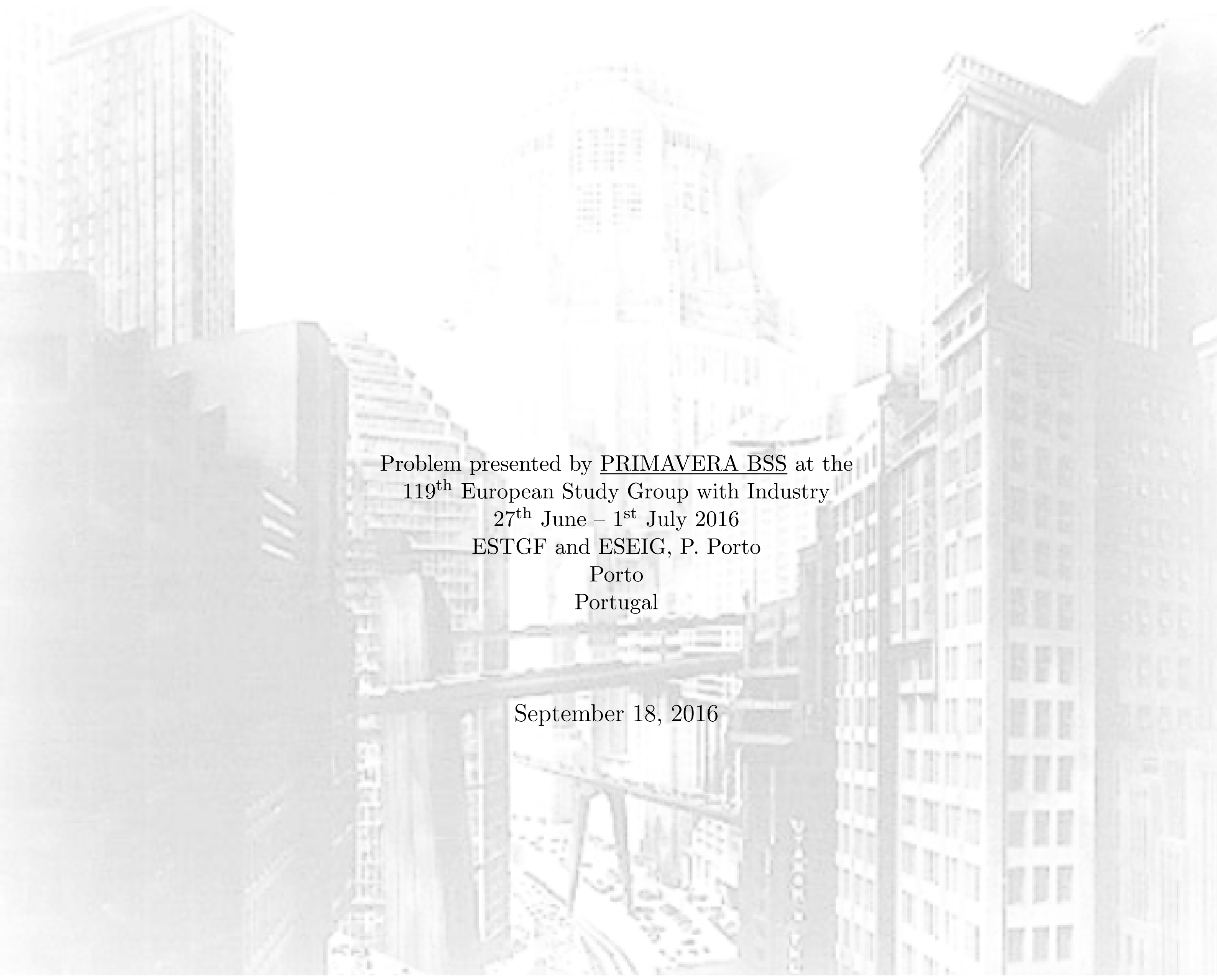
Portuguese Study Groups' Reports

Report on "Optimization of Production Planning"



Problem presented by PRIMAVERA BSS at the
119th European Study Group with Industry
27th June – 1st July 2016
ESTGF and ESEIG, P. Porto
Porto
Portugal

September 18, 2016

Problem presented by: PRIMAVERA BSS

Study group contributors: Adelaide Cerveira, Adérito Araújo, Eliana Silva, Isabel Cristina Lopes, A. Ismael F. Vaz and Rui Borges

Report prepared by:
Adelaide Cerveira (e-mail: cerveira@utad.pt)
Adérito Araújo (e-mail: alma@mat.uc.pt)
Eliana Silva (e-mail: eos@estgf.ipp.pt)
Isabel Cristina Lopes (e-mail: cristinalopes@eu.ipp.pt)
A. Ismael F. Vaz (e-mail: aivaz@dps.uminho.pt)
and
Rui Borges (e-mail: rui.borges@ua.pt)

Executive summary

The challenge proposed by PRIMAVERA BSS software company was to find an effective scheduling algorithm that can add new features to their production planning software, with a good performance (being able to run in less than 10 minutes), and that can be sufficiently generic and adaptable to be used by different industries (metal, furniture, wood, textile, and food industry). The requirements configured a NP-hard problem known in the literature as the Flexible Job Shop Scheduling Problem (FJSSP), for which sophisticated mathematical models and heuristic methods are widely available. The wherein proposed approaches consider sequence dependent setup times and different priorities for the operations. Two approaches are considered in the present report. First, two mathematical models were created to address this problem and give insight to the structure of the problem and its constraints. A second approach proposed the use of heuristics. A constructive heuristic used to find initial solutions is followed by the use of an improvement heuristic, which allows to obtain better solutions at reasonable computational costs. The solutions obtained by the heuristics can be used to warm start the optimal solving procedure using mathematical models with commercial solvers.

1 Introduction

1.1 The challenge

For nearly 10 years, PRIMAVERA BSS has had in its product portfolio a standard industrial production management solution, which covered key features across various industries. It is a very competitive solution regarding logistics, product engineering, budgeting, MRP, and MES. It offers a good user experience and it is an agile tool in a graphical environment. However, it is somewhat limited in the manufacturing planning aspect, both in terms of optimization criteria and in setting production planning priorities. After investing a few years in these areas, PRIMAVERA BSS considers it is time to give its customers other innovative solutions in the field of management planning by applying production optimization and prioritization algorithms.

The aim is to meet industry's needs in an increasingly global and competitive environment, where it is necessary to apply new concepts based on efficient methods to increase production capacity and performance. Given a set of jobs, operation times and costs, delivery dates, available resources, work centers, time schedules and calendars, the objective is to create algorithms and heuristics that can provide the best response to a combination of optimization criteria, such as minimizing the lateness in deliveries, the makespan, and the setup times, or maximizing work centers load level and throughput rate, considering one or more sorting rules (e.g. earliest due dates or shortest processing times).

PRIMAVERA BSS wants to find an effective scheduling algorithm that can add new features to their software, with a good performance (being able to run in less than 10 minutes), and that can be sufficiently generic and adaptable to be used by different industries (metal, furniture, wood, textile, and food industry).

1.2 Scheduling Problems

Scheduling is a decision making process that plays an important role in manufacturing and services industries. The allocation of resources to tasks over given time periods is done on a regular basis and it may have a strong impact in the profit of a company. In a scheduling problem, the goal is to optimize one or more objectives, such as minimizing the completion time of the last task, or minimizing the number of tasks completed after their due dates.

In scheduling problems, a finite set of n jobs and a finite set of m machines (or resources) are given. Each job consists of a fixed sequence of operations, where each operation has to be processed on a predetermined machine, knowing its necessary processing time. It is also known, for each job, its release date (the time that the job arrives at the system), its due date (the date that the job is promised to the customer) and sometimes its weight (a priority factor denoting the importance of that job relative to the other jobs in the system).

Scheduling problems can be *deterministic*, if it is assumed that the parameters of the jobs are known in advance, or *stochastic*, if the job data such as processing times or release dates may not be known, only their probability distributions are known in advance. Usually, in the latter models, there is a single objective function in the form of the *expected value* of some performance measure. For a survey in deterministic scheduling problems see [17] and for stochastic scheduling problems see [25]. In this report we will deal with a deterministic scheduling problem.

The possible machine environments include Single Machine problems, Parallel Machines, Flow Shop, Job Shop, and Open Shop. See [2] for a recent survey on several types of scheduling problems.

In a *Job Shop Scheduling Problem* (JSSP) with m machines each job has its own predetermined route of machines to follow. The machines are continuously available while jobs are independent and available at time zero. The processing times of jobs in the machines are also fixed and known in advance.

The *Flexible Job Shop Scheduling Problem* (FJSSP), first introduced in [6], is an extension of the classical job-shop scheduling problem using a parallel machine environment, where each operation can be processed on any of the machines of a given set (work centers). Each job has its own route of work centers to follow through the shop, in the sense that a job requires processing at each work center on only one machine but any machine in that work center can do. At any one time, each machine can process at most one operation of a job, while each job can be processed by at most one machine. Therefore, in a FJSSP, the focus of the problem is not only on finding an optimal scheduling of the operations, i. e. determining a starting time for each operation while taking into account precedence constraints and machine availability, but also an assignment of the operations to the machines is to be undertaken. A mathematical model for a flexible job shop problem in the Portuguese company TAP engine repair shop is presented in [7].

Most of the times the scheduling also has to account for *setup times.* This is of most relevance if the setup times are *sequence dependent*, which happens if the setup time of processing an operation in a certain machine is different according to which operation was processed immediately before [2]. In [23], a mathematical model is developed for FJSSP with sequence-dependent setup time for minimizing the total tardiness. If the setup time of an operation in a certain machine does not depend on the sequence of operations being processed in that machine, then the setup time can be simply included in the processing time of that operation.

Sometimes *preemption* is allowed, which implies that it is not necessary to keep a job on a machine, once started, until its completion, because the scheduler can interrupt the processing of a job at any moment and put a different job on the machine instead. Afterwards, when the preempted job is put on the same (or another) machine, it will only need the remaining processing time.

Recirculation may occur in a flexible job shop when a job may visit a machine or work center more than once. This is a common phenomenon in the real world [25]. In [13] an integer programming model is presented to deal with intermediate buffers and recirculation.

Most research considers a *static* problem, where a certain number of jobs are available simultaneously and are scheduled in a shop that is idle, where no other job will arrive until all are completely scheduled. Opposed to this there is the *dynamic* problem [1], where the jobs arrive intermittently in an unpredictable manner and arrivals will continue indefinitely into the future. In [12] the flexible job shop problem with new job insertion is considered, i.e., there may be new job(s) coming that have to be inserted into the current solution when the solution is being executed.

The most common performance measures in a scheduling problem are:

- Makespan;
- Tardiness;
- Lateness;

- Earliness;
- Number of Tardy jobs;
- Machine Overload.

The most common objective function is to minimize the *makespan* C_{max}, which is the completion time of the last job to leave the system. Scheduling while minimizing the number of tardy jobs is reviewed in [1], for both static and dynamic (online) problems. The makespan, earliness and tardiness, and machine overload are addressed in [12]. The objective function of a scheduling model can aim to optimize only one of these performance measures, or combine several of them, making it a multiobjective problem. For example, in [11] the makespan is combined with the mean of earliness and tardiness.

The job-shop scheduling optimization problem is known to be NP-hard, i.e., the complexity (and time to obtain an optimal solution) grows exponentially with the size of the problem (number of jobs and operations). In spite of this drawback there are several algorithms to address the problem, which can provide a near optimal solution to the problem.

1.3 The proposed approach

The challenge consists in the proposal and development of algorithms and heuristics for a popular optimization problem, known in the literature as the flexible job-shop scheduling optimization problem.

There exists two main algorithmic approaches to address such problems. The first one consists in developing a *mathematical model*, resulting in an optimization problem where the production planning is modeled as unknowns (variables) to be computed. The resulting optimization formulation is composed by an objective function that measures the quality of the planning, being subject to a set of constraints that model the production planing. Typically, this results in a mixed integer programming problem, since the unknowns take values in the set of integer and real numbers. The mathematical model development allows state-of-the-art software to be used if an optimal solution is to be obtained. However, due to the NP-hard property of the problem, an optimal solution for large optimization problems can only be expected when a significant amount of computation time is available.

The second approach consists in the development of dispatching rules and heuristics to address the job-shop scheduling problems. There are heuristics that can be used to obtain (construct) a solution (usually non optimal), which are called *constructive heuristics*, and there are *improvement heuristics* which are used to improve a given solution.

Note that even though some dispatching rules give reasonable results for some problem instances, it is difficult to predict when or for what type of instances they give good results, because heuristics are highly dependent on the scheduling problem characteristics, i.e., a change in the problem characteristics usually implies a heuristic redesign. Expert human intervention and adjustment of these heuristics can often improve their performance.

The features of a job-shop scheduling problem determine the difficulty of finding an effective model or algorithm for it. A statistical analysis of the relationship between the characteristics of a job-shop instance and its optimal makespan were studied in [22] to bring insight into the job-shop problems structure, to allow a classification of the instances according to its difficulty, and to help choose better heuristics.

We devote the next section to mathematical models in general, where two possible approaches are proposed. Some implementation details and numerical results are provided, in order to illustrate the proposed solution. Section 3 is devoted to heuristics, where constructive and improvement heuristics are described. We end this report by concluding and presenting some recommendations in Section 4.

2 Mathematical models

The mathematical model of an optimization problem corresponds to the minimization (or, equivalently, maximization) of an objective function, subject to a set of constraints. The objective function is a performance measure that depends on a set of unknowns (variables) to be determined, which model the problem in hand. The set of constraints are used to model the relation between the problems variables, being expressed as equalities or/and inequalities.

A *Linear Programming* formulation (LP) is a mathematical model where the objective function and the constraints are linear functions on the decision variables. If there are integer and continuous real valued variables, it is called a *Mixed Integer Programming* model (MIP).

Mathematical models for scheduling problems can be classified according to what the decision variables represent:

- Job completion time variables;
- Linear ordering variables;
- Time indexed variables;
- Network variables;
- Assignment and positional date variables.

Although job completion time is a key metric in assessing the quality of a proposed production schedule, job completion time variables are too simplistic to consider for the parallel machine environment case, especially considering their known poor performance in single machine environments [29].

A binary linear ordering variable is typically equal to one if a given job succeeds another job, thereby describing precedence relationships among all jobs.

Time indexed variables typically assign jobs to time periods. In a time indexed model formulation, a time horizon is discretized into time periods $1, \ldots, l$, where l is an upper bound of the last job's completion time (i.e., makespan), and a binary variable is one if a given job starts processing on a given machine at a given instant in time.

Network variables or "traveling salesman variables" were initially used to model the single machine scheduling problem with sequence dependent processing times, as it was shown to resemble a time-dependent traveling salesman problem (TSP). Parallel machine scheduling problems relate to the capacitated vehicle routing problem (CVRP) when the jobs to be scheduled are modeled as customers (nodes) and the machines represent the vehicles being routed. Each "route" defines a machine's schedule. In this regard, the capacity of each vehicle can be a surrogate measure for an upper bound on the completion time of the last job processed on each machine.

MIP formulations containing assignment and positional date variables specify which job is scheduled next and at what time this job will start processing. In an assignment and positional date model, decision variables are defined based on the notion that each machine has a fixed number of positions or slots into which jobs can be assigned. These positions by construction specify a job's relative position to all other jobs processed on the same machine and therefore, the job sequence on the machine. The binary assignment variables are one if a given job is assigned to a given position on a given machine. Additionally, the non-negative positional date variables denote the completion time of a given job at a given position on a given machine.

In the next subsections, two optimization models are proposed to address the job-shop scheduling problem that fits the PRIMAVERA BSS data and requests.

Both models consider a linear objective function subject to linear constraints, resulting in a linear programming problem. The first model (Model 1), described in Section 2.1, only considers integer (binary) variables and is, therefore, classified as a integer linear programming problem. The second model (Model 2), described in Section 2.2, considers both integer (binary) and continuous (real) variables, resulting in a mixed integer linear programming problem. In Section 2.3 both models are compared. We provide, in Section 2.4, some implementation details used to validate the proposed approach.

2.1 Model 1: Time indexed formulation

A possible optimization formulation to the job-shop scheduling problem is to consider a time indexed formulation, i.e., we consider a set of binary variables that indicate of a given job/operation is assigned to a certain working center for a certain unit of time.

Since each jobs is composed by a set of operations, we are considering single operations in the mathematical model.

Model 1 considers a planning for a set of time slots (e.g. hours) in a H time horizon. We establish the following sets and parameters:

$\mathcal{W}$	The set of work centers;
$\mathcal{O}$	The set of operations;
h	Instant in time ($h = 1, ..., H$);
d_o	Due date of operation o;
f_o	Priority of operation o (smaller value means greater priority);
a_{ow}	Priority of operation o in work center w;
t_{ow}	Transportation time of operation o in work center w;
s_{ow}	Setup time of operation o in work center w;
p_{ow}	Processing time of operation o in work center w;
$r_{o_1o_2}$	Transition time from operation o_1 to operation o_2;
$b_{o_1o_2}$	Takes the value 1 if operation o_1 precedes operation o_2, and 0 otherwise;
c_{wh}	Capacity of work center w at instant h.

The decision variables used in the model are defined as follows:

X_{owh}	Binary variable taking value 1 if the operation o occurs on work center w at the hour h, and 0 otherwise.

The optimization problem is:

$$\min \sum_{w\in\mathcal{W}} \sum_{o\in\mathcal{O}} \sum_{h=1}^{H} f_o \, h \, X_{owh} \tag{1}$$

s.t.:

$$\sum_{h=1}^{H} X_{owh} C_{wh} = p_{ow}, \quad \forall o \in \mathcal{O}, \forall w \in \mathcal{W} \tag{2}$$

$$\sum_{w \in \mathcal{W}} X_{owh} \leq 1, \quad \forall o \in \mathcal{O}, \forall h = 1, ..., H \tag{3}$$

$$(h_1 X_{o_1 w_1 h_1} + t_{o_1 w_1} + r_{o_1 o_2} + s_{o_2 w_2})\, b_{o_1 o_2} \leq h_2 X_{o_2 w_2 h_2}$$
$$\forall o_1, o_2 \in \mathcal{O}, o_1 \neq o_2, \forall w_1, w_2 \in \mathcal{W}, \forall h_1, h_2 = 1, ..., H \tag{4}$$

$$h_1 X_{owh_1} - h_2 X_{owh_2} \leq \sum_{h=1}^{H} X_{owh} - 1,$$
$$\forall o \in \mathcal{O}, \forall w \in \mathcal{W}, \forall h_1, h_2 = 1, ..., H \tag{5}$$

$$X_{owh} \leq a_{ow}, \quad \forall o \in \mathcal{O}\ \forall w \in \mathcal{W}\ \forall h = 1, ..., H \tag{6}$$

$$h X_{owh} \leq d_o, \quad \forall o \in \mathcal{O}\ \forall w \in \mathcal{W}\ \forall h = 1, ..., H \tag{7}$$

$$X_{owh} \in \{0, 1\}, \quad \forall o \in \mathcal{O}\ \forall w \in \mathcal{W}\ \forall h = 1, ..., H \tag{8}$$

The objective function described in (1) takes into consideration the priority of the operation together with the scheduling of the operation (since an operations assigned for a high h will make the objective function to increase). Other user defined objective functions can be considered.

Constraints are used to impose a valid solution. Constraints (2) impose the assignment of all time slots for each operations and working center. Constraints (3) guarantee that each operation, at a given time slot, is not assigned to more than one working center. Constraints (4) account for the precedence of operations and constraints (5) impose the operations time slots to be continuous in time. Constraints (6) account for the working center availability and Equation (7) imposes the due date for all operations. Finally constraints (8) are the variable domain constraints (binary variables).

2.2 Model 2: Continuous time formulation

Model 1 major drawback is the size of the problem, which increases dramatically with the number of time slots. Therefore we herein proposed another optimization problem.

We now consider a continuous time formulation for the problem of scheduling flexible job shops with sequence dependent setup times to minimize the total tardiness taking into account the job priorities. The model is described in detail in [23].

Let us consider a set of jobs $\mathcal{J} = \{1, \ldots, J\}$ and a set of work centers $\mathcal{W} = \{1, \ldots, W\}$, with $J, W \in \mathbb{N}$. Each job j includes a set of operations $\mathcal{O}_j = \{o_{j1}, \ldots, o_{jO_j}\}$, $O_j \in \mathbb{N}$, with its own processing route. To implement the setup time of the first operation on each machine, a dummy job zero is defined. Its single operation (o_{01}) is the first operation to be processed on all machines. It has the processing time of zero. Each work center is eligible to carry out a subset of operation types. For the flexible job shop scheduling problem, at least one work has to be eligible for more than one operation type. The processing time of each operation depends on the work center chosen from the set of eligible work centers for that operation type. A job is done when all its operations are done. The objective is to assign operations of jobs to work centers and then sequence operations on each work center so as to minimize total tardiness $\sum_{j\in\mathcal{J}} T_j$, where $T_j = \max\{C_j - d_j, 0\}$ is the tardiness of job j, being d_j its due date and C_j its completion time of job j. The sequence dependent setup between each two consecutive operations to be processed on the same work center is taken into. The magnitude of this setup depends on job(s) these two operations belong to.

We establish the following sets and parameters:

$\mathcal{J}$	The set of jobs;
$\mathcal{W}$	The set of work centers;
$\mathcal{O}$	The set of operations;
$\mathcal{O}_j$	The set including operations of job j, where $\#\mathcal{O}_j = O_j$;
o_{ji}	The $i-$th operation of job j;
d_j	The due date of job j;
f_j	Priority of job j (smaller value means greater priority);
p_{jiw}	The processing time of $i-$th operation of job j if it is processed by machine w;
$s_{j\hat{j}w}$	The setup time of job j immediately after job $\hat{j}$ on machine w;
e_{jiw}	Takes value 1 if work center w is eligible $i-$th operation of job j, and 0 otherwise;
M	A large positive number.

The decision variables used in the model are defined as follows:

$X_{o_{ji}o_{\hat{j}\hat{i}}w}$	Binary variable taking value 1 if o_{ji} is processed immediately after $o_{\hat{j}\hat{i}}$ on work center w, and 0 otherwise, where $o_{ji} \neq o_{\hat{j}\hat{i}}$;
C_{ji}	Continuous variable for the completion time of o_{ji};
T_j	Continuous variable for the tardiness of job j.

The model is:

$$\min \sum_{j\in\mathcal{J}} T_j f_j \tag{9}$$

s.t.:

$$\sum_{\hat{j}\in\mathcal{J}\cup\{0\}} \sum_{o_{\hat{j}\hat{i}}\in\mathcal{O}_{\hat{j}}} \sum_{w\in\mathcal{W}} X_{o_{ji}o_{\hat{j}\hat{i}}w} = 1, \forall j \in \mathcal{J}, \forall o_{ji} \in \mathcal{O}_{j_1} \tag{10}$$

$$\sum_{\hat{j}\in\mathcal{J}\cup\{0\}} \sum_{o_{\hat{j}\hat{i}}\in\mathcal{O}_{\hat{j}}} X_{o_{ji}o_{\hat{j}\hat{i}}w} = e_{o_{ji}w}, \forall j \in \mathcal{J}, \forall o_{ji} \in \mathcal{O}_j, \forall w \in \mathcal{W} \tag{11}$$

$$\sum_{j\in\mathcal{J}} \sum_{o_{ji}\in\mathcal{O}_j} \sum_{w\in\mathcal{W}} X_{o_{ji}o_{\hat{j}\hat{i}}w} \leq 1, \forall \hat{j} \in \mathcal{J}, \forall o_{\hat{j}\hat{i}} \in \mathcal{O}_{\hat{j}} \tag{12}$$

$$\sum_{j\in\mathcal{J}} \sum_{o\in\mathcal{O}_j} X_{o_{ij}o_{01}w} \leq 1, \forall w \in \mathcal{W} \tag{13}$$

$$\sum_{j\in\mathcal{J}} \sum_{o_{ji}\in\mathcal{O}_j} X_{o_{ji}o_{\hat{j}\hat{i}}w} \leq \sum_{j\in\mathcal{J}\cup\{0\}} \sum_{o_{ji}\in\mathcal{O}_j} X_{o_{\hat{j}\hat{i}}o_{ij}w}, \qquad \forall \hat{j} \in \mathcal{J}, \forall o_{\hat{j}\hat{i}} \in \mathcal{O}_{\hat{j}}, \forall w \in \mathcal{W} \tag{14}$$

$$C_{ji} \geq C_{ji-1} + \sum_{\hat{\jmath} \in \mathcal{J} \cup \{0\}} \sum_{\hat{\jmath}=1}^{O_{\hat{\jmath}}} \sum_{w \in \mathcal{W}} X_{o_{ji} o_{\hat{\jmath}\hat{\imath}} w} (p_{jiw} + s_{j\hat{\jmath}w}),$$
$$\forall j \in \mathcal{J}, \forall i = 1, ..., O_j \quad (15)$$

$$C_{ji} \geq C_{\hat{\jmath}\hat{\imath}} + \sum_{w \in \mathcal{W}} X_{o_{ji} o_{\hat{\jmath}\hat{\imath}} w} (p_{jiw} + s_{j\hat{\jmath}w}) - M \left(1 - \sum_{w \in \mathcal{W}} X_{o_{ji} o_{\hat{\jmath}\hat{\imath}} w} \right),$$
$$\forall j \in \mathcal{J}, \forall \hat{\jmath} \in \mathcal{J} \cup \{0\}, \forall i = 1, ..O_j, \forall \hat{\imath} = 1, ..O_{\hat{\jmath}} \quad (16)$$

$$T_j \geq C_{jO_j} - d_j, \forall j \in \mathcal{J} \quad (17)$$

$$C_{ji}, T_j \geq 0, \forall j \in \mathcal{J}, \forall i = 1, ..., O_j \quad (18)$$

$$X_{jo\hat{\jmath}\hat{o}w} \in \{0,1\}, \forall j \in \mathcal{J}, \forall \hat{\jmath} \in \mathcal{J} \cup \{0\}, \forall o_{ji} \in \mathcal{O}_j, \forall o_{\hat{\jmath}\hat{\imath}} \in \mathcal{O}_{\hat{\jmath}}, \forall w \in \mathcal{W} \quad (19)$$

where the $C_{j0} = C_{01} = 0$. Constraint set (10) ensures that every operation is scheduled exactly once. Moreover, it assures that each operation has exactly one preceding operation. Constraint set (11)) is to specify that each operation is assigned to one of its eligible work center. Constraint set (12) ensures that every operation could have at most one succeeding operation, because the operation in last position of each work center has no succeeding operation. Constraint set (13) assures the dummy operation to be the first operation on work center. Constraint set (14) assures that only operations on the same work center can be consecutive operations. Constraint set (15) is to implement this fact that a job cannot be processed on two different work centers at the same time. Constraint set (16) is to make sure that a work center cannot also process two different operations simultaneously. Constraint set (17) calculates the tardiness of jobs. Constraint sets (18) and (19) define the decision variables.

2.3 Dimension of the models

In [23] the author compares the effectiveness of Model 2 presented in the previous subsection with three other models available in the literatures and conclude that Model 2 was more effective than the other in terms of both size and computational complexity. In this subsection we compare the dimension of both models presented in this report. The results are shown in Table 1. In Model 1 the number of operations for each job was considered to be less or equal than the number of work centers.

Model 1	
Number of binary variables	JW^2H
Number of constrains	$J^2W^4H^2 + 2JW^2H + JW^2 + JWH$
Model 2	
Number of binary variables	J^2W^3
Number of continuous variables	$JW + J$
Number of constrains	$J^2W^2 + 2JW^2 + 3JW + J + W$

Table 1: Comparison of the models.

According to the results in Table 1, we may conclude that, in the case of a short time planning, the number of variables in Model 1 is less than the number of variables in Model 2. However, the number of constrains is considerably higher in Model 1 when compared with Model 2.

2.4 Implementation details

We provide an implementation of Model 2 simplification, using the data provided by PRIMAVERA BSS. Due to its easy to use we take advantage of the AMPL [21] modeling language. The AMPL modeling language allows a high level description of the model and the possibility to solve the problems by using state-of-the-art optimization software. The AMPL modeling language allows an optimization problem to be described in its modeling language, which it then instantiated with the problem data. The model file and the data file are provided in Appendix A.

AMPL provides a student edition version limited to problems with less than 300 variables and 300 constraints. Since our test case exceeds these limits we chose to use the NEOS [9, 10, 15] server in order to solve the problem. The Gurobi [16] solver was the solver of election since it addressed this type of optimization problems (mixed integer linear programming problems).

The optimal solution was obtained and the solver output is shown in Appendix B. From the output, we can observe that the makespan (the minimal time to finish all operations) is 229 time units. Start time operations is shown in the `Start` vector, e.g. `OF001PA001Pintar1` operation starts at time `66` and is scheduled in the work center `CT006`, as observed in the `Where` matrix.

The test problem used is a simplified version of Model 2, since some features of the problem were not considered. However, this test problem provides some information on how this type of problems may be addressed.

3 Heuristics

Heuristic methods may not find the optimal solution but they can usually find good solutions, at a small computational cost.

The first heuristic methods proposed to tackle the job-shop problem were simple dispatching rules, such as the Johnson rule, Earliest Due Date first rule (EDD), Minimum Slack first rule (MS), Shortest Queue at the Next Operation rule (SQNO) and the Shifting Bottleneck procedure [25, 27]. Dispatch rules are useful when attempting to find a reasonable good schedule with regard to one single objective. These methods are still used today for their efficiency, but they are not very effective at exploring the search space and may get caught at local optimums.

More sophisticated approximation methods called metaheuristics have been used for JSP with good results, such as simulated annealing [26], tabu search [30, 24], ant colony optimization [28], particle swarm optimization [20] and, more recently, discrete harmony search [11].

Genetic Algorithms have gained a well-earned reputation in being one of the best methods in solving JSSP. Although these still have shortcomings like premature convergence, and unsuitability for searching in small areas of the search space that are likely close to the optimal solutions, these can be overcome with improvements in the encoding and decoding schemes, genetic operators, hybridizing with other algorithms, or designing parallel genetic algorithms [3, 4, 8, 14, 19].

In heuristic methods two major categories are: constructive heuristics and improvement heuristics. Constructive heuristics aims to obtain a solution for the proposed problem by using some procedure to construct, most of the time iteratively, a feasible solution. An improvement heuristic starts from a given initial feasible solution, possibly the one obtained by the constructive heuristic, and looks for a better one usually by using some neighborhood search procedure, performing small changes in the solution. In Section 3.1 a constructive heuristic is described and, in Section 3.2 an improved heuristic is proposed. The notation used is the same as defined in previous sections.

3.1 Constructive Heuristic

The heuristic proposed as follows aims to tackle a general flexible job-shop scheduling problem [18, 5, 23, 2] - the problem in the literature that best fits the description provided by PRIMAVERA BSS. The proposed method can be easily adapted to fit other scheduling problems.

The herein proposed heuristic corresponds to a constructive method as it aims to obtain an initial feasible solution for the problem. Although these methods often provide interesting solutions in very short computing times, solutions can be further improved by using an improvement heuristic or metaheuristics.

The main underlying philosophy of the proposed method is to schedule each job at a time to the work center with the highest priority. The pseudocode is presented in Algorithm 1 and further detail is provided as follows. The algorithm uses the same notation as in Model 2.

The method starts out by initializing the first available instant for each work center (which may correspond to the beginning of the planning horizon), line 1 in Algorithm 1. Then, all jobs are ordered according to a given criteria (line 3) and, for break the ties, the due date and the release date of the jobs are considered. In the proposed algorithm the jobs are ordered according to their priority, this can however be changed according to the decision makers' interests, as well as the procedure for break the ties.

After the jobs are ordered, they are assigned to work centers following the established order (loop in lines 5-17). As the operations of each job may have precedence, i.e., some operations cannot start until others are completed, the second loop (lines 6-16) ensures that each operation is only scheduled if all preceding operations have been performed or if it does not have a preceding operation.

The loop in lines 8-15 handles the assignment of operations to work centers. Line 9 handles the ordering given to work centers, where, as with the ordering of jobs, it may be adapted to fit other criteria (possibly definable by the decision maker). Then, the operation is scheduled to the first work center in the ordered list, and the work centers' available time instant is updated according to the real processing time. The real processing time, calculated in line 11, should take into account the setup time, the transition time and the capacity of the work center.

The algorithm ends when all jobs (and corresponding operations) have been scheduled. Note that this method focuses on performing forward scheduling (from a given date onwards), however it can easily be adapted to perform backward scheduling (or both).

Aspects not yet considered in the constructive method, which can be included further on, are sequence dependent and sequence independent setup times.

Algorithm 1 Constructive Heuristic

```
 1: for each $w \in \mathcal{W}$, initialize first available instant, $T_w \leftarrow 0$ end for
 2: // Ordering jobs
 3: reorder $\mathcal{J}$ by priority. In case of ties use the due date, followed by the release date
 4: // Working centers assignment
 5: for each $j \in \mathcal{J}$
 6:     while $O_j \neq \emptyset$ do
 7:         let $P_{iO_j}$ be the list of preceding operations of operation $i$ from $O_j$
 8:         for each $i \in O_j$ such that $P_{iO_j} = \emptyset$
 9:             let $W_i$ be the list of working centers able to perform $i$, ordered
                   by $T_{w_i}$ (in case of ties, by priority concerning $i$)
10:             assign $i$ to the $W_i$ at the corresponding instant $T_{w_i}$
11:             calculate real processing time, $PT \leftarrow RPT(i, W_i, T_{w_i}, TS, TT)$
12:             $T_{w_i} \leftarrow T_{w_i} + PT$
13:             remove $i$ from $O_j$
14:             for all $k \in O_j$ remove $i$ from $P_{kO_j}$ end for
15:         end for
16:     end while
17: end for
```

3.2 Improvement Heuristic

The proposed improvement heuristic starts from a given initial solution and, for each work center, performs admissible swaps (obeying to the list of precedents of both operations) in the scheduled operations in order to improve the solution. The swaps are performed while there are improvements. The type of improvements being searched may change according to the decision maker's goals. Some examples are: (1) feasible swaps can be done looking for a reduction in the preparation time; (2) the swaps can be performed favoring work centers with bigger capacity, if there are periods where the working center is not processing; or (3) swaps can be done in order to eliminate the processing gaps in the work center (thus, possibly shortening the makespan). The definition of the criteria to conduct the swaps may lead to different solutions and should be in accordance with the decision makers' interests.

The pseudocode is presented in Algorithm 2 and further detail is provided as follows.

The method starts with a given feasible solution, line 1. For each work center, the swaps are performed while there are improvements (loop in lines 5-18). All the possible swaps are analyzed (line 7-line 12) but it is considered only in the case that the obtained solution is better than the best current solution (lines 9-11).

The algorithm ends when all the improvements are tested in all work centers. This method performs a swap once an improvement has been found ("first improvement"); however, a "best improvement" approach may also be implemented.

4 Conclusions and recommendations

The job-shop scheduling optimization problem is a well documented optimization problem. There are two main algorithm strategies to obtain an optimal solution. The first to be presented in this report is based on a mathematical model represented by a mixed integer

Algorithm 2 Improvement Heuristic

```
1: $bestSolution \leftarrow initialSolution$
2: // Operation permutation within work centers
3: for each $w \in \mathcal{W}$
4:     $improvement \leftarrow true$
5:     while $improvement = true$ do
6:         $Sol \leftarrow \emptyset$
7:         for each swap in operations order assigned to $w$ in $bestSolution$,
             obeying to the list of precedents of both operations
8:             perform the swap, obtaining $solution$
9:             if $solution$ is better than $bestSolution$ then
10:                $Sol \leftarrow Sol \cup \{solution\}$
11:            end if
12:        end for
13:        if $Sol = \emptyset$ then $improvement \leftarrow false$
14:        else
15:            order $Sol$ based on decreasing order of $improvement$
16:            $bestSolution \leftarrow$ first solution in $Sol$
17:        end if
18:    end while
19: end for
```

linear programming problem, to whom several state-of-the-art optimization solvers can be applied. The optimization problem can be solved by using deterministic (e.g. Gurobi) or stochastic solvers (e.g. genetic algorithms). Usually these type of algorithms can prove convergence to an optimal solution, in spite of the high computational time needed to obtain such a solution for high dimensional problems.

Heuristics (constructive and improvement) can also be applied to the job-shop scheduling problem. While heuristics can obtain a solution in a short computational time, there is no guarantee to obtain an optimal solution.

The recommendation is to implement constructive and improvement heuristics in order to obtain solutions after a short computational time and, if allowed, a deterministic solver should be used to obtain an optimal solution. The use of a deterministic solver can be improved by the inclusion of a cutoff point given by the upper bound obtained from the heuristic. Heuristics can be easily implemented in the PRIMAVERA BSS software. For the deterministic solver a third party software can be linked to the PRIMAVERA BSS software (most solvers are available as a DLL that can be easily linked to the PRIMAVERA BSS software).

A AMPL modeling and data files

The model file:

```
# Huge value
param Huge;

################### WorkCenter

set WorkCenters;

################### Operations (includes assemble operations)
set Operations;
```

```
# Times
param OpSetupTime{Operations,WorkCenters}, default 0;
# Zero processing time means not possible to assign
param OpProcessingTime{Operations,WorkCenters}, default 0; # 2*Huge;
param OpWaitingTime{Operations,WorkCenters}, default 0;
param OpTransportationTime{Operations,WorkCenters}, default 0;

# Operations
param OpPrecedences{op1 in Operations,op2 in Operations}>=0, default 0;
param OpDueDate{Operations};
param OpPriority{Operations};

# Costs
param WorkingCost{Operations,WorkCenters}>=0, default 20;

# Single Operations
set SingleOps;

set Products;

param ProductsSingleOps{Products,SingleOps}, binary;

##################### Jobs

set Jobs;

param JobName{Jobs};

param JobProductType{Jobs}, symbolic;

param JobReleaseDate{Jobs}>=0;

param JobDueDate{Jobs}>=0;

param JobPriority{Jobs}>=0;

param JobQuantity{Jobs}>0;


var Where{Operations,WorkCenters} binary;
var Start{Operations}>=0;
var Precedences{o1 in Operations,o2 in Operations: o1!=o2} binary;
var makespan>=0;

minimize fx:
    makespan;

# Account for the makespan (last operation to be complete)
subject to MinCompletion {o in Operations}:
    Start[o]+
      sum{w in WorkCenters: OpProcessingTime[o,w]>0}
        ((OpSetupTime[o,w]+OpProcessingTime[o,w]+OpWaitingTime[o,w]+OpTransportationTime[o,w])*Where[o,w])
          <=makespan;

# Comply with the due date
subject to DueDates {o in Operations}:
    Start[o]+
      sum{w in WorkCenters: OpProcessingTime[o,w]>0}
        ((OpSetupTime[o,w]+OpProcessingTime[o,w]+OpWaitingTime[o,w]+OpTransportationTime[o,w])*Where[o,w])
          <=OpDueDate[o];

# An operation cannot be assigned to more than one working center
subject to MakeOPNonSimultaneous1 {o in Operations}:
    sum{w in WorkCenters: OpProcessingTime[o,w]>0} Where[o,w]=1;

# An operation cannot be assigned to a working center with zero processing time
subject to MakeOPNonSimultaneous2 {w in WorkCenters, o in Operations: OpProcessingTime[o,w]=0}:
    Where[o,w]=0;

# Two operations cannot precede each other
subject to PrecedencesEx{o1 in Operations, o2 in Operations: o1!=o2}:
    Precedences[o1,o2]+Precedences[o2,o1]=1;

# Impose precedences in operations
subject to PrecedenceTime {o1 in Operations, o2 in Operations: OpPrecedences[o1,o2]=1}:
    Start[o1]+
      sum{w in WorkCenters: OpProcessingTime[o1,w]>0}
        ((OpSetupTime[o1,w]+OpProcessingTime[o1,w]+OpWaitingTime[o1,w]+OpTransportationTime[o1,w])*Where[o1,w])
          <=Start[o2];

# Different operations cannot be assigned to the same working center
subject to NoSimultaneous1 {o1 in Operations, o2 in Operations, w in WorkCenters: o1!=o2 and OpProcessingTime[o1,w]>0}:
    Huge*(1-Precedences[o1,o2])+Huge*(1-Where[o1,w])+Huge*(1-Where[o2,w])+(Start[o2]-Start[o1])>=
    (OpSetupTime[o1,w]+OpProcessingTime[o1,w]+OpWaitingTime[o1,w]+OpTransportationTime[o1,w]);

subject to NoSimultaneous2 {o1 in Operations, o2 in Operations, w in WorkCenters: o1!=o2 and OpProcessingTime[o2,w]>0}:
    Huge*Precedences[o1,o2]+Huge*(1-Where[o1,w])+Huge*(1-Where[o2,w])+(Start[o1]-Start[o2])>=
    (OpSetupTime[o2,w]+OpProcessingTime[o2,w]+OpWaitingTime[o2,w]+OpTransportationTime[o2,w]);
```

The data file:

```
param Huge := 10000;

set WorkCenters := CT001 CT002 CT003 CT004 CT005 CT006 CT007 CT008 CT009 CT010;

set Products := PA001 PA002 PA003 CP001 CP002;

set SingleOps := Cortar Pintar Soldar Laminar Secar;

param ProductsSingleOps (tr) : PA001 PA002 PA003 CP001 CP002 :=
Cortar 1 1 1 1 1
Pintar 1 0 1 0 0
Soldar 1 1 1 0 0
Laminar 0 0 0 1 0
Secar 1 0 0 0 0;

# Ordens de Fabrico e Sub Ordens de Fabrico
set Jobs := OF001 OF002 OF003 OF004 OF005 OF006 SOF001 SOF002 SOF003 SOF004 SOF005 SOF006;

# Date in hours
param JobProductType := OF001 PA001 OF002 PA002 OF003 PA001 OF004 PA002 OF005 PA002 OF006 PA003 SOF001
                        CP001 SOF002 CP002 SOF003 CP001 SOF004 CP002 SOF005 CP002 SOF006 CP001;

param JobDueDate := OF001 600 OF002 1200 OF003 2000 OF004 500 OF005 600 OF006 500 SOF001 600 SOF002 1200
                        SOF003 2000 SOF004 500 SOF005 600 SOF006 500;

param JobPriority := OF001 5 OF002 2 OF003 0 OF004 0 OF005 0 OF006 0 SOF001 5 SOF002 2 SOF003 0
                     SOF004 0 SOF005 0 SOF006 0;

param JobQuantity := OF001 1 OF002 1 OF003 1 OF004 1 OF005 1 OF006 1 SOF001 1 SOF002 1 SOF003 1
                     SOF004 1 SOF005 1 SOF006 1;

let Operations:={};

for {j in Jobs} {
    for {s in SingleOps} {
        if ProductsSingleOps[JobProductType[j],s]=1 then {
            for {q in 1..JobQuantity[j]}
                let Operations:= Operations union {j & JobProductType[j] & s & q};
            }
        }
};


for {j in Jobs} {
    for {s in SingleOps} {
        for {q in 1..JobQuantity[j]} {
            if ProductsSingleOps[JobProductType[j],s]=1 then {
                if s="Cortar" then {
                    let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT001"]:=1;
                    let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT002"]:=2;
                    let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT007"]:=3;
                } else {
                    if s="Laminar" then {
                        let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT008"]:=1;
                        let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT009"]:=2;
                    } else {
                        if s="Pintar" then {
                            let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT005"]:=1;
                            let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT004"]:=2;
                            let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT006"]:=3;
                        } else {
                            if s="Soldar" then {
                                let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT003"]:=1;
                                let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT002"]:=2;
                                let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT001"]:=3;
                            } else {
                                if s="Secar" then {
                                    let OpWorkCentersPriority[j & JobProductType[j] & s & q,"CT010"]:=1;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
};

for {j in Jobs} {
    for {q in 1..JobQuantity[j]} {
        if j="OF001" then {
            let OpSetupTime["OF001" & "PA001" & "Cortar" & q, "CT001"]:=1;
            let OpSetupTime["OF001" & "PA001" & "Cortar" & q, "CT002"]:=1;
            let OpSetupTime["OF001" & "PA001" & "Cortar" & q, "CT007"]:=1;
            let OpSetupTime["OF001" & "PA001" & "Pintar" & q, "CT005"]:=3;
            let OpSetupTime["OF001" & "PA001" & "Pintar" & q, "CT004"]:=3;
            let OpSetupTime["OF001" & "PA001" & "Pintar" & q, "CT006"]:=3;
            let OpProcessingTime["OF001" & "PA001" & "Cortar" & q, "CT001"]:=5;
            let OpProcessingTime["OF001" & "PA001" & "Cortar" & q, "CT002"]:=5;
            let OpProcessingTime["OF001" & "PA001" & "Cortar" & q, "CT007"]:=5;
            let OpProcessingTime["OF001" & "PA001" & "Pintar" & q, "CT005"]:=30;
            let OpProcessingTime["OF001" & "PA001" & "Pintar" & q, "CT004"]:=30;
            let OpProcessingTime["OF001" & "PA001" & "Pintar" & q, "CT006"]:=30;
            let OpProcessingTime["OF001" & "PA001" & "Soldar" & q, "CT003"]:=15;
            let OpProcessingTime["OF001" & "PA001" & "Soldar" & q, "CT002"]:=15;
            let OpProcessingTime["OF001" & "PA001" & "Soldar" & q, "CT001"]:=15;
            let OpProcessingTime["OF001" & "PA001" & "Secar" & q, "CT010"]:=60;
            let OpProcessingTime["SOF001" & "CP001" & "Cortar" & q, "CT001"]:=10;
            let OpProcessingTime["SOF001" & "CP001" & "Cortar" & q, "CT002"]:=11;
            let OpProcessingTime["SOF001" & "CP001" & "Cortar" & q, "CT007"]:=12;
            let OpProcessingTime["SOF001" & "CP001" & "Laminar" & q, "CT008"]:=30;
            let OpProcessingTime["SOF001" & "CP001" & "Laminar" & q, "CT009"]:=30;

            let OpWaitingTime["OF001" & "PA001" & "Secar" & q, "CT010"]:=5;
            let OpTransportationTime["OF001" & "PA001" & "Soldar" & q, "CT003"]:=5;
            let OpTransportationTime["OF001" & "PA001" & "Soldar" & q, "CT002"]:=5;
            let OpTransportationTime["OF001" & "PA001" & "Soldar" & q, "CT001"]:=5;
        } else {
            if j = "OF002" then {
                let OpSetupTime["OF002" & "PA002" & "Cortar" & q, "CT001"]:=1;
                let OpSetupTime["OF002" & "PA002" & "Cortar" & q, "CT002"]:=1;
                let OpSetupTime["OF002" & "PA002" & "Cortar" & q, "CT007"]:=1;
                let OpProcessingTime["OF002" & "PA002" & "Cortar" & q, "CT001"]:=1;
                let OpProcessingTime["OF002" & "PA002" & "Cortar" & q, "CT002"]:=1;
                let OpProcessingTime["OF002" & "PA002" & "Cortar" & q, "CT007"]:=1;
                let OpProcessingTime["OF002" & "PA002" & "Cortar" & q, "CT001"]:=5;
                let OpProcessingTime["OF002" & "PA002" & "Cortar" & q, "CT002"]:=5;
                let OpProcessingTime["OF002" & "PA002" & "Cortar" & q, "CT007"]:=5;
                let OpProcessingTime["OF002" & "PA002" & "Soldar" & q, "CT003"]:=15;
                let OpProcessingTime["OF002" & "PA002" & "Soldar" & q, "CT002"]:=15;
                let OpProcessingTime["OF002" & "PA002" & "Soldar" & q, "CT001"]:=15;
                let OpProcessingTime["SOF002" & "CP002" & "Cortar" & q, "CT001"]:=10;
                let OpProcessingTime["SOF002" & "CP002" & "Cortar" & q, "CT002"]:=11;
                let OpProcessingTime["SOF002" & "CP002" & "Cortar" & q, "CT007"]:=12;
                let OpTransportationTime["OF002" & "PA002" & "Soldar" & q, "CT003"]:=5;
                let OpTransportationTime["OF002" & "PA002" & "Soldar" & q, "CT002"]:=5;
                let OpTransportationTime["OF002" & "PA002" & "Soldar" & q, "CT001"]:=5;
            } else {
                if j="OF003" then {
                    let OpSetupTime["OF003" & "PA001" & "Cortar" & q, "CT001"]:=1;
                        let OpSetupTime["OF003" & "PA001" & "Cortar" & q, "CT002"]:=1;
                        let OpSetupTime["OF003" & "PA001" & "Cortar" & q, "CT007"]:=1;
                        let OpSetupTime["OF003" & "PA001" & "Pintar" & q, "CT005"]:=3;
                        let OpSetupTime["OF003" & "PA001" & "Pintar" & q, "CT004"]:=3;
                        let OpSetupTime["OF003" & "PA001" & "Pintar" & q, "CT006"]:=3;
                        let OpProcessingTime["OF003" & "PA001" & "Cortar" & q, "CT001"]:=5;
                        let OpProcessingTime["OF003" & "PA001" & "Cortar" & q, "CT002"]:=5;
                        let OpProcessingTime["OF003" & "PA001" & "Cortar" & q, "CT007"]:=5;
                        let OpProcessingTime["OF003" & "PA001" & "Pintar" & q, "CT005"]:=30;
                        let OpProcessingTime["OF003" & "PA001" & "Pintar" & q, "CT004"]:=30;
                        let OpProcessingTime["OF003" & "PA001" & "Pintar" & q, "CT006"]:=30;
                        let OpProcessingTime["OF003" & "PA001" & "Soldar" & q, "CT003"]:=15;
                        let OpProcessingTime["OF003" & "PA001" & "Soldar" & q, "CT002"]:=15;
                        let OpProcessingTime["OF003" & "PA001" & "Soldar" & q, "CT001"]:=15;
                        let OpProcessingTime["OF003" & "PA001" & "Secar" & q, "CT010"]:=60;
                        let OpProcessingTime["SOF003" & "CP001" & "Cortar" & q, "CT001"]:=10;
                        let OpProcessingTime["SOF003" & "CP001" & "Cortar" & q, "CT002"]:=11;
                        let OpProcessingTime["SOF003" & "CP001" & "Cortar" & q, "CT007"]:=12;
                        let OpProcessingTime["SOF003" & "CP001" & "Laminar" & q, "CT008"]:=30;
                        let OpProcessingTime["SOF003" & "CP001" & "Laminar" & q, "CT009"]:=30;
                        let OpWaitingTime["OF003" & "PA001" & "Secar" & q, "CT010"]:=5;
                        let OpTransportationTime["OF003" & "PA001" & "Soldar" & q, "CT003"]:=5;
                        let OpTransportationTime["OF003" & "PA001" & "Soldar" & q, "CT002"]:=5;
                        let OpTransportationTime["OF003" & "PA001" & "Soldar" & q, "CT001"]:=5;
                    } else {
                        if j="OF004" then {
                            let OpSetupTime["OF004" & "PA002" & "Cortar" & q, "CT001"]:=1;
                            let OpSetupTime["OF004" & "PA002" & "Cortar" & q, "CT002"]:=1;
                            let OpSetupTime["OF004" & "PA002" & "Cortar" & q, "CT007"]:=1;
                            let OpProcessingTime["OF004" & "PA002" & "Cortar" & q, "CT001"]:=1;
                            let OpProcessingTime["OF004" & "PA002" & "Cortar" & q, "CT002"]:=1;
                            let OpProcessingTime["OF004" & "PA002" & "Cortar" & q, "CT007"]:=1;
                            let OpProcessingTime["OF004" & "PA002" & "Cortar" & q, "CT001"]:=5;
                            let OpProcessingTime["OF004" & "PA002" & "Cortar" & q, "CT002"]:=5;
                            let OpProcessingTime["OF004" & "PA002" & "Cortar" & q, "CT007"]:=5;
                            let OpProcessingTime["OF004" & "PA002" & "Soldar" & q, "CT003"]:=15;
                            let OpProcessingTime["OF004" & "PA002" & "Soldar" & q, "CT002"]:=15;
                            let OpProcessingTime["OF004" & "PA002" & "Soldar" & q, "CT001"]:=15;
                            let OpProcessingTime["SOF004" & "CP002" & "Cortar" & q, "CT001"]:=10;
                            let OpProcessingTime["SOF004" & "CP002" & "Cortar" & q, "CT002"]:=11;
                            let OpProcessingTime["SOF004" & "CP002" & "Cortar" & q, "CT007"]:=12;
                            let OpTransportationTime["OF004" & "PA002" & "Soldar" & q, "CT003"]:=5;
                            let OpTransportationTime["OF004" & "PA002" & "Soldar" & q, "CT002"]:=5;
                            let OpTransportationTime["OF004" & "PA002" & "Soldar" & q, "CT001"]:=5;
                        } else {
                            if j="OF005" then {
                                let OpSetupTime["OF005" & "PA002" & "Cortar" & q, "CT001"]:=1;
```

```
                                let OpSetupTime["OF005" & "PA002" & "Cortar" & q, "CT002"]:=1;
                                let OpSetupTime["OF005" & "PA002" & "Cortar" & q, "CT007"]:=1;
                                let OpProcessingTime["OF005" & "PA002" & "Cortar" & q, "CT001"]:=1;
                                let OpProcessingTime["OF005" & "PA002" & "Cortar" & q, "CT002"]:=1;
                                let OpProcessingTime["OF005" & "PA002" & "Cortar" & q, "CT007"]:=1;
                                let OpProcessingTime["OF005" & "PA002" & "Cortar" & q, "CT001"]:=5;
                                let OpProcessingTime["OF005" & "PA002" & "Cortar" & q, "CT002"]:=5;
                                let OpProcessingTime["OF005" & "PA002" & "Cortar" & q, "CT007"]:=5;
                                let OpProcessingTime["OF005" & "PA002" & "Soldar" & q, "CT003"]:=15;
                                let OpProcessingTime["OF005" & "PA002" & "Soldar" & q, "CT002"]:=15;
                                let OpProcessingTime["OF005" & "PA002" & "Soldar" & q, "CT001"]:=15;
                                let OpProcessingTime["SOF005" & "CP002" & "Cortar" & q, "CT001"]:=10;
                                let OpProcessingTime["SOF005" & "CP002" & "Cortar" & q, "CT002"]:=11;
                                let OpProcessingTime["SOF005" & "CP002" & "Cortar" & q, "CT007"]:=12;
                                let OpTransportationTime["OF005" & "PA002" & "Soldar" & q, "CT003"]:=5;
                                let OpTransportationTime["OF005" & "PA002" & "Soldar" & q, "CT002"]:=5;
                                let OpTransportationTime["OF005" & "PA002" & "Soldar" & q, "CT001"]:=5;
                                } else {
                                    if j="OF006" then {
                                        let OpSetupTime["OF006" & "PA003" & "Cortar" & q, "CT001"]:=1;
                                        let OpSetupTime["OF006" & "PA003" & "Cortar" & q, "CT002"]:=1;
                                        let OpSetupTime["OF006" & "PA003" & "Cortar" & q, "CT007"]:=1;
                                        let OpSetupTime["OF006" & "PA003" & "Pintar" & q, "CT005"]:=3;
                                        let OpSetupTime["OF006" & "PA003" & "Pintar" & q, "CT004"]:=3;
                                        let OpSetupTime["OF006" & "PA003" & "Pintar" & q, "CT006"]:=3;

                                        let OpProcessingTime["OF006" & "PA003" & "Cortar" & q, "CT001"]:=1;
                                        let OpProcessingTime["OF006" & "PA003" & "Cortar" & q, "CT002"]:=1;
                                        let OpProcessingTime["OF006" & "PA003" & "Cortar" & q, "CT007"]:=1;
                                        let OpProcessingTime["OF006" & "PA003" & "Pintar" & q, "CT005"]:=3;
                                        let OpProcessingTime["OF006" & "PA003" & "Pintar" & q, "CT004"]:=3;
                                        let OpProcessingTime["OF006" & "PA003" & "Pintar" & q, "CT006"]:=3;
                                        let OpProcessingTime["OF006" & "PA003" & "Cortar" & q, "CT001"]:=5;
                                        let OpProcessingTime["OF006" & "PA003" & "Cortar" & q, "CT002"]:=5;
                                        let OpProcessingTime["OF006" & "PA003" & "Cortar" & q, "CT007"]:=5;
                                        let OpProcessingTime["OF006" & "PA003" & "Pintar" & q, "CT005"]:=30;
                                        let OpProcessingTime["OF006" & "PA003" & "Pintar" & q, "CT004"]:=30;
                                        let OpProcessingTime["OF006" & "PA003" & "Pintar" & q, "CT006"]:=30;
                                        let OpProcessingTime["OF006" & "PA003" & "Soldar" & q, "CT003"]:=15;
                                        let OpProcessingTime["OF006" & "PA003" & "Soldar" & q, "CT002"]:=15;
                                        let OpProcessingTime["OF006" & "PA003" & "Soldar" & q, "CT001"]:=15;

                                        let OpProcessingTime["SOF006" & "CP001" & "Cortar" & q, "CT001"]:=10;
                                        let OpProcessingTime["SOF006" & "CP001" & "Cortar" & q, "CT002"]:=11;
                                        let OpProcessingTime["SOF006" & "CP001" & "Cortar" & q, "CT007"]:=12;
                                        let OpProcessingTime["SOF006" & "CP001" & "Laminar" & q, "CT008"]:=30;
                                        let OpProcessingTime["SOF006" & "CP001" & "Laminar" & q, "CT009"]:=30;

                                        let OpTransportationTime["OF006" & "PA003" & "Soldar" & q, "CT003"]:=5;
                                        let OpTransportationTime["OF006" & "PA003" & "Soldar" & q, "CT002"]:=5;
                                        let OpTransportationTime["OF006" & "PA003" & "Soldar" & q, "CT001"]:=5;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }


for {j in Jobs} {
    for {s in SingleOps} {
        if ProductsSingleOps[JobProductType[j],s]=1 then {
            for {q in 1..JobQuantity[j]} {
                let OpDueDate[j & JobProductType[j] & s & q]:= JobDueDate[j];
                let OpPriority[j & JobProductType[j] & s & q]:= JobPriority[j];
            }
        }
    }
};


# Precedences for sub OFs
#for {j in {"OF001", "OF002", "OF003", "OF004", "OF005", "OF006"}} {
for {j in {"OF001", "OF003"}} {
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences["S" & j & JobProductType["S" & j] & "Laminar" & q, j & JobProductType[j] & "Cortar" & q]:=1;
    }
}

for {j in {"OF002", "OF004", "OF005"}} {
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences["S" & j & JobProductType["S" & j] & "Cortar" & q, j & JobProductType[j] & "Cortar" & q]:=1;
    }
}

for {j in {"OF006"}} {
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences["S" & j & JobProductType["S" & j] & "Laminar" & q, j & JobProductType[j] & "Cortar" & q]:=1;
    }
}

# Precedences inside OFs
for {j in {"OF001", "OF003"}}{
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences [j & "PA001" & "Cortar" & q, j & "PA001" & "Soldar" & q]:=1;
        let OpPrecedences [j & "PA001" & "Soldar" & q, j & "PA001" & "Pintar" & q]:=1;
        let OpPrecedences [j & "PA001" & "Pintar" & q, j & "PA001" & "Secar" & q]:=1;
    }
}

for {j in {"SOF001", "SOF003"}}{
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences [j & "CP001" & "Cortar" & q, j & "CP001" & "Laminar" & q]:=1;
    }
}

for {j in {"OF002", "OF004", "OF005"}}{
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences [j & "PA002" & "Cortar" & q, j & "PA002" & "Soldar" & q]:=1;
    }
}

for {j in {"OF006"}}{
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences [j & "PA003" & "Cortar" & q, j & "PA003" & "Soldar" & q]:=1;
        let OpPrecedences [j & "PA003" & "Soldar" & q, j & "PA003" & "Pintar" & q]:=1;
    }
}

for {j in {"SOF006"}}{
    for {q in 1..JobQuantity[j]} {
        let OpPrecedences [j & "CP001" & "Cortar" & q, j & "CP001" & "Laminar" & q]:=1;
    }
}


for {j1 in {"OF001","OF003"}, j2 in {"OF006"}} {
    for {q1 in 1..JobQuantity[j1]}{
        for {q2 in 1..JobQuantity[j2]}{
            let OpTransTime[j1 & JobProductType[j1] & "Pintar" & q1,j2 & JobProductType[j2] & "Pintar" & q2]:=3;
        }
    }
}
```

B NEOS Gurobi output for the test problem

The Gurobi solver output obtained in the NEOS platform.

```
*************************************************************

   NEOS Server Version 5.0
   Job#     : 4854803
   Password : UZbOsmCE
   Solver   : milp:Gurobi:AMPL
   Start    : 2016-08-11 19:15:57
   End      : 2016-08-11 19:16:18
   Host     : NEOS HTCondor Pool

   Disclaimer:

   This information is provided without any express or
   implied warranty. In particular, there is no warranty
   of any kind concerning the fitness of this
   information  for any particular purpose.
*************************************************************
File exists
You are using the solver gurobi_ampl.
Checking ampl.mod for gurobi_options...
Checking ampl.com for gurobi_options...
Executing AMPL.
processing data.
processing commands.
Executing on neos-7.neos-server.org

Presolve eliminates 2139 constraints and 191 variables.
Adjusted problem:
746 variables:
719 binary variables
27 linear variables
2348 constraints, all linear; 9702 nonzeros
674 equality constraints
1674 inequality constraints
1 linear objective; 1 nonzero.

Gurobi 6.5.0: threads=4
outlev=1
Optimize a model with 2348 rows, 746 columns and 9702 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 1e+04]
  Objective range [1e+00, 1e+00]
  Bounds range    [1e+00, 2e+03]
  RHS range       [1e+00, 3e+04]
Found heuristic solution: objective 230
Presolve removed 1517 rows and 502 columns
Presolve time: 0.01s
Presolved: 831 rows, 244 columns, 3995 nonzeros
Variable types: 27 continuous, 217 integer (217 binary)

Root relaxation: objective 2.290000e+02, 112 iterations, 0.00 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

*    0     0               0     229.0000000  229.00000  0.00%     -    0s

Explored 0 nodes (174 simplex iterations) in 0.03 seconds
Thread count was 4 (of 64 available processors)

Optimal solution found (tolerance 1.00e-04)
Best objective 2.290000000000e+02, best bound 2.290000000000e+02, gap 0.0%
Optimize a model with 2348 rows, 746 columns and 9702 nonzeros
Coefficient statistics:
  Matrix range    [1e+00, 1e+04]
  Objective range [1e+00, 1e+00]
  Bounds range    [1e+00, 2e+03]
  RHS range       [1e+00, 3e+04]
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   1.612688e+02   0.000000e+00      0s
      33    2.2900000e+02   0.000000e+00   0.000000e+00      0s

Solved in 33 iterations and 0.00 seconds
Optimal objective  2.290000000e+02
Gurobi 6.5.0: optimal solution; objective 229
174 simplex iterations
plus 33 simplex iterations for intbasis
Where [*,*]
:                    CT001 CT002 CT003 CT004 CT005 CT006 CT007 CT008 CT009 CT010 :=
OF001PA001Cortar1      0     0     0     0     0     0     1     0     0     0
OF001PA001Pintar1      0     0     0     0     0     1     0     0     0     0
OF001PA001Secar1       0     0     0     0     0     0     0     0     0     1
OF001PA001Soldar1      0     0     1     0     0     0     0     0     0     0
OF002PA002Cortar1      1     0     0     0     0     0     0     0     0     0
OF002PA002Soldar1      0     0     1     0     0     0     0     0     0     0
OF003PA001Cortar1      0     0     0     0     0     0     1     0     0     0
OF003PA001Pintar1      0     0     0     0     1     0     0     0     0     0
OF003PA001Secar1       0     0     0     0     0     0     0     0     0     1
OF003PA001Soldar1      0     0     1     0     0     0     0     0     0     0
OF004PA002Cortar1      0     0     0     0     0     0     1     0     0     0
```

```
OF004PA002Soldar1      0    0    1    0    0    0    0    0    0    0
OF005PA002Cortar1      0    1    0    0    0    0    0    0    0    0
OF005PA002Soldar1      0    0    1    0    0    0    0    0    0    0
OF006PA003Cortar1      0    1    0    0    0    0    0    0    0    0
OF006PA003Pintar1      0    0    0    1    0    0    0    0    0    0
OF006PA003Soldar1      0    1    0    0    0    0    0    0    0    0
SOF001CP001Cortar1     1    0    0    0    0    0    0    0    0    0
SOF001CP001Laminar1    0    0    0    0    0    0    0    0    1    0
SOF002CP002Cortar1     1    0    0    0    0    0    0    0    0    0
SOF003CP001Cortar1     0    0    0    0    0    0    1    0    0    0
SOF003CP001Laminar1    0    0    0    0    0    0    0    0    1    0
SOF004CP002Cortar1     0    0    0    0    0    0    1    0    0    0
SOF005CP002Cortar1     0    1    0    0    0    0    0    0    0    0
SOF006CP001Cortar1     0    0    0    0    0    0    1    0    0    0
SOF006CP001Laminar1    0    0    0    0    0    0    0    1    0    0
;

Start [*] :=
  OF001PA001Cortar1  40     OF003PA001Soldar1 111   SOF001CP001Laminar1  10
  OF001PA001Pintar1  66     OF004PA002Cortar1  85    SOF002CP002Cortar1  10
   OF001PA001Secar1  99     OF004PA002Soldar1  91    SOF003CP001Cortar1   0
  OF001PA001Soldar1  46     OF005PA002Cortar1  20   SOF003CP001Laminar1  75
  OF002PA002Cortar1  20     OF005PA002Soldar1  26    SOF004CP002Cortar1  12
  OF002PA002Soldar1 209     OF006PA003Cortar1 170    SOF005CP002Cortar1   9
  OF003PA001Cortar1 105     OF006PA003Pintar1 196    SOF006CP001Cortar1  28
  OF003PA001Pintar1 131     OF006PA003Soldar1 176   SOF006CP001Laminar1 140
   OF003PA001Secar1 164    SOF001CP001Cortar1   0
;
```

References

[1] M. O. Adamu and A. O. Adewumi. Minimizing the weighted number of tardy jobs on multiple machines: A review. *Journal of Industrial and Management Optimization*, 12(4):1465–1493, 2016.

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

[3] M. Amirghasemi and R. Zamani. An effective asexual genetic algorithm for solving the job shop scheduling problem. *Computers & Industrial Engineering*, 83:123 – 138, 2015.

[4] L. Asadzadeh. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. *Computers & Industrial Engineering*, 85:376 – 383, 2015.

[5] D. Behnke and M.J. Geiger. Test instances for the flexible job shop scheduling problem with work centers. Technical report, Research Report RR-12-01-01, 2012.

[6] P. Brucker and R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, 1990.

[7] J.O. Cerdeira, I.C. Lopes, and E.C. Silva. Optimal scheduling of aircrafts' engines repair process. In Graham Kendall, Greet Vanden Berghe, and Barry McCollum, editors, *6th Multidisciplinary International Scheduling Conference: Theory & Applications*, pages 620–623, Gent, Belgium, Aug 2013.

[8] D. Cinar, J.A. Oliveira, Y.I. Topcu, and P. M. Pardalos. A priority-based genetic algorithm for a flexible job shop scheduling problem. *Journal of Industrial and Management Optimization*, 12(4):1391–1415, 2016.

[9] J. Czyzyk, M.P. Mesnier, and J.J. Moré. The neos server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998.

[10] E. Dolan. The neos server 4.0 administrative guide. Technical report, Mathematics and Computer Science Division, Argonne National Laboratory, 2001.

[11] K. Z. Gao, P. N. Suganthan, Q. K. Pan, T. J. Chua, T. X. Cai, and C. S. Chong. Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. *Journal of Intelligent Manufacturing*, 27(2):363–374, 2016.

[12] K. Z. Gao, P. N. Suganthan, M. F. Tasgetiren, Q. K. Pan, and Q. Q. Sun. Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion. *Computers & Industrial Engineering*, 90:107 – 117, 2015.

[13] M.C. Gomes, A.P. Barbosa-Póvoa, and A.Q. Novais. Optimal scheduling for flexible job shop operation. *International Journal of Production Research*, 43(11):2323–2353, 2005.

[14] J. F. Gonçalves and M. G. C. Resende. An extended akers graphical method with a biased random-key genetic algorithm for job-shop scheduling. *International Transactions in Operational Research*, 21(2):215–246, 2014.

[15] W. Gropp and J. J. Moré. Optimization environments and the neos server. In M.D. Buhmann and A. Iserles, editors, *Approximation Theory and Optimization*, pages 167–182. Cambridge University Press, 1997.

[16] Gurobi Optimization Inc. Gurobi optimizer reference manual, 2015.

[17] A.S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390 – 434, 1999.

[18] I. Kacem, S. Hammadi, and P. Borne. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, 60:245–276, 2002.

[19] M. Kurdi. An effective new island model genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 67:132 – 142, 2016.

[20] Tsung-Lieh Lin, Shi-Jinn Horng, Tzong-Wann Kao, Yuan-Hsin Chen, Ray-Shine Run, Rong-Jian Chen, Jui-Lin Lai, and I-Hong Kuo. An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications*, 37(3):2629–2636, 2010.

[21] AMPL Optimization LLC. AMPL: A modeling language for mathematical programming. `http://www.ampl.com`, 2012.

[22] S. Mirshekarian and D.N. Sormaz. Correlation of job-shop scheduling problem features with scheduling efficiency. *Expert Systems with Applications*, 62:131 – 147, 2016.

[23] M. Mousakhani. Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. *International Journal of Production Research*, 51:3476–3487, 2013.

[24] B. Peng, Z.Lü, and T.C.E. Cheng. A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53:154 – 164, 2015.

[25] M.L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer, 2012.

[26] Tsuyoshi Satake, Katsumi Morikawa, Katsuhiko Takahashi, and Nobuto Nakamura. Simulated annealing approach for minimizing the makespan of the general job-shop. *International Journal of Production Economics*, 60-61:515 – 522, 1999.

[27] Veronique Sels, Nele Gheysen, and Mario Vanhoucke. A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270, 2012.

[28] Minseok Seo and Daecheol Kim. Ant colony optimisation with parameterised search space for the job shop scheduling problem. *International Journal of Production Research*, 48(4):1143–1154, 2010.

[29] Yasin Unlu and Scott J. Mason. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785 – 800, 2010.

[30] ChaoYong Zhang, PeiGen Li, ZaiLin Guan, and YunQing Rao. A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. *Computers & Operations Research*, 34(11):3229 – 3242, 2007.