processes

MDPI

Article

An Improved Arc Flow Model with Enhanced Bounds for Minimizing the Makespan in Identical Parallel Machine Scheduling

Anis Gharbi * and Khaled Bamatraf

Industrial Engineering Department, King Saud University, Riyadh 11421, Saudi Arabia
* Correspondence: a.gharbi@ksu.edu.sa

Abstract: In this paper, an identical parallel machine problem was considered with the objective of minimizing the makespan. This problem is NP-hard in the strong sense. A mathematical formulation based on an improved arc flow model with enhanced bounds was proposed. A variable neighborhood search algorithm was proposed to obtain an upper bound. Three lower bounds from the literature were utilized in the improved arc flow model to improve the efficiency of the mathematical formulation. In addition, a graph compression technique was proposed to reduce the size of the graph. As a consequence, the improved arc flow model was compared with an arc flow model from the literature. The computational results on benchmark instances showed that the improved arc flow model outperformed the literature arc flow model at finding optimal solutions for 99.97% of the benchmark instances, with the overall percentage of the reduction in time reaching 87%.

Keywords: identical parallel machines; improved arc flow; integer programming; scheduling; variable neighborhood search

Citation: 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. https://doi.org/10.3390/pr10112293

Academic Editors: Danyu Bai, Xin Chen, Dehua Xu and Jedrzej Musial

Received: 3 October 2022
Accepted: 23 October 2022
Published: 4 November 2022

1. Introduction

Scheduling is a decision-making process that deals with optimizing resource allocation to perform a collection of tasks in production or manufacturing processes. Scheduling is involved in many real-life applications, such as jobs in a manufacturing plant, customers waiting for services in front of a teller's window, or airplanes waiting for clearance to land or take off at an airport. Many studies have been carried out by researchers in several scheduling environments, such as single machine scheduling and parallel machine scheduling, to fill some gaps in these problems.

Single machine scheduling requires only the sequencing of jobs. However, scheduling problems with more than one machine, such as parallel machines, involve both resource allocation and sequencing, rather than simple sequencing [1]. The importance of parallel machine scheduling may be viewed from both theoretical and a practical perspectives [2]. From a theoretical perspective, it is a general case of the single machine and a special case of the flexible flow shop. A practical perspective is important since parallel operations are frequent in the real world. The makespan objective in parallel machines is of considerable interest, since it balances the load between the parallel machines.

In the parallel machine scheduling problem, each job is processed on only one machine and each machine can process one job at a time. The identical parallel machine is a variant of the machine scheduling problem that is characterized by a set of n independent jobs $J = J_1, J_2, \ldots, J_n$ to be scheduled in m identical parallel machines $M = M_1, M_2, \ldots, M_m$, where $(m < n)$. Each job J can be processed in only one machine with a processing time p_j that is identical on all the machines. Preemption is not allowed. Therefore, once a machine is processing a job, it must complete its processing without interruption. The objective is to minimize the total completion time of the last processed job in the schedule (makespan).

The identical parallel machine scheduling problem to minimize the makespan is described by Graham et al. [3] in terms of three fields: $\alpha|\beta|\gamma$ as $P_m||C_{max}$, where P_m indicates the identical parallel machine environment with m machines, the second empty field, which indicates no preemption or precedence constraints on the jobs, and C_{max} in the third field, which indicates the makespan minimization objective. This problem is proven to be NP-hard [4]. Therefore, efficient exact algorithms were needed to be able to solve large-size instances to optimality in less computational time.

The remainder of this paper is organized as follows: Section 2 presents a literature review of the identical parallel machine scheduling problem and the arc flow model. Section 3 details the methodology used in the problem. The computational results and discussion are provided in Section 4. Finally, the conclusion and future work are shown in Section 5.

2. Literature Review

The first known heuristic applied to the identical parallel machine was the longest processing time (*LPT*) rule proposed by Ronald L. Graham [5]. *LPT* works by sorting jobs in a non-increasing order of processing times and assigning jobs to the least-loaded machine one by one until assigning all the jobs. The worst-case approximation ratio for the *LPT* rule is $4/3–1/(3m)$, where m is the number of machines. Recently, Della Croce and Scatamacchia [6] revisited the *LPT* rule and improved the ratio to $4/3–1/(3m\text{-}1)$. Other popular approximation heuristics combine ($P_m||C_{max}$) with bin-packing techniques: *MULTIFIT* [7], *COMBINE* [8], and *LISTFIT* [9]. These heuristics provide a better worst-case performance, but with higher computation times.

Metaheuristics also play an important role in obtaining good solutions for parallel machine scheduling problems. Among these metaheuristics are the genetic algorithm [10], simulating annealing [11], the variable neighborhood search [12], and other metaheuristics in the literature.

Over the past years, researchers have increased their interest in finding efficient exact methods capable of solving large and hard instances to optimality in less computational time. Mokotoff [1] conducted a survey on parallel machine scheduling. Dell'Amico and Martello [13] proposed a branch-and-bound algorithm based on sophisticated lower and upper bounds and some dominance rules for ($P_m||C_{max}$). Mokotoff [14] proposed an exact method based on the cutting plane method for ($P_m||C_{max}$). Then, Dell'Amico and Martello [15] noted that their previous work [13], which had been published before the work of Mokotoff [14], obtained better results in terms of the computation time and solved all the studied instances to optimality. Dell'Amico et al. [16] presented a scatter search algorithm and exact algorithm based on branch-and-price algorithms, which make use of the duality between $P||C_{max}$ and the bin-packing problem. Haouari et al. [17] proposed lower bounds based on the lifting procedure. The authors also proposed two heuristics that required iteratively solving a subset sum problem. The previous work was extended by Haouari and Jemmali [18], enhanced lower bounds were obtained, a new heuristic was proposed based on solving a sequence of 0–1 knapsack problem, and these bounds were embedded in a branch-and-bound algorithm. To test the performance of their algorithm, the authors identified a set of hard instances for which the ratio between the number of jobs and the number of machines was equal to 2.5. The proposed branch-and-bound solved only about 68% of a total of 240 generated instances with different numbers of jobs and machines.

The arc flow approach has been used recently in classical optimization problems, and allows modeling with a pseudo-polynomial number of variables and constraints. For a cutting-stock problem, de Carvalho [19] proposed a branch-and-price approach for an arc-flow formulation. Next, it was extended for the bin-packing problem in De Carvalho [20]. An alternative arc-flow formulation for the cutting-stock problem was proposed in [21,22], which used a graph compression technique that reduced the size of the constructed graph without affecting the optimal solution. These formulations were recently

tested and compared in [23] against several other models and problem-specific algorithms on one-dimensional bin-packing and cutting-stock problems. J. Martinovic et al. [24] compared the arc-flow model with a one-cut model for the one-dimensional cutting-stock problem and presented reduction techniques for both models. M. Mrad et al. [25] proposed a graph compression method to an arc flow formulation for a two-dimensional strip-cutting problem. Other applications of arc flow include berth allocation problems [26], vehicle-routing problems [27], and facility location problems [28].

In the area of scheduling, Mrad and Souayah [29] proposed an arc flow formulation for the parallel machine scheduling problem to minimize the makespan. The proposed mathematical model outperformed other proposed methods from the literature, since it solved to optimality most of the hard instances from the literature in a few seconds on average. On the other hand, some hard instances were still unsolved within a predefined elimination time, mainly because the number of jobs was relatively large and the ratio of the number of jobs to the number of machines was greater than or equal to 2.25.

A. Kramer et al. [30] studied the identical parallel machine scheduling problem to minimize the total weighted completion time. An enhanced arc flow formulation with reduction techniques was proposed, which reduced the number of variables and constraints. As a consequence, large instances with up to 400 jobs were solved to optimality and instances with up to 1000 jobs provided a low optimal gap. Then, A. Kramer et al. [31] extended the previous work of A. Kramer et al. [30] by adding a release time constraint for each job. A mixed-integer linear program and a branch-and-price algorithm that relied on the decomposition of an arc-flow formulation and the use of exact and heuristic methods were proposed for solving pricing subproblems. S. Wang et al. [32] studied deterministic and parallel machine scheduling location problems and proposed a network flow-based formulation, two formulation-based heuristics, and one polynomial time algorithm. Trindade et al. [33,34] proposed an arc flow formulation for parallel and single batch processing machine scheduling with non-identical job sizes and a machine capacity. A. Kramer et al. [35] proposed five different formulations for identical parallel machine scheduling with family setup times to minimize the total weighted completion time. The formulations were one commodity formulation, three arc flow formulations, and a set covering formulation. The results showed that one of the arc flow formulations and a set covering formulation yielded a better performance. de Lima et al. [36] conducted a survey on the foundation of the arc flow formulation and showed the relation between their network and dynamic programming. The survey also discussed the main solution methods for solving large-scale arc flow models and their main applications. de Lima et al. [37] proposed a network flow framework to address huge network issues in arc flow model solutions.

To summarize, we can say that so far, there are two kinds of proposed approaches for makespan minimization on parallel machines: heuristics and exact methods. On the one hand, heuristics have the potential to find good solutions in a reasonable amount of time. However, they do not guarantee the optimality of the provided solutions. For instance, one of the most recent heuristics for the studied problem [6] can still be outperformed by older methods for some benchmark instances. On the other hand, exact algorithms deliver optimal schedules, but are limited by the size of the instances (number of jobs/machines) and the relatively high computation time. To illustrate, the state-of-the-art exact method [29] still fails to solve some instances with as much as 154 jobs. Moreover, it requires thousands of seconds to solve problems with less than 200 jobs.

The main aim of this paper was to move towards an efficient implementation of the arc flow model for makespan minimization in an identical parallel machine scheduling problem. Hence, the arc flow model proposed by Mrad and Souayah [29] was considered. The improvements were made by proposing enhanced bounds and graph compression techniques to reduce the number of variables and constraints in the constructed arc flow model. A variable neighborhood search (*VNS*) algorithm was proposed with five neighborhood structures and an initial solution obtained from the schedule of the longest processing

time (*LPT*) rule as an upper bound. Three lower bounds from the literature were considered for the improved arc flow model. A better upper bound was considered as a graph compression technique since it reduced the size of the graph. As a consequence, the number of variables was reduced. Furthermore, another proposed graph compression technique eliminated scheduling some jobs on the same machine, which made the resulting lower bound of the problem exceed the current upper bound obtained by the *VNS* algorithm. It is worth noting that devising an improved arc flow formulation has an important application in reducing the computation time for solving hard optimization problems. This can be attested by our experimental results as well as those of previous successful implementations of such techniques [30,31].

3. Methodology

In this section, the methodology of applying the improved arc flow model with enhanced bounds for the identical parallel machine scheduling problem to minimize the makespan is explained. The first step was to calculate the bounds of the problem. Three lower bounds were applied from the literature and the maximum among them was selected as the lower bound for the problem. The second step was to find an upper bound using the *LPT* rule. In Step 3, the obtained upper bound from the *LPT* rule became an input for the proposed *VNS* algorithm to further improve, if possible, the upper bound. The lower and upper bounds were then used to construct the graph of the improved arc flow model. To build the graph, a graph compression technique was proposed to determine the set of jobs that would have an arc leaving node 0. Then, the whole graph was constructed. After that, a mathematical formulation was proposed for the improved arc flow model. Finally, the mathematical model was solved with Cplex 12.10. If a solution was found within 20 min, then an optimal solution was found. Otherwise, the instance was not solved to optimality within the specified time limit. The flowchart of the methodology is shown in Figure 1. The details for each step are illustrated in the following subsections.

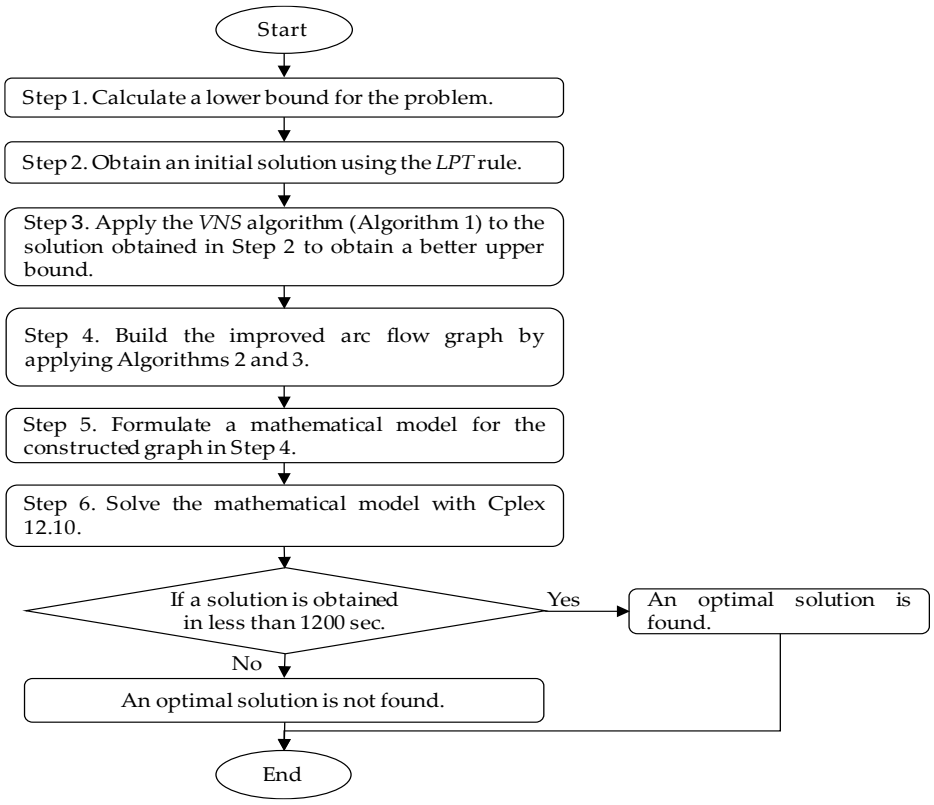

Figure 1. Flowchart of the improved arc flow methodology.

3.1. Lower Bounds

The lower bounds used in this paper are presented as follows, assuming that $P_1 \leq P_2 \leq \ldots \leq P_n$. These lower bounds were proposed by Dell'Amico and Martello [13]:

$$LB_1 = \left\lceil \frac{\sum_{j=1}^{n} P_j}{m} \right\rceil \quad (1)$$

$$LB_2 = P_n \quad (2)$$

$$LB_3 = P_{(n-m)} + P_{(n-m+1)} \quad (3)$$

The best lower bound is the maximum value among them. Therefore,

$$LB = max(LB_1,\ LB_2,\ LB_3) \quad (4)$$

3.2. Upper Bound

In this paper, the variable neighborhood search (*VNS*) was proposed as an upper bound for the arc flow graph. The *VNS* is detailed in the following subsection.

3.2.1. Variable Neighborhood Search

The variable neighborhood search (*VNS*) is a metaheuristic approach proposed by Mladenovi'c and Hansen [38] that performs systematic neighborhood structures to improve the quality of an existing solution. To start the *VNS* algorithm, an initialization phase is required. In the initialization phase, an initial solution "*S*" is obtained (randomly or by using a heuristic method), neighborhood structures "$N_k(S)$", $k = 1, 2, ..., k_{max}$ are designed, where k is the index of a neighborhood and k_{max} is the maximum number of neighborhoods, and the stopping criteria are determined.

The Initial Solution

In this work, we proposed the longest processing time (*LPT*) rule as the initial solution for the *VNS* algorithm. The *LPT* rule works by first sorting the jobs in non-increasing order of their processing times and assigning the jobs to the smallest available machine until all jobs have been assigned.

The Neighborhood Structures

The neighborhood structures are methods designed to enhance the local search in the *VNS* algorithm. To design the neighborhood structures for an identical parallel machine scheduling problem with a makespan objective, two terminologies will be used:

- Problem machine (*Pm*): a machine in which the total scheduling time is the makespan.
- Non-problem machine (*NPm*): a machine in which the total scheduling time is less than the makespan.

The five neighborhood structures used in this work are explained as follows:

1. Move (one): move a job i from a problem machine (*Pm*) to a non-problem machine (*NPm*) if (makespan of *Pm* (C_{Pm})-makespan of *NPm* (C_{Pm}) > processing time of job i (p_i));
2. Exchange (one–one): exchange a job i from a (*Pm*) with a job j from a (*NPm*) if ($p_i - p_j > 0$) and ($C_{Pm} - C_{NPm} > p_i - p_j$);
3. Exchange (two–one): exchange two jobs, i and j, from a (*Pm*) with a job k from a (*NPm*) if ($p_i + p_j - p_k > 0$) and ($C_{Pm} - C_{NPm} > p_i + p_j - p_k$);
4. Exchange (one–two): exchange a job from a (*Pm*) i with two jobs, j and k, from a (*NPm*) if ($P_i - (p_j + p_k) > 0$) and ($C_{Pm} - C_{NPm} > p_i - (p_j + p_k)$;
5. Exchange (two–two): exchange two jobs, i and j, from a (*Pm*) with two jobs, k and l, from a (*NPm*) if ($p_i + p_j - (p_k + p_l) > 0$) and ($C_{Pm} - C_{NPm} > p_i + p_j - (p_k + p_l)$.

To illustrate how the neighborhood structures work, consider six jobs to be scheduled in two identical parallel machines. The processing times in minutes of each job are shown in Table 1.

Table 1. Processing times of jobs for neighborhood structures example.

Job No.	Processing Time (Minutes)
1	10
2	8
3	5
4	3
5	2
6	1

Let us consider the initial solution with C_{max} = 19 obtained by the schedule shown in Figure 2. From Equation (4), $LB = max(LB_1,\ LB_2,\ LB_3) = max\left(\left\lceil\frac{29}{2}\right\rceil,\ 10,\ 5+8\right) = 15$. The problem machine (*Pm*) is Machine 1, while Machine 2 is the non-problem machine (*NPm*). It is worth noting that the selection of jobs for each neighborhood structure was selected arbitrarily in this example just to illustrate how these neighborhood structures work.

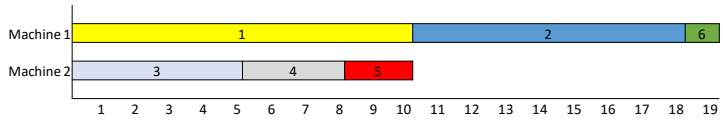

Figure 2. The initial schedule of the illustrative example.

Since the condition (makespan of *Pm* (1)-makespan of *NPm* (2) > p_6) is satisfied, then applying the "Move (one)" neighborhood structure to move Job 6 from *Pm* (1) to *NPm* (2) yields C_{max} = 18 as illustrated in Figure 3.

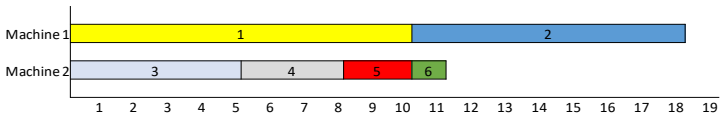

Figure 3. The schedule after applying the "Move (one)" neighborhood structure.

Now, the conditions ($p_2 - p_5 > 0$) and ($C_{Pm} - C_{NPm} > p_2 - p_5$) are satisfied. Applying the "Exchange (one–one)" neighborhood structure results in obtaining C_{max} = 17 with the schedule illustrated in Figure 4. In this schedule, Machine 2 becomes the *Pm* machine and Machine 1 is the *NPm* machine.

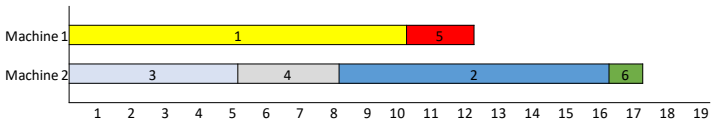

Figure 4. The schedule after applying the "Exchange (one–one)" neighborhood structure.

With the conditions ($p_4 + p_2 - p_1 > 0$) and ($C_{Pm} - CN_{Pm} > p_4 + p_2 - p_1$) being satisfied, the "Exchange (two–one)" neighborhood structure is applied by exchanging Jobs 4 and 2 from *Pm* (2) with Job 1 from *NPm* (1). The resulting schedule with C_{max} = 16 is shown in Figure 5.

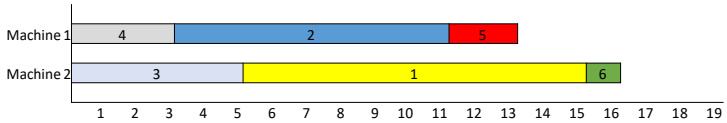

Figure 5. The schedule after applying the "Exchange (two–one)" neighborhood structure.

Note that the "Exchange (one–two)" neighborhood structure is similar to the previous one, with the difference being that one job from *Pm* is exchanged with two jobs from *NPm*. Here, the conditions are not satisfied for the schedule in Figure 5.

Finally, the fifth neighborhood structure, "Exchange (two–two)", is applied by exchanging Jobs 1 and 6 from *Pm* (2) with Jobs 2 and 5 from *NPm* (1). The obtained schedule with C_{max} = 15 is shown in Figure 6. Note that this is the optimal makespan, since it equals *LB*.

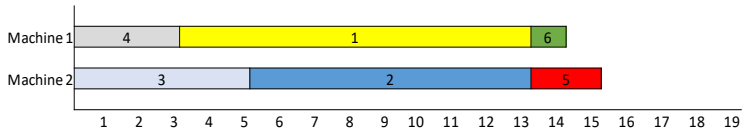

Figure 6. The optimal schedule after applying the "Exchange (two–two)" neighborhood structure.

Steps of Variable Neighborhood Search Algorithm

The VNS algorithm is detailed by Algorithm 1.

Algorithm 1: VNS algorithm.

```
Obtain an initial Solution.
Find the set of problem machines S_Pm and the set of non-problem machines S_NPm.
Initialize iteration = 0, Max_iterations = 1, PmTime = makespan;
if (makespan ! = LB):
while (makespan ! = LB && iteration< Max_iterations)
        r = 0, q = 0;
        while (r < size of SPm && makespan ! = LB)
                while (q < size of SNPm && makespan != LB)
                        SPm1 = SPm[r], SNPm1 = SNPm[q], k = 0;
                        while (k< kmax)
                                reduced = 0;
                                switch(k):
case (0): Move (One); break;
case (1): Exchange (One-One); break;
case (2): Exchange (Two-One); break;
case (3): Exchange (one-Two); break;
case (4): Exchange (Two-Two); break;
                                end switch
                                if makespan is enhanced in any neighborhood structure:
                                        reducd = 1, Update PmTime;
                                if(reduced = 1)
                                 Update (PmTime, S_Pm and S_NPm) Makespan= PmTime;
                                        if (size of SNPm > 0 && makespan != LB)
                                                r = 0, q = 0, k = 0, SPm1 = SPm[r], SNPm1 = SNPm[q];
                                        else break;
                                else k = k + 1;
                        end while
                        q = q + 1;
                end while
                r = r + 1;
        end while
        iteration = iteration + 1;
end while
end if
```

It is worth noting that the maximum number of iterations in the *VNS* algorithm is important only when generating the initial solution randomly. Therefore, at each iteration, a random schedule is generated and neighborhood structures are applied. The final solution is the best makespan among all iterations. On the other hand, obtaining the initial solution using a heuristic such as the *LPT* rule will have no effect as the number of iterations increases. Therefore, only one iteration is needed in the proposed *VNS* algorithm.

3.3. The Improved Arc Flow Model

In this section, we will show how to construct a graph for the proposed improved arc flow model, give a numerical example to present the impact of the proposed model on reducing the number of variables compared to the arc flow model proposed by Mrad and Souayah [29], and formulate the studied problem based on the improved arc flow model.

3.3.1. Graph Building

The arc flow model for the $P_m||C_{max}$ problem was constructed by finding h disjoint paths for each job on a graph initialized from the starting time at node 0 up to the ending time at node T, where: $0 \leq T \leq UB$. Consider a graph G with vertices V and arcs A, $G = (V, A)$,x where $V = \{0, \ldots, UB\}$ with the notation v representing a general index of each vertex and A to represent the combination between the arcs of jobs A_n, where $n = \{1, \ldots, N\}$ and the loss arc is A_o $(A = A_n \cup A_o)$. The loss arc is an arc connecting each vertex (except vertices 0 and UB) and the UB vertex. Each arc is defined by three terms: an initial node s, a final node d, and a job n representing this arc such that:

$$A_n = \{(s,\ d, n) : 0 < s < d < UB,\ d - s = P_n\}\ and\ A_o = \{(v, UB, -1\} : \forall v \in V \backslash \{0, UB)\}$$

where s and d represent, respectively, the initial and final nodes of the job arc A_n.

The set of arcs for each job n is constructed as follows: a directed arc is created between two vertices s and d if there is a job n with process time $P_n = d - s$, given that d $\leq UB$ and $s < d$. This set, i.e., A_n, can be reduced by applying some breaking symmetry rules [29]:

1. Jobs are sorted in decreasing order of their processing times.
2. An arc of a job n can't be created from a node that is added from the arc of the same job.
3. Arcs for each job only start from the created nodes of previously considered jobs, assuming that node 0 is created at the beginning of the graph-building process.

In addition to the above-listed breaking symmetry rules, a graph compression technique was proposed that prevents some arcs of jobs from being created from node 0. The idea behind this technique is based on the fact that, by knowing the makespan of some scheduled jobs on one machine, we can find the lower bound of the remaining jobs on the remaining $(m - 1)$ machines, where m represents the total number of machines. If the resulting lower bound exceeds the upper bound, these jobs will not have arcs from node 0. Instead, the arcs will be created from the first node created after node 0. Another graph compression technique was used for reducing the number of loss arcs. A loss arc is only needed for the last job in each machine schedule so that the flow conservation is satisfied. Therefore, only a loss arc is created from a node v that satisfied $\lceil (\sum_{n \in N} P_n - v)/(m-1) \rceil \leq UB$. The graph compression algorithm for the arcs of jobs is illustrated in Algorithm 2.

Algorithm 2: Graph compression algorithm for job arcs.

1. Initialize: *removedJobs*←0, *SumOfProcessTimes*←0
2. Sort jobs in increasing order of processing times.
3. **for** $n \in N$:
   SumOfProcessTimes+ = p_n
   if($\lceil (\sum_{n \in N} p_n - SumOfProcessTimes)/(m-1) \rceil > UB$
   removedJobs++;
   else break;
4. Return JobsSelection = NumOfJobs-removedJobs;

The selected number of jobs from Algorithm 2 was used to build the graph from node 0 while the remaining jobs were used to build the graph from the next node after node 0, as illustrated in Algorithm 3.

Algorithm 3: Graph building of the improved arc flow model.

Initialize: V [0, . . . ,UB]←0, $JobArcs$←0, $lossArcs$←0, $NewV$←∅, A_n←∅, and A_o←∅.
V [0]←1
for $n \in N$: (n = 0; n < $JobsSelection$; n++);
$NewNodes$←∅
for $v \in V$:
if($V[v]$ = 1 && ($v + p_n \leq UB$))
A_n←$A_n \cup (v, v + p_n, n)$
$NewNodes$←$NewNodes \cup (v + p_n)$
$JobArcs$++
end if
end for
for $v \in NewNodes$
$V[NewNodes[v]]$ = 1
end for
for $n \in N$: (n = $JobsSelection$; n < N; n++)
$NewNodes$←∅
for $v \in V \backslash \{node\ 0\}$:
if($V[v]$ = 1 && ($v + p_n \leq UB$))
A_n←$A_n \cup (v, v + p_n, n)$
$NewNodes$←$NewNodes \cup (v + p_n)$
end if
end for
for $v \in NewNodes$
$V[NewNodes[v]]$ = 1
end for
for $v \in V$:
if($V[v]$ = 1)
$NewV$←$NewV \cup V[v]$
for $v \in NewV \backslash \{0, UB\}$:
if($\lceil (\sum_{n \in N} p_n - NewV[v]) / (m-1) \rceil \leq UB$)
A_o←$A_o \cup (NewV[v], UB, -1)$
$lossArcs$++;
end for
Return $NewV$, A_n, A_o

3.3.2. Numerical Example

Consider five jobs to be scheduled in two identical parallel machines. The processing times in minutes of each job are shown in Table 2.

Table 2. Processing times of jobs for graph-building example.

Job No.	Processing Time (Minutes)
1	10
2	8
3	5
4	3
5	2

Let LB = 14 min using Equation (4) and UB = 15 min obtained by both the *LPT* and *VNS* algorithms. The arc flow graph built using the algorithm proposed by [29] gives 11 nodes, 15 job arcs, and 9 loss arcs. Algorithm 2 can then be applied to check whether any job arcs can be reduced. Starting from the job with the least processing time, i.e., Job 5, the new lower bound is checked when one machine has only Job 5.

$$LB_{new}\ (\text{job } 5) = \lceil (\sum_{n \in N} p_n - SumOfProcessTimes)/(m-1) \rceil = \lceil (28-2)/(2-1) \rceil = 28 \text{ min} > UB. \text{ Remove Job 5.}$$

$$LB_{new}\ (\text{jobs 5 and 4}) = \lceil ((28-(2+3))/(2-1) \rceil = 23 \text{ min} > UB. \text{ Remove Job 4.}$$

$$LB_{new}\ (\text{jobs 5, 4, and 3}) = \lceil ((28-(2+3+5))/(2-1) \rceil = 18 \text{ min} > UB. \text{ Remove Job 3.}$$

$$LB_{new}\ (\text{jobs 5, 4, 3, and 2}) = \lceil ((28-(2+3+5+8))/(2-1) \rceil = 10 \text{ min} < UB \text{ stop;}$$

Therefore, Jobs 3, 4, and 5 will not have arcs created from node 0.

The graph compression for loss arcs, when applied, yields only one loss arc flow from node 13 that satisfies $\lceil (\sum_{n \in N} p_n - NewV[j])/(m-1) \rceil \leq UB \Rightarrow \left\lceil \frac{28-13}{2-1} \right\rceil = 15 \text{ min} = UB$.

The arc flow network, as proposed in [29], is shown in Figure 7. The resulting improved arc flow network with seven nodes, nine job arcs, and one loss arc is shown in Figure 8. The optimal solution is shown in Figure 9.

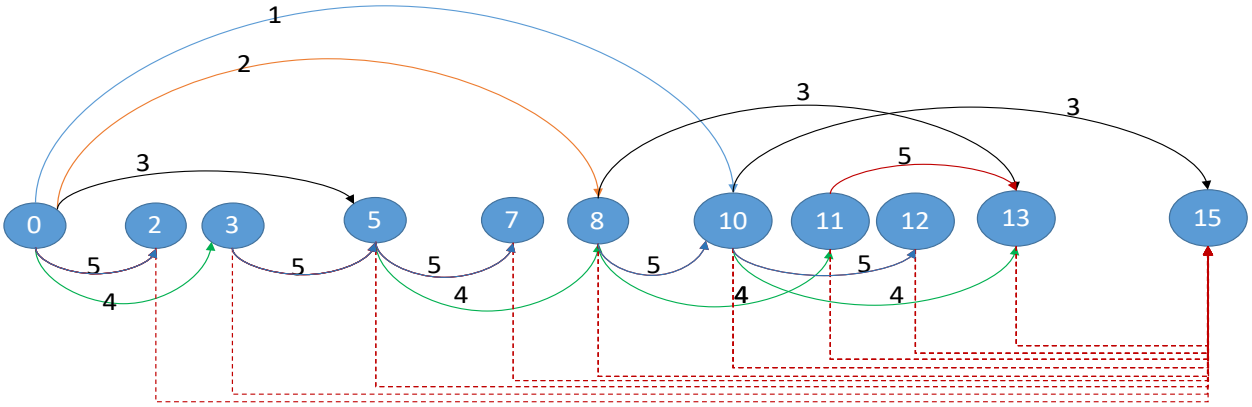

Figure 7. Arc flow network.

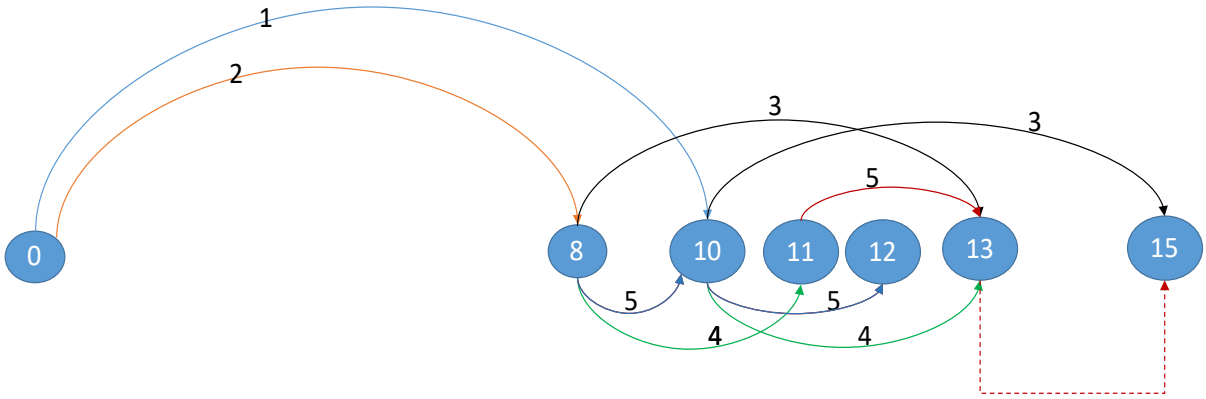

Figure 8. Improved arc flow network.

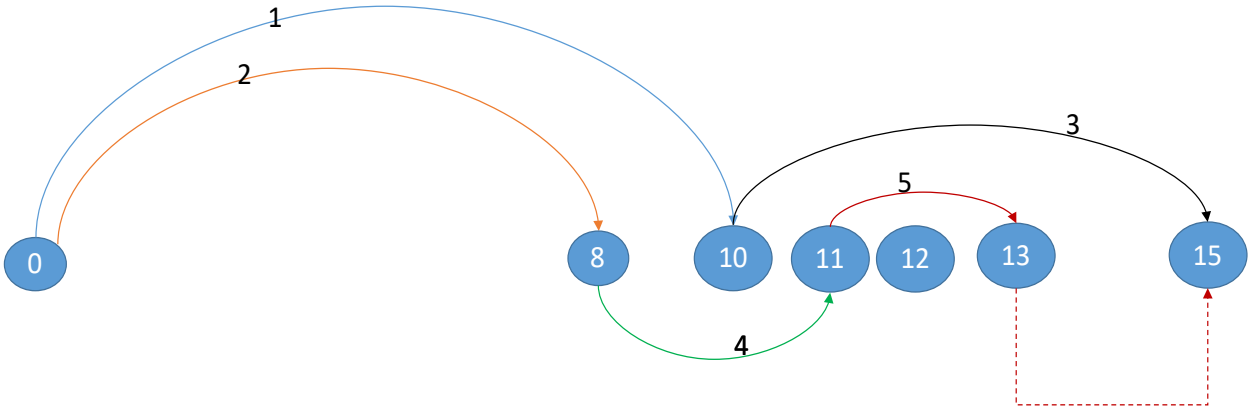

Figure 9. Optimal solution.

3.3.3. Mathematical Formulation of the Improved Arc Flow Model

This section presents a mathematical formulation for the improved arc flow model in which h independent paths containing all jobs should be selected to minimize the makespan.

Parameters:

A_n: set of job n arcs: $\forall n \in N$.

A_o: set of loss arcs.

A: set of all arcs: $A = A_n \cup A_o$.

V: set of nodes.

Decision Variables:

$$x_{sd} = \begin{cases} 1 \; if \; arc \; (s,d) \; is \; selected \\ 0 \; otherwise \end{cases} \forall s,d \in A \backslash \{A_o\}$$

$$x_{sd} \in \{0,1,\ldots,m\} \; \forall s,d \in A_o$$

$$C_{max} : makespan \; of \; the \; final \; schedule$$

Objective Function:

$$minimize \; C_{max} \tag{5}$$

Constraints:

$$C_{max} \geq d * x_{sd} \;\; \forall (s,d) \in A \backslash \{A_o\} \;\&\; d \geq LB \tag{6}$$

$$\sum_{(0,d) \in A \backslash \{A_o\}} x_{0d} = m \tag{7}$$

$$\sum_{(v,d) \in A} x_{vd} - \sum_{(s,v) \in A} x_{sv} = 0 \quad \forall v \in V \backslash \{0, UB\} \;\&\; d = s \tag{8}$$

$$\sum_{(s,d) \in A_n} x_{sd} = 1 \;\; \forall n \in N \tag{9}$$

$$x_{sd} \in \{0,1\} \; \forall (s,d) \in A \backslash \{A_o\} \tag{10}$$

$$x_{sd} \in \{0,\ldots,m\} \; \forall (s,d) \in A_o \tag{11}$$

$$LB \leq C_{max} \leq UB \tag{12}$$

The objective in Equation (5) minimizes the makespan. The makespan is the value of Constraint (6) that corresponds to the maximum final node of the job arc selected, given that it is greater than or equal to the lower bound. Constraint (7) means that the total number of arcs leaving node 0 must be equal to the number of machines, since each connected arc in the final solution represents a schedule of the jobs in one machine. The flow conservation constraints are presented in Equation (8) to ensure that for the intermediate nodes, if an arc enters a node, it must leave that node. Constraint (9) ensures one arc for each job. Constraints (10) and (11) are binary and integer decision variables, respectively. The binary

decision variables are for the job arcs, while the integer decision variables are for the loss arcs and are limited to m. Finally, Constraint (12) illustrates the bounds of the makespan.

4. Computational Results and Discussion

The proposed improved arc flow model was coded in C++ language and the mathematical model was solved using Cplex 12.10. The proposed model was run on an Intel® Core i7-4930 k CPU @3.40 GHz and 34.0 GB of RAM. The improved arc flow mode was compared with the arc flow model proposed by Mrad and Souayah [29]. For a fair comparison, Mrad and Souayah's algorithm [29] was run on the same PC. Both algorithms used the same *LPT* rule. However, the algorithm proposed by Mrad and Souayah [29] used *LPT* as a *UB* for their arc flow model. On the other hand, the *LPT* rule in the proposed improved arc flow model was used as an initial solution for the *VNS* algorithm. All algorithms had a time limit per instance = 1200 s. We will first mention the benchmark instances used in this paper, and then present the computational results.

4.1. Benchmark Instances

In this paper, we considered the benchmark instances proposed by Mrad and Souayah [29] for the $P||C_{max}$ problem. The benchmark instances were set by first considering the type of instances based on the ratio n/m, which has the following values: $n/m \in \{2, 2.25, 2.5, 2.75, and\ 3\}$. Then, for each ratio, the processing times of the jobs were generated based on seven types of classes. Class 1, Class 2, and Class 3 were generated using a discrete uniform distribution with the intervals [1100], [2100], and [50,100], respectively. Class 4 and Class 5 both were generated using a uniform distribution with a mean $\mu = 100$, whereas the standard deviations were $\sigma = 20$ for Class 4 and $\sigma = 50$ for Class 5. Class 6 was generated using a discrete uniform distribution in the interval $[n, 4n]$ and Class 7 was generated using a normal distribution with $\mu = 4n$ and $\sigma = n$. In each type of ratio, each class had 10 combinations of n jobs and m machines. Each combination had 10 instances. Therefore, the total number of generated instances was 3500 instances. The combination of n and m is shown along with the results in Tables 3 and 4.

Table 3. Comparison of the mean CPU times (s) on Classes 1–4.

n/m	*n*	*m*	**Class 1**		**Class 2**		**Class 3**		**Class 4**	
			AF	***IAF***	***AF***	***IAF***	***AF***	***IAF***	***AF***	***IAF***
2	20	10	0.0771	0.03526	0.0392	0.02603	0.0208	0.01761	0.0249	0.02234
	40	20	0.1737	0.05529	0.0924	0.03867	0.031	0.02814	0.0491	0.04469
	60	30	0.2219	0.06117	0.1332	0.05604	0.0359	0.03502	0.0739	0.0896
	80	40	0.2901	0.10273	0.1821	0.08742	0.0536	0.05234	0.0974	0.08841
	100	50	0.3773	0.11568	0.2228	0.1126	0.0585	0.04419	0.1159	0.08476
	120	60	0.6823	0.17723	0.3848	0.19154	0.0687	0.05955	0.3592	0.29095
	140	70	0.7539	0.20155	0.3492	0.15197	0.0697	0.0609	0.385	0.23621
	160	80	0.9325	0.20272	0.3482	0.16256	0.0828	0.06271	0.2658	0.17787
	180	90	1.3566	0.32176	0.5148	0.20159	0.0838	0.06604	0.667	0.39902
	200	100	1.3098	0.27153	0.6568	0.24949	0.0902	0.0799	0.6158	0.30558
2.25	36	16	0.1362	0.03315	0.2356	0.07377	0.2605	0.04179	0.4068	0.06536
	54	24	0.3236	0.10993	0.4543	0.14164	0.6416	0.08264	1.1268	0.10604
	72	32	0.4944	0.12799	0.7636	0.20798	1.1344	0.09529	2.4926	0.15411
	90	40	0.6538	0.15185	1.5459	0.29656	1.7575	0.13245	4.7855	0.28571
	108	48	1.1464	0.30887	1.5738	0.39998	2.5898	0.15727	6.3198	0.29995
	126	56	1.1979	0.31386	1.4974	0.36307	3.701	0.23519	9.6255	0.51936
	144	64	1.6843	0.29565	2.0626	0.51169	3.5738	0.26678	10.4568	0.51346
	162	72	3.3008	0.70363	1.4588	0.41291	5.7881	0.29463	12.1321	0.66862
	180	80	2.4148	0.53003	2.9578	0.62703	6.0689	0.27774	15.0448	0.87539
	198	88	3.4346	0.58915	4.2044	0.90267	5.8534	0.36493	20.1808	0.9274

Table 3. *Cont.*

n/m	*n*	*m*	Class 1		Class 2		Class 3		Class 4	
			AF	*IAF*	*AF*	*IAF*	*AF*	*IAF*	*AF*	*IAF*
2.5	20	8	0.1083	0.0385	0.1152	0.04067	0.1106	0.03566	0.1261	0.04121
	40	16	0.2846	0.09865	0.3873	0.14398	0.4902	0.07143	0.7562	0.118
	60	24	0.3197	0.13935	0.4997	0.19241	1.1325	0.13782	1.981	0.17063
	80	32	0.761	0.15574	0.945	0.39747	1.2863	0.17443	4.8875	0.32915
	100	40	1.5134	0.25792	0.9268	0.32406	2.5409	0.25719	7.3425	0.5099
	120	48	1.0613	0.33994	1.1129	0.40538	2.5391	0.32216	8.3755	0.65916
	140	56	1.6769	0.43768	1.2984	0.58982	4.4313	0.47202	15.584	0.88034
	160	64	1.6984	0.50926	1.7829	0.63383	4.4785	0.49164	22.5773	1.24152
	180	72	2.0898	0.69955	1.6739	0.72796	5.7358	0.77288	21.9303	1.50267
	200	80	2.9188	1.0436	2.389	0.91475	6.2676	0.82127	26.0193	1.36693
2.75	22	8	0.1746	0.06549	0.1118	0.04953	0.1503	0.10234	0.1498	0.05319
	44	16	0.2987	0.13958	0.2941	0.16233	0.4824	0.12948	0.9315	0.13893
	66	24	0.458	0.18898	0.5029	0.21454	1.2847	0.24273	2.1105	0.31473
	88	32	0.9481	0.32052	0.7255	0.34517	2.3522	0.32662	4.0638	0.68388
	110	40	1.3911	0.44476	1.3694	0.58596	2.7157	0.46807	5.6202	0.92024
	132	48	2.007	0.77728	2.1578	0.67103	3.7053	0.58596	9.8738	1.63563
	154	56	3.0137	0.94141	2.2004	1.02719	4.9362	0.92185	18.9603	1.60869
	176	64	2.5642	0.9183	2.8724	1.26501	8.2793	1.03136	26.5827	1.83095
	198	72	4.4431	1.33729	2.9442	1.22386	11.5653	1.22666	39.6504	2.86454
	220	80	4.0233	1.3861	3.7405	1.58992	12.9586	1.41616	40.1836	3.00184
3	36	12	0.2643	0.15999	0.2042	0.1107	0.1376	0.05944	0.2147	0.07534
	54	18	0.3655	0.19001	0.372	0.23032	0.2106	0.13473	0.3618	0.188
	72	24	0.8163	0.35061	0.8154	0.37059	0.3485	0.20355	0.6852	0.38274
	90	30	1.2901	0.48275	0.9756	0.49174	0.5079	0.33681	1.3291	0.53297
	108	36	1.8092	0.80422	1.3209	0.82615	0.6825	0.46908	1.6426	0.82015
	126	42	2.3945	1.03214	1.9296	1.1105	0.9414	0.4378	2.5098	1.1539
	144	48	2.5131	1.09889	2.3889	1.10151	1.2057	0.74086	3.1758	1.44368
	162	54	2.8894	1.53295	3.8152	1.54957	1.4903	0.83249	3.8759	2.3585
	180	60	4.8238	1.83409	3.909	1.71357	2.0983	1.16229	4.321	1.96692
	198	66	4.8045	1.66324	3.1346	2.21478	2.617	1.03554	6.4585	2.29203

4.2. Comparison of the Arc Flow Model and the Improved Arc Flow Model

In this section, a comparison between the arc flow model developed by Mrad and Souayah [29] and the proposed improved arc flow model was conducted. The results of the mean CPU times in seconds for both models are shown in two tables. Table 3 presents a comparison of the two models for Classes 1–4, whereas Table 4 presents a comparison for Classes 5–7.

The results in Table 3 show that for $n/m = 2$, the arc flow (*AF*) model obtained the optimal solution in less than one second for almost all of the instances (except the last two (n, m) combinations of Class 1. On the other hand, the improved arc flow (*IAF*) model optimally solved all these instances in less than 0.40 s. The results suggest that instances with a ratio of $n/m = 2$ seem to be the easiest ones. Indeed, for $n/m = 2.25$, the mean CPU time for the arc flow model gradually increased, especially for some classes and some combinations of n and m. For instance, the mean CPU time reached 20.1808 s for Class 4 with $n = 198$ and $m = 88$. In contrast, the improved arc flow model still found the optimal solution for all classes in less than one second. Instances with ratios of 2.5 and 2.75 seem to be the hardest ones, where the maximum recorded CPU times are observed for both the arc flow model (40.1836 s) and the improved arc flow model (3.00184 s).

Table 4. Comparison of the mean CPU times (s) on Classes 5–7.

n/*m*	*n*	*m*	Class 5		Class 6		Class 7	
			AF	*IAF*	*AF*	*IAF*	*AF*	*IAF*
2	20	10	0.0546	0.04656	0.0269	0.02201	0.0262	0.02078
	40	20	0.1705	0.06297	0.0815	0.05031	0.0536	0.05235
	60	30	0.4372	0.1362	0.1665	0.13898	0.1202	0.11593
	80	40	0.3268	0.07655	0.3842	0.19209	0.3982	0.1859
	100	50	0.8312	0.16459	0.4343	0.168	0.7655	0.39649
	120	60	0.7307	0.11953	1.1057	0.43703	1.1212	0.59539
	140	70	1.1903	0.16661	2.3826	1.00759	2.0292	1.05471
	160	80	1.0261	0.16348	2.0445	0.55197	1.8262	0.96158
	180	90	1.5475	0.24215	3.0562	1.22568	3.3475	1.27347
	200	100	0.7404	0.20512	8.9567	1.84374	8.5372	1.73166
2.25	36	16	0.4131	0.1387	0.4715	0.11412	0.5134	0.07636
	54	24	0.6209	0.28156	1.3551	0.22126	2.7366	0.16857
	72	32	1.7974	0.56845	4.3433	0.39424	8.3424	0.31553
	90	40	3.383	1.20387	13.1277	0.82299	29.066	0.58036
	108	48	2.9051	0.65903	30.982	1.80474	71.9907	1.33954
	126	56	8.5506	1.50027	53.108	4.90213	139.6497	2.30687
	144	64	9.3202	2.22944	80.9374	8.66699	236.505	7.65391
	162	72	10.0352	1.44473	167.0883	14.16798	332.8585	8.8986
	180	80	12.4277	1.67685	273.3291	41.52442	510.0146	19.45461
	198	88	5.0249	1.15768	447.1832	48.11236	646.0028	38.52895
2.5	20	8	0.1538	0.0684	0.11	0.03498	0.1349	0.03906
	40	16	0.7204	0.27853	0.7472	0.16238	1.2275	0.12669
	60	24	2.1967	0.57895	2.4699	0.5594	6.6195	0.43938
	80	32	2.4471	0.85103	9.2258	1.96191	23.2269	1.24437
	100	40	3.8316	0.9225	31.2072	4.67202	55.3321	3.96598
	120	48	8.6085	1.21499	63.2689	15.33955	120.1153	9.11037
	140	56	6.8704	2.14134	103.8499	21.15889	278.7284	12.64758
	160	64	10.0015	5.03549	159.5447	48.12724	603.3516	20.61922
	180	72	14.5461	2.63308	649.3818	86.00928	1072.787	70.51515
	200	80	7.5844	2.28975	731.0712	135.807	1066.674	80.6796
2.75	22	8	0.2144	0.11063	0.1243	0.04628	0.1512	0.07119
	44	16	0.7747	0.32776	0.985	0.28238	1.6435	0.27984
	66	24	1.0913	0.43548	2.6566	0.74888	10.0207	1.5892
	88	32	3.7268	0.97732	6.8621	1.64574	50.0149	4.71296
	110	40	3.8373	1.25362	16.7783	4.02252	150.8236	12.19673
	132	48	8.0449	2.39961	43.7506	10.56909	556.608	40.22264
	154	56	9.2885	1.73566	397.3662	28.1359	1110.385	86.52077
	176	64	10.6577	2.67341	535.7069	57.68455	1183.907	133.7526
	198	72	7.821	4.75334	862.2826	69.34135	1200	252.22
	220	80	11.6044	5.52537	984.8733	120.0307	1200	377.1867
3	36	12	0.5453	0.22471	0.3122	0.14289	0.399	0.1651
	54	18	1.4733	0.52422	0.8638	0.62028	1.4451	0.55498
	72	24	1.9235	0.68507	2.6791	1.45521	5.7497	1.59318
	90	30	3.1898	1.18902	5.9322	2.45864	9.0364	3.58461
	108	36	4.9936	1.75473	13.2389	5.31357	26.4002	7.36371
	126	42	5.9733	2.93804	52.8808	9.54336	25.9556	17.25403
	144	48	9.3373	3.83546	45.7367	20.37307	269.8698	21.57858
	162	54	12.0203	4.61985	249.3259	29.50285	765.1593	52.96896
	180	60	10.536	5.94832	297.5118	37.64389	528.8696	93.47038
	198	66	12.5762	6.14091	840.8658	108.2606	1007.825	164.7304

Overall, the results in Table 3 show that, although both models obtained the optimal solution within the time limit, the improved arc flow model clearly outperformed the arc flow model in terms of the required CPU time.

From Table 4, we observed that Class 5 can be considered the simplest class compared to Classes 6 and 7. Indeed, all instances of this class were solved to optimality in a relatively small time for both models. The maximum CPU time was 14.5461 s and 6.14091 s for the arc flow model and the improved arc flow model, respectively. Classes 6 and 7 showed a dramatic outperformance of our proposed approach with respect to the arc flow model. Indeed, the improved arc flow (*IAF*) model was, on average, 5.68 and 12.70 times faster than the arc flow (*AF*) one on Classes 6 and 7, respectively. This ratio could reach as much as 60.53 times for *Class* 7 with n = 126 and m = 56. Moreover, we observed that, for two combinations of n and m (n = 198, m = 72 and n = 220, m = 80) in *Class* 7, the arc flow model (*AF*) reached the maximum time limit of 1200 s without finding the optimal solution for any of the considered 20 instances. On the other hand, the proposed improved arc flow model (*IAF*) was able to solve 19 of these instances to optimality within an average CPU time of 252.22 s and 377.1867 s, respectively.

To present overall insight into the performance of the proposed improved arc flow model, Table 5 shows the total time of the arc flow model [29], the total time of the proposed improved arc flow algorithm, and the percentage of improvement (% *improvement*) in the total mean CPU time for each type of ratio (n/m). The % *improvement* calculated how much savings in time were obtained by the proposed improved arc flow model compared with the arc flow model [29] and was calculated as follows:

$$\% \, improvement = 100 \times \frac{(Time_{AF} - Time_{IAF})}{Time_{AF}} \tag{13}$$

Table 5. Comparison of the total mean CPU time (s).

n/m	*AF*	*IAF*	*% Improvement*
2	562.67	184.78	67.16
2.25	32,495.649	2243.8063	93.09
2.5	51,981.90	5477.01	89.46
2.75	86,054.30	12,546.10	85.42
3	42,882.7581	6419.338	85.03
Total Time	213,977.2771	26,871.0343	87.44

A positive % *improvement* indicated that the proposed improved arc flow model obtained a lower total CPU time compared with the arc flow model [29].

The results in Table 5 show that the *IAF* algorithm considerably reduced the total time for all types of ratios. The percentage of time reduction ranged from 67.16% up to 93.09% with the overall percentage of reduction in time reaching 87.44%. Figure 10 shows a visual representation of the total CPU times of both algorithms.

To know how many instances out of 10 were not solved to optimality for both algorithms, Table 6 illustrates the number of unsolved instances for the combinations of (n, m) in Classes 6 and 7, i.e., instances where the solver could not reach the optimum solution within the time limit (1200 s). The instances of the other (n, m) combinations for Classes 6 and 7 are not shown in Table 6 since an optimal solution was found for both models.

The results show that Class 6 and Class 7 with the ratios (n/m) = 2.5, 2.75, and 3 were the hardest instances for the *AF* model [29]. The total number of unsolved instances for Class 6 and Class 7 in all the ratios was 19 and 58 instances, respectively. Therefore, a total of 77 instances out of 3500 instances were not solved to optimality by the *AF* model [29], with a percentage of 97.80% solved instances. On the other hand, only one instance in Class 7 of the ratio (n/m) = 2.75 with a combination (n = 220, m = 80) was not solved to optimality by the proposed improved arc flow model. The percentage of solved instances was 99.97% with an increase of 2.17% in the solved instances compared with the *AF* model [29].

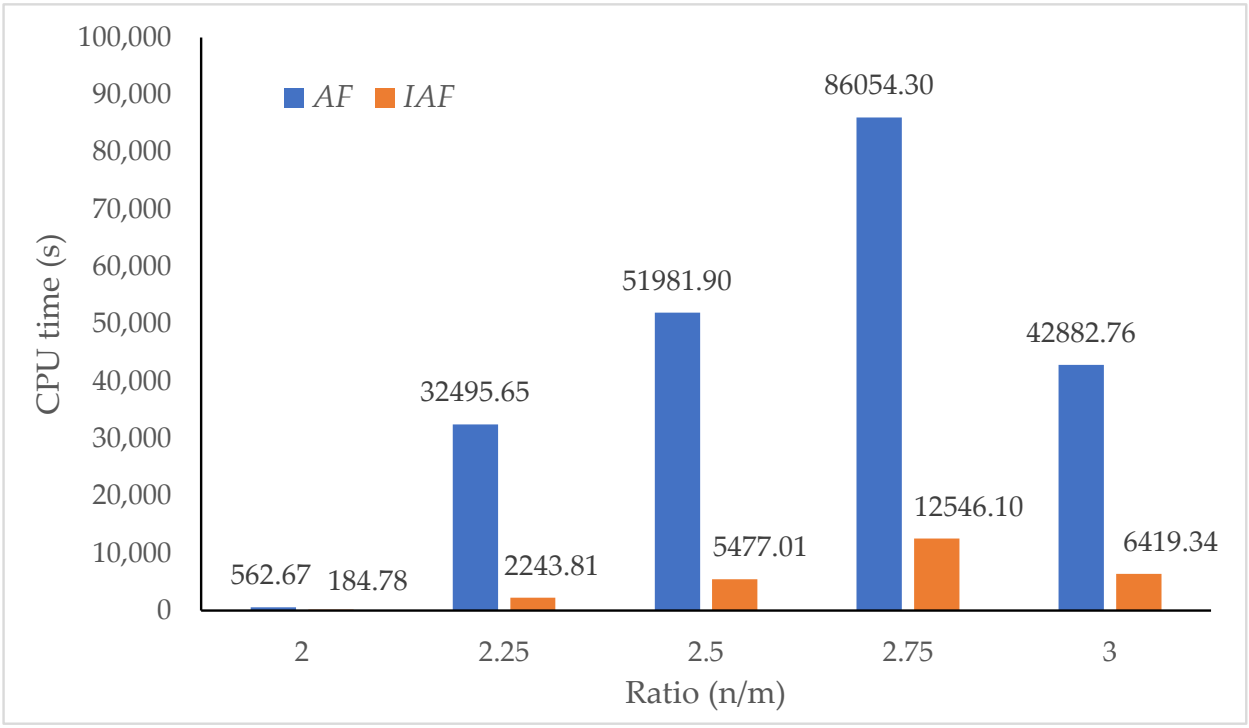

Figure 10. Bar chart for total CPU time comparison (s).

Table 6. Number of unsolved instances.

n/m	*n*	*m*	*Class 6*		*Class 7*	
			AF	*IAF*	*AF*	*IAF*
2.5	180	72	3	0	5	0
	200	80	3	0	6	0
2.75	154	56	0	0	7	0
	176	64	1	0	9	0
	198	72	5	0	10	0
	220	80	4	0	10	1
3	162	54	0	0	2	0
	180	60	0	0	3	0
	198	66	3	0	6	0

5. Conclusions and Future Work

The scheduling of identical parallel machine scheduling problems to minimize the makespan was studied in this paper. An improved arc flow model with a mathematical formulation was proposed for this problem. Enhanced upper and lower bounds were utilized in the arc flow model to improve its efficiency. A variable neighborhood search was proposed as an upper bound that started with an initial solution obtained using the longest processing time rule, and five neighborhood structures were used to improve the local search. A graph compression technique that prevented some jobs from appearing together on the same machine was proposed to reduce the size of the graph and therefore reduce the number of variables in the mathematical model. The proposed improved arc flow model was compared with an arc flow model from the literature. The computational results on benchmark instances showed that the improved arc flow model outperformed the arc flow model from the literature at reducing the total time needed to solve instances, with the overall reduction in time reaching 87%. In addition, the number of solved instances to optimality increased to 99.97% of the instances.

Future work should investigate other graph compression techniques for this scheduling problem and apply the improved arc flow model in other scheduling environments, such as unrelated parallel machine scheduling.

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 work was funded by the National Plan for Science, Technology and Innovation (MAARIFAH), King Abdulaziz City for Science and Technology, Kingdom of Saudi Arabia, award number 13-MAT1544-02.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

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

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

References

1. Morozoff, E. Parallel machine scheduling problems: A survey. *Asia-Pac. J. Oper. Res.* **2001**, *18*, 193.
2. Michael, L.P. *Scheduling: Theory, Algorithms, and Systems*; Springer: Berlin/Heidelberg, Germany, 2018.
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. Gary, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; WH Freeman and Company: New York, NY, USA, 1979.
5. Graham, R.L. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math.* **1969**, *17*, 416–429. [CrossRef]
6. della Croce, F.; Scatamacchia, R. The longest processing time rule for identical parallel machines revisited. *J. Sched.* **2020**, *23*, 163–176. [CrossRef]
7. Coffman, J.; Edward, G.; Garey, M.R.; Johnson, D.S. An application of bin-packing to multiprocessor scheduling. *SIAM J. Comput.* **1978**, *7*, 1–17. [CrossRef]
8. Lee, C.-Y.; Massey, J.D. Multiprocessor scheduling: Combining LPT and MULTIFIT. *Discret. Appl. Math.* **1988**, *20*, 233–242. [CrossRef]
9. Gupta, J.N.; Ruiz-Torres, A.J. A LISTFIT heuristic for minimizing makespan on identical parallel machines. *Prod. Plan. Control* **2001**, *12*, 28–36. [CrossRef]
10. Min, L.; Cheng, W. A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artif. Intell. Eng.* **1999**, *13*, 399–403. [CrossRef]
11. Lee, W.-C.; Wu, C.-C.; Chen, P. A simulated annealing approach to makespan minimization on identical parallel machines. *Int. J. Adv. Manuf. Technol.* **2006**, *31*, 328–334. [CrossRef]
12. 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]
13. Dell'Amico, M.; Martello, S. Optimal scheduling of tasks on identical parallel processors. *ORSA J. Comput.* **1995**, *7*, 191–200. [CrossRef]
14. Mokotoff, E. An exact algorithm for the identical parallel machine scheduling problem. *Eur. J. Oper. Res.* **2004**, *152*, 758–769. [CrossRef]
15. Dell'Amico, M.; Martello, S. A note on exact algorithms for the identical parallel machine scheduling problem. *Eur. J. Oper. Res.* **2005**, *160*, 576–578. [CrossRef]
16. Dell'Amico, M.; Iori, M.; Martello, S.; Monaci, M. Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS J. Comput.* **2008**, *20*, 333–344. [CrossRef]
17. Haouari, M.; Gharbi, A.; Jemmali, M. Tight bounds for the identical parallel machine scheduling problem. *Int. Trans. Oper. Res.* **2006**, *13*, 529–548. [CrossRef]
18. Haouari, M.; Jemmali, M. Tight bounds for the identical parallel machine-scheduling problem: Part II. *Int. Trans. Oper. Res.* **2008**, *15*, 19–34. [CrossRef]
19. de Carvalho, J.V. Exact solution of cutting stock problems using column generation and branch-and-bound. *Int. Trans. Oper. Res.* **1998**, *5*, 35–44. [CrossRef]
20. de Carvalho, J.V. Exact solution of bin-packing problems using column generation and branch-and-bound. *Ann. Oper. Res.* **1999**, *86*, 629–659. [CrossRef]

21. Brandao, F.; Pedroso, J.P. Bin packing and related problems: General arc-flow formulation with graph compression. *Comput. Oper. Res.* **2016**, *69*, 56–67. [CrossRef]
22. Brandao, F.D.A. Cutting & Packing Problems: General Arc-Flow Formulation with Graph Compression. Ph.D. Thesis, Universidade do Porto, Porto, Portugal, 2017.
23. Delorme, M.; Iori, M.; Martello, S. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *Eur. J. Oper. Res.* **2016**, *255*, 1–20. [CrossRef]
24. Martinovic, J.; Scheithauer, G.; de Carvalho, J.V. A comparative study of the arcflow model and the one-cut model for one-dimensional cutting stock problems. *Eur. J. Oper. Res.* **2018**, *266*, 458–471. [CrossRef]
25. Mrad, M.; Ali, T.G.; Balma, A.; Gharbi, A.; Samhan, A.; Louly, M. The Two-Dimensional Strip Cutting Problem: Improved Results on Real-World Instances. *Eurasia Proc. Educ. Soc. Sci.* **2021**, *22*, 1–10. [CrossRef]
26. Kramer, A.; Lalla-Ruiz, E.; Iori, M.; Voß, S. Novel formulations and modeling enhancements for the dynamic berth allocation problem. *Eur. J. Oper. Res.* **2019**, *278*, 170–185. [CrossRef]
27. Macedo, R.; Alves, C.; de Carvalho, J.V.; Clautiaux, F.; Hanafi, S. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *Eur. J. Oper. Res.* **2011**, *214*, 536–545. [CrossRef]
28. Kramer, R.; Iori, M.; Vidal, T. Mathematical models and search algorithms for the capacitated-center problem. *INFORMS J. Comput.* **2020**, *32*, 444–460. [CrossRef]
29. Mrad, M.; Souayah, N. An arc-flow model for the makespan minimization problem on identical parallel machines. *IEEE Access* **2018**, *6*, 5300–5307. [CrossRef]
30. Kramer, A.; Dell'Amico, M.; Iori, M. Enhanced arc-flow formulations to minimize weighted completion time on identical parallel machines. *Eur. J. Oper. Res.* **2019**, *275*, 67–79. [CrossRef]
31. Kramer, A.; Dell'Amico, M.; Feillet, D.; Iori, M. Scheduling jobs with release dates on identical parallel machines by minimizing the total weighted completion time. *Comput. Oper. Res.* **2020**, *123*, 105018. [CrossRef]
32. Wang, S.; Wu, R.; Chu, F.; Yu, J.; Liu, X. An improved formulation and efficient heuristics for the discrete parallel-machine makespan ScheLoc problem. *Comput. Ind. Eng.* **2020**, *140*, 106238. [CrossRef]
33. Trindade, R.S.; de Araújo, O.C.; Fampa, M. Arc-flow approach for parallel batch processing machine scheduling with non-identical job sizes. In *International Symposium on Combinatorial Optimization*; Springer: Warsaw, Poland, 2020; pp. 179–190.
34. Trindade, R.S.; de Araújo, O.C.B.; Fampa, M. Arc-flow approach for single batch-processing machine scheduling. *Comput. Oper. Res.* **2021**, *134*, 105394. [CrossRef]
35. Kramer, A.; Iori, M.; Lacomme, P. Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization. *Eur. J. Oper. Res.* **2021**, *289*, 825–840. [CrossRef]
36. de Lima, V.L.; Alves, C.; Clautiaux, F.; Iori, M.; de Carvalho, J.M.V. Arc flow formulations based on dynamic programming: Theoretical foundations and applications. *Eur. J. Oper. Res.* **2022**, *296*, 3–21. [CrossRef]
37. de Lima, V.L.; Iori, M.; Miyazawa, F.K. Exact solution of network flow models with strong relaxations. *Math. Program.* **2022**, 1–34. [CrossRef]
38. Mladenović, N.; Hansen, P. Variable neighborhood search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100. [CrossRef]