A Study of Constraint Programming Heuristics for the Car-Sequencing Problem

Mohamed Siala[a,b], Emmanuel Hebrard[a,c], Marie-José Huguet[a,b]

[a] *CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France*
[b] *Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France*
[c] *Univ de Toulouse, LAAS, F-31400 Toulouse, France*

Abstract

In the car-sequencing problem, a number of cars has to be sequenced on an assembly line respecting several constraints. This problem was addressed by both Operations Research (OR) and Constraint Programming (CP) communities, either as a decision problem or as an optimization problem. In this paper, we consider the decision variant of the car sequencing problem and we propose a systematic way to classify heuristics for solving it. This classification is based on a set of four criteria, and we consider all relevant combinations for these criteria. Some combinations correspond to common heuristics used in the past, whereas many others are novel. Not surprisingly, our empirical evaluation confirms earlier findings that specific heuristics are very important for efficiently solving the car-sequencing problem (see for instance [17]), in fact, often as important or more than the propagation method. Moreover, through a criteria analysis, we are able to get several new insights into what makes a good heuristic for this problem. In particular, we show that the criterion used to select the most constrained option is critical, and the best choice is fairly reliably the "load" of an option. Similarly, branching on the type of vehicle is more efficient than branching on the use of an option. Overall, we can therefore indicate with a relatively high confidence which is the most robust strategy, or at least outline a small set of potentially best strategies.

Last, following a remark in [14] stating that the notion of *slack* used in heuristics induces a pruning rule, we propose an algorithm for this method and experimentally evaluate it, showing that, although computationally cheap and easy to implement, this is in practice a very efficient way to solve car-sequencing benchmarks.

Keywords: Car-sequencing, Heuristics, Tree Search Methods, Constraint Programming

1. Introduction

The car-sequencing problem comes from the automotive industry and has a long history in constraint programming. In this problem, a number of cars have to be sequenced on an assembly line. Each class of cars requires a set of options. However, the working station handling a given option can only mount it on a fraction of the cars passing on the line. Each option j is then associated with a fractional number p_j/q_j standing for its capacity (no more than p_j cars with this option j should occur in any sub-sequence of length q_j).

The car-sequencing was first introduced as a Constraint Satisfaction Problem (CSP) modeled using CHIP [2], and later an optimization variant of this problem was used as benchmark for the Roadef Challenge in 2005 (see [18] for a survey of the methods used during the Challenge). The decision version of the problem, that we shall consider in this paper, asks if a given set of cars can be sequenced on the assembly line in such a way that the line never has to slow down. This problem is NP-hard in the strong sense [8].

Our contributions are divided in two major points. First, we provide an empirical study regarding car-sequencing heuristics. To the best of our knowledge, this is the most complete heuristics study for this problem. For the experiments, all heuristics are included in a complete chronological backtracking method. Moreover, we combine these heuristics to several known filtering algorithms to evaluate the trade-off between search and propagation.

We show the interest of some new heuristics in our experiments as well as the impact of the branching strategy comparatively to filtering algorithms.

This empirical study is based on a new classification of heuristics for the car sequencing. This classification is based on a set of four criteria: branching variables, exploration directions, selection of branching variables and aggregation functions for this selection. In particular, we show that the way of selecting the most constrained option is critical, and the best choice is fairly reliably the "load" of an option, that is the ratio between its demand and the capacity of the corresponding machine. Similarly, branching on the class of vehicle is more efficient than branching on the use of an option.

Overall, even though results vary greatly from instance to instance, we can therefore indicate with a relatively high confidence which is the most robust strategy, or at least outline a small set of potentially best strategies.

Second, we propose a new filtering rule, using the notion of *slack* introduced by Puget and Régin [14]. This filtering can be used only when variables are explored in the lexicographic order. However, as shown by Smith [17], this is a good strategy for this problem. Moreover, this filtering rule greatly improves the performance of our model whilst being almost free computationally.

In this paper, we are mainly interested in heuristics for solving the decision variant of the car-sequencing problem, although we shall draw a link with propagation in Section 5. However, such study provides insights on resolving optimization car-sequencing variants. In fact, there are two standard types of objective functions for the car-sequencing problem. One somehow counts the number of constraint violations. This is the objective function used for instance in the Roadef challenge 2005 [18]. The other considers the delay incurred because of the violations of the capacity constraints. It was proposed by Hindi and Ploszajski [7] and latter on used in [10, 11, 16]. The latter objective is a simple and effective way of setting up the assembly line so that it can run smoothly. In this case, the number of extra empty slots gives an exact measure of the delay incurred. Solving a sequence of the satisfaction problem where a number of extra empty slots are added, for instance through binary search, is a natural way of optimizing this objective function. Furthermore, methods based on constraint programming are still competitive and branching heuristics are significantly contributing to this. For instance, they are relevant within a Large Neighborhood Search (LNS) approach such as the one proposed in Perron and Shaw [10, 11]. In this context, one needs to solve a small CSP standing for a small part of the problem. Solving this sub-problem is expected to be fast, hence we rely on branching and selection heuristics to speed up this task. The heuristics are therefore likely to make a significant difference.

The rest of the article is organized as follows. In Section 2, we describe the car-sequencing problem and discuss the related constraint satisfaction models. In Section 3, we propose and classify a number of new and existing heuristics, which we empirically evaluate and analyze in Section 4. Then, in Section 5, we introduce a simple filtering rule to propagate capacity constraints coupled with cardinality. Finally, we show in Section 6 the interest of the proposed filtering rule and provide a comparison with state-of-the-art propagators for this problem as well as other approaches.

2. The Car-sequencing problem

2.1. Problem description

In the car-sequencing problem, n vehicles have to be produced on an assembly line. There are k classes of vehicles and m types of options. Each class $c \in \{1, \dots, k\}$ is associated with a demand D_c^{class}, that is, the number of occurrences of this class on the assembly line, and a set of options $\mathcal{O}_c \subseteq \{1, \dots, m\}$. Each option is handled by a working station able to process only a fraction of the vehicles passing on the line. The capacity of an option j is defined by two integers p_j and q_j, such that no subsequence of size q_j may contain more than p_j vehicles requiring option j.

A solution of the problem is then a sequence of cars satisfying both demand and capacity constraints.

For convenience, we shall also define, for each option j, the corresponding set of classes of vehicles requiring this option $\mathcal{C}_j = \{c \mid j \in \mathcal{O}_c\}$, and the option's demand $D_j = \sum_{c \in \mathcal{C}_j} D_c^{class}$.

Example 2.1. *Consider the simple case of 7 slots (i.e. $n = 7$) with 3 classes $\{c_1, c_2, c_3\}$ and 4 options such that:*

- $\mathcal{O}_{c_1} = \{1, 3\}$, $\mathcal{O}_{c_2} = \{2, 3\}$, $\mathcal{O}_{c_3} = \{4\}$.
- $D_{c_1}^{class} = 2$, $D_{c_2}^{class} = 3$, $D_{c_3}^{class} = 2$
- p_i/q_i *(lexicographically):* $1/2$; $2/3$; $3/5$; $3/6$.

From above, we obtain:

- $\mathcal{C}_1 = \{1\}$, $\mathcal{C}_2 = \{2\}$, $\mathcal{C}_3 = \{1, 2\}$ *and* $\mathcal{C}_4 = \{3\}$
- $D_1 = 2$, $D_2 = 3$, $D_3 = 5$ *and* $D_4 = 2$

The sequence $[c_2,\ c_2,\ c_1,\ c_3,\ c_3,\ c_2,\ c_1]$ *is a possible solution for this instance.*

2.2. Constraint Programming

Many combinatorial problems can be easily formulated using a constraint-based model. In this approach, the problems are usually solved through complete search tree methods (we refer the reader to [9, 15] for a comprehensive introduction). The efficiency of this approach depends on several factors: the heuristics used to select the most promising variables and values; how the search tree is expanded and how the constraints are propagated during the search.

The resolution methods that we shall study in this paper are based on depth-first search with chronological backtracking and propagation. In this context, we need to decide at each step on which variable should we branch, and which value it should be assigned to. Both decisions are based on heuristics. Such heuristics can be defined declaratively as a variable and a value ordering, respectively. We call their combination a *branching strategy.* We explore then the search space in a depth-first manner following the variable and value ordering prescribed by the branching strategy. The efficiency of this kind of methods clearly relies on the number of expanded nodes during search. Moreover, the shape and the size of the search tree is highly dependent on the variable and value orderings. The basic principles explaining the efficiency of variable and value orderings are well known. On the one hand, we want to choose the value that has the best chances to lead to a solution, if the current sub-tree has one. This has been called *promise* [3] or *succeed-first* principle. On the other hand, we want to choose the variable that will lead to fail as soon as possible, when the current sub-tree is inconsistent.

The other crucial factor when exploring the search space is propagation. A *propagator* (called also *filtering algorithm*) associated to a constraint is a procedure that removes, for some variables, some values (or set of values) that cannot satisfy the constraint w.r.t the current assignment. The set of propagators are repeatedly called whenever a domain change occurs during search until no more reduction is possible. A good trade-off between heuristics and filtering algorithms is mandatory for better performance.

2.3. Constraint-based models for the car-sequencing

We use a standard *CP* modeling with two sets of variables. The first set corresponds to n integer variables $\{x_1, \dots, x_n\}$ (called class variables) taking values in $\{1, \dots, k\}$ and standing for the class of vehicles in each slot of the assembly line. The second set of variables corresponds to nm Boolean variables $\{y_1^1, \dots, y_n^m\}$ (called option variables), where y_i^j stands for whether the vehicle in the i^{th} slot requires option j.

There are three sets of constraints.

1. *Demand constraints*: for each class $c \in \{1..k\}$, $|\{i \mid x_i = c\}| = D_c^{class}$. This constraint is usually enforced with a Global Cardinality Constraint (Gcc) [13, 12]
2. *Capacity constraints*: for each option $j \in \{1..m\}$, no subsequence of size q_j involves more than p_j options of this type: $\sum_{l=i}^{i+q_j-1} y_l^j \leq p_j$, $\forall i \in \{1, \ldots, n - q_j + 1\}$. In order to factor out as much as possible the propagation aspect from the study, we used several models in order to diversify the data set. More precisely, we shall consider four models, differentiated by how capacity constraints are modeled and thus propagated. For each option j, these constraints can be expressed in one of the following alternatives:
   (a) a naive decomposition using sum constraints (denoted by Sum).
   (b) a model using the *Global Sequencing Constraint* (Gsc) proposed by Régin and Puget [14].
   (c) a model using the AtMostSeqCard constraint recently proposed in [16].
   (d) a model combining the Gsc and the AtMostSeqCard propagators (since the pruning obtained by both approaches is incomparable).
3. *Channeling*: option and class variables are channeled through simple constraints: $y_i^j = 1 \Leftrightarrow j \in \mathcal{C}_{x_i}, \forall j \in \{1, ..., m\}, \forall i \in \{1, ..., n\}$. Each such constraint is implemented using a set of simple binary constraints $x_i = c \Rightarrow y_i^j = 1,\ \forall j \in \mathcal{O}_c$ and $x_i = c \Rightarrow y_i^j = 0,\ \forall j \in \{1, \ldots, m\} \setminus \mathcal{O}_c$.

2.4. Related work

Regarding the search strategy, two main principles are known to be important for the car-sequencing problem. First, the sequence of variables to branch should follow the assembly line itself. Indeed, the structure in chain of capacity constraints makes it difficult to achieve any inference far away from a modified variable in the sequence [17]. Second, one should assign the most constrained class or option first. This has been perceived as a fail-first strategy, hence surprising since succeed-first strategies should be better for selecting the next branch to follow. However, as pointed out in [17], since the solutions to this problem are permutations of a multiset of values, choosing the most constrained one when it is still possible actually yields the least constrained sub-problem. In fact, in this sense, it is indeed a succeed-first strategy.

In [17], a lexicographical exploration of the integer variables x_1 up to x_n, standing for classes of vehicles, was advocated as an interesting search strategy. Three parameters were considered for choosing the most constrained class: the number of options per class (denoted as *max option*), the tightness of each option (ie. the capacity constraint q/p) and the usage of each option (i.e. usage rate $\frac{d.q/p}{n}$).

In [14], the authors proposed to branch on option variables y_i^j, exploring the sequence consistently with their position on the assembly line, however starting from the middle towards the extremities. Indeed variables at bothe ends are subject to fewer capacity constraints than variables within the sequence. Moreover, they introduced for the first time the notion of *slack* for selecting the most constrained option.

In [6], several heuristics were compared for solving an optimization variant of this problem. These heuristics are based on the usage rate previously defined for selecting the next variables to assign in the sequence. They consider two ways for aggregating these values (using the maximum value, lexicographically, or a simple sum) when branching on class variables. Two possibilities of using the usage rate were compared : static and dynamic (i.e. updated at each node). Note that the static values of usage rate, load or slack are all equivalent. Their experiments showed essentially the interest of dynamic heuristics comparatively to static ones. The same observation is made in [1] where a dynamic load was used with a class variable branching and a simple summation to aggregate the values.

3. Heuristics Classification

3.1. Classification criteria

We propose to classify the heuristics related to this problem according to four criteria:

- The type of *branching* decisions: that is whether we branch on classes or options.
- The order in which we *explore* the variables along the assembly line: one can start from the left of the sequence and progress to the right, or start from the middle of the assembly line widen to the sides.

- The measure used to *select* the most constrained options.
- The function used to *aggregate* the evaluation of the different options in order to choose the next class of vehicles.

Notice that among the many combinations of these four criteria, some correspond to existing heuristics, however some are novel. For each criterion, there are several alternatives, we present each of them in the following.

3.1.1. Branching

The branching is either the assignment of a class to a slot, that is, branching on class variables x_i, or the assignment of an option to a slot, that is, branching on option variables y_i^j. The former was used in [17], while the latter was proposed in [14]. Notice that when branching on option variables, we always set it to the value 1, which amounts at forcing to the corresponding option to be represented in that slot. We therefore consider these two cases denoted respectively *class* and *opt*.

3.1.2. Exploration

Heuristics that do not follow the sequence of variables along the assembly line generally have poor performances. Indeed, the structure in chain of capacity constraints makes it difficult to achieve any inference far away from a modified variable in the sequence [17]. In the literature, two exploration orders were considered: either lexicographical order on class variables or from the middle to the sides of the sequence. For each type of variables, we therefore consider these two exploration cases denoted respectively *lex* and *mid*.

3.1.3. Selection

The best heuristics are those selecting first the most constrained option or class. Observe that since each class is defined by a set of options, then all we care about is the hardness of the options. We therefore consider the following indicators proposed in the literature to select the most constrained option:

- The capacity q_j/p_j: The greater the ratio q_j/p_j, the more constrained is the option. In fact, a greater ratio q_j/p_j has more impact on neighboring slots as it is shown in example 3.1.

Example 3.1. *Let o_1 and o_2 be two options with $1/3$ and $2/3$ as capacity ratios (p_j/q_j) respectively. Consider now a sequence of 5 slots in which we have to choose between o_1 and o_2 on the third position. The two parts of the following figure show the impact of each option. In fact, by choosing o_1, all neighboring slots can no longer contain this option because of the at most $1/3$ constraints.*

		y_i^1					y_i^2		
0	***0***	*1*	***0***	***0***	.	.	*1*	.	.

- The residual demand (d_j): This value is equal to the total demand (of a given option) minus the number of cars containing this option already allocated ($d_j = (D_j - \sum_{i=1}^{n} min(\mathcal{D}(y_i^j)))$. Clearly, a greater demand makes it more difficult to fit the cars requiring this option on the line.

- The *load* δ_j: This parameter combines the residual demand with the capacity ratio: $\delta_j = d_j \frac{q_j}{p_j}$. In fact, this value is tied to the number of slots required to mount d_j times the option j. A greater value of the load is therefore more constrained.

- The slack σ_j: Let n_j be the number of slots available for option j. The slack of an option j is $\sigma_j = n_j - \delta_j$. Since we want higher values to indicate more constrained options, we use in fact $n - \sigma_j$.

- The usage rate ρ_j: This value is defined as the load divided by the number of remaining slots: $\rho_j = \delta_j / n_j$. It therefore represents how much of the remaining space will be occupied by vehicles requiring this option.

Based on these indicators, we consider five methods to evaluate the options. Each method returns an indicative value on how constrained is an option. In other words, the option maximizing the given parameter will be preferred in the next decision. In the following, we denote the above selection criteria respectively by q/p, d, δ, $n - \sigma$ and ρ. Moreover, we consider the constant function 1 as another possible selection criterion. This is proposed so that our classification also includes the *max option* heuristic [17] where each class is evaluated simply by its number of options. Note also that we consider only dynamic evaluation with the four criteria : demand, load, usage rate and slack since they outperform the static versions [1, 6].

3.1.4. Aggregation

In the case of *class* branching, since classes are defined as a set of options, the decision is most often done by summing up the "scores" of the options for each class. However, there are many ways to aggregate these values. We therefore propose to add the method used for the aggregation as a fourth criterion.

Let $f : \{1, \dots, m\} \mapsto \mathbb{R}$ be a scoring function. We denote $f(\mathcal{O}_c)$ the tuple formed by the sorted scores of class c's options, i.e., $f(\mathcal{O}_c) = \langle f(j_1), \dots, f(j_{|\mathcal{O}_c|})\rangle$ such that $\{j_1, \dots, j_{|\mathcal{O}_c|}\} = \mathcal{O}_c$ and $f(j_l) \geq f(j_{l+1})\ \forall l \in [1, \dots, |\mathcal{O}_c| - 1]$. We shall consider the following ordering relations between classes:

- Sum of the elements ($\leq_{\sum}$): $c_1 \leq_{\sum} c_2$ iff $\sum_{v \in f(\mathcal{O}_{c_1})} v \leq \sum_{v \in f(\mathcal{O}_{c_2})} v$.
- Euclidean norm ($\leq_{Euc}$): $c_1 \leq_{Euc} c_2$ iff $\sum_{v \in f(\mathcal{O}_{c_1})} v^2 \leq \sum_{v \in f(\mathcal{O}_{c_2})} v^2$.
- Lex order ($\leq_{lex}$): $c_1 \leq_{lex} c_2$ iff $f(\mathcal{O}_{c_2})$ comes lexicographically after $f(\mathcal{O}_{c_1})$.

Example 3.2. *To illustrate aggregation functions, we consider example 2.1 and suppose that one branch on classes. In table 1, we give the different values of each selection parameter for all options.*

Table 1: Values of the selection criteria for each option

Selection parameter \ Options	1	2	3	4
1	1	1	1	1
q/p	2	1.5	1.66	2
d	2	3	5	2
δ	4	4.5	8.33	4
$n - \sigma$	4	4.5	8.33	4
ρ	0.57	0.64	1.19	0.57

In order to emphasize the impact of aggregation functions, we propose to study the different scores for each class using the q/p parameter. Recall that each class is defined by a set of options, we obtain in table 2 the corresponding values for each class.

In table 3, we report the order of preferences given by the different aggregations. The class having the higher score will be selected first and so on.

Table 2: Classes' scores using the parameter q/p

Options \ Classes	c_1	c_2	c_3
1	2	-	-
2	-	1.5	-
3	1.66	1.66	-
4	-	-	2

Table 3: Scores & Heuristic decisions

Agg.	Scores			Heuristic preferences
	c_1	c_2	c_3	
$\leq_{\sum}$	3.66	3.16	2	[c_1, c_2, c_3]
$\leq_{Euc}$	6.75	5	4	[c_1, c_2, c_3]
$\leq_{lex}$	[2, 1.66, -, -]	[1.66, 1.5,-,-]	[2,-,-,-]	[c_1, c_3, c_2]

Although we treat a simple case, one can observe that decisions can be influenced by aggregation functions. The behavior of $\leq_{lex}$ here was different from the others. It prefers c_3 rather than c_2 for the second variable selection.

3.2. Heuristics structure

In the rest of the article, we denote the set of heuristics as follows: $\langle\{class, opt\}, \{lex, mid\}, \{1, q/p, d, \delta, n-\sigma, \rho\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$. Observe, however, that not all combinations make sense. For instance, the aggregation function does not matter when branching on options. Therefore, using the new classification, we obtain 42 possible heuristics:

- $\langle\{class\}, \{lex, mid\}, \{q/p, d, \delta, n-\sigma, \rho\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$: The 30 heuristics that branches on *class* variables with the two exploration strategies $\{lex, mid\}$, the five selection parameters $\{q/p, d, \delta, n-\sigma, \rho\}$ and the 3 aggregation techniques $\{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}$.

- $\langle\{opt\}, \{lex, mid\}, \{q/p, d, \delta, n-\sigma, \rho\}, \emptyset\rangle$: 10 heuristics branching on option variables with the two exploration possibilities $\{lex, mid\}$ and the five selection parameters $\{q/p, d, \delta, n-\sigma, \rho\}$.

- $\langle\{class\}, \{lex, mid\}, \{1\}, \{\leq_{\sum}\}\rangle$: The two possible heuristics related to the particular case of *max option*.

Among the many combinations defined by this structure, there are several existing heuristics as well as new ones. In the literature, only few heuristics have been studied.

First, the *max option* heuristic proposed in [17] branches on *class* variables lexicographically (*lex*) and the most constrained class is then selected using the sum ($\leq_{\sum}$) aggregation. It therefore corresponds to $\langle class, lex, 1, \leq_{\sum} \rangle$.

Second, in [6], the authors proposed to use the usage rage with *class* branching, lexicographical exploration (*lex*) and $\leq_{\sum}$, $\leq_{lex}$ for aggregation. They correspond to $\langle class, lex, \delta, \{\leq_{\sum}, \leq_{lex}\}\rangle$. Similarly, the authors of [1] proposed a *class* branching using $\leq_{\sum}$ for aggregation in a lexicographical exploration (*lex*), however, using the load δ and the capacity q/p for selection (i.e. $\langle class, lex, \{\delta, q/p\}, \leq_{\sum}\rangle$). Finally, the heuristic proposed in [14] is based on *option* branching, exploring the sequence from the middle to the sides using the slack as a selection criteria. This heuristic corresponds to $\langle opt, mid, n - \sigma, \emptyset\rangle$.

To the best of our knowledge, all other heuristics are new and there is no comparative study for evaluating the impact of each classification criterion.

4. Evaluating the new structure

In this section, we evaluate the impact of the proposed classification criteria for the heuristics. We slightly perform randomization as a simple mechanism to deal with the Heavy tail phenomena [5]. In particular, with a low probability (2% for classes and 5% for options[1]), the second best choice (provided by the heuristic) is taken.

All the experiments were run on Intel Xeon CPUs 2.67GHz under Linux and are available via `http://homepages.laas.fr/msiala/car-sequencing`. For each instance, we launched 5 randomized runs per heuristic with a 20 minutes time cut-off. All models are implemented using Ilog-Solver.

We use benchmarks available from the CSPLib [4] divided into three groups. The first group of the CSPLib contains 70 satisfiable instances having 200 cars, 5 options and from 18 to 30 classes, it is denoted by *set1*. The second group of the CSPLib corresponds to instances with 100 cars, 5 options and from 19 to 26 classes. In this group there are 4 satisfiable instances , denoted

[1]Those values were arbitrarily chosen. The impact of branching on an option variable being lower, a higher probability was necessary.

by *set2* and 5 unsatisfiable instances denoted by *set3*. The third group of the CSPLib contains 30 larger instances (ranging from 200 to 400 vehicles, 5 options and from 19 to 26 classes). *Set4* concerns the 7 instances from this group that are known to be satisfiable. At the top of each table, we mention, for each data set, the total number of instances with an indication on their feasibility (i.e. satisfiable: S and unsatisfiable U). The status of the 23 remaining instances are still unknown. They are often treated in an optimization context, hence are not considered in our experimentations.

The global set of the 42 previously defined heuristics $\langle\{class, opt\}, \{lex, mid\}, \{1, q/p, d, \delta, n-\sigma, \rho\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$ is combined with four propagators: $\langle$ Sum, Gsc, AtMostSeqCard, Gsc$\oplus$AtMostSeqCard$\rangle$ leading to 168 different configurations. The latter is applied to each set of instances (i.e. $70 + 4 + 5 + 7$ instances) with 5 randomized runs. The total average CPU time for these experiments is around 244 days.

We say that a run (related to an instance and a given configuration) is successful if either a solution was found or unsatisfiability was proven. For each set of instances, we report the percentage of successful runs (*%sol*) [2], the CPU time (*time*) in seconds both averaged over all successful runs and number of instances.

Experimental results are divided in thee parts. We first compare the many combinations of heuristic factors by giving the results for each one. Then, we study the proposed classification by evaluating each factor separately. Finally, we provide a comparison related to the efficiency and confidence of each factor

4.1. Impact of each heuristic

In this paragraph, we report the results of each heuristic separately on each set of instances averaged over the four propagators.

The set of heuristics corresponds to all possible combinations of parameters given by: $\langle\{class, opt\}, \{lex, mid\}, \{1, q/p, d, \delta, n-\sigma, \rho\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$ leading to the 42 heuristics presented in Section 3.

Table 4 shows the results of our experiments. For each heuristic, we indicate in column (Ref) whether it is already known (with the corresponding reference) or not (with '-'). Recall that, in these experiments, we consider

[2]Since *set*3 contains only unsatisfiable instances, then %*sol* corresponds to the percentage of instances for which the heuristics prove the unsatisfiability

only dynamic evaluation with the four criteria : demand, load, usage rate and slack. For each set of instances, we report the percentage of successful runs ($\%sol$) and the CPU time ($time$). The last two columns summarize the results over all set of instances. The column ($\%tot$) gives the total percentage of solved instances and the column ($\%dev$) gives the deviation in percent of a given heuristic to the heuristic solving the maximum number of instances. Bold values give the best heuristics w.r.t $\%sol$.

For the easiest set (set1), 16 heuristics solve all instances in less than a second. Among them, 3 are known heuristics whereas 13 correspond to new combinations. It should be noted that all these configurations use a *class* branching and a load-based selection (i.e. $\rho, \delta, n-\sigma$). Interestingly, twisting one parameter from a heuristic can have a dramatic effect. For instance, the heuristic $\langle opt, lex, n-\sigma, \emptyset\rangle$ resolves only 32, 71% of this set whereas changing only the branching criterion to *class* (i.e. $\langle class, lex, n-\sigma, \{\leq_{lex}, \leq_{\sum}, \leq_{Euc}\}\rangle$) leads to a complete resolution (i.e. 100%).

For set2 and set3, the heuristic $\langle opt, lex, \rho, \emptyset\rangle$ gives the best results with 75% in $33.3s$ for set2 and 25% in $211.3s$ for set3. Also, the heuristics $\langle class, mid, d, \{\leq_{lex}, \leq_{Euc}\}\rangle$ has the same number of successful runs compared to $\langle opt, lex, \rho, \emptyset\rangle$ but with higher resolution time. All of these heuristics correspond to new configurations.

Finally, for set4, the best heuristics resolve 25.71% in approximately $3s$ and correspond to the configurations $\langle class, lex, \{\delta, \rho, n-\sigma\}, \leq_{lex}\rangle$. Another heuristic $\langle class, lex, d, \leq_{lex}\rangle$ obtains the same percentage but with higher computation time ($55.3s$).

Overall, the heuristic that has the best results across all data sets and therefore seems to be the more robust is $\langle class, lex, \delta, \leq_{lex}\rangle$ with 86.8% of solved instances (according to the column 'Total'). More generally, heuristics using load-based selection (i.e. δ, $n-\sigma$ and ρ) and class branching obtain better results than the other configurations.

4.2. Criteria analysis

In this part, we aim to evaluate the relative impact of each classification criterion. For each criterion and each data set, we divide all the runs into as many sets as the number of possible values for this criterion. Then, we average the results within each set. For instance, *exploration* can be done either lexicographically (*lex*), or from the middle to the sides (*mid*). We will thus report two sets of statistics, one for *lex* and one for *mid*. Each average

Table 4: Comparison of heuristics averaged over propagation rules

Heuristics				Ref.	Instances								Total	
					set 1 $(70, S)$		set 2 $(4, S)$		set 3 $(5, U)$		set 4 $(7, S)$			
Sel.	Br.	Expl.	Aggr.		%sol	time	%sol	time	%sol	time	%sol	time	$\%tot$	$\%dev$
ρ	*class*	*lex*	$\leq_{lex}$	[6]	**100.00**	0.6	52.50	59.1	0.00	-	**25.71**	2.9	85.93	1.00
			$\leq_{\sum}$	[6]	**100.00**	0.6	48.75	0.2	0.00	-	10.71	84.4	84.53	2.61
			$\leq_{Euc}$	-	**100.00**	0.6	30.00	0.2	0.00	-	12.85	156.3	83.84	3.42
		mid	$\leq_{lex}$	-	99.92	0.5	53.75	163.5	0.00	-	16.42	50.0	85.17	1.88
			$\leq_{\sum}$	-	**100.00**	**0.5**	51.25	236.6	0.00	-	18.57	5.4	85.29	1.74
			$\leq_{Euc}$	-	**100.00**	**0.5**	51.25	249.3	0.00	-	17.14	30.2	85.17	1.88
	opt	*lex*	-	-	87.00	1.9	**75.00**	**33.3**	**25.00**	**211.3**	5.71	533.4	76.22	12.19
		mid	-	-	87.64	2.9	31.25	0.4	23.00	233.6	14.28	171.1	75.29	13.26
$n-\sigma$	*class*	*lex*	$\leq_{lex}$	-	**100.00**	0.6	52.50	59.2	0.00	-	**25.71**	**2.8**	85.93	1.00
			$\leq_{\sum}$	-	**100.00**	0.6	48.75	0.2	0.00	-	10.71	78.6	84.53	2.61
			$\leq_{Euc}$	-	**100.00**	0.6	48.75	0.1	0.00	-	10.71	79.4	84.53	2.61
		mid	$\leq_{lex}$	-	**100.00**	0.6	53.75	169.7	0.00	-	18.57	33.1	85.41	1.61
			$\leq_{\sum}$	-	**100.00**	**0.5**	51.25	236.9	0.00	-	22.14	29.0	85.58	1.41
			$\leq_{Euc}$	-	99.92	0.5	51.25	236.3	0.00	-	22.14	28.8	85.52	1.48
	opt	*lex*	-	-	32.71	21.7	43.75	236.8	13.00	190.7	0.00	-	29.42	66.11
		mid	-	[14]	38.14	13.0	26.25	33.7	18.00	260.8	0.00	-	33.31	61.62
δ	*class*	*lex*	$\leq_{lex}$	-	**100.00**	0.6	71.25	42.4	0.00	-	**25.71**	3.0	**86.80**	0.00
			$\leq_{\sum}$	[1]	**100.00**	0.6	48.75	0.3	0.00	-	10.71	100.2	84.53	2.61
			$\leq_{Euc}$	-	**100.00**	0.6	48.75	0.3	0.00	-	10.71	87.3	84.53	2.61
		mid	$\leq_{lex}$	-	**100.00**	**0.5**	37.50	38.2	0.00	-	15.00	51.5	84.36	2.81
			$\leq_{\sum}$	-	**100.00**	**0.5**	68.75	167.9	0.00	-	20.71	42.8	86.28	0.60
			$\leq_{Euc}$	-	**100.00**	**0.5**	68.75	166.5	0.00	-	20.00	16.2	86.22	0.67
	opt	*lex*	-	-	98.57	1.2	36.25	111.7	0.00	-	22.85	5.8	83.78	3.48
		mid	-	-	98.92	3.7	43.75	3.8	0.00	-	21.42	88.8	84.29	2.89
q/p	*class*	*lex*	$\leq_{lex}$	-	82.85	7.8	0.00	-	0.00	-	0.00	-	67.44	22.31
			$\leq_{\sum}$	[1]	83.35	10.1	18.75	0.1	0.00	-	0.00	-	68.72	20.84
			$\leq_{Euc}$	-	83.42	11.3	18.75	0.09	0.00	-	0.00	-	68.77	20.77
		mid	$\leq_{lex}$	-	84.71	7.9	18.75	95.7	0.00	-	0.00	-	69.82	19.56
			$\leq_{\sum}$	-	85.35	7.7	18.75	100.9	0.00	-	0.00	-	70.34	18.96
			$\leq_{Euc}$	-	84.64	7.5	18.75	96.0	0.00	-	0.00	-	69.77	19.63
	opt	*lex*	-	-	65.71	73.3	0.00	-	0.00	-	0.00	-	53.48	38.38
		mid	-	-	70.71	29.8	12.50	606.4	0.00	-	0.00	-	58.14	33.02
d	*class*	*lex*	$\leq_{lex}$	-	90.92	1.2	37.50	47.4	0.00	-	**25.71**	55.3	77.84	10.32
			$\leq_{\sum}$	-	95.07	1.9	41.25	48.5	0.00	-	17.14	21.5	80.70	7.03
			$\leq_{Euc}$	-	94.50	0.7	43.75	106.5	0.00	-	23.57	40.2	80.87	6.83
		mid	$\leq_{lex}$	-	90.64	1.9	**75.00**	83.4	0.00	-	24.28	5.3	79.24	8.71
			$\leq_{\sum}$	-	94.71	0.6	67.50	68.9	0.00	-	13.57	53.9	81.33	6.30
			$\leq_{Euc}$	-	94.57	0.6	**75.00**	83.2	0.00	-	15.71	50.7	81.74	5.83
	opt	*lex*	-	-	73.78	2.9	56.25	79.5	0.00	-	0.71	282.0	62.73	27.73
		mid	-	-	77.28	13.7	43.75	5.2	0.00	-	7.85	16.5	65.58	24.45
1	*class*	*lex*	$\leq_{\sum}$	[17, 1]	86.92	13.2	18.75	0.1	0.00	-	0.00	-	71.62	17.49
		mid	$\leq_{\sum}$	-	89.92	8.3	63.75	20.3	0.00	-	0.00	-	76.16	12.26

will be over one run per possible completion of the heuristic (21), filtering algorithms (4), randomized runs (5), and instances in the data set.

The following tables (5, 6, 7 and 8) are split in two parts. In the upper one, we report the results for each set and each possible criterion w.r.t the criterion being used averaged over all other criteria. The lower part shows the best results obtained for any possible combination of the other criteria. In these tables, we report the percentage of successful runs (*%sol*), the CPU time (*time*) in seconds both averaged over all successful runs, instances and heuristic criteria. Bold values indicate best results in terms of successful runs (*%sol*). Moreover, in the upper tables, the last column (*%tot*) gives the percentage of solved instances over the all sets.

4.2.1. Branching strategy

Here we compare the two branching strategies: *class* and *opt*. We tested all the possible combinations of heuristics for each strategy. However, as the constant selection parameter 1 is not defined for *opt* variables, we do not consider its heuristics in the following table.

When branching on *opt* variables, we have defined 10 heuristics (since aggregation functions are omitted): $\langle opt, \{lex\ , mid\}, \{q/p, d, \delta, n-\sigma, \rho\}, \emptyset\rangle$, that is 200 tests for each instance. To have consistent comparison with *class* branching, we separate its results by aggregation functions. That is $\langle class, \{lex\ , mid\}, \{q/p, d, \delta, n-\sigma, \rho\}, \leq_{lex}\rangle$, $\langle class, \{lex\ , mid\}, \{q/p, d, \delta, n-\sigma, \rho\}, \leq_{Euc}\rangle$ and $\langle class, \{lex\ , mid\}, \{q/p, d, \delta, n-\sigma, \rho\}, \leq_{\sum}\rangle$.

Table 5: Evaluation of the branching variants

Av. Bran.	set1 $(70, S)$			set2 $(4, S)$			set3 $(5, U)$			set4 $(7, S)$			Global
(×200)	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%tot
opt	73.0	102023.9	14.1	36.8	287139.5	82.0	**7.9**	53275.4	225.6	7.2	207502.8	107.9	62.2
class, $\leq_{lex}$	94.9	26120.0	2.0	45.2	481410.8	84.9	0.0	-	-	**17.7**	98707.8	22.5	80.7
class, $\leq_{\sum}$	**95.8**	27209.1	2.1	**46.3**	327601.5	95.7	0.0	-	-	12.4	156300.3	44.6	**81.1**
class, $\leq_{Euc}$	95.7	27563.3	2.1	45.5	463196.6	107.9	0.0	-	-	13.2	107599.7	52.9	81.0
Best Bran.													
opt	**100.0**	98577.4	10.3	75.0	7251.3	0.5	**40.0**	46211.8	261.8	25.7	629016.8	130.7	
class, $\leq_{lex}$	**100.0**	184.7	0.0	**100.0**	730687.4	89.5	0.0	-	-	**28.5**	29632.6	58.5	
class, $\leq_{\sum}$	**100.0**	184.2	0.0	95.0	904739.2	96.3	0.0	-	-	25.7	34705.3	54.8	
class, $\leq_{Euc}$	**100.0**	184.4	0.0	**100.0**	211830.5	128.8	0.0	-	-	**28.5**	47435.1	75.4	

The upper part of Table 5 shows that branching on classes is usually better than branching on options. However, the latter is more efficient on proving infeasibility (i.e. line *opt* on set3). The most efficient branching averaged over the other factors is with the aggregation $\leq_{\sum}$ but the two other aggregation ($\leq_{lex}$ or $\leq_{Euc}$) are closed. This result is confirmed by the lower part of the table.

4.2.2. Exploration

To evaluate the exploration parameters, we consider for each $\omega \in \{lex, mid\}$ the following heuristics:

- $\langle class, \omega, \{q/p, d, \delta, n - \sigma, \rho\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$.
- $\langle opt, \omega, \{q/p, d, \delta, n - \sigma, \rho\}, \emptyset\rangle$.
- $\langle class, \omega, \{1\}, \{\leq_{\sum}\}\rangle$.

These three sets cover all possible combinations of heuristics leading to 420 tests for each parameter $\omega \in \{lex, mid\}$ and each instance. The results are shown in Table 6.

Table 6: Evaluation of the exploration variants

Av. Expl.	set1 $(70, S)$			set2 $(4, S)$			set3 $(5, U)$			set4 $(7, S)$			Global
(×420)	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%tot
lex	89.2	50617.6	5.6	40.0	259229.0	46.6	1.8	52295.1	204.3	11.3	120652.6	54.2	75.5
mid	**90.3**	42167.0	4.1	**46.7**	479360.9	126.5	**1.9**	54184.0	245.5	**12.7**	139829.4	42.8	**76.8**
Best Expl.													
lex	**100.0**	184.8	0.0	**100.0**	730687.4	89.5	**40.0**	46211.8	261.9	**28.5**	29632.6	58.5	
mid	**100.0**	183.5	0.0	**100.0**	213028.8	129.1	36.0	63984.8	307.6	**28.5**	1357.4	9.2	

In the first part of table 6, we can see that exploring the sequence from the middle then widening to the sides is in average slightly but consistently beneficial. Recall that the rationale for starting in the middle is that variables in the extremities are subject to fewer capacity constraints.

However, in the second part of table 6, we can see that in terms of successful runs, exploring the sequence using the lexicographical order leads to better results for proving unsatisfiability. This could be explained by the fact that when starting in the middle of the sequence, we effectively split the problem into essentially disjoint subproblems (there is actually a weak link through demand constraints).

Overall, the exploration parameter does not seem to be as critical as the branching parameter.

4.2.3. Selection

Here, we evaluate the selection criterion for choosing the most-constrained option. In this case, there are two possible sets of heuristics for each parameter $\omega \in \{q/p, d, \delta, n - \sigma, \rho\}$:

- $\langle class, \{lex, mid\}, \omega, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$
- $\langle opt, \{lex, mid\}, \omega, \emptyset\rangle$

That is 8 heuristics for each ω combined with the 4 propagators and the 5 runs. We therefore have 160 tests for each instance (reported in table 7).

The special case of *max option* is presented separately at the end of Table 7 because the number of tested heuristics is different. In this case, there is only 2 heuristics $\langle class, 1, \{lex, mid\}, \{\leq_{\sum}\}\rangle$, that is 40 tests for each instance.

Table 7: Evaluation of the selection variants

Av. Selec.	set1 $(70, S)$			set2 $(4, S)$			set3 $(5, U)$			set4 $(7, S)$			Global
(×160)	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%tot
ρ	96.8	1628.8	1.0	49.2	480035.3	99.9	**6.0**	49922.8	222.0	15.1	136850.7	81.7	82.6
$n - \sigma$	83.8	5773.4	2.3	47.0	699885.1	126.9	3.8	58466.5	231.4	13.7	103897.2	33.3	71.7
δ	**99.6**	3292.6	1.0	52.9	254264.1	74.8	0.0	-	-	**18.3**	98161.0	41.5	**85.1**
q/p	80.0	195896.5	17.7	13.2	135511.2	123.0	0.0	-	-	0.0	-	-	65.8
d	88.9	25988.2	2.7	**55.0**	254347.0	68.8	0.0	-	-	16.0	185381.6	36.8	76.2
1 (×40)	88.4	130722.2	10.7	41.2	28165.2	15.8	0.0	-	-	0.0	-	-	73.8
Best Selec.													
ρ	**100.0**	184.8	0.0	75.0	7251.3	0.5	**40.0**	46211.8	261.9	25.7	4843.0	0.4	
$n - \sigma$	**100.0**	184.8	0.0	75.0	1009607.4	124.1	32.0	75445.9	351.0	25.7	4843.0	0.4	
δ	**100.0**	184.8	0.1	**100.0**	730687.4	89.5	0.0	-	-	25.7	4843.0	0.4	
q/p	98.8	7208.4	3.4	25.0	68.2	0.1	0.0	-	-	0.0	-	-	
d	**100.0**	178.7	1.2	**100.0**	213028.8	129.1	0.0	-	-	**28.5**	29632.6	58.5	
1	99.7	58773.0	9.9	85.0	51740.9	36.9	0.0	-	-	0.0	-	-	

The upper part of Table 7 shows that using the *load* solves more instances in average over the all sets and for satisfiable sets (set1, set2 and set4) only. Surprisingly, the *load* gives better results than *slack* and *usage rate*, despite the fact that both *slack* and *usage rate* are defined using the *load* and the number of available slots in the variable's sequence. However the *usage rate* criteria seems to work better both in average and for the best results for unsatisfiable instances. Moreover, in the second part of the table, one can note that the *demand* obtains good results.

This can be explained by the manner in which the benchmarks were generated. In fact, these instances, especially the hardest ones, are built in such way that they have a usage rate close to 1 [4]. Since the number of available slots is initially identical for all options, they also have the same (low) slack and the same (high) load. Therefore the heuristics based on these criteria (ie. *load*, *slack* and *usage rate*) cannot effectively discriminate values at the root of the search tree. However, recall that the load is defined as the product of the demand and the capacity. These two factors do not contribute equally, and therefore will favor different sets of options. In other words, one of them is bound to take a better decision, whilst the other is bound to take a worse one. We believe that this bias in the generation of the benchmarks explains the surprisingly good results of the demand (d) as well as the bad results of the capacity q/p along with the *load*, the *slack* and the *usage rate*.

4.2.4. Aggregation

Aggregation functions are only used with *class* branching. For each parameter $\omega \in \{\leq_{lex} \leq_{\sum} \leq_{Euc}\}$, we have the 10 following heuristics combined with the propagators and the random runs (i.e. 200 tests for each ω and each instance):

- $\langle class, \{lex, mid\}, \{q/p, d, \delta, n - \sigma, \rho\}, \omega\rangle$

The constant parameter for selection 1 is not considered in these experiments since it is only defined with the $\leq_{\sum}$ aggregation. The results are given in Table 8.

Table 8: Evaluation of the aggregation variants

Av. Agg.	set1 $(70, S)$			set2 $(4, S)$			set3 $(5, U)$			set4 $(7, S)$			Global
(×200)	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%tot
$\leq_{lex}$	94.9	26120.0	2.0	45.2	481410.8	84.9	0.0	-	-	**17.7**	98707.8	22.5	80.7
$\leq_{\sum}$	**95.8**	27209.1	2.1	**46.3**	327601.5	95.7	0.0	-	-	12.4	156300.3	44.6	**81.1**
$\leq_{Euc}$	95.7	27563.3	2.1	45.5	463196.6	107.9	0.0	-	-	13.2	107599.7	52.9	81.0
Best Agg.													
$\leq_{lex}$	**100.0**	184.7	0.0	**100.0**	730687.4	89.5	0.0	-	-	**28.5**	29632.6	58.5	
$\leq_{\sum}$	**100.0**	184.2	0.0	95.0	904739.2	96.3	0.0	-	-	25.7	34705.3	54.8	
$\leq_{Euc}$	**100.0**	184.4	0.0	**100.0**	211830.5	128.8	0.0	-	-	**28.5**	47435.1	75.4	

As we can see in the first part of this table, the three aggregation functions provide in average similar results except for the hardest instances (set4) where

$\leq_{lex}$ solved more instances. Considering all instances, $\leq_{\sum}$ solves the large number of problems. No solution were found for unsatisfiable instances as in our case, only *opt* branching can solve these instances (i.e. which by default do not use any aggregation function). However, regarding the best results in the second part of the table, when using $\leq_{lex}$ and $\leq_{Euc}$, one can obtain better performances in terms of resolved instances.

4.3. Criteria Analysis Conclusions

We have previously evaluated the average best choice of each criterion (in terms of solved instances). However, this choice is not the best on each set of instances. Instead, we can determine the best choice for each data set, called the "perfect" choice. The *Confidence* of the average best choice can then be defined by the ratio between the average best choice and the perfect choice. Similarly, we can consider the "worst" choice for each data set, and subsequently, define the *Significance* of a given factor using the ratio between the worst and the perfect choice as $1 - worst/perfect$.

Table 9: *Confidence* and *significance* for each factor

	Confidence	*Significance*
Branching	0.989	0.247
Selection	0.995	0.231
Exploration	1.000	0.017
Aggregation	0.995	0.015

In Table 9, we give the values of *Confidence* and *Significance* for each factor (branching, selection, exploration and aggregation)This table shows that there is high confidence for each selected average best choice (between 0.989 and 1.0): that is, exploration from middle to sides using a class branching, load selection, and a sum aggregation. When considering the significance of each criterion, one can observe that only two of them (branching and selection) have a valuable impact. For the two other criteria (i.e. exploration and aggregation), there is little impact on the results when changing the parameters.

Therefore, the most robust heuristics will be those branching on classes variables and selecting options using the load criterion, that is $\langle class, \{lex, mid\}, \delta, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$.

5. Slack-based Filtering Rule

When analyzing the heuristics, we have seen that selecting the options using the load, the slack, or the usage rate is beneficial. In this section, we shall see that one can go one step further, and use the same idea to prune the search tree at a very cheap computational cost.

5.1. Slack and Pruning the Search Tree.

In [14], it is observed that if the slack (σ_j) of an option j is negative, then the problem is unsatisfiable. Indeed, the load (δ_j) tends to represent the number of required slots to mount all the occurrences of an option. Since the slack is the difference between the available number of slots and the load, a negative value suggests infeasibility since we need more slots than are available. However, one has to be careful about boundaries issues since the capacity constraints are truncated at the extremities of the assembly line. For instance, consider an option j with $p_j = 1$, $q_j = 3$ and $d_j = 2$. The slack is negative as soon as there are less then six slots remaining ($n_j < 6$), however a line with only four slots is sufficient if we put the two classes requiring this option on both ends of the line. In other words, the load is an accurate measure of how many slots are needed for a given option, however only for large values of demand and length of the assembly line.

In order to take into account this issue of boundaries, we propose the following alternative definition for the load.

$$\delta'_j = q_j(\lceil d_j/p_j \rceil - 1) + \begin{cases} p_j & \text{if } d_j \bmod p_j = 0 \\ d_j \bmod p_j & \text{otherwise} \end{cases}$$

Notice however that the formula above is true only if the unassigned slots are contiguous in the assembly line. Therefore, in the following we assume that we explore the assembly line from left to right, and we describe a simple rule to filter the domain of a sequence of Boolean variables $y_1, \dots, y_n$ subject to capacity constraints with a fixed cardinality. (i.e. $\sum_{l=1}^{q} y_{i+l} \leq p \ \forall i \in [1, \dots, n-q+1]$ and $\sum_{i=1}^{n} y_i = d$)

Proposition 5.1. *For each option j, δ'_j is a lower bound on the number of required slots.*

Proof. Consider a sequence of δ'_j unassigned Boolean variables subject to a capacity constraint p_j/q_j and a demand d_j. Now assign the first p_j variables

Figure 1: Assignment of an option with capacity 3/5.

y^j	$_0$ **1**	$_1$ **1**	$_2$ 1	$_3$ 0	$_4$ 0	$_5$ **1**	$_6$ **1**	$_7$ 1	$_8$ 0	$_9$ 0	$_{10}$ **1**	$_{11}$ **1**	$_{12}$ 1	$_{13}$ 0	$_{14}$ 0	$_{15}$ **1**	$_{16}$ **1**
	3					2		6									

to 1, then the $q_j - p_j$ next variables to 0 and repeat this $(\lceil d_j/p_j \rceil - 1)$ times. Then fill the remaining variables with the value 1. The sequence built in this way is of length δ'_j and cardinality d_j. Moreover, every subsequence of length q_j has exactly p_j times the value 1, therefore, it is not possible to obtain the same cardinality in a shorter sequence, hence δ'_j is a lower bound. $\square$

5.2. Filtering the Domains

We suppose now that all variables up to a rank $i - 1$ are ground. To make the notation lighter we rename the sequence of unassigned variables $y_i, \ldots, y_n$ to: $y_0, \ldots, y_{n-i}$.

When the real load δ' is greater than the residual number of slots $n-i+1$, then we should fail since δ' is a lower bound on the number of required slots. When $\delta' = n - i + 1$, we can filter out some values. Moreover, we can prune inconsistent values in the domains of the option variables when the load is equal to the remaining number of slots. We illustrate this situation in Example 5.1

Example 5.1. *Figure 1 shows a possible assignment for a sequence* $y^j_0, \ldots, y^j_{16}$, *with capacity* $3/5$ *and demand* 11. *Consider the two slots indexed* 5 *and* 6, *corresponding to the variables* y^j_5 *and* y^j_6. *On the left, there are* 5 *slots, hence we can fit at most* 3 *vehicles with the option* j, *indeed fitting* 4 *vehicles requires* $6 = 5(\lceil 4/3 \rceil - 1) + 4 \bmod 3$ *slots. Similarly, on the right, one cannot fit more than* 6 *vehicles with option* j *since fitting* 7 *vehicles would require* 11 *slots. Therefore, since the total demand is* 11, *we can conclude that* $11 - 6 - 3 = 2$ *vehicles with option* j *must fit in the slots* 5 *and* 6. *In other words, both* y^j_5 *and* y^j_6 *must be equal to* 1.

Now we formally define a pruning rule that can detect all such forced assignments (e.g., it detects all bold faced 1's in Figure 1).

Theorem 5.1. *The following filtering rule is correct:*

If $\delta' = n - i + 1$, then if $d \bmod p = 0$, we impose $y_i = 1$ for all i such that $i \bmod q < p$. Otherwise (i.e. $d \bmod p \neq 0$), we impose $y_i = 1$ for all i such that $i \bmod q < (d \bmod p)$.

Proof. Suppose that ($d \bmod p \neq 0$). Then there exists two integers k and r such that $d = k.p + r$. Notice that in this case, we have $\delta' = q.k + r$. Consider a subsequence $y_a, \ldots, y_b$ such that $a \bmod q = 0$ and $b = a + r - 1$, i.e., such that the rule above applies. Then there exist two integers α and β such that $a = \alpha \cdot q$ and $n - i - b = \beta \cdot q$ (since $n - i + 1 = \delta' = q.k + r$).

Now using $n - i - b = \beta \cdot q$, we show that $n - i + 1 = \beta \cdot q + a + r$ then $n - i + 1 = (\alpha + \beta) \cdot q + r$ and hence $k = \alpha + \beta$ (since $n - i + 1 = q.k + r$).

However, by definition of α and β, we may argue that the number of occurrences of the value 1 on $y_0, \ldots, y_{a-1}$ is at most $\alpha \cdot p$ and at most $\beta \cdot p$ on $y_{b+1}, \ldots, y_{n-i}$.

Now since the demand $d = (\alpha + \beta).p + r$ then all the p variables the subsequence $y_a, \ldots, y_b$ must take the value 1.

We use a similar argument for the second case. Suppose that $d \bmod p = 0$, consider a subsequence $y_a, \ldots, y_b$ such that $a \bmod q = 0$ and $b = a + p - 1$.

Then there exist two integers α and β such that $a = \alpha \cdot q$ and $n - i - b = \beta \cdot q$. Therefore, the number of occurrences of the value 1 on $y_0, \ldots, y_{a-1}$ is at most $\alpha \cdot p$ and at most $\beta \cdot p$ on $y_{b+1}, \ldots, y_{n-i}$.

Now using the demand $d = k \cdot p$, and $\delta' = q\ (\lceil d/p\ \rceil - 1) + p$ we show that $n - i + 1 = q(k - 1) + p$. However, since $b = a + p - 1$, $a = \alpha \cdot q$ and $n - i - b = \beta \cdot q$, then $k = \alpha + \beta + 1$ and all p variables the subsequence $y_a, \ldots, y_b$ must take the value 1. □

Figure 2 and 3 depict the proposed pruning. On the one hand, when $d \bmod p = 0$, the only possible arrangement of vehicles that satisfy the capacity constraint is to start the sequence with p vehicles requiring the option, then $q - p$ vehicles not requiring the option and repeat (see Figure 2). Notice that because of the capacity constraint, all other variables must take the value 0. On the other hand, when $d \bmod p \neq 0$, one must start the sequence with $d \bmod p$ vehicles requiring the option, then the following $q - (d \bmod p)$ slots can be filled arbitrarily as long as exactly p vehicles requiring this options are fitted in the q first slots. Here again, the initial sequence must be repeated throughout (see Figure 3).

5.3. Time Complexity

Observe that this rule is extremely cheap to enforce. Once one has computed the load, the domain filtering can be achieved in $O(k)$ where k is

Figure 2: Filtering when $d \bmod p = 0$

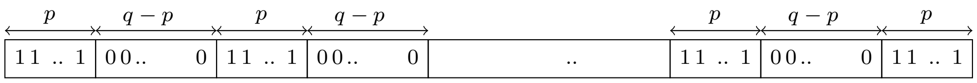
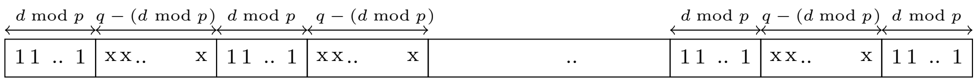


Figure 3: Filtering when $d \bmod p \neq 0$



the number of option variables forced to take the value 1. Indeed, when $d \bmod p \neq 0$ we can jump over the variables which are not forced to take the value 1, since their position is given by a simple recursion. In the worst case, i.e., when $d \bmod p = 0$, this is the same time complexity as achieving arc consistency on the AtMostSeqCard constraint [16]. However whilst the propagator of AtMostSeqCard always achieves its worst case complexity, this rule rarely does in practice.

6. Evaluating the filtering rules

In order to evaluate the slack-based filtering rule, we propose to compare its results against the other propagators. Notice first that, as we mentioned in section 5, this rule can be applied only with *lex* branching. In fact, we can use the following set of heuristics $\langle\{class, opt\}, lex, \{1, q/p, d, \delta, n-\sigma, \rho\}, \{\leq_{\sum}, \leq_{Euc}, \leq_{lex}\}\rangle$. That is 21 different heuristics for each filtering algorithm. The experiments concern 9030 configuration per propagator.

Table 10: Evaluation of the filtering variants (averaged over all heuristics)

Filtering (×21)	set1 (70 × 5)			set2 (4 × 5)			set3 (5 × 5)			set4 (7 × 5)			Global
	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%tot
sum	75.8	190636.0	11.2	22.6	792179.8	44.4	0.0	-	-	7.7	194651.7	17.0	63.4
Gsc	94.8	1639.4	4.2	44.0	38673.7	49.2	**2.8**	49417.9	260.8	12.1	35302.0	64.3	80.4
AtMostSeqCard	91.2	36285.7	3.9	**49.2**	**411514.8**	**46.2**	1.5	68873.9	15.1	**13.1**	**239317.8**	**41.4**	77.7
Gsc ⊕ AtMostSeqCard	**95.1**	**1585.1**	**4.3**	44.0	35711.3	45.4	**2.8**	**46330.2**	**248.6**	12.5	32258.4	80.9	**80.6**
Slack-based	90.5	55384.8	3.8	43.3	627443.4	43.9	1.7	82815.9	16.1	12.2	356073.4	34.8	76.7

Table 10 shows that the extra filtering of the rule we introduced or from the existing ones (i.e. AtMostSeqCard, Gsc and Gsc⊕ AtMostSeqCard)

Table 11: Best results for filtering variants

Filtering	set1 (70 × 5)			set2 (4 × 5)			set3 (5 × 5)			set4 (7 × 5)		
	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time	%sol	avg bts	time
sum	**100.0**	**184.8**	**0.0**	75.0	7251.3	0.5	0.0	-	-	25.7	4843.0	0.4
Gsc	**100.0**	184.8	1.2	75.0	18073.7	58.2	**40.0**	46211.8	261.9	**28.5**	29632.6	58.5
AtMostSeqCard	**100.0**	**184.8**	**0.0**	**100.0**	**730687.4**	**89.5**	20.0	60460.4	13.5	**28.5**	**31617.6**	**6.0**
Gsc⊕AtMostSeqCard	**100.0**	184.8	1.2	75.0	16923.7	55.0	**40.0**	**46196.7**	**259.7**	**28.5**	17252.6	40.8
Slack-based	**100.0**	**184.3**	**0.0**	75.0	510189.0	35.1	20.0	70573.6	14.0	**28.5**	332430.9	34.3

does help a lot. For instance, at least 90% of the instances of the first set are resolved irrespectively of the heuristic being used against 75,89% with the default decomposition (i.e. sum). The difference is even greater for the other sets.

At the outset, the gain of the new propagator seems comparable with the others. However, this method is in fact very different from the Gsc and quite close to the AtMostSeqCard constraint.

The Gsc constraint saves many more backtracks than the others. It does not subsume the pruning of our filtering rule, but it is much stronger in terms of search tree size. However the overhead of our method is negligible, whereas Gcs greatly slows down the search at the same time that it reduces its size.

Consider now the propagation method as a fifth criterion (i.e. in addition to the heuristic factors). We calculated its *Confidence* and *Significance* according to the same formula given in Section 4.3. Their values are equal to (respectively) 0.996 and 0.217. This is similar to all other criteria in terms of confidence (i.e. between 0.989 and 1.0), but slightly less than the *Significance* of branching and selection. This emphasizes the importance of these factors which are at least as important as the propagation level.

Overall, we observe that the choice of the search strategy has a very significant impact on the efficiency of the method. For instance, on the set of easiest instances, when averaging across all heuristics, the "worst" filtering method (decomposition into sum constraints) is successful in about 20% less runs than the best (Gsc+ AtMostSeqCard).

However, now averaging across all four models, the worst heuristic $\langle opt, lex, n-\sigma, -\rangle$, is successful 56% less runs than one of the many heuristics solving all easy instances (see table 4). For harder instances (set2, 3 and 4), these choices are even more important, with a 42% gap between the best and worst model, whilst the worst heuristics (in this case $\langle opt, lex, p/q, -\rangle$) do not solve any instances. It is hardly a surprise to observe that the choice of search strategy is a critical one. However, whilst the aim of this study was to better understand what makes a good heuristic for the car-sequencing problem, it was relatively surprising to find out that minor variations around known heuristics would bring such a substantial gain.

Finally, as a comparison with other CP-based methods [19, 10, 6]. The first set was totally resolved in less than a second unlike other methods. The second set results are as good as the best ones. However, the Regular constraint [19] maintains best results for proving infeasibility by a difference of one instance. We are not aware of any approach treating the fourth set of as decision problems. These instances are often treated in an optimization context.

7. Conclusion

Throughout this paper, we empirically studied a large set of heuristics for the car-sequencing problem and proposed to classify these heuristics using 4 criteria: branching variables, exploration directions, parameters for the selection of branching variables, and aggregation functions for these criteria. The experiments show the interest of some classification criteria (the branching and the selection) and a low impact of the other criteria (exploration directions and aggregation functions). Moreover, the study shows that one criterion can drastically effect the heuristic behavior.

The second contribution of this paper is the slack-based filtering algorithm. Even though quite simple and easy to implement, the filtering algorithm introduced in this paper is often as good as state-of-the-art propagators for the car-sequencing problem, and better in some cases.

A natural extension of this analysis is to study the impact of these heuristics for solving optimization variants for the car-sequencing problem. Such study gives insights for the industrial and optimization communities working on this problem. This can be achieved by decomposing the problem into a sequence of satisfaction problems or by using branching strategies directly within meta-heuristics such as Local Search and Ant Colony Optimization (see for instance [6]).

Acknowledgement

We would like to thank our Cork Constraint Computation Center colleagues for giving us access to their computing resources.

References

[1] Simon Boivin, Marc Gravel, Michaël Krajecki, and Caroline Gagné. Résolution du problème de car-sequencing à l'aide d'une approche de type FC. In *Premières Journées Francophones de Programmation par Contraintes*, pages 11–20, 2005.

[2] Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck. Solving the Car-Sequencing Problem in Constraint Logic Programming. In *Proceedings of ECAI*, pages 290–295, 1988.

[3] Pieter Andreas Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of ECAI*, pages 31–35, 1992.

[4] Ian P. Gent and Toby Walsh. CSPlib: a Benchmark Library for Constraints. In *Proceedings of CP*, pages 480–481, 1999.

[5] Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 431–437, 1998.

[6] Jens Gottlieb, Markus Puchta, and Christine Solnon. A Study of Greedy, Local Search and Ant Colony Optimization Approaches for Car Sequencing Problems. In *Proceedings of Applications of Evolutionary Computing*, pages 246–257, 2003.

[7] Khalil S. Hindi and Grzegorz Ploszajski. Formulation and solution of a selection and sequencing problem in car manufacture. *Computers & Industrial Engineering*, 26(1):203 – 211, 1994.

[8] Tamas Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335, 2004.

[9] Kim Marriott and Peter J Stuckey. *Programming with constraints: an introduction.* MIT press, 1998.

[10] Laurent Perron and Paul Shaw. Combining Forces to Solve the Car Sequencing Problem. In *Proceedings of CPAIOR*, pages 225–239, 2004.

[11] Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation Guided Large Neighborhood Search. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 468–481. Springer, 2004.

[12] Claude-Guy Quimper, Alexander Golynski, Alejandro López-Ortiz, and Peter van Beek. An Efficient Bounds Consistency Algorithm for the Global Cardinality Constraint. *Constraints*, 10(2):115–135, 2005.

[13] Jean Charles Régin. Generalized Arc Consistency for Global Cardinality Constraint. In *Proceedings of AAAI*, volume 2, pages 209–215, 1996.

[14] Jean-Charles Régin and Jean-François Puget. A Filtering Algorithm for Global Sequencing Constraints. In *Proceedings of CP*, pages 32–46, 1997.

[15] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming.* Elsevier, 2006.

[16] Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. An Optimal Arc Consistency Algorithm for a Particular Case of Sequence Constraint. *Constraints*, 19(1):30–56, 2014.

[17] Barbara M. Smith. Succeed-first or Fail-first: A Case Study in Variable and Value Ordering. Technical report, University of Leeds, 1996.

[18] Christine Solnon, Van Dat Cung, Alain Nguyen, and Christian Artigues. The car sequencing problem: Overview of state-of-the-art methods and industrial case-study of the ROADEF'2005 challenge problem. *European Journal of Operational Research*, 191:912–927, 2008.

[19] Willem J. van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal. New Filtering Algorithms for Combinations of Among Constraints. *Constraints*, 14(2):273–292, 2009.