Solving the Unrelated Parallel Machine Scheduling Problem with Additional Resources Using Constraint Programming

Chapter · January 2018

DOI: 10.1007/978-3-319-75420-8_67

CITATIONS
5

READS
285

2 authors:

Taha Arbaoui
Université de Technologie de Troyes
33 PUBLICATIONS **86** CITATIONS

SEE PROFILE

Farouk Yalaoui
Université de Technologie de Troyes
324 PUBLICATIONS **3,342** CITATIONS

SEE PROFILE

Solving The Unrelated Parallel Machine Scheduling Problem With Additional Resources Using Constraint Programming

Taha Arbaoui and Farouk Yalaoui

Laboratory of Industrial Systems Optimization
Charles Delaunay Institute ICD-LOSI, UMR CNRS 6281
University of Technology of Troyes
12 rue Marie Curie, CS 42060-10004, Troyes, France
{*taha.arbaoui, farouk.yalaoui*}*@utt.fr*

Abstract. This work studies the Unrelated Parallel Machine scheduling problem subject to additional Resources (UPMR). A set of jobs are to be processed by a set of unrelated parallel machines. The processing time and the number of needed resources for each job depend on the machine processing it. Resources are renewable and available in a limited amount. The objective to minimize is the maximum completion time. We formulate the problem using a constraint programming model and solve it using the state-of-the-art solver. We compare the results of this model against the existing approaches of the literature on two sets of small and medium instances. On the set of small instances, we show that the proposed model outperforms existing approaches and optimality is attained for all instances of the set. We further investigate its performance on the medium instances and show that it is able to reach more optimal solutions than any performing approach.

Keywords: Parallel Machine Scheduling, Constraint Programming, Resources

1 Introduction

Parallel Machine Scheduling problem (PMS) is one of the most studied scheduling problems in the literature due to its importance, application and complexity. Three types of PMS problems exist: unrelated, uniform and identical. In the identical type, the processing time of a job is identical on all machines. Essentially, this creates symmetry between the machines and an ordering is possible. As in the identical type, the processing time of a job in the uniform type is identical for all machines. However, machines have different speeds and the processing of a job depends on its processing time and the machine's speed. The unrelated type is the most general one. Jobs' processing times completely depend on the machine.

Different objective functions are considered in the literature. The maximum completion time (makespan) is the most studied objective function. Several approaches have been proposed to tackle PMS problems with different constraints

and objectives. Vallada and Ruiz [14] studied the unrelated PMS problem with sequence-dependent setup times and proposed a genetic algorithm to solve it. Tran et al. [13] studied the same problem and proposed decomposition approaches to attain optimal solutions. Yalaoui and Chen [15] studied the identical PMS with setup times and the job splitting property and proposed an efficient heuristic to solve the problem. The heuristic was later improved by Tahar et al. [12] using a matheuristic. Arbaoui and Yalaoui [3] proposed a bender's decomposition approach for the same problem. For a detailed review on scheduling problems in general and PMS in particular, we refer the reader to [10] and [2].

The PMS problems have been thoroughly studied with different constraints, especially with setup times, no-wait and unavailability constraints. Fewer works, however, tackled the PMS problems with resource constraints. There exist two types of PMS problems with resource constraints: static and dynamic. In the static type, the resource allocation is fixed throughout the schedule, whereas in the dynamic type resources can be assigned and reassigned depending on jobs' assignments.

Among the earliest works that studied PMS problems subject to resource constraints is the one of Blazewicz et al. [5]. The authors proposed a classification for the scheduling problems according to the type and number of resources considered. Ruiz-Torres et al. [11] studied two variants of the problem. The first considers a pre-assignment of jobs to machines, a configuration that the second ignores. To solve the first variant, an integer programming model was introduced while the second variant was tackled using multiple heuristics. Edis and Oguz [6] studied the unrelated PMS with resource constraints and developed a lagrangian-based constraint programming method. Edis and Ozkarahan [8] introduced a hybrid integer and constraint programming approach to tackle the resource-constrained identical PMS problem with machine eligibility restrictions. They showed that the combination of integer and constraint programming yields better results and allows to attain more optimal solutions on a benchmark of 200 instances.

The PMS problem studied in this paper was introduced by Fanjul-Peyro et al. [9] and denoted UPMR. The problem involves assigning jobs to unrelated parallel machines while respecting needed resources for each job. Resources are renewable, i.e. when a job finishes being processed, resources can be used again by another job. Such resources represent operators, tools or dies that are necessary in certain manufacturing fields. This problem is a dynamic UPMR since resource allocation is not fixed before solving the problem. The authors proposed two MIP models, one adapted from an existing work and another based on the strip packing problem. Using these two MIP models, they also introduced three matheuristics and analyzed their performance against each other on two sets of small and medium instances. We introduce a Constraint Programming model and compare it to their approaches. We show that the proposed CP model outperforms all their approaches by attaining optimal solutions for all small instances and more optimal solutions than all matheuristics for medium instances.

The remainder of the paper is organized as follows. Section 2 describes the studied problem. In Section 3, we recall the mixed integer formulation presented in [9] and used to obtain lower bounds for the problem. Section 4 details the Constraint Programming model that was designed for the problem and Section 5 discusses its results and a comparison between existing approaches and the CP model. Finally, the conclusion is to be found in Section 6.

2 Problem description

The Unrelated Parallel Machine Scheduling Problem with additional Resources (UPMR) is described follows. A set of jobs are to be scheduled on a number of unrelated parallel machines while consuming renewable resources. Each job requires a number of scarce resources that depends on the machine and the job. At each moment, the number of consumed resources must not exceed a fixed amount of available renewable resources. Once a machine finishes processing a job, the resources consumed by the finished job are again available for use. The objective considered in work is the minimization of the maximum completion time.

One or more types of resources can be considered for UPMR (operators, tools, materials, etc.). In this work, we focus on the UPMR with one additional resource type. Hence, all jobs share the same type of resources and the same total amount. Moreover, it is assumed that the amount of the resources needed for each job on each machine does not affect the processing time.

The set of machines is made of M machines and the set of jobs of N jobs. For the sake of simplicity, let i denote a machine ($i \in \{1, \ldots, M\}$) and j denote a job ($j \in \{1, \ldots, N\}$). The unrelated PMS problem implies that the processing time of job j depends on machine i, thus referred to as p_{ij}. Each job requires a number of resources r_{ij} that depends on both machine i and job j. The total number of available resources is expressed by R_{max}.

Scheduling problems become more complex when resource constraints are considered. For the studied problem, the difference is noticeable. To illustrate it, we provide a numerical example. Consider a set of six jobs to be placed in three machines and the total number of the renewable resources is four units. The processing times and needed resources for each job are given in Table 1.

Solving the example with and without considering resources gives two completely different solutions. When it is solved without considering the resources,

Table 1. A numerical example of six jobs, three machines and four units of resources.

Jobs	Processing times					
	1	2	3	4	5	6
M1	3	7	4	8	6	3
M2	4	5	3	2	7	2
M3	2	1	4	5	1	10

Jobs	Needed resources					
	1	2	3	4	5	6
M1	4	2	2	1	2	2
M2	1	1	3	3	3	2
M3	2	3	2	1	3	1

i.e. solving the classical unrelated PMS (UPM), one may find an optimal solution with a $C_{max} = 4$. When resources are considered, the optimal solution has $C_{max} = 8$.

3 Mixed Integer Linear Model

In [9], the authors introduced two mixed integer linear models, the first is adapted from the UPMR presented in [7] denoted UPMR-S, and the second is based on the strip packing problem denoted UPMR-P. Recall that when resource constraints are not considered, UPMR becomes an Unrelated Parallel Machine scheduling problem (UPM). In [9], authors showed that the UPM MIP model can be used to obtain a lower bound and was used to evaluate the solutions of the proposed matheuristics.

Model UPMR-S uses the following decision variables: $x_{ijk} = 1$ if job j is assigned to machine i and finishes being processed at time k. Since job j can never be finished before time p_{ij}, $k \geq p_{ij}$. Model UPMR-S is described as follows:

$$Minimize \quad C_{max} \tag{1}$$

$$\sum_{i=1}^{M} \sum_{k \geq p_{ij}} k \; x_{ijk} \leq C_{max} \qquad j = 1, ..., N \tag{2}$$

$$\sum_{i=1}^{M} \sum_{k \geq p_{ij}} x_{ijk} = 1 \qquad j = 1, ..., N \tag{3}$$

$$\sum_{j=1}^{N} \sum_{s \in \{max(k,p_{ij}),...,k+p_{ij}-1\}} x_{ijs} \leq 1 \qquad i = 1, ..., M, \quad k = 1, ..., K_{max} \tag{4}$$

$$\sum_{i=1}^{M} \sum_{j=1}^{N} \sum_{s \in \{max(k,p_{ij}),...,k+p_{ij}-1\}} r_{ij} \; x_{ijs} \leq R_{max} \qquad k = 1, ..., K_{max} \tag{5}$$

$$x_{ijk} \in \{0, 1\} \qquad i = 1, ..., M, \quad j = 1, ..., N, \quad k = 1, ..., K_{max} \tag{6}$$

Equation (1) is the objective function. Constraints (2) are used to obtain the maximum completion time. Constraints (3) state that a job can be assigned to exactly one machine and constraints (4) force it to finish at one specific time. Finally, constraints (5) are used to avoid exceeding the total number of available resources.

K_{max} is an important parameter of the formulation. It is inherently related to the maximum completion time. Setting it too small or too big might have consequences on the performance of the formulation. If K_{max} is set too small, solving the formulation becomes impossible since some jobs might not be assigned to a certain time. Setting it too big makes solving the formulation difficult since this leads to a large number of variables. Contrary to M and N, the maximum numbers of machines and jobs, K_{max} can be calculated in different manners. Fanjul-Peyro et al. [9] practically observed that setting $K_{max} = \min_{i \in \{1,\dots,M\}} \sum_{j=1}^{N} p_{ij}$ provides the best results. They tested other tighter values but the solver faced difficulties in finding feasible solutions.

4 Constraint Programming Model

Constraint Programming (CP) is considered as an alternative modeling tool for linear and Integer Programming (IP). It has seen an increasing interest due to the facility and the flexibility one finds when using it to model combinatorial problems. While CP and IP have their strengths and weaknesses, CP approaches are mostly used to search for feasible solutions while IP approaches have an emphasis on optimality and improving lower bounds. When solving a constraint programming model, the solver focuses on reducing variables' domains so as to obtain multiple feasible solutions. Solving an integer programming model requires both improving the lower bounds (among others LP relaxations) and upper bounds (feasible solutions) to reduce the gap between both bounds.

CP has seen an increasing interest in the scheduling community due to its successful application on different problems [8]. Solvers are improved along the years and some solvers offer a special framework for scheduling problems that allows efficient modeling of complex and specific scheduling constraints like the no overlap constraint, which is typically imposed between the jobs of the same machine. For a detailed review on the use of CP in scheduling, we refer the reader to [4].

Contrary to linear and integer programming, the constraint programming model largely depends on the modeling framework used to express the variables, the constraints and the objective function. We choose to utilize the IBM CP Optimizer in this study. We only provide the description of the structures used in the CP model as the details of the modeling framework can be found in [1].

On top of the usual continuous and integer variables, CP Optimizer offers a special variable called the *interval* variable. This variable is characterized by two integers representing its start time and its end time. Moreover, an interval variable can be optional i.e. it can be ignored (or deleted) in the solution. When the interval variable is considered in the solution, it is said to be *present*. Otherwise, it is considered *absent*. The length of an interval variable is defined as the difference between its end time and its start time.

In our model, v_{ij} is an interval variable referring to the processing of job j on machine i. Thus, we have $M \times N$ optional interval variables v_{ij}. The length of variable v_{ij} is equal to p_{ij}.

The optionality of interval variables allows us to select only one interval variable for each job so that a job is assigned to a single machine. This is achieved using the *alternative* constraint. The alternative constraint is applied to a set (at least two) of interval variables and states that if one of the interval variables is present, all the other interval variables should be absent. This constraint is applied to all variables v_{ij} of job j to enforce that only one of them should be present and the others have to be absent, i.e. the job j can be assigned to only one machine.

Jobs scheduled on the same machine should not overlap. To avoid this situation, we use the *noOverlap* global constraint applied to a set of interval variables to avoid overlapping. The *noOverlap* constraint is applied to all v_{ij} variables of the same machine i.

Machine completion times are expressed using interval variables O_i. Naturally, the start time of variable O_i is equal to the start time of the first job scheduled on this machine and its end time is equal to the end time of the last job scheduled on it. To accomplish this synchronization, the *Span* global constraint is utilized. It states that one interval variable (O_i in our case) should be spanned over a number of variables ($v_{ij}, \forall j$), i.e. it starts with the first interval variable v_{ij} and ends with the last interval variable v_{ij}.

To be able to model the resource constraint, the cumulative constraint is used. By applying this constraint, denoted *Pulse* constraint in the used framework, each time an interval variable v_{ij} is present, r_{ij} (the amount of consumed resources by job j on machine i) is taken into account. At any time, the total consumed resources should be no more than R_{max}.

The CP model is comprised of two types of interval variables v_{ij} (optional) and O_i and can be written as follows:

$$Minimize \qquad max \;\; O_i \tag{7}$$

$$Alternative(v_{1,j}, \cdots, v_{M,j}) \quad j = 1, \cdots, N \tag{8}$$

$$noOverlap(v_{i,1}, \cdots, v_{i,N}) \quad i = 1, \cdots, M \tag{9}$$

$$Span(O_i, v_{i,1}, \cdots, v_{i,N}) \quad i = 1, \cdots, M \tag{10}$$

$$\sum_{i=1}^{M} \sum_{j=1}^{N} Pulse(v_{ij}) \leq R_{max} \tag{11}$$

Equations (7) express the makespan that corresponds to the maximum of machines' completion times. Equations (8) enforce the presence of one v_{ij} variable, i.e. job j is assigned to at least one of the machines. Equations (9) ensure that jobs assigned to the same machine do not overlap. Equations (10) establish the link between variables v_{ij} and O_i. Finally, the resource constraint is respected using Equations (11).

5 Experimental results

This section details the different experimentations conducted on the CP model and a comparison with existing results in the literature on an established benchmark. It particularly illustrates the strengths and difficulties faced when dealing with the CP model and analyzes its performance with regards to the lower bounds.

5.1 Environment and benchmarking

The MIP and the CP models were implemented in C++ using the Concert library of the IBM Optimization Studio (CPLEX and CP Optimizer respectively) and compiled using GCC 6.3.1. The experimentations were run on an Intel(R) Xeon(R) CPU E5-2650 v2@2.60GHz with 8GB of memory. The number of threads in each run was set to one. The time limit set for each run is one hour.

Fanjul-Peyro et al. [9] introduced a benchmark of 900 instances divided into two sets: small and medium instances. The small set contains instances with 8, 12 and 16 jobs. The medium set contains instances with 20, 25 and 30 jobs. The numbers of machines considered in both sets are 2, 4 and 6. For a detailed description of the instances, please refer to [9].

5.2 Lower bounds

As previously stated in Section 2, the solution of UPM represents a lower bound for UPMR. To solve UPM, model UPMR-S can be used without constraints (5). We practically observed that solving this formulation without constraints (5) and end times of jobs (variables x_{ij} are used instead) requires a few seconds and is therefore used to obtain the lower bound denoted LB. It should be noted that this lower bound was the best among three tested in [9], especially for medium instances.

5.3 Results and comparison with existing approaches

In this section, we compare the results obtained by the proposed CP model against the existing approaches in the literature. Since the problem was introduced by Fanjul-Peyro et al. [9], the only existing approaches proposed to solve the problem are the ones described in their work. As presented previously, they proposed two MIP models and derived six matheuristics from them. We first compare the CP model to both models, UPMR-S and UPMR-P, since the former allows to reach optimality more often whereas the latter obtains more feasible solutions. We later compare the CP model to matheuristics MAF-S and MAF-P, the best matheuristics in terms of obtaining optimal solutions.

To check the optimality for a given solution of the UPMR, the authors used lower bound LB obtained by solution the Unrelated Parallel Machine (UPM) problem as described in Section 5.2.

Table 2 presents a comparison between the CP model and the existing MIP models (UPMR-S and UPMR-P). The first column represent the number of jobs considered. For each model, a set of four columns are provided. Column 0%*Gap* gives the number of instances for which optimality was proved within the time limit (one hour). Column Opt provides the number of instances for which the optimal solution was reached but optimality was not proved by solving the models. Instead, a solution is qualified as optimal if its fitness is equal to LB. Column Feas presents the number of instances for which a feasible non-optimal solution is obtained. Finally, column NoSol gives the number of instances for which no solution was found.

Table 2. Comparison of the CP model against existing formulations in the literature.

	UPMR-S [9]				UPMR-P [9]				CP model			
N	0%Gap	Opt	Feas	NoSol	0%Gap	Opt	Feas	NoSol	%Gap	Opt	Feas	NoSol
8	137	147	13	0	146	150	4	0	150	150	0	0
12	84	87	65	1	47	85	103	0	150	150	0	0
16	53	65	78	19	1	26	149	0	150	150	0	0
20	21	45	108	21	0	7	150	0	114	20	36	0
25	6	18	114	30	0	1	150	0	56	36	94	0
30	1	5	106	43	0	0	150	0	49	40	101	0

For the small instances, one can remark that the CP model obtains the optimal solution for all the 150 instances. It is clear that, on the set of small instances, the MIP models UPMR-S and UPMR-P provide lower performances compared to the CP model, particularly model UPMR-S which struggles to find feasible solutions for instances with 16 jobs. It is also worth noting that both MIP models were not able to reach the 0% Gap optimality for all the instances, even for smallest instances (with 8 jobs).

For medium instances, the CP model faces difficulties as the number of jobs increases. For example, for 20-job instances, it is still able to reaches optimality for 134 instances whereas it reaches optimality for 89 instances in the case of 30-job instances. It is worth noting that the CP model is also able to obtain feasible solutions for all 900 instances. When comparing UPMR-S to UPMR-P, one can see that the former reaches more optimal solutions than the latter. However, it faces difficulties in finding feasible solutions as the size of instances increases, which is not the case of UPMR-P. The CP model overcomes both difficulties. It is able to find feasible solutions for all the instances, regardless of their size. Moreover, it finds more optimal solutions than both models combined.

Table 3 presents a comparison between matheuristics MAF-S and MAF-P (the best performing heuristics) and the proposed CP model. For each size of instances N in the first column, three columns Opt, Feas and NoSol are shown for each approach. Column Opt presents the number of optimal solutions found

by the approach while columns Feas and NoSol show respectively the number of feasible non-optimal solutions and the number of instances for which no solution was found.

One can clearly remark that, on the set of small instances, the CP model outperforms both matheuristics by finding optimal solutions for all the 450 instances while both matheuristics MAF-S and MAF-P find at maximum optimal solutions for half instances (75 for 8-job instances).

Table 3. Comparison of the CP model against existing matheuristics in the literature.

	MAF-S [9]			MAF-P [9]			CP model		
N	Opt	Feas	NoSol	Opt	Feas	NoSol	Opt	Feas	NoSol
8	75	75	0	75	75	0	150	0	0
12	72	78	0	72	78	0	150	0	0
16	64	85	1	71	79	0	150	0	0
20	64	85	1	67	83	0	134	16	0
25	48	99	3	60	90	0	92	58	0
30	50	98	2	76	74	0	89	61	0

For the medium instances, the difference is noticeable. In the case of the 20-job instances, the proposed CP model reaches optimality for 134 instances whereas matheursistics MAF-S and MAF-P respectively find optimality for 64 and 67 instances.

From the presented results, one can clearly see that the proposed CP model outperforms existing exact and heuristic approaches in the literature. Most importantly, it is able to prove optimality for the set of small instances and reaches optimality for more medium instances than all existing approaches. However, one can clearly see that, despite finding all feasible solutions, the proposed CP model finds less optimal solutions as the number of jobs increases.

6 Conclusion

We studied the unrelated PMS problem with additional resources with an objective of minimizing the maximum completion time. The problem, though very practical and realistic, was recently introduced and studied by Fanjul-Peyro et al. [9]. Resources are renewable and consumed by jobs as they are processed on the machines. Like the processing times, the number of needed resources by the job depends on the machine on which it is processed.

We proposed a constraint programming model that was implemented using the scheduling framework of the IBM CP Optimizer. Using this approach, optimality was reached for all the small instances. The approach was also able to outperform existing approaches as it reached more optimal solutions than all approaches.

The proposed approach confirms the important and interesting application of constraint programming in scheduling. Despite the promising results, it is expected that the approach faces increasing difficulty as the size of the instances grows. Therefore, further research on larger instances should be directed towards using metaheuristics, hyper-heuristics and decomposition approaches.

References

1. IBM Software, 2016. IBM ILOG CPLEX Optimization Studio 12.6.
2. Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378, 2015.
3. Taha Arbaoui and Farouk Yalaoui. An exact approach for the identical parallel machine scheduling problem with sequence-dependent setup times and the job splitting property. In *Industrial Engineering and Engineering Management (IEEM), 2016 IEEE International Conference on*, pages 721–725. IEEE, 2016.
4. Philippe Baptiste, Claude Le Pape, and Wim Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Science & Business Media, 2012.
5. Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.
6. Emrah B Edis and Ceyda Oguz. Parallel machine scheduling with additional resources: a lagrangian-based constraint programming approach. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 92–98. Springer, 2011.
7. Emrah B Edis and Ceyda Oguz. Parallel machine scheduling with flexible resources. *Computers & Industrial Engineering*, 63(2):433–447, 2012.
8. Emrah B Edis and Irem Ozkarahan. A combined integer/constraint programming approach to a resource-constrained parallel machine scheduling problem with machine eligibility restrictions. *Engineering Optimization*, 43(2):135–157, 2011.
9. Luis Fanjul-Peyro, Federico Perea, and Rubén Ruiz. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, 260(2):482–493, 2017.
10. Michael Pinedo. *Scheduling: theory, algorithms and systems*. Springer, 5th edition, 2012.
11. Alex J Ruiz-Torres, Francisco J López, and Johnny C Ho. Scheduling uniform parallel machines subject to a secondary resource to minimize the number of tardy jobs. *European Journal of Operational Research*, 179(2):302–315, 2007.
12. D N Tahar, F Yalaoui, C Chu, and L Amodeo. A linear programming approach for identical parallel machine scheduling with job splitting and sequence-dependent setup times. *International Journal of Production Economics*, 99(1):63–73, 2006.
13. Tony T Tran, Arthur Araujo, and J Christopher Beck. Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, 28(1):83–95, 2016.
14. Eva Vallada and Rubén Ruiz. A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):612–622, 2011.
15. F Yalaoui and C Chu. An efficient heuristic approach for parallel machine scheduling with job splitting and sequence-dependent setup times. *IIE Transactions*, 35(2):183–190, 2003.