Leveraging the Learning Power of Examples in Automated Constraint Acquisition *

Christian Bessiere[1], Remi Coletta[1], Eugene C. Freuder[2], and Barry O'Sullivan[2]

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

[2] Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{e.freuder|b.osullivan}@4c.ucc.ie

Abstract. Constraint programming is rapidly becoming the technology of choice for modeling and solving complex combinatorial problems. However, users of constraint programming technology need significant expertise in order to model their problem appropriately. The lack of availability of such expertise can be a significant bottleneck to the broader uptake of constraint technology in the real world. In this paper we are concerned with automating the formulation of constraint satisfaction problems from examples of solutions and non-solutions. We combine techniques from the fields of machine learning and constraint programming. In particular we present a portfolio of approaches to exploiting the semantics of the constraints that we acquire to improve the efficiency of the acquisition process. We demonstrate how inference and search can be used to extract useful information that would otherwise be hidden in the set of examples from which we learn the target constraint satisfaction problem. We demonstrate the utility of the approaches on a case-study domain.

1 Introduction

Constraint programming is rapidly becoming the technology of choice for modelling and solving complex combinatorial problems. However, users of constraint programming technology need significant expertise in order to model their problem appropriately. The ability to assist users to model a problem in the constraint satisfaction paradigm is of crucial importance in making constraint programming accessible to non-experts. However, there are many obstacles which must be overcome. For example, in some situations users are not capable of fully articulating the set of constraints they wish to model. Instead users can only present us with example solutions and non-solutions of the target constraint satisfaction problem (CSP) they wish to articulate. This situation arises in many real-world scenarios. In purchasing, a human customer may not be able to provide the sales agent with a precise specification of his set of constraints because

* 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.

he is unfamiliar with the technical terms that are required to specify each constraint. Alternatively, in a data-mining context we may have access to a large source of data in the form of positive and negative examples, and we have been set the task of generating a declarative specification of that data. Earlier work in this area has focused on the generalization problem, inspired by work from the field of Inductive Logic Programming [**?**]. Here we focus on combining techniques from constraint processing and machine learning to develop a novel approach to constraint acquisition.

We have proposed an algorithm, CONACQ , that is capable of acquiring a model of a CSP from a set of examples [2]. The algorithm is based on version space learning [7]. Version spaces are a standard machine learning approach to concept learning. A version space can be regarded as a set of hypotheses for a concept that correctly classify the training data received; in Section 2 we shall present an example which will serve both a pedagogical role and demonstrate the problem we address in this paper.

However, the CONACQ algorithm suffers from a serious malady that has significant consequences for its ability to acquire constraint networks efficiently. In particular, this malady arises because we are acquiring networks of constraints, some of which may be *redundant* [1, 4, 9]. Informally, for now, we can regard a constraint as being redundant if it can be removed from a constraint network without affecting the set of solutions. While redundant constraints have no effect on the set of solutions to a CSP, they can have a negative effect on the acquisition process. In particular, when using version spaces to represent the set of consistent hypotheses for each constraint, redundancy can prevent us from converging on the most specific hypotheses for the target network, eventhough the set of training examples is sufficient for this to occur. As a consequence, for a given constraint in the network, its version space may not be sufficiently explicit, but rather contain constraints which are far too general. This is a significant problem since the size of each version space has a multiplicative effect on the number of possible CSPs that that correctly classify the training examples.

In this paper we present a portfolio of approaches to handling redundant constraints in constraint acquisition. In particular, we address the issue of how to make each constraint as explicit as possible based on the examples given. We shall present an approach based on the notion of *redundancy rules*, which can be regarded as a special-case of relational consistency [5]. We shall show that these rules can eliminate some, but not all, forms of redundancy. We shall then demonstrate a second approach, based on the notion of *backbone detection*, which is far more powerful.

The remainder of this paper is organized as follows. Section 2 presents a simple example of how acquiring redundant constraints can have an adverse effect on the constraint acquisition process. Section 3 presents some formal definitions of the concepts that underpin our approach. We formalize the notion of redundancy in constraint networks, and show how the problem identified in Section 2 can be easily addressed. Section 4 presents a more powerful approach to dealing with redundancy due to disjunctions of constraints. Section 5 presents an empirical evaluation of the various approaches presented in the paper and presents a detailed discussion of our results. A number of concluding remarks are made in Section 6.

2 An Illustrative Example

The CONACQ algorithm maintains a separate version space for each potential constraint in the CSP. A solution to the target CSP (positive example) provides examples for each constraint in the problem, since *all* constraints must be satisfied in order for an example to be classified as positive. However, negative examples are more problematic to process, since violating at least *one* constraint is sufficient for an example to be classified as negative. Therefore, a negative example provides a disjunction of possible examples. It is only when the algorithm can deduce which constraints must have been violated to classify an example as negative are the appropriate version spaces updated. An example below will demostrate this point clearly.

We demonstrate the potential problems that can arise due to redundancy during an interactive acquisition session using CONACQ . Consider the hypothesis space of constraints presented in Figure 1(a). The general-to-specific ordering over the set of constraints is based on set inclusion; more general constraints are placed higher in the hypothesis space. We assume in our example that all constraints in our target problem can be expressed using this hypothesis space. The constraint $\top$ is the universal constraint – all tuples are accepted. The constraint $\perp$ is the null constraint – no tuples are accepted.

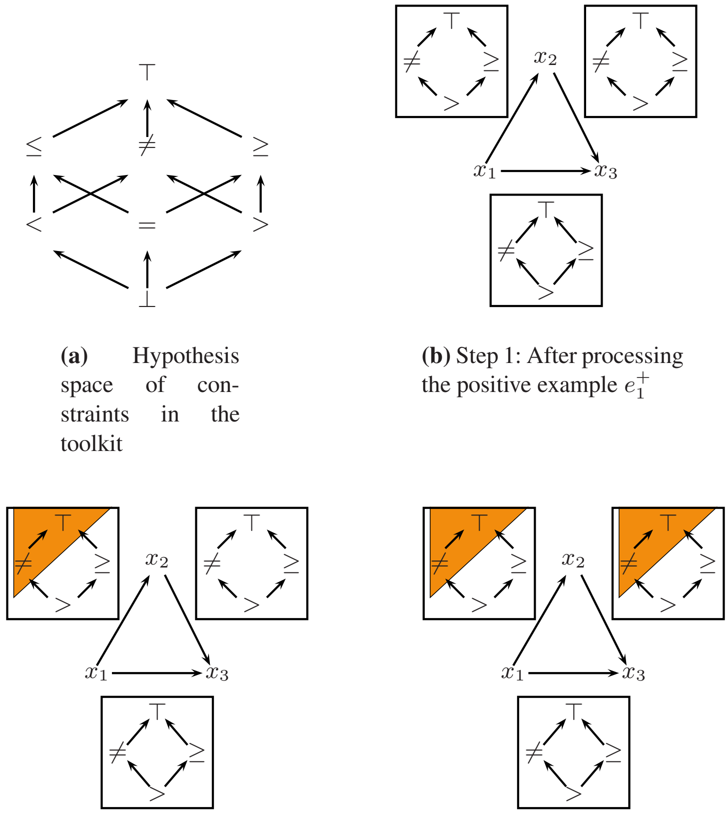
For the purposes of this example, we wish to acquire a CSP involving 3 variables, x_1, x_2 and x_3, with domains $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3, 4\}$. The set of constraints in the target network is $\{x_1 > x_2, x_1 > x_3, x_2 > x_3\}$. Note that it is sufficient only to acquire two of these constraints, the third one being redundant. In Table 1 the set of examples that will be provided to the acquisition system is presented. The set of examples comprises one positive example (a solution to the target CSP) and two negative examples (non-solutions). Figures 1(b)–1(d) illustrate the effect of each example in turn on the version spaces of the constraints in the network.

E	x_1	x_2	x_3
e_1^+	4	3	1
e_2^-	2	3	1
e_3^-	3	1	2

Table 1. Examples for Fig. 1

Figure 1(b) presents the state of each of the constraint version spaces after the first (and only) positive example, e_1^+, has been processed. We can see that the version space of each constraint now contains four hypotheses: $>, \neq, \geq$ and $\top$. The other hypotheses are eliminated because they are inconsistent with e_1^+. Specifically, if $x_1 = 4 \wedge x_2 = 3$ can be part of a solution, then the constraint between these variables must be *more general than or equal to* $>$. Therefore, we can ignore the possibility that this constraint can be either $=$, $<$, $\leq$ or $\perp$. Essentially. we know that any CSP that can be expressed in terms of the constraints presented in Figure 1(a) must comprise constraints that are no more specific than those in the version spaces presented in Figure 1(b). Similar reasoning allows us to reduce the version space for each constraint to that illustrated in Figure 1(b).

Figure 1(c) presents the effect of processing example e_2^-, the first negative example. Of the three constraints in the problem, e_2^- differs by only one constraint, c_{12}, compared to the constraints implied by the the positive example e_1^+. Therefore, we can further refine the version space of constraint c_{12} by removing both $\neq$ and $\top$. We illustrate this as using a colored shading over those hypotheses that are removed from the version space. Similarly, the reason why e_3^- is classified as negative is due to a single constraint: namely c_{23}. Figure 1(d) illustrates the result of processing negative example e_3^-.



(a) Hypothesis space of constraints in the toolkit

(b) Step 1: After processing the positive example e_1^+

(c) Step 2: After processing the negative example e_2^-

(d) Step 3: After processing the negative example e_3^-

Fig. 1. Acquiring a redundant constraint prevents one version space from converging.

After processing the negative examples e_2^-, e_3^-, the version spaces for the constraint between variables x_1 and x_2 and between variables x_2 and x_3 are reduced to the set of hypotheses $\{>, \geq\}$. However, the version space for the constraint between variables x_1 and x_3 has not. Instead, this version space contains four possible hypotheses: $\{>, \geq, \neq, \top\}$.

This is an unfortunate since we cannot now find a set of negative examples which will help this version space to reduce any closer to the target constraint. For example, to eliminate the hypothesis $\neq$, we need a negative example with $x_1 < x_3$ but necessarily satisfying all other acquired constraints, i.e., satisfying their most specific possible alternative: $x_1 > x_2$ and $x_2 > x_3$, so that the only possible reason to reject it is $x_1 < x_3$. Clearly no such example exists. As a consequence, our constraint acquisition algorithm cannot converge any further. However, it should be pointed out that it was not due to a deficiency in our set of examples that precluded convergence in this case, but as a result of attempting to acquire redundant constraints using the CONACQ algorithm. Specifically, in our example the constraint between x_1 and x_3 is redundant.

Therefore, it is clear that redundant constraints can prevent us from converging on the most specific hypotheses consistent with a set of examples. However, by exploiting the fact that we are acquiring constraint networks, we can rely on various search and inference techniques to help us leverage the learning power of the examples that have been provided to us. In the ideal, helping us to converge on the target hypothesis much more quickly. In the next section we present an approach to handling redundant constraints which would have overcome the problem we have experienced in this example.

3 Redundancy Rules

In this section, we introduce formal definitions of the basic concepts used in this paper. We then propose definitions of redundancy and redundancy rules, before presenting an approach to dealing with redundant constraints in the CONACQ acquisition process.

3.1 Basic definitions

A finite *constraint network* N consists of a finite set of variables $X = \{x_1, \ldots, x_n\}$, a set of domains $D = \{D(x_1), \ldots, D(x_n)\}$, where the domain $D(x_i)$ is the finite set of values that variable x_i can take, and a set of constraints $C = \{c_1, \ldots, c_m\}$. Each constraint c_i is defined by the ordered set $var(c_i)$ of the variables it involves, and a set $sol(c_i)$ of allowed combinations of values. An assignment of values to the variables in $var(c_i)$ *satisfies* c_i if it belongs to $sol(c_i)$. A *solution* to a constraint network is an assignment of a value from its domain to each variable such that every constraint in the network is satisfied. When all the constraints in C involve exactly 2 variables, we say that the constraints and the network are *binary*. This is the case we will study in the rest of the paper since it greatly simplifies notation. We will use $c(x_i, x_j)$ and c_{ij} interchangeably to refer to $sol(c)$ where $var(c) = (x_i, x_j)$. However, all the results are essentially the same for constraints of any arity.

As seen in the previous section, redundancy is a crucial notion that we need to tackle if we want to speed up version space convergence during the constraint acquisition process.

Definition 1 (Redundancy) *Given a constraint network $N = (X, D, C)$, we say that a constraint $c \in C$ is **redundant** wrt N iff the set of solutions of N is the same as the set of solutions of $N_{-c} = (X, D, C \setminus \{c\})$. We note $N_{-c} \models c$.*

3.2 Redundancy in CONACQ

The CONACQ algorithm has been proposed in [2]. Its inputs are a set X of variables with their domains, a set of examples $E = E^+ \cup E^-$, and a bias B. An example $e \in E$ is an assignment of values to variables from X that must be a solution of the target constraint network (if $e \in E^+$) or non solution (if $e \in E^-$).

The bias is composed of constraint scopes (sets of variables on which a constraint c has to be guessed), attached with a set of constraint types that are the different possibilities for $sol(c)$. In the simplest case, where we guess a complete network of binary constraints, the bias contains all pairs of variables from X as possible scopes, attached with all the binary constraint types available in the toolkit. The set of possible constraints on (x_i, x_j) is denoted by its bias, B_{ij}.

The output of CONACQ is any constraint network that has the same set X of variables with their domains, and a set of constraints chosen from the bias such that every element of E^+ is solution and none from E^-. Since the number of constraint networks satisfying these criteria during the acquisition process can be huge (exponential), CONACQ uses version space techniques and maintains only a most specific bound S_{ij} and a most general bound G_{ij} for each pair of variables (x_i, x_j) belonging to the bias. Any constraint in the toolkit subsumed by G_{ij} and subsuming S_{ij} is a candidate for c_{ij} (namely, belongs to the version space).

Theorem 1. *Let X, D, B, E be the input of* CONACQ *. Let $c_{ij} \in B_{ij}$. If there exists $\{c_{ik}, c_{kj}\} \in B_{ik} \times B_{kj}$ such that $E \models \{c_{ik}, c_{kj}\}$ and c_{ij} is redundant wrt $(X, D, \{c_{ik}, c_{kj}\})$ then the version space cannot shrink its bounds on (x_i, x_j) more than $S_{ij} = c_{ij}$ and $G_{ij} = \top$.*

Proof. Let $c'_{ij} \in B_{ij}$ a constraint subsumed by c_{ij}. Suppose there exists $e \in E^-$ such that e violates c'_{ij}. (This is the only way to remove c'_{ij} from the version space.) We can decrease the local general bound G_{ij} under c'_{ij} only if no other constraint in the version space can reject e. Now, we know that $E \models \{c_{ik}, c_{kj}\}$. Hence, when e is presented, we are guaranteed that c_{ik} and c_{kj} are still higher than their respective lower bounds S_{ik} and S_{kj} (otherwise E would cause some version spaces to collapse, and we could infer what we want on S_{ij} and G_{ij}). If e violates c'_{ij}, it also violates c_{ij} since c'_{ij} is subsumed by c_{ij}. It thus violates $\{c_{ik}, c_{kj}\}$ since c_{ij} is redundant wrt $(X, D, \{c_{ik}, c_{kj}\})$. As a result, we cannot decide that c'_{ij} is the necessary culprit for e's rejection since there exists constraints between S_{ik} and c_{ik}, and between S_{kj} and c_{kj}, which are both in the version space, and could reject e. So, G_{ij} cannot decrease under $\top$.

Regarding S_{ij}, it will increase higher than c_{ij} if and only if there exists $e \in E^+$ that violates c_{ij}. However, if e violates c_{ij}, it also violates $\{c_{ik}, c_{kj}\}$ (see above), which contradicts the assumption that $E \models \{c_{ik}, c_{kj}\}$. □

3.3 Formal definition of redundancy rules

A constraint in a constraint network can be seen as a *constraint type* (or first order predicate) in which we substitute network variables for variables in the predicate. For example, the generic predicate $P(s, t) =$ '$s < t$' of arity $n(P) = 2$ can produce the constraint $x_1 < x_2$ in a constraint network involving x_1 and x_2, or the constraint $y_3 < y_5$ in another constraint network.

Since the process of modeling a problem is usually done using a given constraint toolkit, it seems reasonable to study the concept of redundancy with respect to the set of constraint types available in that toolkit. Let us first define the concept of redundancy rule for general constraint types.

Definition 2 (Redundancy rule) *Let T be a set of constraint types. The Horn clause*

$$\forall t_1, \dots t_n \bigwedge_i P_i(t_{i_1} \dots t_{i_{n(P_i)}}) \models Q(t_{j_1} \dots t_{j_{n(Q)}})$$

with $P_i \in T$ $\forall i$, and $Q \in T$, is a ***redundancy rule*** *wrt T iff there is at least one variable t_{j_h} in Q that appears in some P_i, and for any constraint network N for which a substitution*[1] *θ maps the rule into N, we have*

$$N_{-\theta(Q)} \models \theta(Q).$$

If $|\{P_i\}| = k$, we say that the rule is a k-redundancy rule.

We immediately focus our attention on redundancy rules in a binary constraints setting where, if in addition we work on a complete network of binary constraints, it is sufficient to deal with 2-redundancy rules [6].

Definition 3 (Binary redundancy rule) *Let T be a set of constraint types of arity 2. A* ***binary redundancy rule*** *is a redundancy rule wrt T of the form:*

$$\forall t_1, t_2, t_3, P_1(t_1, t_2) \wedge P_2(t_2, t_3) \models Q(t_1, t_3).$$

Example 1 The Horn clause $\forall x, y, z.(x \geq y) \wedge (y \geq z) \models (x \geq z)$ is a binary 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 subsumes the '$\geq$' constraint between the first argument of the first constraint and the second argument of the second constraint.

Given the set T of constraint types available in a toolkit, redundancy rules can be built for the toolkit independently of the problem we will acquire. Thus, redundancy rules can be included as part of the constraint toolkit, in much the same way as propagators are often included in constraint toolkits, at least for the most common constraints.

3.4 Redundancy rules in CONACQ

We saw in Theorem 1 that it can sometimes occur that the local version space for the constraint between a pair of variables (x_i, x_j) can reach a state where it becomes impossible to make its general bound more specific (thus reducing its size) because it contains a constraint that is redundant with respect to the other constraints already learned by CONACQ . To avoid this problem, we can simply trigger the relevant redundancy rule from the toolkit each time its left-hand side is true, namely the rule becomes "*active*" in a version space.

Definition 4 (Active Redundancy Rule) *Given a binary rule $R = P_1(t_1, t_2) \wedge P_2(t_2, t_3) \models Q(t_1, t_3)$, a version space V, and a mapping θ substituting variables of V for variables in R, we say that R is* ***active*** *in V wrt θ if $P_1(\theta(t_1), \theta(t_2))$ is subsumed by $G(\theta(t_1), \theta(t_2))$, and $P_2(\theta(t_2), \theta(t_3))$ is subsumed by $G(\theta(t_2), \theta(t_3))$*

[1] As in most toolkits, we require that θ is 'locally' injective, namely two different t_{i_h}'s in the same P_i cannot map on the same network variable.

Definition 5 (Satisfying a Redundancy Rule) *Let θ be a mapping substituting variables of a version space V for variables in a rule $R = P_1(t_1, t_2) \wedge P_2(t_2, t_3) \models Q(t_1, t_3)$. We say that R is satisfied on V wrt θ if $Q(\theta(t_1), \theta(t_3))$ is subsumed by $G(\theta(t_1), \theta(t_3))$.*

Thus, when a rule R is active with respect to a mapping θ, we can force it to be satisfied (or *apply* it) by modifying the general bound of the constraint on which θ maps its right hand side. This modification does not affect the set of possible networks admitted by the version space. We state this more formally in Definition 6 and Theorem 2.

Definition 6 (Version Space Equivalence) *Let V and V' be two version spaces defined on the same variables and bias. We say that V and V' are* ***equivalent*** *iff for any constraint network N obtained by picking a constraint between S_{ij} and G_{ij} for each (x_i, x_j) in V there exists a constraint network N' obtained the same way from V' such that N and N' have the same solutions.*

Theorem 2. *Let V be a version space. Let V' be the version space obtained after a rule R has been applied to V. If R was active on V, then V' and V are equivalent.*

Proof. Suppose there exists a constraint network N in V for which none of the constraint networks in V' have the same set of solutions. This means that the constraint r_{ij} added by the rule R has decreased the general bound G'_{ij} in V'. The constraints allowed by G'_{ij} all reject some solution of N (by assumption). This is necessarily due to r_{ij}. Thus, r_{ij} cannot be redundant wrt N. By definition of what an active redundancy rule is, we deduce that R cannot be active in V, which contradicts the assumption. □

This property guarantees that we can safely apply all the redundancy rules that are active, reducing the size of the version space while its semantics is not affected.

The complexity of applying all the binary rules in a version space is in $O(m \times |\mathcal{B}|^2)$, with $|\mathcal{B}|$ the number of constraint scopes in the bias and m the number of binary rules in the toolkit. For k-redundancy rules this is in $O(m \times |\mathcal{B}|^k)$. Applying k-redundancy rules to a constraint network is a relaxation of relational k-consistency [5]. However, relational k-consistency requires space exponential in the number of variables in the redundant constraint while in our approach we only generate constraints from the toolkit, thus keeping constant space for each constraint.

Example 2 We now apply the method above to example of Figure 1. After processing the examples $\{e_1^+, e_2^-, e_3^-\}$, we know that even in the loosest constraint network still possible, we have $x_1 \geq x_2$ and $x_2 \geq x_3$. Therefore, the rule described in Example 1 is active. By applying it, we can reduce the possible constraint types between x_1 and x_3 to $\{>, \geq\}$.

4 Higher-Order Redundancy

While redundancy rules can eliminate a particular type of redundancy, there are cases where applying these rules on the version space is not sufficient to find all redundancies. Redundancy rules are well-suited to discovering constraints that are redundant because of conjunctions of other constraints. However, as we shall show in Section 4.1, a constraint can be redundant because of a conjunction of *disjunctions* of constraints. We refer to this as higher-order redundancy. Since our redundancy rules are in the form of Horn clauses, they cannot tackle such redundancies. After a brief description of the way CONACQ stores the information about negative examples, we will show how to tackle these complex redundancies.

4.1 Another Example

In the scenario illustrated in Figure 2, we use the same set of variables and domains as those used in the example presented in Section 2. However, in this case the target network comprises the set of constraints $\{x_1 = x_2, x_1 = x_3, x_2 = x_3\}$. Furthermore, in this example all negative instances differ from e_1^+ by *at least* two constraints (see the table in Figure 2).

E	x_1	x_2	x_3
e_1^+	2	2	2
e_2^-	3	3	4
e_3^-	1	3	3

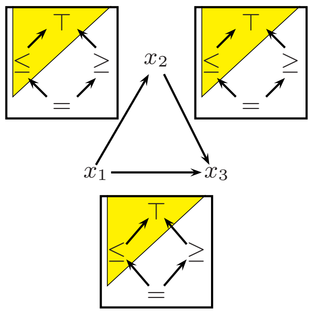


Fig. 2. None of the version spaces have converged.

After processing the positive example e_1^+, each version space contains four consistent hypotheses, the most specific hypothesis in each being $=$. The version spaces are depicted in Figure 2. However, each of the negative examples does not contain enough information to immediately reduce any of the version spaces for our constraints any further. For example, negative example e_2^- may be negative because of either constraint c_{12} or c_{23}, or indeed both. Therefore, none of the version spaces of the constraints in our example can be reduced further (indicated with dark shading in Figure 2, as opposed to the lighter shade used earlier to depict hypotheses being removed from a version space). The version spaces in this example each contain 4 hypotheses 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, no further restrictions can be made on the version spaces of the constraints in our problem. Consequently, none of the version spaces converge. Simply applying redundancy rules also does not help. An alternative approach is required, which will be presented next.

4.2 Storing negative examples in CONACQ

As briefly described above, when a negative example e^- is presented to CONACQ , it is encoded as a clause $cl_e = l_{ij}^{U_{ij}} \vee \ldots \vee l_{km}^{U_{km}}$ where U_{ij} is the set of most general constraint types available for c_{ij} that reject e^- (i.e., that are violated by e^-). The literal $l_{ij}^{U_{ij}}$ is true if any possible constraint type for c_{ij} in its local version space is at least as specific as the given bound U_{ij}. This is the case if any constraint type r in the general bound G_{ij} of c_{ij} is at least as specific as a constraint type in U_{ij}. This is the condition for c_{ij} to reject e^-. Hence, the clause cl_e means that at least one of the constraints c_{ij} having a literal $l_{ij}^{U_{ij}}$ in cl_e has to be at least as specific as its U_{ij} to reject e^-.

We should point out that a clause does not necessarily contain a literal for each constraint we have to find in the bias. Each constraint c_{ij} for which the specific bound S_{ij} is already more general than U_{ij} will not reject e^-. It is then useless to put a literal for it in the clause since this literal will be forced to be false. For example, if $e_k^- = \{x_1 = 1; x_2 = 1; x_3 = 3\}$ and $S_{12} = \{\geq\}$, c_{12} cannot reject e_k^-. In addition, not all elements of E^- have a stored clause in CONACQ . It can indeed appear that an example is already definitively rejected by some constraint in the version space. For example, take again the e_k^- above and imagine $G_{23} = \{\geq\}$. e_k^- cannot satisfy c_{23}. Hence, it is useless to add a clause in CONACQ to express that e_k^- should be rejected.

The set of all the clauses containing the necessary information about E^- is denoted by $\mathcal{K}$. Since a constraint network assigns a single constraint c_{ij} to each pair of variables (x_i, x_j), it leads to an interpretation for every literal $l_{ij}^{U_{ij}}$ in $\mathcal{K}$. By construction, it is guaranteed that for any constraint network leading to a satisfying interpretation for $\mathcal{K}$, all $e^- \in E^-$ are non-solutions. (See [2] for more details.)

4.3 Finding higher-order redundancies

In the example in Section 4.1 we have seen a case where a constraint is implied by the set of negative examples received by CONACQ , but redundancy rules are not able to detect this by themselves. However, all the information necessary to deduce this constraint is contained in the set of redundancy rules and the set $\mathcal{K}$ of clauses encoding the negative examples. The reason for their inability to detect it is that rules are in the form of Horn clauses that we apply only when *all* predicates in the left-hand side are true (i.e., we apply unit propagation on these clauses). To tackle this issue we can build the set $\mathcal{R}$ of all possible substitutions on the given bias for available rules. For each rule $R = P_1(t_1, t_2) \wedge P_2(t_2, t_3) \models Q(t_1, t_3)$, for each substitution θ that maps P_i's and Q on possible constraints in the bias, a clause $\neg l^{P_1}_{\theta(t_1),\theta(t_2)} \vee \neg l^{P_2}_{\theta(t_2),\theta(t_3)} \vee l^{Q}_{\theta(t_1),\theta(t_3)}$ is added to the set $\mathcal{R}$. This process can be done as soon as the bias is given, before the beginning of the acquisition process.

In addition, since the semantics of a literal l_{ij}^U is: 'c_{ij} is at least as specific as U', we need also to link literals involving the same constraint scope. For example, if we have $l_{ij}^{>}$ true, then a literal $l_{ij}^{\geq}$ should not be able to take the value false. Hence, we need a third set of clauses, the set $\mathcal{L}$ containing $\neg l_{ij}^U \vee l_{ij}^{U'}$ for each pair (x_i, x_j) such that U subsumes U'. These subsumption clauses between two literals l_{ij}^U and $l_{ij}^{U'}$ need only to be included if l_{ij}^U appears in $\mathcal{K}$ and subsumes $l_{ij}^{U'}$ that appears in $\mathcal{R}$. Adding subsumption clauses between two literals in $\mathcal{K}$ would not activate any more rules. This is an important property since the fact that $l_{ij}^{U'}$ comes from $\mathcal{R}$ implies that $|U'| = 1$, which ensures polynomial space for $\mathcal{L}$.

We now have a base of ground clauses, $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$, that contains all available information about rules and negative examples. If a literal l_{ij}^{U} in $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$ appears positively in all models of $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$ (i.e., it belongs to the backbone [8]), we can reduce the local version space of c_{ij} to constraints at least as specific as U. By construction of $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$, it is indeed impossible to assign c_{ij} to a constraint more general than U and at the same time reject all negative instances in E^-.

Therefore, after the presentation of a new negative instance e from E^-, we have to build the corresponding clause cl_e, add it to $\mathcal{K}$, update $\mathcal{L}$ if necessary, and test if the addition of cl_e causes some literal[2] to enter the backbone of $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$.

The process that we described above guarantees that all the possible redundancies will be detected.

Theorem 3. *Given a version space V, a set $E = E^+ \cup E^-$ of examples, a constraint type r, and the sets $\mathcal{K}, \mathcal{R}, \mathcal{L}$ built as described above, if r is a possible constraint on (x_i, x_j) and r can be inferred from V, the set of rules of the toolkit, and E^-, then the literal l_{ij}^{r} is a member of the backbone of $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$.*

Proof. Let r be a (most specific) possible constraint on (x_i, x_j) that can be inferred from V, the set of rules of the toolkit, and E^-. Suppose l_{ij}^{r} does not belong to the backbone of $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$. By assumption, r is the head of some rules in the toolkit (otherwise CONACQ by itself can learn r on (x_i, x_j)). Then, l_{ij}^{r} is the head of a subset $\mathcal{R}'$ of the rules in $\mathcal{R}$. Then there exists a model M of $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$ for which none of the rules $R \in \mathcal{R}'$ has all the literals of its tail set to true. There are two cases. Either none of the networks N_M built from M allow a solution violating r on (x_i, x_j), which means that a rule that would infer l_{ij}^{r} from M is missing in $\mathcal{R}$, or some N_M allows solutions violating r on (x_i, x_j), which means that r cannot be inferred since there exists a network rejecting all E^- (by construction of N_M), and allowing solutions rejected by r on (x_i, x_j). Both cases contradict the assumption. Finally, if r was not the most specific constraint that could be learned on (x_i, x_j) (for example $r =$'$\leq$' while $l_{ij}^{<}$ was inferred) the proof holds for the most specific constraint r', and the clauses added to $\mathcal{L}$ permit to infer l_{ij}^{r} from $l_{ij}^{r'}$. $\square$

However, this process is quite expensive from a computational point of view, since testing if a literal belongs to the backbone of a formula is a coNP-complete problem. This prevents the use of such a technique on big formulae, but as we are concerned with an interactive acquisition process, it is reasonable to assume that the version spaces we need to handle will be small enough to permit a human user to deal with them, and consequently we expect that the speed-of-response for backbone detection will be acceptable. The experimental section will discuss this feature more deeply.

[2] Note that a literal is a candidate to enter the backbone only if it appears in the right-hand side of a Horn clause from $\mathcal{R}$ (or it belongs to a unary clause, obviously). Furthermore, the backbone cannot contain negative literals since $\mathcal{R}$ and $\mathcal{L}$ are Horn bases and $\mathcal{K}$ contains only positive clauses.

Example 3 We now apply the above method to the example presented in Section 4.1. The set $\mathcal{R}$ of redundancy rules used in this example is presented in Table 2. It provides a subset of possible binary rules associated with the bias in Figure 1(a). As presented previously, the set $\mathcal{L}$ is built dynamically only when required by $\mathcal{K}$ and $\mathcal{R}$, so we initialize it to $\emptyset$.

Table 2. Binary redundancy rules for the sample problem

r_1	$l_{12}^{\geq} \wedge l_{23}^{\geq} \models l_{13}^{\geq}$
r_2	$l_{13}^{\geq} \wedge l_{23}^{\leq} \models l_{12}^{\geq}$
r_3	$l_{13}^{\leq} \wedge l_{23}^{\geq} \models l_{12}^{\leq}$
r_4	$l_{12}^{\leq} \wedge l_{23}^{\leq} \models l_{13}^{\leq}$
r_5	$l_{12}^{\leq} \wedge l_{13}^{\geq} \models l_{23}^{\geq}$
r_6	$l_{12}^{\geq} \wedge l_{13}^{\leq} \models l_{23}^{\leq}$

After receiving e_2^-, $\mathcal{K} = \{(l_{13}^{\geq} \vee l_{23}^{\geq})\}$, to apply technique presented in Section 4.3, we test if either $l_{13}^{\geq}$ or $l_{23}^{\geq}$ belong to the backbone of $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$. However, running a SAT solver allows us to determine that both $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L} \cup \{\neg l_{13}^{\geq}\}$ and $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L} \cup \{\neg l_{23}^{\geq}\}$ have solutions. Since the backbone detection did not find any literal, at this stage, we cannot deduce anything more than using previous methods.

However, after receiving e_3^-, $\mathcal{K} = \{(l_{13}^{\geq} \vee l_{23}^{\geq}), (l_{12}^{\geq} \vee l_{13}^{\geq})\}$ we detect the new backbone. We run a SAT solver on $\mathcal{R} \cup \mathcal{K} \cup \mathcal{L} \cup \{\neg l_{13}^{\geq}\}$ and because of the minimal conflict set $\mathcal{K} \cup \{r_1\} \cup \{\neg l_{13}^{\geq}\}$, it fails. Therefore, $l_{13}^{\geq}$ belongs to the backbone and we can use this to refine the version space of constraint c_{13}, removing from it the constraint types $\leq$ and $\top$.

In this example, it is clear that the backbone detection on $\mathcal{K} \cup \mathcal{R} \cup \mathcal{L}$ has permitted us to detect (and learn) a redundant constraint that redundancy rules alone did not.

5 Empirical Study

To compare the approaches to exploiting redundancy to improve the quality of the acquired CSP that we have proposed in this paper, we studied their effects on a sample class of CSP. The bias used in this experiment is the same as that presented in Figure 1(a). Our experiments involved generating target CSPs, which we then attempted to acquire by presenting examples of solutions and non-solutions of them to an acquisition system based on either (we also provide the label used in Table 3 to identify that configuration of the acquisition system) (a) CONACQ on its own (CONACQ *standard*); (b) CONACQ using redundancy rules only (CONACQ $+rules$); (c) CONACQ using both redundancy rules and backbone detection (CONACQ $+rules + backbone$). These form the columns in Table 3.

In each case we computed a representable set of solutions (non-solutions) to the target CSP which were used as a source of positive (respectively, negative) examples for the acquisition system. We generated target CSPs with 12 variables, 12 values in each

Table 3. Comparison of the capability of each acquisition system to exploit redundancy in the larger problem studied (12 variables, 12 values, 30 constraints).

Redundant Pattern *Length* $\{constraints\}$		CONACQ *standard* $\|VS\|$ (secs)	CONACQ +*rules* $\|VS\|$ (secs)	CONACQ +*rules* +*backbone* $\|VS\|$ (secs)	$\#Exs$
none		4.29×10^9 (< 1)	6.71×10^7 (3)	1.68×10^7 (46)	$> 10^3$
n/3	$\{\leq, \geq\}$	4.10×10^3 (< 1)	64 (2)	1 (29)	360
n/2	$\{\leq, \geq\}$	1.72×10^{10} (< 1)	4.10×10^3 (2)	1 (23)	190
n	$\{\leq, \geq\}$	1.44×10^{17} (< 1)	2.62×10^5 (2)	1 (21)	90
n/3	$\{=, <, >\}$	2.68×10^8 (< 1)	1.02×10^3 (2)	1 (27)	280
n/2	$\{=, <, >\}$	7.38×10^{19} (< 1)	4.19×10^7 (2)	1 (23)	170
n	$\{=, <, >\}$	2.08×10^{34} (< 1)	6.87×10^{10} (2)	1 (21)	70
n	$\{=, <, >\}$	5.07×10^{30} (< 1)	1.07×10^9 (3)	1 (40)	$> 10^3$

domain, 30 constraints and varied the degree of redundancy in them. Clearly, during the acquisition process it is not known between which variables there are constraints so we must assume a complete graph comprising 66 constraints, giving us 66 local version spaces.

The number of examples used in each experiment was equal to the number required for CONACQ using both redundancy rules and backbone detection to converge. However, we set a maximum number of examples at just above 10^3 after which we would terminate the acquisition process.

For each acquisition system setup (the 3 different configurations of CONACQ), we recorded the total time (in seconds, *secs*) required to process the set of examples and the final size of the version space, denoted by $|VS|$. The number of examples is denoted by $\#Exs$ in the last column. We present averages of 10 runs of the experiment.

We have studied the effects of controlling the redundancy in each CSP in two ways (giving us the rows in Table 3). Firstly, we introduced patterns of constraints in the target network of various lengths. In the experiment we used lengths based on the number of variables in the problem: specifically, we use lengths n, $n/2$ and $n/3$ (*Length* column). Secondly, for each length of pattern we selected a pattern of constraints with controlled characteristics and introduced these into the target network. In the experiment we selected patterns of the same constraint selected either from the set $\{\leq, \geq\}$ (looser constraints) or $\{<, =, >\}$ (tighter constraints). For example, a path of length n based on $\{<, =, >\}$ is $x_0 > x_1 > x_2 > x_3 > x_4 > x_5 > x_6 > x_7$, while a path of length $n/2$ based on $\{\leq, \geq\}$ is $x_0 \leq x_1 \leq x_2 \leq x_3$ (see $\{constraints\}$ column). As a "straw-man" we also present results for a target CSP where no pattern was introduced into the network. Our results are presented in Table 3.

It can be clearly seen from Table 3 that, in terms of the size of the resultant version spaces, exploiting redundancy rules with CONACQ improves upon CONACQ alone in all situations. However, exploiting redundancy rules leads to an increase in the amount of time required by the the acquisition to process the set of examples since it relies on the construction of $\mathcal{R}$, derived from the set of negative examples. Combining backbone detection and redundancy rules with CONACQ improves upon CONACQ with redundancy rules, in terms of version space size, but offers a considerably slower response time due to the use of the SAT solver[3].

Furthermore, we can see that as the level of redundancy in the target problem increases, from $n/3$ to n, regardless of the constraints involved in the redundant pattern, the ability of standard CONACQ to converge deteriorates dramatically. It is also interesting to note that CONACQ with redundancy rules also does progressively worse on these networks. 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 cases.

Simply combining redundancy rules with CONACQ is sufficient to detect much of the redundancy that is completely discovered by backbone detection. Specifically, comparing the standard CONACQ column with the CONACQ $+rules$ column, we can see that there are orders-of-magnitude differences in the size of the version spaces, with a very minor increase in processing time (approximately double in most cases). Note that CONACQ $+rules + backbone$ requires and order of magnitude more time, but achieves convergence.

Finally, the effect of the tightness of the redundancy pattern introduced into the problem has interesting consequences. On a target network involving looser redundant patterns, those from $\{\leq, \geq\}$, positive instances play a central role in the acquisition of the problem. Specifically, more of them are required for convergence. Furthermore, after receiving positives instances, each local version space is smaller than would be the case if the redundant patterns were made up of the tighter constraints from $\{=, <, >\}$. For example, when $\geq$ is the target, $\{\geq, \top\}$ is the largest possible version space, while for $>$ it is $\{>, \neq, \geq, \top\}$. This explains the exponential difference in version space size between tighter and looser target networks presented in Table 3.

In summary, this experimental evaluation demonstrates that CONACQ on its own is insufficient to fully exploit redundancy during the acquisition process and that convergence may not be possible. The more sophisticated approaches that we propose based on redundancy rules and backbone detection are far superior. However, there is a trade-off to be considered between the need to find the tightest specification on the target network versus the response time of the acquisition system.

We have seen that adding backbone detection and redundancy rules together to enhance CONACQ is best in terms of convergence, but has a high response time cost. Just exploiting redundancy rules with CONACQ offers a very fast response time, with the abilities to converge quite significantly also. Obviously, it is an application-specific and/or problem-specific issue how this tradeoff should be dealt with. For example, in an interactive context, speed-of-response is a critical factor and, therefore, simply relying on redundancy rules seems to be an ideal compromise. In such an application, backbone detection could be run as a background process, further refining the version spaces that represent the target CSP.

---

[3] The SAT solver used for backbone detection is `zchaff`, version `2003.12.04`, `http://ee.princeton.edu/~chaff/zchaff.php`.

6 Conclusions and Future Work

In this paper we were concerned with automating the formulation of constraint satisfaction problems from examples of solutions and non-solutions. We have combined techniques from the fields of machine learning and constraint programming. In particular we have presented a portfolio of approaches to exploiting the semantics of the constraints that we acquire to improve the efficiency of the acquisition process.

We have demonstrated that the CONACQ algorithm on its own is insufficient to fully exploit redundancy during the acquisition process. The more sophisticated approaches that we propose based on redundancy rules and backbone detection are far superior. However, there is a tradeoff to be considered between the need to find the tightest specification on the target network versus the response time of the acquisition system. We have seen that adding backbone detection and redundancy rules together to enhance CONACQ is best but has a high response time cost, while just exploiting redundancy rules with CONACQ offers a very fast response time, with the abilities to converge quite significantly towards the target CSP.

Our future work in this area will look at a number of important issues that must be addressed in real-world acquisition contexts. For example, techniques for handling noise and errors in the process are of critical importance also, particularly if human users are providing the training examples [**?**].

References

1. C.W. Choi, J.H.M. Lee, and P.J. Stuckey. Propagation redundancy in redundant modelling. In *Proceedings of CP-2003*, volume LNCS 2833, pages 229–243. Springer, September 2003.
2. R. Coletta, C. Bessiere, B. O'Sullivan, E.C. Freuder, S. O'Connell, and J. Quinqueton. Constraint acquisition as semi-automatic modeling. In *Proceedings of AI'03*, pages 111–124, December 2003.
3. R. Coletta, C. Bessiere, B. O'Sullivan, E.C. Freuder, S. O'Connell, and J. Quinqueton. Semi-automatic modeling by constraint acquisition. In *Proceedings of CP-2003, Short paper, LNCS 2833, Springer Kinsale, Cork, Ireland.*, pages 812–816, September 2003.
4. A. Dechter and R. Dechter. Removing redundancies in constraint networks. In *Proceedings of AAAI-87*, pages 105–109, 1987.
5. R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173(1):283–308, 1997.
6. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.
7. T. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.
8. R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Ttroyansky. Determining computational complexity from characteristic 'phase transition'. *Nature*, 400:133–137, July 1999.
9. B.M. Smith. Succeed-first or fail-first: A case study in variable and value ordering. Technical Report 96.26, School of Computer Studies, University of Leeds, September 1996.