Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times

Luis Fanjul-Peyró[a], Rubén Ruiz[a], Federico Perea[a,*]

[a] *Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain.*

Abstract

Parallel machine scheduling problems have many practical and industrial applications. In this paper we study a generalization which is the Unrelated Parallel Machine scheduling problem with machine and job sequence Setup times (UPMS) with makespan minimization criterion. We propose new mixed integer linear programs and a mathematical programming based algorithm. These new models and algorithms are tested and compared with the existing ones in an extensive and comprehensive computational campaign. The performance of two popular commercial solvers (CPLEX and Gurobi) is also compared in the experiments. Results show that the proposed methods significantly improve on existing methods and are able to obtain solutions for extremely large instances of up to 1000 jobs and eight machines with relative deviations from lower bounds below 0.8%.

Keywords: Parallel machine, Scheduling, Sequence dependent setup times, Makespan.

1. Introduction

Industrial manufacturing scheduling entails the assignment of production activities to a limited number of available production resources. After the assignment, these activities must be scheduled with the objective of optimizing one or more key criteria. There are literally hundreds of different scheduling

*Corresponding author. Tel: +34 96 387 70 00. Ext: 74914. Fax: +34 96 387 74 99
Email addresses: lfpeyro@hotmail.com (Luis Fanjul-Peyró), rruiz@eio.upv.es (Rubén Ruiz), perea@eio.upv.es (Federico Perea)

problems as they model many different production processes. One important scheduling problem is the so called parallel machines scheduling problem, where n independent jobs (indexed by j) have to be assigned and scheduled to m machines (indexed by i) that may process jobs in parallel. Each job must be manufactured by exactly one machine without preemption. No machine can process more than one job at a time, and in principle any job can be assigned to any machine. In the most general case machines are said to be unrelated, meaning that the time needed to process a given job depends on the machine to which it is assigned. This time is known in advance, is deterministic and denoted as p_{ij}, $i \in \{1, 2, \ldots, m\}$, $j \in \{1, 2, \ldots, n\}$. Unrelated Parallel Machines scheduling problems (UPM) model high output production shops or even central stages in certain production processes, for example, the kiln firing stage in ceramic tile manufacturing. Without any other additional constraints or considerations, in the UPM, a job is processed when all previously assigned jobs on the same machine are completed. This completion time is referred to as C_j. With this in mind, the most commonly studied objective is the minimization of the maximum completion time. This is known as the makespan or $C_{\max}$. The UPM with this criterion is denoted as $R//C_{\max}$ (Graham et al., 1979). This problem is $\mathcal{NP}$-Hard even for the simplest case of just two identical parallel machines, denoted by $P2//C_{\max}$ (Lenstra et al., 1977). Furthermore, the order in which the jobs are processed on a given machine is irrelevant to optimizing the $C_{\max}$. It is, therefore, a sort of assignment problem.

However, practical industrial problems include a large number of additional considerations, such as the possibility of machine disruptions (see Yin et al. (2017)), competing agents that share the machines (see Yin et al. (2016)) and many others. Among these, it can be argued that the most common one is the presence of setup times. Setups are usually non-productive activities carried out on the production lines between the production of consecutive products in the sequence. Cleaning, adjustments, reconfigurations and color preparations, etc. are examples of setups. The vast majority of production processes require these setups. As a result, the literature on scheduling with setups is extensive. Allahverdi (2015) has published a recent review (out of a series of three) which gathers no less than 500 papers in just the last 10 years.

There are several types of setups. The most complex ones are setup times that depend on the machine and job sequence and at the same time are separable from the processing times. This paper considers the Unrelated Parallel Machine scheduling problem with sequence dependent Setup times

(UPMS) or $R/s_{ijk}/C_{\max}$, where s_{ijk} denotes the amount of setup time needed at machine i after having finished job j if the next job to be processed on this machine is k. Note that the setup times usually satisfy the triangular inequality, i.e.: $s_{ijk} \leq s_{ij\ell} + p_{i\ell} + s_{i\ell k}$, $i \in \{1, 2, \ldots, m\}$, $j, k, \ell \in \{1, 2, \ldots, n\}$, $j \neq k, j \neq \ell, k \neq \ell$. While in theory this is only needed for some mathematical models, it makes sense from a practical point of view. Note that with the addition of setup times the UPMS is significantly harder than the UPM. First of all, the job sequence on each machine is no longer irrelevant as the setup times depend on the order in which jobs are processed on each machine. As a matter of fact, a special case of the UPMS with a single machine and where all processing times are zero can be modeled as a particular Traveling Salesman Problem (TSP). While the literature on the UPM is extensive (see Fanjul-Peyro and Ruiz, 2010, 2011 for some references), the UPMS has been, comparatively speaking, much less studied. Given its complexity, most existing literature deals with heuristics and metaheuristics, and the exact approaches proposed are only valid for relatively small to medium instances. One of the contributions of this paper is new mathematical reformulations for the UPMS. Mathematical programming models cannot compete with the various and powerful heuristics and metaheuristics designed to solve large-scale instances of the UPMS. However, since they help to better understand the problem, are easy to implement, replicate and modify, we devoted some effort to improving existing mixed integer linear programs (MILP) for the UPMS. Besides, a MILP model is often preferable to a specifically tailored algorithm as results are arguably harder to reproduce in this later case. The MILP models presented in this paper are able to solve instances of considerable size to almost optimality and with very small deviations from lower bounds. Additionally, based on these efficient MILPs, another contribution of this paper is the design of an efficient mathematical programming based algorithm. These methods can be used in exact algorithms (like the one in Section 5), or in combination with heuristic techniques. The latter results in the so-called matheuristics, see Fanjul-Peyro et al. (2017) for an example of the application of these techniques to scheduling problems. An added advantage of using MILP models is that one can benefit from the continuous improvements in MILP solvers. The same models will solve larger instances, and with shorter computational times if solvers become more efficient. Two recent state-of-the-art solvers will be compared in the appendix.

In short, the contribution of this paper is twofold:

- new mathematical models for the UPMS that solve instances of up to $n = 400$ jobs and $m = 4$ machines, with deviations with respect to optimal solutions of less than 1%. This is a huge step from the state-of-the-art model found in the literature, which reported optimal solutions to instances of $n = 60$ jobs in Avalos-Rosales et al. (2015), although we were able to efficiently solve instances of $n = 200$ jobs and $m = 4$ machines with that model just by changing the solver (see Section 4). This fact supports our statement that mathematical models greatly benefit from the improvements in solvers. Furthermore, the models presented in this paper are more robust against changes in the solver than this state-of-the-art model, as we will show in the Appendix.

- a new exact algorithm that combines some ideas from the literature and a new methodology with the proposed MILP models. This algorithm which solves instances of up to $n = 1000$ jobs and $m = 8$ machines with deviations with respect to optimal solutions of less than 1%. This is a significant improvement over the best known exact algorithm proposed for this problem by Tran et al. (2016), which was able to solve instances of up to 120 jobs (see Section 5).

The rest of the paper is structured as follows. Section 2 reviews the literature on the problem. The problem is stated in Section 3 along with the state-of-the-art mathematical model to solve it. The first contribution of this paper is the MILP models introduced in Section 4. A new exact algorithm based on these MILP models is described in Section 5. All these models and algorithms are comprehensively tested in Section 6. Finally, in Section 7 conclusions and future research directions are given. An Appendix shows two more MILP models and a comparison between CPLEX and Gurobi.

2. Literature review

As stated, the complexity of the UPMS has resulted in many studies with proposals of heuristic and metaheuristic algorithms. However, some efforts have been made to design exact approaches to solve the UPMS. The model presented in Guinet (1991) served as a basis for other MILP approaches, although optimality was guaranteed in small instances only. Some improvements on this MILP can be found in Vallada and Ruiz (2011), where problems of up to 14 jobs could be solved to optimality. In Vallada and Ruiz (2012), an adaptation of the model presented in Balakrishnan et al.

(1999) is proposed for weighted earliness-tardiness minimization. To the best of our knowledge, this model, which has a significantly reduced number of binary variables, has not been adapted to the makespan objective before, something that will be carried out in the Appendix. It was not until the last few years that much larger instances of the UPMS were solved to optimality. Avalos-Rosales et al. (2015) proposed a MILP that efficiently solved some instances of up to 60 jobs and 8 machines. A similar MILP previously served as a master problem in some iterative algorithms presented in Tran and Beck (2012) and Tran et al. (2016). The sizes of the problems solved in these last papers are much larger, reaching $n = 120$ jobs, although still far from the sizes we will solve in Section 6.

Because exact algorithms did not prove efficient in solving real life instances of this problem until recently, more papers on heuristic and metaheuristic approaches can be found in the literature. In Glass et al. (1994), some metaheuristics that rely on a local search are explored. A heuristic based on set partitioning is proposed in Al-Salem (2004). Later, another heuristic in Rabadi et al. (2006) and a tabu search in Helal et al. (2006) proved closer to optimality than Al-Salem (2004). All these algorithms were improved on by the ant colony optimization presented in Arnaout et al. (2010), while the genetic algorithm (GA) presented by Vallada and Ruiz (2011) was able to give even smaller deviations from best known solutions. Another GA was proposed by Yilmaz Eroglu et al. (2014). A complex inmune-based method was presented by Diana et al. (2014). More recently Wang et al. (2016) presented a hybrid between an estimation of distribution and iterated greedy methods with good results. In any case, and as we will show in Section 6, we are able to solve very large instances to almost optimality with the proposed approach.

Note that we are not reviewing the large body of research concerning problems related to the UPMS with other objectives, constraints and situations as it is beyond the scope and space limitations in this paper. The interested reader is referred to (Allahverdi, 2015) for an in-depth review.

3. State-Of-The-Art model

The Unrelated Parallel Machine scheduling problem with Setups (UPMS) takes the following input data: 1) A set of jobs $N = \{1, ..., n\}$, indexed by j, k, ℓ. For modeling purposes it will be useful to enlarge the set N so that it includes a dummy job denoted by 0. This new set is denoted by $N_0 = N \cup \{0\}$. The dummy job is needed in the TSP-like formulations of the problem, and can be viewed as the depot in the TSP. 2) A set of machines $M = \{1, ..., m\}$ that can process these jobs, indexed by i. 3) A matrix $P \in \mathbb{R}_+^{(m)\times(n+1)}$, with the processing times $p_{ij} \geq 0$ that job j needs on machine i where no preemption is allowed. Note that $p_{i0} = 0,\ \forall\ i \in M$. 4) A matrix $S \in \mathbb{R}_+^{m\times(n+1)\times(n)}$ with the setup times. After processing job j on machine i a setup time $s_{ijk} \geq 0$ is needed if the next job processed on i is k. The objective is to find the sequence of jobs processed on each machine that minimizes the makespan. We now define the concept of successor and predecessor in the UPMS context.

Definition 3.1. *Let $j, k \in N$ be two jobs processed on the same machine $i \in M$. We say that k is the successor (predecessor) of j on machine i if j is processed immediately before (after) k, and there are no other jobs processed between them on i.*

Note that k is the successor of j if and only if j is the predecessor of k.

To the best of the authors' knowledge, the most efficient MILP for the UPMS is the one found in Avalos-Rosales et al. (2015). This formulation uses variables X, Y, C:

- $X_{ijk} = 1$ if k is the successor of j on machine i, zero otherwise.
- $Y_{ij} = 1$ if j is processed on machine i, zero otherwise.
- $C_j \geq 0$ is the completion time of job j.

We will see later that variables Y are relaxed in this formulation. The MILP presented in Avalos-Rosales et al. (2015), that we denote as AAA, consists of:

$$\min\ C_{\max} \tag{1}$$

$$\text{s.t.} \sum_{j\in N_0, k\in N, k\neq j} s_{ijk}X_{ijk} + \sum_{j\in N} p_{ij}Y_{ij} \leq C_{\max},\ i \in M \tag{2}$$

$$\sum_{k\in N} X_{i0k} \leq 1,\ i \in M \tag{3}$$

$$\sum_{i\in M} Y_{ij} = 1,\ j \in N \tag{4}$$

$$Y_{ij} = \sum_{k\in N_0, j\neq k} X_{ijk},\ i \in M, j \in N \tag{5}$$

$$Y_{ik} = \sum_{j\in N_0, j\neq k} X_{ijk},\ i \in M, k \in N \tag{6}$$

$$C_k - C_j + V(1 - X_{ijk}) \geq s_{ijk} + p_{ik},$$
$$j \in N_0, k \in N, j \neq k, i \in M \tag{7}$$

$$C_0 = 0 \tag{8}$$

$$C_{\max} \geq C_j,\ j \in N \tag{9}$$

$$X_{ijk} \in \{0,1\},\ Y_{ij} \geq 0, C_j \geq 0.$$

Constraints (2) define the makespan (note that the left hand side of these constraints define the amount of time that machine i is busy). Constraints (3) ensure that at most, one job is scheduled as the first on each machine after the dummy job. Constraints (4) state that each job is to be processed on exactly one machine. Constraints (5) ensure that all jobs have one successor (possibly the dummy, as this model includes dummy jobs at the end of each machine) on the machine in which they are processed. Analogously, constraints (6) ensure that all jobs have one predecessor on the machine in which they are processed. Constraints (7) provide a right processing order and break subtours. They impose that if k is the successor of j on machine i, then k should be completed in at least $s_{ijk} + p_{ik}$ units of time after j is completed. Constraint (8) sets the completion time of the dummy job to zero. Constraints (9) are feasible cuts that proved efficient in Avalos-Rosales et al. (2015). The reader should note that the structure of the problem allows for variables Y_{ij} to be relaxed and are therefore defined as positive instead of binary. This is true because Y_{ij} is defined as the sum of binary variables X_{ijk}, constraints (5) and (6), this sum being bounded from above by 1 in constraints (4). In Avalos-Rosales et al. (2015), it is reported that this MILP model (referred to as model 2b in the original paper) efficiently solves some instances of up to 60 jobs.

In order to avoid confusion, in this paper variables are represented by capital letters, while parameters are represented by small letters.

4. The proposed models

The traveling salesman problem (TSP) is a well-known problem in combinatorial optimization. Given a graph (N, A), with an identified origin node usually called the *depot* (which may or may not be in N), the TSP asks for the minimum length route that visits all nodes of N exactly once while starting and finishing at the depot. If only one machine is considered, and the setup times are sequence dependent (like our problem), then the corresponding scheduling problem is a TSP, see Pinedo (2005). When one considers that more than one salesman is available, and that each city must be visited by exactly one salesman, we have a multiple traveling salesman problem (m-TSP), see Bektas, 2006; Kara and Bektas, 2006. This would cover the case of identical machines processing jobs in parallel. A special case of m-TSP arises when the costs associated with traversing the arcs in A need not be the same for all salesmen. This new problem is called the heterogeneous m-TSP, and better reflects our UPMS problem, as machines process jobs at different speeds, and the setup times also depend on the machines. To the best of our knowledge, this heterogeneous m-TSP has seldom been addressed in the literature (Sundar and Rathinam, 2015). However, its extension as a heterogeneous fleet vehicle routing problem has received much more attention. The interested reader is referred to the survey in Koç et al. (2016).

In summary, the UPMS can be seen as an heterogeneous m-TSP, in which the jobs correspond to cities, and the machines correspond to salesmen. If two cities j and k are visited one after the other by the same salesman i in the heterogeneous m-TSP, in the corresponding UPMS we say that k is the successor of j on machine i. The cost for the salesman (machine) i of traversing the arc linking cities j and k (of processing job j and then k) is equal to $p_{ij} + s_{ijk}$.

The MILPs that we propose in this section share the structure defined by equations (1) to (6) which include variables X and Y. Note that constraints (7) are basically subtour elimination constraints (SEC) and there are many options for these in the underlying m-TSP. In this section, we substitute (7), (8) and (9) by other SEC and cuts. Note that without these three sets of constraints variables C_j are no longer needed. Instead, we define the following set of variables:

- $U_j \in \mathbb{Z}_+$ is a lower limit on the number of jobs processed before j on the machine where j is processed. We will see that due to the structure of the constraints involving these variables they can be relaxed to $U_j \geq 0$.

4.1. Subtour elimination constraints

Two different subtour elimination constraints will be tested to substitute (7). The first one is:

$$U_j - U_k + n \sum_{i \in M} X_{ijk} \leq n - 1, \; j, k \in N, j \neq k. \tag{10}$$

This set of constraints ensures that, if k is the successor of j on any machine (and therefore $\sum_{i \in M} X_{ijk} = 1$), then $U_k \geq U_j + 1$. Otherwise ($\sum_{i \in M} X_{ijk} = 0$), they set the upper bound $U_j \leq n - 1$. These constraints are adapted from the well-known Miller-Tucker-Zemlin (MTZ) constraints, and have been extensively used to break subtours of the single TSP and its variants ever since they were proposed in Miller et al. (1960).

The second SECs are adapted from those proposed by Desrochers and Laporte (1991) for the single TSP, which make the relation between the X and the U variables stronger than in the MTZ constraints:

$$U_j - U_k + n \sum_{i \in M} X_{ijk} + (n-2) \sum_{i \in M} X_{ikj} \leq n - 1, \; j, k \in N, j \neq k. \tag{11}$$

These constraints will be denoted as DL. Note that they impose for any pair of jobs j and k where k is the successor of j, that $U_k = U_j + 1$ (as opposed to $U_k \geq U_j + 1$ in MTZ). In any other case, they impose the upper bound $U_j \leq n - 1$ (like in MTZ).

4.2. Valid inequalities

It should be noted that both the MTZ and DL constraints were designed for the standard TSP (with one salesman only), where variables U_j take *all* integer values ranging from 0 to $n - 1$. In the UPMS, this is not necessarily true (this would be true only if one machine processed all jobs). To illustrate, consider a UPMS problem instance with 10 jobs and more than one machine. If one of the machines processes jobs 1,2,3, we could have $U_1 = 0, U_2 = 1, U_3 = 2$. But we could also have $U_1 = 7, U_2 = 8, U_3 = 9$ or infinite many other different combinations. In order to reduce the feasible set without missing potential optimal solutions, the following sets of valid inequalities are proposed to narrow down the possible values that variables U_j may take.

The first set of valid inequalities we propose are adapted from some initially designed for an m-TSP in Kara and Bektas (2006). These constraints, denoted as KB, are:

$$U_j + (n-2)\sum_{i\in M} X_{i0j} - \sum_{i\in M} X_{ij0} \leq n-2,\ j \in N, \tag{12}$$

$$U_j + \sum_{i\in M} X_{i0j} \geq 1,\ j \in N. \tag{13}$$

The following proposition proves that KB are actually valid inequalities.

Proposition 4.1. *Constraints KB,* (12) *and* (13)*, are valid inequalities for the UPMS.*

Proof. The proof is similar to the one found in Kara and Bektas (2006). Only four cases are possible:

1. If j is the first and last job to be processed on its machine, $\sum_{i\in M} X_{i0j} = 1, \sum_{i\in M} X_{ij0} = 1$, the two constraints imply $0 \leq U_j \leq 1$.
2. If j is the first but not the last job to be processed on its machine, $\sum_{i\in M} X_{i0j} = 1, \sum_{i\in M} X_{ij0} = 0$, the two constraints imply $U_j = 0$.
3. If j is the last but not the first job to be processed on its machine, $\sum_{i\in M} X_{i0j} = 0, \sum_{i\in M} X_{ij0} = 1$, the two constraints imply $1 \leq U_j \leq n-1$.
4. If j is neither the first nor the last job to be processed on its machine, $\sum_{i\in M} X_{i0j} = 0, \sum_{i\in M} X_{ij0} = 0$, the two constraints imply $1 \leq U_j \leq n-2$.

□

We now propose another set of valid inequalities, by slightly modifying (12) so it becomes:

$$U_j + (n-1)\sum_{i\in M} X_{i0j} \leq n-1,\ j \in N. \tag{14}$$

These constraints impose that if a job j is the first job on one machine then $U_j = 0$ regardless whether j is also the last job or not. The new set of valid inequalities consists of (14) and (13), and is denoted as AM. The following proposition, which can trivially be proved, shows that these constraints are actually valid inequalities for the UPMS problem.

Proposition 4.2. *Constraints AM,* (14) *and* (13)*, are valid inequalities for the UPMS.*

By combining the two types of SEC, and the two sets of valid inequalities proposed, we build six MILP models. They all share the structure defined by (1) to (6). Their differences rely on the type of SEC constraint used and the valid inequalities applied (if any). The different models proposed are denoted as XXX-YY, where XXX stands for the type of SEC used (MTZ or DL), and YY stands for the type of valid inequalities (blank if no valid inequalities are added, KB if (12) and (13) are added and AM if (14) and (13) are added). Table 1 summarizes this notation.

	No valid inequalities	(12) and (13)	(14) and (13)
SEC (10)	MTZ	MTZ-KB	MTZ-AM
SEC (11)	DL	DL-KB	DL-AM

Table 1: Notation of different MILP models proposed. All models include (1) to (6). By rows, the type of SEC applied. By columns, the valid inequalities added (if any).

As we will show in the experiments section, our models are able to solve instances of relatively large sizes (say 400 jobs and four machines). However, if one wants to solve problems with more jobs and machines (say 1000 jobs and eight machines), then other types of algorithms are needed. Therefore, in the next section we describe an exact algorithm that uses variants of the MILPs shown above.

5. A mathematical programming based algorithm

Recently, Tran et al. (2016) have published a branch and check decomposition algorithm that proves efficient when solving instances with up to 120 jobs. In their decomposition methods, a master problem, which basically consists of constraints (1) to (6) relaxing X variables and imposing $Y \in \{0, 1\}$, is solved. This solution gives a feasible assignment of jobs to machines. The cycles created in the solutions obtained by this master problem are broken by means of the Concorde TSP solver[1], yielding optimal schedules on each

[1] `http://www.math.uwaterloo.ca/tsp/concorde.html`

machine for the assignments given by the master problem. This algorithm iterates in this way, possibly adding cuts, until optimal solution (cycle-free) is found or a given time limit is reached.

In this section we present an algorithm which takes some of the ideas presented in Tran et al. (2016) with a new methodology and combines them with our MILP models in order to efficiently solve UPMS instances of large sizes. Basically, we use our relaxed model as the master problem, find an assigment of jobs to machine and then find a feasible solution for each machine afterwards, by means of a unique MILP. More specifically, the algorithm works with the following master problem:

$$\begin{aligned} \min\ & C_{\max} \\ \text{s.t.:}& (2), (3), (4), (5), (6) \qquad\qquad \textit{(Master)} \\ & CUTS \\ & X_{ijk} \in [0,1],\ Y_{ij} \in \{0,1\}. \end{aligned}$$

Note that no SEC constraints are added and that X variables are relaxed, whereas Y variables are binary, as opposed to our original MILP models (see Section 4). A solution to $Master$ will be denoted as $(C_{\max}^M, X^M, Y^M)$. Note that this solution gives a feasible *assignment* of jobs to machines by means of Y^M, but not necessarily a feasible *sequence* of jobs in each machine.

In the first iteration of the algorithm, the master problem is solved with $CUTS = \varnothing$, allowing for a 2% gap, similar to the gap used in Tran et al., 2016. Additionally, in this first iteration a maximum CPU time equal to 90% of the total time allowed for the algorithm is imposed. In this way, we ensure that there is time left to find a feasible cycle-free sequence in the remaining 10% of the time. We chose 90% because it yielded good trade-offs between the efficiency and the quality of the solution returned in preliminary experiments. In subsequent iterations the next feasible solution to the master problem will be looked for. In either case (first iteration or following iterations), these solutions yield feasible job-machine assignments, given by the values of variable Y, denoted by Y^M. However, no feasible sequence is guaranteed as the X variables are relaxed and no subtour elimination constraints are included. The integrality of the X variables will be enforced in the next phase of the algorithm, when solving the sequencing problem.

From the assignments Y^M obtained in the master problem, a feasible sequence is built by solving the complete MILP model in which we minimize the sum of the machine completion times using one of our models. Since this assignment is fixed, we define the parameter $y^M = Y^M$, to be used in the next model. We chose the combination MTZ-AM since it yielded the best results (see the experiments section). Therefore, this problem, that we call $Sequencing(y^M)$, results in:

$$
\begin{aligned}
&\min \sum_{i \in M} \sum_{j \in N_0, k \in N, k \neq j} s_{ijk} X_{ijk} + \sum_{j \in N} p_{ij} y_{ij}^M && (15)\\
&\text{s.t.:} (3), (5), (6), (10), (14), (13) \qquad (Sequencing(y^M))\\
&\quad X_{ijk} \in \{0,1\}, U_j \geq 0.
\end{aligned}
$$

It should be noted that in (5) and (6), variables Y_{ij} are substituted by the assignment previously found in the master problem y_{ij}^M. A solution to $Sequencing(y^M)$ will be denoted as $(C_{\max}^*, X^*, y^M)$. Note that this solution is a feasible sequence of jobs in each machine, given by X^*, that is, a feasible solution to the UPMS.

Afterwards, if the solution to *Master* was optimal, cuts are added to the master problem as in Tran et al. (2016), Section 4.4. If N_i^h denotes the set of jobs assigned to machine i in the master problem of iteration h, the proposed cut at iteration h (denoted by $CUT(h)$) is:

$$CUT(h): \quad C_{\max} \geq C_{\max}^{hi*} - \sum_{j \in N_i^h} (1 - Y_{ij}) \theta_{hij}, \tag{16}$$

where $C_{\max}^{hi*}$ is the makespan found in the master problem of iteration h for machine i and $\theta_{hij} = p_{ij} + \max_{k \in N_i^h, k \neq j} \{s_{ikj}\}$ is the sum of the processing time of job j on machine i, plus the maximum setup time between any of the jobs assigned to i at iteration h and job j itself. This cut imposes a lower bound on the makespan in future iterations and is proven to not remove globally optimal solutions, see Tran et al. (2016), Theorem 1. Therefore, the solutions to *Master* are lower bounds to the optimal solution of the global problem.

After updating the cuts of the master problem a new iteration starts unless the best feasible solution found so far is proven to be optimal, or there is no more time left in the algorithm. Note that if the solution to the master problem is optimal, and its value is equal to the value of the best feasible solution found, then such a feasible solution is guaranteed to be optimal and the algorithm stops (as the Master problem gives a lower bound on the optimal solution to UPMS). We denote this algorithm as MPA (Mathematical

Programming based Algorithm). Algorithm 1 presents a pseudocode of MPA.

The main differences between the proposed MPA and the algorithms in Tran et al. (2016) are:

- We first use the master problem, without a lower bound, to find a feasible assignment. The feasible sequence is found using one of the MILPs proposed obtainign a feasible solution for all machines in a single go, instead of repeatedly calling Concorde's TSP solver for each machine. We did not use Concorde because of the huge size of the problems we aim at solving. Note that, in order to transform the sequencing problem of the UPMS into a standard TSP, one needs to triple the number of jobs in the scheduling problem in order to get the number of nodes in the TSP, see Tran et al. (2016). This implies that, for example when dealing with an instance of $n = 1000$ jobs and $m = 2$ machines, each one of the two machines will have to process around 500 jobs. Therefore, for the corresponding sequencing problem one needs to solve a TSP with at least 1500 nodes. Furthermore, this has to be carried out for each machine. While Concorde is able to solve large TSP instances, the CPU times quickly escalate after 500 nodes and make the approach basically intractable for such large instances.

- Our first solution to the master allows a 2% of gap (similar to the LBBD in Tran et al. (2016)) and uses at most, 90% of the total time of the algorithm. In following iterations we find the next solution to the master (like the branch-and-check in Tran et al., 2016).

- We do not calculate a feasible sequence if the solution to the master problem does not have a value lower than the best solution feasible for the UPMS found so far.

- We do not add cuts until the master problem solution is optimal (without a gap). This allows us to move towards good solutions without wasting time trying to improve solutions that are not promising.

6. Computational experiments

Three groups of instances have been used for the computational campaign: small, medium and large. For the small instances, we run our six MILP models, two models introduced in the Appendix, and the best model found in

```
Data: A UPMS problem instance
Set $STOP = False, BestVal = +\infty, TimeLeft =$
  $TotalTime, CUTS = \varnothing, h = 0$
while $STOP = False$ and $Timeleft > 0$ do
    $h = h + 1$;
    if $h = 1$ then
        Solve $Master$ to $GAP \leq 2\%$ or $TimeAvailable = 0.9 Timeleft.$;
    else
        Find the next solution to $Master$ ;
    end
    Let $(C_{\max}^{M}, X^{M}, Y^{M})$ be the solution found ;
    Let $C_{\max}^{hi*}$ be the makespan of machine $i$ in this iteration $h$;
    Set $y^{M} = Y^{M}$ and update $Timeleft$;
    if $C_{\max}^{M} < BestVal$ then
        Solve $Sequencing(y^{M})$ with $TimeAvailable = TimeLeft$;
        Let $(C_{\max}^{*}, X^{*}, y^{M})$ be the solution found;
        $BestVal = \min\{BestVal, C_{\max}^{*}\}$;
        if $(C_{\max}^{M}, X^{M}, Y^{M})$ is optimal in Master then
            $CUTS = CUTS \cup \{CUT(h)\}$
        end
    else
        if $(C_{\max}^{M}, X^{M}, Y^{M})$ is optimal in Master then
            $STOP = True$, optimal solution found;
        end
    end
    Update $Timeleft$.
end
```

Algorithm 1: Pseudocode of MPA. *TimeAvailable* denotes the maximum CPU time given to the solver. Note that the solution to *Sequencing*(y^{M}), $(C_{\max}^{*}, X^{*}, y^{M})$, is a feasible solution to UPMS whereas the solution to *Master*, $(C_{\max}^{M}, X^{M}, Y^{M})$, may not be feasible for the UPMS.

the literature (AAA). Since these instances were not useful for comparing the performance of the different MILP models, their results are not explained here but in the Appendix. For the medium instances, we also run our mathematical-programming-based algorithm (MPA). In order to compare our MPA with the methodology proposed in Tran et al. (2016), we implemented a Branch-And-Check algorithm using our tools: Gurobi instead of SCIP as a solver, and our mathematical programming models instead of Concorde in the sequencing problem. We denote this implementation as B&C. We were forced to carry out our implementation with these changes since we were not able to reproduce the code provided by the authors in Tran et al. (2016) after a large amount of hours devoted to it and frequent communication exchanges with the original authors, who kindly helped us. This implementation is far more similar to MPA and allows for a better comparison between the two methodologies. Besides, as we will see in the results, this implementation yields better results than those reported in Tran et al. (2016). In the large instances, we run the two MILP models that produced the best results in the medium instances (AAA, MTZ-AM), our algorithm MPA and the implementation of the Branch-and-Check methodology of Tran et al. (2016) (B&C).

6.1. Instances generation and experimental setting

The sets of small and large instances have been created for this paper and are available from the authors upon request. In these two sets, the processing times p_{ij} were randomly generated following an integer uniform distribution $U(1, 100)$. The rest of the input data is explained below. The set of medium instances is the same as in Tran et al. (2016), which in turn are obtained from Arnaout et al. (2010). Note that after the communication exchanges with Tony T. Tran, it was found that the results shown in Tran et al. (2016) for the instances of Arnaout et al. (2010) are not correct as these instances contain initial setup times (a setup before the first job in the sequence), which are placed in the diagonals of the setup matrix ($j = k$) but in Tran et al. (2016) it was assumed that these setups where in the first column. We want to underline that, although these results are not correct, the algorithm in Tran et al. (2016) is valid, and the incorrectness of the results comes from a confusion when reading the input data of the instances in Arnaout et al. (2010). Therefore, we had to get a new set of results over the same instances from Tony T. Tran, which we kindly appreciate.

For the small set we have three different factors defining each instance with the following levels: 1) $n \in \{10, 20, 30, 40\}$, 2) $m \in \{2, 4, 6, 8\}$ and 3) Setup times randomly generated following four different integer uniform distributions: $\{U(1,9), U(1,49), U(1,99), U(1,124)\}$. We generated 10 instances of each combination of factors, having in total $10 \times 4 \times 4 \times 4 = 640$ instances. The results over this set of instances is shown in the appendix, as they did not give significant information about the differences in performance between the models and algorithms tested.

In the medium instances the number of jobs is $n \in \{40, 60, 80, 120\}$. The following number of machines were tested: for $n = 40$, $m = 2$, for $n \in \{60, 80\}$, $m \in \{2, 4\}$, for $n = 100$, $m \in \{2, 4, 6\}$, and for $n = 120$, $m \in \{2, 4, 6, 8\}$. 15 replicates are given for each combination, having in total 180 instances. These are the instances employed by Tran et al. (2016).

Finally, for the large instances, we again have all combinations of the following factors: 1) $n \in \{200, 400, 600, 800, 1000\}$, 2) $m \in \{2, 4, 6, 8\}$ and 3) Setup times randomly generated as in the small instances. We generated 10 instances of each combination of factors, having in total $10 \times 5 \times 4 \times 4 = 800$ instances.

The MILP models were run for a maximum CPU time of one hour (3600 seconds) for the small instances, and three hours (10800 seconds), as in Tran et al. (2016), for the medium and large instances. The solver of choice was Gurobi version 7.0.2, because the MILP models tested seem to perform better with Gurobi than with CPLEX, see the Appendix. It must be underlined that such difference is specially significant for the AAA model, which stops being competitive even for medium instances if using CPLEX. Coding is performed with Visual Studio 2015 IDE. The experiments in this paper are carried out on virtual machines with 2 virtual processing cores and 16 GBytes of RAM running Windows 10 Enterprise 64 bits OS. Virtual machines are managed by an OpenStack virtualization platform running on 12 blades, with four 12-core AMD Opteron Abu Dhabi 6344 processors at 2.6 GHz. and 256 GBytes of RAM each. The virtual machines are used in numbers with a random distribution of experiments so as to speed up the completion of all the experimentation without parallel computing.

We employ different performance indicators, but the main one is the average relative percentage deviation (RPD) with respect to the optimum solution or best lower bound found. This is calculated as follows: $RPD = 100\frac{C_{\max}(Model) - LB(Best)}{LB(Best)}$, where $LB(Best)$ is the highest value found for a given instance between the lower bounds or the optimal solution to any of the MILP models and algorithms.

All binaries and detailed results, logs and files are available as accompanying on-line materials.

6.2. Results and discussion

As we mentioned before, small instances do not show significantly different performance when comparing our MILP models and AAA, and are therefore analyzed in the Appendix for the sake of completeness and readability. Moving on to the medium instances, we show in Table 2 the average results over the 180 instances. Column "Algorithm" refers to the MILP model or algorithm tested, column "RPD shows the average relative percentage deviation, column "Opt" refers to the percentage of instances in which the corresponding algorithm found the optimal solution and proved such optimality, and column "Time" shows the average CPU time in seconds. We test our six proposed MILP models, the AAA MILP model, the implementation of the Branch-and-Check methodology of Tran et al. (2016) (B&C) and our Mathematical-Programming-Based algorithm (MPA). The last row corresponds with the data directly provided by Tony T. Tran for these medium instances. Recall that these results are not the same as those reported in Tran et al. (2016) due to the aforementioned problem when considering the initial setup times. Furthermore, they were obtained on an Intel Core i7 CPU running at 3.0 GHz with 12 GBytes of RAM. Note that this CPU is faster that the CPU we use in our tests. The results from Table 2 show that even model AAA of Avalos-Rosales et al.

Algorithm	RPD	Opt%	Time
AAA	0.41	47	6079
DL	0.52	40	7249
DL-AM	0.48	37	7299
DL-KB	0.52	36	7371
MTZ	0.56	36	7411
MTZ-AM	0.47	39	7069
MTZ-KB	0.54	37	7330
B&C	0.23	44	6129
MPA	0.29	44	6106
Tran et al. (2016)*	0.48	0	10 800

Table 2: Summary of average results in the medium instances (times in seconds). * Data provided by Tony T. Tran via private communication.

(2015), with 0.41% of RPD and 47% of optimal solutions, improves results of Tran et al. (2016), with 0.48% of RPD and 0% optimal solutions. Additionally the implementation of the B&C clearly improves on the results of the original implementation of Tran et al. (2016) (corrected results). We manage to obtain, in a slower computer, an RPD of 0.23 compared to the 0.48 supplied by Tony T. Tran. While this seems a small difference in absolute terms, it is quite large as our implementation gives 48.16% lower RPD results. Besides, the results we were provided by Tony T. Tran did not prove optimality in any instance. Note that the average time of this algorithm is 10800 seconds, the maximum. This is due to the fact that no solution returned was proven to be optimal, and therefore the algorithm would not stop until reaching the maximum CPU time available.

As for the models, AAA seems to perform slightly better than ours in these instances (lower RPD and higher optimality rate). Regarding our models, MTZ-AM seems to perform slightly better than the others (lowest average RPD and second highest optimality rate). The tests with medium instances are still not sufficient to check the best method in the comparison as AAA, B&C and MPA are all below 0.5% in RPD. It has to be stressed that these percentage deviations are obtained either from optimum solutions or from the best known lower bounds, which basically means that the algorithms are very close to optimality. Therefore, we had to compare in larger instances.

Table 3 shows the average results over the set of large instances, broken down by n and m values for the two MILP models that yielded the best results in the medium instances: AAA and MTZ-AM. Globally speaking, MTZ-AM performs best in terms of average RPD. However, in the case of 200 jobs, AAA seems to give slightly better results. It is for $n = 400$ that our proposed model MTZ-AM clearly outperforms the AAA model. Furthermore, AAA needs long CPU times (even longer than the allowed 10800 seconds). The reason for this excess in time is that Gurobi cannot be stopped until heuristics and root node are solved and, in many instances, this went over the three-hour CPU time limit. In any case, more than four machines presents a problem for both models, as is the case when $n > 400$. We would like to stress however, that average RPD values of less than 1% from optimum solutions or lower bounds in the UPMS problem, up to 400 jobs and four machines, is a big improvement over the previous recent literature where no more than 60 jobs could be solved to this degree of precision. However, regarding optimality rates, AAA seems to find optimal solutions more often than MTZ-AM. The reader should note that this heavily depends on the solver used, as AAA model yields worse results when using CPLEX (see the

Appendix) in comparison with our MILP models.

		AAA			MTZ-AM		
n	m	RPD	Opt%	Time	RPD	Opt%	Time
200	2	0.00	100	405	0.01	72	4056
	4	0.28	27	8159	0.60	25	8714
	6	1.21	20	9743	2.41	0	10 802
	8	4.74	0	10 857	4.88	0	10 817
Average		1.56	37	7291	1.98	24	8597
400	2	7.14	92	2965	0.09	52	6224
	4	26.41	10	11 439	0.85	10	10 056
	6	552.88	2	13 045	167.18	2	10 558
	8	1796.33	0	14 227	528.92	0	10 803
Average		595.69	26	10 419	174.26	16	9410
Tot. average		298.63	31	8855	88.12	20	9003

Table 3: Summary of results in the large instances for the best proposed MILP model and AAA. Times in seconds.

In order to compare our methodology with that of the Branch-And-Check in Tran et al. (2016), we test both implementations over the large instances (using the same solver, Gurobi, and the same way of finding feasible sequences, model MTZ-AM). The results of these experiments are shown in Table 4. We added to the table three additional columns. "Best" corresponds to the time at which the best feasible solution returned by the algorithm was found. "Master" and "Sched" show the average CPU time in seconds spent solving the master problem and the sequencing problem respectively. We note that both algorithms perform similarly when $n = 200$ in terms of the quality of solution (RPD), although B&C seems to find the best solution a bit faster and has a small RPD advantage. However, for $n \in \{400, 600\}$ we see how our MPA produces much lower average RPD than B&C and also utilizes shorter CPU times. It is worth noting that such RPD is computed against the best lower bound given by any of the algorithms. B&C was not able to cope with instances larger than $n = 400$ and $m = 8$ or $n = 600$ and $m \geq 4$. Still, we tested the proposed MPA algorithm for instances of really large sizes ($n = 800, 1000$), and we observed that the quality of solutions (again measured against the best lower bound) is excellent, always being below 0.8% average

		B&C					MPA				
n	m	RPD	Time	Best	Master	Sched	RPD	Time	Best	Master	Sched
200	02	0.00	406	114	233	172	0.00	345	101	248	97
	04	0.12	6238	2221	6037	201	0.12	5959	1589	5865	95
	06	0.72	8383	3986	8223	160	0.91	8384	1876	8344	40
	08	2.13	10 186	5570	10 086	100	2.68	10 388	2421	10 376	12
	Average	0.74	6303	2973	6145	158	0.93	6269	1497	6208	61
400	02	0.00	1651	1112	710	941	0.00	924	355	665	259
	04	0.09	7337	4301	5905	1432	0.05	7445	3340	7227	217
	06	0.48	10 157	5427	8824	1333	0.40	10 366	3337	10 267	99
	08	66.10	10 807	5985	9721	1086	0.98	10 807	4957	10 726	80
	Average	16.67	7488	4206	6290	1198	0.36	7385	2997	7221	164
600	02	0.00	4172	3610	597	3575	0.00	1292	926	901	390
	04	133.49	8859	7709	5437	3422	0.05	8196	5353	7777	415
	06	282.93	10 139	7122	7252	2887	0.27	10 221	6546	9980	241
	08	454.70	10 815	8350	8206	2606	0.87	10 734	5653	10 603	130
	Average	217.78	8496	6698	5373	3122	0.30	7611	4620	7315	294
800	02						0.00	3036	2802	1628	1407
	04						0.07	9878	7096	9377	485
	06						0.65	10 741	7123	10 420	307
	08						0.78	10 835	4957	10 644	191
	Average						0.37	8623	5494	8017	598
1000	02						0.00	5444	5229	1952	3484
	04						0.17	10 361	7057	9685	673
	06						0.70	10 833	5747	10 483	350
	08						0.76	10 386	6029	9984	272
	Average						0.41	9256	6015	8026	1195
	T. aver.	78.40	7429	4626	5936	1493	0.47	7829	4125	7358	462

Table 4: Summary of results in the large instances for the B&C reimplementation and the proposed MPA. Times in seconds

RPD. It is also interesting to observe that both algorithms spend most of the CPU time solving the master problem. In our MPA, the proportion of time spent on the sequencing problem is around 5.9% of the total time used. Therefore, even if Concorde was a faster option than the adaptation of our MTZ-AM for the sequencing part, the global improvement in either algorithm would be residual as it would affect this 5.9% of the total time used. As a general observation, our proposed MPA is able to generate average relative percentage deviations from lower bounds of 0.41% in the largest instances of 1000 jobs for the UMPS. This significantly improves upon the previous recent results in the literature by Tran et al. (2016) of about 2.48% RPD for $n = 120$.

It is interesting to note the high computational times for the solution of the scheduling problem within the B&C algorithm. This is due to the fact that the differences between B&C and MPA are important, and have an effect in the scheduling problem time. Basically, B&C carries out many more master-scheduling iterations than MPA, as B&C makes every master solution feasible whereas MPA does not call the scheduling subproblem until a 2% gap or lower is reached. This results in the large differences in time in this part of the algorithms.

We observe that the RPD values of B&C are already close to 500% on average for $n = 600$ and 8 machines. For $n = 800$, the RPD values climbed so high that resulted in absurd averages. For $n = 1000$ the algorithm started having memory problems and no feasible solutions were found. Therefore we avoided including the results of the B&C algorithm for $n \in \{800, 1000\}$.

7. Conclusions and future research

In this paper we have studied the unrelated parallel machine scheduling problem with sequence dependent setup times (UPMS). We have proposed improvements on existing mathematical formulations, based on adaptations of subtour elimination constraints. More precisely, we have modeled the UPMS as a particular heterogeneous m-TSP. Furthermore, in order to accelerate the solution of the proposed models, we have also proposed two sets of valid inequalities that proved useful in the experiments. Thanks to these improvements, our mixed integer linear programs (MILP) are able to give near-optimal solutions to instances of up to 400 jobs and four machines in less than three hours. The best MILP model proposed in the literature, Avalos-Rosales et al. (2015) and referred to as AAA, has also been tested in this paper, and is able to cope with instances of up to 200 jobs and eight machines. In any case, even the best model combined with the most effective solver is unable to cope with larger instances, with a limit that surfaces around 400 jobs and six machines. For this reason, we have developed a mathematical-programming-based algorithm (MPA) that uses the MILP models we introduced before in a decomposition approach. This algorithm is able to obtain solutions that are close to optimality all the way up to 1000 jobs and eight machines, which is a significant leap forward compared to the existing literature on this problem (60 jobs for MILP models and 120 jobs for exact algorithms). Furthermore, we have observed that a large percentage of the feasible solutions provided by MPA when the time limit is reached are indeed optimal, but MPA is not able to prove this optimality for such large instances in the three-hour CPU time limit. Another important conclusion reached in this paper relates to the choice of a suitable solver. An updated solver is able to solve instances of up to 200 jobs for the AAA model of Avalos-Rosales et al. (2015), up from the 60 jobs of the original authors just two years before the writing of this paper. The same can be said about our reimplementation of the B&C of Tran et al. (2016). We have been able to solve instances of up to 600 jobs with an efficient reimplementation. Comparatively, the original authors were only able to solve instances of up to 120 jobs with faster computers and yet obtained worse results. It is clear that the choice of solver and version goes a long way in relation to efficiently solving the UPMS. As shown in the Appendix, we conclude that Gurobi is more suitable than CPLEX for the MILP models analyzed in this paper.

Avenues for future research could go in multiple directions. Objectives different to makespan are interesting for practical reasons. Additional production resources (like personnel) are as limited as machines and need assignment as well. Additionally, more constraints and practical situations could be added to the problem such as release dates, overlaps and waiting times etc.

Acknowledgments

The authors are partially supported by the Spanish Ministry of Economy and Competitiveness, under the project “SCHEYARD – Optimization of Scheduling Problems in Container Yards” (No. DPI2015-65895-R) financed by FEDER funds. Special thanks are due to our colleague Alfredo MarÃn, for fruitful discussions that helped improve this paper and to Tony T. Tran and coauthors for all the help received during the reimplementation of their efficient methods.

References

Al-Salem, A. (2004). Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times. *Engineering Journal of the University of Qatar*, 17:177–187.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Arnaout, J., Rabadi, G., and Musa, R. (2010). A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 21(6):693–701.

Avalos-Rosales, O., Angel-Bello, F., and Alvarez, A. (2015). Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. *International Journal of Advanced Manufacturing Technology*, 76(9-12):1705–1718.

Balakrishnan, N., Kanet, J. J., and Sridharan, S. V. (1999). Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers and Operations Research*, 26(2):127–141.

Bektas, T. (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209–219.

Desrochers, M. and Laporte, G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36.

Diana, R. O. M., F., d. M., de Souza, S. R., and de Almeida Vitor, J. F. (2014). An immune-inspired algorithm for an unrelated parallel machines' scheduling problem with sequence and machine dependent setup-times for makespan minimisation. *Neurocomputing*, 163:94–105.

Fanjul-Peyro, L., Perea, F., and Ruiz, R. (2017). Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, 260(2):482–493.

Fanjul-Peyro, L. and Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69.

Fanjul-Peyro, L. and Ruiz, R. (2011). Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research*, 38(1):301–309.

Glass, C., Potts, C., and Shade, P. (1994). Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*, 20(2):41–52.

Graham, R. L., Lawler, E., Lenstra, J., and Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326.

Guinet, A. (1991). Textile production systems: a succession of non-identical parallel processor shops. *The Journal of the Operational Research Society*, 42(8):655–671.

Helal, M., Rabadi, G., and Al-Salem, A. (2006). A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. *International Journal of Operations Research*, 3(3):182–192.

Kara, I. and Bektas, T. (2006). Integer linear programming formulations of multiple salesman problems and its variations. *European Journal of Operational Research*, 174(3):1449–1458.

Koç, C., Bektas, T., Jabali, O., and Laporte, G. (2016). Thirty years of heterogeneous vehicle routing. *European Journal of Operational Research*, 249(1):1–21.

Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulations and traveling salesman problems. *Journal of Association for Computing Machinery*, 7(4):326–329.

Pinedo, M. L. (2005). *Planning and Scheduling in Manufacturing and Services*. Series in Operations Research. Springer, New York.

Rabadi, G., Moraga, R., and Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing*, 17(1):85–97.

Sundar, K. and Rathinam, S. (2015). An exact algorithm for a heterogeneous, multiple depot, multiple traveling salesman problem. In *Proceedings of the International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 366–371, Denver, Colorado, USA.

Tran, T. T., Araujo, A., and Beck, J. C. (2016). Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing*, 28(1):83–95.

Tran, T. T. and Beck, J. C. (2012). Logic-based benders decomposition for alternative resource scheduling with sequence dependant setups. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 774–780, Montpellier, France.

Vallada, E. and Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3):611–622.

Vallada, E. and Ruiz, R. (2012). Scheduling unrelated parallel machines with sequence dependent setup times and weighted earliness-tardiness minimization. In Ríos-Mercado, R. and Ríos-Solís, Y., editors, *Just-in-Time Systems*. Springer.

Wang, L., Wang, S., and Zheng, X. (2016). A hybrid estimation of distribution algorithm for unrelated parallel machine scheduling with sequence-dependent setup times. *IEEE/CAA Journal of Automatica Sinica*, 3(3):235–246.

Yilmaz Eroglu, D., Ozmutlu, H. C., and Ozmutlu, S. (2014). Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times. *International Journal of Production Research*, 52(19):5841–5856.

Yin, Y., Cheng, S.-R., Cheng, T. C. E., Wang, D.-J., and Wu, C.-C. (2016). Just-in-time scheduling with two competing agents on unrelated parallel machines. *Omega*, 63:41–47.

Yin, Y., Wang, Y., Cheng, T. C. E., Liu, W., and Li, J. (2017). Parallel-machine scheduling of deteriorating jobs with potential machine disruptions. *Omega*, 69:17–28.

8. Appendix

In this appendix we show two other MILP models that we have tested during our research, both based on previous research. We decided not to include them in the paper as they yield poor results, as we will see in the experiments performed over small instances.

8.1. A Standard Model

The model in Vallada and Ruiz (2011) can be considered as the standard one, and is detailed here for the sake of completeness. This model, denoted as ST, uses the following variables:

- $X_{ijk} = 1$ if k is the successor of j on machine i, zero otherwise.
- $C_{ij} \geq 0$ is the completion time of job j on machine i.

- $C_{\max}$ is the maximum completion time (makespan).

From these variables this model consists of:

$$
\begin{aligned}
&\min C_{\max} \\
&\text{s.t.} \sum_{i\in M}\sum_{\substack{j\in N_0\\ j\neq k}} X_{ijk} = 1,\ k\in N && (17)\\
&\sum_{i\in M}\sum_{\substack{k\in N\\ j\neq k}} X_{ijk} \leq 1,\ j\in N && (18)\\
&\sum_{k\in N} X_{i0k} \leq 1,\ i\in M && (19)\\
&\sum_{\substack{\ell\in N_0\\ \ell\neq k,\ell\neq j}} X_{i\ell j} \geq X_{ijk},\ j,k\in N, j\neq k, i\in M && (20)\\
&C_{ik} + V(1-X_{ijk}) \geq C_{ij} + s_{ijk} + p_{ik},\ j\in N_0, k\in N, j\neq k,\ i\in M && (21)\\
&C_{i0} = 0,\ i\in M && (22)\\
&C_{\max} \geq C_{ij},\ j\in N,\ i\in M && (23)\\
&X_{ijk}\in\{0,1\}, C_{ik}\geq 0.
\end{aligned}
$$

where V is a sufficiently large constant. (17) ensures that every job k has a predecessor in N_0, and is processed on one machine in M. (18) ensures that every job j has at most one successor in N and is processed on, at most, one machine in M. (19) imposes that, for every machine in M, there is at most one job in N that is the first to be processed. (20) ensures that for every machine in M, if j is a predecessor of k, both in N, then j must have a predecessor on this machine which could be the dummy job. (21) imposes that, if j and k are successive, then the completion time of job k is at least the completion time of job j, plus the setup time between j and k, plus the processing time of job k. (22) imposes that the completion time of the dummy job is 0 on any machine. Finally, (23) ensures that the makespan is not lower than any of the jobs' completion times.

8.2. A two-index model

We now adapt the model in Balakrishnan et al. (1999), originally proposed for a different but related problem, to the UPMS. It is a two-index model, as opposed to the other MILP models we consider in this paper. Three new sets of variables are needed:

- $X_{jk} = 1$ if k is processed after j (not necessarily an immediate successor).

- $Y_{ij} = 1$ if j is processed on machine i, zero otherwise.

- $C_j \geq 0$ is the completion time of job j.

From these variables, this model that we denote as BL, consists of:

$$\min C_{\max}$$

$$\text{s.t.} \sum_{i \in M} Y_{ij} = 1, \; j \in N \tag{24}$$

$$Y_{ij} + \sum_{\substack{i' \in M \\ i' \neq i}} Y_{i'k} + X_{jk} \leq 2, \; 1 \leq j < n, k > j, \; i \in M \tag{25}$$

$$C_k - C_j + V(3 - X_{jk} - Y_{ij} - Y_{ik}) \geq p_{ik} + s_{ijk}, \quad 1 \leq j < n, k > j, \; i \in M \tag{26}$$

$$C_j - C_k + V(2 + X_{jk} - Y_{ij} - Y_{ik}) \geq p_{ij} + s_{ikj}, \quad 1 \leq j < n, k > j, \; i \in M \tag{27}$$

$$C_j \geq p_{ij} Y_{ij}, \; j \in N, i \in M \tag{28}$$

$$C_{\max} \geq C_j, \; j \in N \tag{29}$$

$$X_{jk} \in \{0,1\}, Y_{ij} \in \{0,1\}, C_j \geq 0.$$

(24) ensures that every job in N is processed on exactly one machine. (25) imposes that, if a job j is processed on i and k is its successor, then k cannot be processed on another machine that is not i. (26) and (27) control the completion times of any pair of jobs. Note that these constraints are quite convoluted but it suffice to say that they have to be satisfied for the jobs that follow a sequence on a machine. Particular to the BL model, setup times must satisfy the triangular inequality. This is not the case for the ST model.

8.3. Results over small instances

Table 5 shows the average results for the small instances. Since we noted that the solver used may significantly affect the relative performance of the models, we decided to test another solver: CPLEX 12.7.0. We chose Gurobi and CPLEX because they are arguably the most common solvers used in industry and academia, and they are also the best performers according to the updated results of Hans Mittelmann (`http://plato.asu.edu/ftp/milpc.html`). We show the average RPD across the 640 small instances, the percentage of optimal solutions found ($OPT\%$) and the average CPU time used by each method (in seconds). Note that due to space considerations it is not possible the break down all results by n and m values as the resulting tables are excessively large. The complete results in Excel spreadsheets are available as accompanying online materials. We observe that both models ST and BL obtain much poorer results than AAA and our six models. Therefore, ST and BL are discarded and will not be tested in the other sets of instances. We also see that the performance of AAA and our six models is similar, both in terms of the quality of solution (RPD), percentage of optimal solutions ($OPT\%$) and speed (Time). We also note that Cplex performs a little better in

	CPLEX			Gurobi		
Model	RPD	$OPT\%$	Time	RPD	$OPT\%$	Time
ST	31.78	30.94	2527.43	34.39	32.50	2483.17
BL	4.32	48.91	1955.78	6.47	50.31	1910.22
AAA	0.17	94.06	340.14	0.13	94.69	297.82
DL	0.13	95.16	283.98	0.15	94.38	286.06
DL-AM	0.13	95.78	268.23	0.15	94.84	291.96
DL-KB	0.13	95.31	275.50	0.16	94.38	320.38
MTZ	0.14	95.47	280.36	0.16	94.53	293.94
MTZ-AM	0.15	95.47	284.44	0.17	94.22	309.76
MTZ-KB	0.14	95.47	290.16	0.16	93.75	340.85

Table 5: Summary of results for the small instances (times in seconds).

our six models, and Gurobi performs a little better in the AAA model. In any case, it seems that the small instances are not large enough to discern between AAA and our proposed models or between CPLEX and Gurobi.

8.4. Results over medium instances: CPLEX vs. Gurobi

In Table 6 we see the comparison between CPLEX and Gurobi over the medium instances, when solving model AAA, our models proposed in this paper, our reimplementation of the B&C of Tran et al. (2016), our MPA, and the results provided by Tony T. Tran via private communication (different from those in Tran et al. (2016)). We here stress the differences observed between the two solvers employed. Most algorithms and models, and particularly AAA, DL and MTZ, show in comparable CPU times much better performance in Gurobi compared to CPLEX. In total CPLEX obtains an average RPD of 3.75% vs. 0.45% of Gurobi. In a quick ANOVA experiment this difference is statistically significant ($p - value = 0.0065$). Although not detailed here, for larger instances CPLEX even fails to even provide feasible solutions. This motivates the choice of Gurobi for the remainder of this paper.

	CPLEX		Gurobi	
Model	RPD	Time	RPD	Time
AAA	7.28	6872.50	0.41	6078.67
DL	12.31	7194.53	0.52	7248.96
DL-AM	1.80	7209.79	0.48	7298.79
DL-KB	1.19	6984.65	0.52	7371.38
MTZ	4.44	7255.06	0.56	7411.06
MTZ-AM	1.61	6951.35	0.47	7069.08
MTZ-KB	1.09	6930.56	0.54	7329.78
B&C	0.32	6339.99	0.23	6128.67
MPA	0.35	6332.68	0.29	6106.18
	SCIP + Concorde			
Tran et al. (2016)	0.48	10 800.00		

Table 6: Summary of results in the medium instances (times in seconds).