Chapter 19
Understanding Complexity in a Practical Combinatorial Problem Using Mathematical Programming and Constraint Programming

Beatriz B. Oliveira and Maria Antónia Carravilla

Abstract Optimization problems that are motivated by real-world settings are often complex to solve. Bridging the gap between theory and practice in this field starts by understanding the causes of complexity of each problem and measuring its impact in order to make better decisions on approaches and methods. The Job-Shop Scheduling Problem (*JSSP*) is a well-known complex combinatorial problem with several industrial applications. This problem is used to analyse what makes some instances difficult to solve for a commonly used solution approach – Mathematical Integer Programming (MIP) – and to compare the power of an alternative approach: Constraint Programming (CP). The causes of complexity are analysed and compared for both approaches and a measure of MIP complexity is proposed, based on the concept of load per machine. Also, the impact of problem-specific global constraints in CP modelling is analysed, making proof of the industrial practical interest of commercially available CP models for the *JSSP*.

Keywords Job-shop scheduling problem · Mathematical programming
Constraint programming · Global constraints · Complexity

19.1 Introduction

The Job-Shop Scheduling Problem (*JSSP*) and its many extensions and variations have been thoroughly studied on the field of Optimization, due to its computational difficulty. This problem is NP-hard [8] and was even qualified as "one of the most computationally stubborn combinatorial problems" [3]. For these reasons, it will be analysed in this paper as a starting point to understand the meaning of *complexity* for both approaches: Mathematical Programming and Constraint Programming.

B. B. Oliveira (✉) · M. A. Carravilla
INESC TEC and Faculty of Engineering, University of Porto, Porto, Portugal
e-mail: beatriz.oliveira@fe.up.pt

M. A. Carravilla
e-mail: mac@fe.up.pt

A. I. F. Vaz et al. (eds.), *Operational Research*, Springer Proceedings
in Mathematics & Statistics 223, https://doi.org/10.1007/978-3-319-71583-4_19

The *JSSP* summarily consists of scheduling J jobs to be processed in M machines with the goal of minimizing the time it takes for all the operations to end (makespan). Each job takes a fixed processing time in each machine and there is a certain sequence of machines that it should follow – each job has its own specific sequence. The operations must be completed – once a job starts on a specific machine, it must take the full processing time. Also, a machine can only handle one job at the time.

The *JSSP* has numerous applications in the industry, namely on production planning and scheduling. In the past years, several extensions and adaptations are arising in the literature [9, 12], many concerning real applications.

This paper has two main objectives. Firstly, it aims to gain some insights regarding the factors that induce complexity on the Mathematical Programming approach to this problem. As this approach is commonly used to solve real-world problems, understanding these factors can help practitioners identify beforehand the challenges and/or opportunities that arise from the characteristics of the problem and therefore make better decisions regarding e.g. solution methods.

Secondly, it aims to understand whether the alternative approach of Constraint Programming (CP) is useful to tackle to the "difficult to solve" instances. Above all, there is the goal to understand the power of the commercially available CP models for the *JSSP* and, consequently, whether they are a realistic solid alternative for practitioners.

The paper is thus divided into two main sections. The first section is related to the Mathematical Programming approach. It presents a mixed-integer formulation for this combinatorial problem that is commonly used in the literature and assesses its computational power with eighty-two literature instances. The main objective is to understand which factors are more relevant to explain the complexity and "stubbornness" of the *JSSP*, namely to understand the impact of size. Therefore, on Sect. 19.2.1, the MIP model is presented; the computational results are discussed on Sect. 19.2.2. In this section, a potential measure of instance *complexity* is also presented and tested. The second part is devoted to the Constraint Programming approach to the *JSSP* (Sect. 19.3). Here, three different CP formulations are presented (Sect. 19.3.1) and tested against the performance of the MIP model (Sect. 19.3.2). One of the CP formulations is a model commercially available, including global constraints specifically developed for this problem. The main goal is herein to understand the power and limitations of CP programs, namely the commercially available one. Finally, on Sect. 19.4 the main conclusions are drawn.

The overall purpose of this paper is to draw attention to the impact of understanding and measuring impact when tackling specially complex problems, such as real-world applications of the *JSSP*. This purpose is materialised in the following contributions:

- A quantitative-based discussion on factors that induce complexity for a MIP approach to the *JSSP*, namely the total size of the instance and relationship between number of jobs and machines;
- A proposed measure of MIP-complexity of *JSSP* instances based on load per machine;

- A comparison of the computational power of different CP formulations for the *JSSP*, with insights related with commercially available CP models, which show significant advantages when tackling complex instances;
- A comparison between CP and MIP approaches to the *JSSP*, establishing their advantages and shortcomings;
- A quantitative evidence of the benefits of using commercially available CP models to tackle complex *JSSP* instances and settings.

19.2 Mathematical Programming Approach

19.2.1 MIP Model

This model follows the one proposed by Applegate and Cook [3], based on the *disjunctive programming problem* presented in Balas [5].

Sets and indexes

$\mathscr{J}$	set of jobs ($|\mathscr{J}| = J$)
$\mathscr{M}$	set of machines $|\mathscr{M}| = M$)
$i, j \in \mathscr{J}$	index of jobs ($i, j = \{1, \dots, J\}$)
$k \in \mathscr{M}$	index of machines ($k = \{1, \dots, M\}$)

Parameters

p_{ik}	the kth operation (machine) for job i
t_{ik}	processing time of job i on machine k
L	significantly large value

Decision Variables

x_{ik}	starting time of job i on machine k
y_{ijk}	(= 1) if job i is scheduled before job j in machine k
c	makespan

Mathematical Program

$$\min \quad c \tag{19.1}$$

$$\text{s.t.} \quad x_{i\,p_{ik}} \geq x_{i\,p_{i\,k-1}} + t_{i\,p_{i\,k-1}}, \qquad \forall i \in \mathscr{J}, \forall k \in \mathscr{M} \backslash \{1\} \tag{19.2}$$

$$x_{ik} \geq x_{jk} + t_{jk} - L y_{ijk}, \qquad \forall k \in \mathscr{M}, \forall i, j > i \in \mathscr{J} \tag{19.3}$$

$$x_{jk} \geq x_{ik} + t_{ik} - L(1 - y_{ijk}), \qquad \forall k \in \mathscr{M}, \forall i, j > i \in \mathscr{J} \tag{19.4}$$

$$c \geq x_{i\,p_{iM}} + t_{i\,p_{iM}} \qquad \forall i \in \mathscr{J} \tag{19.5}$$

$$x_{ik}, c \geq 0, \qquad \forall i \in \mathscr{J}, k \in \mathscr{M} \tag{19.6}$$

$$y_{ijk} \in \{0, 1\}, \qquad \forall i, j > i \in \mathscr{J}, k \in \mathscr{M} \tag{19.7}$$

The goal of this model is to minimize the total makespan (Eq. 19.1). The first constraint (Eq. 19.2) ensures that a job can only start to be processed in a machine if it has finished its processing on the previous machine (clearly, this does not apply to the first operation of the job).

The second and third constraints are related to the order of the jobs in each machine (Eqs. 19.3 and 19.4). They state that if a job precedes another job in a certain machine, then its starting time is not limited by the latter. However, if a job comes after another job in a machine, it can only start when this one has finished. The following constraint (Eq. 19.5) states that the total makespan is given by the job that finishes the processing last (i.e., based on the last machine: the Mth machine the job will go through).

Finally, the decision variable that sets the starting time of each job on each machine as well as the makespan should be non-negative, and the precedence variable is set as a binary (Eqs. 19.6 and 19.7).

19.2.2 Computational Tests and Results

In order to understand the behaviour of this MIP model, 82 instances from the literature were run. The computational tests were run on a Toshiba Personal Computer with 16 Gigabyte of RAM memory, and with a processor Intel(R) Core(TM) i7-4600U CPU @ 2.10 GHz 2.70 GHz. The model was developed in a OPL Project in IBM ILOG CPLEX Optimization Studio 12.6.1.0 and the MIP Solver used was CPLEX. The time limit set for the resolution of each instance was 30 min. The instances' characteristics and solution can be found on the Appendix (Table 19.4) and the main results are summarised on Table 19.1.

Table 19.1 Summary of the average results of the MIP model

J	M	#instances	Avg. sol. time (s)	Sol. time std. dev. (s)	Avg. gap (%)	Gap std. dev.
6	6	1	0	0	0	0
10	5	5	13	18	0	0
10	10	18	142	286	0	0
15	5	5	*1.800*	0	29	5%
15	10	5	*1.800*	0	11	8%
15	15	5	*1.800*	0	12	3%
20	5	6	*1.800*	0	47	5%
20	10	10	*1.800*	0	39	9%
20	15	8	*1.800*	0	37	2%
20	20	4	*1.800*	0	22	4%
30	10	5	*1.800*	0	55	2%
50	10	10	*1.800*	0	76	3%

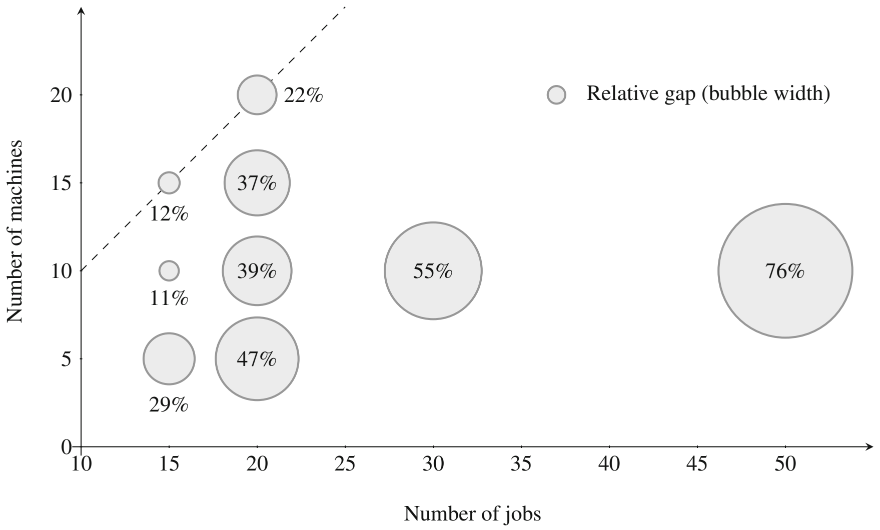

Fig. 19.1 Average MIP relative optimality gap attained per instance size

As it is possible to observe on Table 19.1, the solver was only able to prove optimality within the time limit (30 min) for the twenty-four smallest instances.[1] The results are herein analysed for both resulting groups:

Instances Solved to Optimality:

For the smallest instances, solved to optimality, the average solution time increases with the size of the instances. Nevertheless, the magnitude of the variability of solution times for instances of the same size (given by its standard deviation, in Table 19.1) suggests that the size is not the only factor that translates into complexity and, consequently, to longer run times.

Instances Not Solved to Optimality:

For the fifty-eight instances that were not solved to optimality in 30 min, the value of the optimality gap attained may shed some light on the matter of complexity. Firstly, it is interesting to compare the average results of the instances of type $(J, M) = (20, 5)$ and of type $(J, M) = (15, 15)$. The latter are bigger in terms of number of variables and constraints, yet the solver is able to achieve significantly lower gaps for them (47% vs. 12%). This is another evidence that the "easiness to solve" is not only linked with the size of the problem.

Figure 19.1 represents the average gap attained for instances of the same size. The horizontal axis represents the number of jobs in the instances, the vertical axis represents the number of machines, and the width of the bubbles depicts the size of the average optimality gap. On an horizontal reading, it is possible to see that the optimality gap increases as instances tackle an increasing number of jobs. This may

[1]Instances `ft06`, `ft10`, `abz5`, `abz6`, `la01–la05`, `la16–la20`, `orb01–orb10`.

be due to the fact that the number of jobs influences the most the size and thus, at some (limited) degree, the complexity of the model and its difficulty to solve to optimality. In fact, due to the structure of the binary variable $y_{ijk}, \forall i, j > i \in \mathscr{J}, k \in \mathscr{M}$, it can be seen that $J = |\mathscr{J}|$ has more impact on the number of variables that $M = |\mathscr{M}|$. Due to the structure of Constraints 19.3 and 19.4, it also strongly influences the number of constraints. In conclusion, this horizontal increase seems a natural consequence of the size of the instances, which is heavily dependent on the number of jobs (J).

However, if one considers a fixed number of jobs (for example, $j = 20$ on the horizontal axis) and analyses the impact of the number of machines in the optimality gap, the effect is reversed. It seems that, for a fixed number of jobs, the optimality gap achieved in the same run time is better for more balanced instances, with the same number of jobs and machines (even if that means a bigger number of variables and constraints).

Regarding the observable "easiness" to solve balanced instances (in terms of number of jobs and machines), one may consider several hypotheses. Please consider the following definition of optimality gap for a minimization problem: $Gap = (UB - LB)/LB$, where LB stands for the *lower bound*, given by the best relaxed solution found on the Branch-and-Bound nodes not pruned at a time; and UB stands for the *upper bound*, which is given by the best integer solution found so far. Due to the complexity of the methods and heuristics utilized by CPLEX to speed up the Branch-and-Bound procedure, it is difficult to understand how the *lower bound* evolves in different instances. Therefore, the main hypothesis presented to explain this behaviour is related with the update velocity of the *upper bound*. Analysing the data herein presented, it may be possible to say that the balanced structure (i.e., a similar number of jobs and machines) makes it easier for the solver to find admissible (integer) solutions and thus update the *upper bound*.

One could also argue that the effect of the increasing number of machines (that reduces the optimality gap for the same number of jobs) is not a matter of balance but of the existence of alternative admissible solutions. That is to say that it is not the *balance between the two dimensions* of the problem that is at stake but *only the number of machines*. In fact, if a certain problem is characterized by a big number of machines (and if the precedence of operations is not too restraining) it is possibly easier to find alternative optima, as well as admissible integer solutions that update the *upper bound*.

Other factors may influence the complexity of the instance, namely the precedences and processing times that may hinder or help the procedure to find admissible integer solutions, which may explain the high standard deviation of the solution times observed for the twenty-four smallest instances.

A Measure of Complexity

In order to further understand the concept of instance complexity, a measure of the load of solicitations applied to each machine throughout the scheduling horizon was designed. Dividing the horizon in partitions of the same size, the goal was to understand whether the load applied in the partitions (the need or demand of jobs to be scheduled there) was constant, and what was the maximum value it achieved throughout the horizon. So, for some partition p of the horizon, the *needs-to-capacity ratio* (r) is given by:

$$r_p = \frac{\text{count}(ES_{ik} \in p)}{M} \quad , \forall i \in \mathscr{J}, \forall k \in \mathscr{M}$$

Here, M is, as before, the number of machines, and ES_{ik} stands for the *earliest start* of job i in machine k. The earliest start is defined by the precedence rules of each job:

$$ES_{ik} = \sum_m t_{im} \quad , \forall m \in \mathscr{M} : p_{io} = m, \; p_{io'} = k, \; o < o'$$

Here, as before, t_{im} stands for the processing time of job i on machine m and p_{io} represents the oth machine where job i is processed. Therefore, the earliest start of a job on a machine is the result of the sum of the processing times of that same job on the preceding machines. For example, if job 1 can only be processed in machine C after being processed in machines A and B then the *earliest start* of job 1 on machine C is equal to the time it takes to be processed in machines A and B. Therefore, the *needs-to-capacity ratio* is able to represent, at a certain extent, what are the needs of jobs to be scheduled in each machine, taking into consideration the precedences.

This measure was calculated for the eighty-two instances run. In order to adapt the partitions of the scheduling horizon to each instance, the length of the partitions was based on the instances' minimum value of the processing times:

$$\text{length} = 2 \cdot \min_{ik}(t_{ik}), \quad \forall i \in \mathscr{J}, \forall k \in \mathscr{M}$$

Figure 19.2 represents the evolution of this ratio for instances `la10` and `la37`. Both instances consider 15 jobs yet in `la10` there are 5 machines whilst in `la37` there are 15 machines. Despite the smaller size, the MIP solver achieved an optimality gap of 35% for the former (vs. 15% for the latter).

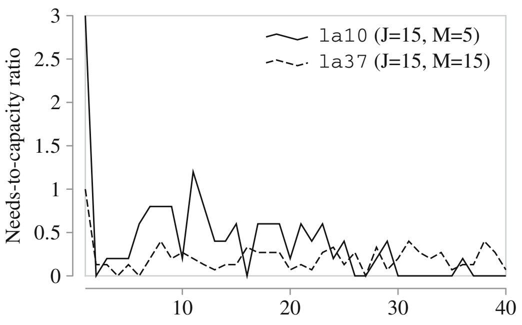

Fig. 19.2 Evolution of the *needs-to-capacity ratio* for the first 40 partitions of the scheduling horizon of instances `la10` and `la37`

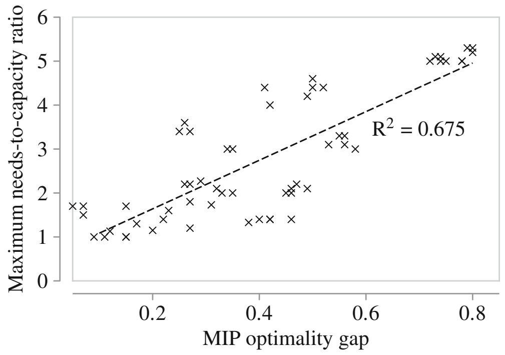

Fig. 19.3 Correlation between the optimality gap attained by the MIP solver and maximum *needs-to-capacity ratio* of the instances in Footnote 2

As explained before, the *needs-to-capacity ratio* represents, for each partition of time, the number of jobs whose *earliest start* in some machine falls within the partition, divided over the total number of machines. It is thus an average measure of the load per machine. Figure 19.2 is hence able to display the additional complexity of `la10`: the maximum and overall level of solicitations (needs) per machine is higher than in `la37` and has more variation throughout time. This supports the statement that the precedences and processing times in the instance may hinder the solver to find admissible integer solutions and influence the solution time/quality more than the number of variables and constraints.

In order to compare instances in a easier way, the *maximum* ratio attained throughout the partitions of the scheduling horizon was calculated for each instance. Figure 19.3 represents the correlation between the maximum ratio and the optimality gap attained by the MIP solver (for those instances in which the solver was unable to prove optimality within the time limit[2]). Table 19.5 in the Appendix also presents these metrics, together with the quartile in which each maximum ratio falls when comparing the maximum needs-to-capacity ratio of the eighty-two instances. This table also contains information regarding the number of partitions, their length, and the standard deviation of the ratio throughout the horizon.

Figure 19.3 represents each instance by its optimality gap and maximum *needs-to-capacity ratio*. It is possible to observe that the ratio increases as the gap increases, i.e. as the MIP solver finds it more and more difficult to reach (or prove) optimality. The R^2 statistics with a value of 67% shows a good correlation between these two metrics, which supports the hypothesis that the ratio is a good indicator of an instance's complexity for the MIP solver. That is to say, the relation between the load of jobs that, at a certain point in time, are soliciting the machines and the capacity the system has to respond to these solicitations (i.e. the number of machines) has a direct influence on the difficulty the solver has to reach and prove optimality for a given instance.

[2]Instances in which the MIP solver was unable to prove optimality within the 30 min time limit: `abz7–abz9`, `ft20`, `la06–la15`, `la21–la40`, `swv01–swv20`, `yn1–yn4`.

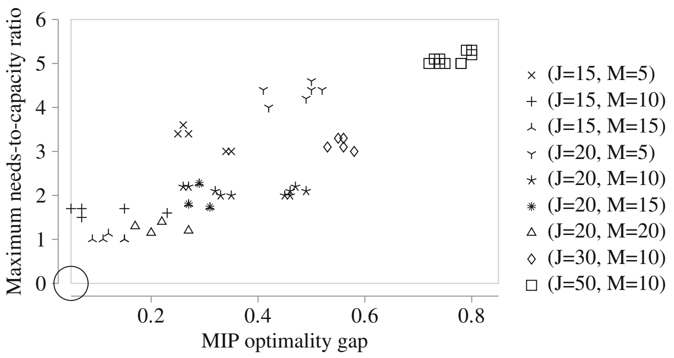

Fig. 19.4 Correlation between the optimality gap attained by the MIP solver and maximum *needs-to-capacity ratio* of the previous instances, categorized by size (number of jobs J and number of machines M)

Figure 19.4 shows the size characteristics (number of jobs and machines) of the instances in Fig. 19.3. It is interesting to see from a different perspective the conclusion drawn before: an increase of the size does not imply an increase of complexity. At the same time, if one considers a specific group of instances of the same size, videlicet the instances with $(J, M) = (20, 20)$ (triangle-shaped points), the maximum ratio attained is very similar amongst the instances while the optimality gap still shows some variation, not fully explained by the ratio (the dots representing the instances are spread horizontally on Fig. 19.4). This shows that are still other factors that may contribute to the complexity of the instance besides size and load of solicitations on the machines.

19.3 Constraint Programming Approach

"CP solving is based on intelligently enumerating all possible variable-value combinations" [13].

Constraint Programming is based on a flexible and intuitive modelling language, allowing for logic constraints, non-linear constraints, and almost every expression possible over the variables. It also requires that the domain of the variables (finite set of possible values) is set beforehand [13].

A CP program assigns values to variables without violating the constraints [4]. A search tree is built with several nodes, where each node is a solution, with some variables with a value already assigned to them. Each node then branches to several other nodes, each representing a similar solution but with other variable (from the remaining without assignment) allocated to some possible value. There are as many *child nodes* as there are possible values in the domain of the variable to assign [1]. The mechanism of constraint propagation consists on decreasing the domain of the variables not assigned yet due to the value assigned to a certain value in a iteration, and thus decreasing the number combinations to test [13].

Global constraints are a powerful element of CP programs. A global constraint is, by definition, a constraint that involves more than two variables. When designed to seize to the full potential the structure of the problem, these can be a powerful tool to facilitate constraint propagation and thus reduce significantly the search tree. It is important to recall that, although there are widely used global constraints (e.g. *alldiff*), some of the best performing global constraints that solve specially difficult problems are usually designed specifically for them. For the most studied problems in CP, global constraints have been designed and enhanced in the past years [10]. It is also important to note that in CP, unlike MIP problems, increasing the number of constraints may indeed be beneficial as it helps propagation.

Due to their characteristics, it is often the case that CP programs should perform well, by nature and theory, at finding admissible solutions, although only being able to prove optimality by explicitly searching all nodes and comparing their results. MIP solvers, on the other hand, although having more difficulty at finding admissible solutions, are able to prove optimality by setting upper and lower bounds and determining the optimality gap, i.e. it is not needed to *explicitly* explore the full solution space.

Scheduling problems are a classical application of CP due to their above-discussed "stubbornness". Therefore, specific well-performing global constraints have been designed and tested for it. Moreover, since it was previously inferred that the difficulty to find admissible integer solutions may be an important factor on the "stubbornness" of this problem (Sect. 19.2.2), Constraint Programming should be an appropriate approach to tackle this problem. So, in the next section, three different CP models are presented and tested. The first two are CP variations of the previously presented MIP model. The third one uses global constraints developed specifically for this problem and different decision variables.

19.3.1 CP Models

Three CP models are used in this paper to understand and compare the power of Constraint Programming models, namely and especially of global constraints.

The first model (CP1), is nearly a translation of the MIP model presented in Sect. 19.2.1. Since logic constraints can be used in CP, Constraints 19.3 and 19.4 (of the MIP model) can be translated into one disjunctive equation, and thus there is no need to use the auxiliary binary variables y_{ijk}. The second model (CP2) is a variation of CP1, where the *makespan* variable (c) is eliminated (as well as Constraint 19.5). Since CP does not require its model elements to be linear, the objective function can be stated as the minimization of the maximum value amongst several, which was not possible using Mathematical Linear Programming. Finally, the third model (CP3) is based on the IBM ILOG CPLEX Optimization Studio example *sched_jobshop* for

CP and seizes all the advantages of Constraint Programming: the decision variables' representation is more appropriate and two specifically developed global constraints are used.

19.3.1.1 CP1

Using the same sets, indexes and parameters of the MIP Model presented:

Decision Variables

$$x_{ik} \in \{0, \ldots, 10000\} \textit{ and integer} \text{ starting time of job } i \text{ on machine } k$$
$$c \in \{0, \ldots, 10000\} \textit{ and integer} \text{ makespan}$$

CP1 Model

$$\min \quad c \tag{19.8}$$
$$\text{s.t.} \quad x_{i\,p_{ik}} \geq x_{i\,p_{i\,k-1}} + t_{i\,p_{i\,k-1}}, \qquad \forall i \in \mathscr{J}, \forall k \in \mathscr{M} \backslash \{1\} \tag{19.9}$$
$$x_{ik} \geq x_{jk} + t_{jk} \quad \vee \quad x_{jk} \geq x_{ik} + t_{ik}, \qquad \forall k \in \mathscr{M}, \forall i, j > i \in \mathscr{J} \tag{19.10}$$
$$c \geq x_{i\,p_{iM}} + t_{i\,p_{iM}} \qquad \forall i \in \mathscr{J} \tag{19.11}$$

The main difference of this model (vs. the MIP model) is the absence of the y_{ijk} variables and the presence of the disjunctive Constraint 19.10. In fact, CP models allow for this kind of logic constraints, which state that either the inequality on the left or the inequality of the right must hold. That is to say that for each machine either job i waits for job j to end (if it is scheduled after it) or the other way around. This type of constraint allows for this disjunction "or-or" to be stated without the auxiliary binary variable that states whether or not job i precedes job j.

19.3.1.2 CP2

Using the same sets, indexes and parameters of the MIP Model presented:

Decision Variables

$$x_{ik} \in \{0, \ldots, 10000\} \textit{ and integer} \text{ starting time of job } i \text{ on machine } k$$

CP2 Model

$$\min \quad \max_i \{x_{i\,p_{iM}} + t_{i\,p_{iM}}\} \tag{19.12}$$
$$\text{s.t.} \quad x_{i\,p_{ik}} \geq x_{i\,p_{i\,k-1}} + t_{i\,p_{i\,k-1}}, \qquad \forall i \in \mathscr{J}, \forall k \in \mathscr{M} \backslash \{1\} \tag{19.13}$$
$$x_{ik} \geq x_{jk} + t_{jk} \vee x_{jk} \geq x_{ik} + t_{ik}, \qquad \forall k \in \mathscr{M}, \forall i, j > i \in \mathscr{J} \tag{19.14}$$

When comparing with CP1, this model has a different objective function and, consequently, the variable c and the Constraint 19.11 are no longer needed. This is possible because CP allows for the use of non-linear equations, such as Eq. 19.12.

19.3.1.3 CP3

Using the same sets, and indexes of the MIP Model presented:

Parameters

O_{ik} : (m, t) operation of job i in the kth machine, represented by a tuple of the machine in question (m) and the associated processing time (t)

Decision Variables

w_{ik}, sizeO_{ik} : t **interval** that represents the operation of job i on machine k, which must have the length (size) of the processing time t of operation O_{ik}

s_k **sequence** of the above-defined intervals (jobs) on machine k; built from the intervals for all i jobs and o machines such that $O_{io} : m = k$

CP3 Model

$$\min \quad \max_i \{ \text{endOf}(w_{iM}) \} \tag{19.15}$$

$$\text{s.t.} \quad \text{noOverlap}(s_k) \qquad \forall k \in \mathscr{M} \tag{19.16}$$

$$\text{endBeforeStart}(w_{ik}, w_{ik+1}) \qquad \forall i \in \mathscr{J}, \forall k \in \mathscr{M} \backslash \{M\} \tag{19.17}$$

This model is considerably different from the previous two (and, consequently, from the MIP Model) mainly because the decision variables were changed in order to enable the use of two global constraints (Constraints 19.16 and 19.17), which were designed specifically for the *JSSP* and are able to fulfil the Constraint Programming potential to its maximum in this type of problems. In fact, our variables are now *intervals*, representing the length of the job operations by its size (which is a parameter) and their positioning by its start time (which is the decision variable) and finish time, and *sequences of intervals*, which represent the job scheduling in each machine. In fact, although the second decision variable is somehow repeating what the first one already says, it is needed in order to build Constraint 19.16, which will bring significant computational advantages.

The objective function is similar to the one on CP2, and attempts to minimize the makespan, i.e. the last interval or operation to end. Constraint 19.16 uses the structure of the decision variable sequence s, and ensures that there is no overlap of jobs in every machine. It has the same function as Constraints 19.3 and 19.4 in the MIP Model. Constraint 19.17 ensures that a job only starts on a machine after it has finished on the previous. It has the same function as Constraint 19.2 in the MIP Model.

19.3.2 Computational Tests and Results

In order to understand the behaviour and performance of these CP models versus the MIP model, eleven instances were chosen from the eighty-two tested for the Mathematical Programming approach (Sect. 19.2.2). These instances were chosen based on their performance with the MIP Model. For each size of instance (number of jobs and number of machines), the hardest instance to solve with Mathematical Programming was chosen, so as to increase sample variability. The "difficulty" criterion was the solution time if the instance was solved within the 30 min time limit, or the optimality gap attained if the solver was stopped without proving optimality. It is also possible to observe that the instances chosen come from all four quartiles, when classified using the maximum needs-to-capacity ratio (Table 19.5, in the Appendix), so as to increase diversity.

The computational tests were run on the same computer and the models were also developed and run on IBM ILOG CPLEX Optimization Studio 12.6.1.0 using the CP Optimizer. The time limit set for the resolution of each instance was also 30 min. The eleven instances chosen and the summary of the results are presented on Table 19.2. Here, the values in bold are the best solutions found across models.

These results point to the initial hypothesis that Constraint Programming is most useful when it involves well-performing global constraints, which were able to capture and use the problem structure to propagate constraints and prune branches more

Table 19.2 Results of CP models, comparing with MIP model

	Objective function				Solution time (s)			
	MIP	CP1	CP2	CP3	MIP	CP1	CP2	CP3
`la10`	**958**	**958**	**958**	**958**	*1.800*	*1.800*	*1.800*	0
`la14`	**1.292**	**1.292**	**1.292**	**1.292**	*1.800*	*1.800*	*1.800*	0
`orb01`	**1.059**	1.096	1.070	**1.059**	1.013	*1.800*	*1.800*	28
`la23`	**1.032**	**1.032**	**1.032**	**1.032**	*1.800*	*1.800*	*1.800*	0
`la30`	**1.355**	**1.355**	**1.355**	**1.355**	*1.800*	*1.800*	*1.800*	2
`la37`	1.418	1.430	1.410	**1.397**	*1.800*	*1.800*	*1.800*	5
`la34`	1.749	1.721	1.740	**1.721**	*1.800*	*1.800*	*1.800*	1
`abz7`	702	691	691	**663**	*1.800*	*1.800*	*1.800*	*1.800*
`swv10`	2.024	1.915	1.961	**1.827**	*1.800*	*1.800*	*1.800*	*1.800*
`yn4`	1.054	1.044	1.061	**986**	*1.800*	*1.800*	*1.800*	*1.800*
`swv13`	4.307	3.829	3.948	**3.149**	*1.800*	*1.800*	*1.800*	*1.800*

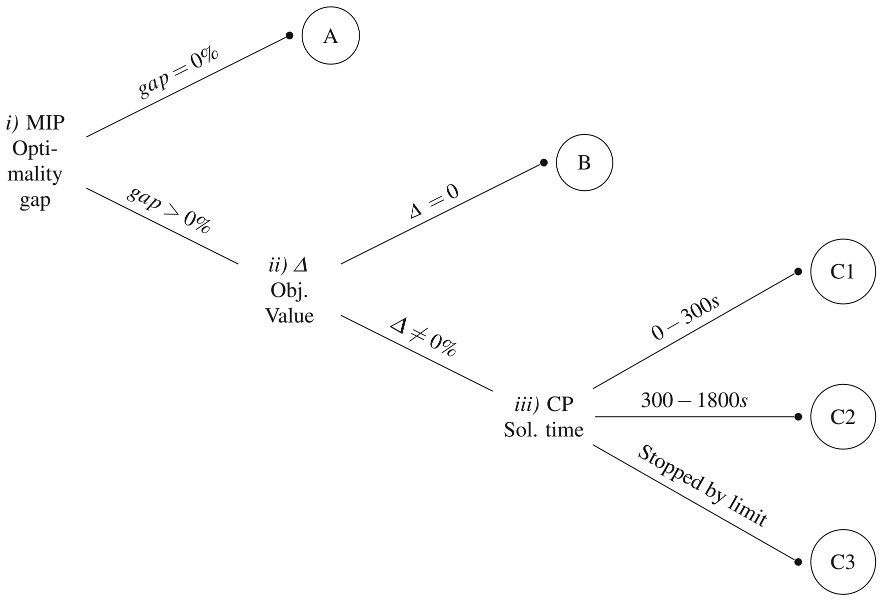

Fig. 19.5 Characteristics of the different types of instances

easily. In fact, when comparing CP1 or CP2 with the MIP model, one is able to conclude that these Constraint Programming models bring little or no advantage. As the instances get bigger, the combinations of the domains to test get exponentially large and hinder a good performance of the model. However, when the global constraints and a more appropriate representation are used (CP3), this complexity is more easily tackled and the results point to a significantly better performance of CP when compared with a MIP approach. Comparing the performance of CP1 and CP2 (which were very similar), it is also possible to conclude that reducing the number of constraints does not bring clear advantages, as is more usual in MIP programs.

In order to compare in more detail the Mathematical Programming and Constraint Programming approaches, the eighty-two instances were run with the full-performing CP model (CP3). The results are presented in the Appendix (Table 19.6[3]). From the results, the instances were categorized in six categories (A, B, C1, C2, C3), depending on three sequential criteria: *(i)* whether or not the MIP model was able to prove the optimality of the solution found (i.e. attaining 0% optimality gap), *(ii)* the difference on the objective function values found by both approaches, and *(iii)* the run time of the CP model. Figure 19.5 depicts the instance categorization and the respective characteristics.

Instances of type A are the instances that the MIP model solved within the time limit, i.e. the optimality of the solution was proven (with 0% gap). The instances where the MIP solution optimality was not proven were divided in two groups: the

[3]This table also repeats the MIP results in order to facilitate the comparison instance by instance.

Table 19.3 Summary of average results for CP model CP3, comparing with MIP model

Instance type	(#)	Avg size[5]	Avg MIP sol. time (s)	Avg CP sol. time (s)	Avg Δ obj. value (%)
A	(24)	87	109	6	0
B	(17)	154	*1.800*	0	0
C1	(18)	278	*1.800*	44	3
C2	(1)	225	*1.800*	313	−1
C3	(22)	341	*1.800*	*1.800*	−11

instances where the solution value found by the CP Optimizer (within the time limit, hence optimal) was equal to the one attained by the MIP model (type B) and the instances where the solution values were different[4] (type C). The instances of type C were also divided according to the solution time of the CP program. One should notice that the program was stopped by the time limit only for category C3; for C1 and C2, it stopped because it had explored all possible solutions. Table 19.3 presents the summary of the average results for each type of instance.

The average size[5] of the instances increases from A to C, yet there is no linear relation with size and the distribution of the instances of type C in C1, C2 and C3. In fact, C1 contains both smaller instances (15 jobs and 5 machines) and some of the biggest instances (50 jobs and 10 machines). At the same time, other instances of the same sizes are classified as C3. This is an indication that also in Constraint Programming size is not a clear indicator of complexity.

Instances of type A (the smallest) were solved to optimality by both approaches, and their optimality was proven also in both cases.[6] Nevertheless, the CP approach was considerably faster.

For instances of type B, since the CP Optimizer was able to explore all possible solutions under the time limit, it proved the optimality of the best solution found. As there is no difference between the solution value found by CP and MIP, it is possible to state that both approaches reached the optimum. However, although the MIP model was also able to reach the optimum for these instances, it was not able to prove so within the time limit. Considering that for these 17 instances the CP Optimizer took, in average, under a second to attain the optimal solution, this approach was able to almost instantaneously prove optimality whilst the MIP solver was not able to do so in 30 min. This conclusion does not contrast with the notion that Constraint Programming is a technique more prone to admissibility rather than optimality: in

[4]I.e., there existed a delta on the objective function value given by $\Delta = (\text{ObjValue}_{CP} - \text{ObjValue}_{MIP})/\text{ObjValue}_{MIP}$.

[5]Since the number of variables and constraints is different in the two models, size is herein considered as the number of jobs multiplied by the number of machines.

[6]MIP model: proven by optimality gap. CP Model: the only way for a CP program to prove optimality is by exploring all possible solutions; therefore, if the solver stops before the time limit, it means that optimality was proven.

fact, it was its power to efficiently check admissibility that allowed for the optimality to be proven in these cases.

As for instances of type C, it is important to notice that there are 18 instances categorized as C1, 22 as C3 yet only one as C2. Moreover, the instance categorized as C2 is close to the threshold of C1 (the run time is 313 s and the threshold is 300 s). In fact, either the CP Optimizer is swift to explore all possible solutions (under 5 min) or it reaches the time limit (30 min) without being able to do so. Moreover, as it as mentioned above, the size is not a good indicator of this behaviour: when analysing the biggest instances (`swv11-swv20`), it is possible to conclude that in half of the instances the optimal solution is found in under 3 s while in the other half the CP Optimizer is stopped after 30 min without being able to explore all possible solutions. Nevertheless, there are significant improvements on the objective function value for all sub-categories of C, especially on C3.

It is therefore interesting to study how swiftly the CP Optimizer finds its best solution in 30 min. As an example, Fig. 19.6 presents the value of the incumbent solutions found throughout the run time for instance `swv12`. The dotted line represents the best solution found by the MIP solver for this instance (with an optimality gap of 79%). It is possible to see that the steep improvement on the objective function value occurs in the beginning. In fact, approximately 80% of the improvement (vs. the MIP solution) occurs in 20 s, and 95% occurs in 2 min.

It is also interesting to analyse the swiftness to find a good solution for those instances in which both approaches were stopped by the 30 min time limit and where the CP presented only a slightly better performance in terms of the best objective function value found in this limit. Figure 19.7 shows the evolution of the incumbent value for both approaches for instance `yn1` (20 jobs, 20 machines) in which the improvement attained by the CP approach (vs. the MIP approach) was the smallest (1%).

As it is possible to see, the CP model was able to achieve a good solution before the MIP model. In fact, for the CP approach there was one major improvement in the

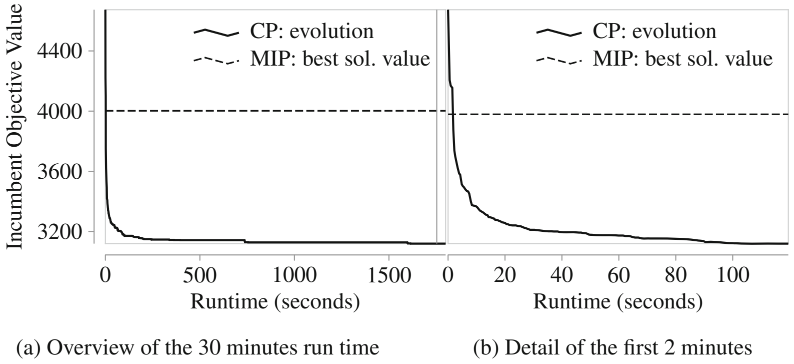

Fig. 19.6 Evolution of the incumbent objective function value for instance `swv12` throughout the 30 min run time with model CP3

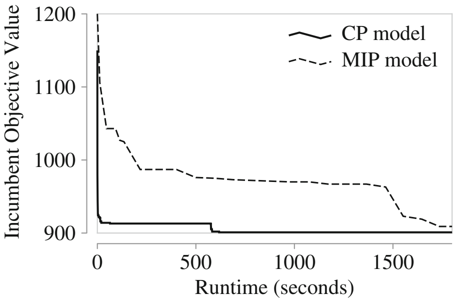

Fig. 19.7 Evolution of the incumbent objective function value for instance `yn1` throughout the 30 min run time with the MIP model and CP model (CP3)

first seconds and a smaller one after approximately 10 min. As for the MIP model, there were three major improvements: the first after a few seconds, the second at around 2–3 min, and the third in the last 10 min of the time horizon. This may lead to the conclusion that this full-performing CP model is quicker than the MIP model at finding good solutions even in the case where both approaches reach the same solution within the same time limit.

One final note to highlight the fact that the needs-to-capacity ratio was not a good predictor of complexity (or “difficulty to solve”) for the CP Optimizer. In fact, there was not a clear correlation between the CP solution time and the needs-to-capacity ratio quartile of the instances (Table 19.5, in the Appendix).

19.4 Conclusions

The first objective of this paper was to understand the impact of the instances’ size on the intrinsic complexity of the *JSSP* and on its consequent “difficulty to solve”. The problem was described and a MIP Model from the literature was presented and used to solve eighty-two instances. The results enabled a detailed analysis of the instances’ characteristics and responsiveness to the MIP model.

From the analysis of the performance of the Mathematical Programming approach, it was possible to conclude that an instance size has a significant impact on its easiness to solve, yet it is not the only significant factor. In fact, there was significant variability on solution time for same-size instances, and for bigger instances that were not solved to optimality there was not always visible a straightforward relationship between size and optimality gap achieved.

In this approach, the size of the problem is determined by two characteristics: the number of jobs and the number of machines. From the structure of the problem and its formulation, the number of jobs has a bigger impact on the number of variables and number of constraints, hence a bigger impact on the size. Moreover, the increase on the number of jobs seems to have a straightforward relationship with the increase of optimality gap achieved. As for the number of machines, it was possible to conclude that its impact is not direct. For a specific number of jobs, it seems that an increasing or more balanced number of machines makes the instances easier to solve despite being of a bigger size. An hypothesis proposed to explain this behaviour is related to the easiness of the solver to find admissible integer solutions in such a problem, hence being able to update the upper bound and the optimality gap more frequently.

In order to further understand the concept of *complexity*, a measure of the load applied to each machine throughout the scheduling horizon was designed. The *needs-to-capacity ratio* developed shows a significant correlation with the optimality gap attained by the MIP solver, thus being an adequate predictor of the "difficulty to solve" of an instance, for a Mathematical Programming approach.

The measure proposed, as well as the insights gained regarding the factors that induce complexity, can be useful e.g. when tackling an industrial scheduling application of the *JSSP*. Analysing the number of jobs and machines and other characteristics of the actual problem, it will be possible to understand whether the MIP model can easily solve the problem to optimality thus eliminating the need for more complex solution approaches or whether these are justifiably needed.

The second objective of this paper was to compare the impact of using Constraint Programming instead of Mathematical Programming to tackle the complexity of the *JSSP*. It was concluded that the main power of CP when tackling these hard, complex, combinatorial problems comes from the tailored-made global constraints that are able to swiftly improve propagation, find admissible solutions and thus significantly decrease the solution time. In fact, the CP models that used the same variables as the MIP model and similar constraints – which benefited only from part of the CP modelling flexibility by the use of logic and non-linear constraints – performed as well as, or worse than, the MIP model for the instances tested. The CP model that used a different representation of the variables and solid global constraints, however, was able to get significant improvements either in solution time and quality of the solutions found.

It was also possible to conclude that for some instances, although the MIP solver presented high optimality gap values at the end of the time limit, it was indeed able to reach the optimum. Nevertheless, it was not able to prove so. This conclusion is built on the fact that, within seconds, the (well-performing) CP model was able to test all possible admissible solutions and state that said solution was the best possible. Moreover, it was possible to conclude that the instance size has a significantly smaller impact in Constraint Programming than in Mathematical Programming. In fact, half of the largest instances tested were solved to optimality in a few seconds while the run of the remaining half was stopped by the 30 min time limit. There were also indicators that the full-performing CP approach may be swifter than the MIP approach at finding good solutions, even when it is not able to prove optimality in the specified time limit and it is not able to attain major improvements on the best solution value found.

Since the best performing CP model, which presents all the above-mentioned advantages, is commercially available, these conclusions may have a significant impact on practical application of optimization methods. In fact, it is pointed out that there are Constraint Programming models available for this type of problem that may be a better alternative to the "traditional" MIP approach for some problems.

Future work will be focused on comparing other characteristics of both approaches, such as the number of feasible solutions found in a specific time frame, which might be useful when using hybrid solution methods such as matheuristics to solve this problem. Additionally, it would be interesting to study the reasons why the needs-to-capacity ratio does not appear to be a good predictor of “difficulty to solve” for CP models.

Acknowledgements The first author was supported by grant SFRH/BD/103362/2014 from FCT - Fundação para a Ciência e Tecnologia (Portuguese Foundation for Science and Technology). This work was also partially financed by the ERDF - European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project “POCI-01-0145-FEDER-006961”, and by National Funds through the FCT - Fundação para a Ciência e Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013.

Appendix

See Tables 19.4, 19.5 and 19.6.

Table 19.4 Size and characteristics of the instances run and corresponding results with the MIP model

Instance name	Jobs	Machines	Obj. function value	Solution time (s)	Optimality gap (%)
ft06	6	6	55	0	0
abz5	10	10	1.234	19	0
abz6	10	10	943	4	0
ft10	10	10	930	122	0
la01	10	5	666	7	0
la02	10	5	655	6	0
la03	10	5	597	3	0
la04	10	5	590	2	0
la05	10	5	593	45	0
la16	10	10	945	7	0
la17	10	10	784	13	0
la18	10	10	848	3	0
la19	10	10	842	6	0
la20	10	10	902	4	0
orb01	10	10	1.059	1.013	0
orb02	10	10	888	9	0

(continued)

Table 19.4 (continued)

Instance name	Jobs	Machines	Obj. function value	Solution time (s)	Optimality gap (%)
orb03	10	10	1.005	590	0
orb04	10	10	1.005	38	0
orb05	10	10	887	17	0
orb06	10	10	1.010	596	0
orb07	10	10	397	14	0
orb08	10	10	899	36	0
orb09	10	10	934	56	0
orb10	10	10	944	14	0
la06	15	5	926	1.800	33.91
la07	15	5	890	1.800	26.07
la08	15	5	863	1.800	26.66
la09	15	5	951	1.800	25.00
la10	15	5	958	1.800	34.66
la21	15	10	1.071	1.800	15.07
la22	15	10	932	1.800	7.41
la23	15	10	1.032	1.800	23.42
la24	15	10	940	1.800	4.68
la25	15	10	979	1.800	6.54
la36	15	15	1.281	1.800	10.54
la37	15	15	1.418	1.800	15.31
la38	15	15	1.207	1.800	11.76
la39	15	15	1.240	1.800	8.55
la40	15	15	1.245	1.800	14.78
abz7	20	15	702	1.800	31.45
abz8	20	15	715	1.800	28.95
abz9	20	15	721	1.800	27.04
ft20	20	5	1.198	1.800	48.51
la11	20	5	1.222	1.800	50.31
la12	20	5	1.039	1.800	40.69
la13	20	5	1.150	1.800	41.91
la14	20	5	1.292	1.800	52.46
la15	20	5	1.207	1.800	49.59
la26	20	10	1.218	1.800	26.93
la27	20	10	1.288	1.800	33.39
la28	20	10	1.224	1.800	31.74
la29	20	10	1.220	1.800	26.42
la30	20	10	1.355	1.800	34.64

(continued)

Table 19.4 (continued)

Instance name	Jobs	Machines	Obj. function value	Solution time (s)	Optimality gap (%)
swv01	20	10	1.563	1.800	46.45
swv02	20	10	1.579	1.800	45.40
swv03	20	10	1.618	1.800	47.42
swv04	20	10	1.625	1.800	48.51
swv05	20	10	1.633	1.800	45.81
swv06	20	15	1.868	1.800	40.26
swv07	20	15	1.736	1.800	38.28
swv08	20	15	2.046	1.800	41.88
swv09	20	15	1.907	1.800	41.85
swv10	20	15	2.024	1.800	45.65
yn1	20	20	913	1.800	17.50
yn2	20	20	954	1.800	21.91
yn3	20	20	945	1.800	20.33
yn4	20	20	1.054	1.800	27.42
la31	30	10	1.784	1.800	55.77
la32	30	10	1.850	1.800	52.68
la33	30	10	1.719	1.800	54.53
la34	30	10	1.749	1.800	57.72
la35	30	10	1.888	1.800	55.58
swv11	50	10	3.767	1.800	77.84
swv12	50	10	4.002	1.800	78.68
swv13	50	10	4.307	1.800	80.37
swv14	50	10	4.081	1.800	79.66
swv15	50	10	3.722	1.800	78.02
swv16	50	10	2.952	1.800	73.41
swv17	50	10	2.915	1.800	72.10
swv18	50	10	2.950	1.800	74.16
0swv19	50	10	3.107	1.800	74.67
swv20	50	10	3.009	1.800	74.46
abz5 – abz9 in Adams et al. [2]			ft06, ft10, ft20 in Fisher and Thompson [6]		
la01 – la40 in Lawrence [7]			orb01 – orb10 in Applegate and Cook [3]		
swv01 – swv20 in Storer et al. [11]			yn1 – yn4 in Yamada and Nakano [14]		

Table 19.5 Needs-to-capacity ratio calculation for each instance

Needs-to-capacity ratio						
Instance name	Optimality gap (%)	Maximum value	Quartile (of max)	Std. deviation	# partitions	Partition length
ft06	0	1.17	1st	0.30	27	2
abz5	0	2.00	3rd	0.55	9	100
abz6	0	1.20	2nd	0.34	18	40
ft10	0	1.00	1st	0.11	159	4
la01	0	2.60	3rd	0.63	17	24
la02	0	2.40	3rd	0.64	15	24
la03	0	2.00	3rd	0.43	31	14
la04	0	2.00	3rd	0.35	42	10
la05	0	2.20	3rd	0.39	39	10
la16	0	1.00	1st	0.19	51	14
la17	0	1.00	1st	0.17	67	10
la18	0	1.00	1st	0.17	65	10
la19	0	1.00	1st	0.19	56	10
la20	0	1.20	2nd	0.20	60	12
orb01	0	1.10	1st	0.18	68	10
orb02	0	1.00	1st	0.16	50	12
orb03	0	1.00	1st	0.16	66	10
orb04	0	1.10	1st	0.17	76	10
orb05	0	1.10	1st	0.18	55	10
orb06	0	1.00	1st	0.19	61	12
orb07	0	1.20	2nd	0.25	28	10
orb08	0	1.20	2nd	0.18	66	10
orb09	0	1.00	1st	0.17	69	10
orb10	0	1.10	1st	0.17	78	10
la06	34	3.00	3rd	0.63	29	14
la07	26	3.60	4th	0.77	25	16
la08	27	3.40	4th	0.59	38	10
la09	25	3.40	4th	0.64	26	14
la10	35	3.00	3rd	0.51	44	10
la21	15	1.70	2nd	0.28	48	14
la22	7	1.50	2nd	0.23	59	10
la23	23	1.60	2nd	0.23	73	10
la24	5	1.70	2nd	0.24	75	10
la25	7	1.70	2nd	0.25	77	10
la36	11	1.00	1st	0.17	69	14
la37	15	1.00	1st	0.15	95	10
la38	12	1.13	1st	0.15	98	10

(continued)

Table 19.5 (continued)

Needs-to-capacity ratio						
Instance name	Optimality gap (%)	Maximum value	Quartile (of max)	Std. deviation	# partitions	Partition length
la39	9	1.00	1st	0.15	95	10
la40	15	1.00	1st	0.15	91	10
abz7	31	1.73	2nd	0.41	19	22
abz8	29	2.27	3rd	0.53	21	22
abz9	27	1.80	2nd	0.54	23	22
ft20	49	4.20	4th	0.44	111	4
la11	50	4.60	4th	0.90	31	14
la12	41	4.40	4th	0.76	41	10
la13	42	4.00	4th	0.71	42	10
la14	52	4.40	4th	0.72	45	10
la15	50	4.40	4th	0.80	32	12
la26	27	2.20	3rd	0.36	50	14
la27	33	2.00	3rd	0.27	64	10
la28	32	2.10	3rd	0.31	81	10
la29	26	2.20	3rd	0.31	76	10
la30	35	2.00	3rd	0.30	77	10
swv01	46	2.00	3rd	0.14	356	2
swv02	45	2.00	3rd	0.15	330	2
swv03	47	2.20	3rd	0.15	334	2
swv04	49	2.10	3rd	0.15	351	2
swv05	46	2.10	3rd	0.14	390	2
swv06	40	1.40	2nd	0.09	483	2
swv07	38	1.33	2nd	0.09	449	2
swv08	42	1.40	2nd	0.09	527	2
swv09	42	1.40	2nd	0.09	476	2
swv10	46	1.40	2nd	0.13	231	4
yn1	17	1.30	2nd	0.23	33	20
yn2	22	1.40	2nd	0.31	37	20
yn3	20	1.15	1st	0.28	35	20
yn4	27	1.20	2nd	0.27	37	20
la31	56	3.30	4th	0.46	73	10
la32	53	3.10	3rd	0.43	79	10
la33	55	3.30	4th	0.46	75	10
la34	58	3.00	3rd	0.43	60	10
la35	56	3.10	3rd	0.45	68	10
swv11	78	5.00	4th	0.30	397	2
swv12	79	5.30	4th	0.31	390	2

(continued)

Table 19.5 (continued)

Needs-to-capacity ratio						
Instance name	Optimality gap (%)	Maximum value	Quartile (of max)	Std. deviation	# partitions	Partition length
swv13	80	5.20	4th	0.31	382	2
swv14	80	5.30	4th	0.31	382	2
swv15	78	5.00	4th	0.32	339	2
swv16	73	5.10	4th	0.31	372	2
swv17	72	5.00	4th	0.33	308	2
swv18	74	5.00	4th	0.32	331	2
swv19	75	5.00	4th	0.31	339	2
swv20	74	5.10	4th	0.32	333	2

Table 19.6 Instances run (with *J* jobs and *M* machines) with MIP model and CP model CP3

Instance	MIP optimality gap (%)	MIP obj. function value	CP3 Obj. function value	MIP sol. time (s)	CP3 sol. time (s)
ft06	0.0	55	55	0	0
abz5	0.0	1.234	1.234	19	7
abz6	0.0	943	943	4	2
ft10	0.0	930	930	122	12
la01	0.0	666	666	7	0
la02	0.0	655	655	6	0
la03	0.0	597	597	3	0
la04	0.0	590	590	2	1
la05	0.0	593	593	45	0
la16	0.0	945	945	7	2
la17	0.0	784	784	13	2
la18	0.0	848	848	3	2
la19	0.0	842	842	6	10
la20	0.0	902	902	4	3
orb01	0.0	1.059	1.059	1.013	28
orb02	0.0	888	888	9	7
orb03	0.0	1.005	1.005	590	19
orb04	0.0	1.005	1.005	38	10
orb05	0.0	887	887	17	7
orb06	0.0	1.010	1.010	596	13
orb07	0.0	397	397	14	4
orb08	0.0	899	899	36	5
orb09	0.0	934	934	56	3
orb10	0.0	944	944	14	2

(continued)

Table 19.6 (continued)

Instance	MIP optimality gap (%)	MIP obj. function value	CP3 Obj. function value	MIP sol. time (s)	CP3 sol. time (s)
la06	33.9	926	926	1.800	0
la07	26.1	890	890	1.800	0
la08	26.7	863	863	1.800	0
la09	25.0	951	951	1.800	0
la10	34.7	958	958	1.800	0
la21	15.1	1.071	1.046	1.800	76
la22	7.4	932	927	1.800	9
la23	23.4	1.032	1.032	1.800	0
la24	4.7	940	935	1.800	35
la25	6.5	979	977	1.800	52
la36	10.5	1.281	1.268	1.800	13
la37	15.3	1.418	1.397	1.800	5
la38	11.8	1.207	1.196	1.800	313
la39	8.5	1.240	1.233	1.800	18
la40	14.8	1.245	1.222	1.800	82
abz7	31.5	702	663	1.800	1.800
abz8	29.0	715	680	1.800	1.800
abz9	27.0	721	686	1.800	1.800
ft20	48.5	1.198	1.165	1.800	1
la11	50.3	1.222	1.222	1.800	0
la12	40.7	1.039	1.039	1.800	0
la13	41.9	1.150	1.150	1.800	0
la14	52.5	1.292	1.292	1.800	0
la15	49.6	1.207	1.207	1.800	0
la26	26.9	1.218	1.218	1.800	1
la27	33.4	1.288	1.235	1.800	254
la28	31.7	1.224	1.216	1.800	31
la29	26.4	1.220	1.153	1.800	1.800
la30	34.6	1.355	1.355	1.800	2
swv01	46.4	1.563	1.413	1.800	1.800
swv02	45.4	1.579	1.475	1.800	203
swv03	47.4	1.618	1.406	1.800	1.800
swv04	48.5	1.625	1.488	1.800	1.800
swv05	45.8	1.633	1.438	1.800	1.800
swv06	40.3	1.868	1.710	1.800	1.800
swv07	38.3	1.736	1.667	1.800	1.800
swv08	41.9	2.046	1.810	1.800	1.800
swv09	41.8	1.907	1.698	1.800	1.800

(continued)

Table 19.6 (continued)

Instance	MIP optimality gap (%)	MIP obj. function value	CP3 Obj. function value	MIP sol. time (s)	CP3 sol. time (s)
swv10	45.7	2.024	1.827	1.800	1.800
yn1	17.5	913	901	1.800	1.800
yn2	21.9	954	910	1.800	1.800
yn3	20.3	945	910	1.800	1.800
yn4	27.4	1.054	986	1.800	1.800
la31	55.8	1.784	1.784	1.800	1
la32	52.7	1.850	1.850	1.800	0
la33	54.5	1.719	1.719	1.800	0
la34	57.7	1.749	1.721	1.800	1
la35	55.6	1.888	1.888	1.800	1
swv11	77.8	3.767	3.021	1.800	1.800
swv12	78.7	4.002	3.119	1.800	1.800
swv13	80.4	4.307	3.149	1.800	1.800
swv14	79.7	4.081	2.970	1.800	1.800
swv15	78.0	3.722	2.960	1.800	1.800
swv16	73.4	2.952	2.924	1.800	0
swv17	72.1	2.915	2.794	1.800	0
swv18	74.2	2.950	2.852	1.800	0
swv19	74.7	3.107	2.843	1.800	2
swv20	74.5	3.009	2.823	1.800	0

References

1. 4COutreachProgram. CSP tutorial (2005), http://4c.ucc.ie/web/outreach/tutorial.html
2. J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling. Manag. Sci. **34**, 391–401 (1988)
3. D. Applegate, W. Cook, A computational study of the job-shop scheduling instance. ORSA J. Comput. **3**, 149–156 (1991)
4. K.R. APT, *Priciples of Constraint Programming* (Cambridge University Press, Cambridge, 2003)
5. E. Balas, Disjunctive programming. Ann. Discret. Math. **5**, 3–51 (1979)
6. H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in *Industrial Scheduling*, ed. by J.F. Muth, G.L. Thompson (Prentice-Hall, Englewood Cliffs, 1963), pp. 225–251
7. S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement), in *Graduate School of Industrial Administration*, Carnegie-Mellon University, Pittsburgh, Pennsylvania (1984)
8. J.K. Lenstra, Computational complexity of discrete optimization problems. Ann. Discret. Math. **4**, 121–140 (1979)
9. J. Poppenborg, S. Knust, J. Hertzber, Online scheduling of flexible job-shops with blocking and transportation. Eur. J. Ind. Eng. **6**, 497–518 (2012)

10. F. Rossi, P. van Beek, T. Walsh (eds.), *Handbook of Constraint Programming* (Elsevier, Amsterdam, 2006)
11. R.H. Storer, S.D. Wu, R. Vaccari, New search spaces for sequencing instances with application to job shop scheduling. Manag. Sci. **38**, 1495–1509 (1992)
12. K. Thörnblad, A.B. Strömberg, M. Patriksson, T. Almgren, Scheduling optimisation of a real flexible job shop including fixture availability and preventive maintenance. Eur. J. Ind. Eng. **9**, 126–145 (2015)
13. W.J. van Hoeve, Introduction to constraint programming, in *ACP Summer School on Theory and Practice of Constraint Programming*, September 24–28, 2012, Wrocław, Poland (2012)
14. T. Yamada, R. Nakano, A genetic algorithm applicable to large-scale job-shop instances, in *PPSN'2 Proceedings of the 2nd International Workshop on Parallel Problem Solving from Nature*, ed. by R. Manner, B. Manderick (1992), pp. 281–290