Learning Implied Global Constraints

Christian Bessiere
LIRMM-CNRS
U. Montpellier, France
bessiere@lirmm.fr

Remi Coletta
LRD
Montpellier, France
coletta@l-rd.fr

Thierry Petit
LINA-CNRS
École des Mines de Nantes, France
thierry.petit@emn.fr

Abstract

Finding a constraint network that will be efficiently solved by a constraint solver requires a strong expertise in Constraint Programming. Hence, there is an increasing interest in automatic reformulation. This paper presents a general framework for learning implied *global* constraints in a constraint network assumed to be provided by a non-expert user. The learned global constraints can then be added to the network to improve the solving process. We apply our technique to global cardinality constraints. Experiments show the significance of the approach.

1 Introduction

Encoding a problem as a constraint network (called *model*) that will be solved efficiently by a constraint toolkit is a difficult task. This requires a strong expertise in Constraint Programming (CP), and this is known as one of the main bottlenecks to the widespread of CP technology [Puget, 2004]. CP experts often start by building a first model that expresses correctly the problem without any other requirement. Then, if the time to solve this model is too high, they refine it by adding new constraints that do not change the set of solutions but that increase the propagation capabilities of the solver. In Ilog Solver 5.0. user's manual, a `Gcc` constraint is added to improve the solving time of a graph coloring problem (page 279 in [Ilog, 2000]). In [Régin, 2001], new constraints are added in a sports league scheduling problem. Such constraints are called *implied* constraints [Smith *et al.*, 1999]. The community has studied automatic ways of generating implied constraints from a model. Colton and Miguel [Colton and Miguel, 2001] exploit a theory formation program to provide concepts and theorems which can be translated into implied constraints. In Hnich *et al.* [Hnich *et al.*, 2001], some implied linear constraints are derived by using an extension of the Prolog Equation Solving System [Sterling *et al.*, 1989]. In all these frameworks the new generated constraints are derived from structural properties of the original constraints. They do not depend on domains of variables.

This paper presents a framework based on ideas presented in [Bessiere *et al.*, 2005]. This framework contains two original contributions in automatic generation of implied constraints: First, it permits to learn global constraints with parameters; Second, the implied constraints are learned according to the actual domains of the variables in the model.

Learning global constraints is important because global constraints are a key feature of constraint programming. They permit to reduce the search space with efficient propagators. Global constraints generally involve parameters. For instance, consider the `NValue`$(k, [x_1, \ldots, x_n])$ constraint. It holds iff k is equal to the number of different values taken by $x_1, \ldots, x_n$. k can be seen as the parameter, which leads to different sets of allowed tuples on the x_i's depending on the value it takes. The increase of expressiveness provided by the parameters increases the chances to find implied constraints in a network. A learning algorithm will try, for example, to learn the smallest range of possible values for k s.t. the `NValue` is an implied constraint. The tighter the range for k, the more the learned constraint can reduce the search space. Many constraints from [Beldiceanu *et al.*, 2005] may be used as implied constraints involving parameters.

Learning implied constraints according to the actual domains is important because constraints learned that way take into account more than just the structure of the network and the constraints. They are closer to the real problem to be solved. For instance, if we know that X, Y and Z are 0/1 variables, we can derive $X = Z$ from the set of constraints $X \neq Y, Y \neq Z$. Without knowledge on the domains, we cannot derive anything. Even an expert in CP has difficulties to find such constraints because they depend on complex interactions between domains and constraints.

2 Motivation Example

Let us focus on a simple problem where n tasks have to be scheduled. All tasks have the same duration d. Each task i requires a given amount m_i of resource. Steps of time are represented by integers starting at 0. At any time, the amount of resource used by all tasks is bounded by $maxR$. The starting times of any two tasks must be separated by at least two steps of time. Random precedence constraints between start times of tasks are added. The makespan (maximum time) is equal to $2 * (n-1) + d$ (the best possible makespan w.r.t. the constraint on starting times). The question is to find a feasible schedule. We implemented a naive model with Choco [Choco, 2005]. The start time of task i is represented by a variable x_i with domain $[0..2 * (n-1)]$ (because of the makespan), and the resource capacity is represented at each unit of time t by an

	Naive model	Implied constraint
n / d / n_1 / n_2 / $maxR$	#nodes / time (sec.)	#nodes / time (sec.)
4 / 4 / 4 / 0 / 4	—— / > 60	46 / 0.05
4 / 4 / 4 / 0 / 3	42,129 / 7.50	46 / 0.03
4 / 4 / 4 / 0 / 2	85 / 0.20	25 / 0.05
4 / 4 / 3 / 1 / 4	45 / 0.05	45 / 0.05
4 / 4 / 3 / 1 / 3	33 / 0.05	32 / 0.05
4 / 4 / 3 / 1 / 2	7 (unsat) / 0.05	3 (unsat) / 0.03

Table 1: Results on the naive model (left) and when an implied Gcc is added to the model (right). (n tasks of duration d among which n_j tasks use j resources.)

array $R[t][1..n]$ of variables, where $R[t][i] = m_i$ if task i is active at time t, $R[t][i] = 0$ otherwise. Constraints ensuring consistency between x_i and $R[t][i]$ and constraints on separation of starting times are primitive ones. Sum constraints are used for expressing resource capacity ($\sum_{i=1}^{n} R[t][i] \leq maxR$). We place us in the context of a non expert user, so we run the default search heuristics of Choco (the variable with minimum domain first, assigned with its smallest available value). We put a cutoff at 60 seconds. Table 1(left) reports results on instances where $n = 4, d = 4$ and n_j is the number of tasks requiring j resources. Only two precedence constraints are added. It shows that even on these small instances, our naive model can be hard to solve with standard solving techniques. If we want to solve this problem with more tasks, we definitely need to improve the model. An expert-user will see that constraints on the starting time of tasks induce limits on the number of tasks that can start at each point of time.[1] The makespan is indeed equal to the minimal possible value for scheduling all the tasks if they are separated by at least two units. We therefore have to place tasks as soon as possible. This means that one task starts at time zero, one task at time 2, etc. This can be expressed explicitly with a global cardinality constraint (Gcc). Gcc($[x_1, \ldots, x_n], [p_{v_1}, \ldots, p_{v_k}]$) holds iff $|\{i \mid x_i = v_j\}| = p_{v_j}$ for all j. So, we add to our model a Gcc constraint that involves all starting times $x_1, \ldots, x_n$ and guarantees that the n first even values of time are taken at least once: Gcc($[x_1, \ldots, x_n], [p_0, \ldots, p_{2n-1}]$), where $p_v \in 1 \ldots n$ if v is even, $p_v \in 0 \ldots n$ otherwise. As shown in Table 1(right), adding this simple Gcc constraint to the naive model leads to significant improvements. This example shows that for a non-expert user, even very small problems can lead to bad performance of the solver. Adding simple implied constraints to the model can dramatically reduce the solving cost.

3 Basic Definitions

A *constraint network* $\mathcal{N}$ is defined by a set $\mathcal{X} = \{x_1, \ldots, x_n\}$ of variables, a set $\mathcal{D} = \{D(x_1), \ldots, D(x_n)\}$ of domains of values for the variables in $\mathcal{X}$, and a set $\mathcal{C}$ of constraints. A *constraint* c is defined on its scope $X(c) \subseteq \mathcal{X}$. c specifies which combinations of values (or tuples) are allowed on the variables in $X(c)$. A solution to $\mathcal{N}$ is an assignment of values from their domain to all variables in $\mathcal{X}$ s.t. all constraints in $\mathcal{C}$ are satisfied. $sol(\mathcal{N})$ denotes the set of solutions of $\mathcal{N}$.

[1]The automatic generation of robust search heuristics is out of the scope of this paper.

An *implied* constraint for a constraint network $\mathcal{N}$ is a constraint that does not change the solutions of $\mathcal{N}$. That is, given the constraint network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, the constraint c with $X(c) \subseteq \mathcal{X}$ is an implied constraint for $\mathcal{N}$ if $sol(\mathcal{N}) = sol(\mathcal{N} + c)$ where $\mathcal{N} + c$ denotes the network $(\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{c\})$.

Global constraints are defined on any number of variables. For instance, alldifferent is a global constraint that can be instantiated on scopes of variables of any size. For most of the global constraints, the signature is not completely fixed. Some of their variables can be seen as parameters, that is, 'external' variables that are not involved in any other constraint of the problem. For instance, the Gcc constraint Gcc($[x_1, \ldots, x_n], [p_{v_1}, \ldots, p_{v_k}]$) involves x_i's and p_v's. The p_v's can be variables of the problem (like in [Quimper *et al.*, 2003]), or parameters that take a value in a range (like in [Régin, 1996]). Another example is the Among constraint. Among($[x_1, \ldots, x_n], [v_1, \ldots, v_k], N$) holds iff $|\{i \mid \exists j \in 1..k, x_i = v_j\}| = N$. The v_j's and N can be parameters or variables [Beldiceanu and Contejean, 1994; Bessiere *et al.*, 2006]. So, a global constraint C defined on a scheme $scheme(C)$ can be specified into different types of parametric constraints depending on which elements in $scheme(C)$ are variables or parameters.

Definition 1 (Parametric constraint) *Given a global constraint C defined on $scheme(C)$, a* parametric *constraint derived from C is a global constraint $C[P]$ s.t. $P = \{p_1, \ldots, p_m\}$ is the set of parameters of $C[P]$ with $P \subset scheme(C)$, and $X(C[P])$ is the scope of $C[P]$ with $X(C[P]) = scheme(C) \setminus P$.*

So, in addition to its variables, a constraint can be defined by a set P of parameters. The set of tuples that are allowed by the constraint depends on the values the parameters can take.

Definition 2 (Instance of parametric constraint) *Given a constraint network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, an instance of a parametric constraint $C[P]$ in $\mathcal{N}$ is a constraint $C[P \leftarrow S]$ s.t.:*

- *$X(C[P \leftarrow S]) = X(C[P]) \subseteq \mathcal{X}$,*
- *$P \cap \mathcal{X} = \emptyset$,*
- *$S \subseteq \mathbb{Z}^{|P|}$ is a set of tuples on P,*
- *$C[P \leftarrow S]$ is satisfied by an instantiation e on $X(C[P])$ iff there exists $t \in S$ s.t. the tuple $(e + t)$ on $scheme(C)$ is a satisfying tuple for the global constraint C from which $C[P]$ is derived.*

Example 1 *Let $C[P]$ be the parametric constraint derived from Among($[x_1, \ldots, x_4], [v], N$) with scope $(x_1, \ldots, x_4)$ and parameters $P = (v, N)$. Let $S = \{(1, 3), (2, 1)\}$ be a set of combinations for P. A tuple on $(x_1, \ldots, x_4)$ is accepted by the instance of constraint $C[P \leftarrow S]$ iff it has three occurrences of value 1 or one occurrence of value 2. Tuples $(1, 1, 3, 1)$ and $(1, 1, 2, 3)$ satisfy the constraint. Tuples $(3, 2, 2, 2)$ and $(2, 1, 1, 2)$ do not. We suppose that this parametric constraint allows us specifying* combinations *of parameters that are allowed. In practice, propagation algorithms often put restrictions on how parameters can be expressed.*

Given a parametric constraint $C[P]$, different sets S and S' of tuples for the parameters lead to different constraints $C[P \leftarrow S]$ and $C[P \leftarrow S']$. Hence, when adding an instance of parametric constraint $C[P]$ as implied constraint in a network $\mathcal{N}$, it is worth using a set S of tuples for the parameters which is as tight as possible and $C[P \leftarrow S]$ is implied in $\mathcal{N}$.

4 Learning Implied Parametric Constraints

The objective is to learn a 'target' set $T \subseteq \mathbb{Z}^{|P|}$, as small as possible s.t. $C[P]$ is still an implied constraint. Any $s \in \mathbb{Z}^{|P|}$ which is not necessary to accept some solutions of $\mathcal{N}$ can be discarded from T. The tighter the learned constraint is, the more promising its filtering power is.

Definition 3 (Learning an implied parametric constraint) *Given a constraint network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and a parametric constraint $C[P]$ with $P \cap \mathcal{X} = \emptyset$ and $X(C[P]) \subseteq \mathcal{X}$,* learning an instance of the parametric constraint $C[P]$ *implied in $\mathcal{N}$ consists in finding a set $S \subseteq \mathbb{Z}^{|P|}$ as tight as possible s.t. $C[P \leftarrow S]$ is implied in $\mathcal{N}$.*

A set of tuples T s.t. $C[P \leftarrow T]$ is implied in $\mathcal{N}$ and there does not exist any $S \subset T$ with $C[P \leftarrow S]$ implied in $\mathcal{N}$ is called a target *set for $C[P]$ in $\mathcal{N}$.*

We now give a general process to learn the parameters of implied global constraints. Since our goal is to deal with any type of constraint network, we focus on global constraints C and sets of parameters $P \subseteq scheme(C)$ s.t. $C[P \leftarrow \mathbb{Z}^{|P|}]$ is the universal constraint. This means that if $\mathcal{N}$ contains the variables in $scheme(C) \setminus P$, $C[P \leftarrow \mathbb{Z}^{|P|}]$ is an implied constraint in $\mathcal{N}$. The learning task is to find a small superset of a target set T (see Definition 3). Given a problem $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and a parametric constraint $C[P]$, we denote by:

- ub_T a subset of $\mathbb{Z}^{|P|}$ s.t. $C[P \leftarrow ub_T]$ is an implied constraint,
- lb_T a subset of ub_T s.t. for any $t \in lb_T$, $C[P \leftarrow ub_T \setminus \{t\}]$ is not an implied constraint.

In other words, lb_T represents those combinations of values for the parameters that are necessary in ub_T to preserve the set of solutions of $\mathcal{N}$, and ub_T those for which we have not proved yet that we would not lose solutions without them. The following propositions give some general conditions on how to incrementally tighten the bounds lb_T and ub_T while keeping the invariant $lb_T \subseteq T \subseteq ub_T$, where T is a target.

Proposition 1 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a network $\mathcal{N}$. If there exists $S \subset ub_T$ s.t. $sol(\mathcal{N}) = sol(\mathcal{N} + C[P \leftarrow S])$ then $C[P \leftarrow S]$ is an implied constraint in $\mathcal{N}$. So ub_T can be replaced by S.*

Testing equivalence of the sets of solutions of two networks is coNP-complete. This cannot be handled by classical constraint solvers. Hence, we have to relax the condition to a condition that can more easily be checked by a solver.

Corollary 1 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a network $\mathcal{N}$. If there exists $S \subseteq ub_T$ s.t. $\mathcal{N} + C[P \leftarrow S]$ is inconsistent then $C[P \leftarrow ub_T \setminus S]$ is an implied constraint in $\mathcal{N}$. So ub_T can be replaced by $ub_T \setminus S$.*

The weakness of this corollary is that we have no clue on which subsets S of ub_T to test for inconsistency. But we know that an implied constraint should not remove solutions when we add it to the network. Hence, solutions can help us.

Proposition 2 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a network $\mathcal{N}$ and e be a solution of $\mathcal{N}$. For any $S \subseteq ub_T$ s.t. $C[P \leftarrow S]$ is an implied constraint for $\mathcal{N}$ there exists t in S s.t. e satisfies $C[P \leftarrow \{t\}]$.*

If a given solution is accepted by a unique combination of values for the parameters, then this combination is necessary for having an implied constraint (i.e., it belongs to lb_T).

Corollary 2 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a network $\mathcal{N}$ and e be a solution of $\mathcal{N}$. If there is a unique t in ub_T s.t. e satisfies $C[P \leftarrow \{t\}]$ then t can be put in lb_T.*

By Corollaries 1 and 2 we can shrink the bounds of an initial set of possibilities for the parameters. Algorithm 1 performs a brute-force computation of a set ub_T s.t. $C[P \leftarrow ub_T]$ is an implied constraint for a network $\mathcal{N}$. This algorithm works when no extra-information is known from the parameters. It uses corollaries 1 and 2. The input is a parametric constraint and a constraint network (and optionally an initial upper bound UB on the target set). The output is a set ub_T of tuples for the parameters, s.t. no solution of the problem is lost if we add $C[P \leftarrow ub_T]$ to the network.

Algorithm 1: Brute-force Learning Algorithm

```
  Input: $C[P], \mathcal{N}$ (optionally $UB$);
  Output: $ub_T$;
  begin
     $lb_T \leftarrow \emptyset$;
     $ub_T \leftarrow \mathbb{Z}^{|P|}$ ;                /* or the tighter $UB$ */
     while $lb_T \neq ub_T$ and not 'time-out' do
        Choose $S \subset ub_T \setminus lb_T$;
1       if $sol(\mathcal{N} + C[P \leftarrow S]) = \emptyset$ then
           $ub_T \leftarrow ub_T \setminus S$ ;          /* Corollary 1 */
        else
           pick $e$ in $sol(\mathcal{N} + C[P \leftarrow S])$;
2          if there is a unique $t$ in $ub_T$ s.t. $e$ satisfies
           $C[P \leftarrow \{t\}]$ then
              $lb_T \leftarrow lb_T \cup \{t\}$ ;       /* Corollary 2 */
  end
```

It is highly predictable that Algorithm 1 can be inefficient. The space of possible combinations of values to explore is exponential in the number of parameters. In addition, checking whether $\mathcal{N} + C[P \leftarrow S]$ is inconsistent (line 1) is NP-complete. Even if $\mathcal{N}$ in this learning phase should be only a subpart of the whole problem, it is necessary to use some heuristics to improve the process. Fortunately, in practice, constraints have characteristics that can be used. The next section studies some of these properties.

5 Using Properties on the Constraints

For a given parametric constraint $C[P]$, different representations of parameters may exist. The complexity of associated filtering algorithms generally differs according to the representation used. The more general case studied in Section 4 consists in considering that allowed tuples of parameters for $C[P]$ are given in extension as a set $S \subseteq \mathbb{Z}^{|P|}$.

5.1 Partitioning parameters

In practice, many parametric constraints satisfy the following property: any two different combinations of values in $\mathbb{Z}^{|P|}$ for the parameters correspond to disjoint sets of solutions on $X(C[P])$. For instance, this is the case for the $\mathtt{Gcc}([x_1, \ldots, x_n], [p_{v_1}, \ldots, p_{v_k}])$ constraint: Each combination of values for the parameters $p_{v_1}, \ldots, p_{v_k}$ imposes a fixed number of occurrences for each value $v_1, \ldots, v_k$ in $x_1, \ldots, x_n$.

Definition 4 *A parametric constraint $C[P]$ is called* parameter-partitioning *iff for any t, t' in $\mathbb{Z}^{|P|}$, if $t \neq t'$ then $C[P \leftarrow \{t\}] \cap C[P \leftarrow \{t'\}] = \emptyset$.*

Given an instantiation e on $X(C[P])$, the unique tuple t on P s.t. e satisfies $C[P \leftarrow \{t\}]$ is denoted by t_e.

When a constraint is parameter-partitioning, we have the nice property that there is a unique target set.

Lemma 1 *Given a constraint network $\mathcal{N}$, if a parametric constraint $C[P]$ is parameter-partitioning then there exists a unique target set T for $C[P]$ in $\mathcal{N}$.*

The next proposition tells us that when a constraint is parameter-partitioning, updating the lower bound on the target set is easier than in the general case.

Proposition 3 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a network $\mathcal{N}$ and e be a solution of $\mathcal{N}$. If $C[P]$ is a parameter-partitioning constraint then the unique tuple t_e on P s.t. e satisfies $C[P \leftarrow \{t\}]$ can be put in lb_T.*

This Proposition is a first way to improve Algorithm 1. It allows a faster construction of lb_T because any solution of $\mathcal{N}$ contributes to the lower bound lb_T in line 2.

5.2 Parameters as sets of integers

As far as we know, existing parametric constraints are defined by sets of possible values for their parameters $p_1, \ldots, p_m$ taken separately. This means that the target set T must be a Cartesian product $T(p_1) \times \cdots \times T(p_m)$, where $T(p_i)$ is the target set of values for parameter p_i. In other words, T is derived from the sets of possible values of *each parameter*. This is less expressive than directly considering any subset of $\mathbb{Z}^{|P|}$, but the learning process is easier to handle. Let $lb_T(p_i)$ and $ub_T(p_i)$ be the required and possible values in $T(p_i)$. Corollaries 1 and 2 are specialized in Corollaries 3 and 4.

Corollary 3 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a constraint network $\mathcal{N}$, with $ub_T = \bigtimes_{p_i \in P}(ub_T(p_i))$. Let $p_i \in P$, $S_i \subseteq ub_T(p_i) \setminus lb_T(p_i)$ and $S = \bigtimes_{j \neq i} ub_T(p_j) \times S_i$. If $\mathcal{N} + C[P \leftarrow S]$ has no solution then values in S_i can be removed from $ub_T(p_i)$.*

This corollary says that if a parametric constraint instance $C[P \leftarrow S]$ is inconsistent with a network whereas all its parameters except one (p_i) can take all the values in their upper bound, then all values in $S(p_i)$ can be discarded from $ub_T(p_i)$.

Corollary 4 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a network $\mathcal{N}$ and e be a solution of $\mathcal{N}$. Given a parameter $p_i \in P$ and a value $v \in ub_T(p_i)$, if we have $t[p_i] = v$ for every tuple t in ub_T s.t. e satisfies $C[P \leftarrow \{t\}]$, then v can be put in $lb_T(p_i)$.*

As in Section 5.1, Algorithm 1 can be modified to deal with the specific case where parameters are represented as sets of integers. Corollary 4 tells us when a value must go in $lb_T(p_i)$ for some parameter p_i. Corollary 3 tells us when a value can be removed from $ub_T(p_i)$ for some parameter p_i.

5.3 Parameters as ranges

The possible values for a parameter p_i can be represented by a range of integers, that is, a special case of set where all values must be consecutive w.r.t. the total order on integers. The only possibility for modifying the sets $lb_T(p_i)$ and $ub_T(p_i)$ of a parameter p_i is to shrink their bounds $min(lb_T(p_i))$, $max(lb_T(p_i))$, $min(ub_T(p_i))$ and $max(ub_T(p_i))$. This case restricts even more the possibilities for the combinations of parameters allowed, and simplifies the learning process. Corollaries 1 and 2 can be specialized as Corollaries 5 and 6.

Corollary 5 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a constraint network $\mathcal{N}$, where parameters are represented as ranges. Let $p_i \in P$ and $v < min(lb_T(p_i))$ (resp. $v > max(lb_T(p_i))$) and $S = \bigtimes_{j \neq i} ub(p_j) \times (-\infty..v]$ (resp. $S = \bigtimes_{j \neq i} ub(p_j) \times [v.. + \infty)$). If $\mathcal{N} + C[P \leftarrow S]$ has no solution then $min(ub_T(p_i)) > v$ (resp. $max(ub_T(p_i)) < v$).*

Corollary 6 *Let $C[P \leftarrow ub_T]$ be an implied constraint in a network $\mathcal{N}$ and e be a solution of $\mathcal{N}$. Given a parameter $p_i \in P$ and a value $v \in ub_T(p_i)$, if we have $t[p_i] = v$ for every tuple t in ub_T s.t. e satisfies $C[P \leftarrow \{t\}]$, then $min(lb_T(p_i)) \leq v \leq max(lb_T(p_i))$.*

Thanks to Corollary 5, tightening the upper bounds for a given p_i is simply done by checking if forcing it to take values smaller than the minimum (or greater than the maximum) value of its lower bound leads to an inconsistency in the network. If yes, we know that all these values smaller than the minimum of $lb_T(p_i)$ (or greater than the maximum of $lb_T(p_i)$) can be removed from $ub_T(p_i)$.

6 Making the Learning Process Tractable

One of the operations performed by the learning technique described in Sections 4 and 5 is a call to a solver to check if the network containing a given instance of parametric constraint is still consistent (line 1 in Algorithm 1). This is NP-complete. A key idea to tackle this problem is that implied constraints learned on a relaxation of the original network are still implied constraints on the original network.

Proposition 4 *Given a network $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and a network $\mathcal{N}' = (\mathcal{X}, \mathcal{D}', \mathcal{C}')$, if a constraint c is implied in $\mathcal{N}'$ and $sol(\mathcal{N}) \subseteq sol(\mathcal{N}')$ then c is implied in $\mathcal{N}$.*

Selecting a subset of the constraints.
Thanks to Proposition 4, we can select any subset of the constraints of the network on which we want to learn an implied constraint, and the constraint learned on that subnetwork will be implied in the original network. In the example of Section 2, the `Gcc` constraint is only posted on the variables representing tasks and is an implied constraint on the subnetwork where neither the resource constraints nor the precedence constraints are taken into account. Thus, the underlying network was a simple one that could easily be solved.

Optimization problems.
In an optimization problem, the set of solutions is the set of instantiations that satisfy all constraints and s.t. a cost function has minimal (or maximal) value. If $\mathcal{N} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is the network and f is the cost function, let $\mathcal{N}' = (\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \{c_f^{UB}\})$

n / d / n_1 / n_2 / $maxR$	#nodes	learning(sec.) + solve (sec.)
4 / 4 / 4 / 0 / 4	47	0.3 + 0.05
4 / 4 / 4 / 0 / 3	47	0.3 + 0.05
4 / 4 / 4 / 0 / 2	23	0.3 + 0.02
4 / 4 / 3 / 1 / 4	43	0.5 + 0.05
4 / 4 / 3 / 1 / 3	30	0.3 + 0.02
4 / 4 / 3 / 1 / 2	3 (unsat)	0.5 + 0.02

Table 2: Learning an implied Gcc on the naive model of Section 2: time to learn and solving time. (n tasks of duration d among which n_j tasks use j resources.)

where c_f^{UB} is a constraint accepting all tuples on $\mathcal{X}$ with cost below a fixed upper bound UB (we concentrate on minimization). If UB is greater than the minimal cost, then any implied constraint in $\mathcal{N}'$ accepts all solutions of $\mathcal{N}$ optimal wrt f. The idea is thus to run a branch and bound on $\mathcal{N}$ for a fixed (short) amount of time. UB is set to the value of the best solution found. Then, the classical learning process is launched on the network $\mathcal{N} + c_f^{UB}$. Note that if the learned implied constraint improves the solving phase, it can permit to quickly find a bound UB' better than UB. Using this new bound UB', we can continue the learning process on $\mathcal{N} + c_f^{UB'}$ instead of $\mathcal{N} + c_f^{UB}$. The learned implied constraint will be tighter because $\mathcal{N} + c_f^{UB}$ is a relaxation of $\mathcal{N} + c_f^{UB'}$. We observe a nice cooperation between the learning process and the solving process.

Time limit.

It can be the case that finding a subnetwork easy to solve is not possible. In this case, we have to find other ways to decrease the cost of the consistency test of line 1 in Algorithm 1. A simple way is to put a time limit to the consistency test. If the solver has not finished its search before the limit, ub_T is not updated and the algorithm goes to the next loop.

7 Experiments

We evaluate our learning technique on the Gcc constraint (see Section 2). The Gcc global constraint is NP-hard to propagate when all elements in its scheme are variables [Quimper *et al.*, 2003]. However, when the cardinalities of the values are parameters that take values in a range, efficient algorithms exist to propagate it [Régin, 1996; Quimper *et al.*, 2004]. In addition, the Gcc constraint can express many different features on the cardinalities of the values. This is thus a good candidate for being added as an implied constraint. In all these experiments we learn a Gcc where cardinalities are ranges of integers. Results in Sections 5.1 and 5.3 apply directly. We used the Choco constraint solver [Choco, 2005].

Satisfaction Problem.

The first experiment was performed on the problem of Section 2. The learning algorithm was run on the subnetwork where resource constraints and precedence constraints are discarded. The learning is thus fast (300 to 800 milliseconds for $n = 4$ or $n = 5$). We fixed to 30 the limit on the number of consistency tests of the subnetwork with the added Gcc (line 1 in Algorithm 1). Table 2 shows the results on a model containing the learned implied Gcc. The first time number corresponds to the learning process and the second to the solving time. Results in Table 2 are almost the same as results in Table 1, where the implied Gcc was added by the expert user. This shows that a short run of the learning algorithm gives some robustness to the model wrt the solving process: The problem is consistently easy to solve on all types of instances whereas some of them could not be solved on the naive model of Section 2. Tables 3 and 4 show the results when we increase the number of tasks and their length ($n = d = 5$ instead of 4) and the number of random precedence constraints (4 instead of 2). We stopped the search at 60 seconds. We were able to solve all problems with size $n = 6$ in a few seconds (including the learning time) while most of them are not solvable without the implied Gcc even after several minutes. This first experiment shows that when taking a naive model with a naive search strategy, our learning technique can improve the robustness of the model in terms of solving time. The effort asked to the user was just to have the intuition that “there is maybe some hidden Gcc related to the ordering of tasks”. This is a much lower effort than studying by hand the possible implied constraints for each number of tasks and durations n and d.

n / d / n_1 / n_2 / $maxR$	#nodes	solve (sec.)
5 / 5 / 5 / 0 / 3	—	> 60
5 / 5 / 5 / 0 / 2	53,250 (unsat)	19
5 / 5 / 3 / 2 / 4	—	> 60
5 / 5 / 3 / 2 / 3	1,790 (unsat)	1.2
5 / 5 / 1 / 4 / 5	11,065	3.9
5 / 5 / 1 / 4 / 4	7,985 (unsat)	2.9

Table 3: Naive model with 5 tasks (n tasks of duration d among which n_j tasks use j resources.)

n / d / n_1 / n_2 / $maxR$	#nodes	learning (sec.) + solve (sec.)
5 / 5 / 5 / 0 / 3	63	0.7 + 0.05
5 / 5 / 5 / 0 / 2	361 (unsat)	0.7 + 0.3
5 / 5 / 3 / 2 / 4	92	0.8 + 0.08
5 / 5 / 3 / 2 / 3	123 (unsat)	0.7 + 0.2
5 / 5 / 1 / 4 / 5	39	0.7 + 0.05
5 / 5 / 1 / 4 / 4	154 (unsat)	0.7 + 0.2

Table 4: Learning an implied Gcc on the problems with 5 tasks: time to learn and solving time. (n tasks of duration d among which n_j tasks use j resources.)

Optimization Problem.

At the Institute of Technology of the University of Montpellier (IUT), n students provide a totally ordered list $(u_1, \ldots, u_m)$ of the m projects they prefer: project u_i is strictly preferred to project u_{i+1}. The goal is to assign projects s.t. two students do not share a same project, while maximizing their satisfaction: a student is very satisfied if she obtains her first choice, less if she obtains the second one, etc. Obtaining no project of her list is the worst possible satisfaction. Additional constraints exist such as a limit of 6 projects selected in the set

n	Initial model		Implied constraint	
	#nodes	solve (sec.)	#nodes	learning (sec.) + solve (sec.)
10	110	**0.2**	12	**0.2** + **0.0**
20	137,982	183.2	1,998	**10.2** + **8.4**
60	—	> 12 hours	8.7×10^6	**110.0** + **43min.**

Table 5: Learning an implied Gcc on the 'projects' optimization problem. n is the number of students

of projects proposed by a teacher. A first model was provided by students from that institute who are novices in constraint programming: students are variables, domains are their set of preferred projects plus an extra-value "0", which means that no project in the preferred list was found. Constraints "$(x_i \neq x_j) \vee (x_i = x_j = 0)$" between each pair (x_i, x_j) of students express mutual exclusion on projects. Each student has a preference-variable measuring her satisfaction. Preference-variables are linked to student-variables by table constraints $\{(u_1, 0), \ldots, (u_m, m-1), (0, m)\}$. The objective is to minimize the sum of preference-variables. This model is referred as the *initial model*. We wish to learn an implied Gcc on student variables. The solver is first launched on the initial model for a short time (1 sec.) to get a first solution S_0 and an upper bound on the quality of solutions, as described in Section 6. We had only one real instance (with 60 students), so we derived from the real data some sub-instances with less students (preserving the ratio "number of projects/number of students" and the distribution of the choices).

Left hand side of Table 5 shows the time and number of nodes with the initial model according to the number of students n. Right hand side of Table 5 shows the results obtained when learning an implied Gcc on the student variables. We fixed to $10 \times n$ the limit on the number of consistency tests (line 1 in Algorithm 1), with a cutoff of 0.5 seconds for each of them. Thus, the maximum learning time grows *linearly* with the size of the problem. The learning time includes the generation of an initial positive example. The results show a tremendous speed up on the model refined with a learned implied Gcc compared to the initial model. This shows that learning implied global constraints in an optimization problem can greatly pay off when the model has been designed by a non CP expert.

8 Conclusion

Global constraints are essential in improving the efficiency of solving constraint models. We proposed a generic framework for automatically learning implied global constraints with parameters. This is both the first approach that permits to derive implied global constraints and the first approach that learns implied constraints according to the actual domains, not just constraints derived from structural or syntactical properties. We applied the generic framework to special properties that global constraints often satisfy. Our experiments show that a very small effort spent learning implied constraints with our technique can dramatically improve the solving time.

References

[Beldiceanu and Contejean, 1994] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Mathl. Comput. Modelling*, 20(12):97–123, 1994.

[Beldiceanu *et al.*, 2005] N. Beldiceanu, M. Carlsson, and J-X. Rampon. Global constraint catalog. *SICS Technical report T2005-08, URL: http://www.sics.se/libindex.html#Technical*, 2005.

[Bessiere *et al.*, 2005] C. Bessiere, R. Coletta, and T. Petit. Acquiring parameters of implied global constraints. *Proceedings CP*, pages 747–751, 2005.

[Bessiere *et al.*, 2006] C. Bessiere, E. Hebrard, B. Hnich, Z. Kiziltan, and T. Walsh. Among, common and disjoint constraints. In *CSCLP: Recent Advances in Constraints*, volume 3978 of *LNCS*, pages 29–43. Springer, 2006.

[Choco, 2005] Choco. A Java library for constraint satisfaction problems, constraint programming and explanation-based constraint solving. *URL: http://sourceforge.net/projects/choco*, 2005.

[Colton and Miguel, 2001] S. Colton and I. Miguel. Constraint generation via automated theory formation. *Proceedings CP*, pages 575–579, 2001.

[Hnich *et al.*, 2001] B. Hnich, J. Richardson, and P. Flener. Towards automatic generation and evluation of implied constraints. *Upsala University T.R.*, 2001.

[Ilog, 2000] Ilog. Solver 5.0 User's Manual. *URL: http://www.ilog.fr*, 2000.

[Puget, 2004] J.F. Puget. Constraint programming next challenge: Simplicity of use. *Proceedings CP*, pages 5–8, 2004.

[Quimper *et al.*, 2003] C.-G. Quimper, P. Van Beek, A. Lopez-Ortiz, A. Golynski, and S. B. Sadjad. An efficient bounds consistency algorithm for the global cardinality constraint. *Proceedings CP*, pages 600–614, 2003.

[Quimper *et al.*, 2004] C.-G. Quimper, A. López-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the *global cardinality* constraint. *Proceedings CP*, 2004.

[Régin, 1996] J-C. Régin. Generalized arc consistency for global cardinality constraint. *Proceedings AAAI*, pages 209–215, 1996.

[Régin, 2001] J-C. Régin. Minimization of the number of breaks in sports scheduling problems using constraints programming. *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 57:115–130, 2001.

[Smith *et al.*, 1999] 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.

[Sterling *et al.*, 1989] L. Sterling, A. Bundy, L. Byrd, R. O'Keefe, and B. Silver. Solving symbolic equations with PRESS. *Journal of Symbolic Computation*, pages 71–84, 1989.