# Constraint Acquisition as Semi-Automatic Modeling

Joël Quinqueton, Remi Coletta, Christian Bessiere, Barry O'Sullivan, Eugene C. Freuder, Sarah O'Connell

# Constraint Acquisition as Semi-Automatic Modeling *

Remi Coletta[1], Christian Bessiere[1], Barry O'Sullivan[2], Eugene C. Freuder[2], Sarah O'Connell[2], Joel Quinqueton[1]

[1] LIRMM-CNRS (UMR 5506), 161 rue Ada 34392 Montpellier Cedex 5, France
{coletta,bessiere,jq}@lirmm.fr

[2] Cork Constraint Computation Centre, University College Cork, Ireland
{b.osullivan,e.freuder,s.oconnell}@4c.ucc.ie

**Abstract**

Constraint programming is a technology which is now widely used to solve combinatorial problems in industrial applications. However, using it requires considerable knowledge and expertise in the field of constraint reasoning. This paper introduces a framework for automatically learning constraint networks from sets of instances that are either acceptable solutions or non-desirable assignments of the problem we would like to express. Such an approach has the potential to be of assistance to a novice who is trying to articulate her constraints. By restricting the language of constraints used to build the network, this could also assist an expert to develop an efficient model of a given problem. This paper provides a theoretical framework for a research agenda in the area of interactive constraint acquisition, automated modelling and automated constraint programming.

## 1 Introduction

Over the last 30 years, considerable progress has been made in the field of Constraint Programming (CP), providing a powerful paradigm for solving complex problems. Applications in many areas such as resource allocation, scheduling, planning and design have been reported in the literature [10]. However, the use of CP still remains limited to specialists in the field. Modelling a problem in the constraint formalism requires significant expertise in constraint programming. This precludes novices from being able to use CP on complex problems without the help of an expert. This has a negative effect on the uptake of constraint technology in the real-world by non-experts [5].

In addition, in many practical applications humans find it difficult to articulate their constraints. While the human user can recognize examples of where their constraints

*The collaboration between LIRMM and the Cork Constraint Computation Centre is supported by a Ulysses Travel Grant from Enterprise Ireland, the Royal Irish Academy and CNRS (Grant Number FR/2003/022). This work has also received support from Science Foundation Ireland under Grant 00/PI.1/C075.

should be satisfied or violated, they cannot articulate the constraints themselves. However, by presenting examples of what is acceptable, the human user can be assisted in developing a model of the set of constraints she is trying to articulate. This can be regarded as an instance of constraint acquisition. One of the goals of our work is to assist the, possibly novice, human user by providing semi-automatic methods for acquiring the user's constraints.

Furthermore, even if the user has sufficient experience in CP to encode her problem, a poor model can negate the utility of a good solver based on state-of-the-art filtering techniques. For example, in order to provide support for modelling, some solvers provide facilities for defining constraints extensionally (i.e., by enumerating the set of allowed tuples). Such facilities considerably extend the expressiveness and ease-of-use of the constraints language, thus facilitating the definition of complex relationships between variables. However, a disadvantage of modelling constraints extensionally is that the constraints lose any useful semantics they may have which can have a negative impact on the inference and propagation capabilities of a solver. As a result, the resolution performance of the solver can be significantly deteriorated in the parts of the problem where such constraints are used. Therefore, another goal of our work is to facilitate the expert user who wishes to reformulate her problem (or a part of it that is suspected of slowing down the resolution). Given sets of accepted/forbidden instantiations of the (sub)problem (that can be generated automatically from the initial formulation), the expert will be able, for instance, to test whether an optimised constraint library associated with her solver is able to model the (sub)problem in a way which lends itself to being efficiently solved.

However, constraint acquisition is not only important in an interactive situation involving a human user. Often we may wish to acquire a constraint model from a large set of data. For example, given a large database of tuples defining buyer behaviour in a variety of markets, for a variety of buyer profiles, for a variety of products, we may wish to acquire a constraint network which describes the data in this database. While the nature of the interaction with the source of training data is different, the constraint acquisition problem is fundamentally the same.

The remainder of this paper is organised as follows. Section 2 presents an overview of the related work in this area, Section 3 provides some preliminary definitions on constraint networks. Section 4 briefly presents the machine learning techniques that can be used for our problem. In Section 5, we formulate our problem as a learning problem. Section 6 presents the technique in detail, and proves some properties of the approach that are guaranteed. In Section 7, some of the issues that the approach raises are presented, and their possible effects on the learning process are illustrated by some preliminary experiments. Some concluding remarks are made in Section 8.

# 2 Related Work

Recently, researchers have become more interested in techniques for solving problems where users have difficulties articulating constraints. In [9], the goal of Rossi and Sperduti is not exactly to help the user learning a constraint network, but to help her learning the valuations of the tuples in a semi-ring constraint network where the constraint structures are already given. Freuder and Wallace have considered suggestion strate-

gies for applications where a user cannot articulate all constraints in advance, but can articulate additional constraints when confronted with something which is unacceptable [3]. Freuder and O'Sullivan have focused on generating constraints which model tradeoffs between variables in problems which have become over-constrained during a interactive configuration session [2]. Version spaces have been reported by O'Connell *et al* for acquiring single constraints, with a focus on acquisition from humans where dialog length is a critical factor [8]. The focus of their work was interactive acquisition of constraints from users of differing abilities.

# 3 Preliminaries

**Definition 1 (Constraint Network)** *A constraint network is defined as a triplet* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *where:*

- $\mathcal{X} = \{X_1, \ldots, X_n\}$ *is a set of variables.*
- $\mathcal{D} = \{D_{X_1}, \ldots, D_{X_n}\}$ *is the set of their domains: each variable* $X_i$ *takes its values in the domain* $D_{X_i}$*.*
- $\mathcal{C} = (C_1, \ldots, C_m)$ *is a sequence of constraints on* $\mathcal{X}$ *and* $\mathcal{D}$*, where a constraint* $C_i$ *is defined by the sequence* $var(C_i)$ *of variables it involves, and the relation* $rel(C_i)$ *specifying the allowed tuples on* $var(C_i)$*.*

We regard the constraints as a sequence to simplify the forthcoming notations.

**Definition 2 (Instance)** *Let* $Y = \{Y_1, \cdots, Y_k\}$ *be a subset of* $\mathcal{X}$*. An* instance $e_Y$ *on* $Y$ *is a tuple* $(v_1, \ldots, v_k) \in D_{Y_1} \times \cdots \times D_{Y_k}$*. This instance is partial if* $Y \neq \mathcal{X}$*, complete otherwise (noted* $e$*). An instance* $e_Y$ *on* $Y$ violates *the constraint* $C_i$ *iff* $var(C_i) \subseteq Y$ *and* $e_Y[var(C_i)] \notin rel(C_i)$*.*

**Definition 3 (Solution)** *A complete instance on the set* $\mathcal{X}$ *of variables is a* solution *of the constraint network* $N = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ *iff it does not violate any constraint. Otherwise it is a* non solution*.* $Sol(N)$ *denotes the set of solutions of* $N$*.*

# 4 The Fundamental Problem

As a starting point, we assume that the user knows the set of variables of her problem and their domains of possible values. She is also assumed to be able to classify an instance as positive (a solution) or negative (non-solution). Therefore, the available data are the set $\mathcal{X}$ of the variables of the problem, their domains $\mathcal{D}$, a subset $E^+$ of the solutions of the problem, and a set $E^-$ of non-solutions.

In addition to the "assisting the expert" perspective, the aim is to code the problem efficiently, using only efficient constraint relations between these variables; i.e. a library of constraints with efficient propagation features is assumed to be given. Indications can also be given revealing the possible location of the constraints, by defining variables between which constraints must be found (learned), or by restricting ourselves to binary constraints only. These semantic and structural limitations define the inductive bias:

**Definition 4 (Bias)** *Given a set $\mathcal{X}$ of variables and the set $\mathcal{D}$ of their domains, a* bias $\mathcal{B}$ *on $(\mathcal{X}, \mathcal{D})$ is a sequence $(B_1, \ldots, B_m)$ of local biases, where a local bias $B_i$ is defined by a sequence $var(B_i) \subseteq \mathcal{X}$ of variables, and a set $L(B_i)$ of possible relations on $var(B_i)$.*

The set $L(B_i)$ of relations allowed on a set of variables $var(B_i)$ can be any library of constraints of arity $|var(B_i)|$.

**Definition 5 (Membership of a Bias)** *Given a set $\mathcal{X}$ of variables and the set $\mathcal{D}$ of their domains, a sequence of constraints $\mathcal{C} = (C_1, \ldots, C_m)$* belongs *to the bias $\mathcal{B} = (B_1, \ldots, B_m)$ on $(\mathcal{X}, \mathcal{D})$ if $\forall C_i \in \mathcal{C}, var(C_i) = var(B_i)$ and $rel(C_i) \in L(B_i)$. We note $\mathcal{C} \in \mathcal{B}$.*

The problem consists in looking for a sequence of constraints $\mathcal{C}$ belonging to a given bias $\mathcal{B}$, and whose solution set is a superset of $E^+$ containing no element of $E^-$.

**Definition 6 (Constraint Acquisition Problem)** *Given a set of variables $\mathcal{X}$, their domains $\mathcal{D}$, two sets $E^+$ and $E^-$ of instances on $\mathcal{X}$, and a bias $\mathcal{B}$ on $(\mathcal{X}, \mathcal{D})$, the* constraint acquisition problem *consists in finding a sequence of constraints $\mathcal{C}$ such that:*

$\mathcal{C} \in \mathcal{B}$,

$\forall e^- \in E^-$, $e^-$ *is a non solution of* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, *and,*

$\forall e^+ \in E^+$, $e^+$ *is a solution of* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

If the sets $E^+$ and $E^-$, called the *training data*, are provided by an interaction with the user, then the acquisition problem can be regarded as the modelling phase for the user's problem. Otherwise, it can be regarded as an assistance to the expert for an automatic reformulation of her problem.

We can point out that if $E^+ \cup E^- = D_{X_1} \times \cdots \times D_{X_n}$, and $\mathcal{B}$ is a bias on $(\mathcal{X}, \mathcal{D})$ containing $n(n-1)/2$ local biases such that for each pair of variables $(X_i, X_j)$, $\exists B_i \in \mathcal{B}$ with $var(B_i) = (X_i, X_j)$, and $L(B_i) = \mathcal{P}(D_{X_i} \times D_{X_j})$,[1] then the constraint acquisition problem answers the representability problem of a relation $\rho = E^+$ with a binary constraint network [7].

# 5 Constraint Acquisition as Concept Learning

Concept induction is a well known paradigm in Machine Learning. The underlying problem can be described the following way: given a set $H$ of hypotheses, two training data sets ($E^+$ of positive and $E^-$ of negative instances), find an hypothesis $h$ consistent with this training data, i.e., which rejects all the negative instances and accepts all the positive instances. The concept providing the training data is called the target concept.

In our context, this concept is the unknown network that we are looking for that consistently captures all the information given by the user in the training set. So, in our vocabulary:

- An hypothesis $h$ is a sequence of constraints,

[1] $E$ being a set, $\mathcal{P}(E)$ is the set of subsets of $E$.

- $H$ is the set of possible sequences of constraints belonging to $\mathcal{B}$,
- The target concept is the sequence of constraints $\mathcal{C}$ we are looking for,
- A positive instance is a solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, a negative one is a non-solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

There are many techniques from the field of Machine Learning, from decision trees to neural networks or genetic algorithms. We propose here a method based on version spaces [6], which has several nice properties amongst which the most interesting from our perspective are: they provide two approximations of the target concept, an upper bound and a lower bound; their computation is incremental with respect to the training data; and the result does not depend on the order of the instances in the training set (commutativity). This last property is essential in an interactive acquisition process.

We briefly present version spaces, which rely on the partial-order based on inclusion in the set $H$ of hypotheses.

**Definition 7 (Generalisation relation $\leq_G$)** *Given $(\mathcal{X}, \mathcal{D})$ a set of variables and their domains, an hypothesis $h_1$ is less general than or equal to an hypothesis $h_2$ (noted $h_1 \leq_G h_2$) iff the set of solutions of $(\mathcal{X}, \mathcal{D}, h_1)$ is a subset of this of $(\mathcal{X}, \mathcal{D}, h_2)$.*

A version space does not only provide one consistent hypothesis, but the whole subset of $H$ consistent with the training data:

**Definition 8 (Version Space)** *Given $(\mathcal{X}, \mathcal{D})$ a set of variables and their domains, $E^+$ and $E^-$ two training data sets , and $H$ a set of hypotheses, the* version space *is the set:*

$$V = \{h \in H / E^+ \subseteq Sol(\mathcal{X}, \mathcal{D}, h), E^- \cap Sol(\mathcal{X}, \mathcal{D}, h) = \emptyset\}$$

Because of its nice property of incrementality with respect to the training data, a version space is learned by incrementally processing the training instances of $E^+$ and $E^-$. In addition, due to the $\leq_G$ partial order, a version space $V$ is completely characterised by two boundaries: the specific boundary $S$ of maximally specific (minimal) elements of $V$ (according to $\leq_G$), and the general boundary $G$ of maximally general (maximal) elements.

**Property 1** *Given a version space $V$, and its boundaries $S$ and $G$, $\forall h \in V, \exists s \in S$ and $\exists g \in G / s \leq_G h \leq_G g$.*

In the general case, $V$ is exponential in the size of the data. So, thanks to Property 1, the constraint acquisition problem is restricted to computing the bounds $S$ and $G$ of the version space consistent with $(E^+, E^-)$.

Given a set of hypotheses $H$ on $(\mathcal{X}, \mathcal{D})$, and the training data $(E^+, E^-)$, if there does not exist any $h \in H$ consistent with $(E^+, E^-)$, then the version space acquisition will finish in a state where there exists $s \in S$ and $g \in G$ such that $s \neq g$ and $g \leq_G s$. This is called the *collapsing* state of the version space.

# 6 Learning the Constraint Version Space

In this section, we describe the process of learning the version space corresponding to the constraint acquisition problem on $(\mathcal{X}, \mathcal{D})$ with the two training sets $(E^+, E^-)$ of solutions and non-solutions, and bias $\mathcal{B}$ on $(\mathcal{X}, \mathcal{D})$.

Let us first define the concepts that will be used at the single constraint level. We can project the generalisation relation $\leq_G$ at the constraint level. To be completely consistent with version space theory, we define $L^H(B_i) = L(B_i) \cup \{\bot, \top\}$, where $\bot$ is the empty relation $\bot$, and $\top$ the universal relation. Note that without loss of generality, the universal relation can be stated as belonging to any library of constraints. Thus, $\leq_g$ is a partial order on $L^H(B_i)$ such that $\forall r_1, r_2 \in L^H(B_i), r_1 \leq_g r_2 \Leftrightarrow r_1 \subseteq r_2$.

Given $L_1 \subseteq L^H(B_i)$ and $L_2 \subseteq L^H(B_i)$, we note that $L_1 \leq_g L_2$ iff $\forall r_1 \in L_1, \forall r_2 \in L_2, r_1 \leq_g r_2$.

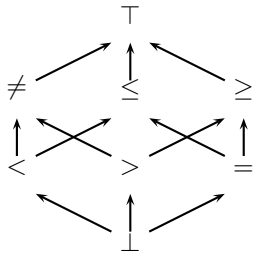



Figure 1: $L^H(B_i)$

**Example 1** Let $L(B_i) = \{<, \leq, =, \geq, >, \neq\}$ be a given local bias. Fig. 1 shows the set $(L^H(B_i), \leq_g)$, which in this case is a lattice.

Restricting ourselves to each constraint individually, we introduce a local version space for each local bias. Because $\leq_g$ is, like $\leq_G$, a partial order, each local version space inherits Property 1. Thus, each local version space is completely characterized by its own local specific and general boundaries.

**Definition 9 (Local boundaries)** *$L(S_i)$ (resp. $L(G_i)$) is the set of relations of $L^H(B_i)$ which appear in an element of $S$ (resp. $G$):*

$$L(S_i) = \{r \in L^H(B_i) / \exists s \in S : (var(Bi), r) \in s\}$$
$$L(G_i) = \{r \in L^H(B_i) / \exists g \in G : (var(Bi), r) \in g\}$$

*$S_i$ and $G_i$ are the corresponding sets of constraints:*

$S_i = \{(var(B_i), r)\}$, *where* $r \in L(S_i)$; $\quad G_i = \{(var(B_i), r)\}$, *where* $r \in L(G_i)$

We are now ready to describe the CONACQ algorithm (Algorithm 1), which takes as input two training sets $E^+$, $E^-$, and returns the corresponding version space $V$ on the bias $\mathcal{B}$. We present step by step the different scenarios that can occur when a training instance is processed.

## 6.1 Instances from $E^+$

A positive instance $e^+$ must be a solution of all the networks $(\mathcal{X}, \mathcal{D}, h)$ for which $h \in V$. So,

$$\forall h \in V, \forall C_i \in h, e^+[var(C_i)] \in rel(C_i)$$

Projecting onto the local version spaces of each local bias $B_i$, we obtain the following property:

**Property 2 (Projection property of $S_i$'s)** *Each local specific boundary $S_i$ must accept all the positives instances. $L(S_i)$ is thus the set of maximally specific relations (minimal w.r.t. $\leq_g$) of $L(B_i)$ that accept all $E^+$:*

$$L(S_i) = min_{\leq_g}\{r \in L^H(B_i) / \forall e^+ \in E^+, e^+[var(B_i)] \in r\}$$

**Corollary 1** *The specific boundary $S$ is the Cartesian product of the local specific boundaries $S_i$'s, i.e., the set of hypotheses, where each constraint takes its relation from $L(Si)$:*

$$S = \bigtimes_{i \in 1..m} S_i$$

From Property 2, when a positive instance $e^+$ is presented, each local bias $B_i$ can be processed individually (line 2 of Algorithm CONACQ). If the specific boundary of a constraint already accepts this positive instance, it is skipped (line 3), else the boundary goes up to the most specific relations of the local version space (i.e., the relations of $L^H(B_i)$ between $L(S_i)$ and $L(G_i)$) that accept $e^+$ (line 4). If no such relation exists, this means that no hypothesis can accept this positive instance. Then, the algorithm terminates since a collapsing state has been encountered (line 5).

## 6.2 Instances from $E^-$

A negative instance $e^-$ must be a non solution for all the networks $(\mathcal{X}, \mathcal{D}, h)$ where $h \in V$. So,

$$\forall h \in V, \exists C_i \in h / e^-[var(C_i)] \notin rel(C_i)$$

Since at least one violated constraint is sufficient for an instance to be regarded as negative, instead of all satisfied constraints necessary in the case of a positive instance, $G$ does not have the projection property exhibited by $S$:[2]

$$L(G_i) \neq max_{\leq_g}\{r \in L^H(B_i) / \forall e^- \in E^-, e^-[var(B_i)] \notin r\}$$

We can only say that $\forall e^- \in E^-, \exists i / \forall r \in L(G_i), e^-[var(B_i)] \notin r$. However, the cause of the rejection (which constraint(s) has been violated) may not be obvious. Furthermore, storing only the local general boundaries $G_i$'s is not sufficient to express this uncertainty.

[2] If this was not the case the constraint defined on $rel(B_i)$ would be sufficient to reject all negative instance of $E^-$.

The traditional approach to version space learning involves storing the set of all possible global boundaries $G$. However, this can require exponential space and time [4]. In order to ensure our algorithm remains polynomial, we do not store this set, but encode each negative instance $e^-$ as a clause, $Cl$. Each constraint that could possibly be involved in the rejection of a negative example $e^-$ will be encoded as a meta-variable in the clause $Cl$. Semantically, the clause $Cl$ represents the disjunction of the possible explanations for the rejection of this negative instance. In other words, it encodes a disjunction of the constraints that could have been responsible for the inconsistency in the instance.

When a negative instance $e^-$ is presented, a new clause, initially empty, is built by processing each local bias $B_i$ one-by-one (lines 12-14). Those biases whose specific boundary, $L(S_i)$, already accepts the negative instance, $e^-$, are skipped (line 15). The reason being that $S_i$ is the maximally specific boundary of the local version space for $B_i$ and, by definition, we know that at each step of the learning process the constraint defined on $rel(B_i)$ cannot be involved in the rejection of $e^-$, since this has already been deemed acceptable by at least one positive example.

For all the other constraints, a subset of which is responsible for the rejection $e^-$, we compute $A_i$, the subset of maximally specific relations (w.r.t. $\leq_g$) between $L(S_i)$ and $L(G_i)$ which accept $e^-[var(B_i)]$, i.e. the least upper bound that $B_i$ must not take if it is proven to be a contributor to the rejection of $e^-$ (line 16). Depending on this set of relations we have two alternative courses of action to consider. Firstly, if the set $A_i$ is empty it means that all possible relations for the constraint defined on $rel(B_i)$ already reject $e^-$. Therefore, every hypothesis in the version space is consistent with $e^-$ so there is nothing to do for $e^-$; we are then ready to process the next instance (line 17). Secondly, if $A_i$ is not empty, we add the meta-variable $(L(G_i) <_g A_i)$ to the clause $Cl$ (line 18). The semantic of this meta-variable is *"if $B_i$ is involved in the rejection of $e^-$, then $G_i$ must be made more specific than $A_i$"*.

To reject $e^-$, a sequence of constraints $h$ must satisfy at least one meta-variable in the clause $Cl$ encoding $e^-$. We will denote this as $h \models Cl$. The set of clauses, characterizing all the set $E^-$, is denoted as $\mathcal{K}$. Below we will summarise the maintenance of these clauses for clarity.

## 6.3 Maintenance of the set of clauses

**i** Maintenance of clauses when positive instances are added.

> If a clause $Cl$ contains the meta-variable $(L(G_i) <_g A_i)$ and if after the processing of positive instances, $L(S_i) \cap A_i$ becomes not empty, then the relations in $L(S_i) \cap A_i$ are no longer a possible explanation for the rejection of the negative instance denoted by $Cl$. Let $A'_i$ be the set of relations of $A_i$ that are not in $L(S_i)$. If $A'_i$ is not empty, the new explanation becomes $(L(G_i) <_g A'_i)$. Otherwise the meta-variable $(L(G_i) <_g A_i)$ is erased from $Cl$ (lines 6-9).

**ii** Empty clause implies that the version space has collapsed.

> When a clause $Cl$, encoding a negative instance $e^-$, is empty, either during its construction (line 19), or following the addition of some positive instances (line 10), it implies that there does not exist a possible explanation for rejecting the negative instance $e^-$, then the algorithm collapses.

**Algorithm 1:** The CONACQ Algorithm

$\mathcal{K} \leftarrow \emptyset$
**foreach** $B_i$ **do** $L(S_i) \leftarrow \{\bot\}$; $L(G_i) \leftarrow \{\top\}$
**foreach** *training instance e* **do**
1 **if** $e \in E^+$ **then**
2 **foreach** $B_i$ **do**
3 **if** $\exists r \in L(S_i)/e[var(B_i)] \notin r$ **then**
4 $L(S_i) \leftarrow min_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $e[var(B_i)] \in r\}$
5 **if** $L(S_i) = \emptyset$ **then** "collapsing"
6 **foreach** $Cl/(L(G_i) <_g A_i) \in Cl$ *and* $A_i \cap L(S_i) \neq \emptyset$ **do**
7 $Cl \leftarrow Cl \setminus (L(G_i) <_g A_i)$
8 $A'_i \leftarrow A_i \setminus L(S_i)$
9 **if** $A'_i \neq \emptyset$ **then** $Cl \leftarrow Cl \cup (L(G_i) <_g A'_i)$
10 **if** $Cl = \emptyset$ **then** "collapsing"
11 **if** $Cl = \{(L(G_i) <_g A_i)\}$ **then**
$G_i \leftarrow max_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $r <_g A_i\}$ ; $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl$
12 **if** $e \in E^-$ **then**
13 $Cl \leftarrow \emptyset$ ; $reject \leftarrow false$
14 **foreach** $B_i$ ***while*** $\neg reject$ **do**
15 **if** $\exists r \in L(S_i)/e[var(B_i)] \notin r$ **then**
16 $A_i \leftarrow min_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $e[var(B_i)] \in r\}$
17 **if** $A_i = \emptyset$ **then** $reject \leftarrow true$
18 **else** $Cl \leftarrow Cl \cup (L(G_i) <_g A_i)$
**if** $\neg reject$ **then**
19 **if** $Cl = \emptyset$ **then** "collapsing"
20 **if** $Cl = \{(L(G_i) <_g A_i)\}$ **then**
$G_i \leftarrow max_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $r <_g A_i\}$ ; $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl$
21 **else if** $\nexists Cl'/Cl' \subseteq Cl$ **then** $\mathcal{K} \leftarrow \mathcal{K} \cup Cl$
22 **foreach** $Cl''/Cl \subset Cl''$ **do** $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl''$

**iii** The one-meta-variable clauses.

The one-meta-variable clauses represent the only possible explanation for the rejection of the coded negative instances. They must be reported on the local version space then removed from $\mathcal{K}$. Let $(L(G_i) <_g A_i)$ be a such clause: $L(G_i)$ must be made more specific than $A_i$.(lines 11 and 20).

**iv** Subsumption of negative instances.

A negative instance $e_1^-$ (encoded by $Cl_1$) subsumes $e_2^-$ (encoded by $Cl_2$) iff $Cl_1 \subseteq Cl_2$. Indeed, if $Cl_1$ is satisfied, then $Cl_2$ too. It is thus useless to store

$Cl_2$. Lines 21-22 of Algorithm CONACQ maintain this minimality of the base of clauses $\mathcal{K}$.

A proof of Correctness of Algorithm CONACQ can be found in [1]

**Example 2** In Figure 2, we describe the learning process on a single constraint $C_{12}$ and where $B_{12}$ is the bias of Figure 1. Initially $L(S_{12}) = \{\bot\}$, and $L(G_{12}) = \{\top\}$. When the positive instance $e_1^+ = (1, 1)$ is received, $L(S_{12})$ goes up in $L^H(B_{12})$ to the most specific relations accepting this tuple (a), namely $\{=\}$. We know now that the final relation of constraint $C_{12}$ will be more general than or equal to "=". When the negative instance $e_2^- = (2, 1)$ is received, it is necessary to restrict $L(G_{12})$ such that it will reject this tuple (b). At this step: $L(S_{12}) = \{=\}$, and $L(G_{12}) = \{\leq\}$. The negative instance $e_3^- = (0, 3)$ forbids the relation "$\leq$" (c). Then $L(S_{12}) = L(G_{12}) = \{=\}$, that is a local convergence.

| E | $X_1$ | $X_2$ |
|---|---|---|
| $e_1^+$ | 1 | 1 |
| $e_2^-$ | 2 | 1 |
| $e_3^-$ | 0 | 3 |

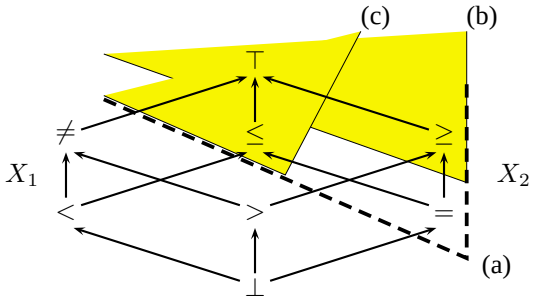
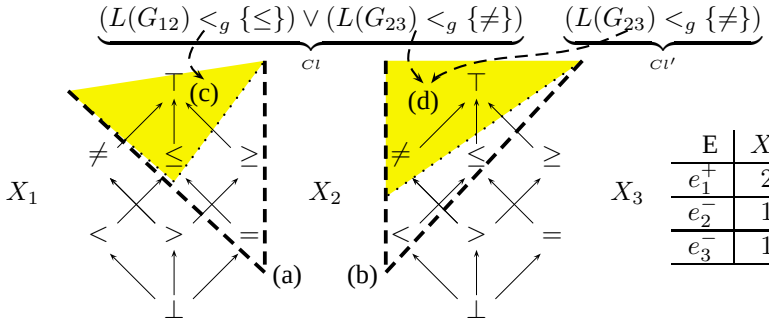



Figure 2: A local example

**Example 3** In Figure 3, we present an example of our algorithm using a constraint network involving three variables $X_1$, $X_2$ and $X_3$. The biases used are $\{B_{12}, B_{23}\}$, where $var(B_{12}) = (X_1, X_2)$, $var(B_{23}) = (X_2, X_3)$ and $L^H(B_{12}) = L^H(B_{23})$, which for the purposes of this example we will assume to be the same as that presented as $L^H(B_i)$ in Figure 1. When processing the positive example $e_1^+ = (2, 2, 5)$, due to the projection property of $S$, $S_{12}$ **and** $S_{23}$ go up (i.e., $L(S_{12}) \leftarrow \{=\}$ and $L(S_{23}) \leftarrow \{<\}$) (a) and (b).

However, since $G$ does not have this projection property: when $e_2^- = (1, 3, 2)$ is received, either $C_{12}$ must reject the tuple $(1, 3)$ **or** $C_{23}$ must reject $(3, 2)$. Therefore, we build the clause $Cl = (rel(C_{12}) <_g^L \{\leq\}) \vee (rel(C_{23}) <_g^L \{\neq, \})$ to store these alternatives (c) or (d). When the negative instance $e_3^- = (1, 1, 0)$ is received, we know that the constraint $C_{12}$ is not involved in its rejection, because $e_3^-[var(C_{12})] = (1, 1)$ is an allowed tuple of $L(S_{12}) = \{=\}$. The only explanation for the rejection of $e_3^-$ is $Cl' = (rel(C_{23}) <_g^L \{\neq, \})$ (d). Note than $Cl'$ subsumes $Cl$ (i.e., $Cl' \subseteq Cl$). The explanation of the rejection of $e_3^-$ (d) is also a valid one for $e_2^-$: $Cl$ becomes subsumed by $Cl'$, and thus is discarded (the explanation (c) is discarded too). After these three instances, the CONACQ result is $L(S_{12}) = \{=\}$, $L(G_{12}) = \{\top\}$, $C_{23}$: $L(S_{23}) = \{<\}$ and $L(G_{23}) = \{\leq\}$.




| E | $X_1$ | $X_2$ | $X_3$ |
|---|---|---|---|
| $e_1^+$ | 2 | 2 | 5 |
| $e_2^-$ | 1 | 3 | 2 |
| $e_3^-$ | 1 | 1 | 0 |

Figure 3: A global example

# 7 Experiments and Observations

We report here on some preliminary experiments to evaluate the learning capabilities of our approach. Rather than focusing on techniques for minimising the number of interactions, our focus here is on studying a number of properties of the CONACQ algorithm which provide motivation for our research agenda.

We performed experiments with a simulated teacher, which plays the role of the user, and a simulated learner that uses the algorithm presented earlier. The teacher has the knowledge of a randomly generated (target) network, represented by the triple $< 50, 8, C >$, defining a problem involving 50 variables with domains $\{1, ..8\}$, and a number $C$ of constraints. Each constraint is randomly chosen from the bias $\{<, =, >, \leq, \neq, \geq\}$. The teacher provides the learner with solutions and non solutions. The learner acquires a version space for the problem using CONACQ algorithm.

## 7.1 Experiment 1: Effect of the order of the instances

In this following experiment, we assess aspects of the runtime characteristics of the CONACQ algorithm. In particular, we study computing time and the size of the version space, while varying the order in which examples are presented. Instances from a set $E$ of size 100 are given by the teacher to the learner based on a $< 50, 8, 50 >$ network. The set $E$ contains 10 positive and 90 negative instances.

Table 1 presents the time needed by the learner to acquire the version space, $V$, for the example set while varying the arrival time of the 10 positive instances. The positive instances were presented at the beginning (a), middle (b), and end (c) of the interaction between teacher and learner.

Table 1: Effect of the timing of the introduction of positive instances

| Introduction time for positives | 0 (a) | 50 (b) | 90 (c) |
|---|---|---|---|
| Computing time (in sec.) | 3.3 | 5.1 | 8.6 |
| $log(\|V\|)$ | 2,234 | 2,234 | 2,234 |

We observe that *"the sooner, the better"* seems to be the good strategy for the introduction of positive instances. Indeed, the specific bound $S$ rises quickly in the space of hypotheses with positive instances, reducing the size of the version space.

Because of that, the CPU time needed is also reduced when positive instances arrive at the beginning. But we can see that the final size of the version space is not affected by the order of the instances. This is due to the commutativity property of version spaces, discussed in Section 5.

## 7.2 Experiment 2: Utility of partial instances

In some cases, the user can reject an instance while justifying it by a negative sub-instance. For example, in a real-estate setting the customer (teacher) might reject an apartment citing the reason that *"this living-room is too small for me"*. The estate agent (learner) knows that the violation is due to the variables defining the living room, which can being very helpful for handling negative examples. The utility of such justified rejections can be measured by providing our learner with partial instances. In the following experiment (Table 2), the teacher provides the learner with 90 partial negative instances (after 10 complete positive ones) in the training data. We consider partial instances involving 2, 5, 10 variables, and report the size of the version space and of the set of clauses after 100 instances have been given.

Table 2: Effect of the partial instances

| Nb of variables involved in instances of $E^-$ | 50 | 10 | 5 | 2 |
|---|---|---|---|---|
| $log(\|V\|)$ | 2,234 | 2,233 | 2,225 | 2,144 |
| $\|K\|$ ($10^4$ meta-variables) | 7.6 | 6.1 | 3.2 | 0 |

We observe that partial instances speed up the process of convergence of the version space. The smaller these partial instances are, the more helpful they are. This opens a promising way of helping the learning process: asking the user to justify why she rejects some instances can assist in reducing the length of the dialog with the teacher. This is a critical issue if we are learning in an interactive setting from a human user.

## 7.3 Experiment 3: Non-representable constraints and version space collapse

We have seen that our algorithm learns a network expressed using a given bias. An overly restrictive bias leads to version space collapse since it is unlikely to be capable of expressing the target network. This happens because some of the constraints of the target network are non-representable in the given bias. In the following experiment, (Table 3), we analyse the effect on the speed of version space collapse by varying the proportion of non-representable constraints in the target network. For each proportion of non-representable constraints we learned 100 different $< 50, 8, 50 >$ target networks. Non-representable constraints, generated randomly, were not members of our bias $\{<, =, >, \leq, \neq, \geq\}$. The table presents the number of times the version space collapsed after 1,000 instances where presented. Note that the focus of this experiment is related to automatic reformulation rather than interactive modelling.

We observe that the more non-representable constraints in the target network, the faster the version space collapses. This is to be expected since version spaces are sensitive to noise (errors) in the training data. However, an interesting observation was

Table 3: Effect of non-representable constraints

| Ratio of non representable constraints | 10% | 25% | 50% |
|---|---|---|---|
| % of collapsing | 47 | 78 | 100 |

made during this analysis. On a number of constraint networks containing very few non-representable constraints we observed that the version space did not collapse, even after millions of instances were presented. This seems to violate Mitchell's theorem of representability [6]. However, what occured in these situations was that the non-representable constraints in the target *become* representable by propagation of other constraints. As an example, consider three variables $X_1$, $X_2$, and $X_3$ linked by the following constraints in the target network: $X_1 = X_2$, $X_2 = X_3$, and $(X_1 = X_3) \vee (X_1 = X_3 + 5)$. Obviously, the constraint between $X_1$ and $X_3$ is not representable in the bias $\{<, =, >, \leq, \neq, \geq\}$, and every network containing it is expected to cause the version space to collapse. However, this is not the case in this example, because we can restrict the constraint between $X_1$ and $X_3$ to $X_1 = X_3$ by transitivity. This transitive closure constraint is representable.

It is worth noting that if we had used a technique that learned each constraint in the network separately, the version space for our problem would have collapsed had we attempted to acquire the constraint between $X_1$ and $X_3$.

### 7.4 Observation: Implicit constraints

The general phenomenon of constraints that can be inferred by other constraints can have another effect, which is to prevent the version space from converging to the smallest possible on each constraint taken separately. Consider again an example with three variables $X_1$, $X_2$ and $X_3$, linked in the target network by the constraints $X_1 = X_2$, $X_2 = X_3$, and $X_1 = X_3$. Furthermore, consider what occurs at some point in the learning process when the version space local to the constraint defined on $(X_1, X_3)$ contains both $X_1 = X_3$ and $X_1 \leq X_3$. If the training data contains only complete instances, it is impossible to converge to the constraint $X_1 = X_3$ because every negative instance that would permit us to discard $X_1 \leq X_3$ from the version space (e.g., $((X_1, 1), (X_2, 2), (X_3, 2))$) will also be rejected by $X_1 = X_2$ or by $X_2 = X_3$. Thus, the boundary $G$ will never determine that culpability lies with $X_1 \leq X_3$.

Applying some levels of local consistency seems to be a promising approach to improving the reduction of the version space, by adding implicit constraints to the learned network. In the previous example, path-consistency would be enough to deduce that the only candidate between $X_1$ and $X_3$ is the constraint $X_1 = X_3$. This phenomenon can cause the $VS$ to keep a size bigger than it should have.

## 8 Conclusion

We have proposed an original method to learn constraint networks from instances that should or should not be solutions. The technique used is based on version spaces, a machine learning paradigm that has good properties (e.g., incrementality, commutativity, wrt the training data) that will be essential in a process interacting with a user.

Even if this paper is mainly a description of the general process of learning a constraint network from instances, we can easily foresee the many applications it could have, in assisting a novice in modelling her problem, or helping an expert to test whether a given library of constraints with good computational properties can encode her problem.

We have presented preliminary experiments that show that our approach raises several important issues, such as the speed of the learning process, or the question of the implicit constraints. Based on these experiments, and the framework in general, we have given an insight into some of the very interesting research issues which are raised by our work.

# References


[1] R. Coletta, C. Bessiere, B. O'Sullivan, E.C. Freuder, S. O'Connell, and J. Quinqueton. Semi-automatic modeling by constraint acquisition. In *CP-03 Second Workshop on Reformulating Constraint Satisfaction Problems*, 2003.

[2] E.C. Freuder and B. O'Sullivan. Generating tradeoffs for interative constraint-based configuration. In Toby Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, pages 590–594, November 2001.

[3] E.C. Freuder and R.J. Wallace. Suggestion strategies for constraint-based matchmaker agents. In *Principles and Practice of Constraint Programming - CP98*, pages 192–204, October 1998.

[4] Haym Hirsh. Polynomial-time learning with version spaces. In *National Conference on Artificial Intelligence*, pages 117–122, 1992.

[5] J. Little, C. Gebruers, D. Bridge, and E.C. Freuder. Capturing constraint programming experience: A case-based approach. In *CP-02 Workshop on Reformulating Constraint Satisfaction Problems*, 2002.

[6] T. Mitchell. Concept learning and the general-to-specific ordering. In *Machine Learning*, chapter 2, pages 20–51. McGraw Hill, 1997.

[7] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7(95-132), 1974.

[8] S. O'Connell, B. O'Sullivan, and E.C. Freuder. Query generation for interactive constraint acquisition. In *Proceedings of the 4th International Conference on Recent Advances in Soft Computing (RASC-2002)*, pages 295–300, December 2002.

[9] F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. *Journal of experimental and theoretical computer science*, 10, 1998.

[10] M. Wallace. Practical applications of constraint programming. *Constraints*, 1(1–2):139–168, 1996.