Acquiring Parameters of Implied Global Constraints

Christian Bessiere, Remi Coletta, Thomas Petit

Acquiring Parameters of Implied Global Constraints

Christian Bessiere[1], Rémi Coletta[1], and Thierry Petit[2]

[1] LIRMM (CNRS/University of Montpellier), 161 Rue Ada, 34392 Montpellier, France.
{bessiere,coletta}@lirmm.fr
[2] LINA (FRE CNRS 2729), École des Mines de Nantes, 4, Rue Alfred Kastler,
FR-44307 Nantes Cedex 3, France.
thierry.petit@emn.fr

Abstract. This paper presents a technique for *learning* parameterized implied constraints. They can be added to a model to improve the solving process. Experiments on implied Gcc constraints show the interest of our approach.

1 Introduction

Automatic model reformulation is a key issue for researchers [5, 4, 3]. The objective is to decrease the expertise required to use constraint programming. One way to improve a model consists of adding *implied* constraints. An implied constraint is not mandatory to express the problem but it helps to solve it [7]. In this paper, we assume that we have a model which expresses the problem, but the solving time is not satisfactory. We wish to improve it by automatically adding new implied constraints. Our idea is to use a *learning algorithm* that deduces new *parameterized* constraints from assignments of values to sets of variables. The set of tuples allowed by a parameterized constraint not only depends on its variables and their domains, but also on some extra information provided by *parameters*, which are not necessarily part of the problem variables. For instance, the NValue$(p, [X_1, \ldots, X_n])$ constraint holds iff p is equal to the number of different values taken by $X_1, \ldots, X_n$. p can be a CSP variable (in its more general definition) or a parameter which leads to different sets of allowed tuples on the X_i's, depending of the values it can take. In our context, a learning algorithm would try, for example, to learn the smallest range of possible values for p s.t. no solution exists outside this range. If the algorithm returns a lower bound $min(p) > 0$, or an upper bound $max(p) < n$, then the learned constraint can be of interest. [1]

2 Learning Parameterized Constraints

The set of tuples allowed by a classical constraint is known once the variables it involves and their domains are set. A parameterized constraint may allow different sets of tuples depending on the possible values for its parameters. Let us first define parameterized constraints in the most general way to let our work be as general as possible.

[1] A lot of existing constraints [1] may be used as implied parameterized constraints: AtLeast, AtMost, Change, Common, Count, Gcc, Max, Min, NValue, etc.

Definition 1. *Given a set of parameters $\Delta = \{p_1, ..., p_{|\Delta|}\}$ taking their values in the set of integers $\mathbb{Z}$, a* parameterized constraint *C is a constraint which expresses a property on the variables it involves (denoted by $var(C)$) depending on the possible values for its set of parameters Δ. Given $s \in \mathbb{Z}^{|\Delta|}$, an assignment of values to the parameters, $C(s)$ refers to the set of allowed tuples of the constraint C when each p_i in Δ takes the i^{th} value in s, noted $s[p_i]$. Given $S \subseteq \mathbb{Z}^{|\Delta|}$, $C(S) = \bigvee_{s \in S} C(s)$.*

Definition 2. *Given a constraint C with parameters Δ and a tuple e on $var(C)$ (or a superset of $var(C)$), S_e contains the combinations s in $\mathbb{Z}^{|\Delta|}$ s.t. $C(s)$ accepts e.*

Our goal is to learn implied constraints on any type of constraint problem: we focus on those parameterized constraints where for any tuple e on $var(C)$, $S_e \neq \emptyset$. The basic idea is to learn the parameters of an implied constraint C by exploiting information provided by solutions and non-solutions of a subproblem (stem from the initial problem by removing variables and constraints). Indeed, if the current model is not good, obviously it should not easily provide instances[2] for the whole problem. By definition, if a learned constraint is valid in a subproblem then it is still valid in the main problem. Variables of the subproblem are then the decision variables involved in the learned implied constraint. W.r.t. optimization problems, in the context of a Branch and Bound algorithm, a way to proceed is to learn new implied constraints at each step of the optimization.[3]

We assume now that the subproblem and the constraint C to learn have been chosen.

Notation 1 *$\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the problem used to learn an implied constraint.*

Definition 3. *An* implied constraint *C for $\mathcal{P}$ is a constraint s.t. $var(C) \subseteq \mathcal{X}$ and the set $Sol(\mathcal{P})$ of solutions of $\mathcal{P}$ is equal to $Sol((\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{C\}))$; Namely, for any instance $e \in Sol(\mathcal{P}), e[var(C)]$ is allowed by C.*

We consider only constraints for which $C(\mathbb{Z}^{|\Delta|})$ is the universal constraint (i.e., for any tuple e, $S_e \neq \emptyset$). Thus, following Definition 3, $C(\mathbb{Z}^{|\Delta|})$ is an implied constraint for the problem $\mathcal{P}$. The objective of a learning algorithm will be to learn a 'target' set $T \subseteq \mathbb{Z}^{|\Delta|}$, as small as possible, s.t. $C(T)$ is still an implied constraint. Any $s \in \mathbb{Z}^{|\Delta|}$ which is not necessary to accept some solutions of $\mathcal{P}$ can be removed from T.

Notation 2 *Given a problem $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and a parameterized constraint C, we denote by $required(T)$ the set of elements r of $\mathbb{Z}^{|\Delta|}$ such that $r \in required(T)$ iff $C(\mathbb{Z}^{|\Delta|} \setminus \{r\})$ is not an implied constraint, namely r is compulsory in T if we want $C(T)$ to be an implied constraint in $\mathcal{P}$. $poss(T)$ denotes those r in $\mathbb{Z}^{|\Delta|}$ for which we do not know if they must belong to $required(T)$.*

In other words, $required(T)$ represents those combinations of values for the parameters that are necessary to preserve the set of solutions of $\mathcal{P}$, and $poss(T)$ those for which we have not proved yet that we would not lose solutions without them. The fact that an implied constraint should not remove solutions when we add it to the problem leads to the following property w.r.t. positive instances.

[2] Given a problem $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, an *instance* e is an assignment of values to variables in $\mathcal{X}$. It is *positive* if e belongs to $Sol(\mathcal{P})$, otherwise it is *negative*.

[3] We may observe a nice cooperation between the learning and the solving phases: new learned constraints help at each step to solve the problem, and thus to learn the next ones.

Property 1. Let e^+ be a positive instance and $T \subseteq \mathbb{Z}^{|\Delta|}$. If $C(T)$ is an implied constraint for $\mathcal{P}$ then $T \cap S_{e^+} \neq \emptyset$.

Proof. e^+ is a solution of $\mathcal{P}$. At least one $s \in T$ should accept that instance to preserve the set of solutions of $\mathcal{P}$. So, $s \in S_{e^+}$ by Definition 2. □

Corollary 1. *Let e^+ be a positive instance of $\mathcal{P}$ and C a parameterized constraint. If there is a unique $s \in \mathbb{Z}^{|\Delta|}$ such that $e^+[var(C)] \in C(s)$ then $s \in required(T)$.*

Negative instances will help to remove from $poss(T)$ the combinations of parameters that can be removed without losing solutions. When receiving a negative instance e^-, we want to know if it is possible to reduce the target set T in order to reject e^- while preserving $Sol(\mathcal{P})$.

Property 2. Let $\mathcal{P}$ be a problem, C a parameterized constraint, and $S \subset \mathbb{Z}^{|\Delta|}$. If $\mathcal{P}$ augmented with the constraint $C(S)$ has no solution then $C(\mathbb{Z}^{|\Delta|} \setminus S)$ is an implied constraint for $\mathcal{P}$.

Proof. By previous assumption, we know that $C(\mathbb{Z}^{|\Delta|})$ is an implied constraint for $\mathcal{P}$. Since $\mathcal{P} \cup \{C(S)\}$ is inconsistent, we know that $\forall e \in Sol(\mathcal{P})$, $S_e \cap S = \emptyset$. Thus $Sol(\mathcal{P} + C(\mathbb{Z}^{|\Delta|})) = Sol(\mathcal{P} + C(\mathbb{Z}^{|\Delta|} \setminus S))$. □

Corollary 2. *Let $\mathcal{P}$ be a problem, C a parameterized constraint, $S \subseteq \mathbb{Z}^{|\Delta|}$, and $poss(T)$ a set s.t. $C(poss(T))$ is implied on $\mathcal{P}$. If $\mathcal{P}$ augmented with the constraint $C(S)$ has no solution then $C(poss(T) \setminus S)$ is an implied constraint for $\mathcal{P}$.*

Algorithm 1: Learning Algorithm for Parameterized Constraints.

```
   Input: C, Δ, E = {e_1, ..., e_k} a set of instances for P.
   Output: poss(T), s.t. C(poss(T)) is an implied constraint for P.
   required(T) ← ∅; poss(T) ← Z^|Δ|;
   while (E ≠ ∅) and (required(T) ≠ poss(T)) do
       Pick e ∈ E;
       if e is positive then
1          if |S_e| = 1 then required(T) ← required(T) ∪ S_e; /* Corol. 1*/
       else
           for some S ⊆ (poss(T) \ required(T)) do
2              if Sol(X, D, C ∪ C(S)) = ∅ then poss(T) ← poss(T) \ S; /* Corol. 2*/
               else put an element of Sol(X, D, C ∪ C(S)) as positive instance in E
```

Checking if $\mathcal{P}$ augmented with the constraint $C(S)$ is inconsistent (line 2) is obviously NP-hard. Even if $\mathcal{P}$ in this learning phase is not supposed to be the whole problem we want to reformulate, it is necessary to follow some heuristics to avoid huge numbers of NP-hard calls to a solver. For a parameterized constraint C, different representations of parameters may exist. The more general case studied until now (Algorithm 1) consists of considering that allowed tuples of parameters for C are given in extension as a set T. Corollary 1 may then seem weak. However, in practice, most of parameterized

constraints are s.t. any two different combinations of parameters in $\mathbb{Z}^{|\Delta|}$ correspond to disjoint sets of allowed tuples on $var(C)$. We call them *parameter-partitioned* constraints. For instance, $\texttt{NValue}(p, [X_1, \ldots, X_n])$ is parameter-partitioned since a single value of p corresponds to an assignment of the X_i.

Corollary 3. *Let e^+ be a positive instance. If C is a parameter-partitioned constraint, $s_{e^+} \in required(T)$, where s_{e^+} is the only element in S_{e^+}.*

This corollary allows a faster construction of $required(T)$ compared with Algorithm 1. Moreover, existing parameterized constraints are usually defined by sets of possible values for their parameters taken separately, which is less expressive than considering directly any subset of $\mathbb{Z}^{|\Delta|}$. It is possible to exploit this fact. We will note $T[p_i]$ for the values of a parameter we wish to learn. Then, $required(T[p_i]) = \{k_i \in \mathbb{Z} \text{ s.t. } \exists s \in required(T), s[p_i] = k_i\}$, namely the set of values for p_i that are required in $T[p_i]$. Similarly, $poss(T[p_i]) = \{k_i \in \mathbb{Z} \text{ s.t. } \exists s \in poss(T), s[p_i] = k_i\}$. If the possible values for a parameter p_i are a **set of integers**, the learning algorithm uses sets of integers to represent $required(T[p_i])$ and $poss(T[p_i])$. The parameterized constraint will be called with $poss(T[p_1]) \times \ldots \times poss(T[p_{|\Delta|}])$. If the possible values for p_i are a **range of integers**, the learning algorithm uses ranges of the form $[min(poss(T[p_i]))..max(poss(T[p_i]))]$ to represent $poss(T[p_i])$ (resp. $required(T[p_i])$). Properties 1 and 2 can be rewritten to fit these two cases.

Property 3. Let e^+ be a positive instance of $\mathcal{P}$ and C a parameterized constraint s.t. parameters are *sets of integers*. If there is a unique $s \in \mathbb{Z}^{|\Delta|}$ such that $e^+[var(C)] \in C(s)$ then for any $p_i \in \Delta$, $s[p_i] \in required(T[p_i])$.

Property 4. Let $\mathcal{P}$ be a problem, C a parameterized constraint where parameters are a *set of integers*, and $poss(T)$ a set s.t. $C(poss(T))$ is implied on $\mathcal{P}$. Let p be a parameter of C, v a value in $poss(T[p])$, and $S = \{s \in poss(T) \text{ s.t. } s[p] = v\}$. If $\mathcal{P}$ augmented with $C(S)$ has no solution then v can be removed from $poss(T[p])$.

Note that for each negative instance e^-, if the constraint is parameter-partitioned, a heuristic can be used to apply property 4 with $v = s_{e^-}[p]$.

Property 5. Let e^+ be a positive instance of $\mathcal{P}$ and C a parameterized constraint s.t. parameters are *ranges*. If there is a unique $s \in \mathbb{Z}^{|\Delta|}$ such that $e^+[var(C)] \in C(s)$ then for any $p_i \in \Delta$, $min(required(T[p_i])) \leq s[p_i] \leq max(required(T[p_i]))$.

Property 6. Let $\mathcal{P}$ be a problem, C a parameterized constraint where parameters are *ranges*, and $poss(T)$ a set s.t. $C(poss(T))$ is implied on $\mathcal{P}$. Let p be a parameter of C, v a value in $poss(T[p])$, and $S = \{s \in poss(T) \text{ s.t. } s[p] \leq v\}$ (respectively: $S = \{s \in poss(T) \text{ s.t. } s[p] \geq v\}$). If $\mathcal{P}$ augmented with $C(S)$ has no solution then $min(poss(T[p])) > v$ (respectively: $max(poss(T[p_i])) < v$).

3 Experiments

We implemented with Choco [2] a learning algorithm for implied $\texttt{Gcc}$ (global cardinality constraints [6]) where parameters are ranges. On the two intentionally naive models

we implemented to evaluate the interest of our algorithm, tables compare the solving time of an initial model with the same model augmented with a learned Gcc.[4]

m_1/m_2	$maxi$	#nodes	time (sec.)
4/0	4	—	> 60
4/0	3	42,129	7.4
4/0	2	85	0.12
3/1	4	45	0.030
3/1	3	33	0.041
3/1	2	7 (no sol)	0.030

(a) Initial model.

m_1/m_2	$maxi$	#nodes	time (sec.)
4/0	4	47	0.297 + 0.026
4/0	3	47	0.279 + 0.018
4/0	2	23	0.265 + 0.015
3/1	4	43	0.312 + 0.031
3/1	3	30	0.286 + 0.019
3/1	2	3 (no sol)	0.279 + 0.013

(b) Augm. Model.

Table 1. Scheduling satisfaction problem with a fixed makespan, precedence constraints, and $m = m_1+m_2$ tasks (requiring one or two resources). $maxi$ is the maximum allowed resource. 15 assignments where used to learn an implied Gcc on a problem relaxed from resource constraints.

n	#nodes	time (sec.)
10	110	0.2
15	2,648	7.6
20	137,982	183.2

(a) Initial Model.

n	$\|E\|$	#nodes	time (sec.)
10	10	12	0.2 + 0.0
15	10	57	2.6 + 1.1
15	20	50	3.8 + 1.1
15	40	50	5.9 + 0.9
20	10	4,801	5.7 + 14.9
20	20	2,800	6.8 + 13.0
20	40	1,998	9.1 + 8.4

(b) Augm. Model.

Table 2. Optimization problem (allocation). n is the problem size (number of variables), $|E|$ is the number of assignments used to learn implied Gcc's.

References

1. N. Beldiceanu. Global constraints as graph properties on a structured network of elementary constraints of the same type. *Proceedings CP*, pages 52–66, 2000.
2. Choco. A Java library for constraint satisfaction problems, constraint programming and explanation-based constraint solving. *URL: http://sourceforge.net/projects/choco*, 2005.
3. S. Colton and I. Miguel. Constraint generation via automated theory formation. *Proceedings CP*, pages 575–579, 2001.
4. A. M. Frisch, C. Jefferson, B. Martinez Hernandez, and Ian Miguel. The rules of constraint modelling. *Proceedings IJCAI*, to appear, 2005.
5. J.F. Puget. Constraint programming next challenge: Simplicity of use. In *Proceedings CP*, pages 5–8, Toronto, Canada, 2004.
6. J-C. Régin. Generalized arc consistency for global cardinality constraint. *Proceedings AAAI*, pages 209–215, 1996.
7. B. Smith, K. Stergiou, and T. Walsh. Modelling the golomb ruler problem. In J.C. Régin and W. Nuijten, editors, *Proceedings IJCAI'99 workshop on non-binary constraints*, Stockholm, Sweden, 1999.

[4] In the tables located at the left the last column indicates the solving time, whereas in the tables located at the right the last column indicates the sum of the learning time and the solving time.