

Article

Variable Neighborhood Search for Minimizing the Makespan in a Uniform Parallel Machine Scheduling

Khaled Bamatraf *  and Anis Gharbi 

Department of Industrial Engineering, College of Engineering, King Saud University, P.O. Box 800, Riyadh 11421, Saudi Arabia; a.gharbi@ksu.edu.sa

* Correspondence: kbamatraf@ksu.edu.sa

Abstract: This paper investigates a uniform parallel machine scheduling problem for makespan minimization. Due to the problem's NP-hardness, much effort from researchers has been directed toward proposing heuristic and metaheuristic algorithms that can find an optimal or a near-optimal solution in a reasonable amount of time. This work proposes two versions of a variable neighborhood search (VNS) algorithm with five neighborhood structures, differing in their initial solution generation strategy. The first uses the longest processing time (LPT) rule, while the second introduces a novel element by utilizing a randomized longest processing time (RLPT) rule. The neighborhood structures for both versions were modified from the literature to account for the variable processing times in uniform parallel machines. We evaluated the performance of both VNS versions using a numerical example, comparing them against a genetic algorithm and a tabu search from existing literature. Results showed that the proposed VNS algorithms were competitive and obtained the optimal solution with much less effort. Additionally, we assessed the performance of the VNS algorithms on randomly generated instances. For small-sized instances, we compared their performance against the optimal solution obtained from a mathematical formulation, and against lower bounds derived from the literature for larger instances. Computational results showed that the VNS version with the randomized LPT rule (RVNS) outperformed that with the LPT rule as the initial solution (LVNS). Moreover, RVNS found the optimal solution in 90.19% of the small instances and yielded an average relative gap of about 0.15% for all cases.

Keywords: longest processing time; scheduling; uniform parallel machines; variable neighborhood search



Citation: Bamatraf, K.; Gharbi, A. Variable Neighborhood Search for Minimizing the Makespan in a Uniform Parallel Machine Scheduling. *Systems* **2024**, *12*, 221. <https://doi.org/10.3390/systems12060221>

Academic Editors: Mohammed M. Mabkhot, Pedro Ferreira and Dorota Stadnicka

Received: 16 May 2024
Revised: 11 June 2024
Accepted: 17 June 2024
Published: 20 June 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In today's competitive manufacturing landscape, increasing efficiency and productivity and minimizing production costs are essential for businesses to remain competitive and sustainable. As a result, optimizing resource allocation and minimizing production completion times through scheduling is becoming increasingly important. Furthermore, efficient scheduling can improve sustainability by reducing energy consumption in the manufacturing industry. This is important since these industries account for a significant proportion of global energy consumption [1]. In China, for instance, approximately 70% of total emissions are related to industrial sectors [2]. From this standpoint, scheduling is becoming essential to improve productivity and energy efficiency.

In this paper, we study a uniform parallel machine scheduling problem for makespan minimization, denoted by $Q_m \mid |C_{max}$, according to Graham et al.'s notations [3]. In the uniform parallel machine, a set of n independent jobs, $J = \{1, 2, \dots, j, \dots, n\}$, with processing times, p_j , such that $p_1 \geq p_2 \geq \dots \geq p_n$, are to be scheduled on m machines, $M = \{1, 2, \dots, i, \dots, m\}$, which may differ in job processing speeds, v_i , such that $v_1 \leq v_2 \leq \dots \leq v_i \leq \dots \leq v_m$, with machine m being the fastest machine. It is assumed that p_j is the processing time of job j on the slowest machine. Therefore, the processing time of each

job on each machine is $p_{ij} = p_j \times v_1/v_i$. The uniform parallel machine transforms into an identical parallel machine scheduling problem when the speeds of the machines are equal, i.e., $v_1 = v_2 = \dots = v_m$. Therefore, the identical parallel machine scheduling problem is a special case of the studied scheduling problem. Each job is assumed to be available at the beginning of scheduling and no preemption is allowed. The problem is NP-hard in the strong sense [4]. Hence, heuristic and metaheuristic algorithms are developed by the researchers to obtain good-quality solutions in shorter computation times. This problem is of practical importance since parallel operations are frequent in manufacturing and services industries. For instance, consider a manufacturing facility that has m machines, where some are newer and more efficient than others. The newer machines process the jobs faster than the older machines [5]. This configuration represents a uniform parallel machine environment. The goal is to balance the load among all machines by minimizing the maximum completion time of the last job in a schedule, which is known as the makespan.

The rest of this paper is organized as follows: Section 2 reviews the existing literature on the uniform parallel machine scheduling problem. Section 3 describes the methodology used to solve the problem under consideration. Section 4 presents the computational results. Finally, Section 5 concludes the paper and provides directions for future work.

2. Literature Review

This section will review the research on the uniform parallel machine scheduling problem and examine various scheduling algorithms and approaches to address this combinatorial problem. Among the various approaches, the longest processing time (*LPT*) rule, introduced by Graham [6], stands out as one of the earliest and simplest heuristics, specifically designed for a special case of uniform machines with equal processing speeds. For identical parallel machine scheduling, the *LPT* rule assigns jobs in a non-increasing order of processing time to the least occupied machine. In contrast, for uniform parallel machine scheduling, the adapted *LPT* rule prioritizes jobs in a non-increasing order of processing time to a machine that completes processing the job sooner [7].

The *LPT* worst-case performance ratio for identical parallel machine scheduling is $4/3 - 1/3m$ [6]. This ratio was improved by Della Croce and Scatamacchia [8] to $(4/3 - 1/(3(m-1)))$ for $m \geq 3$. For the uniform parallel machine, Gonzalez et al. [7] demonstrated that the adapted *LPT* rule obtained a makespan at most twice the optimal solution. Friesen [9] showed that the worst-case bound for the adapted *LPT* rule is within (1.52, 1.67). Mireault et al. [10] derived an explicit function of the speed of the fastest machine as the worst-case ratio for *LPT* on two uniform parallel machines. Koulamas and Kyparisis [11] studied a modified *LPT* rule for two uniform parallel machines that optimally scheduled the three longest jobs and then applied the *LPT* rule to the remaining jobs. This modified *LPT* rule obtained a worst-case ratio of 1.2247. Massabò et al. [12] developed a worst-case performance ratio for the *LPT* heuristic in scheduling problem with two uniform parallel machines. The performance ratio is based on the index of the latest job inserted in the machine with minimum completion time. For the *LPT* rule in uniform parallel machine scheduling, Mitsunobu et al. [13] determined tight approximation ratios of 1.38, 1.43, and 1.46 for problems with 3, 4, and 5 processors, respectively. Friesen and Langston [14] and Chen [15] provided worst-case bounds for a *MULTIFIT* algorithm for uniform parallel machine scheduling.

In addition to the *LPT* heuristic, other heuristics have been proposed for uniform parallel machine scheduling with a makespan objective. Li and Zhang [16] studied this problem under release time constraints and proposed a heuristic based on the longest processing time and largest release date (*LPT-LRD*) rule. To further improve the solution quality, the authors developed a variable neighborhood search algorithm with two neighborhood structures. Sivasankaran et al. [17] proposed a heuristic and evaluated its performance by solving small instances and comparing the results with a mathematical model. Li and Yang [18] developed heuristic algorithms for scheduling uniform parallel machines with head and tails to minimize the makespan. De Giovanni et al. [19] developed

a modified longest processing time rule algorithm and an iterated local search algorithm for the uniform parallel machine scheduling with a makespan objective. Song et al. [20] studied the uniform parallel machine scheduling for a green manufacturing system to minimize the makespan and maximize the maximum lateness while incorporating an upper bound constraint on the total energy cost. The authors proposed a polynomial-time algorithm for a preemptive uniform parallel machine scheduling and a two-approximation algorithm for a non-preemptive uniform parallel machine scheduling.

Metaheuristics also play a crucial role in obtaining good solutions for the considered problem. Balin [21] adapted a genetic algorithm (GA) for non-identical parallel machine scheduling with a makespan objective. The author tested its performance on a numerical example with only four uniform machines and nine jobs. Noman et al. [22] proposed a tabu search (TS) algorithm for the same problem. They compared its performance with the GA developed by Balin [21] using the same numerical example.

Several researchers have investigated the effectiveness of the VNS algorithm for parallel machine scheduling. Li and Cheng [23] proposed a variable neighborhood search (VNS) algorithm to minimize the makespan of a uniform parallel machine with release dates. The proposed VNS algorithm makes use of two distinct initial solutions based on the problem's size. The algorithm also employs two neighborhood structures to obtain improved solutions. Sevkli and Uysa [24] proposed a modified variable neighborhood search algorithm with two neighborhood structures for makespan minimization in identical parallel machine scheduling. The authors compared the proposed algorithm with the solutions of both the genetic algorithm and the LPT rule. Jing and Jun-qing [25] studied the makespan minimization on identical parallel machine scheduling problem and proposed a VNS algorithm with two distinct initial solutions and proposed four neighborhood structures that are applied between a machine with a completion time equal to the makespan and another machine with a completion time less than the makespan. The work of Alharkan et al. [26] added a fifth neighborhood structure to the work of Jing and Jun-qing [25] and studied the order effect of these neighborhood structures in minimizing the makespan. Recently, Gharbi and Bamatraf [27] applied the VNS algorithm with an LPT-based initial solution as an upper bound for an improved arc flow formulation for the identical parallel machine scheduling with a makespan objective. Cheng et al. [28] studied the identical parallel machine scheduling problem with step-deteriorating jobs and proposed a variable neighborhood search algorithm that utilized five neighborhood structures to minimize the total completion time. The works of Senthilkumar and Narayanan [29,30] proposed four different genetic algorithms and three variations of simulated annealing algorithms to minimize the makespan in the uniform parallel machine scheduling problem. Kaabi [31] investigated scheduling on uniform parallel machines with machine unavailability, aiming to minimize the makespan. The author proposed a quadratic model for optimal solutions to small- and medium-sized problems and a novel two-phase algorithm for large-sized problems. To tackle the problem of scheduling on the uniform parallel machine with machine eligibility, job splitting, sequence-dependent setup time, and multiple servers, Kim and Lee [32] proposed a mathematical formulation and four lower bounds. Additionally, they further developed a heuristic algorithm suited for handling large-sized problems.

Due to the NP-hardness of the problem, exact algorithms can still be employed to find optimal solutions for small instances. However, for large instances, metaheuristics are crucial for finding high-quality solutions in a reasonable time. Horowitz and Sahni [33] proposed dynamic programming for $Q_m \parallel C_{max}$. De and Morton [34] presented a mathematical formulation and proposed a branch and bound algorithm for $Q_m \parallel C_{max}$ capable of solving small-sized instances, with up to 5 jobs with 8 machines or 10 jobs with 4 machines. Liao and Lin [35] proposed an optimal algorithm for two uniform parallel machines with a makespan minimization objective. The algorithm was evaluated over a set of jobs, ranging from 10 jobs up to 1000. Later, Lin and Liao [36] extended the previous work and proposed an optimal algorithm for the uniform parallel machine scheduling problem with a makespan objective. The optimal algorithm incorporated a procedure to determine all

the quasi-optimal solutions, a theorem to provide an improved lower bound, an algorithm to determine whether a quasi-optimal solution can be achieved, and another theorem to accelerate the search speed. The optimal algorithm was implemented on randomly generated instances with 3 types of machines, namely, 3, 4, and 5 machines, and a set of jobs ranging from 10 up to 1000 jobs. Popenko et al. [37] proposed a sufficient condition for schedule optimality on uniform parallel machines according to specific criteria. Their work focused on minimizing the makespan among these criteria. Berndt et al. [38] proposed improved support size bounds for integer linear programs (ILPs), leading to a faster approximation algorithm for makespan minimization on uniform parallel machine scheduling. Mallek and Boudhar [39] addressed scheduling on uniform parallel machines with conflict graphs. The authors proposed a mixed-integer linear program (MILP) formulation, along with lower and upper bounds, to minimize the makespan. The work of Soper and Strusevich [40] analyzed schedules with at most one preemption compared to the optimal schedule that allows preemption for minimizing the makespan on three uniform machines. The authors derived tight bounds based on machine speeds. For more literature on parallel machines and uniform parallel machine scheduling, readers are referred to the works in [41,42].

This paper contributes to uniform parallel machine scheduling by proposing two versions of the VNS algorithm to minimize the makespan: the VNS algorithm with the LPT schedule as an initial solution (LVNS) and the VNS algorithm with a random LPT schedule as an initial solution (RVNS). LVNS is inspired by existing VNS applications on identical parallel machines, but its neighborhood structures are modified to account for the variable processing times on uniform parallel machines. The second version, RVNS, introduces a novel element by utilizing a randomized longest processing time (RLPT) rule for initial solution generation alongside the standard LPT rule used in LVNS. Moreover, a comparison of the proposed VNS algorithms with algorithms from the literature on a numerical example shows that the optimal solution can be obtained with much less effort with our proposed methods. Due to the problem's complexity, the effectiveness of each algorithm is evaluated against the optimal solution obtained from a mathematical formulation adapted from the literature for instances with up to 20 jobs. The remaining instances are evaluated against a lower bound from the literature. Furthermore, the impact of the upper bound obtained by LVNS on reducing the solution time in a mathematical formulation is evaluated. This is achieved by comparing the CPU time required with LVNS's upper bound to the CPU time without any specified upper bound in the mathematical formulation.

In summary, the uniform parallel machine scheduling problem is pervasive in industries, including manufacturing and logistics. The proposed VNS algorithms, LVNS and RVNS, offer a practical solution that balances computational efficiency and solution quality. This balance allows them to find near-optimal solutions in a reasonable amount of time, potentially leading to reduced production costs or improved resource utilization in real-world scheduling practices.

3. Research Methodology

The methodology applied in this paper for solving the uniform parallel machine scheduling problem with the makespan objective involved first calculating a lower bound. Three lower bounds from the literature were calculated. The maximum of these values was considered the best lower bound. Furthermore, an exact mathematical formulation from the literature was employed. Then, a description of the proposed VNS algorithms was presented. We evaluated the performance of the VNS algorithms by comparing them against genetic algorithms (GA) and tabu search (TS) using a benchmark example. For small-sized instances with up to 20 jobs, we compared the VNS algorithms against the optimal solution obtained from a mathematical formulation. For larger instances where the number of jobs exceeded 20, the best lower bound was used for evaluation. This assessment enabled a comprehensive assessment of the VNS algorithms' effectiveness across different problem scales. The following subsections illustrate the details for each of the abovementioned steps.

3.1. Lower Bounds

Assuming that the first machine was the slowest, the following are the lower bounds used in this paper:

$$LB_1 = \frac{\sum_{j=1}^n p_{1j}}{\sum_{i=1}^m v_i} \quad (1)$$

$$LB_2 = p_{m1} \quad (2)$$

The lower bound in Equation (1) was simply calculated as the total processing times of jobs on the slowest machine divided by the total speeds of the machines. The speed of each machine was determined by dividing the processing time of a job on the slowest machine by the processing time of that job on the machine of interest. For simplicity, the first job was selected to calculate the speed for each machine, as follows:

$$v_i = \frac{p_{11}}{p_{i1}} \quad \forall i \in \{1, \dots, m\} \quad (3)$$

Equation (2) provides a lower bound on the makespan, which is equal to the processing time of a job with the largest processing time scheduled when it is scheduled on the fastest machine.

In addition to these lower bounds, Lin and Liao [36] proposed an improvement of the lower bound in Equation (2), as follows: consider $P = \sum_{j=1}^n p_{1j}$ is the total processing times of jobs on the slowest machine, $V = \sum_{i=1}^m v_i$ is the total machine speeds, C_{LPT} is the upper bound obtained by the longest processing time rule, and $w_i = \lfloor v_i \times LB_1 \rfloor$ is the largest workload on each machine i that makes the completion time of the last job no more than LB_1 . Therefore, a bigger assigned workload to each machine will contribute to various completion times. $W = \sum_{i=1}^m w_i$ is the total workload assigned in all machines. When these completion times do not exceed the upper bound, C_{LPT} , it is considered a quasi-optimal solution. The number of quasi-optimal solutions for each machine is calculated as:

$$q_i = \lfloor v_i \times C_{LPT} \rfloor - w_i \quad \forall i \in \{1, \dots, m\} \quad (4)$$

The possible quasi-optimal solutions for each machine are expressed as:

$$C_{i,k} = \frac{(w_i + k)}{v_i} \quad \forall i \in \{1, \dots, m\}; \quad \forall k \in \{1, \dots, q_i\} \quad (5)$$

We sorted all quasi-optimal solutions in non-increasing order and denoted these solutions as $C_{[1]} \leq C_{[2]} \leq \dots \leq C_{[mq_m]}$. $C_{[0]}$ is the lower bound, LB_1 , obtained from Equation (1). The improved lower bound is the quasi-optimal solution obtained at position $C_{[P-W]}$. Therefore, the best lower bound for $Q_m \mid \mid C_{max}$ among these calculated lower bounds is the maximum value between them. It is expressed as:

$$LB = \max(LB_1, LB_2, C_{[P-W]}) \quad (6)$$

3.2. Mathematical Formulation

A mathematical formulation for $Q_m \mid \mid C_{max}$, as presented by De and Morton [34], will be presented in this section. Due to the NP-hard nature of the problem, this mathematical formulation can solve instances with a smaller number of machines and jobs to optimality in a reasonable time. As the size of the problem increases, the time grows exponentially. A key benefit of presenting the mathematical formulation is to allow for efficient evaluation of the proposed VNS algorithms' performance across a range of problem sizes. The notations of the mathematical formulation are presented in Table 1.

Table 1. Notations of mathematical formulation.

Variable	Definition
Indices	
i	Index of a machine.
j	Index of a job.
Sets	
M	Set of machines
J	Set of jobs
Parameters	
p_{ij}	Processing time of job j on machine i . $\forall i \in M; \forall j \in J$.
LB	Lower bound
UB	Upper bound
Decision Variables	
x_{ij}	1 if job j is processed on machine i , 0 otherwise. $\forall i \in M; \forall j \in J$.
C_{max}	Makespan of a schedule

Objective function:

$$\min C_{max} \quad (7)$$

Constraints:

$$C_{max} \geq \sum_{j \in J} p_{ij} x_{ij} \quad \forall i \in M \quad (8)$$

$$\sum_{i \in M} x_{ij} = 1 \quad \forall j \in J \quad (9)$$

$$x_{ij} \in \{0, 1\} \quad \forall i \in M; \forall j \in J \quad (10)$$

The objective function in Equation (7) minimizes the makespan. Equation (8) shows that the makespan is the maximum completion time among all machines. Equation (9) shows that a job is assigned to only one machine. Binary decision variables are shown in Equation (10).

To formulate valid bounds of the makespan, constraint (11) was added to the mathematical formulation presented by De and Morton [34]:

$$LB \leq C_{max} \leq UB \quad (11)$$

3.3. Variable Neighborhood Search Algorithms

The variable neighborhood search (VNS) algorithm, developed by Mladenović and Hansen in 1997 [43], is a metaheuristic approach that aims to escape from a local optimal solution by performing different neighborhood structures, " $N_k(S)$ ", $k \in \{1, \dots, k_{max}\}$, to an initial solution (S). Starting from the first neighborhood structure, N_1 , if an improvement to the current solution is found within N_1 , The algorithm updates the solution and restart exploring neighborhoods from N_1 . If no improvement is found in N_1 , the algorithm moves on to the next neighborhood structure, N_2 . This process continues until all neighborhood structures are explored without finding an improvement. Upon reaching this point, the algorithm terminates. The selection of the initial solution method and neighborhood structures significantly impacts the quality of solutions for the VNS algorithm. Therefore, when the initial solution is generated randomly or using a heuristic with some randomization, it is recommended to run the VNS algorithm multiple times for each initial solution. The best solution among all iterations is then selected as the final result.

The variable neighborhood search algorithm is a single-based solution that focuses on improving a single current solution iteratively by exploring its neighborhoods. One advantage of the VNS algorithm is its ease of implementation and versatility, allowing it to be applied to a wide range of complex optimization problems. Another advantage lies in the use of systematic neighborhood structures. This systematic approach can lead to

fast convergence to good solutions and potentially a shorter computational time. Unlike other metaheuristics, such as the genetic algorithm and simulated annealing, which require careful parameter tuning and can be slower than the VNS algorithm, the GA has to evaluate each solution in the population. Moreover, simulated annealing may have slower convergence compared to the VNS algorithm. Nevertheless, VNS has some disadvantages. The main disadvantage occurs when the neighborhoods fail to improve the current solution and the algorithm gets stuck in a local optimum [28].

The flowchart of the proposed VNS algorithms is shown in Figure 1.

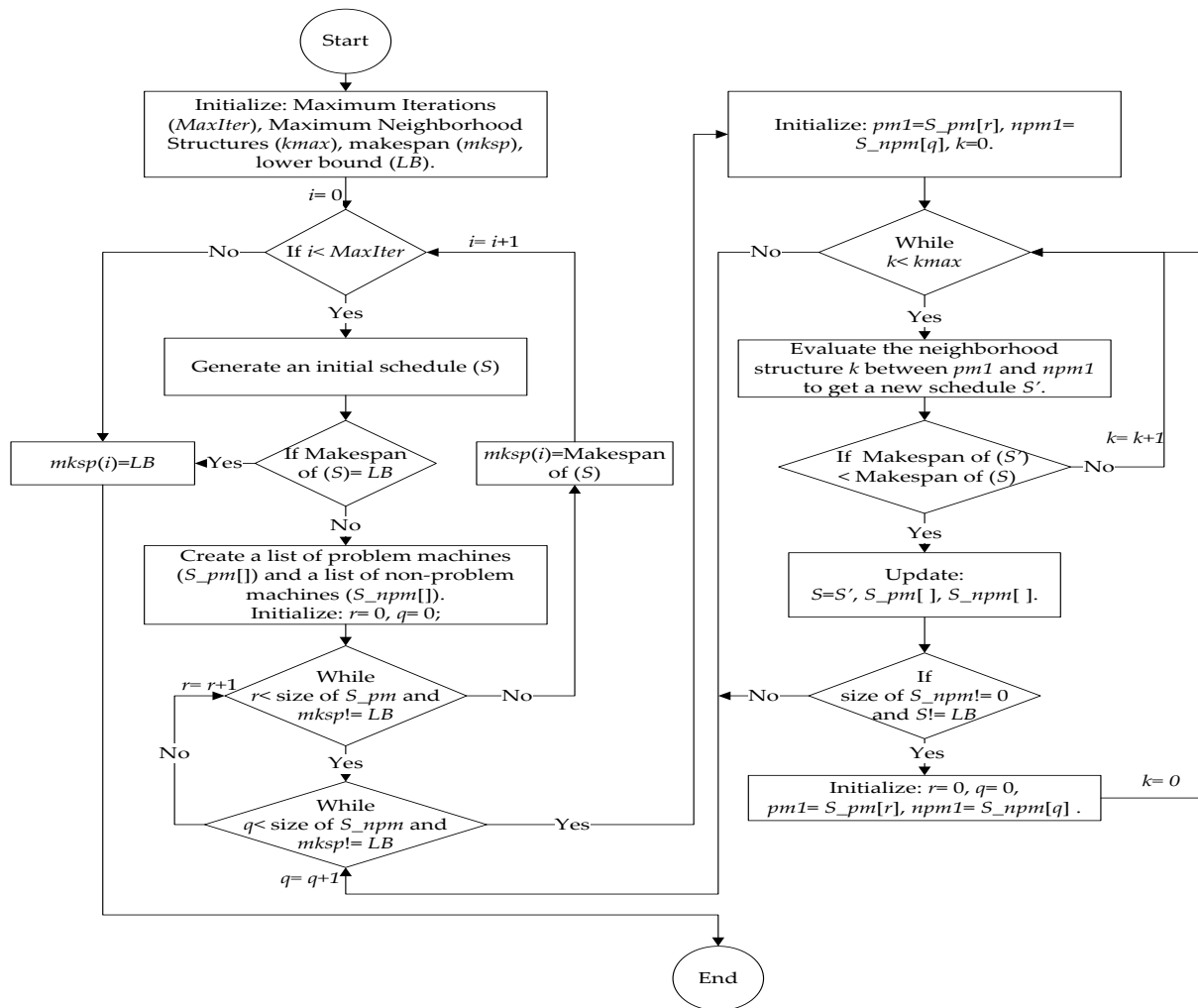


Figure 1. Flowchart of the proposed VNS algorithms.

3.3.1. Initial Solution

This paper introduced two versions of the VNS algorithm. The first version utilizes the longest processing time (LPT) rule to generate an initial solution. The second version adds randomization to the LPT rule. The randomized LPT heuristic will be denoted as RLPT. Both versions of the VNS algorithm applied five neighborhood structures, which will be explained in the following subsection.

The LPT rule prioritizes jobs in a non-increasing order of processing time and then assigns each job to the first available machine [7]. The proposed RLPT rule works by iteratively assigning jobs to machines. At each iteration, it randomly selects one of the two jobs with the largest processing time and assigns that job to the machine with the minimum available time. The process continues until all jobs have been assigned to machines.

3.3.2. Neighborhood Structures

Neighborhood structures attempt to improve the current solution by relocating jobs to machines. In the uniform parallel machine problem with the makespan objective, each neighborhood structure focuses on reducing the completion time of a machine that determines the overall makespan. This machine is referred to as a “problem machine (pm)”. The other machines with completion times less than the makespan are called “non-problem machines (npm)”.

Five neighborhood structures (k_{max}) are proposed in this paper. Each neighborhood structure (k) is detailed as follows:

1. Move (j): a job j is moved from pm to npm if the processing time of job j on the new non-problem machine, npm ($p_{npm,j}$), is less than the difference in completion times between (pm) and (npm), i.e., ($C_{pm} - C_{npm}$).
2. Exchange ($j - k$): a job j from pm is exchanged with a job k from npm if the resulting completion time for each machine does not exceed the makespan of pm “ C_{pm} ”. That is, ($C_{pm} - p_{pm,j} + p_{pm,k}$) < C_{pm} and ($C_{npm} - p_{npm,k} + p_{npm,j}$) < C_{pm} .
3. Exchange ($j - k, l$): two jobs, j and k , from pm are exchanged with one job, l , from npm if ($C_{pm} - p_{pm,j} - p_{pm,k} + p_{pm,l}$) < C_{pm} and ($C_{npm} - p_{npm,l} + p_{npm,j} + p_{npm,k}$) < C_{pm} .
4. Exchange ($j, k - l$): one job, j , from pm is exchanged with two jobs, k and l , from npm if ($C_{pm} - p_{pm,j} + p_{pm,k} + p_{pm,l}$) < C_{pm} and ($C_{npm} - p_{npm,k} - p_{npm,l} + p_{npm,j}$) < C_{pm} .
5. Exchange ($j - k, l - t$): two jobs, j and k , from pm are exchanged with two jobs, l and t , from npm if ($C_{pm} - p_{pm,j} - p_{pm,k} + p_{pm,l} + p_{pm,t}$) < C_{pm} and ($C_{npm} - p_{npm,l} - p_{npm,t} + p_{npm,j} + p_{npm,k}$) < C_{pm} .

3.3.3. Example

Consider three uniform parallel machines with six jobs. The processing times for each job on each machine and the speed of each machine are presented in Table 2.

Table 2. Processing times of jobs for each machine.

Machine	Speed	Job					
		1	2	3	4	5	6
1	1.0000	3822	3458	2912	2730	2002	2002
2	1.3382	2856	2584	2176	2040	1496	1496
3	1.8200	2100	1900	1600	1500	1100	1100

The lower bounds were calculated as $LB_1 = P/V = 4070.511$ and $LB_2 = p_{3,1} = 2100$, and the upper bound as $C_{LPT} = 4700$. The Gantt chart for the LPT schedule is shown in Figure 2.

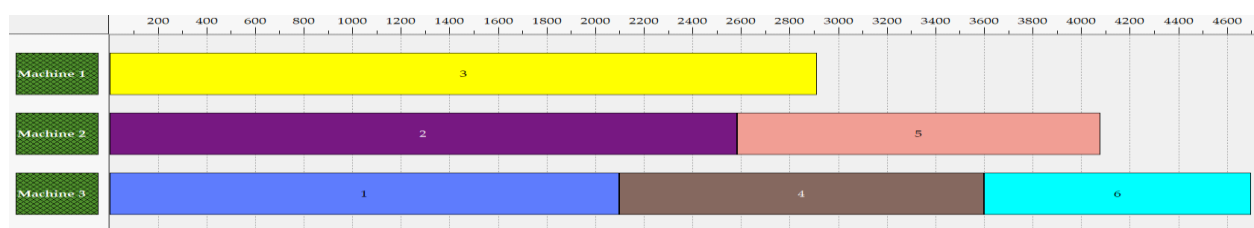


Figure 2. Gantt chart schedule for LPT.

An improvement in LB_1 could be obtained by first calculating the largest workload on each machine, which made the completion time of the last job no more than LB_1 : $w_1 = \lfloor s_1 \times LB_1 \rfloor = \lfloor 1 \times 4070.511 \rfloor = 4070$, $w_2 = \lfloor 1.3382 \times 4070.511 \rfloor = 5447$, and $w_3 = 7408$. Therefore, the total workload assigned was $W = 16,925$. The quasi-optimal solutions between LB_1 and C_{LPT} for each machine were $q_1 = \lfloor s_1 \times C_{LPT} \rfloor - w_1 = \lfloor 1 \times 4700 \rfloor - 4070 = 630$, $q_2 = \lfloor 1.3382 \times 4700 \rfloor - 5447 = 843$, and $q_3 = 1164$. The position

of the improved lower bound will be in the index of $\lfloor P - W \rfloor = \lfloor 16926 - 16925 \rfloor = 1$. Therefore, one quasi-optimal solution will be calculated for each machine. The quasi-optimal solutions for machines 1, 2, and 3 were $C_{1,1} = (w_i + k)/s_i = (4070 + 1)/1 = 4071$, $C_{2,1} = 4071.14$, and $C_{3,1} = 4070.879$, respectively. Then, we arranged all quasi-optimal solutions in increasing order starting with $C_{[0]} = LB_1$. Consequently, the improved lower bound was $C_{\lfloor P - W \rfloor} = C_{[1]} = 4070.879$. The best lower bound among the calculated lower bounds was $LB = \max(LB_1, LB_2, C_{\lfloor P - W \rfloor}) = 4070.879$.

Applying the LVNS algorithm (as detailed in Figure 1), we found that only the exchange $(j - k)$ neighborhood structure could be applied to the LPT schedule. This neighborhood structure allowed the swapping of jobs between machines. In this case, job 1 on problem machine 3 was exchanged with job 3 on non-problem machine 1. The resulting schedule is shown in Figure 3, with a makespan of 4200. This makespan obtained by the LVNS algorithm is equal to the optimal solution found by the mathematical formulation presented in Section 3.2.

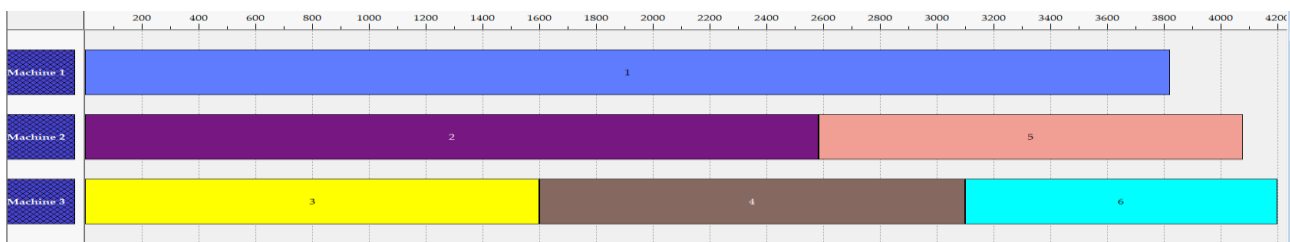


Figure 3. Gantt chart schedule for LVNS.

4. Computational Results and Discussion

The mathematical formulation and the proposed VNS algorithms were coded in C++ using the Cplex 12.10 optimization library. The experiments were conducted on an Intel® Core i7-4930 k CPU @3.40 GHz with 34.0 GB of RAM (Intel Corporate, Santa Clara, CA, USA). A set of instances was generated to evaluate the performance of the proposed algorithms. In the data generation process, four parameters were considered:

- (i) The number of machines (m).
- (ii) The ratio of the number of jobs (n) to the number of machines (m), n/m .
- (iii) The processing times of each job, j , on the fastest machine, m , $p_{m,j}$.
- (iv) The speed for each machine, i , to the fastest machine, S_i .

Four different sizes of machines were selected: $m \in \{3, 4, 5, \text{ and } 10\}$. The ratios of n/m considered were $n/m \in \{2, 3, 4, 5, 10, 20, \text{ and } 30\}$. The processing times of each job on the fastest machine were generated using a discrete uniform distribution: $U \sim [1, P_{max}]$. The three levels of the maximum processing time generated were $P_{max} \in \{25, 50, \text{ and } 100\}$. Finally, the speed factor for each machine to the fastest machine, S_i , was generated using a uniform distribution in the interval $U \sim [1, S_{max}]$, and we selected three levels for S_{max} : $S_{max} \in \{3, 5, \text{ and } 7\}$. The value of S_{max} was rounded to two decimal digits. Ten instances for each combination of $(m, n/m, P_{max}, \text{ and } S_{max})$ were generated. Consequently, the total number of instances generated was 2520.

The heuristics for generating the initial solution (LPT and RLPT) were compared first. Then, the VNS algorithms were evaluated against GA and TS from the literature. After that, the LVNS algorithm was compared with the RVNS algorithm on the generated instances. The relative percentage deviation (RPD) for the makespan of the VNS algorithms, C_1 , was evaluated against the optimal solution or the lower bound, C_2 , as shown in Equation (12):

$$RPD = \frac{C_1 - C_2}{C_2} \times 100 \quad (12)$$

The comparison of the VNS algorithms was carried out against the mathematical formulation for instances with up to 20 jobs. However, for the remaining instances, the VNS algorithms were evaluated against the lower bound.

The parameter settings of the VNS algorithms were the number of neighborhood structures and the number of iterations. The number of neighborhood structures was set to 5. These neighborhoods are described in Section 3.3. The number of iterations was set to one iteration for the LVNS algorithm and ten iterations for the RVNS algorithm. The LVNS algorithm requires only one iteration since each iteration always starts with the same initial schedule. However, each iteration of the RVNS algorithm can potentially generate a different schedule. To determine the number of iterations for the RVNS algorithm, we focused on balancing a trade-off between solution quality and computational time. Therefore, we solved all the generated instances with 100 replicates to investigate the best solution among these 100 replicates for each instance. We noticed that in almost 95% of the instances, the best solution was reached within 10 iterations. Therefore, we considered 10 iterations to obtain good solutions with minimal computation time. In the following subsections, we will present the computational results.

4.1. Comparison of the LPT Rule and RLPT Rule

In this section, we focus on the LPT and RLPT algorithms, investigating their performance in generating initial solutions for the VNS algorithm. Table 3 presents the relative percent deviation (RPD) for both algorithms.

Table 3. Relative percentage deviation comparison between LPT and RLPT.

<i>n/m</i>	<i>m</i>	<i>n</i>	S~[1,3]				S~[1,5]				S~[1,7]				S~[1,10]				S~[1,100]			
			<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,100]	
			LPT	RLPT	LPT	RLPT	LPT	RLPT	LPT	RLPT	LPT	RLPT	LPT	RLPT	LPT	RLPT	LPT	RLPT	LPT	RLPT	LPT	RLPT
2	3	6	4.031	2.966	3.149	3.068	2.738	1.655	3.206	2.030	1.159	2.540	2.378	1.373	1.404	1.622	1.493	1.385	3.454	1.316		
	4	8	4.442	2.969	4.071	4.864	3.933	2.877	1.912	2.835	2.772	1.624	4.809	3.727	5.787	4.083	4.364	2.053	4.119	2.043		
	5	10	5.291	5.315	3.538	3.707	4.839	3.306	6.543	4.975	3.200	2.814	3.863	3.518	2.608	1.137	3.529	3.433	3.309	3.919		
	10	20	6.747	7.354	5.936	7.111	5.591	5.332	6.009	4.570	5.457	4.763	6.285	5.900	5.088	4.958	5.236	5.163	5.886	5.486		
3	3	9	2.351	1.646	3.942	1.117	2.374	1.000	3.156	2.491	2.834	1.768	4.790	1.859	2.262	1.336	3.675	1.833	2.276	0.675		
	4	12	4.277	2.329	3.398	2.787	5.576	3.898	2.837	1.989	3.064	2.368	2.759	1.926	2.442	1.759	2.577	1.570	2.693	2.441		
	5	15	4.351	2.762	5.129	3.557	3.635	2.500	2.645	2.453	6.305	3.904	4.907	4.193	2.665	1.764	2.951	2.613	2.960	2.841		
	10	30	4.628	3.266	2.997	2.735	3.873	3.844	5.035	4.699	4.429	4.007	3.227	2.513	4.154	3.634	3.230	2.735	4.030	4.113		
4	3	12	1.940	0.434	1.804	1.521	2.218	1.411	1.599	1.094	2.245	1.099	1.265	0.944	1.405	0.735	2.636	1.354	1.828	1.085		
	4	16	1.382	1.086	2.146	1.549	3.113	2.191	1.464	1.235	2.335	1.251	3.028	1.315	1.268	0.936	1.384	1.586	2.178	1.450		
	5	20	1.939	1.412	2.266	1.756	2.898	1.844	1.310	1.143	2.156	1.585	1.572	1.201	1.527	1.514	1.654	1.432	2.339	1.877		
	10	40	2.247	1.796	1.874	1.872	1.589	1.616	2.532	1.792	2.644	1.901	2.400	1.950	2.428	1.928	2.281	1.962	2.063	2.273		
5	3	15	1.159	0.441	1.167	0.704	1.471	0.974	1.387	0.592	1.664	1.027	1.478	0.661	0.985	0.512	0.647	0.501	1.443	0.972		
	4	20	0.599	0.381	0.941	0.669	1.334	0.997	0.557	0.241	1.320	0.582	1.362	0.945	1.010	0.668	1.232	0.636	1.466	0.876		
	5	25	1.822	1.034	0.915	1.027	1.672	1.441	1.095	0.957	2.324	1.233	1.102	1.190	1.389	1.058	1.468	1.295	1.371	1.286		
	10	50	1.390	1.275	1.349	1.246	1.614	1.451	1.702	1.316	1.451	1.163	1.400	1.074	1.634	1.360	1.487	1.242	1.342	1.142		
10	3	30	0.480	0.303	0.523	0.365	0.329	0.239	0.465	0.338	0.426	0.209	0.272	0.232	0.648	0.402	0.433	0.159	0.346	0.185		
	4	40	0.566	0.370	0.366	0.279	0.591	0.401	0.478	0.400	0.567	0.365	0.487	0.280	0.439	0.295	0.494	0.320	0.493	0.300		
	5	50	0.630	0.382	0.396	0.259	0.472	0.318	0.407	0.314	0.427	0.281	0.397	0.337	0.428	0.351	0.462	0.378	0.294	0.243		
	10	100	0.539	0.442	0.464	0.378	0.415	0.314	0.364	0.341	0.321	0.274	0.401	0.251	0.412	0.346	0.439	0.331	0.337	0.320		
20	3	60	0.154	0.106	0.116	0.084	0.089	0.058	0.157	0.112	0.135	0.058	0.143	0.052	0.150	0.135	0.140	0.067	0.101	0.048		
	4	80	0.213	0.131	0.089	0.069	0.114	0.079	0.179	0.179	0.079	0.061	0.100	0.059	0.132	0.126	0.086	0.079	0.085	0.069		
	5	100	0.164	0.164	0.134	0.122	0.102	0.100	0.190	0.155	0.176	0.104	0.094	0.080	0.164	0.159	0.118	0.087	0.099	0.082		
	10	200	0.215	0.171	0.099	0.093	0.095	0.070	0.172	0.172	0.141	0.098	0.110	0.095	0.168	0.168	0.109	0.098	0.114	0.085		
30	3	90	0.066	0.066	0.053	0.052	0.045	0.031	0.103	0.103	0.055	0.052	0.041	0.024	0.117	0.094	0.088	0.045	0.043	0.031		
	4	120	0.121	0.121	0.061	0.044	0.047	0.041	0.097	0.097	0.071	0.045	0.035	0.035	0.082	0.082	0.048	0.050	0.043	0.037		
	5	150	0.097	0.097	0.070	0.055	0.036	0.045	0.116	0.109	0.058	0.053	0.043	0.037	0.089	0.089	0.065	0.051	0.048	0.036		
	10	300	0.120	0.120	0.063	0.060	0.059	0.040	0.116	0.109	0.064	0.064	0.053	0.042	0.120	0.120	0.066	0.057	0.049	0.038		
Average			1.856	1.391	1.681	1.470	1.816	1.360	1.637	1.316	1.710	1.260	1.743	1.279	1.465	1.121	1.514	1.161	1.600	1.260		

The results in Table 3 demonstrate that the RLPT algorithm outperformed the LPT algorithm in terms of the RPD. In almost all combinations of (*m*, *n*) and (*S*, *p*), the RLPT algorithm had better average RPD than the LPT algorithm. Considering each combination of (*S*, *p*), the maximum overall average RPD across all machine–job combinations (*m*, *n*) for the RLPT algorithm was 1.470, compared with 1.856 for the LPT algorithm. This indicates that the randomization introduced to the LPT algorithm helped to minimize the makespan

and provided better schedules. Additionally, for both algorithms, the *RPD* tended to decrease as the job-to-machine ratio (n/m) increased.

Furthermore, for each processing time interval, the overall average *RPD* for both algorithms decreased as the interval of the speed of machines increased, i.e., from $S\sim[1, 3]$ to $S\sim[1, 7]$. There was one exception: for the *LPT* rule with processing times in $p\sim[1, 50]$, the overall average *RPD* increased slightly, by about 1.73%, when the speed range went from $S\sim[1, 3]$ to $S\sim[1, 5]$. However, it then decreased as the speed range went to $S\sim[1, 7]$. Figure 4 shows a visual representation of the overall average *RPD* for the *LPT* and *RLPT* algorithms for each combination of machine speed and processing time intervals.

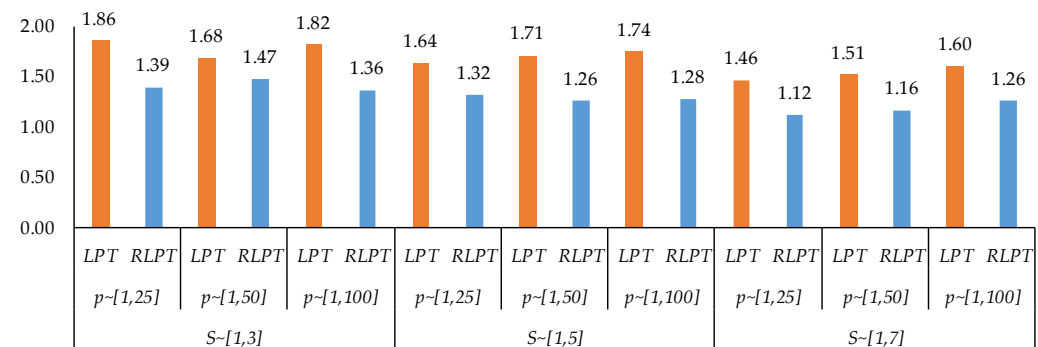


Figure 4. Average *RPD* for the *LPT* and *RLPT* heuristics.

To evaluate the performance of *LPT* and *RLPT* for each combination of (m , n/m , P_{max} , and S_{max}), we further analyzed the results for all ten instances generated for each combination. Table 4 and Figure 5 summarize the percentage of times each algorithm obtained a better, worse, or equal makespan compared to the other algorithm. Results from the table show the superior performance of the *RLPT* heuristic over the *LPT* heuristic across all processing time intervals and machine speed intervals. In all instance combinations, the percentage of times the *RLPT* heuristic provided a better makespan than the *LPT* heuristic exceeded 33.2%, reaching 62.5%. Conversely, the percentage of times *LPT* outperformed *RLPT* did not exceed 24.3%. The proportion of cases where both algorithms achieved the same solution was high, reaching 51.1% for instances with processing times between 1 and 25. However, this percentage decreased to 16.1% as the processing time interval increased.

Table 4. Outperformance comparison of *LPT* and *RLPT* algorithms.

	$p\sim[1,25]$			$p\sim[1,50]$			$p\sim[1,100]$		
	$LPT < RLPT$	$LPT > RLPT$	$LPT = RLPT$	$LPT < RLPT$	$LPT > RLPT$	$LPT = RLPT$	$LPT < RLPT$	$LPT > RLPT$	$LPT = RLPT$
$S\sim[1,3]$	11.8%	44.3%	43.9%	20.7%	47.1%	32.1%	24.3%	57.9%	17.9%
$S\sim[1,5]$	14.3%	33.2%	52.5%	13.2%	54.3%	32.5%	21.4%	62.5%	16.1%
$S\sim[1,7]$	12.1%	36.8%	51.1%	18.2%	49.6%	32.1%	23.6%	55.4%	21.1%

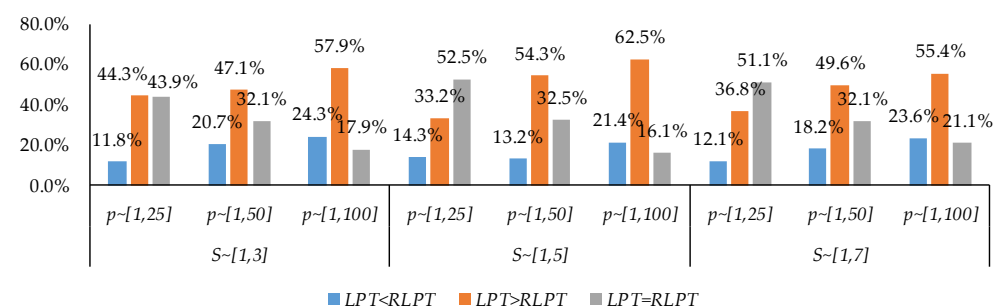


Figure 5. Bar chart for the outperformance comparison between *LPT* and *RLPT* algorithms.

4.2. Performance of the LVNS and RVNS Algorithms

This section presents a comparative analysis of the two proposed variable neighborhood search algorithms (VNS): the LVNS algorithm, which employed the LPT heuristic as an initial solution, and the RVNS algorithm, which employed the RLPT heuristic as an initial solution. First, the proposed VNS algorithm was compared with GA and TS on a numerical example from the literature. Then, the performance between LVNS and RVNS was evaluated on the instances generated in this study.

Table 5 presents a numerical example from Balin [21] involving nine jobs to be scheduled on four uniform parallel machines. The table details the processing times of each job, listed in descending order, and the speed of each machine.

Table 5. Processing times of nine jobs on four uniform parallel machines [21].

Machine	Speed	Job								
		4	8	3	7	6	1	5	2	9
1	1	30	26	24	22	20	18	16	14	14
2	2	15	13	12	11	10	9	8	7	7
3	4	7.5	6.5	6	5.5	5	4.5	4	3.5	3.5
4	5	6	5.2	4.8	4.4	4	3.6	3.2	2.8	2.8

The makespan obtained by the LVNS, RVNS, GA, and TS algorithms, the schedule for each algorithm, and the maximum iterations are presented in Table 6.

Table 6. Comparison of the proposed versions of VNS algorithms with GA [21] and TS [22].

Algorithm	Machine	Schedule of Jobs	Completion Time	Max. Iterations
GA	1	5	14	12
	2	1, 9	16	
	3	2, 6, 8	15.5	
	4	4, 7, 3	15.2	
TS	1	2	14	5000
	2	4	15	
	3	1, 3, 6	15.5	
	4	5, 7, 8, 9	15.6	
LVNS	1	2	14	1
	2	4	15	
	3	5, 6, 8	15.5	
	4	1, 3, 7, 9	15.6	
RVNS	1	9	14	10
	2	4	15	
	3	5, 6, 8	15.5	
	4	1, 2, 3, 7	15.6	

Table 6 shows that the proposed VNS algorithms (LVNS and RVNS) and the TS algorithm achieved the minimum makespan (15.6), which is the optimal solution for this example found by the mathematical formulation. GA obtained a suboptimal makespan of 16 within 12 iterations, as reported by Balin [21]. While the maximum number of iterations used for the TS algorithm is 5000, the authors did not specify the number of iterations required to reach the optimal solution. On the other hand, the LVNS algorithm obtained the optimal solution in one iteration and RVNS achieved the optimal solution more than once within ten iterations. In both GA and TS algorithms, the time taken to reach the best solution was not reported by the authors.

Interestingly, both LVNS and RVNS algorithms converged to the optimal solution, with execution times of 0.0012 s and 0.0069 s, respectively.

Since both *GA* and *TS* algorithms were tested against a small numerical example with nine and four uniform parallel machines, we further tested the performance of the proposed *VNS* algorithms through large-sized instances with up to 10 machines and 300 jobs.

The findings of the comparison between the *LVNS* and *RVNS* algorithms on the generated instances in this study can be analyzed in two parts. Instances with up to 20 jobs and instances with more than 20 jobs.

For instances with up to 20 jobs, Table 7 compares the number of times each of the two *VNS* algorithms, *LVNS* and *RVNS*, found the optimal solution identified by the mathematical formulation presented in Section 3.2.

Table 7. Comparison of the number of optimal solutions for small instances.

<i>n/m</i>	<i>m</i>	<i>n</i>	<i>S</i> ~[1,3]						<i>S</i> ~[1,5]						<i>S</i> ~[1,7]					
			<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]	
			<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>	<i>LVNS</i>	<i>RVNS</i>
2	3	6	9	10	10	10	9	10	8	10	10	10	9	10	10	9	10	10	9	10
	4	8	8	10	9	10	7	10	8	10	8	9	9	10	9	10	9	10	6	10
	5	10	6	9	8	10	5	10	5	9	6	8	8	10	6	10	7	10	6	10
	10	20	0	2	0	3	1	2	1	4	0	3	1	1	1	2	1	5	1	2
3	3	9	6	9	6	9	5	10	6	10	4	10	6	10	8	10	10	10	8	10
	4	12	6	10	6	8	2	10	4	10	5	10	3	8	9	10	3	9	1	9
	5	15	6	10	6	9	1	8	4	10	4	8	2	7	7	9	6	10	2	7
4	3	12	9	10	8	10	6	9	10	10	8	10	7	9	9	10	8	9	5	9
	4	16	9	10	9	10	1	9	8	10	5	8	6	10	6	10	9	10	5	7
	5	20	10	10	7	10	5	9	9	10	6	10	5	8	8	10	7	10	3	9
5	3	15	9	10	9	10	7	10	10	10	9	10	5	10	10	10	10	10	7	10
	4	20	10	10	9	10	8	10	8	10	10	10	9	10	10	10	6	10	5	10
Average			0.733	0.917	0.725	0.908	0.475	0.892	0.675	0.942	0.625	0.883	0.583	0.858	0.775	0.917	0.717	0.942	0.483	0.858

Table 7 demonstrates that, for each combination of (*n*, *m*) and (*S*, *p*) that contains 10 instances, the *RVNS* algorithm generally outperformed the *LVNS* algorithm. The *RVNS* algorithm reached the optimal solution more frequently than the *LVNS* algorithm. On average, for each combination of (*S*, *p*) and across all machine–job combinations (*m*, *n*), the *LVNS* algorithm found the optimal solution in 47.5% to 77.5% of the instances. In comparison, the *RVNS* algorithm showed a higher average number of optimal solutions, ranging between 0.858 and 0.942. Overall, considering all the instances with up to 20 jobs, the *LVNS* algorithm found the optimal solution in 64.35% of cases, while *RVNS* achieved a significantly higher average of 90.19%. This indicates that *RVNS* was more effective at finding better schedules for small-sized problems.

The relative percent deviation (*RPD*) for each of the *VNS* algorithms across all the instances is presented in Table 8. The table shows that, in instances with a ratio of $n/m \leq 5$, in most of the instances, the *RVNS* algorithm generally outperformed the *LVNS* in minimizing the average *RPD*. In a few instances, both *VNS* algorithms achieved equal performance. The table also reveals that in only one specific combination did the *LVNS* algorithm achieve an *RPD* of 0, while the *RVNS* algorithm achieved an *RPD* of 0.1.

Conversely, for instances with a ratio of $n/m > 5$, both algorithms exhibited similar *RPD* values. This suggests that the initial solution had a significant impact on *RPD* for small-sized problems. However, this impact diminished as the size of the problem increased.

Figure 6 visually depicts the average *RPD* for the *LVNS* and *RVNS* algorithms across all machine–job combinations, considering the different machine speeds and processing time intervals. The figure shows that the *RVNS* algorithm achieved both a lower minimum average *RPD* (0.08) and a lower maximum average *RPD* (0.243) compared to the *LVNS* algorithm (minimum: 0.240, maximum: 0.479). Additionally, the average *RPD* across all instances was lower for the *RVNS* algorithm (about 0.15) compared to (about 0.32) the *LVNS* algorithm.

Table 8. Relative percentage deviation comparison between LVNS and RVNS.

<i>n/m</i>	<i>m</i>	<i>n</i>	<i>S</i> ~[1,3]						<i>S</i> ~[1,5]						<i>S</i> ~[1,7]					
			<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]	
			LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS
2	3	6	0.179	0.000	0.000	0.000	0.055	0.000	0.651	0.000	0.000	0.000	0.374	0.000	0.000	0.100	0.000	0.000	0.066	0.000
	4	8	0.606	0.000	0.580	0.000	0.297	0.000	0.306	0.000	0.405	0.211	0.296	0.000	0.233	0.000	0.233	0.000	1.794	0.000
	5	10	0.623	0.023	0.807	0.000	1.373	0.000	2.866	0.267	0.318	0.154	0.102	0.000	0.430	0.000	0.581	0.000	0.573	0.000
	10	20	2.654	0.790	1.833	0.512	1.955	0.674	2.034	0.386	1.468	0.703	1.683	0.800	1.998	1.073	1.206	0.525	2.868	0.809
3	3	9	0.496	0.149	0.409	0.060	0.524	0.000	0.438	0.000	0.704	0.000	0.688	0.000	0.586	0.000	0.000	0.000	0.100	0.000
	4	12	0.408	0.000	0.286	0.096	0.779	0.000	0.873	0.000	0.337	0.000	1.018	0.234	0.071	0.000	0.633	0.033	0.955	0.027
	5	15	0.495	0.000	0.092	0.009	0.465	0.024	0.605	0.000	0.533	0.123	0.600	0.043	0.177	0.030	0.251	0.000	0.452	0.115
	10	30	1.247	1.065	1.059	0.665	0.740	0.386	1.154	1.021	0.777	0.697	0.693	0.386	1.367	1.178	1.110	0.545	0.847	0.534
4	3	12	0.041	0.000	0.104	0.000	0.091	0.012	0.000	0.000	0.138	0.000	0.037	0.004	0.057	0.000	0.067	0.023	0.188	0.014
	4	16	0.046	0.000	0.024	0.000	0.421	0.007	0.102	0.000	0.083	0.075	0.175	0.000	0.184	0.000	0.034	0.000	0.097	0.034
	5	20	0.000	0.000	0.040	0.000	0.094	0.017	0.042	0.000	0.153	0.000	0.104	0.028	0.053	0.000	0.157	0.000	0.342	0.018
	10	40	0.889	0.889	0.538	0.478	0.328	0.233	0.820	0.820	0.492	0.447	0.348	0.226	0.891	0.858	0.545	0.502	0.384	0.237
5	3	15	0.058	0.000	0.053	0.000	0.022	0.000	0.000	0.000	0.062	0.000	0.135	0.000	0.000	0.000	0.000	0.000	0.072	0.000
	4	20	0.000	0.000	0.010	0.000	0.011	0.000	0.076	0.000	0.000	0.000	0.018	0.000	0.000	0.000	0.082	0.000	0.071	0.000
	5	25	0.635	0.610	0.381	0.348	0.167	0.167	0.586	0.586	0.288	0.287	0.219	0.148	0.655	0.655	0.395	0.309	0.202	0.139
	10	50	0.663	0.663	0.418	0.395	0.187	0.172	0.654	0.654	0.342	0.340	0.200	0.176	0.704	0.704	0.351	0.326	0.191	0.172
10	3	30	0.253	0.253	0.134	0.134	0.069	0.069	0.269	0.269	0.136	0.136	0.082	0.082	0.303	0.303	0.125	0.125	0.076	0.076
	4	40	0.345	0.345	0.165	0.165	0.074	0.074	0.292	0.292	0.166	0.166	0.089	0.089	0.270	0.270	0.155	0.155	0.074	0.074
	5	50	0.328	0.328	0.158	0.158	0.075	0.075	0.297	0.297	0.159	0.159	0.080	0.080	0.332	0.332	0.157	0.157	0.087	0.087
	10	100	0.429	0.429	0.158	0.158	0.092	0.092	0.338	0.338	0.183	0.183	0.099	0.099	0.335	0.335	0.162	0.162	0.085	0.085
20	3	60	0.106	0.106	0.067	0.067	0.032	0.032	0.112	0.112	0.058	0.058	0.031	0.031	0.135	0.135	0.067	0.067	0.028	0.028
	4	80	0.131	0.131	0.068	0.068	0.031	0.031	0.173	0.173	0.061	0.061	0.032	0.032	0.126	0.126	0.058	0.058	0.035	0.035
	5	100	0.164	0.164	0.080	0.080	0.036	0.036	0.137	0.137	0.082	0.082	0.044	0.044	0.159	0.159	0.078	0.078	0.046	0.046
	10	200	0.165	0.165	0.084	0.084	0.044	0.044	0.172	0.172	0.079	0.079	0.045	0.045	0.168	0.168	0.089	0.089	0.042	0.042
30	3	90	0.066	0.066	0.040	0.040	0.017	0.017	0.103	0.103	0.052	0.052	0.022	0.022	0.094	0.094	0.037	0.037	0.022	0.022
	4	120	0.121	0.121	0.044	0.044	0.022	0.022	0.097	0.097	0.045	0.045	0.023	0.023	0.082	0.082	0.045	0.045	0.026	0.026
	5	150	0.097	0.097	0.053	0.053	0.024	0.024	0.109	0.109	0.051	0.051	0.027	0.027	0.089	0.089	0.051	0.051	0.025	0.025
	10	300	0.120	0.120	0.060	0.060	0.030	0.030	0.109	0.109	0.064	0.064	0.029	0.029	0.120	0.120	0.057	0.057	0.031	0.031
Average			0.406	0.233	0.277	0.131	0.288	0.080	0.479	0.212	0.259	0.149	0.260	0.095	0.343	0.243	0.240	0.119	0.349	0.096

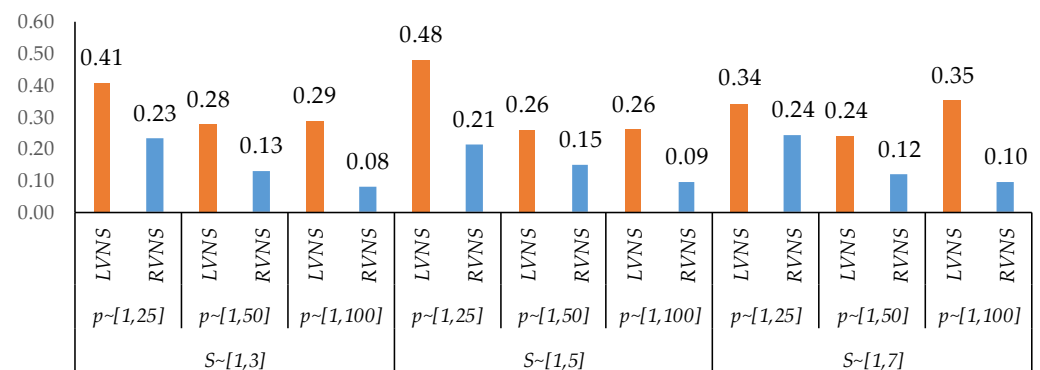


Figure 6. Average RPD for the LVNS and RVNS algorithms.

To evaluate the performance of each algorithm in obtaining a better, worse, or equal makespan compared to the other algorithm, we analyzed the 10 instances of each combination of m , n/m , P_{max} , and S_{max} . Table 9 shows that for the LVNS algorithm, the percent of times it obtained a makespan lower than the RVNS did not exceed 1.1% across all instance combinations. Conversely, for the RVNS algorithm, we noticed that the minimum percentage of instances where it outperformed the LVNS algorithm in terms of makespan was 11.1%. This percentage increased as the processing time interval increased, reaching a maximum of 28.6%. The case when both algorithms obtained an equal makespan had the highest percentage, ranging from 70.4% up to 88.6%. Figure 7 shows a visual representation of the outperformance of each of the VNS algorithms.

Table 9. Performance comparison of LVNS and RVNS algorithms.

	$p \sim [1,25]$			$p \sim [1,50]$			$p \sim [1,100]$		
	LVNS < RVNS	LVNS > RVNS	LVNS = RVNS	LVNS < RVNS	LVNS > RVNS	LVNS = RVNS	LVNS < RVNS	LVNS > RVNS	LVNS = RVNS
$S \sim [1,3]$	0.0%	11.4%	88.6%	0.4%	17.1%	82.5%	1.1%	28.6%	70.4%
$S \sim [1,5]$	0.4%	13.9%	85.7%	0.7%	17.9%	81.4%	1.1%	26.1%	72.9%
$S \sim [1,7]$	0.4%	11.1%	88.6%	0.0%	16.4%	83.6%	1.1%	27.9%	71.1%

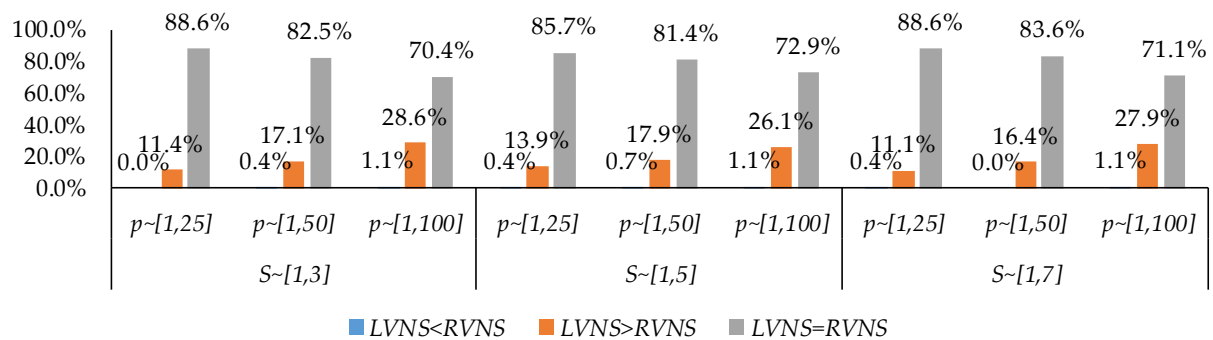
**Figure 7.** Bar chart for the outperformance comparison of LVNS and RVNS algorithms.

Table 10 shows the average CPU time (in seconds) for the LVNS and the RVNS algorithms. Overall, RVNS generally required more CPU time than LVNS. This is likely due to using only one iteration for each instance with LVNS, whereas ten iterations were implemented with RVNS. Moreover, the increase in time in RVNS can also be due to the initial solution, which required visiting the neighborhood structures many times to improve the solution. Once an improvement occurred, the VNS algorithm iteratively re-evaluated all machines and neighborhood structures until there was no further improvement. At this point, the algorithm terminated. The average CPU time for LVNS was consistently equal to 0.001 s. On the other hand, for RVNS, the average CPU time varied between a minimum of 0.155 s and a maximum of 0.311 s.

As observed in the previous comparison of both LVNS and RVNS algorithms, the outperformance of RVNS was in instances when the number of jobs was up to 20, and the equal performance was when the job-machine ratio was $n/m > 5$. This suggests that applying the RVNS algorithm for small instances, despite the slightly increased CPU time, can be beneficial. However, for instances with a job-machine ratio greater than 5, even though the RVNS incurred a higher CPU time, it achieved the same average RPD as the faster LVNS algorithm, which required minimal CPU time.

The time reduction of using the LVNS solution as an upper bound in finding the optimal solution in a mathematical formulation solved with CPLEX is investigated in Figure 8 for the instances where the number of jobs was less than or equal to 20. The figure shows that the CPU time required to solve the scheduling problem was reduced when using the VNS solution as an upper bound. The minimum and maximum percentage reductions in CPU time were 13.49% and 41.44%, respectively. This improvement occurred because the upper bound provided CPLEX with a tighter estimate of the optimal solution. This allowed CPLEX to prune more irrelevant branches of the search tree, leading to a more efficient convergence to the optimal solution. However, it is important to note that for some instances, specifying an upper bound for CPLEX might not always guarantee a reduction in CPU time compared to solving without specifying an upper bound.

Table 10. Comparison of average CPU time (in seconds) between LVNS and RVNS.

<i>n/m</i>	<i>m</i>	<i>n</i>	<i>S</i> ~[1,3]						<i>S</i> ~[1,5]						<i>S</i> ~[1,7]					
			<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]		<i>p</i> ~[1,25]		<i>p</i> ~[1,50]		<i>p</i> ~[1,100]	
			LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS	LVNS	RVNS
2	3	6	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	4	8	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001
	5	10	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001
	10	20	0.000	0.004	0.000	0.005	0.000	0.005	0.000	0.004	0.000	0.005	0.000	0.006	0.000	0.005	0.000	0.005	0.000	0.005
3	3	9	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000	0.000	0.000	0.000
	4	12	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001
	5	15	0.000	0.001	0.000	0.002	0.000	0.002	0.000	0.001	0.000	0.002	0.000	0.002	0.000	0.001	0.000	0.002	0.000	0.002
	10	30	0.000	0.005	0.000	0.006	0.000	0.009	0.000	0.006	0.000	0.008	0.000	0.008	0.000	0.006	0.000	0.008	0.000	0.009
4	3	12	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001
	4	16	0.000	0.001	0.000	0.001	0.000	0.002	0.000	0.001	0.000	0.001	0.000	0.002	0.000	0.001	0.000	0.001	0.000	0.002
	5	20	0.000	0.002	0.000	0.002	0.000	0.003	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.003
	10	40	0.000	0.006	0.000	0.008	0.000	0.011	0.000	0.006	0.000	0.009	0.000	0.012	0.000	0.007	0.000	0.010	0.000	0.013
5	3	15	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001	0.000	0.001
	4	20	0.000	0.001	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.002	0.000	0.002
	5	25	0.000	0.002	0.000	0.003	0.000	0.004	0.000	0.002	0.000	0.003	0.000	0.004	0.000	0.003	0.000	0.003	0.000	0.004
	10	50	0.000	0.008	0.000	0.010	0.000	0.015	0.000	0.009	0.000	0.012	0.000	0.016	0.000	0.008	0.000	0.014	0.000	0.018
10	3	30	0.000	0.005	0.000	0.006	0.000	0.007	0.000	0.005	0.000	0.006	0.000	0.007	0.000	0.006	0.000	0.006	0.000	0.007
	4	40	0.000	0.011	0.000	0.011	0.000	0.014	0.000	0.012	0.000	0.013	0.000	0.014	0.000	0.011	0.000	0.014	0.000	0.016
	5	50	0.000	0.015	0.000	0.014	0.000	0.023	0.000	0.014	0.000	0.015	0.000	0.025	0.000	0.016	0.000	0.023	0.000	0.025
	10	100	0.000	0.035	0.001	0.058	0.001	0.079	0.000	0.044	0.000	0.062	0.001	0.084	0.000	0.061	0.001	0.080	0.001	0.120
20	3	60	0.000	0.074	0.000	0.081	0.000	0.084	0.000	0.060	0.000	0.059	0.000	0.084	0.000	0.060	0.000	0.069	0.000	0.075
	4	80	0.001	0.124	0.000	0.133	0.001	0.165	0.000	0.132	0.000	0.116	0.001	0.166	0.000	0.141	0.001	0.161	0.001	0.185
	5	100	0.001	0.178	0.001	0.212	0.001	0.264	0.001	0.257	0.001	0.220	0.001	0.234	0.001	0.215	0.001	0.247	0.001	0.275
	10	200	0.002	0.451	0.002	0.508	0.002	0.585	0.002	0.490	0.002	0.488	0.003	0.811	0.003	0.802	0.003	0.816	0.005	1.291
30	3	90	0.001	0.349	0.001	0.396	0.001	0.389	0.001	0.312	0.001	0.361	0.001	0.348	0.001	0.329	0.001	0.328	0.001	0.386
	4	120	0.003	0.640	0.002	0.522	0.002	0.788	0.002	0.630	0.002	0.708	0.003	0.833	0.003	0.819	0.002	0.732	0.003	0.952
	5	150	0.003	0.883	0.002	0.733	0.003	1.008	0.003	0.747	0.002	0.696	0.003	1.046	0.003	1.089	0.004	1.164	0.005	1.531
	10	300	0.007	1.536	0.008	2.540	0.011	3.200	0.008	2.329	0.012	3.528	0.012	4.005	0.013	4.093	0.017	5.027	0.011	3.793
Average			0.001	0.155	0.001	0.188	0.001	0.238	0.001	0.181	0.001	0.226	0.001	0.276	0.001	0.274	0.001	0.311	0.001	0.311

**Figure 8.** Average CPU time reduction with the LVNS upper bound.

5. Conclusions and Future Work

In this paper, we studied the scheduling problem of uniform parallel machines with a makespan objective. Two versions of the variable neighborhood search (VNS) algorithm were proposed, differing in their initial solution generation strategy. The first version (LVNS) utilized the longest processing time (LPT) rule for generating the initial solution, while the second version (RVNS) introduced a new variant of the LPT rule by incorporating randomization in job selection. The proposed VNS algorithms were compared with the genetic algorithm and tabu search on a numerical example from the literature. Results showed that the proposed VNS algorithms were competitive and obtained the optimal solution with much less effort. Furthermore, a mathematical formulation and a lower bound were presented to assess the performance of the initial solution heuristics and the two VNS algorithms on randomly generated instances. Computational results showed that the randomized longest processing time rule outperformed the longest processing time rule. In addition, the RVNS algorithm outperformed the LVNS algorithm in finding the optimal solution in 90.19% of small-sized instances and achieved a minimum average relative percentage deviation of 0.15% across all instances. Large-sized instances exhibited

equal performance for both algorithms. We also demonstrated that incorporating an upper bound in the mathematical formulation can significantly reduce the solution time of the CPLEX solver. This research opens doors for exciting advancements. An efficient exact formulation, such as an arc flow formulation, can be developed for the studied uniform parallel machine problem with a makespan objective. This formulation can leverage the VNS algorithm's solutions as an estimate for the upper bound. Furthermore, future work can explore incorporating machine learning into the VNS algorithms to predict promising neighborhood structures during local search, leading to better convergence. Additionally, the VNS algorithm can be extended to handle multiple, competing objectives like minimizing makespan, flow time, tardiness, or energy consumption, making it more relevant to real-world scheduling scenarios with various objectives.

Author Contributions: Conceptualization, K.B. and A.G.; methodology, K.B. and A.G.; software, K.B.; validation, K.B. and A.G.; formal analysis, K.B. and A.G.; investigation, K.B. and A.G.; resources, K.B. and A.G.; data curation, K.B. and A.G.; writing—original draft preparation, K.B.; writing—review and editing, K.B. and A.G.; visualization, K.B. and A.G.; supervision, A.G.; project administration, A.G.; funding acquisition, A.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Researchers Supporting Project (number RSPD2024R-1039), King Saud University, Riyadh, Saudi Arabia.

Data Availability Statement: The data are available upon request from the corresponding author.

Acknowledgments: The authors extend their appreciation to the Researchers Supporting Project (number RSPD2024R1039), King Saud University, Riyadh, Saudi Arabia.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lee, J.-H.; Jang, H. Uniform parallel machine scheduling with dedicated machines, job splitting and setup resources. *Sustainability* **2019**, *11*, 7137. [\[CrossRef\]](#)
2. Huang, J.; Wu, J.; Tang, Y.; Hao, Y. The influences of openness on China's industrial CO₂ intensity. *Environ. Sci. Pollut. Res.* **2020**, *27*, 15743–15757. [\[CrossRef\]](#) [\[PubMed\]](#)
3. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Kan, A.R. Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of Discrete Mathematics*; Elsevier: Amsterdam, The Netherlands, 1979; Volume 5, pp. 287–326.
4. Garey, M.R.; Johnson, D.S. *Computers and Intractability*; Freeman: San Francisco, CA, USA, 1979; Volume 174.
5. Li, K.; Leung, J.T.; Cheng, B.Y. An agent-based intelligent algorithm for uniform machine scheduling to minimize total completion time. *Appl. Soft Comput.* **2014**, *25*, 277–284. [\[CrossRef\]](#)
6. Graham, R.L. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* **1969**, *17*, 416–429. [\[CrossRef\]](#)
7. Gonzalez, T.; Ibarra, O.H.; Sahni, S. Bounds for LPT schedules on uniform processors. *SIAM J. Comput.* **1977**, *6*, 155–166. [\[CrossRef\]](#)
8. Della Croce, F.; Scatamacchia, R. The longest processing time rule for identical parallel machines revisited. *J. Sched.* **2020**, *23*, 163–176. [\[CrossRef\]](#)
9. Friesen, D.K. Tighter bounds for LPT scheduling on uniform processors. *SIAM J. Comput.* **1987**, *16*, 554–560. [\[CrossRef\]](#)
10. Mireault, P.; Orlin, J.B.; Vohra, R.V. A parametric worst case analysis of the LPT heuristic for two uniform machines. *Oper. Res.* **1997**, *45*, 116–125. [\[CrossRef\]](#)
11. Koulamas, C.; Kyparisis, G.J. A modified LPT algorithm for the two uniform parallel machine makespan minimization problem. *Eur. J. Oper. Res.* **2009**, *196*, 61–68. [\[CrossRef\]](#)
12. Massabò, I.; Paletta, G.; Ruiz-Torres, A.J. A note on longest processing time algorithms for the two uniform parallel machine makespan minimization problem. *J. Sched.* **2016**, *19*, 207–211. [\[CrossRef\]](#)
13. Mitsunobu, T.; Suda, R.; Suppakitpaisarn, V. Worst-case analysis of LPT scheduling on a small number of non-identical processors. *Inf. Process. Lett.* **2024**, *183*, 106424. [\[CrossRef\]](#)
14. Friesen, D.K.; Langston, M.A. Bounds for multifit scheduling on uniform processors. *SIAM J. Comput.* **1983**, *12*, 60–70. [\[CrossRef\]](#)
15. Chen, B. Tighter bound for MULTIFIT scheduling on uniform processors. *Discret. Appl. Math.* **1991**, *31*, 227–260. [\[CrossRef\]](#)
16. Li, K.; Zhang, S.-C. Heuristics for uniform parallel machine scheduling problem with minimizing makespan. In Proceedings of the 2008 IEEE International Conference on Automation and Logistics, Qingdao, China, 1–3 September 2008; IEEE: New York, NY, USA, 2018; pp. 273–278.
17. Sivasankaran, P.; Kumar, M.R.; Senthilkumar, P.; Panneerselvam, R. Heuristic to minimize makespan in uniform parallel machines scheduling problem. *Udyog Pragati* **2009**, *33*, 1–15.

18. Li, K.; Yang, S. Heuristic algorithms for scheduling on uniform parallel machines with heads and tails. *J. Syst. Eng. Electron.* **2011**, *22*, 462–467. [\[CrossRef\]](#)
19. De Giovanni, D.; Ho, J.C.; Paletta, G.; Ruiz-Torres, A.J. Heuristics for Scheduling Uniform Machines. In Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong Kong, China, 14–16 March 2018; Volume 2.
20. Song, J.; Miao, C.; Kong, F. Uniform-machine scheduling problems in green manufacturing system. *Math. Found. Comput.* **2024**. [\[CrossRef\]](#)
21. Balin, S. Non-identical parallel machine scheduling using genetic algorithm. *Expert Syst. Appl.* **2011**, *38*, 6814–6821. [\[CrossRef\]](#)
22. Noman, M.A.; Alatefi, M.; Al-Ahmari, A.M.; Ali, T. Tabu Search Algorithm Based on Lower Bound and Exact Algorithm Solutions for Minimizing the Makespan in Non-Identical Parallel Machines Scheduling. *Math. Probl. Eng.* **2021**, *2021*, 1856734. [\[CrossRef\]](#)
23. Li, K.; Cheng, B.-Y. Variable neighborhood search for uniform parallel machine makespan scheduling problem with release dates. In Proceedings of the 2010 International Symposium on Computational Intelligence and Design, Hangzhou, China, 29–31 October 2010; IEEE: New York, NY, USA, 2010; Volume 2, pp. 43–46.
24. Sevcli, M.; Uysal, H. A modified variable neighborhood search for minimizing the makespan on identical parallel machines. In Proceedings of the 2009 International Conference on Computers & Industrial Engineering, Troyes, France, 6–9 July 2009; IEEE: New York, NY, USA, 2009; pp. 108–111.
25. Chen, J.; Li, J.-Q. Efficient variable neighborhood search for identical parallel machines scheduling. In Proceedings of the 31st Chinese Control Conference, Hefei, China, 25–27 July 2012; IEEE: New York, NY, USA, 2012; pp. 7228–7232.
26. Alharkan, I.; Bamatraf, K.; Noman, M.A.; Kaid, H.; Nasr, E.S.A.; El-Tamimi, A.M. An order effect of neighborhood structures in variable neighborhood search algorithm for minimizing the makespan in an identical parallel machine scheduling. *Math. Probl. Eng.* **2018**, *2018*, 3586731. [\[CrossRef\]](#)
27. Gharbi, A.; Bamatraf, K. An Improved Arc Flow Model with Enhanced Bounds for Minimizing the Makespan in Identical Parallel Machine Scheduling. *Processes* **2022**, *10*, 2293. [\[CrossRef\]](#)
28. Cheng, W.; Guo, P.; Zhang, Z.; Zeng, M.; Liang, J. Variable neighborhood search for parallel machines scheduling problem with step deteriorating jobs. *Math. Probl. Eng.* **2012**, *2012*, 928312. [\[CrossRef\]](#)
29. Senthilkumar, P.; Narayanan, S. GA Based Heuristic to Minimize Makespan in Single Machine Scheduling Problem with Uniform Parallel Machines. *Intell. Inf. Manag.* **2011**, *3*, 204–214. [\[CrossRef\]](#)
30. Senthilkumar, P.; Narayanan, S. Simulated annealing algorithm to minimize makespan in single machine scheduling problem with uniform parallel machines. *Intell. Inf. Manag.* **2011**, *3*, 22. [\[CrossRef\]](#)
31. Kaabi, J. Modeling and solving scheduling problem with m uniform parallel machines subject to unavailability constraints. *Algorithms* **2019**, *12*, 247. [\[CrossRef\]](#)
32. Kim, H.-J.; Lee, J.-H. Scheduling uniform parallel dedicated machines with job splitting, sequence-dependent setup times, and multiple servers. *Comput. Oper. Res.* **2021**, *126*, 105115. [\[CrossRef\]](#)
33. Horowitz, E.; Sahni, S. Exact and approximate algorithms for scheduling nonidentical processors. *J. ACM* **1976**, *23*, 317–327. [\[CrossRef\]](#)
34. De, P.; Morton, T.E. Scheduling to minimize makespan on unequal parallel processors. *Decis. Sci.* **1980**, *11*, 586–602. [\[CrossRef\]](#)
35. Liao, C.-J.; Lin, C.-H. Makespan minimization for two uniform parallel machines. *Int. J. Prod. Econ.* **2003**, *84*, 205–213. [\[CrossRef\]](#)
36. Lin, C.-H.; Liao, C.-J. Makespan minimization for multiple uniform machines. *Comput. Ind. Eng.* **2008**, *54*, 983–992. [\[CrossRef\]](#)
37. Popenko, V.; Sperkach, M.; Zhdanova, O.; Kokosiński, Z. On Optimality Conditions for Job Scheduling on Uniform Parallel Machines. In Proceedings of the International Conference on Computer Science, Engineering and Education Applications, Kiev, Ukraine, 26–27 January 2019; Springer: New York, NY, USA, 2019; pp. 103–112.
38. Berndt, S.; Brinkop, H.; Jansen, K.; Mnich, M.; Stamm, T. New support size bounds for integer programming, applied to makespan minimization on uniformly related machines. *arXiv* **2023**, arXiv:2305.08432.
39. Mallek, A.; Boudhar, M. Scheduling on uniform machines with a conflict graph: Complexity and resolution. *Int. Trans. Oper. Res.* **2024**, *31*, 863–888. [\[CrossRef\]](#)
40. Soper, A.J.; Strusevich, V.A. Parametric analysis of the quality of single preemption schedules on three uniform parallel machines. *Ann. Oper. Res.* **2021**, *298*, 469–495. [\[CrossRef\]](#)
41. Mokotoff, E. Parallel machine scheduling problems: A survey. *Asia-Pac. J. Oper. Res.* **2001**, *18*, 193.
42. Senthilkumar, P.; Narayanan, S. Literature review of single machine scheduling problem with uniform parallel machines. *Intell. Inf. Manag.* **2010**, *2*, 457–474. [\[CrossRef\]](#)
43. Mladenović, N.; Hansen, P. Variable neighborhood search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.