

Modelling the Job-Shop Scheduling problem in Linear Programming and Constraint Programming

Pedro Abreu

Faculdade de Engenharia da Universidade do Porto

Abstract. The Job-Shop Scheduling problem is a well-know combinatorial optimization problem. In this problem, we have to schedule several tasks minimizing the total time to process all the tasks. There are several constraints like the order of processing, resource capacity limitation and others. There are several algorithms to search the best solution. We will be looking here at approaches according to the Linear Programming and Constraint Programming paradigms. These algorithms search in a set of possible solution, within defined constraints, with the objective of minimize a function that gives a value to solutions. The solutions can be represented in several ways with different constraints and evaluation functions. In this work, we present different representation for the solution and the respective evaluation function and constraints for Linear Programming and Constraint Programming. We compare the different methods applied to different type of instances of the problem. The instances differ in the dimensions, the durations and the method to generate the instances. The results show that the Linear Programming Model with starting time for solution representation dominates in terms of the performance average the others in almost all instances and classes of the instances.

Key words: modelling, constraint programming, linear programming, job-shop, combinatorial optimization

1 Introduction

The Job-Shop Scheduling (JSS) problem is a well-know combinatorial optimization problem. In the JSS, the data of the problem are K tasks ($\{O_{jm} : j \in J, m \in M\}$) where $J = \{J_1, \dots, J_n\}$ is a set of jobs and $M = \{M_1, \dots, M_n\}$ is a set of machines. It is assumed that a task (O_{jm}) belongs to a job (J_j) and is processed in a specific machine (M_m). The tasks of the same job have an order to be processed. Each task has a processing duration. There are some restrictions such as: in each machine, can only be processed one task at each time and other restriction is that the processing of a task cannot be interrupted. The solution to the problem is the schedule of the beginning time of processing the tasks with the objective to minimize the value of the total time to process all the tasks (makespan). So, the solution with the less makespan is called the solution.

There are several algorithms to search optimal solution for optimization problems. The Linear Programming (LP) and Constraint Programming (CP) are two well-know optimization techniques. The two techniques have the objective to minimize/maximize a function subject to constraints that limit the function domain. The major differences between the two, is that LP is a technique only for optimize linear functions subject to linear constraints. The LP algorithm is called Mixed Integer Programming (MIP) if the domain is a set of integer variables. The CP algorithm is called Constraint Programming for Finite Domains (CLPFD) if the domain is a set of integer variables. So, having a optimization problem, we can define, for LP and CLP, the domain as the solutions for the problem. The constraints are the restrictions of the problem. For example, for the JSS problem, we can define as the domain the starting time of each task and the function to minimize is the total time to process all the tasks.

However, there are several possible representation for the solution of the problem. Above we give a possible representation of the domain, but we can also define the solution for the JSS, for example, as the order of processing the tasks [1]. So, a question can be ask: what algorithm or/and representation we should use for a specific problem of JSS? There are theorems such the No-Free Lunch theorems [10] and benchmark works [3] that shows there aren't any best algorithm for all classes of the problem. In this work, we present 4 methods: 2 different solution representation for the CP algorithm and 2 different solution representation for the LP algorithm. We compare the performance of the 4 methods in several different instances of JSS. The methods are:

- MIP algorithm using the domain as an integer variable for each task that is the starting time of the tasks processing;
- MIP algorithm using the domain as a set of binary variables. Each variable of the domain represents if a task begins in a instance of the time;
- CP using the domain as a integer variable for each task that is the starting time of the tasks processing.
- CP using the domain as an integer variable for each task that is the starting time of the tasks processing. The difference between this method and the previous are some programming issues. This method uses a built-in constraint provided by the software used, unlike in the previous one that uses a simple constraint.

In each method above, we have some characteristic in common thus we can group the methods in different classes. For example, the first, third and the fourth methods have the same representation of the solutions. The first and the second methods are techniques from LP.

These methods are applied to several different instances of JSS problem. The instances are generated randomly given the number of jobs, number of machines, the maximum value of the tasks duration and the method to generate the durations of the tasks. Given different values to the characteristic of the generation parameters, we can group the instances in classes. For example, an instance with 3 jobs and 4 machines and an instance with 3 jobs and 2 machines belong to the class of instances with 3 jobs. As we said above, on of the objectives of the

work it's to compare the performance of the methods proposed. So, the results of the experiments effectuated, we have not only compare the performance of the methods individually in all instances, but also we have analyze the performance of the classes of methods within the classes of instances describe above. We want to find out if there are some characteristic of the methods or/and of the instances that influences more the performance.

The Job-Shop Scheduling problem and details about the generation of instances are described in Section 2. The methods used in this work are describe in detail and explained in Section 3 the methods of LP and in Section 4 the methods of the CP. The experiments and results are in Section 5 that are discussed in Section 6. We present the conclusions about this work in Section 7.

2 Basic Job-Shop Scheduling

2.1 Description

The deterministic job-shop scheduling problem can be seen as the most general of the classical scheduling problems. Formally, this problem can be described as follows. A finite set J of n jobs $\{J_1, J_2, \dots, J_n\}$ has to be processed on a finite set M of m machines $\{M_1, M_2, \dots, M_m\}$. Each job J_i must be processed once on every machine M_j , so each job consists of a chain of m tasks. Let O_{ij} represent the operation of job J_i on machine M_j , and let p_{ij} be the processing time required by task O_{ij} .

The operations of each job J_i have to be scheduled in a predetermined given order, i.e. there are precedence constraints between the operations of each job J_i . Let $\prec$ be used to denote a precedence constraint, so that $O_{ik} \prec O_{il}$ means that job J_i has to be completely processed on machine M_k prior to being processed on machine M_l . Each job has its own flow pattern through the machines, so the precedence constraints between operations can be different for each job. Other additional constraints also have to be satisfied. Each machine can only process one job at a time (capacity constraints). Also, preemption is not allowed, so operations cannot be interrupted and must be fully processed once started. Let t_{ij} denote the starting time of operation O_{ij} . The objective is to determine starting times t_{ij} for all operations, in order to optimize some objective function, while satisfying the precedence, capacity and no-preemption constraints. The duration in which all operations for all jobs are completed is denoted as the makespan $C_{\max}$. In this paper, we consider as objective function the minimization of the makespan:

$$\begin{aligned} C_{\max}^* &= \min(C_{\max}) \\ &= \min_{\text{feasibleschedules}} (\max(t_{ij} + p_{ij})), \\ &\quad \forall J_i \in J, M_j \in M. \end{aligned}$$

In following, we present an example of a JSS instance. The instance has 3 jobs and 3 machines. In Table 1, we describe the data of the instance that includes the processing duration and the order within the job of all the tasks. In the Table 1, we can obtain the information of the tasks, for example, in the first line

of the data we conclude that the tasks from the job 1 (first column) processed in machine 1 (second column) is processed in 1 unit of the time (third column) and is the second (last column) to be processed in the job that belongs.

Table 1. Information of a JSS instance

Job	Machine	Duration	Order
1	1	1	2
1	2	3	1
1	3	4	3
2	1	4	2
2	2	2	1
2	3	4	3
3	1	4	3
3	2	4	2
3	3	2	1

Above, we defined that the objective is to determine the starting time t_{ij} for all tasks, the set of all starting times is called a schedule. In Figure 1, we show two schedules of two possible solutions for the example above. The x-axis is represented the time and in y-axis the job identification. The color rectangles are the tasks and those tasks with the same color are processed in same machine with the identification number at the top left.

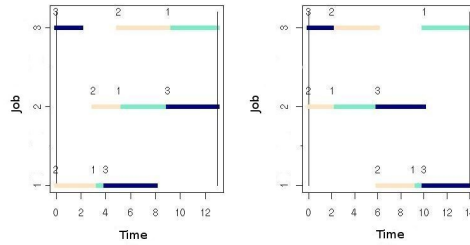


Fig. 1. Two possible schedules

However, it isn't mandatory that the search of the solution be in the starting time solution-space. The solution-space is the set of all solutions and starting time is a way to represent the solution for the problem. There are others possible representations. One example is a list of priority list between the tasks for each machine [1].

The job-shop scheduling problem is NP-hard [2, 6], and notoriously difficult to solve. Many papers have been published on the job-shop scheduling problem. A comprehensive survey of job shop scheduling techniques can be found in [4].

2.2 Instance Generation

In this section, we describe the method for generate different instances of JSS. We need to define the number of jobs, number of machines, the maximum duration and the method for generate the duration of the tasks. There are several methods for generate the duration of the tasks concerning the probability distribution type and the type of the tasks target. In this work, we use the method called No-Correlated for generate the duration of the tasks. This method uses one probability distribution to generate different values for each tasks that represents the duration. The value must be between 1 and the maximum duration defined. The probability distribution can be an Uniform Distribution [8] or a Gaussian Distribution [9], with average the half value of the maximum duration value. For generate an instance, we have to define the processing order, for each job, of the tasks. For this purpose, we generate N permutations randomly from a sequence of integers from 1 to M . The order of each job is the sequence obtained. For example, an instance with 3 jobs and 4 machines and a sequence 4 2 3 1, for the job 1, obtained with randomly permutation of the sequence 1 to 4. So, the first task to be processed in job 1 is the task to be processed in machine 4, the second task to be processed in job 1 is the task to be processed in machine 2, and so on.

3 Linear Programming

A linear programming problem may be defined as the problem of maximizing or minimizing a linear function subject to linear constraints. The constraints may be equalities or inequalities. Formally, a linear programming problem is to find an n-vector, $X = (X_1, \dots, X_n)^T$, to maximize/minimize

$$c^T x = c_1 X_1 + \dots + c_n X_n$$

subject to the constraints

$$\begin{aligned} a_{11}X_1 + a_{12}X_2 + \dots + a_{1n}X_n &\leq b_1 \\ a_{21}X_1 + a_{22}X_2 + \dots + a_{2n}X_n &\leq b_2 \\ &\vdots \\ a_{m1}X_1 + a_{m2}X_2 + \dots + a_{mn}X_n &\leq b_m \end{aligned}$$

and

$$X_1 \geq 0, X_2 \geq 0, \dots, X_n \geq 0$$

The following models uses the parameter:

$nrJobs$ is the number of jobs;

$nrMachines$ is the number of machines;

p_{jm} is the duration of the task from job j processed in machine m ;

maq_{jo} is the machine of the task from job j that is processed in the order o

3.1 Model 1

In this section, we describe one model used. In this model, the solution for the problem is represented as binary variables for each instant of time and each task. Each variable is 1 if in that instant the task began case contrary the value is 0. Let's formally describe the model. The variable that represents the domain of the method are:

$$X_{jmt} = \begin{cases} 1 & \text{if the task from job } j \text{ processed in machine } m \\ & \text{starts in instant } t \\ 0 & \text{case contrary} \end{cases}$$

$C_{\max}$ is the duration in which all operations for all jobs are completed;

The model (M_1) is the following:

$$v(M_1) = \text{minimize } C_{\max} \quad (1)$$

$$\forall j \in J, o \in \{2, \dots, |O|\}, \sum_{t \in T} t \times X_{j,maq_{jo},t} \geq p_j m + \sum_{t \in T} t \times X_{j,maq_{j(o-1)},t} \quad (2)$$

$$\sum_{j \in J} \sum_{k \in \{\max(0, t-d_{jm}) \dots t\}} X_{jmk} \leq 1 \quad (3)$$

$$p_{j,maq_{j|M|}} + \sum_{t \in T} t \times X_{j,maq_{j,nrMachines},t} \leq C_{\max} \quad (4)$$

$$\forall j \in J, \forall m \in M, \sum_{t \in T} X_{j,m,t} = 1 \quad (5)$$

The goal is defined in (Equation (1)), i.e., to minimize the makespan ($C_{\max}$). The makespan $C_{\max}$ is the time that the processing of all tasks are finish as in Equation (4).

The processing order of the tasks, for each job, is imposed in Equation (2) and the capacity of one task per time in each machine is in Equation (3). In Equation (2), we have that the start time ($\sum_{t \in T} t \times X_{j,maq_{jo},t}$) of the task from job j and processed in machine of the order o in job j (maq_{jo}) is higher than the end time of the previous one (in order $o-1$) in the same job j . In Equation (3), the second sum part equal to one if in instant t , the task from job j processed in machine m is being processed. So, for each machine in each instance, the sum in all jobs must be 0 or 1. Finally, the Equation (5) impose that a task can only begin one time.

3.2 Model 2

In this section, we describe one model used. In this model, the solution for the problem is represented as the starting times for each task. Let's formally describe the model. The variable that represents the domain of the method are:

$Y_{mj_1j_2} \begin{cases} 1 & \text{if, in machine } m, \text{ the task from job } j_1 \text{ is processed} \\ & \text{first than the task from job } j_2 \\ 0 & \text{if, in machine } m, \text{ the task from job } j_2 \text{ is processed} \\ & \text{first than the task from job } j_1 \end{cases}$
 X_{jm} is the starting of processing time of the task from job j processed in machine m ;
 $C_{\max}$ is the duration in which all operations for all jobs are completed;

The model (M_2) is the following:

$$v(M_2) = \text{minimize } C_{\max} \quad (6)$$

$$\forall j \in J, \forall m \in M : X_{jm} \leq K \quad (7)$$

$$\forall j \in J, \forall m \in M : X_{jm} \geq 0 \quad (8)$$

$$\forall j \in J, \forall o \in \{2, \dots, nrMachines\} : X_{jmaq_{j(o-1)}} + p_{jmaq_{j(o-1)}} \leq X_{j,maq_{jo}} \quad (9)$$

$$\forall j_1 \in J, \forall j_2 \in J, \forall m \in M : X_{j_1m} \geq X_{j_2m} + p_{j_2m} - K \times Y_{mj_1j_2}, j_1 \neq j_2 \quad (10)$$

$$\forall j_1 \in J, \forall j_2 \in J, \forall m \in M : X_{j_2m} \geq X_{j_1m} + p_{j_1m} - K \times (1 - Y_{mj_1j_2}) \quad (11)$$

$$\forall j \in J : p_{j,maq_{j,nrMachines}} + X_{j,maq_{j,nrMachines}} \leq C_{\max} \quad (12)$$

The goal is define in (Equation (1)), i.e., to minimize the makespan ($C_{\max}$). The makespan $C_{\max}$ is the time that the processing of all tasks are finish as in Equation (12). The bounds of the variable X_{jm} are defined in Equation (7) and (8).

The processing order of the tasks, for each job, is imposed in Equation (9) and the capacity of one task per time in each machine is in Equation (10) and Equation (11). The Equation (9), we have that the start time ($X_{j,maq_{jo,t}}$) of the task from job j and processed in machine of the order o in job j (maq_{jo}) is higher than the end time of the previous one (in order $o - 1$) in the same job j . Using the representation of this model (X_{jm}) as the starting time of the task j in machine m , it's enough to impose the restriction if two tasks of same machines cannot be processed at same time. We can use the expression $X_{j_1m} + p_{j_1m} < X_{j_2m} \vee X_{j_2m} + p_{j_2m} < X_{j_1m}$. However, this expression is non-linear. So, we use an auxiliary variable $Y_{mj_1j_2}$.

4 Constraint Programming

The Constraint Programming is a programming language that is oriented to relationships or constraints among entities.

The Constraint Programming can be used for tackling combinatorial problems such as the JSS. For combinatorial problems, we deal with finite domains and the CP class of techniques used for finite domains are called Constraint Languages Programming for Finite Domains (CLPFD). For CLPFD, it must be define the search variables, the domain of all variables used in the search and the constraints. The domain of a variable is the set of possible values that

the variable can take, but not all points (all variables with a value) are feasible solutions for the problem. These feasible solution are validated by constraints. Given a domain for all variables and the constraints between the variables, the search is done using backtracking search [7].

In this work, we use for CP the Sicstus ¹ software that it's a Prolog engine. The domain of the variables is defined using the term *domain(+Variables, +MinValue, +MaxValue)* where *+Variables* is a list of variables, *+MinValue* is the minimum value of the domain and *+MaxValue* is the maximum value of the domain. For the backtracking search is use the built-in *labeling(: Options, +Variables)* predicate. The *: Options* argument controls the order in which variables are selected for assignment (variable choice heuristic), the way in which choices are made for the selected variable (value choice heuristic), and whether all solutions or a single, optimal solution should be found [5]. In this work, we only define the objective of the search that is to minimize the makespan, so *Options* = [*minimize(C_{max})*]. The argument *Variables* is a list of the variable where the backtracking search is done.

4.1 Model 1

In this section, we describe a model using CP for JSS. The search variable is X_{jm} the end of processing time of the task from job j processed in machine m . The domain of the variables X_{jm} is from 0 to MAX where $MAX = \sum_{i \in J, j \in M} p_{ij}$. The objective is to *minimize(C_{max})* where *maximum(C_{max}, X_{jm})*, i.e, minimize the makespan that is the maximum of the end time of processing time of the tasks.

The restrictions are imposed by the user predicates *precedenceConstraint/1* and *machineConstraint/1*. The predicate *precedenceConstraint(+AllTasksJobs)* is the constraint of the order of processing between tasks of the same job. The parameter *AllTasksJobs* is a list of all tasks of the problem represented with the term *task(O_i, D_i, E_i, H_i, Id_i)* where O_i is the start time, D_i the non-negative duration, E_i the end time, H_i the resource consumption (if positive) or production (if negative), and Id_i is the task identifier [5]. The predicate *precedenceConstraint/1* run the list of lists where each is grouped the tasks belonging to same job. For each list, that are ordered the tasks by the same order impose by the problem, the predicate run two by two tasks and impose that

$$E_1 \# = < S_2$$

where E_1 is the final time of a task 1 that is previous to the task 2 with start time of processing S_2 . The second constraint *machineConstraint(+AllTasksMachines)* impose that only one task be processed by time in each machine. The parameter *AllTasksMachines* is a list with terms *task(O_i, D_i, E_i, H_i, M_i)* where O_i is the start time, D_i the non-negative duration, E_i the end time, H_i the resource consumption (if positive) or production (if negative), and M_i a machine identifier[5].

¹ <http://www.sics.se/sicstus.html>

This predicate *machineConstraint/1* run the list and compares all pairs of tasks and if the two are processed in same machine then

$$|med_1 - med_2| \# \geq \frac{p_1 + p_2}{2} \quad (13)$$

where $med_i = end_i - \frac{duration_i}{2}$ is the middle point between the start and end time of the task O_i , p_i is the duration of task O_i and end_i is the end time of processing the task O_i . The Equation (13) means that the distance between the two middle points of the tasks ($|med_1 - med_2|$) must be higher than the half value of the duration of the two tasks. We show that this Inequation (13) imply the desired constraint of the problem, that is,

$$end_1 < end_2 \Rightarrow end_1 < start_2 \vee end_2 < end_1 \Rightarrow end_2 < start_1 \quad (14)$$

The demonstration is following one:

$$|med_2 - med_1| > \frac{d_1 + d_2}{2} \quad (15)$$

$$\Leftrightarrow med_2 - med_1 > \frac{d_1 + d_2}{2} \vee med_2 - med_1 < -\frac{d_1 + d_2}{2} \quad (16)$$

$$\Leftrightarrow start_2 > end_1 \vee end_2 < start_1 \quad (17)$$

If $end_1 < end_2$ then $start_1 < end_2$, by definition and with the result from (17), $start_2 > end_1$ is true. The same with $end_1 > end_2$ then $start_2 < end_1$, by definition, and with the result from (17), $end_2 < start_1$ is true.

4.2 Model 2

As in model from Section 4.1, the solution variable is the processing operation end time. The model is similar excepts instead of using the predicate *machineConstraint/1* for restrict the capacity of processing one task per machine at same time, we use a built-in predicate *cumulatives/3* provided by Sicstus software. The predicate *cumulatives(+AllTasksMachine, +AllMachines, +Options)* from a set of n tasks and m machines maintains the limit of tasks to be processed by a machine at same time. The variable *AllTasksMachine* is a list of term *task(O_i, D_i, E_i, H_i, M_i)* where O_i is the start time, D_i the non-negative duration, E_i the end time, H_i the resource consumption (if positive) or production (if negative), and M_i a machine identifier. The predicate *cumulatives* needs information about the limits of tasks for each machine. This information is in list *AllMachines* with the a term *machine(M_j, L_j)*, for each machine, where M_j is the identifier and L_j is the resource bound of the machine. In this problem of Job-Shop Scheduling, the resource bound is a superior limit of 1 task, so $L_j = 1, \forall j \in J$ and in variable *Options* it must have the term *bound(upper)*. More information about the predicate *cumulatives/3* is in the Sicstus manual [5]

5 Experiments

The experiments are applied to randomly generated instances described in Section 2. For generate the instances, we defined the dimension (number of jobs and machines), the maximum duration and method to generate the duration times. For this work, it was used the combination between all of the following parameters:

- Number of jobs: 5(only CP methods and LP-model-2),4 and 3;
- Number of machines: 5(only CP methods and LP-model-2), 4 and 3;
- Maximum duration: 5 and 10 (only CP methods and LP-model-2);
- Methods for duration: No-Correlated with Uniform Distribution and No-Correlated with Gaussian Distribution;

For each combination, it was created 20 instances of the same type. For example, there are 20 instances with 3 jobs and 3 machines with maximum duration of the operation of 5 and the were generated with the uniform distribution. The algorithms are two models of LP and two models from CP. The LP models are described in Section 3 are: the algorithm that we called LP-model-1 is described in Section 3.1 and the algorithm that we called LP-model-2 is described in Section 3.2. The CP models are described in Section 4 are: the algorithm that we called CP-model-1 is described in Section 4.1 and the algorithm that we called CP-model-2 is described in Section 4.2. For the method LP-model-1, we didn't run in instances with 5 job or 5 machines or 10 for maximum duration , because of very high running time. The software used in the LP algorithms is the "GNU Linear Programming Kit" (glpk) available in ². For the CP algorithm is used a commercial software, called Sicstus as it was mentioned before. For each method and instance, we repeated the experiment 10 times. In Table 2, we present the average results of the time needed to the algorithms or classes of algorithms (in the column 1) to solve the instances grouped by some characteristic. The first group of lines (LP and CP) is the average of time needed to solve by the algorithms of LP and CP. The second group (binaryStart and startTime), is the average of duration needed for solving the instances for the algorithms with the same domain representation. For example, the values of the column "All" is for all instances and in subcolumn "3" from column "Job" is the value for instances with 3 jobs. The fourth line of values are the average of time that algorithms with domain representation as the start time of the tasks (LP-model-2,CP-model-1 and CP-model-2) needed to solve each class of instances. In Table 3, we show the percentage of instances that each algorithm have the best performance comparing to the others. In both table, the comparison between two methods are made in the same set of instances. So, when comparing with method LP-model-1, we didn't used instance with 5 jobs or 5 machines or 10 for maximum duration, since we didn't run this type of instance with this method. In Table 3, the group A are the instances used for comparing the 4 methods and the group B are the instances used only for the methods of CP and the LP-model-2.

² <http://www.gnu.org/software/glpk/>

Table 2. Information of the average duration for solving the instances of each class

	All	Nr.Jobs			Nr.Machines			Distrib.	
		3	4	5	3	4	5	unif.	gauss.
LP	28.318	2.490	54.146	0.337	4.928	51.708	3.862	27.950	28.686
CP	0.089	0.086	0.0931	12.329	0.086	0.093	10.230	0.088	0.090
binaryStart	56.604	4.967	108.241	-	9.829	103.379	-	55.869	57.339
startTime	0.070	0.061	0.079	8.33	0.066	0.074	6.889	0.069	0.071
LP-model-1	56.603	4.967	108.241	-	9.829	103.379	-	55.869	57.339
LP-model-2	0.033	0.014	0.051	0.337	0.02	0.038	0.207	0.032	0.033
CP-model-1	0.090	0.086	0.094	8.77	0.087	0.093	7.050	0.089	0.091
CP-model-2	0.088	0.085	0.093	15.88	0.085	0.092	13.410	0.088	0.089

Table 3. Percentage of instance that an algorithm have the best performance

Group	LP-model-1	LP-model-2	CP-model-1	CP-model-2
A	0	95.63	1.87	2.5
B	-	62.67	19.82	17.5

6 Discussion

There are several observations that we can make from the results from Table 2. The first one, it's that ranking of methods about the performance is the same whatever is the class of instances. We can see that the method with best average performance is LP-model-2, then CP-model-1, CP-model-2 and finally LP-model-1. However, in average, the two models from CP are better than the two of LP. This is because the model 1 of LP have very poor performances. It's interesting to observe that values of the performance of the LP methods increase more than the methods of CP methods when the dimensions of the instances increases. About the distribution used for generate the duration, it have small differences between the Gaussian and Uniform distributions. However, the average performance for all method is slightly lower for instances with durations generated with Uniform distribution even when the maximum duration is very low (5) that makes the samples of the two distribution not very different. Perhaps, if we increase the maximum duration we can see more amplified the trend observed. From Table 3, we observe that the method LP-model-2 is better than the others in 96% of the instances. When we increase the dimensions of the job, machine or maximum duration the dominance of the method LP-model-2 is much lower. The percentage of winnings is only 62%.

7 Conclusion

In this work, we present several models for Linear Programming and Constraint Programming of Job-Shop Scheduling. The differences between the models are the representation of the solution and the programming language used. We compare the average of time needed for each method find the optimal solution in

different class of instances. In average, the best method is the same for all classes of the instances - a model where the representation of the solution is the starting time of the tasks using the Linear Programming techniques. There are evidences that when we increase the dimension of instances the ranking between methods can change since the performance of LP methods increase more than the CP methods and the number of instances that CP methods wins have a high increase only adding a job or a machine. Another trend noted is that the instances with durations generated with Gaussian distribution can be harder for solve, but the differences are very low for any conclusion. The model of the problem used is very important. We can see that for two different models for LP, we have a performance so different. The results between the two models from CP are surprising. We expect better results with CP-model-2 since the *cumulative* is an optimized to schedule tasks.

For future work, we should increase the number of representation and experiment all methods in more instances with higher dimensions and generate with different methods for a higher diversification of the instances. The objective is to find some answer to the questions raised during the work.

References

1. Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—i: representation. *Comput. Ind. Eng.*, 30(4):983–997, 1996.
2. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, San Francisco, California, 1979.
3. A. Jain and S. Meeran. A state-of-the-art review of job-shop scheduling techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.
4. A. S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113:390–434, 1999.
5. Intelligent Systems Laboratory. *SICStus Prolog Users Manual*. Swedish Institute of Computer Science, PO Box 1263 SE-164 29 Kista, Sweden, 4.0.4 edition, June 2008.
6. J. K. Lenstra and A. H. G. Rinnooy Kan. Computational complexity of discrete optimisation problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.
7. Kim. Marriott and P. J. Stuckey. *Programming with constraints : an introduction*. MIT Press, Cambridge, Mass., 1998.
8. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.
9. Jean-Paul Watson, Laura Barbulescu, Adele E. Howe, and Darrell Whitley. Algorithm performance and problem structure for flow-shop scheduling. In *AAAI/IAAI*, pages 688–695, 1999.
10. David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.