 7228–7232, IEEE, 2012.

[24] M. Sevkli and A. Z. Sevkli, *A Stochastically Perturbed Particle Swarm Optimization for Identical Parallel Machine Scheduling Problems*, INTECH Open Access Publisher, 2012.

[25] D. Laha, "A simulated annealing heuristic for minimizing makespan in parallel machine scheduling," in *Proceedings of the International Conference on Swarm, Evolutionary, and Memetic Computing*, pp. 198–205, Springer, 2012.

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