A SAT-Based Version Space Algorithm for Acquiring Constraint Satisfaction Problems

Christian Bessiere\textsuperscript{1}, Remi Coletta\textsuperscript{1}, Frédéric Koriche\textsuperscript{1}, and Barry O’Sullivan\textsuperscript{2}

\textsuperscript{1} LIRMM, CNRS / U. Montpellier, Montpellier, France
\textsuperscript{2} Cork Constraint Computation Centre, University College Cork, Ireland
{bessiere,coletta,koriche}@lirmm.fr, b.osullivan@4c.ucc.ie

Abstract. Constraint programming is rapidly becoming the technology of choice for modelling and solving complex combinatorial problems. However, users of this technology need significant expertise in order to model their problems appropriately. The lack of availability of such expertise is a significant bottleneck to the broader uptake of constraint technology in the real world. We present a new SAI-based version space algorithm for acquiring constraint satisfaction problems from examples of solutions and non-solutions of a target problem. An important advantage is the ease with which domain-specific knowledge can be exploited using the new algorithm. Finally, we empirically demonstrate the algorithm and the effect of exploiting domain-specific knowledge on improving the quality of the acquired constraint network.

1 Introduction

Over the last thirty years, considerable progress has been made in the field of Constraint Programming (CP), providing a powerful paradigm for solving combinatorial problems. Applications in many areas, such as resource allocation, scheduling, planning and design have been reported in the literature [16]. Informally, the basic idea underlying constraint programming is to model a combinatorial problem as a constraint network, i.e., using a set of variables, a set of domain values and a collection of constraints. Each constraint specifies a restriction on some set of variables. For example, a constraint such as $x_1 \leq x_2$ states that the value assigned to $x_1$ must be less or equal than the value assigned $x_2$. A solution of the constraint network is an assignment of domain values to variables that satisfies every constraint in the network. The Constraint Satisfaction Problem (CSP) is hence the problem of finding a solution for a given network.

However, the specification of constraint networks still remains limited to specialists in the field. Actually, modelling a combinatorial problem in the constraints formalism requires significant expertise in constraint programming. One of the reasons for this bottleneck stems from the fact that, for any problem at hand, different models of this problem are possible, and two distinct constraint networks that represent the same problem can critically differ on performance. An expert in constraint programming typically knows how to decompose the problem into a set of constraints for which very efficient propagation algorithms have been developed. Such a level of background knowledge precludes novices from being able to use constraint networks on complex problems.
without the help of an expert. Consequently, this has a negative effect on the uptake of constraint technology in the real world by non-experts.

To alleviate this issue, this paper envisions the possibility of acquiring a constraint network from a set of examples and a library of constraints. The constraint acquisition process is regarded as an interaction between a user and the learner. The user has a combinatorial problem in mind, but does not know how this problem can be modelled as an efficient constraint network. Yet, the user has at her disposal a set of solutions (positive examples) and non-solutions (negative examples) for this problem. For its part, the learner has at its disposal a library of constraints for which efficient propagation algorithms are known. The goal for the learner is to induce a constraint network that uses combinations of constraints defined from the library and that is consistent with the solutions and non-solutions provided by the user.

The main contribution of this paper is a SAT-based algorithm, named CONACQ (for Constraint ACQuisition), that is capable of learning a constraint network from a set of examples and a library of constraints. The algorithm is based on the paradigm of version space learning [11]. In the context of constraint acquisition, a version space can be regarded as the set of all constraint networks defined from the given library that are consistent with the received examples. The key idea underlying the CONACQ algorithm is to consider version-space learning as a satisfiability problem. Namely, any example is encoded as a set of clauses using as atoms the constraint vocabulary defined from the library, and any model of the resulting satisfiability problem captures an admissible constraint network for the corresponding acquisition problem.

This approach has a number of distinct advantages. Firstly and most importantly, the formulation is generic, so we can use any SAT solver as a basis for version space learning. Secondly, we can exploit powerful SAT concepts such as unit propagation and backbone detection [12] to improve learning rate. Thirdly, and finally, we can easily incorporate domain-specific knowledge in constraint programming to improve the quality of the acquired network. Specifically, we develop two generic techniques for handling redundant constraints in constraint acquisition. The first is based on the notion of redundancy rules, which can deal with some, but not all, forms of redundancy. The second technique, based on backbone detection, is far more powerful.

2 Preliminaries

A constraint network consists of a set of variables, a set of domain values and a set of constraints. We assume that the set of variables and the set of domain values are finite, pre-fixed and known to the learner. This vocabulary is, thus, part of the common knowledge shared between the learner and the user. Furthermore, the learner has at its disposal a constraint library from which it can build and compose constraints. The problem is to find an appropriate combination of constraints that is consistent with the examples provided by the user. Finally, for sake of clarity, we shall assume that every constraint defined from the library is binary. This assumption greatly simplifies the notation used in the paper. Yet, we claim that the results presented here can be easily extended to constraints of higher arity.
More formally, the constraint vocabulary consists of a finite set of variables $X$ and a finite set of domain values $D$. We implicitly assume that every variable in $X$ uses the same set $D$ of domain values, but this condition can be relaxed in a straightforward way. The cardinalities of $X$ and $D$ are denoted $n$ and $d$, respectively.

A binary constraint is a tuple $c = (\text{var}(c), \text{rel}(c))$ where $\text{var}(c)$ is a pair of variables in $X$ and $\text{rel}(c)$ is a binary relation defined on $D$. The sequence $\text{var}(c)$ is called the scope of $c$ and the set $\text{rel}(c)$ is called the relation of $c$. With a slight abuse of notation, we shall often use $c_{ij}$ to refer to the constraint with relation $c$ defined on the scope $\{x_i, x_j\}$. For example, $\leq_{12}$ denotes the constraint specified on $\{x_1, x_2\}$ with relation “less than or equal to”. A binary constraint network is a set $C$ of binary constraints.

A constraint library is a collection $B$ of binary constraints. From a constraint programming point of view, any constraint $B$ is a set of constraints for which (efficient) propagation algorithms are known. A constraint network $C$ is said to be admissible for a library $B$ if for each constraint $c_{ij}$ in $C$ there exists a set of constraints $\{b_{1ij}, \ldots, b_{kij}\}$ in $B$ such that $c_{ij} = b_{1ij} \cap \cdots \cap b_{kij}$. In other words, a constraint network is admissible for some library if each constraint in the network is defined as the intersection of a set of allowed constraints from the library.

An example is a map $e$ that assigns to each variable $x$ in $X$ a domain value $e(x)$ in $D$. Equivalently, an example $e$ can be regarded as a tuple in $D^n$. An example $e$ satisfies a binary constraint $c_{ij}$ if the pair $(e(x_i), e(x_j))$ is an element of $c_{ij}$. An example $e$ satisfies a constraint network $C$ if $e$ satisfies every constraint in $C$. If $e$ satisfies $C$, then $e$ is called a solution of $C$; otherwise, $e$ is called a non-solution of $C$. In the following, $\text{sol}(C)$ denotes the set of solutions of $C$.

Finally, a training set consists of a pair $(E^+, E^-)$ of sets of examples. Elements of $E^+$ are called positive examples and elements of $E^-$ are called negative examples. A constraint network $C$ is said to be consistent with a training set $(E^+, E^-)$ if every example in $E^+$ is a solution of $C$ and every example in $E^-$ is a non-solution of $C$.

**Definition 1 (Constraint Acquisition Problem).** Given a constraint library $B$ and a training set $(E^+, E^-)$, the Constraint Acquisition Problem is to find a constraint network $C$ admissible for the library $B$ and consistent with the training set $(E^+, E^-)$.

**Example 1.** Consider the vocabulary defined by the set $X = \{x_1, x_2, x_3\}$ and the set $D = \{1, 2, 3, 4, 5\}$. In the following, the symbols $\top$ and $\bot$ refer to the total relation and the empty relation over $D$, respectively. Let $B$ be the constraint library defined as follows: $B = \{\top_{12}, \leq_{12}, \neq_{12}, \geq_{12}, \top_{23}, \leq_{23}, \neq_{23}, \geq_{23}\}$.

Note that the relations $\leq_{12}$, $\neq_{12}$, $\geq_{12}$ and $\neq_{23}$ can be derived from the intersection closure of $B$. Now, consider the two following networks $C_1 = \{\leq_{12} \cap \geq_{12}, \top_{23} \cap \leq_{23}, \neq_{23}\}$ and $C_2 = \{\leq_{12} \cap \geq_{12}, \leq_{23} \cap \geq_{23}\}$. Each network is admissible for $B$. Finally, consider the training set $E$ formed by the three examples $e_1^+ = ((x_1, 1), (x_2, 2), (x_3, 5))$, $e_2^- = ((x_1, 1), (x_2, 3), (x_3, 3))$, and $e_3^- = ((x_1, 1), (x_2, 1), (x_3, 1))$. The first example is positive and the last two are negative. We can easily observe that $C_1$ is consistent with $E$, while $C_2$ is inconsistent with $E$.

The following lemma captures an important semantic property of constraint networks. It will be frequently used in the remaining sections.
Lemma 1. Let $\mathcal{B}$ be a constraint library, $\mathcal{C}$ be a constraint network admissible for $\mathcal{B}$ and $e$ be an example. Then $e$ is a non-solution of $\mathcal{C}$ iff there exists a pair of constraints $b_{ij}$ and $c_{ij}$ such that in $b_{ij} \in \mathcal{B}$, $c_{ij} \in \mathcal{C}$, $c_{ij} \subseteq b_{ij}$ and $e$ does not satisfy $b_{ij}$.

Proof. ($\Rightarrow$) Let us consider that $e$ is a non-solution of $\mathcal{C}$. By definition, there exists a constraint $c_{ij} \in \mathcal{C}$ such that $e$ does not satisfy $c_{ij}$. It follows that the pair $(e(x_i), e(x_j))$ is not an element of $c_{ij}$. Furthermore, since $\mathcal{C}$ is admissible for $\mathcal{B}$, there exists a set $\{b_{ij}^1, \ldots, b_{ij}^k\}$ of constraints in $\mathcal{B}$ such that $c_{ij} = b_{ij}^1 \cap \cdots \cap b_{ij}^k$. Consequently, the pair $(e(x_i), e(x_j))$ is not an element of $b_{ij}^1 \cap \cdots \cap b_{ij}^k$. It follows that $(e(x_i), e(x_j))$ is not an element of $b_{ij}$, for some constraint $b_{ij}$ in the set $\{b_{ij}^1, \ldots, b_{ij}^k\}$. By construction, $c_{ij} \subseteq b_{ij}$. Since $e$ does not satisfy $b_{ij}$, the result follows.

($\Leftarrow$) Now, let us assume that there exists a pair of constraints $b_{ij}$ and $c_{ij}$ such that in $b_{ij} \in \mathcal{B}$, $c_{ij} \in \mathcal{C}$, $c_{ij} \subseteq b_{ij}$ and $e$ does not satisfy $b_{ij}$. Obviously, the pair $(e(x_i), e(x_j))$ is not an element of $b_{ij}$. Since $c_{ij} \subseteq b_{ij}$, it follows that $(e(x_i), e(x_j))$ is not an element of $c_{ij}$. Therefore, $e$ does not satisfy $c_{ij}$ and hence, $e$ is a non-solution of $\mathcal{C}$.  

3 The CONACQ Algorithm

In this section we present a SAT-based algorithm for acquiring constraint satisfaction problems based on version spaces. Informally, the version space of a constraint acquisition problem is the set of all constraint networks that are admissible for the given library and that are consistent with the given training set. In the SAI-based framework this version space is encoded in a clausal theory, and each model of the theory is a candidate constraint network.

Let $\mathcal{B}$ be a constraint library. An interpretation over $\mathcal{B}$ is a map $I$ that assigns to each constraint atom $b_{ij}$ in $\mathcal{B}$ a value $I(b_{ij})$ in $\{0, 1\}$. A transformation is a map $\phi$ that assigns to each interpretation $I$ over $\mathcal{B}$ the corresponding constraint network $\phi(I)$ defined according to the following condition:

$$c_{ij} \in \phi(I) \iff c_{ij} = \bigcap \{b_{ij'} \in \mathcal{B} : i = i', j = j' \text{ and } I(b_{ij'}) = 1\}.$$ 

The transformation is not necessarily injective. However, it is surjective: for every network $\mathcal{C}$ admissible for $\mathcal{B}$ there exists a corresponding interpretation $I$ such that $\phi(I) = \mathcal{C}$. Indeed, for each constraint $c_{ij}$ in $\mathcal{C}$, consider the set of all constraints $\{b_{ij}^1, \ldots, b_{ij}^k\}$ in $\mathcal{B}$ such that $c_{ij} = b_{ij}^1 \cap \cdots \cap b_{ij}^k$. Set $I(b_{ij}^1) = \cdots = I(b_{ij}^k) = 1$. Then $\phi(I) = \mathcal{C}$.

A literal is either an atom $b_{ij}$ in $\mathcal{B}$, or its negation $\neg b_{ij}$. Notice that $\neg b_{ij}$ is not necessarily a constraint: it merely captures the absence of $b_{ij}$ in the learned network. A clause is a disjunction of literals, and a clausal theory is a conjunction of clauses. An interpretation $I$ is a model of a clausal theory $\mathcal{K}$ if $\mathcal{K}$ is true in $I$ according to the standard propositional semantics. The set of all models of $\mathcal{K}$ is denoted $Models(\mathcal{K})$.

The SAI-based formulation of constraint acquisition is presented as Algorithm 1. The algorithm starts from the empty theory (line 1) and iteratively builds a set of clauses for each received example (lines 2-6). The resulting theory encodes all candidate networks for the constraint acquisition problem.
Algorithm 1: The CONACQ Algorithm

\begin{algorithmic}
  \State {\textbf{input}}: a training set \((E^+, E^-)\) and a constraint library \(B\)
  \State {\textbf{output}}: a set of clauses \(K\)
  \State \(K \leftarrow \emptyset\)
  \For {\text{training example } e}
    \State \(\kappa_e \leftarrow \{b_{ij} \in B : e \text{ does not satisfy } b_{ij}\}\)
    \If {\(e \in E^-\)}
      \State \(K \leftarrow K \land (\bigvee_{b_{ij} \in \kappa_e} b_{ij})\)
    \EndIf
    \If {\(e \in E^+\)}
      \State \(K \leftarrow K \land \bigwedge_{b_{ij} \in \kappa_e} \neg b_{ij}\)
    \EndIf
  \EndFor
  \If {UnitPropagation\(K\) detects \(\perp\)} \Return {\text{"collapsing"}}
\end{algorithmic}

This result is formalised in the next theorem. Let \(B\) be a constraint library and \((E^+, E^-)\) a training set. Then the version space of \((E^+, E^-)\) with respect to \(B\), denoted \(V_B(E^+, E^-)\), is the set of all constraint networks that are admissible for \(B\) and that are consistent with \((E^+, E^-)\).

**Theorem 1 (Correctness).** Let \((E^+, E^-)\) be a training set and \(B\) be a library. Let \(K\) be the clausal theory returned by CONACQ with \(B\) and \((E^+, E^-)\) as input. Then

\[ V_B(E^+, E^-) = \{ \phi(I) : I \in \text{Models}(K) \}. \]

**Proof.** \((\Rightarrow)\) Let \(C\) be a candidate network in \(V_B(E^+, E^-)\). Since \(\phi\) is surjective, there exists an interpretation \(I\) such that \(\phi(I) = C\). Suppose that \(I\) is not a model of \(K\). We show that this leads to a contradiction. If \(I\) is not a model of \(K\) then there is at least one example \(e\) in the training set such that \(I\) falsifies the set of clauses generated from \(e\). Since \(e\) is either positive or negative, two cases must be considered. First, suppose that \(e \in E^+\). In this case, \(I(b_{ij}) = 1\) for at least one atom \(b_{ij}\) in \(\kappa_e\), the set of literals encoding \(e\). By construction of \(\phi(I)\), there must exist a constraint \(c_{ij}\) in \(C\) such that \(c_{ij}\) is contained in \(b_{ij}\). By Lemma 1, \(e\) is a non-solution of \(C\) and hence, \(C\) cannot be a member of \(V_B(E^+, E^-)\). Now, suppose that \(e \in E^-\). By construction, \(I(b_{ij}) = 0\) for each \(b_{ij} \in \kappa_e\). Therefore, there is no constraint \(c_{ij} \in C\) contained in some \(b_{ij}\) such that \(b_{ij}\) rejects \(e\). By contraposition of Lemma 1, \(e\) is a solution of \(C\) and hence, \(C\) cannot be a member of \(V_B(E^+, E^-)\).

\((\Leftarrow)\) Let \(I\) be a model of \(K\) and \(C\) be \(\phi(I)\). Assume that \(C\) is not in \(V_B(E^+, E^-)\). We show that this leads to a contradiction. Obviously, \(C\) must be inconsistent with at least one example \(e\) in the training set. Again, two cases must be considered. Suppose that \(e \in E^+\). Since \(e\) is a non-solution of \(C\) then, by Lemma 1, there exists a pair of constraints \(b_{ij} \in B\) and \(c_{ij} \in C\) such that \(c_{ij} \equiv b_{ij}\) and \(e\) does not satisfy \(b_{ij}\). By construction, \(I(b_{ij}) = 1\). It follows that, \(I\) is not a model of \(\bigwedge_{b_{ij} \in \kappa_e} \neg b_{ij}\). Therefore, \(I\) cannot be a model of \(K\). Now, suppose that \(e \in E^-\). Since \(e\) is a solution of \(C\) then, by contraposition of Lemma 1, there is no pair of constraints \(b_{ij} \in B\) and \(c_{ij} \in C\) such that \(c_{ij} \equiv b_{ij}\) and \(e\) does not satisfy \(b_{ij}\). Therefore, \(I(b_{ij}) = 0\) for each \(b_{ij}\) in \(B\) that rejects \(e\). It follows that \(I\) is not a model of \(\bigvee_{b_{ij} \in \kappa_e} b_{ij}\). Hence, \(I\) cannot be a model of \(C\). \(\square\)

The CONACQ algorithm provides an implicit representation of the version space of the constraint acquisition problem. This representation allows the learner to perform
several useful operations in polynomial time. We conclude this section by examining
the complexity of these operations. In the following, we consider a library $B$ containing
$b$ constraints and a training set $(E^+, E^-)$ containing $m$ examples.

A version space has collapsed if it is empty. In other words, there is no constraint
network $C$ admissible for $B$ such that $C$ is consistent with the training set $(E^+, E^-)$.

**Proposition 1 (Collapse).** The collapsing test takes $O(bm)$ time.

*Proof.* Based on Theorem 1, we know that $V_B(E^+, E^-)$ is empty iff $K$ is unsatisfiable.
The size of $K$ is upper bounded by $b$. Then, the size of $K$ is bounded by $mb$. By con-
struction, $K$ is a dual Horn formula where each clause contains at most one negative
literal. In this setting, unit propagation, which requires $O(K)$ time, is enough to deter-
mine whether $K$ is satisfiable or not [3]. Therefore, the collapsing test can be done in
$O(bm)$ time. □

The membership test involves checking whether or not a constraint network belongs
to the version space of the problem.

**Proposition 2 (Membership).** The membership test takes $O(bm)$ time.

*Proof.* Let $C$ be a constraint network and $I$ an interpretation such that $C = φ(I)$. Based
on Theorem 1, determining whether $C$ belongs to $V_B(E^+, E^-)$ is equivalent to deter-
mining whether $I$ is a model of $K$. Since the size of $K$ is bounded by $mb$, the
membership test takes $O(bm)$ time. □

The update operation involves computing a new version space once a new example
e has been added to the training set.

**Proposition 3 (Update).** The update operation takes $O(b)$ time.

*Proof.* Checking whether a binary constraint is satisfied or violated by an example $e$ is
$O(1)$. The number of such checks is bounded by $b$ (line 3 of Algorithm 1). □

Consider a pair of training sets $(E_1^+, E_1^-)$ and $(E_2^+, E_2^-)$, and their correspon-
ding version spaces $V_B(E_1^+, E_1^-)$ and $V_B(E_2^+, E_2^-)$. The intersection operation requires
computing the version space $V_B(E_1^+, E_1^-) \cap V_B(E_2^+, E_2^-)$. In the following, we assume
that $(E_1^+, E_1^-)$ and $(E_2^+, E_2^-)$ contain $m_1$ and $m_2$ examples, respectively.

**Proposition 4 (Intersection).** The intersection operation takes $O(b(m_1 + m_2))$ time.

*Proof.* Let $K_1$ and $K_2$ be the representations of the version spaces $V_B(E_1^+, E_1^-)$ and
$V_B(E_2^+, E_2^-)$, respectively. In the SAT-based framework, the representation of the
version space $V_B(E_1^+, E_1^-) \cap V_B(E_2^+, E_2^-)$ is simply obtained by $K_1 \land K_2$. □

Finally, given a pair of training sets $(E_1^+, E_1^-)$ and $(E_2^+, E_2^-)$, and their correspon-
ding version spaces $V_B(E_1^+, E_1^-)$ and $V_B(E_2^+, E_2^-)$, we may wish to determine whether
$V_B(E_1^+, E_1^-)$ is a subset of (resp. equal to) $V_B(E_2^+, E_2^-)$.

**Proposition 5 (Subset and Equality).** The subset and equality tests take $O(b^4m_1m_2)$
time.
Proof. Let $K_1$ and $K_2$ be the representations of the version spaces $V_B(E^+_1, E^-_1)$ and $V_B(E^+_2, E^-_2)$, respectively. Based on Theorem 1, we know that determining whether $V_B(E^+_1, E^-_1)$ is a subset of $V_B(E^+_2, E^-_2)$ is equivalent to deciding whether $Models(K_1)$ is a subset of $Models(K_2)$. This is equivalent to deciding whether $K_1$ entails $K_2$. By application of Lemma 5.6.1 from [9], the entailment problem of two Horn or dual Horn formulas $K_1$ and $K_2$ can be decided in $O(|K_1||K_2|)$ time. It follows that the subset operation takes $O(b^2m_1m_2)$ time. For the equality operation, we simply need to check whether $K_1$ entails $K_2$ and $K_2$ entails $K_1$. \[\square\]

4 Exploiting Domain-specific Knowledge

In constraint programming, constraints can be interdependent. For example, two constraints such as $\geq_1$ and $\geq_3$ impose a restriction on the relation of any constraint defined on the scope $(x_1, x_3)$. This is a crucial difference with propositional logic where atomic variables are pairwise independent. As a consequence of such interdependency, some constraints in a network can be redundant. For example, the constraint $\geq_3$ is redundant with $\geq_1$ and $\geq_3$. An important difficulty for the learner is its ability to “detect” redundant constraints. This problem is detailed in the following example.

Example 2. Consider a vocabulary formed by a set of variables $\{x_1, x_2, x_3\}$ and a set of domain values $D = \{1, 2, 3, 4\}$. The learner has at its disposal the constraint library $B = \{\top_{12}, \leq_{12}, \neq_{12}, \geq_{12}, \top_{23}, \leq_{23}, \neq_{23}, \geq_{23}, \top_{13}, \leq_{13}, \neq_{13}, \geq_{13}\}$. We suppose that the target network is given by $\{\geq_{12}, \geq_{13}, \geq_{23}\}$. The training set is given in Table 1. In the third column of the table, we present the growing clausal theory $K$ obtained after processing each example and after performing unit propagation.

After processing each example in the training set, the constraints $\geq_{12}$ and $\geq_{23}$ have been found. Yet, the redundant constraint $\geq_{13}$ has not. For the scope $(x_1, x_3)$ the version space contains four possible networks where $c_{13}$ can alternatively be $\geq_{13}$, $\geq_{13}$, $\neq_{13}$, or $\top_{13}$. In fact, the version space cannot converge to the target concept since it is impossible to find a set of negative examples which would force the learner to reduce its version space. Indeed, in order to converge we would need a negative example $e$ where $e(x_1) < e(x_3)$, $e(x_1) \geq e(x_2)$ and $e(x_2) \geq e(x_3)$. Due to the semantics of inequality constraints, no such example exists. Consequently, the inability for the learner to detect redundancy may hinder the convergence process and hence, can overestimate the number of candidate models in the version space.

As illustrated in the previous example, redundancy is a crucial notion that must be carefully handled if we need to allow version space convergence, or at least if we want to

| Table 1. A set of examples and the corresponding set of clauses $K$ (unit propagated), illustrating the effect of redundancy. |
|---|---|---|---|
| $e_1$ | $4$ | $3$ | $1$ | $(\neg \leq_{12}) \land (\neg \leq_{13}) \land (\neg \leq_{23})$ |
| $e_2$ | $2$ | $3$ | $1$ | $(\neg \leq_{12}) \land (\neg \leq_{13}) \land (\neg \leq_{23}) \land (\geq_{12})$ |
| $e_3$ | $3$ | $1$ | $2$ | $(\neg \leq_{12}) \land (\neg \leq_{13}) \land (\neg \leq_{23}) \land (\geq_{12}) \land (\geq_{23})$ |
have a more accurate idea of which parts of the target network are not precisely learned. The notion of redundancy is formalised as follows. Let \( C \) be a constraint network and \( c_{ij} \) a constraint in \( C \). We say that \( c_{ij} \) is redundant in \( C \) if \( \text{sol}(C \setminus \{c_{ij}\}) = \text{sol}(C) \). In other words, \( c_{ij} \) is redundant if the constraint network obtained by deleting \( c_{ij} \) from \( C \) is equivalent to \( C \).

### 4.1 Redundancy Rules

Any binary constraint \( b_{ij} \) can be seen as a first-order atom \( b(x_1, x_2) \), where \( b \) is a predicate symbol and \( x_1, x_2 \) are variables that take values in the domain \( D \). For example, the constraint \( \leq_{12} \) can be regarded as a first-order atom \( x_1 \leq x_2 \). From this perspective, a constraint network can be viewed as a conjunction of first-order binary atoms. In order to tackle redundancy, we may introduce first-order rules that convey some knowledge about dependencies between constraints. A redundancy rule is a Horn clause:

\[
\forall x_1, x_2, x_3, b(x_1, x_2) \land b'(x_2, x_3) \implies b''(x_3, x_3).
\]

such that for any constraint network \( C \) for which a substitution \( \theta \) maps \( b(x_1, x_2) \), \( b'(x_2, x_3) \) and \( b''(x_1, x_3) \) into in \( C \), the constraint \( b''_{\theta(x_1), \theta(x_3)} \) is redundant in \( C \).

As a form of background knowledge, the learner can use redundancy rules in its acquisition process. Given a library of constraints \( B \) and a set \( R \) of redundancy rules, the learner can start building each possible substitution on \( R \). Namely, for each rule \( b(x_1, x_2) \land b'(x_2, x_3) \implies b''(x_1, x_3) \) and each substitution \( \theta \) that maps \( b(x_1, x_2) \), \( b'(x_2, x_3) \), and \( b''(x_1, x_3) \) to constraints \( b_{ij} \), \( b'_{jk} \) and \( b''_{ik} \) in the library, a clause \( \neg b_{ij} \lor \neg b'_{jk} \lor \neg b''_{ik} \) can be added to the clausal theory \( K \).

**Example 3.** The Horn clause \( \forall x, y, z. (x \geq y) \land (y \geq z) \implies (x \geq z) \) is a redundancy rule since any constraint network in which we have two constraints ‘\( \geq \)’ such that the second argument of the first constraint is equal to the first argument of the second constraint implies the ‘\( \geq \)’ constraint between the first argument of the first constraint and the second argument of the second constraint.

We can apply the redundancy rule technique to Example 2. After performing unit propagation on the clausal theory \( K \) obtained after processing the examples \( \{e^+_1, e^+_5, e^-_3\} \), we know that \( >_{15} \) and \( >_{23} \) have to be set to 1. When instantiated on this constraint network, the redundancy rule from Example 3 becomes \( \geq_{12} \land \geq_{23} \implies \geq_{13} \). Since all literals of the left part of the rule are forced by \( K \) to be true, we can fix literal \( \geq_{13} \) to 1.

The tractability of CONACQ depends on the fact that the clausal theory \( K \) is a dual Horn formula. While we are no longer left with such a formula once \( K \) is combined with the set of redundancy rules \( R \), it is nonetheless the case that satisfiability testing for \( K \land R \) remains tractable: \( K \land R \) is satisfiable iff \( K \) is. The only effect that redundancy rules have is to give an equivalent, but potentially smaller version space for the target network.

### 4.2 Backbone Detection

While redundancy rules can handle a particular type of redundancy, there are cases where applying these rules on the version space is not sufficient to find all redundancies.
Specifically, redundancy rules are only able to discover implications of “conjunctions” of constraints. However, more complex forms of redundancies can arise due to combinations of “conjunctions” and “disjunctions” of constraints. This higher-order form of redundancy is illustrated in the following example.

**Example 4.** Consider the example in Table 2 where the target network comprises the set of constraints \{=_{12}, =_{13}, =_{23}\} and all negative examples differ from the single positive example by at least two constraints. The version space in this example contains 4 possible constraints for each scope, due to the disjunction of possible reasons that would classify the negative examples correctly. Without any further information, particularly negative examples which differ from the positive example by one constraint, redundancy rules cannot restrict the version space any further.

**Table 2.** A set of examples and the corresponding set of clauses \(K\) (unit propagated), illustrating the effect of higher-order redundancy.

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>((\neg =<em>{12}) \land (\neg =</em>{13}) \land (\neg =_{23}))</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>((\neg =<em>{12}) \land (\neg =</em>{13}) \land (\neg =<em>{23}) \land (\geq</em>{13} \lor \geq_{23}))</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>((\neg =<em>{12}) \land (\neg =</em>{13}) \land (\neg =<em>{23}) \land (\geq</em>{13} \lor \geq_{23}) \land (\geq_{12} \lor \geq_{13}))</td>
</tr>
</tbody>
</table>

In Example 4, there is a constraint that is implied by the set of negative examples but redundancy rules are not able to detect. However, all the information necessary to deduce this constraint is contained in the set of redundancy rules and \(K\). The reason for their inability to detect it is that the redundancy rules are in the form of Horn clauses that are applied only when all literals in the left-hand side are true (i.e., unit propagation is performed on these clauses). However, the powerful concept of backbone of a propositional formula can be used here. Informally, a literal belongs to the backbone of a formula if it belongs to all models of the formula [12]. Once the literals in the backbone are detected, they can be exploited to update the version space.

If an atom \(b_{ij}\) appears positively in all models of \(K \land R\), then it belongs to its backbone and we can deduce that \(c_{ij} \subseteq b_{ij}\). Indeed, by construction of \(K \land R\), the constraint \(c_{ij}\) cannot reject all negative examples in \(E^-\) and, at the same time, be more general than \(b_{ij}\). Thus, given a new negative example \(e\) in \(E^-\), we simply need to build the corresponding clause \(\kappa_e\), add it to \(K\), and test if the addition of \(\kappa_e\) causes some literal to enter the backbone of \(K \land R\). The process above guarantees that all the possible redundancies will be detected.

**Example 5.** We now apply this method to Example 4. To test if the literal \(\geq_{13}\) belongs to the backbone, we solve \(R \cup K \cup \{\neg \geq_{13}\}\). If the redundancy rule \(\geq_{12} \land \geq_{23} \rightarrow \geq_{13}\) belongs to \(R\), we detect inconsistency. Therefore, \(\geq_{13}\) belongs to the backbone. The version space can now be refined, by setting the literal \(\geq_{13}\) to 1, effectively removing from the version space the constraint networks containing \(\leq_{13}\) or \(\leq_{13}\).
5 Experiments

We have performed several experiments in order to validate the effectiveness of the CONACQ algorithm and the various approaches to exploiting domain-specific knowledge presented in Section 4. We implemented CONACQ using SAT4J.\footnote{Available from http://www.sat4j.org.} For each experiment, the vocabulary contains 12 variables and 12 domain values per variable. The target constraint networks are sets of binary constraints defined from the set of relations \{\leq, \neq, \geq\}. The learner is not informed about the scope of the constraints, so the available library involves all 66 possible binary constraint scopes. The level of dependency between constraints is controlled by introducing constraint “patterns” of various lengths and type. Patterns are paths of the same constraint selected either from the set \{\leq, \geq\} (looser constraints) or \{<, =, >\} (tighter constraints). For example, a pattern of length \(k\) based on \{<, =, >\} could be \(x_1 > x_2 > \ldots > x_k\). Based on the parameter \(k\) and the type of constraint, we examined 7 types of target networks. In the first, the variables were connected arbitrarily. In the others, we introduce a single pattern of length \(n/3, n/2\) or \(n\), with constraints taken from either \{\leq, \geq\} or \{<, =, >\}. The remaining constraints in the problem were selected randomly.

We ran 100 experiments of each type and report average results in Table 3. The first column specifies the length and type of allowed patterns. The three next columns report the results obtained by the basic algorithm (CONACQ), the algorithm with redundancy rules (CONACQ + rules), and the algorithm with redundancy rules and backbone detection (CONACQ + rules + backbone). Each column is divided in two parts. The left part is the number of models of the formula \(K\). This number is obtained using the binary decision diagram compilation tool \texttt{CLaB}\footnote{Available from http://www-2.cs.cmu.edu/~runej/systems/clab.html.} when \(|V_B|\) is smaller than \(10^4\). An estimate, exponential in the number of free literals in \(K\), is presented otherwise. From Theorem 1, this corresponds to the number of candidate networks encoded in the version space for the acquired problem. The right part measures the average time needed to process an example in seconds on a Pentium IV 1.8 GHz processor. Finally, the last column reports the number of examples needed to obtain convergence of at least one of the algorithms. The threshold on the number of possible examples is fixed to 1000. The training set contains 10\% of positive examples and 90\% of negative examples. We chose such an unbalanced proportion because positive examples are usually much less frequent than negative ones in a constraint network. Negative examples were partial non-solutions to the problem involving a subset of variables. The cardinality of this subset was selected from a uniform distribution over the interval \([2, 5]\).

Based on these results, we can make several important observations. Firstly, we note that the rate of convergence improves if we exploit domain-specific knowledge. In particular, the variant of CONACQ using redundancy rules and backbone detection is able to eliminate all redundant networks in all experiments with patterns. In contrast, the performance of the first two algorithms decreases as the length of redundant patterns increases. This is most clearly noticeable if one compares the top-line of the table, where no redundant pattern was enforced, with the last line in the table, where a pattern of length \(n\) was present, keeping the number of examples constant in both
Table 3. Comparison of the CONACQ variants (CSPs have 12 variables, 12 values, 18 constraints).

| Redundant Pattern | CONACQ | CONACQ +rules | CONACQ +rules +backbone | #E|x |
|-------------------|--------|---------------|----------------------------|------|
| Length | | $V_B$(secs) | $V_B$(secs) | $V_B$(secs) | |
| none | 4.29 $\times 10^9$ (0.11) | 6.71 $\times 10^7$ (0.32) | 1.68 $\times 10^7$ (2.67) | 1000 |
| n/3 | 4.10 $\times 10^9$ (0.11) | 64 (0.31) | 1 (2.61) | 360 |
| n/2 | 1.72 $\times 10^{10}$ (0.11) | 4.10 $\times 10^9$ (0.32) | 1 (2.57) | 190 |
| n | 1.44 $\times 10^{11}$ (0.11) | 2.62 $\times 10^9$ (0.32) | 1 (2.54) | 90 |
| n/3 | 2.68 $\times 10^{10}$ (0.11) | 1.02 $\times 10^8$ (0.32) | 1 (2.60) | 280 |
| n/2 | 7.38 $\times 10^{10}$ (0.11) | 4.19 $\times 10^7$ (0.32) | 1 (2.58) | 170 |
| n | 2.08 $\times 10^{11}$ (0.11) | 6.87 $\times 10^6$ (0.32) | 1 (2.54) | 70 |
| n | 9.01 $\times 10^{12}$ (0.11) | 2.04 $\times 10^4$ (0.32) | 1 (0.24) | 1000 |

cases. When no redundant pattern was enforced, simply combining redundancy rules with CONACQ is sufficient to detect much of the redundancy that is completely discovered by backbone detection. Secondly, we observe that for patterns involving tighter constraints ($\leq$, $\geq$, or $>$), significantly better improvements are obtained as we employ increasingly powerful techniques for exploiting redundancy. Thirdly, we observe that the learning time progressively increases with the type of method used. The basic CONACQ algorithm is about 3 times faster than CONACQ + rules and 25 times faster than CONACQ + rules + backbone. Clearly, there is a tradeoff to be considered between learning-rate and learning time.

6 Related Work

Recently, researchers have become interested in techniques that can be used to acquire constraint networks in situations where a precise statement of the constraints of the problem is not available [4, 10, 14, 15]. The use of version space learning as a basis for constraint acquisition has received some attention from the constraints community [1, 2, 13]. Version space learning [11] is a standard approach to concept learning. A variety of representations for version spaces have been proposed in an effort to overcome the worst-case exponential complexity of version space learning [5–8].

The approach we propose is quite novel with respect to the existing literature on both constraint acquisition and version space learning. We formalise version space learning as a satisfiability problem, which has the advantage of being able to exploit advances in SAT solvers, backbone detection, and unit propagation, to dramatically enhance learning rate. However, it is the fact that incorporating domain-specific knowledge into the acquisition process which gives the approach considerable power.

7 Conclusions

Users of constraint programming technology need significant expertise in order to model their problem appropriately. In this paper we have proposed a SAT-based version space
algorithm that is capable of learning a constraint network from a set of examples and
a library of constraints. This approach has a number of distinct advantages. Firstly,
the formulation is generic, so we can use any SAT solver as a basis for version space
learning. Secondly, we can exploit efficient SAI techniques such as propagation and
backbone detection to improve learning rate. Finally, we can easily incorporate domain-
specific knowledge into constraint programming to improve the quality of the acquired
network. Our empirical evaluation convincingly demonstrated the power of exploiting
domain-specific knowledge as part of the acquisition process.

Acknowledgments. This work was supported by a Ulysses Travel Grant from Enter-
prise Ireland, the Royal Irish Academy and CNRS (Grant Number HR/2003/022). This
work also received support from Science Foundation Ireland under Grant 00/PL.1/C075.

References

1. C. Bessiere, R. Coletta, E.C. Freuder, and B. O’Sullivan. Leveraging the learning power
of examples in automated constraint acquisition. In Proceedings of CP-2004, LNCS 3258,
2. R. Coletta, C. Bessiere, B. O’Sullivan, E.C. Freuder, S. O’Connell, and J. Quinqueton. Con-
3. W.F. Dowling and J.H. Gallier. Linear-time algorithms for testing the satisfiability of propo-
4. E.C. Freuder and R.J. Wallace. Suggestion strategies for constraint-based matchmaker
5. D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant’s learning
9. H. Kleine Büning and T. Lettmann. Propositional Logic: Deduction and Algorithms. Cam-
12. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Tirovansky. Determining com-
cedings of the PRICAI Workshop on Induction of Complex Representations, August 1996.
15. F. Rossi and A. Sperduti. Acquiring both constraint and solution preferences in interactive
1996.