Highlights

- We propose new arc-time index formulations for scheduling with periodic maintenances
- We devise a simple iterative exact algorithm that yields high quality results
- All existing benchmark instances are solved to optimality
- New instances are introduced and most are solved optimally

Exact solution of the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times

Vitor Nesello[a], Anand Subramanian[b], Maria Battarra[c,*], Gilbert Laporte[d]

[a] *Universidade Federal da Paraíba, Departamento de Engenharia de Produção, Brazil*
[b] *Universidade Federal da Paraíba, Departamento de Sistemas de Computação, Brazil*
[c] *University of Bath, School of Management, BA2 7AY, Bath, United Kingdom*
[d] *CIRRELT and HEC Montréal, Montréal, Canada H3T 2A7*

Abstract

The single-machine scheduling problem with periodic maintenances and sequence-dependent setup times aims at scheduling jobs on a single machine in which periodic maintenances and setups are required. The objective is the minimization of the makespan. We propose an exact algorithm based on the iterative solution of three alternative arc-time-indexed models. Extensive computational experiments are carried out on 420 benchmark instances with up to 50 jobs, and on 405 newly proposed instances involving up to 150 jobs. We compare the results found by all formulations with those obtained by the best available mathematical formulation. All instances from the existing dataset are solved to optimality for the first time.

Keywords: Combinatorial Optimisation, Scheduling, Periodic Maintenances, Makespan, Arc-time-indexed formulation

1. Introduction

The single-machine scheduling problem with periodic maintenances and sequence-dependent setup times (1MPS) aims at scheduling a set J of jobs on a single machine, assuming the machine requires both periodic maintenances and setup activi-

*Corresponding author: Maria Battarra. School of Management, University of Bath, UK. *E-mail addresses:* m.battarra@bath.ac.uk, maria.battarra@gmail.com

ties. Scheduled maintenances take place exactly every P units of time until every job $j \in J$ is completed, and the time required for a maintenance service is p_0 (hereafter the index 0 refers to a maintenance). Once a job or a maintenance $i \in J_+ = J \cup \{0\}$ has been completed, the machine has to be reconfigured to perform another job or a maintenance $j \in J_+$. This activity is referred to as a setup and necessitates $s_{ij} \geq 0$ time units. Examples from the literature involving setup times are (Allahverdi et al., 2008; Liao et al., 2016; Herr and Goel, 2016; Subramanian et al., 2014). During maintenance and setup activities the machine cannot process any job and preemption is not allowed, that is, jobs are not resumable. Each job $i \in J$ has an associated processing time $p_i > 0$. The objective is the minimization of the makespan, i.e., the time C_{max} at which the last processed job is completed. Figure 1 shows an example of solution of a 1MPS instance containing five jobs. Processing times and maintenance durations are represented by white rectangles, setups are hatched rectangles.

Figure 1: Example of a solution of a 1MPS instance with 5 jobs.

According to the three-field notation by Graham et al. (1979), 1MPS can be denoted as $1|pm, s_{ij}|C_{max}$ (Lee and Kim, 2012), which is clearly $\mathcal{NP}$-hard since it includes problem $1|s_{ij}|C_{max}$ as special case when $pm = 1$. Large instances of the $1|s_{ij}|C_{max}$ can be solved to optimality by reducing the problem to the symmetric traveling salesman problem and using, for example, *Concorde* (Applegate et al., 2006). The introduction of periodic maintenances makes the problem much more difficult to solve. In fact, the makespan minimization makes 1MPS equivalent to a heterogeneous fleet vehicle scheduling problem with a time limit P. In this case, two vehicle types are considered: type I has a fixed cost P and no variable cost, while type II has no fixed cost, but has a variable cost equal to the time consumption in the route. One vehicle of type II is available.

1MPS was introduced by Ángel Bello et al. (2011a) who motivated the practical relevance of the problem by citing industrial applications in the textile, manufacturing of printed circuit boards and chemical industries, in which machines necessitate

periodic maintenances. The authors referred to an increasingly vast body of literature for scheduling problems with maintenances.

We would like to mention the relevance and importance of the 1MPS not just in the operational context of job sequencing on a single machine, but also in the tactical context of quantifying the costs of outsourcing production or delivery operations. Assuming that maintenances model the time at which employers or machines are most typically idle (i.e., from 17:00 until 9:00 of the subsequent day), a 1MPS solution quantifies the time needed to complete a set of jobs, including setup times or changeover durations from a job to the next.

Single-machine scheduling problems with periodic maintenances have recently become popular in the scheduling community, given their combinatorial complexity and relevance in real-world applications. For the sake of brevity, Table 1 summarizes the most recent contributions in the literature about the 1MPS and related variants for single-machine scheduling problems with maintenance activities. Here we review the three papers with minimum makespan and setup times, which correspond to the problem considered in our study.

Ángel Bello et al. (2011a) proposed a mixed integer linear programming (MILP) formulation with polynomial number of variables and constraints (i.e., a compact formulation) for the 1MPS, strengthened the formulation with valid inequalities and initialized the exact algorithm with upper bounds obtained using a heuristic algorithm. The same authors tackled the problem heuristically in Ángel Bello et al. (2011b), using a GRASP. Finally, Pacheco et al. (2013) obtained higher quality results for the problem with a multi-start tabu search (TS) metaheuristic and an enhanced version of the compact formulation (CF) presented in Ángel Bello et al. (2011a). To the best of our knowledge, CF appears to be the most effective MILP for the 1MPS. However, it still fails to solve some instances involving 20 jobs.

The main contribution of this paper is to present mathematical models for the 1MPS and to develop an iterative exact algorithm that is simple to implement and yields high quality results. The algorithm iteratively solves a MIP formulation of a restricted version of the 1MPS in which the number of maintenances is fixed. Secondary

Table 1: Overview of the problem variants

Authors	Objective	Characteristics	Algorithms
Liao and Chen (2003)	Min max tardiness	Due dates	B&B, heuristic
Sbihi and Varnier (2008)	Min max tardiness	Flexible maint.	B&B, heuristic
Chen (2006a)	Min total flow time	-	B&B, heuristic
Chen (2006b)	Min mean flow time	Flexible maint.	MILP, heuristic
Chen (2007)	Min total flow time and max tardiness	-	B&B, heuristic
Chen (2009)	Min # tardy jobs	-	B&B, heuristic
Lee and Kim (2012)	Min # tardy jobs	-	MILP, heuristic
Ji et al. (2007)	Min makespan	-	Approximation alg.
Chen (2008)	Min makespan	Flexible maint.	MILP, heuristic
Xu et al. (2009)	Min makespan	Flexible maint.	Approximation alg.
Low et al. (2010)	Min makespan	-	Ant colony
Hsu et al. (2010)	Min makespan	Max # jobs	Heuristics
Ángel Bello et al. (2011a)	Min makespan	Setup times	B&B, heuristic
Ángel Bello et al. (2011b)	Min makespan	Setup times	GRASP
Pacheco et al. (2013)	Min makespan	Setup times	MILP, tabu search
Zade and Fakhrzad (2013)	Min makespan	-	MILP, genetic alg.
Yu et al. (2014)	Min makespan	-	Approximation alg.
Cui and Lu (2014)	Min makespan	Release dates	B&B

contributions of this article are i) to test the effectiveness of using arc-time-indexed formulations embedded within an iterative algorithm for the 1MPS, in which the number of maintenances is fixed at each iteration, ii) to develop an enhanced and innovative arc-time-indexed formulation for the 1MPS, iii) to provide new best-known solutions for several 1MPS benchmark instances, and iv) to generate and solve new and more challenging instances for the 1MPS.

The remainder of the paper is organized as follows. Section 2 presents the formulation of Pacheco et al. (2013). Section 3 describes the proposed arc-time-indexed mathematical models. Section 4 introduces the iterative algorithm. Section 5 compares the performance of the three models within (and, when possible, without applying) the iterative framework and provides extensive computational results highlighting which is the most adequate arc-time-indexed formulation. Moreover, our results are compared with those of Pacheco et al. (2013). Conclusions follow in Section 6.

2. The compact formulation (CF) of Pacheco et al. (2013)

In CF a *block* is any time interval between consecutive maintenances. Binary variables v^1_{ij} and v^2_{ij} set precedence relationships among jobs. More precisely, v^1_{ij} is equal to 1 if and only if job j is scheduled after job i in the last block. Similarly, v^2_{ij} is equal to 1 if and only if job j is scheduled after job i in any block excluding the last one. Variable y_j is equal to 1 if and only if j is processed in the last block and the number of blocks is m. Finally, u_j is the completion time of job j with respect to the beginning of the block in which j is processed. The MILP is as follows:

$$\text{minimize} \quad P(m-1) + \sum_{i=0}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} (s_{ij} + p_j) v^1_{ij} \tag{1}$$

subject to

$$\sum_{j \in J} v^1_{0j} = \sum_{i \in J} v^1_{i0} = 1 \tag{2}$$

$$\sum_{j \in J} v^2_{0j} = \sum_{i \in J} v^2_{i0} = m - 1 \tag{3}$$

$$\sum_{\substack{j \in J_+ \\ i \neq j}} v^1_{ij} = y_i \qquad i \in J \tag{4}$$

$$\sum_{\substack{i \in J_+ \\ i \neq j}} v^1_{ij} = y_j \qquad j \in J \tag{5}$$

$$\sum_{\substack{j \in J_+ \\ i \neq j}} v^2_{ij} = 1 - y_i \qquad i \in J \tag{6}$$

$$\sum_{\substack{i \in J_+ \\ i \neq j}} v^2_{ij} = 1 - y_j \qquad j \in J \tag{7}$$

$$\begin{aligned} &u_i - u_j + (P + s_{ij} + p_j)(v^1_{ij} + v^2_{ij}) \\ &+ (P - s_{ji} - p_i)(v^1_{ji} + v^2_{ji}) \leq P \qquad i, j \in J, j \neq i \end{aligned} \tag{8}$$

$$(s_{0i} + p_i)(v^1_{0i} + v^2_{0i}) \leq u_i \leq P - (s_{i0} + p_0)(v^1_{i0} + v^2_{i0}) \qquad i \in J \tag{9}$$

$$\sum_{i\in J_+}\sum_{\substack{j\in J_+\\ i\neq j}}(p_i+s_{ij})v_{ij}^2 \leq P\times m \tag{10}$$

$$\sum_{i\in J_+}\sum_{\substack{j\in J_+\\ i\neq j}}(p_i+s_{ij})v_{ij}^1 \leq P \tag{11}$$

$$v_{ij}^k \in \{0,1\} \qquad k\in\{1,2\}$$

$$i,j\in J_+, j\neq i \tag{12}$$

$$u_i, y_i \geq 0 \qquad i\in J \tag{13}$$

$$m \geq 0. \tag{14}$$

The objective function (1) computes the makespan as the sum of the block durations (except the last one) and the time instant at which the machine finishes processing the last job in the final block. Constraints (2) and (3) state that only one job can be processed after and before a maintenance activity. Constraints (4)–(7) impose precedence conditions. Constraints (8) set the maximum block duration, and Constraints (9) define the job processing and setup times. Constraints (12) and (13) specify the domains of the variables.

We attempted to strengthen CF using a subset of the so-called *k-path cuts*, which were introduced by Kohl et al. (1999) for the vehicle routing problem with time windows. Let $K(S)$ be the minimum number of blocks required to schedule a subset of jobs $S\subset J$; the cuts state that $\sum_{i\in J\setminus\{S\}}\sum_{j\in S}(v_{ij}^1+v_{ij}^2)\geq K(S)$. Computing the value of $K(S)$ is $\mathcal{NP}$-hard as one needs to solve the $1|s_{ij}|C_{max}$ over S. When $K(S)=2$, the inequalities are referred to as *2-path cuts*, and it is typically within computational reach to separate the cuts exactly when $|S|$ is small. We implemented a cutting plane-based algorithm using the separation procedure presented in Kohl et al. (1999) for the 2-path cuts. Our tests revealed that the cuts improved our solutions for very few instances at the expense of increasing the overall average CPU time. We therefore decided to report the results for the CF without cuts.

3. Arc-time-indexed models

As shown by Pacheco et al. (2013), the 1MPS can be formulated as a MILP with a polynomial number of variables and constraints; this type of model can be solved directly using a commercial solver. However, the model of Pacheco et al. (2013) fails to solve some instances involving as few as 20 jobs. In what follows, we present three alternative models for the 1MPS. While all of our models formally describe the 1MPS, the first two cannot be used as inputs for a commercial solver because the number of maintenances m necessary to complete all jobs is one of the indices of some constraints. In Section 4, we explain how all models can be solved by means of an iterative procedure.

The models we propose are all based on arc-time-indexed variables, which link jobs to an instant in the time horizon, for example by specifying whether an arc is traversed by starting or ending at a certain time. The number of variables is typically proportional to the length of the time horizon (it is pseudo-polynomial), and is therefore arbitrarily large. The success of formulations based on arc-time-indexed variables is due to the strength of their linear relaxation, compared with formulations that contain a polynomial number of variables, whereas their weakness lies in their size. These models belong to the class of *time-index formulations*. The first time-index formulation for a single machine scheduling problem was proposed by Dyer and Wolsey (1990). Since then, research has focused on the adoption of time-index formulations solved by algorithms that can mitigate the explosion of the size of the search space.

Examples of successful methods based on arc-time-indexed formulations are those of Sourd (2009), Tanaka et al. (2009) and Pessoa et al. (2010). Nogueira et al. (2014) compared the performance of formulations for single-machine scheduling problems with sequence-dependent setup times and release dates; the results of their tests confirm that the arc-time-indexed formulation produces high quality lower bounds and outperforms other formulations for instances with up to 15 jobs. However, larger instances are more often solved to optimality when modelled by two-index formulations. More recently, Boland et al. (2016) proposed the use of *time buckets* to reduce the size

of the search space in single-machine scheduling problems. Significant improvements can thus be achieved when processing times have small and clustered values.

In what follows, we denote by T_{ij} the set of times at which job or maintenance j can be scheduled immediately after job or maintenance i without violating any constraint, i.e., a job j cannot be scheduled after a job i if the start time of j plus its processing time p_j overlaps with any of the maintenance activities. Note that the values T_{ij} depend on the choice of arc-time formulation.

3.1. ATF

Our first model is called ATF for arc-time formulation. Define the set $J_{ATF} = J \cup \{0, M_1, \ldots, M_{m-1}\}$ in which 0 is the initial and final state of the machine and $M_1, \ldots, M_{m-1}$ are the maintenances to be performed. Binary decision variable x_{ij}^t equals 1 if and only if $j \in J_{ATF}$ is processed exactly after $i \in J_{ATF}$ at time $t \in T_{ij}$. Let $\mathbf{x}$ be the (x_{ij}^t) vector. Since the maintenance activities have fixed starting times and jobs are not resumable, idle time may occur. In order to account for this possibility, we allow variables x_{jj}^t to exist. Such variables mean that the machine is idle between times $t-1$ and t and no setup is executed, thus $s_{jj} = 0, j \in J$. For ease of representation, we define variables p'_{ij} equal to 1 if $j = i$ and equal to p_j otherwise, and we assume the schedule starts and returns to maintenance 0 after completing all jobs. The ATF is as follows:

$$\underset{m,\mathbf{x}}{\text{minimize}} \quad \sum_{i \in J} \sum_{t \in T_{i0}} (t - s_{i0}) x_{i0}^t \tag{15}$$

subject to

$$\sum_{j \in J} x_{0j}^{s_{0j}} = 1 \tag{16}$$

$$\sum_{\substack{i \in J_{ATF} \\ i \neq j}} \sum_{t \in T_{ij}} x_{ij}^t = 1 \qquad j \in J_{ATF} \tag{17}$$

$$\sum_{\substack{j \in J_{ATF} \\ t \in T_{ji}}} x_{ji}^t - \sum_{\substack{j \in J_{ATF} \\ t' \in T_{ij} \\ t' = t + p'_{ij} + s_{ij}}} x_{ij}^{t'} = 0 \qquad i \in J_{ATF}, t = 0, \ldots, mP \tag{18}$$

$$x_{ij}^t \in \{0,1\} \qquad i,j \in J_{ATF}, t \in T_{ij} \tag{19}$$

$$m \geq 0. \tag{20}$$

The objective function (15) computes the makespan by minimising the time at which the last job is completed, in other words, the time at which the path returns to the dummy maintenance 0. Constraint (16) states that exactly one job must be processed immediately after the maintenance 0. Constraints (17) force every job and each maintenance to be performed exactly once. Constraints (18) ensure flow conservation. Note that these constraints also ensure that each maintenance activity will be scheduled at the correct time. In addition, the number of constraints and variables of each constraint depends explicitly on m (this is why the model cannot be solved by a commercial solver). Constraints (19) and (20) define de domains of the variables.

ATFs yield a suitable graphical representation when the vertices of the network are plotted against time. Such a representation depicts a sequence of jobs over time as a path in a network and provide an effective visual interpretation of the size of the model, as well as that of the solutions. An example of this representation is given in Figure 2. Table 2 contains the data of the instance used in our example, in which the block duration P is set to eight time units, and the maintenance takes one time unit. Four jobs have to be scheduled and the tables provide the setup times and the processing times. Note that setup times are also necessary if a job is executed immediately after the initial state 0, after a maintenance $M_i, i = 1, \ldots, m,$, or if a job is executed immediately before a maintenance (or before the final state 0). In this latter case, the maintenance and eventually the idle time do not affect the makespan, but determine the feasibility of the last block. Figure 2 depicts the ATF network representing the solution corresponding to the sequence of jobs $(3, 2, 1, 4)$. The vertical lines correspond to setup times, while the horizontal lines are either processing or idle times.

Table 2: Instance of 1MPS with $P = 8$

Setup times						Processing times	
	$0/M_i$	1	2	3	4	j	p_j
$0/M_i$	0	2	1	2	1	p_0/p_{M_i}	1
1	1	0	2	3	1	1	1
2	1	1	0	3	3	2	1
3	2	2	1	0	2	3	2
4	1	2	4	2	0	4	1

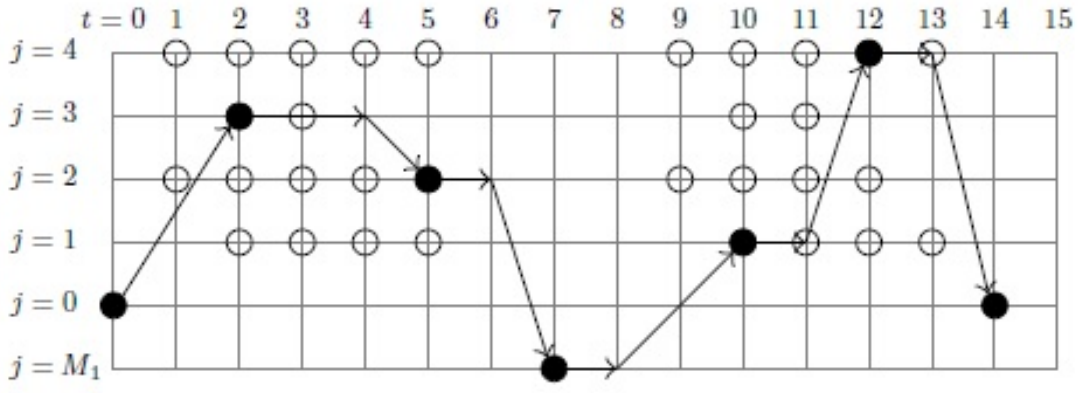


Figure 2: ATF network and solution representation.

3.2. ATF-M

The second model is called ATF-M ("M" stands for the maintenance index) and the main characteristic is that variables make use of an additional index to identify blocks. Let $K = \{1, \ldots, m\}$ be the set of blocks. Binary variable z_{ij}^{tk} is equal to 1 if and only if job $j \in J$ starts immediately after job $i \in J_+$ on block $k \in K$ at time $t \in T_{ij}$. Note that idle times do not need to be modelled as in ATF because jobs are allowed to return to 0 before the scheduled maintenance. Variables z_{jj}^{tk} are therefore not required. The ATF-M is as follows:

$$\text{minimize} \quad P(m-1) + \sum_{i \in J} \sum_{t \in T_{i0}} (t - s_{i0}) z_{i0}^{tm} \tag{21}$$

subject to

$$\sum_{k \in K} \sum_{i \in J_+} \sum_{t \in T_{ij}} z_{ij}^{tk} = 1 \qquad j \in J \tag{22}$$

$$\sum_{t \in T_{ij}} \sum_{j \in J} z_{j0}^{tk} = \sum_{j \in J} z_{0j}^{s_{0j},k} = 1 \qquad k \in K \tag{23}$$

$$\sum_{\substack{j \in J_+ \\ t \in T_{jik}}} z_{ji}^{tk} - \sum_{\substack{j \in J_+ \\ t' \in T_{ij} \\ t' = t + p_i + s_{ij}}} z_{ij}^{t'k} = 0 \qquad k \in K, i \in J$$

$$t = 0, \ldots, P - p_0 \tag{24}$$

$$z_{ij}^{tk} \in \{0, 1\} \qquad i, j \in J_+, i \neq j$$

$$k \in K, t \in T_{ij} \tag{25}$$

$$m \geq 0. \tag{26}$$

Constraints (22) mean that every job is executed in one block. Constraints (23) state that, for each block $k \in K$, one job must be performed first (after 0). Constraints (24) ensure flow conservation. Note that both the sets of Constraints (23) and (24) are indexed in m, and therefore, as for the previous model, ATF-M cannot be solved by a commercial solver. We do not provide a graphical interpretation of ATF-M solutions because a visual representation of a four-index variables model would be

too complex and uninformative.

3.3. ATF-P

Our final model, called ATF-P, considers a single block of $P-p_0$ time units (hence the "P"). The decision variables are the same as in ATF. The main difference between ATF and ATF-P is the length of the time interval, which is restricted to $P-p_0$ time units in the ATF-P, in comparison to mP units in ATF. In order to reduce the time horizon to $P-p_0$ units of time and keep track of the makespan, m simultaneous but distinct paths have to be identified. The first $m-1$ paths start and end at the dummy maintenance 0, the last path starts and ends at the dummy maintenance $0'$. The latter path represents the final block, therefore, $J_{ATF\text{-}P} = J \cup \{0, 0'\}$. As in ATF-M, the variables x^t_{jj} are not necessary to model idle time. The ATF-P model is as follows:

$$\text{minimize} \quad P(m-1) + \sum_{\substack{i \in J \\ t \in T_{i0'}}} (t - s_{i0'}) x^t_{i0'} \tag{27}$$

subject to

$$\sum_{j \in J} x^{s_{0j}}_{0j} = m - 1 \tag{28}$$

$$\sum_{j \in J} x^{s_{0'j}}_{0'j} = 1 \tag{29}$$

$$\sum_{\substack{i \in J_{ATF\text{-}P} \\ i \neq j}} \sum_{t \in T_{ij}} x^t_{ij} = \begin{cases} m-1 & \text{if } j = 0 \\ 1 & \text{otherwise} \end{cases} \qquad j \in J_{ATF\text{-}P} \tag{30}$$

$$\sum_{\substack{j \in J_{ATF\text{-}P} \\ t \in T_{ji}}} x^t_{ji} - \sum_{\substack{j \in J_{ATF\text{-}P} \\ t' \in T_{ij} \\ t' = t + p_i + s_{ij}}} x^{t'}_{ij} = 0 \qquad i \in J, t = 0, \ldots, P - p_0 \tag{31}$$

$$x^t_{ij} \in \{0, 1\} \qquad i, j \in J_{ATF\text{-}P}, t \in T_{ij} \tag{32}$$

$$m \geq 0. \tag{33}$$

The objective function (27) minimizes the makespan. Constraint (28) forces $m-1$

jobs to be processed immediately after the dummy maintenance 0. Constraint (29) means that only one job is processed immediately after the dummy maintenance $0'$ at time $t = s_{0'j}$. Constraints (30) guarantee that every job is performed exactly once, and vertex 0 is visited after $m - 1$ jobs. Constraints (31) ensure flow conservation. Constraints (32) and (33) define the nature of the variables. Note that in contrast with ATF and ATF-M, this formulation can be solved directly by a commercial solver.

The solution of the Table 2 instance is depicted in Figure 3 adopting the ATF-P network. It can be noted that ATF-P drastically reduces the size of the search space; $m - 1$ paths depart from vertex 0 and terminate within time $P - p_0$ at vertex 0, whereas the last path departs from vertex $0'$ and returns to the same vertex within the same time frame. This representation highlights the similarities between ATF-P and problems with parallel machines. Each of the blocks can in fact be seen as an independent machine; the completion time of the last job performed on the m^{th} machine is considered to compute the overall makespan. Note that the objective function guarantees that the m^{th} machine performs at least one job.

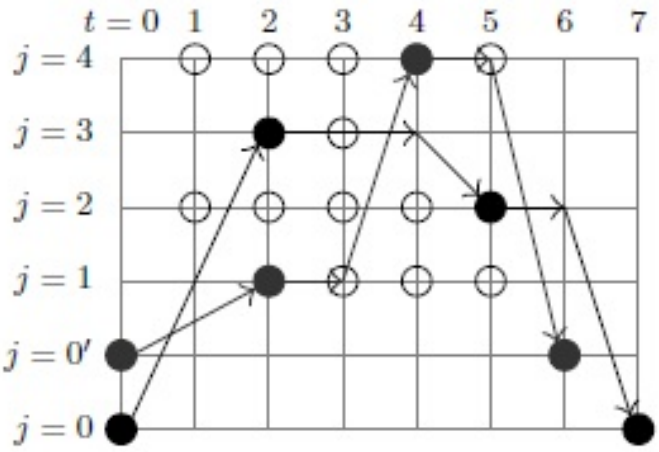


Figure 3: Network using ATF-P

3.4. Sizes of the models

Table 3 provides a comprehensive comparison of the numbers of constraints and variables of the models presented. It can be noted that all three models proposed in this paper rely on a pseudo-polynomial number of variables and constraints, whereas CF is polynomial in both the numbers of variables and constraints. Nevertheless, the ATF-P has a number of variables and constraints proportional to $P - p_0$. While this value can be arbitrarily large, it is significantly smaller than mP and $m(P - p_0)$ (the coefficients in the complexity formula of the number of constraints for ATF and ATF-M), when m is large. Despite the fact that ATF and ATF-M have a larger number of variables and constraints than ATF-P, these formulations can more easily incorporate future problem extensions, as, for example, incompatibilities between jobs in the same block and time-dependent setup or processing times.

Table 3: Sizes of the models

Formulation	**#Constraints**	**#Variables**
CF	$\|J\|^2 + 5\|J\| + 6$	$2(\|J\| + 1)^2 + 2\|J\| + 1$
ATF	$1 + (\|J\| + m) + Pm(\|J\| + m)$	$(\|J\| + m)^2(\max_{ij}\{\|T_{ij}\|\})$
ATF-M	$\|J\|m(P - p_0) + \|J\| + m$	$(\|J\| + 1)^2(P - p_0)m$
ATF-P	$\|J\|(P - p_0 + 1) + 4$	$(\|J\| + 2)^2(P - p_0) + 1$

4. An iterative exact algorithm for the 1MPS

The algorithm we propose iteratively solves the 1MPS by fixing the number of maintenances, starting from a valid lower bound, until the minimum number of maintenances allowing the scheduling of all jobs is reached. At each iteration, either the problem is solved to optimality by the MILP associated with a given model (which receives the number of maintenances as an input datum) and the algorithm terminates, or an infeasible solution is returned, meaning that at least an extra maintenance is required to achieve feasibility.

Each formulation used in our algorithm assumes that the number of maintenances to be performed is known *a priori*. This allows solving ATF and ATF-M to optimality using a solver, as well as the ATF-P and CF.

The algorithm starts by computing the following trivial lower bound $\mathbf{m}$ on the number of maintenances necessary to perform all jobs:

$$\mathbf{m} = \left\lfloor \frac{|J| \times \min_{i,j \in J_+} s_{ij} + \sum_{j \in J} p_j}{P - p_0} \right\rfloor. \tag{34}$$

The models presented in Section 3 and CP are initialized assuming that exactly $\mathbf{m}$ maintenances are necessary to complete all jobs and to reconfigure the machine for the final maintenance. If a feasible solution is found by the model, then it is optimal; otherwise, $\mathbf{m}$ is incremented by one unit and the model is solved again.

The number of iterations of this algorithm is bounded by $|J|$, but our computational results indicate that typically only a few iterations are necessary, and the computing time needed to confirm that $\mathbf{m}$ is insufficient to accommodate all jobs is typically small. More details about the performance of the algorithm will be provided in Section 5.

5. Computational experiments

We have conducted tests using a single core on an Intel Core i7 with 3.4 Ghz and 16 GB of RAM, with operating system Ubuntu 12.04. All formulations were solved by CPLEX 12.6 using default settings. The same computational environment was used to perform the tests regarding the CF formulation.

5.1. Benchmark Instances

In this section we describe the instances used to test the proposed formulations and solution methods. We first present the benchmark dataset suggested by Pacheco et al. (2013), and we then introduce a set of new benchmark instances.

5.1.1. Instances of Pacheco et al. (2013)

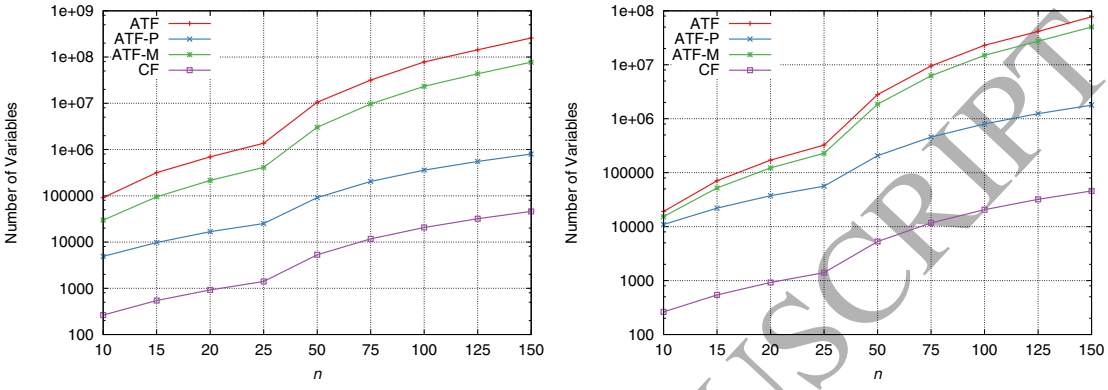
The number of jobs in the instances of Pacheco et al. (2013) is set as $|J| = 10, 12, 15, 20, 30, 40, 50$. These authors have used a discrete uniform distribution to generate the processing and setup times within the intervals $[2, 8]$, $[4, 12]$ and $[5, 20]$. For ease of presentation, we associate a value S to each interval, in particular, $S = 1, 2$

and $S = 3$, respectively. They have also considered different length for the blocks, namely: $2.25d_m$, $2.5d_m$, $3d_m$, $4d_m$, where $d_m = \max_{i \in J}\{(s_{0i} + p_i + s_{i0} + p_0)/2\}$. Since five instances were generated for each combination of n, S and block length (P), there are in total $5 \times 7 \times 3 \times 4 = 420$ instances.

5.1.2. Newly proposed instances

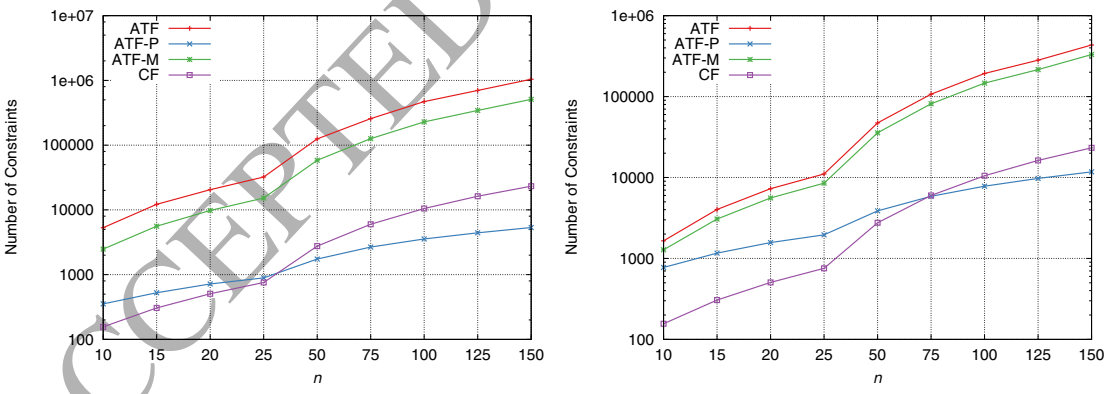
The new set of instances was generated as follows. The number of jobs was set as $|J| = 10$, 15, 20, 25, 50, 75, 100, 125, 150, whereas the processing and setup times were randomly generated using a discrete uniform distribution in the intervals $[2, 4]$ ($S = 1$), $[5, 10]$ ($S = 2$) and $[10, 20]$ ($S = 3$). The length of a block was computed as $\alpha \times \max_{i,j \in J_+}\{p_i + s_{ij}\}$, where α is either 2, 3 or 4. Five instances were generated for each combination involving the different values of n, S and α, thus resulting in $5 \times 9 \times 3 \times 3 = 405$ instances. This set of instances differs from that of Pacheco et al. (2013) for the following reasons: the triangular inequality is not satisfied, the range of the values of job processing times do not overlap, and the length of a block is computed using different tightness parameters.

Figures 4 and 5 show, in logarithmic scale, how the average number of variables and constraints of each formulation varies with the value of α for the newly generated instances. For the sake of conciseness, we chose to present only the graphs for $\alpha = 2$ and $\alpha = 4$. Note that the larger the value of α, the larger the length of the block in the instance. As reported in Appendix B, the performance of the formulations is highly dependent on the value of this parameter. According to Table 3, we can observe that the number of variables and constraints in CF only depends on the number of jobs. However, this is not the case for the arc-time-indexed formulations, for which the numbers of variables and constraints are clearly sensitive to α, i.e., the size of the block. More precisely, the average numbers of variables and constraints of ATF and ATF-M tend to rapidly increase for smaller values of α, but not for ATF-P. It is interesting to observe that, for larger values of n, the average number of constraints of ATF-P tends to be smaller than in CF.



(a) $\alpha = 2$ (b) $\alpha = 4$

Figure 4: Average number of variables



(a) $\alpha = 2$ (b) $\alpha = 4$

Figure 5: Average number of constraints

5.2. Standalone formulations versus iterative framework

This section compares the results obtained by using a standalone mathematical programming model when this is possible (ATF-P and CF) and the results achievable by using the same models within the iterative algorithm (see Appendix A for detailed results). Recall that this comparison cannot be extended to ATF and ATF-M, because m if one of the indices of some of their constraints. Table 4 reports the results on the instances proposed by Pacheco et al. (2013), Table 5 reports the results on the new set of instances. The tables are organised as follows: the rows represent the average results for a set value of n and α. The results of the ATF-P standalone model and iterative algorithm are reported first, followed by the same information for the CF standalone model and algorithm. For each method, we then report the number of optimal **#opt** solutions obtained, the average computing time for instances in which both methods found the optimal solution t^***(s)**, and the average percentage gap **Gap*****(%)** with respect to the linear relaxation value for the instances that both methods failed to solve to optimality. Note that for the Pacheco et al. (2013) instances both the ATF-P standalone and the ATF-P iterative algorithm always finds optimal solutions, so we only report the time column.

The iterative framework is systematically capable of finding at least as many optimal solutions as the standalone formulation within smaller computing times. When both approaches fail to find the optimal solution, the iterative framework typically yields smaller average percentage gaps. In what follows, we will therefore present results obtained by applying the iterative algorithm for all models, given that this approach empirically outperforms the corresponding standalone formulation.

5.3. Summary of the results

We now present an overview on the performance of the formulations embedded within the iterative algorithm. We first analyze the behaviour of the gap with respect to α. It is clear that the performance of ATF-M and CF depends on the value of α, as depicted in Figure 6. We do not include ATF nor ATF-P in this analysis because the first formulation would yield very large gaps on large-sized instances, and the average gap line of the second formulation would not be distinguishable from the 0%

Table 4: Results comparing the standalone formulation to the iterative framework for the Pacheco et al. (2013) instances

		CF						ATF-P	
		Standalone			Iterative			Standalone	Iterative
n	α	# opt	t^* (s)	Gap* (%)	# opt	t^* (s)	Gap* (%)	t^* (s)	t^* (s)
10	2.25	15	5.1	-	15	1.6	-	0.0	0.0
	2.50	15	1.0	-	15	0.5	-	0.0	0.0
	3.00	15	5.2	-	15	0.1	-	0.1	0.1
	4.00	15	1.1	-	15	0.1	-	0.7	0.4
12	2.25	15	16.6	-	15	5.7	-	0.0	0.0
	2.50	15	9.4	-	15	3.1	-	0.0	0.1
	3.00	15	17.8	-	15	0.4	-	1.0	0.1
	4.00	15	9.5	-	15	0.1	-	0.8	0.7
15	2.25	15	147.7	-	15	71.2	-	0.0	0.1
	2.50	15	21.6	-	15	11.4	-	0.1	0.1
	3.00	15	150.9	-	15	2.6	-	1.1	0.3
	4.00	15	22.3	-	15	0.2	-	4.6	0.6
20	2.25	7	2580.0	11.3	8	1471.4	9.6	0.1	0.1
	2.50	13	1373.7	8.2	14	1191.7	0.7	0.6	0.3
	3.00	15	64.2	-	15	86.7	-	2.0	0.7
	4.00	15	4.1	-	15	4.9	-	151.6	2.3
30	2.25	0	-	10.7	0	-	9.8	0.7	0.7
	2.50	1	648.4	7.9	1	2241.3	7.5	1.8	1.0
	3.00	4	215.2	3.2	4	495.5	3.3	4.9	3.0
	4.00	14	900.7	-	15	770.3	-	15.8	10.8
40	2.25	0	-	11.4	0	-	11.3	1.2	1.0
	2.50	0	-	10.0	0	-	9.7	3.2	3.3
	3.00	0	-	5.6	0	-	5.6	6.0	10.7
	4.00	4	270.8	1.5	4	493.8	1.4	65.6	37.0
50	2.25	0	-	13.5	0	-	12.9	1.4	2.5
	2.50	0	-	11.2	0	-	10.8	25.0	3.8
	3.00	0	-	5.7	0	-	5.5	136.8	15.0
	4.00	2	113.1	2.4	3	822.1	2.4	659.8	98.5
Avg.			229.2	8.1		184.1	7.7	38.7	6.9
Tot.		255			259				

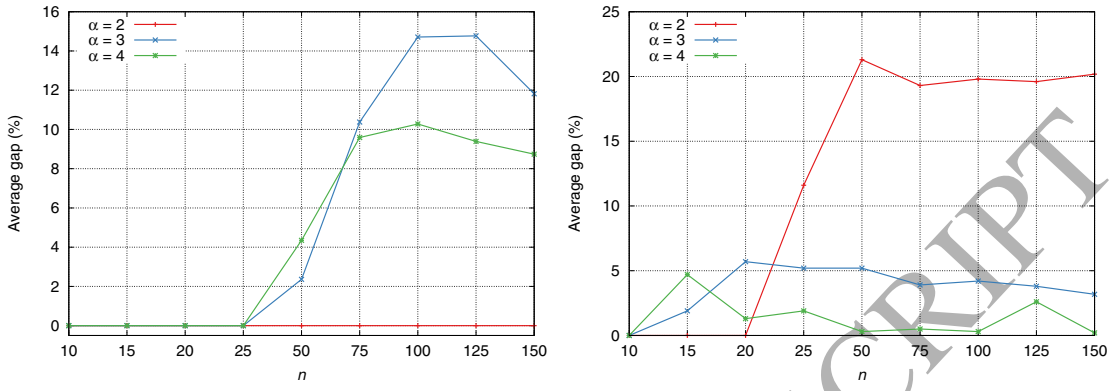
gap line. ATF-M yields better lower bounds for smaller values of α, whereas CF exhibits the opposite behaviour, especially for $\alpha = 4$, where CF finds relatively good lower bounds. Nevertheless, as CF struggles to find high quality upper bounds, the formulation still fails to prove the optimality of the instances with $\alpha = 4$ and $n > 25$. This may also explain the large number of nodes explored in the branch-and-bound tree, sometimes even exceeding the memory limit of 10 GB before the time limit of 7200 seconds for $n > 20$. The results show that the quality of the lower bound found by CF is sensitive to the value of α, even though the numbers of variables and constraints of this formulation are not a function of this parameter.

Table 5: Results comparing the standalone formulation to the iterative framework for the newly proposed instances

		CF						ATF-P					
		Standalone			Iterative			Standalone			Iterative		
n	α	# opt	t^* (s)	Gap* (%)	# opt	t^* (s)	Gap* (%)	# opt	t^* (s)	Gap* (%)	# opt	t^* (s)	Gap* (%)
10	2	15	0.7	-	15	0.6	-	15	0.0	-	15	0.1	-
	3	15	3.0	-	15	0.7	-	15	0.1	-	15	0.1	-
	4	15	1.8	-	15	1.8	-	15	4.8	-	15	0.4	-
15	2	15	78.1	-	15	32.2	-	15	0.0	-	15	0.2	-
	3	9	207.1	1.8	10	113.2	1.9	15	0.3	-	15	0.3	-
	4	14	34.2	4.5	14	98.4	4.7	15	8.5	-	15	1.3	-
20	2	12	861.8	-	15	318.1	-	15	0.1	-	15	0.3	-
	3	4	3366.5	5.3	3	3432.7	5.7	15	0.8	-	15	0.9	-
	4	9	1353.3	1.1	10	1165.1	1.3	15	17.9	-	15	4.8	-
25	2	3	669.8	15.3	7	114.2	11.6	15	0.1	-	15	0.5	-
	3	1	1745.8	5.5	1	1928.1	5.2	15	2.6	-	15	2.0	-
	4	6	940.2	1.7	5	692.6	1.9	15	25.8	-	15	14.5	-
50	2	0	-	21.3	0	-	21.3	15	0.3	-	15	3.9	-
	3	0	-	5.2	0	-	5.2	15	20.1	-	15	20.0	-
	4	0	-	0.4	0	-	0.3	15	651.8	-	15	545.0	-
75	2	0	-	19.3	0	-	19.3	15	0.6	-	15	12.2	-
	3	0	-	3.9	0	-	3.9	15	91.5	-	14	142.9	-
	4	0	-	0.5	0	-	0.5	10	1173.0	1.0	11	460.2	0.5
100	2	0	-	19.8	0	-	19.8	15	1.0	-	15	27.6	-
	3	0	-	4.2	0	-	4.2	15	271.7	-	14	155.5	-
	4	0	-	0.3	0	-	0.3	10	1892.4	1.4	10	649.7	0.5
125	2	0	-	19.7	0	-	19.6	15	3.5	-	15	56.7	-
	3	0	-	3.8	0	-	3.8	14	948.0	-	14	746.1	-
	4	0	-	2.6	0	-	2.6	6	1866.6	25.8	9	1024.0	25.5
150	2	0	-	20.2	0	-	20.2	15	2.4	-	15	87.8	-
	3	0	-	3.2	0	-	3.2	10	248.8	0.3	13	2069.4	0.1
	4	0	-	13.5	0	-	13.5	5	1231.0	20.9	8	3487.0	31.7
Avg.			771.9	7.9		658.1	7.7		313.5	9.9		352.3	11.7
Tot.		118			125			370			378		

Despite clearly dominating the results found by the other formulations in terms of lower and upper bounds, the overall performance of ATF-P still depends on the value of α. For example, while ATF-P solved all instances with $\alpha = 2$, it failed to solve a few instances with $\alpha = 3$ and $\alpha = 4$ (for $n = 75$ and $n = 100$). The average gap for the instances of up to 100 jobs and that were not solved to optimality was always smaller than 1%, thus demonstrating the good quality of the ATF-P bounds. Moreover, in order to better illustrate the influence of α on the performance of ATF-P, we present in Figure 7 (in logarithmic scale) the average CPU time in seconds spent by the iterative algorithm over the referred formulation for different values of α.

We can observe that the CPU time increases with the value of α, that is, the



(a) ATF-M (b) CF

Figure 6: Comparison between the average gaps (%) of ATF-M and CF for different values of α

larger the size of the blocks, the harder the instances for ATF-P. This result may seem counter-intuitive since the numbers of constraints and variables of ATF-P decrease as α increases (see Figures 4 and 5). However, since the number of possible schedules between two consecutive maintenances increases with the size of the block, this may possibly explain the difficulty faced by ATF-P in solving those type of instances.

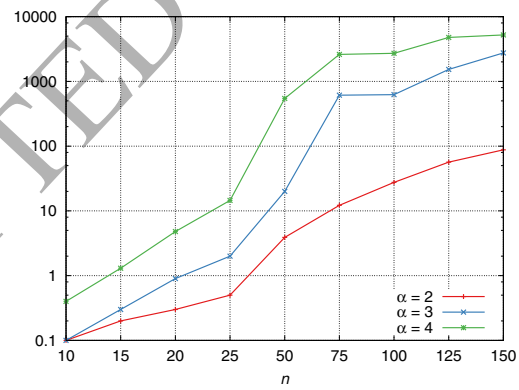


Figure 7: Average execution CPU time in seconds spent by the iterative algorithm over ATF-P

Tables 6 and 7 present a summary of the results obtained for the instances proposed by Pacheco et al. (2013) and those proposed in this work, respectively. In this case, **Gap(%)** denotes the average percentage gap between the lower bound and the

best-known upper bound, t**(s)** corresponds to the total average CPU time in seconds and we chose to report the average gap obtained after solving the root node (**Gap$_r$**) for the last iteration of the iterative algorithm, so as to provide a more accurate idea of the quality of the root relaxation of each formulation. The symbol "-" denotes that no lower bound was found for at least for one of the instances in the set after two hours of computing time, therefore making it impossible to compute the Gap(%).

It is worth mentioning that values of Gap(%) and Tree Size for the arc-time-indexed formulations are associated with the last iteration of the exact algorithm, i.e., the one for which the minimum feasible number of maintenances is used. The CPU times reported for such formulations correspond to the total execution time of the exact algorithm. A time limit of 7200 seconds and 10 GB of memory were imposed. Note that the Gap(%) values are computed with respect to the best-known upper bounds and therefore Gap(%) = 0.00 does not necessarily imply that a solution was proven optimal (i.e., the upper bounds identified by the algorithm at hand can be higher than the best upper bound). Moreover, a CPU time smaller than 7200 seconds may mean that the memory limit of 10 GB was exceeded.

Table 6: Summary of the results for the instances of Pacheco et al. (2013)

	CF				ATF				ATF-M				ATF-P			
n	Gap (%)	Gap$_r$ (%)	t (s)	# opt	Gap (%)	Gap$_r$ (%)	t (s)	# opt	Gap (%)	Gap$_r$ (%)	t (s)	# opt	Gap (%)	Gap$_r$ (%)	t (s)	# opt
10	0.00	9.19	0.59	60	0.00	0.14	1.6	60	0.00	0.00	0.5	60	0.00	0.00	0.1	60
12	0.00	8.87	2.34	60	0.00	0.35	3.5	60	0.00	0.59	1.1	60	0.00	0.00	0.2	60
15	0.00	8.20	21.37	60	0.00	0.64	9.6	60	0.00	0.66	2.3	60	0.00	0.02	0.3	60
20	1.14	9.24	1551.79	52	0.00	0.90	76.3	60	0.00	0.97	13.8	60	0.00	0.24	0.9	60
30	4.79	7.82	5153.43	20	0.00	1.38	707.0	60	0.00	1.21	205.1	60	0.00	0.17	3.9	60
40	6.89	7.49	4463.76	4	0.67	0.92	4345.4	37	0.07	0.89	1190.9	56	0.00	0.45	13.0	60
50	7.76	9.11	3762.19	3	0.02	0.64	4530.9	38	0.12	0.76	2208.1	49	0.00	0.29	29.9	60
Total	-	-	-	259	-	-	-	377	-	-	-	405	-	-	-	420

When considering for the 420 instances of Pacheco et al. (2013), ATF, ATF-M, ATF-P, and CF solve, 89.8%, 96.4%, 100%, and 61.7% of the instances to optimality, respectively. As for the 405 instances introduced in this work, ATF, ATF-M, ATF-P, and CF are capable of solving 40.7%, 61.0%, 93.3%, and 30.9% of the instances to optimality, respectively. Furthermore, the quality of root relaxation of the arc-time-indexed formulations, when considering the minimum feasible number

Table 7: Summary of the results for the newly proposed instances

n	CF				ATF				ATF-M				ATF-P			
	Gap (%)	Gap_r (%)	t (s)	# opt	Gap (%)	Gap_r (%)	t (s)	# opt	Gap (%)	Gap_r (%)	t (s)	# opt	Gap (%)	Gap_r (%)	t (s)	# opt
10	0.00	7.93	1.04	45	0.00	0.19	11.3	45	0.00	0.00	1.9	45	0.00	0.03	0.2	45
15	0.32	9.30	1172.64	39	0.00	0.42	181.3	45	0.00	0.11	10.9	45	0.00	0.00	0.6	45
20	1.55	10.74	3479.09	28	0.00	0.33	1187.4	45	0.00	0.28	65.5	45	0.00	0.02	2.0	45
25	4.04	10.62	5379.65	13	0.09	0.35	3425.7	30	0.00	0.26	238.2	45	0.00	0.04	5.7	45
50	8.96	9.13	4532.21	0	-	-	-	-	2.24	0.88	2889.0	32	0.00	0.07	189.6	45
75	7.89	7.92	5233.19	0	-	-	-	-	6.66	1.84	5091.1	19	0.05	0.08	1078.8	40
100	8.12	8.12	6256.39	0	-	-	-	-	8.93	1.63	6147.5	12	0.06	0.07	1161.5	39
125	8.65	8.67	6925.65	0	-	-	-	-	10.51	10.51	6815.0	3	0.84	0.85	1713.2	38
150	8.81	8.81	7200.00	0	-	-	-	-	9.33	9.34	7195.9	1	0.01	0.03	2686.4	36
Total	-	-	-	125	-	-	-	165	-	-	-	247	-	-	-	378

of maintenances, is superior to that obtained by CF. In the case of ATF-P, the average root gap is always smaller than 1.00%.

6. Conclusions

We have considered the single-machine scheduling problem with periodic maintenances and sequence-dependent setup times (1MPS). This difficult combinatorial optimization problem was solved by embedding some arc-time-indexed formulations within an iterative algorithm. This algorithm iterates on the minimum feasible number of blocks necessary to schedule all jobs. It starts by fixing the number of blocks to a lower bound and by solving an arc-time-indexed formulation to check whether a feasible scheduling solution exists with this number of blocks. If not, the number of blocks is incremented by one and the model is solved again. The procedure is reiterated until a feasible (and optimal) solution is found.

The three arc-time-indexed formulations introduced in this paper are simple due to a clearly defined time horizon within the iterative algorithm (i.e., the number of blocks). This enabled us to solve two models that cannot be solved directly by a commercial solver (ATF and ATF-M). Our third model, called ATF-P, contains a number of variables and a number of maintenances that are pseudo-polynomial in the duration of a single block, and not of the whole time horizon. It can be solved directly by CPLEX or by the iterative algorithm.

Extensive computational experiments confirmed that solving the 1MPS by embedding any of the three arc-time-indexed formulations within the iterative algorithm is clearly better than solving the model previously proposed by Pacheco et al. (2013). The ATF-P is the best of the four models used in the comparison, whether it is embedded within the iterative framework, or compared with CF when both models are solved directly by CPLEX. Solving the ATF-P model within the iterative framework yields optimal solutions for all of the benchmark instances with 50 jobs, and for all but 27 of the 405 newly generated instances.

Acknowledgments

This work was partially supported by the Brazilian research agency CNPq, grant 305223/2015-1, and by the Canadian Natural Sciences and Engineering Research Council under grant 2015-06189. This support is gratefully acknowledged. We also thank Ada Álvarez and Joaquín Pacheco for providing the instances proposed in Pacheco et al. (2013) and the original source code for the CF. We thank Prof. Artur Pessoa for his support and clarifications in the implementation of the separation procedures. Thanks are due to the referees for their valuable comments.

References

A. Allahverdi, C. Ng, T. Cheng, M. Kovalyov, A survey of scheduling problems with setup times or costs, European Journal of Operational Research 187 (3) (2008) 985 – 1032.

C.-J. Liao, C.-H. Lee, H.-T. Tsai, Scheduling with multi-attribute set-up times on unrelated parallel machines, International Journal of Production Research 54 (16) (2016) 4839–4853.

O. Herr, A. Goel, Minimising total tardiness for a single machine scheduling problem with family setups and resource constraints, European Journal of Operational Research 248 (1) (2016) 123 – 135.

A. Subramanian, M. Battarra, C. Potts, An Iterated Local Search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times, International Journal of Production Research 52 (9) (2014) 2729–2742.

R. Graham, E. Lawler, J. Lenstra, A. Kan, Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, in: E. J. P.L. Hammer, B. Korte (Eds.), Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver, vol. 5 of *Annals of Discrete Mathematics*, Elsevier, 287 – 326, 1979.

J.-Y. Lee, Y.-D. Kim, Minimizing the number of tardy jobs in a single-machine scheduling problem with periodic maintenance, Computers & Operations Research 39 (2012) 2196–2205.

D. L. Applegate, R. E. Bixby, V. Chvátal, W. J. Cook, The Traveling Salesman Problem: A Computational Study, Princeton University Press, 2006.

F. Ángel Bello, A. Álvarez, J. Pacheco, I. Martínez, A single machine scheduling problem with availability constraints and sequence-dependent setup costs, Applied Mathematical Modelling 35 (2011a) 2041–2050.

C. Liao, W. Chen, Single-machine scheduling with periodic maintenance and nonresumable jobs, Computers & Operations Research 30 (2003) 1335–1347.

M. Sbihi, C. Varnier, Single-machine scheduling with periodic and flexible periodic maintenance to minimize maximum tardiness, Computers & Industrial Engineering 55 (2008) 830–840.

W.-J. Chen, Minimizing Total Flow Time in the Single-Machine Scheduling Problem with Periodic Maintenance, Journal of the Operational Research Society 57 (2006a) 410–415.

J. Chen, Single-machine scheduling with flexible and periodic maintenance, Journal of the Operational Research Society 57 (2006b) 703–710.

W.-J. Chen, An efficient algorithm for scheduling jobs on a machine with periodic maintenance, International Journal of Advanced Manufacturing Technology 34 (11–12) (2007) 1173–1182.

W.-C. Chen, Minimizing number of tardy jobs on a single machine subject to periodic maintenance, Omega 37 (2009) 591–599.

M. Ji, Y. He, T. Cheng, Single-machine scheduling with periodic maintenance to minimize makespan, Computers & Operations Research 34 (2007) 1764–1770.

J.-S. Chen, Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan, European Journal of Operational Research 190 (2008) 90–102.

D. Xu, Y. Yin, H. Li, A note on scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan, European Journal of Operational Research 197 (2009) 825–827.

C. Low, C.-J. Hsu, C.-T. Su, A modified particle swarm optimization algorithm for a single-machine scheduling problem with periodic maintenance, Expert Systems with Applications 37 (2010) 6429–6434.

C.-J. Hsu, C. Low, C.-T. Su, A single-machine scheduling problem with maintenance activities to minimize makespan, Applied Mathematics and Computation 215 (11) (2010) 3929–3935.

F. Ángel Bello, A. Álvarez, J. Pacheco, I. Martínez, A heuristic approach for a scheduling problem with periodic maintenance and sequence-dependent setup times, Computers & Mathematics with Applications 61 (2011b) 797–808.

J. Pacheco, F. Ángel Bello, A. Álvarez, A multi-start tabu search method for a single-machine scheduling problem with periodic maintenance and sequence-dependent set-up times, Journal of Scheduling 16 (2013) 661–673.

A. Zade, M. Fakhrzad, A Dynamic Genetic Algorithm for Solving a Single Machine Scheduling Problem with Periodic Maintenance, ISRN Industrial Engineering 2013 (2013) 11 pages, article ID 936814.

X. Yu, Y. Zhang, G. Steiner, Single-machine scheduling with periodic maintenance to minimize makespan revisited, Journal of Scheduling 17 (3) (2014) 263–270.

W.-W. Cui, Z. Lu, Integrated production scheduling and periodic maintenances on a single machine with release dates, in: 2014 IEEE International Conference on Automation Science and Engineering (CASE), Taipei, Taiwan, 2014.

N. Kohl, J. Desrosiers, O. Madsen, M. Solomon, F. Soumis, 2-Path Cuts for the Vehicle Routing Problem with Time Windows, Transportation Science 33 (1) (1999) 101–116.

M. E. Dyer, L. A. Wolsey, Formulating the single machine sequencing problem with release dates as a mixed integer program, Discrete Applied Mathematics 26 (1990) 255–270.

F. Sourd, New exact algorithms for one-machine earliness-tardiness scheduling, INFORMS Journal on Computing 21 (2009) 167–175.

S. Tanaka, S. Fujikuma, M. Araki, An exact algorithm for single-machine scheduling without machine idle time, Journal of Scheduling 12 (2009) 575–593.

A. Pessoa, E. Uchoa, M. Poggi, R. Rodrigues, Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems, Mathematical Programming Computation 2 (2010) 259–290.

T. Nogueira, C. de Carvalho, M. Ravetti, Analysis of mixed integer programming formulations for single machine scheduling problems with sequence dependent setup

times and release dates, Tech. Rep., Universidade Federal de Minas Gerais, Brazil, 2014.

N. Boland, R. Clement, H. Waterer, A bucket indexed formulation for nonpreemptive single machine scheduling problems, INFORMS Journal on Computing 28 (1) (2016) 14–30.