Received November 20, 2017, accepted December 20, 2017, date of publication January 8, 2018, date of current version February 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2789678

An Arc-Flow Model for the Makespan Minimization Problem on Identical Parallel Machines

MEHDI MRAD[1] **AND NIZAR SOUAYAH**[2]
[1]Department of Industrial Engineering, College of Engineering, King Saud University, Riyadh 11451, Saudi Arabia
[2]Department of Natural Sciences, Community College, King Saud University, Riyadh 11451, Saudi Arabia

Corresponding author: Nizar Souayah (nsouayah@ksu.edu.sa)

This was supported by the Deanship of Scientific Research at King Saud University through the Research Group Project under Project RG-1438-079.

ABSTRACT In this paper, we consider the basic makespan minimization problem on identical parallel machines. The aim of this study is to solve, to optimality the *hard* instances of the literature. We present an integer linear program based on an innovative arc-flow model inspired from the duality between the bin packing problem and the parallel-machine scheduling problem. The proposed mathematical model is tested on a large set of hard instances. The results of computational experiments attest the efficiency of the new approach and prove that it outperforms the existing ones. In addition to its efficiency, the proposed method is simple and easy to use.

INDEX TERMS Arc-flow model, integer programming, parallel machines, scheduling.

I. INTRODUCTION

In this study, we solve the identical parallel-machine scheduling problem with the objective of minimizing the makespan. This basic problem can be described as follows: a set J of n jobs has to be scheduled on m identical parallel machines (with $n > m \geq 2$). Each job $j \in J$ has to be processed for p_j units of time. Each machine processes at most one job at a time and the preemption is not allowed. This problem, denoted by $P\|C_{max}$ using the classification introduced by Graham *et al.* [1] is *NP*-hard in the strong sense [2].

This fundamental problem has been massively treated in the literature. Due to its NP-hardness, only some attempts are made to solve the $P\|C_{max}$ problem with exact methods. Mokotoff [17] builds an exact algorithm based on cutting plane techniques. The algorithm essentially consists of iterative additions of valid inequalities, starting from the solution obtained by successive linear programming relaxations. Moreover, the experimental results show that the suggested approach is also a powerful tool for producing fast high quality feasible solutions. Later, Dell'Amico and Martello [18] propose a branch-and-bound algorithm that turns out to outperform the exact algorithm by Mokotoff [17]. Their branch-and-bound algorithm is based on sophisticated lower and upper bounds and some dominance rules.

Dell'Amico *et al.* [19] present an exact algorithm based on a branch-and-price scheme using the dual relationship with the bin packing problem. In fact, they solve a sequence of feasibility test of bin packing problems while iteratively updating the size of the used bins. The experimental results on a large set of benchmark instances show the effectiveness of the proposed approach. Haouari *et al.* [20], develop a bounding stratgy based on lifting procedures. They derive lower bounds that outperform existing bounds for the $P\|C_{max}$ problem. Also, they suggest two heuristics based on subset-sum problem (SSP). Motivated by the obtained results, Haouari and Jemmali [21] propose an exact branch-and-bound algorithm. The lower bounds used are the bounds developed in [20] and improved by the SSP-based enhancement procedure in order to obtain stronger ones. They propose an improved variant of SSP-based heuristic that requires solving a sequence of 0-1 knapsack problems. The developed branch-and-bound algorithm includes a new branching scheme and proves to be able to solve the quasi-totality of benchmark instances in the literature. While testing the performance of their algorithm, the authors identify a class of hard instances for which $\frac{n}{m} = 2.5$. Their proposed branch-and-bound fails to solve 32% of these *hard* instances. Also, it is shown that for $n \geq 30$, the performance of the algorithm deteriorates as the number of unsolved instances becomes relatively high.

Recently, Walter and Lawrinenko [22] have focused on solving the hard instances identified by Haouari and Jemmali [21]. They derive a new branch-and-bound algorithm based mainly on restricting the solution space using the so-called path-related dominance rules which represent the principal contribution of their paper. Indeed, the authors buid their branch-and-bound using classical lower and upper bounds in the literature; especially those proposed by [20] and [21]. They test their algorithm on a large set of instances suggested by Dell'Amico and Martello [18]. They consider 5 classes of uniform and non-uniform distributions. As identified by Haouari and Jemmali [21] the hard instances depend on the ratios of $\frac{n}{m}$. The computational results show that the proposed method solves instances with at most 33 jobs in case of $\frac{n}{m} \in \{2, 2.25, 2.5, 2.75\}$. In order to test the performance and the limitation of the approach, the authors propose two new particularly hard classes of instances. The experimental studies show that the algorithm fails to solve 22% of 480 tested instances (with $n \leq 100$) in the two new classes.

Lately, some researchers treated the considered problem under some constraints [4], [5], [7].

On the other hand, heuristic methods and approximate solutions have been intensively attempted. Riera *et al.* [3] propose several heuristic algorithms based on the list-scheduling strategies. The obtained results are close to the optimal solutions given by the dynamic programming algorithm proposed by Blazewicz [6] and also close to the bound of McNaughton [8]. Mokotoff [9] proposes an approximation algorithm based on linear programming formulations using cutting planes algorithm. This algorithm essentially consists of iterative computations of lower bounds of C_{max} using the successive linear programming relaxations. Lee *et al.* [10] propose a simulated annealing (SA) approach to solve the $P\|C_{max}$ problem. The computational results show that the SA heuristic is more efficient than other heuristics encountered in the literature. Mokotoff *et al.* [11] present several heuristics using list scheduling algorithms. They propose new assignment rules that begin with the Longest Processing Time (LPT) rule. The main contribution is to determine how to balance, in the assignment procedure, between the LPT rule and the new rules. Brueggemann *et al.* [12] propose an approximative solution inspired from the idea of local search and based on new improvement assignment rule called *move-optimal* assignment. This method outperforms Graham's LPT-algorithm [13]. Chiaselotti *et al.* [14] present a new $nlog(n)$ iterative algorithm with worst-case performance ratio. The proposed algorithm combines partial solutions which are obtained by partitioning the set of jobs into suitable families of subsets. Laha [15] presents an improved simulated annealing heuristic. The experimentation proves that the obtained results are better than those produced by Lee *et al.* [10] as well as other existing heuristics for large-size instances. Tang and Luo [16] propose, for the $P\|C_{max}$ problem, an iterated local search (ILS) algorithm obtained from an integer programming model combined with a variable number of cyclic exchanges.

According to this brief bibliographic study, it is clear that the problem of minimization of makespan on identical parallel machines has been extensively investigated especially for approximate solutions.

Despite the efforts of Haouari and Jemmali [21] and Walter and Lawrinenko [22] challenging instances of the literature are still unsolved. The aim of this paper is to present a new approach to solve exactly this *challenging* class of instances. Therefore, we propose a new efficient integer linear program based on an arc-flow model. Our approach is also based on the duality between the parallel machine scheduling problem and the bin packing problem. In fact, each job can be considered as an item with size equal to p_j $(j = 1, \ldots, n)$ and that must be packed into a limited number of identical bins (machines) of capacity C. If a feasible solution of this problem exists, this means that C is an upper bound of the $P\|C_{max}$ problem. Hence, the bin packing problem can be treated as a parallel machine scheduling problem aiming to minimize the number of required machines and not exceeding a pre-specified makespan C.

This duality was adopted by many researchers as an alternative strategy to solve the $P\|C_{max}$ problem. Krause *et al.* [23] are among the first researchers applying this approach in 1975. They introduce an algorithm named first-fit-increase (FFI) in order to obtain a bound of the problem. Hochbaum and Shmoys [24] prove through this duality that finding an approximation algorithm for the minimum makespan problem can be reduced to the problem of finding a dual approximation algorithm for the bin-packing problem. Alvim and Ribeiro [25] propose a hybrid bin-packing-based heuristic for the multiprocessor scheduling problem to minimize the makespan. Dell'Amico *et al.* [19] suggest an exact algorithm for $P\|C_{max}$ based on the dual relation with the bin packing problem as discussed in Section 1. We refer to Cheng and Sin [26], for more general surveys on the bin packing approach for the parallel machine scheduling problem.

The remainder of this paper is organized as follows. In Section II, we introduce the integer linear program based on the arc-flow model. We describe the upper bound used to build the proposed graph and we provide a detailed explanation of the graph structure and the different reduction criteria applied to reduce the graph size. In section III, we report the details of the implementation and the computational results. Finally, in section IV we provide some concluding remarks and perspectives.

II. SOLUTION APPROACH AND MATHEMATICAL PROGRAM

The proposed method in this paper consists on solving an integer linear program based on an-arc flow model. This simple approach compared with the sophisticated methods of the literature exploits the duality between the bin packing and the $P\|C_{max}$ problems. Indeed, we show that it is possible to solve exactly the $P\|C_{max}$ problem in a single shot while using this duality. Whereas, the best approach aiming to solve exactly the $P\|C_{max}$ based on the duality with the bin packing problem considers a sequence of feasibility test problems [19]. This concept has been proved to be effective in the literature [28], [29], [30], [31], [32], [33].

Let $J = \{1, \dots, n\}$ be the set of n jobs with positive processing times $p_1, p_2, \dots, p_n$, to be scheduled on m $(m < n)$ identical parallel machines in order to minimize the makespan.

A. GRAPH STRUCTURE

Let UB be an upper bound of the $P\|C_{max}$ problem. Determining a valid solution to parallel machines problem can be modelled as just finding m disjoint paths in an acyclic directed graph with $UB+1$ vertices. Consider a graph $G = (V, A)$ with $V = \{0, \dots, UB\}$ and $A = A' \cup A''$, where:

$A' = \cup_{k \in J} A_k$ and A_k is the set of arcs representing the item k.

$A_k = \{(i, j) : 0 < i < j \leq UB, j - i = p_k\}$ and

$A'' = \{(j, UB) : \forall j \in V \setminus \{UB\}\}$.

The set A' is constructed as follows: It exists a directed arc between two vertices i and j if there is a job with size $j - i$. To reduce computational efforts and solve larger instances, the set A' can be reduced, by using some breaking symmetry rules:

(i) The jobs are ordered according to a decreasing order of their processing times.

(ii) An arc of size p_t can only have its head at a vertex j which is the tail of another arc of size p_k, for $p_k > p_t$, or at node 0.

The pseudo code that builds the graph may be presented as follow:

Algorithm 1 Graph Construction
```
V ⟵ {0, UB}
for $i \in J$ do
        SetofNewNodes ⟵ $\emptyset$
    for $j \in V \setminus \{UB\}$ do
        if $(j + p_i \leq UB)$ then
            $A' \longleftarrow A' \cup (j, j + p_i)$
            $A_i \longleftarrow A_i \cup (j, j + p_i)$
          SetofNewNodes ⟵ SetofNewNodes $\cup (j + p_i)$
        end if
    end for
          $V \longleftarrow V \cup$ SetofNewNodes
end for
$A'' = \emptyset$
for $j \in V \setminus \{UB\}$ do
    $A'' = A'' \cup \{j, UB\}$
end for
```

The set A'' is considered as the set of virtual arcs that ensure the existence of the path between any node in the graph and the node UB.

The reduction of the set A' will eventually reduce the size of V which will include only nodes that are head or tails of arcs in A' . $V = \{0, UB\} \cup \{j / \exists (i, j) \in A'\}$. Since each arc in A'' is related to a node in $V \setminus \{UB\}$, the reduction of V will certainly result in the reduction of A''.

B. DESCRIPTION OF THE UPPER BOUND UB

There is a large literature on the upper bounds of the makespan minimization on parallel machines. We use an easy heuristic delivered by the LPT (Longest Processing Time) rule. This heuristic consists of two steps:

- Order the jobs by the non-increasing processing time.
- Assign them on the earliest available machine.

C. EXAMPLE

We consider a set of four jobs to be scheduled on two machines. Let $J = \{1, 2, 3, 4\}$, $p_1 = 5$, $p_2 = 3$, $p_3 = 3$ and $p_4 = 2$. Let's take for example $UB = 6$.

Accordingly, to the Algorithm 1, the arcs related to each job will be appended to the set of the arcs A'. Figure 1 will display the configuration of the graph at each iteration of the algorithm.

D. MATHEMATICAL MODEL

The scheduling problem considered in this paper is formulated as the problem of determining m disjoint paths between vertex 0 and vertex UB covering all the jobs.

Notations: Let us consider the decision variables x_{ij} defined as follows:

$$x_{ij} = \begin{cases} 1 & \text{if the arc } (i, j) \text{ is considered in the solution} \\ & \qquad\qquad\qquad \forall (i, j) \in A'. \\ 0 & \text{otherwise.} \end{cases}$$

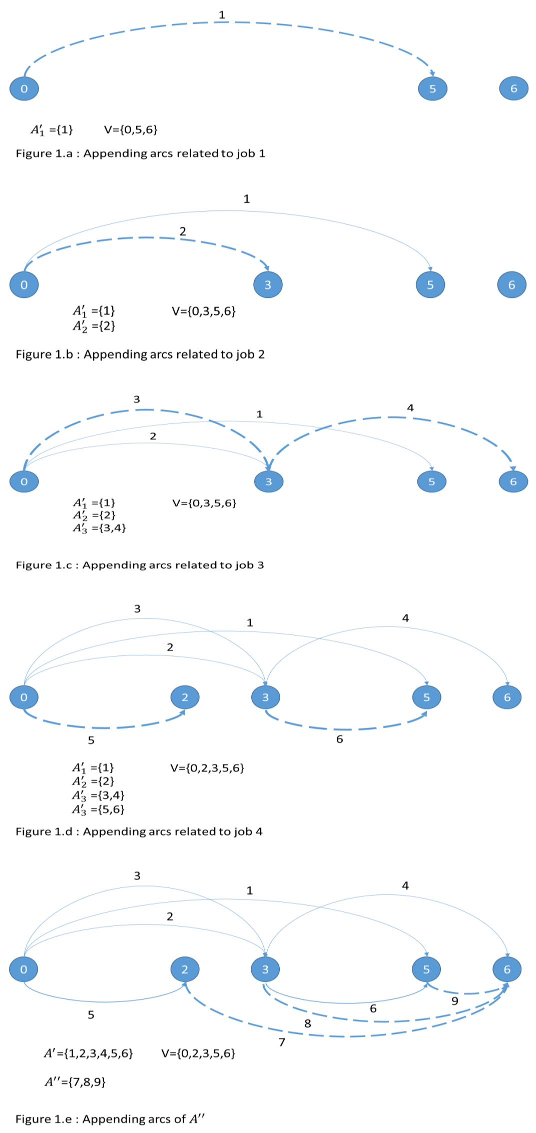
$x_{ij} \in \{0, \dots, m\} \forall (i, j) \in A''$.

z is the variable indicating the makespan.

Using the notations introduced above, the resulting model is given by:

$$\text{minimize } z \tag{1}$$
$$\text{subject to } z \geq j x_{ij} \quad \forall (i, j) \in A' \tag{2}$$
$$\sum_{(0,j) \in A} x_{0,j} = m \tag{3}$$
$$\sum_{(i,j) \in A} x_{ij} - \sum_{(j,i) \in A} x_{ji} = 0 \quad \forall j \in V \setminus \{0, UB\} \tag{4}$$
$$\sum_{(i,j) \in A_k} x_{ij} = 1 \quad \forall k \in J \tag{5}$$
$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in A' \tag{6}$$
$$x_{ij} \in \mathbb{N} \quad \forall (i, j) \in A'' \tag{7}$$
$$z \geq 0. \tag{8}$$

The objective function (1) minimizes the makespan. The set of constraints (2) ensures that the maximum completion time is larger or equal than the completion time of each job.



A'₁ ={1} V={0,5,6}
Figure 1.a : Appending arcs related to job 1
A'₁ ={1} A'₂ ={2} V={0,3,5,6}
Figure 1.b : Appending arcs related to job 2
A'₁ ={1} A'₂ ={2} A'₃ ={3,4} V={0,3,5,6}
Figure 1.c : Appending arcs related to job 3
A'₁ ={1} A'₂ ={2} A'₃ ={3,4} A'₃ ={5,6} V={0,2,3,5,6}
Figure 1.d : Appending arcs related to job 4
A'={1,2,3,4,5,6} V={0,2,3,5,6}
A''={7,8,9}
Figure 1.e : Appending arcs of A''

FIGURE 1. The graph building.

Constraints (3) imply that the number of paths in the solution is m indicating that there are m machines to be used. Constraints (4) represent the flow conservation constraints. Constraints (5) state that, each job $k \in J$ should appear once in the final solution. Constraints (6), (7) and (8) define the value range of the variables.

Generating a lower bound (LB) for the $P\|C_{max}$ problem can enhance the proposed integer linear program. In fact the constraint (2) can be replaced by:

$$z \geq jx_{ij} \quad \forall (i,j) \in A'; j \geq LB. \tag{9}$$

in this case, constraint (8) is to be replaced by:

$$LB \leq z \leq UB. \tag{10}$$

A straightforward lower bound that will be adopted is: $LB = \lceil \frac{\sum p_j}{m} \rceil$.

TABLE 1. Comparison between Haouari and Jemmali algorithm and the proposed arc flow model on the hard instances.

		H&J algorithm		Arc flow model	
n	m	*Time*	*NS*	*Time*	*NS*
20	8	1.131	0	0.009	0
30	12	9.307	0	0.026	0
40	16	3.429	2	0.044	0
50	20	68.945	7	0.083	0
60	24	1.928	7	0.088	0
70	28	2.070	10	0.168	0
80	32	3.023	8	0.264	0
90	36	4.008	11	0.521	0
100	40	5.252	6	0.456	0
150	60	26.994	9	1.495	0
200	80	20.146	12	7.801	0

III. COMPUTATIONAL RESULTS

In order to test performance of the proposed method, we have coded our algorithm in C++ language and have run it on an Intel Core i7-2600 (3.4 GHz) and 16 GB RAM while running Windows 7 Professional (64-bit). Additionally, we have used CPLEX 12.6 in concert technology.

The tests were done on benchmark instances from the literature and a large set of new hard instances generated according to the method used in [22]. We note that this type of instances was introduced first by Haouari and Jemmali [21].

A. DATA GENERATION

The first type of hard instances was identified by Haouari and Jemmali in [21]. For these instances, the ratio n/m is 2.5. The instances are generated as follows:

- The processing times are drawn from discrete uniform distribution on $[\frac{n}{5}, \frac{n}{2}]$
- For each value of $n \in \{20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200\}$, a set of 20 instances were randomly generated. Hence, the total number of Haouari and Jemmali instances is 220.

Later on, Walter and Lawrinenko [22] focused on this type of instances and extended it to obtain a large set of hard instances where the ratio n/m has the following values: $n/m \in \{2, 2.25, 2.5, 2.75\}$. In this paper we consider also $n/m = 3$. We indicate that the generated values refer to the processing times. For each type of these hard instances, 7 classes are proposed in [22]. Thus, we generate the processing times according to the following distributions:

- Class 1: discrete uniform distribution in [1, 100]
- Class 2: discrete uniform distribution in [20, 100]
- Class 3: discrete uniform distribution in [50, 100]
- Class 4: normal distribution with $\mu = 100$ and $\sigma = 20$
- Class 5: normal distribution with $\mu = 100$ and $\sigma = 50$
- Class 6: discrete uniform distribution in $[n, 4n]$
- Class 7: normal distribution with $\mu = 4n$ and $\sigma = n$

Each class includes 10 pairs (n, m), with:

- $n \in \{20, 40, 60, 80, 100, 120, 140, 160, 180, 200\}$ if $n/m \in \{2, 2.5\}$
- $n \in \{36, 54, 72, 90, 108, 126, 144, 162, 180, 198\}$ if $n/m \in \{2.25, 3\}$

TABLE 2. Results for ratio $n/m = 2$.

		Class 1		Class 2		Class 3		Class 4		Class 5		Class 6		Class 7	
		Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS
n	**m**														
20	10	0.050	0	0.045	0	0.006	0	0.007	0	0.091	0	0.036	0	0.031	0
40	20	0.171	0	0.130	0	0.016	0	0.053	0	0.255	0	0.078	0	0.066	0
60	30	0.325	0	0.182	0	0.033	0	0.093	0	0.500	0	0.250	0	0.182	0
80	40	0.379	0	0.228	0	0.040	0	0.125	0	0.523	0	0.477	0	0.657	0
100	50	0.443	0	0.214	0	0.065	0	0.159	0	0.754	0	0.511	0	1.021	0
120	60	0.746	0	0.454	0	0.080	0	0.448	0	1.475	0	1.212	0	2.140	0
140	70	0.898	0	0.462	0	0.074	0	0.500	0	1.409	0	4.166	0	4.736	0
160	80	2.091	0	0.245	0	0.045	0	0.198	0	1.155	0	1.859	0	2.566	0
180	90	0.638	0	0.245	0	0.045	0	0.198	0	1.609	0	11.475	0	4.948	0
200	100	0.638	0	0.245	0	0.045	0	0.198	0	1.119	0	39.636	0	12.457	0

TABLE 3. Results for ratio $n/m = 2.25$.

		Class 1		Class 2		Class 3		Class 4		Class 5		Class 6		Class 7	
		Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS
n	**m**														
36	16	0.315	0	0.389	0	0.426	0	1.102	0	0.772	0	0.742	0	1.671	0
54	24	0.452	0	0.929	0	0.937	0	2.434	0	3.529	0	2.400	0	7.707	0
72	32	0.703	0	1.170	0	1.547	0	4.766	0	6.902	0	7.258	0	38.635	0
90	40	0.916	0	2.553	0	2.2142	0	8.830	0	14.713	0	26.206	0	89.791	0
108	48	1.762	0	2.631	0	3.374	0	14.594	0	11.400	0	67.549	0	149.387	0
126	56	3.257	0	2.820	0	4.127	0	18.363	0	17.778	0	95.114	0	315.837	0
144	64	5.031	0	2.493	0	4.861	0	26.828	0	17.742	0	235.465	0	456.235	1
162	72	4.875	0	4.594	0	7.074	0	44.261	0	6.198	0	357.455	0	843.496	1
180	80	4.435	0	4.079	0	8.546	0	41.998	0	9.683	0	515.585	0	1082.984	8
198	88	4.838	0	4.113	0	10.486	0	47.964	0	22.166	0	479.452	2	> 1200	10

TABLE 4. Results for ratio $n/m = 2.5$.

		Class 1		Class 2		Class 3		Class 4		Class 5		Class 6		Class 7	
		Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS
n	**m**														
20	8	0.152	0	0.1573	0	0.143	0	0.187	0	0.336	0	0.120	0	0.239	0
40	16	0.458	0	0.550	0	0.698	0	1.396	0	2.496	0	1.174	0	2.290	0
60	24	0.545	0	0.659	0	1.624	0	3.711	0	7.453	0	5.892	0	19.475	0
80	32	1.179	0	1.098	0	2.358	0	8.227	0	9.281	0	17.570	0	80.944	0
100	40	1.371	0	1.254	0	3.189	0	14.182	0	7.687	0	57.461	0	163.582	0
120	48	1.159	0	1.382	0	3.583	0	22.387	0	7.044	0	147.636	0	332.801	0
140	56	2.051	0	2.355	0	6.240	0	34.862	0	8.527	0	246.233	1	624.906	1
160	64	2.001	0	1.685	0	8.491	0	46.229	0	13.861	0	654.573	2	893.458	6
180	72	2.500	0	1.837	0	9.409	0	63.741	0	16.376	0	415.254	6	962.345	9
200	80	2.6899	0	2.288	0	12.403	0	83.346	0	9.061	0	547.553	8	> 1200	10

- $n \in \{22, 44, 66, 88, 110, 132, 154, 176, 198, 220\}$ if $n/m \in \{2.75\}$.

For each investigated (n, m) combination, there are 10 instances. So the total number of new generated instances is 3500.

B. RESULTS

In order to test the effectiveness of our approach, we run our algorithm and the branch and bound algorithm of Haouari and Jemmali in [21] on the 220 challenging instances for the ratio $n/m = 2.5$. We indicate that for the comparison, we use the same code introduced in [21]. The summary of the results is presented in Table 1. In this table, H&J algorithm refeers to the algorithm in [21]. In addition, (*Time*) is the mean CPU time in seconds computed over 20 instances and (*NS*) represents the number of unsolved instances. The results reveal that our algorithm is distinctly more efficient than the branch and bound proposed in [21]. Indeed, all the 220 instances are solved while in [21] 72 instances were not solved. In addition to its ability to solve all instances our approach proves to be much more efficeint in terms of computational time than that in [21]. In fact, Table 1 shows that the proposed arc-flow model solves most of the instances in less than one second. Further, it is up to 830 times faster than the one in [21].

As in [22] we have tested the 3500 instances described in section III-A. The results are displayed in Tables 2-6. In these tables, we have provided for each combination (n, m) and for

TABLE 5. Results for ratio $n/m = 2.75$.

		Class 1		Class 2		Class 3		Class 4		Class 5		Class 6		Class 7	
		Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS
n	**m**														
22	8	0.207	0	0.151	0	0.224	0	0.313	0	0.644	0	0.156	0	0.262	0
44	16	0.478	0	0.441	0	0.879	0	2.024	0	1.726	0	1.264	0	4.010	0
66	24	0.532	0	0.565	0	1.552	0	4.656	0	2.645	0	2.7715	0	29.054	0
88	32	1.018	0	0.941	0	2.910	0	13.591	0	8.871	0	10.351	0	127.175	0
110	40	1.597	0	1.260	0	3.232	0	16.748	0	13.707	0	53.121	0	347.371	0
132	48	1.825	0	1.628	0	5.830	0	38.402	0	19.736	0	72.502	0	690.861	0
154	56	3.412	0	2.363	0	11.701	0	46.670	0	12.865	0	316.627	1	1064.960	7
176	64	2.966	0	2.866	0	18.195	0	48.930	0	23.764	0	71.297	6	> 1200	10
198	72	3.443	0	2.634	0	22.411	0	48.397	0	79.901	0	65.620	7	> 1200	10
220	80	4.103	0	3.892	0	26.127	0	65.258	0	68.800	0	191.032	9	> 1200	10

TABLE 6. Results for ratio $n/m = 3$.

		Class 1		Class 2		Class 3		Class 4		Class 5		Class 6		Class 7	
		Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS	Time	NS
n	**m**														
36	12	0.290	0	0.401	0	0.146	0	0.483	0	1.435	0	0.466	0	0.848	0
54	18	0.572	0	0.457	0	0.313	0	0.600	0	1.474	0	1.143	0	2.074	0
72	24	1.355	0	0.865	0	0.414	0	0.948	0	3.970	0	4.439	0	4.786	0
90	30	1.745	0	1.230	0	0.589	0	1.664	0	6.782	0	17.973	0	74.858	0
108	36	2.100	0	1.477	0	0.880	0	2.246	0	11.707	0	22.373	0	298.235	0
126	42	2.600	0	2.009	0	1.082	0	2.924	0	21.671	0	50.010	0	148.721	0
144	48	3.664	0	2.443	0	1.260	0	6.711	0	24.765	0	86.159	1	64.669	4
162	54	4.021	0	2.449	0	2.181	0	4.269	0	25.349	0	196.007	2	96.742	5
180	60	5.695	0	4.235	0	2.177	0	7.319	0	31.799	0	122.694	3	191.510	6
198	66	8.584	0	4.591	0	2.722	0	6.882	0	65.997	0	428.627	7	511.740	9

each class the mean CPU time in seconds (*Time*) computed over 10 instances and the number of unsolved instances (*NS*) where the maximal computation time per instance was set to 1200 seconds.

We see from Table 2, associated to the ratio $n/m = 2$, that all instances have been solved to optimality. Moreover, most of the instances (530 out of 700) were solved in less that one second. In addition, for the pair (200, 100), the mean computation time is less than 8 seconds. At this point, it is necessary to mention that the method proposed in [22] solved only instances with 30 jobs for classes 1-5 and failed to solve many instances with 80 and 100 jobs for classes 6-7.

Furthermore, regarding the ratio $n/m = 2.25$ (Table 3), the proposed method solved all the instances for classes 1-5, and did not solve some instances of classes 6 and 7 where $n \geq 144$. For this type of instances, [22] was able to solve only instances with 27 jobs.

The same remarks can be observed for the ratios $n/m = 2.5$ and $n/m = 2.75$ (Table 4 and Table 5). In fact, our algorithm solved all the instances for classes 1-5. In addition, in classes 6 and 7, the proposed approach is able to solve all instances with $n \leq 120$, while in [22] they did not solve 99 instances over 240 where the maximum number of jobs does not exceed 80 jobs.

These computational results on the set of hard instances with ratios $n/m \in \{2, 2.25, 2.5, 2.75\}$ attest the effectiveness of the proposed algorithm. On the other hand, we observe that the hardness of the instances is proportional to the ratio n/m mainly for the classes 6 and 7 and when the number of jobs is large. In order to explore and identify the hard instances more accurately, we have generated a new type of instances with ratio $n/m = 3$. The results obtained for this type are displayed in Table 6. Notice that, as for the previous types of ratios $n/m = 2 - 2.75$, all the instances of classes 1-5 are solved. Moreover, only 18.5% of instances of classes 6 and 7 of ratio $n/m = 3$ are unsolved. These results show that the instances with ratio $n/m = 3$ are easier to solve than those with ratios 2.5 and 2.75.

Table 7 summarizes the number of unsolved instances over the entire test bed of 3500 instances. For each class, we give the percentage of unsolved instances. The table confirms that the most hard instances are those with ratios $n/m = 2.5$ and 2.75 especially for classes 6 and 7. But, it should be

TABLE 7. Percentage of unsolved instances.

Classes \ Ratios	n/m=2	n/m=2.25	n/m=2.5	n/m=2.75	n/m=3
Class 1	0	0	0	0	0
Class 2	0	0	0	0	0
Class 3	0	0	0	0	0
Class 4	0	0	0	0	0
Class 5	0	0	0	0	0
Class 6	0	2	17	23	12
Class 7	0	20	26	37	24
Total	0	3.14	6.14	8.57	5.28

noted that for these two classes, the proposed algorithm can solve all the instances with less than 120 jobs. Furthermore, we can note that only 4.62% of the 3500 instances are unsolved.

Through this extensive experimental study, we see that the performance of our algorithm deteriorates when the number of jobs is larger ($n \geq 140$) only for classes 6 and 7. This allows us to conclude that these classes include the hardest instances that can be considered as new challenging instances of the $P||C_{max}$ problem. Overall, our algorithm shows a high performance for all the tested ratios n/m.

IV. CONCLUSION

In this paper, we have proposed a mathematical programming approach based on an arc-flow model for the $P||C_{\max}$ problem. In this formulation, reduction criteria were applied to reduce the size of the proposed graph and consequently to reduce the number of variables in the model. The commercial solver (CPLEX) was used to test the proposed formulation on 3720 hard instances. The computation results show the efficiency of the new approach and attest that it outperforms other methods presented in the literature, because in few seconds on the average, it solves to optimality most of hard instances. In addition, the proposed method is very easy to use and to implement compared to other approaches in the literature.

Nevertheless, some hard instances are still unsolved, mainly for the classes 6 and 7 when the number of jobs is relatively large. Hence, future investigation will explore these classes.

REFERENCES

[1] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Ann. Discrete Math.*, vol. 5, pp. 287–326, Dec. 1979.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.

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

[4] A. C. Beezão, J.-F. Cordeau, G. Laporte, and H. H. Yanasse, "Scheduling identical parallel machines with tooling constraints," *Eur. J. Oper. Res.*, vol. 257, no. 3, pp. 834–844, 2017.

[5] J. E. C. Arroyo and J. Y.-T. Leung, "An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times," *Comput. Ind. Eng.*, vol. 105, pp. 84–100, Mar. 2017.

[6] J. Blażewicz, "Selected topics in scheduling theory," *North-Holland Math. Stud.*, vol. 132, pp. 1–59, Dec. 1987.

[7] A. Hamzadayi and G. Yildiz, "Modeling and solving static *m* identical parallel machines scheduling problem with a common server and sequence dependent setup times," *Comput. Ind. Eng.*, vol. 106, pp. 287–298, Apr. 2017.

[8] R. McNaughton, "Scheduling with deadlines and loss functions," *Manage. Sci.*, vol. 6, no. 1, pp. 1–12, 1959.

[9] E. Mokotoff, "Scheduling to minimize the makespan on identical parallel machines: An LP-based algorithm," *Investigación Operativa*, vol. 8, nos. 1–3, pp. 97–108, Jul./Dec. 1999.

[10] W.-C. Lee, C.-C. Wu, and P. Chen, "A simulated annealing approach to makespan minimization on identical parallel machines," *Int. J. Adv. Manuf. Technol.*, vol. 31, nos. 3–4, pp. 328–334, 2006.

[11] E. Mokotoff, J. L. Jimeno, and A. I. Gutiérrez, "List scheduling algorithms to minimize the makespan on identical parallel machines," *Sociedad Estadistica Investigación Opemtiva*, vol. 9, no. 2, pp. 243–269, 2001.

[12] T. Brueggemann and J. L. Hurink, "Performance of a very large-scale neighborhood for minimizing makespan on parallel machines," *Electron. Notes Discrete Math.*, vol. 25, pp. 29–33, Aug. 2006.

[13] R. L. Graham "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, vol. 17, no. 2, pp. 416–429, 1969.

[14] G. Chiaselotti, M. I. Gualtieri, and P. Pietramala, "Minimizing the makespan in nonpreemptive parallel machine scheduling problem," *J. Math. Model. Algorithms*, vol. 9, no. 1, pp. 39–51, 2010.

[15] D. Laha, "A simulated annealing heuristic for minimizing makespan in parallel machine scheduling," in *Swarm, Evolutionary, and Memetic Computing. SEMCCO* (Lecture Notes in Computer Science), vol. 7677, B. K. Panigrahi, S. Das, P. N. Suganthan, and P. K. Nanda, Eds. Berlin, Germany: Springer, 2012.

[16] L. Tang and J. Luo, "A new ILS algorithm for parallel machine scheduling problems," *J. Intell. Manuf.*, vol. 17, no. 5, pp. 609–619, 2006.

[17] E. Mokotoff, "An exact algorithm for the identical parallel machine scheduling problem," *Eur. J. Oper. Res.*, vol. 152, no. 3, pp. 758–769, 2004.

[18] M. Dell'Amico and S. Martello, "A note on exact algorithms for the identical parallel machine scheduling problem," *Eur. J. Oper. Res.*, vol. 160, no. 2, pp. 576–578, 2005.

[19] M. Dell'Amico, M. Iori, S. Martello, and M. Monaci, "Heuristic and exact algorithms for the identical parallel machine scheduling problem," *INFORMS J. Comput.*, vol. 20, no. 3, pp. 333–344, 2008.

[20] M. Haouari, A. Gharbi, and M. Jemmali, "Tight bounds for the identical parallel machine scheduling problem," *Int. Trans. Oper. Res.*, vol. 13, no. 6, pp. 529–548, 2006.

[21] M. Haouari and M. Jemmali, "Tight bounds for the identical parallel machine-scheduling problem: Part II," *Int. Trans. Oper. Res.*, vol. 15, no. 1, pp. 19–34, 2008.

[22] R. Walter and A. Lawrinenko, "Effective solution space limitation for the identical parallel machine scheduling problem," Working Papers Supply Chain Manage. Friedrich-Schiller-Univ. Jena, Jena, Germany, Tech. Rep. 2/2013, 2013.

[23] K. L. Krause, Y. Y. Shen, and H. D. Schwetman, "Analysis of several task-scheduling algorithms for a model of multiprogramming computer systems," *J. ACM*, vol. 22, no. 4, pp. 522–550, Oct. 1975.

[24] D. S. Hochbaum and D. B. Shmoys, "Using dual approximation algorithms for scheduling problems theoretical and practical results," *J. ACM*, vol. 34, no. 1, pp. 144–162, 1987.

[25] A. C. F. Alvim and C. C. Ribeiro, "A hybrid bin-packing heuristic to multiprocessor scheduling," in *Experimental and Efficient Algorithms. WEA* (Lecture Notes in Computer Science), vol. 3059, C. C. Ribeiro and S. L. Martins, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 1–13.

[26] T. C. E. Cheng and C. C. S. Sin, "A state-of-the-art review of parallel-machine scheduling research," *Eur. J. Oper. Res.*, vol. 47, no. 3, pp. 271–292, 1990.

[27] A. H. G. R. Kan, *Machine Scheduling Problems Classification, Complexity and Computations*. The Hague, The Netherlands: Martinus Nijhoff, 1976.

[28] C. Briand, S. Ourari, and B. Bouzouia, "An efficient ILP formulation for the single machine scheduling problem," *RAIRO-Oper. Res.*, vol. 44, no. 1, pp. 61–71, 2010.

[29] M. Haouari and M. Kharbeche, "An assignment-based lower bound for a class of two-machine flow shop problems," *Comput. Oper. Res.*, vol. 40, no. 7, pp. 1693–1699, 2013.

[30] A. Kooli and M. Serairi, "A mixed integer programming approach for the single machine problem with unequal release dates," *Comput. Oper. Res.*, vol. 51, pp. 323–330, Nov. 2014.

[31] J. M. Valério de Carvalho, "Exact solution of bin-packing problems using column generation and branch-and-bound," *Ann. Oper. Res.*, vol. 86, pp. 629–659, Jan. 1999.

[32] R. Macedo, C. Alves, and J. M. Valério de Carvalho, "Arc-flow model for the two-dimensional guillotine cutting stock problem," *Comput. Oper. Res.*, vol. 37, no. 6, pp. 991–1001, 2010.

[33] M. Mrad, "An arc flow-based optimization approach for the two-stage guillotine strip cutting problem," *J. Oper. Res. Soc.*, vol. 66, no. 11, pp. 1850–1859, 2015.

MEHDI MRAD received the Ph.D. degree in operations research from the University of Tunis. He is currently an Associate professor with the Department of Industrial Engineering, king Saud University. His interests include network design, vehicle routing, scheduling, and cutting problems. He is attracted by the application of different optimization techniques to model and solve real-life complicated problems from different industrial fields.

NIZAR SOUAYAH was born in Tunis, Tunisia, in 1978. He received the B.S. degree in mathematics from the Faculty of Science of Tunis, the M.S. degree in engineering mathematics from the Tunisia Polytechnic School, in 2004, and the Ph.D. degree in applied mathematics from Tunis University, Tunisia, in 2010.

From 2004 to 2010, he was a Lecturer with the University of Sousse. From 2010 to 2013, he was an Assistant Professor with the University of Gafsa. Since 2013, he has been an Assistant Professor with the Natural Sciences Department, Community College, King Saud University. He is the author of several articles. His research interests include combinatorial optimization and scheduling problem.