Learning the Parameters of Global Constraints Using Branch-and-Bound

Émilie Picard-Cantin[1], Mathieu Bouchard[2],
Claude-Guy Quimper[1], and Jason Sweeney[2]

[1] Université Laval
emilie.picard-cantin.1@ulaval.ca, claude-guy.quimper@ift.ulaval.ca
[2] PetalMD
mathbouchard@gmail.com, jason.pierre.sweeney@gmail.com

Abstract. Precise constraint satisfaction modeling requires specific knowledge acquired from multiple past cases. We address this issue with a general branch-and-bound algorithm that learns the parameters of a given global constraint from a small set of positive solutions. The idea is to cleverly explore the possible combinations taken by the constraint's parameters without explicitly enumerating all combinations. We apply our method to learn parameters of global constraints used in timetabling problems such as Sequence and SubsetFocus. The later constraint is our adaptation of the constraint Focus to timetabling problems.

Keywords: Constraint Acquisition, Timetabling, Machine Learning, CSP, Global Constraints, Brand-and-Bound

1 Introduction

Modeling a constraint satisfaction problems requires specific knowledge acquired from multiple past cases, each model being different from the last. For example, CSPs of nurse timetabling problems for two different hospitals most likely use similar constraints but with different parameters. A hospital might require to work in the emergency ward no more than 3 days out of 7 while another hospital might set the limit to no more than 4 days out of 9. A system able to determine the parameters that created a set of existing solutions would greatly speedup the modeling process. This explains the popularity of modeling automation in the recent years.

A global constraint has variables, encoding solutions, and known parameters that define the relation between the variables. In our context, we are given a global constraint and example solutions, so the variables are known but the parameters are unknown. We want to learn the parameters that generated the examples.

The main contribution of this paper is an algorithm that learns the parameters of given global constraints from a small pool of examples. This algorithm can be applied to constraints such as Among and Sequence, commonly used in

timetabling. We use a branch-and-bound to quickly and cleverly travel through all the combinations of parameters of the constraint and to determine which combination best describes the given examples.

We also introduce SubsetFocus, a constraint useful in the modelization of timetabling problems, and we show how to learn its parameters from examples.

2 Background

We present the global constraints that are studied in this paper. We then report important past contributions related to the automation of constraint modeling and to the learning of parameters for global constraints.

2.1 Global constraints

Let $\mathrm{C}(\boldsymbol{X}, \boldsymbol{a})$ be a global constraint where $\boldsymbol{X} = [X_1, \ldots, X_n]$ are integer variables denoted in upper case and $\boldsymbol{a} = [a_1, \ldots, a_m]$ are parameters denoted in lower case. Let $S_{\mathrm{C}}(\boldsymbol{a})$ be the solution set for C with parameters $\boldsymbol{a}$ and let $S_{\mathrm{C}_i}(x, \boldsymbol{a}) = S_{\mathrm{C}}(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_m)$, where all a_j in $\boldsymbol{a}$ are fixed except for a_i.

We present global constraints that are studied in this paper. Let $s_{ij} = \{i, i+1, \ldots, j\}$ be a sequence. The constraint $\textsc{Focus}(\boldsymbol{X}, Y, l, k)$ [28] is satisfied if and only if $\boldsymbol{X}$ and Y are such that there exists a set $\mathcal{S}$ of disjoint sequences s_{ij} such that

1. $|\mathcal{S}| \leq Y$
2. $X_b > k \iff \exists s_{ij} \in \mathcal{S}$ s.t. $b \in s_{ij}, \quad \forall b \in [1, \ldots, n]$
3. $j - i + 1 \leq l, \quad \forall s_{ij} \in \mathcal{S}$

Focus controls the number and the length of subsequences of variables greater than k.

Example 1. Suppose we have $\textsc{Focus}([X_1, \ldots, X_7], Y, l = 2, k = 1)$ with $\mathrm{dom}(X_i) = \{1, 2, 3, 4\}$ and $\mathrm{dom}(Y) = \{2, 3\}$. Then, $[2, 1, 3, 3, 1, 1, 1]$ and $[2, 3, 4, 1, 1, 2, 1]$ are solutions because $\mathcal{S} = \{s_{11}, s_{34}\}$ and $\mathcal{S} = \{s_{12}, s_{33}, s_{66}\}$ are valid sets. $[2, 1, 2, 1, 2, 1, 2]$ is not a solution because the cardinality of the only possible set $\mathcal{S} = \{s_{11}, s_{33}, s_{55}, s_{77}\}$ is greater than $\max(\mathrm{dom}(Y))$.

Some global constraints are applied to a subset of values in their original definition. We choose to encode this set with a vector $\boldsymbol{z}$. $z_v = 1$ if the value v is in the set and $z_v = 0$ otherwise. In other words, the vector $\boldsymbol{z}$ is the bitset encoding of a set. By abuse of notation, we will consider $\boldsymbol{z}$ sometimes as a binary vector, sometimes as a set of values.

We propose a new constraint $\textsc{SubsetFocus}(\boldsymbol{X}, l, m, \boldsymbol{z})$ (SF), a generalization of Focus, that controls the number and the length of subsequences of variables that belong to a set of values. $\textsc{SubsetFocus}(\boldsymbol{X}, l, m, \boldsymbol{z})$ is satisfied if $\boldsymbol{X}$ is such that there exists $\mathcal{S}$, a set of disjoint sequences s_{ij}, such that:

1. $|\mathcal{S}| \leq m$

2. $X_b = v \wedge z_v = 1 \iff \exists s_{ij} \in \mathcal{S}$ s.t. $b \in s_{ij}, \quad \forall b \in [1, \ldots, n]$
3. $j - i + 1 \leq l, \quad \forall s_{ij} \in \mathcal{S}$
4. $s_{ij} \in \mathcal{S} \Rightarrow s_{j+1,j'} \notin \mathcal{S}, \quad \forall j' \geq j + 1.$

In other words, SubsetFocus$(\boldsymbol{X}, l, m, \boldsymbol{z})$ is satisfied if the assignment $\boldsymbol{X}$ has fewer than m stretches of maximum length l with values in the set defined by $\boldsymbol{z}$. SubsetFocus can be used to limit the number of stretches of night shifts to m while limiting the length of the stretches to l in a medical timetabling problem.

We introduce SubsetFocus to fulfill a request from PetalMD, the company financing this research, to create medical schedules with clusters of night shifts. The set $\mathcal{S}$ represents sequences of consecutive night shifts. In our context, two consecutive night shifts should not be considered as two separate stretches of work as it could be with Focus. Hence the fourth condition of SubsetFocus.

Example 2. Suppose we have SubsetFocus$([X_1, \ldots, X_7], l = 2, m = 2, \boldsymbol{z} = [0, 1, 1, 1])$ with $\text{dom}(X_i) = \{1, 2, 3, 4\}$. Then, $[2, 1, 3, 3, 1, 1, 1]$ is a solution because $\mathcal{S} = \{s_{11}, s_{34}\}$ is a valid set. $[2, 3, 4, 1, 1, 1, 1]$ is not a solution because s_{13} is not a valid sequence according to condition 3 and all other combinations would violate condition 4.

Among$(\boldsymbol{X}, l, u, \boldsymbol{z})$ [4] ensures that at most u and at least l variables take values in $\boldsymbol{z}$. Sequence$(\boldsymbol{X}, l, u, w, \boldsymbol{z})$ (Seq) [4] ensures that for every subset of w consecutive variables in $[X_1, \ldots, X_n]$ Among$([X_i, \ldots, X_{i+w-1}], l, u, \boldsymbol{z})$ holds.

Let $\text{occ}_v = |\{i : X_i = v\}|$ be the number of occurrences of value v in $\boldsymbol{X}$. GCC$(\boldsymbol{X}, [l_1, \ldots, l_m], [u_1, \ldots, u_m])$ (or GlobalCardinality constraint) [30] ensures that $\text{occ}_v \in [l_v, u_v]$, for each value $v \in \{1, \ldots, m\}$. AtMostNValue$(\boldsymbol{X}, k)$ (AtLeastNValue$(\boldsymbol{X}, k)$) [26] ensures that the variables take at most (at least) k different values.

Balance$(\boldsymbol{X}, b)$ [9] ensures that the balance b is the difference between the most occurring value and the least occurring value, among assigned values only.

$$b = \max_{v \in \{X_i\}_{i=1}^n} \text{occ}_v - \min_{v \in \{X_i\}_{i=1}^n} \text{occ}_v \,.$$

The constraint AtMostBalance$(\boldsymbol{X}, b)$ [16] ensures that the balance is at most b. The variant AtMostBalance$^*(\boldsymbol{X}, b)$ also takes into consideration non-occurring values, i.e.: $b \geq \max_v \text{occ}_v - \min_v \text{occ}_v$.

Pesant et al. [27] count the solutions for multiple global constraints. In particular, they propose a dynamic programming approach to count the solutions satisfying Regular$([X_1, \ldots, X_n], \mathcal{A})$, that ensures the word $[X_1, \ldots, X_n]$ belongs to the regular language described by the finite automaton $\mathcal{A}$. The idea is to encode Regular as a layered graph and then recursively count all paths, from the last layer to the first.

2.2 Constraint Acquisition

There exist multiple approaches to learn, from a set of solutions, which constraints form a model. The *model seeker* [8, 5] learns a CSP from positive examples. It lists the global constraints satisfied by all examples using the *constraint*

seeker [6, 7], which uses multiple criteria to order the constraints according to pertinence. One criterion is the number of assignments that satisfy the constraint. If this number is small, that indicates that the solution come from a small subset of possible assignments. This is more likely to occur if the constraint was imposed.

Many propose interactive constraint acquisition systems learning a complete CSP from examples by asking queries to the user. Among the most recent publications : Bessiere et al. [10, 15, 14], Daoudi et al. [20], and Arcangioli and Lazaar [1].

Bessiere et al. [12, 13], Bartak et al. [2], and Charnley et al. [19] study the acquisition of implied constraints from a CSP in order to improve the resolution time. Bessiere et al. [11] use the examples to remove redundant constraints from the model during the constraint acquisition phase.

Kiziltan et al. [21], Little et al. [23], and Lopez and Lallouet [25] study the translation of a problem description written in natural language into a formal model.

Machine learning techniques can also be used to learn part of a model. Bonfietti et al. [17], Bartolini et al. [3], and Lombardi et al. [24] train neural networks or decision trees to recognize a solution to a problem and then embed the trained neural network or the trained decision trees into a global constraint.

Campigotto et al. [18] and Kolb [22] study the acquisition of the utility function of the optimization model.

Suraweera et al. [31] propose a system that learns parameters for template constraints previously defined from historical schedules. The quality of a parameter set is the distance to the real set of parameters that created the examples. They choose the parameters with the highest quality.

2.3 Learning parameters of a global constraint

Picard-Cantin et al. [29] consider a global constraint $\mathrm{C}(\boldsymbol{X}, \boldsymbol{a})$ whose scope $\boldsymbol{X}$ is known and want to learn its parameters $\boldsymbol{a}$ from a small set of examples $E = \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_q\}$. Each parameter a_i must take a value from a predefined set of values called *domain* denoted $\mathrm{dom}(a_i)$. They list all combinations of parameters satisfied by the examples and choose the combination with the lowest probability of being satisfied, since it has the highest chance of being imposed by a mathematical model. Let $G_{\mathrm{C}}(\boldsymbol{a})$ be the probability that a random assignment $\boldsymbol{X}$ satisfies $\mathrm{C}(\boldsymbol{X}, \boldsymbol{a})$. The goal is to solve the following optimization problem.

$$\begin{aligned} \min_{\boldsymbol{a}} \quad & G_{\mathrm{C}}(\boldsymbol{a}) \\ \text{s.t.} \quad & \bigwedge_{\boldsymbol{e} \in E} \mathrm{C}(\boldsymbol{e}, \boldsymbol{a}) \\ & a_i \in \mathrm{dom}(a_i) \end{aligned}$$

The method assumes that the domains of $X_i \in \boldsymbol{X}$ are identical. Let $p_v = P[X_i = v]$, the probability of assigning the value v to any variable. Recall that

$S_{\mathrm{C}}(\boldsymbol{a})$ is the solution set for C with parameters $\boldsymbol{a}$. Therefore, $P[\boldsymbol{e}] = \prod_{i=1}^{n} p_{e_i}$ and $G_{\mathrm{C}}(\boldsymbol{a})$ is the sum of the probabilities of each solution.

$$G_{\mathrm{C}}(\boldsymbol{a}) = \sum_{\boldsymbol{e} \in S_{\mathrm{C}}(\boldsymbol{a})} P[\boldsymbol{e}] = \sum_{\boldsymbol{e} \in S_{\mathrm{C}}(\boldsymbol{a})} \prod_{i=1}^{n} p_{e_i} \tag{1}$$

Picard-Cantin et al. encode the constraints using an automaton that they transform into a Markov chain by adding on each transition the probability of reading the associated value. The Markov chain efficiently computes the probability $G_{\mathrm{C}}(\boldsymbol{a})$. To solve the optimization problem, they iterate over all combinations of parameters $\boldsymbol{a}$ while avoiding *dominated* parameters.

With this method, Picard-Cantin et al. [29] learn the parameters of global constraints that can be encoded as Regular, such as Sequence and Among. The drawback is that listing all combinations of parameters for a global constraint quickly becomes infeasible, specifically when a parameter is a set. For example, there are $\frac{n^2}{2}\sum_{i=1}^{k}\binom{k}{i}$ combinations of parameters for SubsetFocus$([X_1, \ldots, X_n], l, m, [z_1, \ldots, z_k])$. In this case, having a clever way to explore those combinations becomes crucial.

3 Methodology

We present a more refined approach to explore the space of parameters. We also show how to compute $G_{\mathrm{C}}(\boldsymbol{a})$ for the constraints described in Section 2.1.

We propose a general branch-and-bound to explore the values that can be given to the parameters. The objective of the algorithm is to find the parameter values minimizing the probability function G for a given global constraint. Therefore, the bounding algorithm needs to compute a lower bound on G.

The branch-and-bound is presented in Algorithm 1 and it requires two sub-algorithms specific to the global constraint for which we wish to learn the parameters. The first, FilterParameters$_{\mathrm{C}}([\mathrm{dom}'(a_1), \ldots, \mathrm{dom}'(a_m)], E)$, filters the domains of the parameters using the given examples. The second, ComputeLowerBound$_{\mathrm{C}}([\mathrm{dom}'(a_1), \ldots, \mathrm{dom}'(a_m)])$, computes an optimist lower bound on the probability function G. Those two are called at each node of the search tree.

3.1 Monotonicity

We also identify a situation where a subset of parameters can be fixed to their extreme values for the computation of the lower bound on G, simplifying bound computation. These parameters are such that the probability function G is monotonic w.r.t. those parameters (see Section 3.1).

Let $F : D^n \to \mathbb{R}$ be a multivariate function. F is said to be *monotonic w.r.t.* a_i if and only if $F_i(x, \boldsymbol{a}) = F(a_1, \ldots, a_{i-1}, x, a_{i+1} \ldots, a_m)$ is monotonic.

Algorithm 1: BranchAndBound(C, $[\mathrm{dom}(a_1), \ldots, \mathrm{dom}(a_m)]$, E)

```
1  S ← {⟨0, [dom(a_1), ..., dom(a_m)]⟩}
2  BestSolution ← null
3  BestBound ← ∞
4  while min{lb | ⟨lb, [dom(a_1), ..., dom(a_m)]⟩ ∈ S} < BestBound do
5      ⟨lb, [dom(a_1), ..., dom(a_m)]⟩ ← arg min_{⟨lb,[dom(a_1),...,dom(a_m)]⟩∈S} lb
6      Remove ⟨lb, [dom(a_1), ..., dom(a_m)]⟩ from S
7      if |dom(a_i)| = 1 ∀i ∈ {1, ..., m} then
8          BestSolution ← [dom(a_1), ..., dom(a_m)]
9          BestBound ← lb
10     else
11         Choose a_i such that |dom(a_i)| > 1
12         for v ∈ dom(a_i) do
13             for j ∈ {1, ..., m}\{i} do
14                 dom'(a_j) = dom(a_j)
15             dom'(a_i) = {v}
16             FilterParametersC([dom'(a_1), ..., dom'(a_m)], E)
17             lb' ← ComputeLowerBoundC([dom'(a_1), ..., dom'(a_m)])
18             S ← S ∪ {⟨lb', [dom'(a_1), ..., dom'(a_m)]⟩}
19 return BestSolution,BestBound
```

Theorem 1. *A global constraint* $\mathrm{C}(\boldsymbol{X}, \boldsymbol{a})$ *has a monotonic probability function* $G_{\mathrm{C}}(\boldsymbol{a})$ *w.r.t.* a_i *if either*

$$x \leq y \Rightarrow S_{\mathrm{C}_i}(x, \boldsymbol{a}) \subseteq S_{\mathrm{C}_i}(y, \boldsymbol{a}), \quad \forall x, y \in \mathrm{dom}(a_i)$$

or

$$x \leq y \Rightarrow S_{\mathrm{C}_i}(y, \boldsymbol{a}) \subseteq S_{\mathrm{C}_i}(x, \boldsymbol{a}), \quad \forall x, y \in \mathrm{dom}(a_i).$$

Proof. Suppose that for all $x, y \in \mathrm{dom}(a_i)$, $x \leq y$ implies $S_{\mathrm{C}_i}(x, \boldsymbol{a}) \subseteq S_{\mathrm{C}_i}(y, \boldsymbol{a})$. Then, for all $x, y \in \mathrm{dom}(a_i)$ such that $x \leq y$.

$$\begin{aligned}
& G_{\mathrm{C}}(a_1, \ldots, a_{i-1}, y, a_{i+1}, a_m) \\
= & \sum_{\boldsymbol{e} \in S_{\mathrm{C}_i}(y, \boldsymbol{a})} P[\boldsymbol{e}] \\
= & \sum_{\boldsymbol{e} \in S_{\mathrm{C}_i}(x, \boldsymbol{a})} P[\boldsymbol{e}] + \sum_{\boldsymbol{e} \in S_{\mathrm{C}_i}(y, \boldsymbol{a}) \setminus S_{\mathrm{C}_i}(x, \boldsymbol{a})} P[\boldsymbol{e}] \\
\geq & G_{\mathrm{C}}(a_1, \ldots, a_{i-1}, x, a_{i+1}, a_m)
\end{aligned}$$

A similar argument holds when $x \leq y \Rightarrow S_{\mathrm{C}_i}(y, \boldsymbol{a}) \subseteq S_{\mathrm{C}_i}(x, \boldsymbol{a}), \forall x, y \in \mathrm{dom}(a_i)$. □

If $G_{\mathrm{C}}(\boldsymbol{a})$ is monotonic w.r.t a_i for all $i \in \{1 \ldots, m\}$, then we can easily compute the lower bound on $G_{\mathrm{C}}(\boldsymbol{a})$ at each node of the branch-and-bound. We only need to fix the parameters to their extreme values. To ensure that the examples satisfy the constraint, we apply filtering techniques to prune the domains of the parameters. Note that the parameter n is always known in our case.

Corollary 1. *$G_{\textsc{SubsetFocus}}(n, l, m, \boldsymbol{z})$ is monotonic w.r.t l and m, but not $\boldsymbol{z}$.*

Proof. For all $l_1 \leq l_2$, we have $j - i + 1 \leq l_1 \leq l_2$ for all $s_{ij} \in \mathcal{S}$, therefore $S_{\textsc{SubsetFocus}}(n, \boldsymbol{X}, l_1, m, \boldsymbol{z}) \subseteq S_{\textsc{SubsetFocus}}(n, \boldsymbol{X}, l_2, m, \boldsymbol{z})$. For all $m_1 \leq m_2$, we have $|\mathcal{S}| \leq m_1 \leq m_2$, so $S_{\textsc{SubsetFocus}}(n, \boldsymbol{X}, l, m_1, \boldsymbol{z}) \subseteq S_{\textsc{SubsetFocus}}(n, \boldsymbol{X}, l, m_2, \boldsymbol{z})$. By Theorem 1, $G_{\mathrm{SF}}(n, l, m, \boldsymbol{z})$ is monotonic w.r.t parameters l and m.

Let $p_1 = 0.73$, $p_2 = 0.23$, and $n = 3$. We have $G_{\textsc{SubsetFocus}}(3, 3, 1, [1, 1]) = 1.0$, $G_{\textsc{SubsetFocus}}(3, 3, 1, [1, 0]) \approx 0.8561$, $G_{\textsc{SubsetFocus}}(3, 3, 1, [0, 1]) \approx 0.9468$, and $G_{\textsc{SubsetFocus}}(3, 3, 1, [0, 0]) = 1.0$. Therefore, $G_{\textsc{SubsetFocus}}(n, l, m, \boldsymbol{z})$ is not monotonic w.r.t. any z_i. □

Corollary 2. *$G_{\textsc{Among}}(n, l, u, \boldsymbol{z})$ is monotonic w.r.t. l and u, but not $\boldsymbol{z}$.*

Proof. For all l_1, l_2 such that $0 \leq l_1 \leq l_2 \leq u$, if $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \geq l_2$ then we have $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \geq l_1$. Therefore, $S_{\textsc{Among}}(n, \boldsymbol{X}, l_2, u, \boldsymbol{z}) \subseteq S_{\textsc{Among}}(n, \boldsymbol{X}, l_1, u, \boldsymbol{z})$. For all u_1, u_2 such that $u_1 \leq u_2 \leq n$, if $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \leq u_1$ then we have $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \leq u_2$. Meaning that $S_{\textsc{Among}}(n, \boldsymbol{X}, l, u_1, \boldsymbol{z}) \subseteq S_{\textsc{Among}}(n, \boldsymbol{X}, l, u_2, \boldsymbol{z})$. Therefore, by Theorem 1 $G_{\textsc{Among}}(n, l, u, \boldsymbol{z})$ is monotonic w.r.t. l and u.

Let $p_1 = 0.73$, $p_2 = 0.23$, and $n = 3$. We have $G_{\textsc{Among}}(3, 2, 2, [1, 0]) \approx 0.2878$, $G_{\textsc{Among}}(3, 2, 2, [1, 1]) = 0.0$, $G_{\textsc{Among}}(3, 3, 3, [1, 1]) = 1.0$ and $G_{\textsc{Among}}(3, 3, 3, [1, 0]) \approx 0.389$. Therefore, $G_{\textsc{Among}}(n, l, u, \boldsymbol{z})$ is not monotonic w.r.t. any z_i. □

Corollary 3. *$G_{\textsc{Seq}}(n, l = 0, u, w, \boldsymbol{z})$ is monotonic w.r.t. u and w, but not $\boldsymbol{z}$.*

Proof. For all $0 \leq u_1, u_2$ s.t. $u_1 \leq u_2 \leq n$, if $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \leq u_1$ then $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \leq u_2$ and $S_{\textsc{Seq}}(n, \boldsymbol{X}, 0, u_1, w, \boldsymbol{z}) \subseteq S_{\textsc{Seq}}(n, \boldsymbol{X}, 0, u_2, w, \boldsymbol{z})$. If $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \leq u$ for all subsequences of length w_2 and if $w_1 \leq w_2$, then $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \leq u$ for all subsequences of length w_1 and $S_{\textsc{Seq}}(n, \boldsymbol{X}, 0, u, w_2, \boldsymbol{z}) \subseteq S_{\textsc{Seq}}(n, \boldsymbol{X}, 0, u, w_1, \boldsymbol{z})$. Therefore, by Theorem 1 $G_{\textsc{Seq}}(n, 0, u, w, \boldsymbol{z})$ is monotonic w.r.t. u and w. By Corollary 2, $G_{\textsc{Seq}}(n, 0, u, w, \boldsymbol{z})$ is not monotonic w.r.t. $\boldsymbol{z}$ when $n = w = 3$. □

Corollary 4. *$G_{\textsc{Seq}}(n, l, u = n, w, \boldsymbol{z})$ is monotonic w.r.t. l and w, but not $\boldsymbol{z}$.*

Proof. For all l_1, l_2 such that $0 \leq l_1 \leq l_2 \leq n$, if $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \geq l_2$ for all subsequences of length w then $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \geq l_1$ and $S_{\textsc{Seq}}(n, \boldsymbol{X}, l_2, u, w, \boldsymbol{z}) \subseteq S_{\textsc{Seq}}(n, \boldsymbol{X}, l_1, u, w, \boldsymbol{z})$. If $w_1 \leq w_2$ and if $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \geq l$ for all subsequences of length w_1, then we have $\sum_{v \in \boldsymbol{z}} \mathrm{occ}_v \geq l$ for all subsequences of length w_2 and $S_{\textsc{Seq}}(n, \boldsymbol{X}, l, u, w_1, \boldsymbol{z}) \subseteq S_{\textsc{Seq}}(n, \boldsymbol{X}, l, u, w_2, \boldsymbol{z})$. Therefore, $G_{\textsc{Seq}}(n, l = 0, u, w, \boldsymbol{z})$ is monotonic w.r.t. u and w by Theorem 1. $G_{\textsc{Seq}}(n, l, u = n, w, \boldsymbol{z})$ is not monotonic w.r.t. $\boldsymbol{z}$ by Corollary 2 using $n = w = 3$. □

Corollary 5. *$G_{\mathrm{GCC}}(n, \boldsymbol{l}, \boldsymbol{u})$ is monotonic w.r.t. each l_i and each u_i.*

Proof. For any $v \in \{1, \ldots, m\}$, if $l_v \leq l'_v$ and $u_v \leq u'_v$, then $l_v \leq l'_v \leq \text{occ}_v \leq u_v \leq u'_{v1}$. Therefore, $S_{\text{GCC}}(n, \boldsymbol{l}, \boldsymbol{u}) \subseteq S_{\text{GCC}}(n, [l_1, \ldots, l_{v-1}, l'_v, l_{v+1}, \ldots, l_m], \boldsymbol{u})$ and $S_{\text{GCC}}(n, \boldsymbol{l}, \boldsymbol{u}) \subseteq S_{\text{GCC}}(n, \boldsymbol{l}, [u_1, \ldots u_{v-1}, u'_v, u_{v+1}, \ldots u_m])$. □

Corollary 6. *Both* $G_{\text{AtMostNValue}}(n, k)$ *and* $G_{\text{AtLeastNValue}}(n, k)$ *are monotonic w.r.t.* k*.*

Proof. Solutions with at most k values form a subset of the solutions with at most $k + 1$ values. A symmetric argument holds for AtLeastNValue. □

Corollary 7. *Both* $G_{\text{AtMostBalance}}(n, b)$ *and* $G_{\text{AtMostBalance}^*}(n, b)$ *are monotonic w.r.t.* b*.*

Proof. Solutions with a balance no greater than b form a subset of solutions with a balance no greater than $b + 1$, regardless of how the balance is computed. □

The previous theorem and corollaries are used in the computation of the bound on G in the branch-and-bound algorithm. These results show that we can temporarily fix the monotonic parameters to their extreme values to compute an optimist bound.

3.2 Bounding and filtering specific constraints

We show how to implement the functions $\text{FilterParameters}_{\text{C}}$ and $\text{ComputeLowerBound}_{\text{C}}$ for the constraints Among, Sequence, SubsetFocus, GCC, AtMostNValue, AtLeastNValue, AtMostBalance, and AtMostBalance*.

SubsetFocus We define the function $\text{ComputeLowerBound}_{\text{SubsetFocus}}$ that computes a lower bound on the probability $G_{\text{SubsetFocus}}(n, l, m, \boldsymbol{z})$. To take into consideration the other parameters, we consider the probability p that a variable X_i is assigned to a value v such that $z_v = 1$. During the branch-and-bound, the set $\boldsymbol{z}$ is only partially defined depending on which parameter z_v is assigned. However, this partial assignment allows bounding of the probability p as follows. The summations apply on instantiated parameters.

$$\alpha := \sum_{v:z_v=1} p_v, \qquad \beta := 1 - \sum_{v:z_v=0} p_v, \qquad \alpha \leq p \leq \beta$$

A solution $\boldsymbol{e}$ can be mapped to a binary vector $\boldsymbol{y}$ such that $y_i = 1 \iff \boldsymbol{e}_i = v \wedge z_v = 1$. The satisfiability of the constraint can be tested simply by checking the number and the length of the stretches of ones in the vector $\boldsymbol{y}$. Let $\gamma(\boldsymbol{e}) = |\{i \mid \boldsymbol{e}_i = v \wedge z_v = 1\}| = \sum_{i=1}^{n} y_i$ be the number of variables assigned to a value in the considered set for a given solution $\boldsymbol{e}$. Let $A(n, l, m, k)$ be the number of vectors $\boldsymbol{y}$ of size n with exactly k components set to 1 that satisfies

the constraint with parameters l and m. The probability $G_{\text{SUBSETFOCUS}}(n, l, m, \boldsymbol{z})$ can be computed as follows.

$$G_{\text{SF}}(n, l, m, \boldsymbol{z}) = \sum_{\boldsymbol{e} \in S_{\text{SF}}(\boldsymbol{a})} P[\boldsymbol{e}] = \sum_{\boldsymbol{e} \in S_{\text{SF}}(\boldsymbol{a})} p^{\gamma(\boldsymbol{e})}(1-p)^{n-\gamma(\boldsymbol{e})} \tag{2}$$

$$= \sum_{k=0}^{n} A(n, l, m, k) p^k (1-p)^{n-k}. \tag{3}$$

Therefore, the probability $G_{\text{SUBSETFOCUS}}(n, l, m, \boldsymbol{z})$ can be bounded with

$$G_{\text{SF}}(n, l, m, \boldsymbol{z}) \geq \sum_{k=0}^{n} A(n, l, m, k) \min_{\alpha \leq p \leq \beta} p^k (1-p)^{n-k}.$$

Differentiating $x^k(1-x)^{n-k}$ yields the optima $k/n, 0, 1$. Since $0 \leq \alpha \leq \beta \leq 1$ and since k/n is a maximum, the bound used in the branching algorithm is

$$\sum_{k=0}^{n} A(n, l, m, k) \min_{p \in \{\alpha, \beta\}} p^k (1-p)^{n-k}. \tag{4}$$

We now have to show how to compute the function $A(n, l, m, k)$. Let $D(l, k, m)$ be the number of ways to split a sequence of length k into exactly m subsequences of length at most l. The function $D(l, k, m)$ can be recursively computed.

$$D(l, k, m) = \begin{cases} 1 & \text{if } k = m = 0 \\ 0 & \text{if } m = 0 \wedge k > 0 \\ 0 & \text{if } m > 0 \wedge k = 0 \\ \sum_{j=1}^{\min(k,l)} D(l, k-j, m-1) & \text{otherwise} \end{cases}$$

To count the number of feasible binary vectors $\boldsymbol{y}$, we count how many ways we can segment a sequence of k 1s into exactly i sequences using the function $D(l, k, i)$. We insert $i-1$ zeros, one between each segment. There exist $\binom{n-k+1}{i}$ ways to insert the remaining $n-k-(i-1)$ zeros before or after the segments to obtain a vector of length n. Finally, we let the number of segments i vary between 1 and m.

$$A(n, l, m, k) = \sum_{i=1}^{m} D(l, k, i) \binom{n-k+1}{i}.$$

Since $G_{\text{SF}}(n, l, m, \boldsymbol{z})$ is monotonic w.r.t. m and l by Corollary 1, the function $\text{COMPUTELOWERBOUND}_{\text{SUBSETFOCUS}}$ returns the bound (4) by setting l and m to their smallest value in their domains.

During the branch-and-bound, we apply a minimal filtering to the domains of parameters l and m. Here is how we define $\text{FILTERPARAMETERS}_{\text{SUBSETFOCUS}}$. We suppose that all uninstantiated variables z_i could be instantiated to zero. In

that situation, we compute the length of the largest stretch in an example $\boldsymbol{e} \in E$ and set this length as a lower bound on $\text{dom}(l)$. To compute a lower bound on $\text{dom}(m)$, for each example $\boldsymbol{e} \in E$, we create a binary vector $\boldsymbol{y}$ by setting $y_i = z_{\boldsymbol{e}_i}$ if $z_{\boldsymbol{e}_i}$ is instantiated. When $z_{\boldsymbol{e}_i}$ is uninstantiated, we greedily assign a value to y_i to minimize the number of stretches in $\boldsymbol{y}$. We set $\min(\text{dom}(m))$ to the maximum number of stretches observed in one example. The parameters $\boldsymbol{z}$ are not filtered.

Lemma 1. *The bound on* $G_{\text{SUBSETFOCUS}}(n, l, m, \boldsymbol{z})$ *from* (4) *is tight when variables are instantiated.*

Proof. Since $\boldsymbol{z}$ are fixed, then $\sum_{v:z_v=1} p_v = \alpha = \beta = p$. Therefore, (4) is equal to (3) and the bound is tight. □

Sequence Consider the SEQUENCE constraint whose parameter l is known to be 0. We therefore need to learn parameters u, w, and $\boldsymbol{z}$. The probability $G_{\text{SEQ}}(n, u, w, \boldsymbol{z})$ can be bounded in a similar way as we did for SUBSETFOCUS. Let $B(n, u, w, k)$ be the number of binary vectors $\boldsymbol{y}$ of length n such that exactly k components are set to one and the sum of any subsequence of w components is at most u. Therefore, the function $\text{COMPUTELOWERBOUND}_{\text{SEQUENCE}}$ returns the following lower bound on $G_{\text{SEQ}}(n, u, w, \boldsymbol{z})$.

$$G_{\text{SEQ}}(n, u, w, \boldsymbol{z}) = \sum_{k=0}^{n} B(n, u, w, k) p^k (1-p)^{n-k} \tag{5}$$

$$\geq \sum_{k=0}^{n} B(n, u, w, k) \min_{p \in \{\alpha, \beta\}} p^k (1-p)^{n-k} \tag{6}$$

Since $G_{\text{SEQ}}(n, u, w, \boldsymbol{z})$ is monotonic w.r.t. u and w by Corollary 3, we compute $B(n, u, w, k)$ with u fixed to its minimal value and w fixed to its maximal value. We use dynamic programming to compute the function $B(n, u, w, k)$. Let $F(n, u, w, k, [s_1, \ldots, s_w])$ be the number of feasible binary vectors $\boldsymbol{y}$ of length n with exactly k components set to one and whose last w components are $[s_1, \ldots, s_w]$.

$$F(n, u, k, [s_1, \ldots, s_w]) = \begin{cases} 0 & \text{if } \sum_{i=1}^{n} s_i > \min(u, k) \\ 0 & \text{if } n - w + \sum_{i=1}^{n} s_i < k \\ 1 & \text{if } n = w \\ F(n-1, u, k - s_w, [0, s_1, \ldots, s_{w-1}]) & \\ +F(n-1, u, k - s_w, [1, s_1, \ldots, s_{w-1}]) & \text{otherwise} \end{cases}$$

$$B(n, u, w, k) = F(n + w, u, k, [\underbrace{0, \ldots, 0}_{w \text{ times}}])$$

The two first cases in function F occur when there are too many or two few ones in the sequence $[s_1, \ldots, s_w]$ to be extended to a feasible sequence of length n. The third case occurs when $n = w$ and $[s_1, \ldots, s_w]$ is a feasible solution. The last case computes the number of solutions with one fewer component. The

function $B(n, u, w, k)$ gives the number of sequences of length $n + w$ whose last w components are null.

We define FILTERPARAMETERS$_{\text{SEQUENCE}}$. Once the parameter w is instantiated, the parameter u can be filtered. For every solution $\boldsymbol{e} \in E$ and for every subsequence of length w in $\boldsymbol{e}$, we count the number of variables that must belong to $\boldsymbol{z}$. The largest encountered value gives a lower bound on u.

Lemma 2. *The bound on $G_{\text{SEQ}}(n, u, w, \boldsymbol{z})$ from (6) is tight when variables are instantiated.*

Proof. Since $\boldsymbol{z}$ are fixed, then $\sum_{v:z_v=1} p_v = \alpha = \beta = p$. Therefore, (6) is equal to (5) and the bound is tight. □

Among The probability that a random assignment has exactly k variables assigned to a value v such that $z_v = 1$ follows a binomial distribution. The probability to satisfy the constraint AMONG with parameters l, u, and $\boldsymbol{z}$ is therefore

$$G_{\text{AMONG}}(n, l, u, \boldsymbol{z}) = \sum_{k=l}^{u} \binom{n}{k} p^k (1-p)^{n-k} \tag{7}$$

As we did for SUBSETFOCUS, the function COMPUTELOWERBOUND$_{\text{AMONG}}$ returns the following lower bound.

$$G_{\text{AMONG}}(n, l, u, \boldsymbol{z}) \geq \sum_{k=l}^{u} \binom{n}{k} \min_{p \in \{\alpha, \beta\}} p^k (1-p)^{n-k} \tag{8}$$

By Corollary 2, we can simply fix u to its minimal value and l to its maximal value for the bound computation.

During the search, one can filter the parameters l and u using the function FILTERPARAMETERS$_{\text{AMONG}}$. Let $M = \{v \mid z_v = 1\}$ be the set of values that are considered and $C = \{v \mid z_v = 1 \vee z_v \text{ is uninstantiated}\}$ be the set of values that might be considered. One can set the lower bound of $\text{dom}(l)$ to $\min_{\boldsymbol{e} \in E} |\{i \mid \boldsymbol{e}_i \in M\}|$ and the upper bound of $\text{dom}(u)$ to $\max_{\boldsymbol{e} \in E} |\{i \mid \boldsymbol{e}_i \in C\}|$.

Lemma 3. *The bound on $G_{\text{AMONG}}(n, l, u, \boldsymbol{z})$ from (8) is tight when variables are instantiated.*

Proof. Since $\boldsymbol{z}$ are fixed, then $\sum_{v:z_v=1} p_v = \alpha = \beta = p$. Therefore, (8) is equal to (7) and the bound is tight. □

Other constraints Unlike with SUBSETFOCUS, SEQUENCE, and AMONG, the parameters for GCC, ATMOSTNVALUE, ATLEASTNVALUE, ATMOSTBALANCE, and ATMOSTBALANCE* are all monotonic and independent. This means that the branch-and-bound finds the optimal solution for these last constraints without backtracking. For the GCC for example, the optimal choice is always $\min(\text{dom}(l_i))$ when branching on a l_i parameter and $\max(\text{dom}(u_i))$ when branching on a u_i parameter by Corollary 5. The algorithm finds the optimal solution on the first try with the help of the filtering algorithm for the parameters.

4 Experiments

We test our branch-and-bound algorithm on two different benchmarks, one for SubsetFocus and one for Sequence, both solved on an Intel Core i7 3.40GHz with 4 Gb of RAM running Linux. All the code was written in Python, except for the brute force algorithm that learns the parameters of Sequence, for which Picard-Cantin et al. [29] provided their code written in R.

Since Among is a specific case of Sequence, we do not experiment on this constraint. Furthermore, we do not need to experiment on GCC, AtMostNValue, AtLeastNValue, AtMostBalance or AtMostBalance* since the solution is found without backtracks.

Random timetables were generated for the benchmark. A sequence is the schedule of an employee where each component of the sequence corresponds to a task for a given time slot. The task 0 is a special task that corresponds to a day off. This value is known not to belong to the considered set and therefore $z_0 = 0$ for both experiments: the one with SubsetFocus and the one with Sequence.

Note that the generated example have no particular structure as they are chosen uniformly among solutions that satisfy either SubsetFocus or Sequence with the given parameters and therefore they have maximal entropy. The examples are not real, but they are possibly not tight, meaning that they satisfy constraints stricter than the one imposed.

4.1 SubsetFocus

The benchmark for SubsetFocus is composed of 600 randomly generated instances. An instance is composed of a set of $d + 1$ values $\{0, \ldots, d\}$, a vector of assignment probability for the values $\boldsymbol{p} = [p_0, \ldots, p_d]$ ($p_v = P[X_i = v]$), a horizon n, and the set of parameters $(l, m, \boldsymbol{z})$. We generated instances with $d \in \{10, 30\}$, $l \in \{1, \ldots, 10\}$, $m \in \{1, \ldots, 10\}$, and $n \in \{100, 200, 300\}$. The vector $\boldsymbol{p}$ is generated such that $\sum_v p_v = 1$ and $\sum_{v:z_v=1} p_v \in \{0.2, 0.8\}$. Finally, for each instance defined by $(d, \boldsymbol{z}, \boldsymbol{p}, l, m, n)$, we generate ten (10) solutions that satisfy SubsetFocus with parameters $l, m, \boldsymbol{z}$ and whose task occurrences are proportional to their probabilities $\boldsymbol{p}$. The learning algorithms are given $\{0, \ldots, d\}$, $\boldsymbol{p}$, n, and a subset of the ten solutions (according to the experiment). Their goal is to learn the parameters $(l, m, \boldsymbol{z})$ that generated the solutions.

We compare the branch-and-bound algorithm described earlier with a brute force algorithm that lists all possible sets $\boldsymbol{z}$ (with $z_0 = 0$), that computes lower bounds on l and m from examples, that uses the monotonicity to fix l and m to their minimums, and that finally computes the probability of each resulting combination of parameters for SubsetFocus. This later technique is equivalent to the one proposed by Picard-Cantin et al. [29].

Figure 1 shows the resolution time to learn the parameters of SubsetFocus from a single solution. The branch-and-bound dominates the brute force by solving every instance in fewer than 73 seconds while the brute force sometimes reaches the 6-minute timeout. Figure 2 compares the number of times that the probability of the constraint for given parameters is computed using (3). That is

once per node for the branch-and-bound and once per parameter combination for the brute force. Once again, the branch-and-bound dominates the brute force. Figure 3 shows how many times, in percentages, the algorithms correctly predict the parameters given the number of examples that are provided. Since both algorithms return the same parameters when they solve an instance under the time limit, we only consider the branch-and-bound algorithm for this analysis. The low success rates are not alarming since the initial constraints were overly permissive in many cases. Therefore, the examples produced tend to satisfy more restrictive constraints that the algorithms detect.

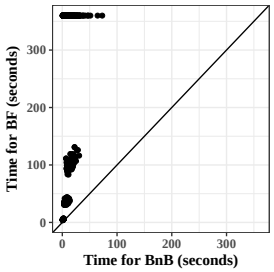

Fig. 1: Resolution time (log scale) for SubsetFocus

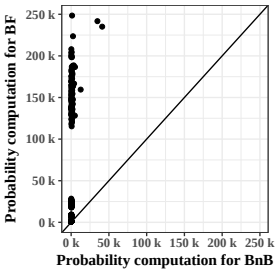

Fig. 2: Number of probability computations for SubsetFocus

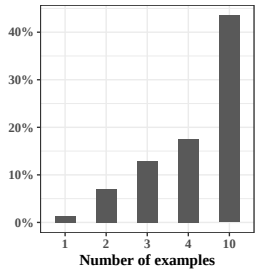

Fig. 3: Percentage of correctly learned instances for SubsetFocus

More details are given in Table 1 on the prediction quality of the learning algorithm according to the number of examples considered. For this comparison, we keep the three best predictions for each instance, meaning we keep the three sets of parameters for SubsetFocus satisfying the examples and with the lowest probability $G_{\text{SubsetFocus}}$. The rank of an instance is the place of the initial constraint in the prediction list. If the initial constraint was not in the best three sets of parameters returned by the algorithm, then we consider the rank to be ∞. The results of the first column are represented in Figure 3. From these results, we can say that if the learning algorithm has to give a short list of candidates from which the user would choose to add to the mathematical model, it would return the correct constraint in the first two choices 63% of the time, considering ten (10) examples. Moreover, the correct constraint seldom takes the third position.

4.2 Sequence

The benchmark for Sequence is composed of 84 randomly generated instances. An instance is composed of a set of values $\{0, \ldots, d\}$, a vector of assignment probability for the values $\boldsymbol{p} = [p_0, \ldots, p_d]$ ($p_v = P[X_i = v]$), a horizon n, and the set of parameters $(u, w, \boldsymbol{z})$. We fix $l = 0$ for all instances.

	Rank of initial constraint				
Num. of examples	1	2	3	∞	Num. of instances
1	8	46	1	545	600
2	42	119	0	439	600
3	78	148	0	374	600
4	105	172	0	323	600
5	139	170	0	291	600
10	261	117	0	222	600

Table 1: Results for SubsetFocus. Number of instances for which the initial constraint was ranked first, second, third or was not found.

We generated instances with $d \in \{10, 30\}$, $w \in \{5, 6, 7\}$, $u \in \{1 + 3k \mid 1 + 3k \leq w \wedge k \in \mathbb{N}\}$, and $n \in \{100, 200, 300\}$. The vectors $\boldsymbol{z}$ and $\boldsymbol{p}$ are generated such that $\sum_v p_v = 1$ and $\sum_{v:z_v=1} p_v \in \{0.2, 0.8\}$. Finally, for each instance defined by $(d, \boldsymbol{z}, p, u, w, n)$, we generate ten (10) solutions that satisfy Sequence with parameters $u, w, \boldsymbol{z}$ and whose task occurrences are proportional to their probabilities $\boldsymbol{p}$.

Figure 4 shows the computation times. We can separate the instances into two subsets according to the number of values $d \in \{10, 30\}$. When the number of values is small, the brute force algorithm is faster because it does not have to keep track of partial problems like the branch-and-bound does. When the number of values is large, the brute force algorithm has too many combinations to test and the branch-and-bound is faster. Therefore, the branch-and-bound algorithm for Sequence is more useful when the problem to solve is large. Figure 5 explains why the branch-and-bound is faster on larger instances as it shows that the number of probability computations is smaller for the branch-and-bound algorithm. Figure 6 shows how many times the algorithms correctly predict the parameters given the number of examples that are provided. As for SubsetFocus, we give more information about the prediction quality of the algorithm in Table 2. For Sequence, we observe that the correct constraint is returned by the learning algorithm among the first two choices 59.5% of the time when we consider ten examples.

	Rank of initial constraint				
Num. of examples	1	2	3	∞	Num. of instances
1	19	7	0	58	84
2	29	7	0	48	84
3	32	8	0	44	84
10	48	2	0	34	84

Table 2: Results for Sequence. Number of instances for which the initial constraint was ranked first, second, third or was not found.

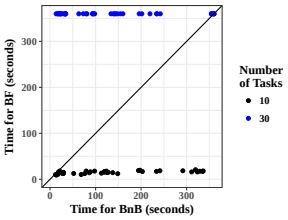

Fig. 4: Resolution time (log scale) for SEQUENCE

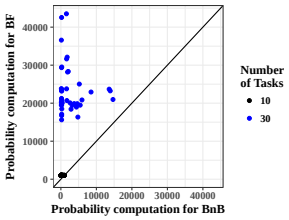

Fig. 5: Number of probability computations for SEQUENCE

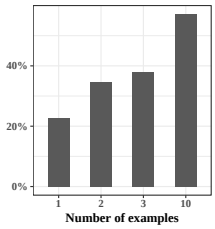

Fig. 6: Percentage of correctly learned instances for SEQUENCE

4.3 Discussion

The proposed branch-and-bound is more efficient than the state-of-the-art algorithms and it can be applied to multiple global constraints. We explained how to compute a lower bound on G_C for eight global constraints.

A drawback of the technique is that it does not detect overly permissive constraints with either algorithm, the branch-and-bound or the brute force. This is due to the fact that a permissive constraint has a lot of solutions and those solutions might satisfy more restrictive constraints. To avoid overfitting and therefore learning constraints that are too restrictive, one needs to increase the number of examples. In our experiments, we obtained satisfying results with only ten examples. We didn't test overly permissive constraints as we are certain not to find them as the most probable constraints and probably not even in the top 3. Having prediction errors on permissive constraints is not as important as missing tight constraints since the former have a smaller impact on the solutions.

5 Conclusion

We showed how a branch-and-bound can be used to learn the parameters of global constraints from positive examples. We showed how the monotonicity can be exploited to obtain tight bounds on the probability that a random assignment satisfies a constraint. Some constraints have for parameter a set of values $\boldsymbol{z}$. The set is used to limit the number of occurrences of its values. This parameter is not monotonic. Nevertheless, we presented a technique to bound the probability by counting the number of solutions that satisfy the constraint with a fixed number of values belonging to a set. Experiments show that the new technique is more time efficient than the state of the art algorithms, based on benchmarks inspired from timetabling problems.

References

1. Arcangioli, R., Lazaar, N.: Multiple constraint acquisition. In: Proc. of the 2015 International Conf. on Constraints and Preferences for Configuration and Recommendation and Intelligent Techniques for Web Personalization. CPCR+ITWP'15, vol. 1440, pp. 16–20. CEUR-WS.org (2015)
2. Barták, R., Čepek, O., Surynek, P.: Discovering implied constraints in precedence graphs with alternatives. Annals of Operations Research 180(1), 233–263 (2010)
3. Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Neuron constraints to model complex real-world problems. In: Princ. and Practice of Constraint Programming–CP 2011, pp. 115–129 (2011)
4. Beldiceanu, N., Contejean, E.: Introducing global constraints in chip. Mathematical and computer Modelling 20(12), 97–123 (1994)
5. Beldiceanu, N., Ifrim, G., Lenoir, A., Simonis, H.: Describing and generating solutions for the edf unit commitment problem with the modelseeker. In: International Conf. on Principles and Practice of Constraint Programming. pp. 733–748 (2013)
6. Beldiceanu, N., Simonis, H.: A constraint seeker: Finding and ranking global constraints from examples. In: Proc.of the 17th International Conf. of Principles and Practice of Constraint Programming (CP 2011). pp. 12–26 (2011)
7. Beldiceanu, N., Simonis, H.: Using the global constraint seeker for learning structured constraint models: a first attempt. In: The 10th Int. Workshop on Constraint Modelling and Reformulation (ModRef 2011), Perugia, Italy. pp. 20–34 (2011)
8. Beldiceanu, N., Simonis, H.: A model seeker: Extracting global constraint models from positive examples. In: Proc.of the 18th International Conf. of Principles and Practice of Constraint Programming (CP 2012). pp. 141–157 (2012)
9. Beldiceanu, N., Carlsson, M., Demassey, S., Petit, T.: Global constraint catalogue: Past, present and future. Constraints 12(1), 21–62 (2007)
10. Bessiere, C., Coletta, R., Daoudi, A., Lazaar, N., Mechqrane, Y., Bouyakhf, E.H.: Boosting constraint acquisition via generalization queries. In: ECAI. pp. 99–104 (2014)
11. Bessiere, C., Coletta, R., Freuder, E.C., O'Sullivan, B.: Leveraging the learning power of examples in automated constraint acquisition. In: Principles and Practice of Constraint Programming–CP 2004, pp. 123–137 (2004)
12. Bessiere, C., Coletta, R., Petit, T.: Acquiring parameters of implied global constraints. In: Principles and Practice of Constraint Programming-CP 2005, pp. 747–751 (2005)
13. Bessiere, C., Coletta, R., Petit, T.: Learning implied global constraints. In: IJCAI. pp. 44–49 (2007)
14. Bessiere, C., Daoudi, A., Hebrard, E., Katsirelos, G., Lazaar, N., Mechqrane, Y., Narodytska, N., Quimper, C.G., Walsh, T.: New approaches to constraint acquisition. In: Data Mining and Constraint Programming, pp. 51–76 (2016)
15. Bessiere, C., Koriche, F., Lazaar, N., O'Sullivan, B.: Constraint acquisition. Artificial Intelligence (In Press) (2015)
16. Bessiere, C., Hebrard, E., Katsirelos, G., Kiziltan, Z., Picard-Cantin, E., Quimper, C.G., Walsh, T.: The balance constraint family. In: Principles and Practice of Constraint Programming. pp. 174–189. Springer (2014)
17. Bonfietti, A., Lombardi, M., Milano, M.: Embedding decision trees and random forests in constraint programming. In: Integration of AI and OR Techniques in Constraint Programming, pp. 74–90 (2015)

18. Campigotto, P., Passerini, A., Battiti, R.: Active learning of combinatorial features for interactive optimization. In: International Conf. on Learning and Intelligent Optimization. pp. 336–350 (2011)
19. Charnley, J., Colton, S., Miguel, I.: Automatic generation of implied constraints. In: ECAI. vol. 141, pp. 73–77 (2006)
20. Daoudi, A., Lazaar, N., Mechqrane, Y., Bessiere, C., Bouyakhf, E.H.: Detecting types of variables for generalization in constraint acquisition. In: Tools with Artificial Intelligence (ICTAI), 2015 IEEE 27th International Conf. on. pp. 413–420. IEEE (2015)
21. Kiziltan, Z., Lippi, M., Torroni, P.: Constraint detection in natural language problem descriptions. In: Proc.of the twenty-fifth international joint conf. on artificial intelligence, IJCAI 2016, new york, usa, july 9–15, 2016 (2016)
22. Kolb, S.: Learning constraints and optimization criteria. In: Proc.of the First Workshop on Declarative Learning Based Programming (2016)
23. Little, J., Gebruers, C., Bridge, D.G., Freuder, E.C.: Using case-based reasoning to write constraint programs. In: CP. p. 983 (2003)
24. Lombardi, M., Milano, M., Bartolini, A.: Empirical decision model learning. Artificial Intelligence (2016)
25. Lopez, M., Lallouet, A.: On learning csp specifications. In: The 15th International Conf. on Principles and Practice of Constraint Programming Doctoral Program Proceedings. p. 70 (2009)
26. Pachet, F., Roy, P.: Automatic generation of music programs. In: International Conf. on Principles and Practice of Constraint Programming. pp. 331–345 (1999)
27. Pesant, G., Quimper, C.G., Zanarini, A.: Counting-based search: Branching heuristics for constraint satisfaction problems. Journal of Artificial Intelligence Research 43, 173–210 (2012)
28. Petit, T.: Focus: A constraint for concentrating high costs. In: Principles and Practice of Constraint Programming. pp. 577–592 (2012)
29. Picard-Cantin, É., Bouchard, M., Quimper, C.G., Sweeney, J.: Learning parameters for the sequence constraint from solutions. In: International Conf. on Principles and Practice of Constraint Programming. pp. 405–420 (2016)
30. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: Proc. of the 13th National Conf. on Artificial Intelligence - Volume 1 (AAAI 1996). pp. 209–215. AAAI Press (1996)
31. Suraweera, P., Webb, G.I., Evans, I., Wallace, M.: Learning crew scheduling constraints from historical schedules. Transportation research part C: emerging technologies 26, 214–232 (2013)