Constraint Acquisition*

Christian Bessiere
University of Montpellier, France

Frédéric Koriche
University of Artois, France

Nadjib Lazaar
University of Montpellier, France

Barry O'Sullivan
University College Cork, Ireland

Abstract

Constraint programming is used to model and solve complex combinatorial problems. The modeling task requires some expertise in constraint programming. This requirement is a bottleneck to the broader uptake of constraint technology. Several approaches have been proposed to assist the non-expert user in the modelling task. This paper presents the basic architecture for acquiring constraint networks from examples classified by the user. The theoretical questions raised by constraint acquisition are stated and their complexity is given. We then propose Conacq, a system that uses a concise representation of the learner's version space into a clausal formula. Based on this logical representation, our architecture uses strategies for eliciting constraint networks in both the passive acquisition context, where the learner is only provided a pool of examples, and the active acquisition context, where the learner is allowed to ask membership queries to the user. The computational properties of our strategies are analyzed and their practical effectiveness is experimentally evaluated.

1 Introduction

Over the last forty 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 [2, 15, 32].

*This paper is based on material published in [5, 7, 8, 9]. It additionally formalizes the several problems addressed in these papers and proves several important complexity results that remained open.

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 on x_1 must be less or equal than the value on x_2. A solution of the constraint network is an assignment of variables to domain values that satisfies every constraint in the network. The Constraint Satisfaction Problem (CSP) is hence the problem of finding a solution for a given constraint network.

However, the construction of constraint networks still remains limited to specialists in the field. Actually, it has long been recognized that modeling a combinatorial problem in the constraint formalism requires significant expertise in constraint programming [19, 20, 34]. Such a level of knowledge precludes novices from being able to use constraint networks 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, we propose to *acquire* a constraint network from a set of examples. This approach is exemplified by the following scenario. Suppose that a human user would like to build a constraint network in order to solve a series of repetitive tasks. Usually, the tasks are instances of the same combinatorial problem and only differ in the set of domains associated with variables. For example, in a time-tabling problem, it could be the problem of assigning a teacher and a time slot to every courses given to all classes, regardless of the actual time-slots at which teachers are available for the current semester. In practice, the user has already solved several instances of the problem without the help of a solver and knows how to classify an example as a solution or a non-solution to it. Based on these considerations, the overall aim of constraint acquisition is to induce from examples a general constraint network that adequately represents the target problem. This approach allows us to use the learned network with different initial domains in order to solve further tasks supplied by the user.

In a nutshell, the constraint acquisition process can be regarded as an interplay between the user and the learner. The user has in mind a target problem but does not know how this problem can be modeled as an efficient constraint network. Yet, the user has at her disposal a set of solutions (positive examples) and non-solutions (negative examples) of the problem. For its part, the learner has at its disposal a set of variables on which the examples are defined and constraint language. The overall goal of the learner is to induce a constraint network that uses combinations of constraints defined from the language and that is consistent with the solutions and non-solutions provided by the user.

There are two constraint acquisition approaches that naturally emerge from this vision. In *passive* constraint acquisition, the learner cannot ask queries of the user. In this setting, the key goal of the learner is to inform the user about the state of its "version space", that is, the set of networks that are consistent with the pool of examples. For example, the learner can be required to determine whether its version space has converged in order to inform the user that no more examples are required to capture the target problem. By

contrast, in *active* constraint acquisition, the learner can choose an example and ask whether it is a solution, or not, of the target problem. As a helpful teacher, the user supplies the correct response. In this setting, the goal of the learner is to find a short sequence of membership queries that rapidly converges towards the target problem.

What we call *user* in this paper is not necessarily a *human* user. For instance, in [29], the learner tries to acquire a constraint network representing the sequences of elementary operations that constitute a valid action for a robot. The classification of actions as positive or negative depends on the success or failure of the action on the robot simulator. The simulator can be ran as much as we want to produce examples. However, in standard constraint acquisition, classifying examples requires an answer from a human user. Hence, we should seek to minimise the size of the training set required to acquire a target problem.

In this paper we formally define the main constraint acquisition problems related to passive and active acquisition. Several important complexity results are provided. We show for instance that it is polynomial to decide if the version space still contains networks consistent with the examples. Testing convergence is, however, coNP-complete. We also show that in the context of active constraint acquisition, constraint networks are not learnable in general with a polynomial number of membership queries. In order to solve the different problems that arise from constraint acquisition, we develop a constraint acquisition architecture, named Conacq, which is based on a compact and efficient representation of version spaces into clausal formulas. This basic component is equipped with different methods, inspired from SAT techniques, used to handle consistency, convergence and identification problems in passive and active constraint acquisition. The complexity of our methods is analyzed.

The necessary background in constraint programming and concept learning is introduced in Section 2. Section 3 gives a specification of the different constraint acquisition problems examined in this study and analyses their complexity. In Section 4, we present a technique for representing version spaces and we describe Conacq.1, a passive acquisition algorithm using that representation. Section 5 shows how to use background knowledge to improve the learning process. Active acquisition is presented in Section 6 together with an algorithm to generate good queries, that is, queries that allow to learn the target problem with as few queries as possible. Experiments are reported in Section 7. Finally, we compare our framework with related work in Section 8, and conclude this work in Section 9.

2 Background

In this section, we introduce some useful notions in constraint programming and concept learning.

2.1 Vocabulary and Constraint Networks

In the field of constraint programming, combinatorial problems are represented as constraint networks. We consider here a single domain for all the variables in our networks. This really is a notational convenience as the actual domain of a variable can be restricted by the constraint relations. It is possible to think of our unified domain as the union of all of the actual problem domains.

We first define what we call a *vocabulary*. Intuitively, a vocabulary specifies the space of all complete assignments of the variables from their domain.

Definition 1 (Vocabulary)**.** *A* vocabulary *is a pair* $\langle X, D\rangle$ *such that* X *is a finite set* $\{x_1, \cdots, x_n\}$ *of variables, and* D *is a finite subset of* $\mathbb{Z}$ *called the domain.*

We will assume that the learner starts from a prefixed vocabulary; the variables and the domain are known to the learner, and its goal is merely to acquire a set of constraints over this vocabulary. This justifies our slightly non-standard definition of constraint network.

Definition 2 (Constraint Network)**.** *A* constraint network *over a given vocabulary* $\langle X, D\rangle$ *is a finite set* C *of* constraints*. Each constraint* c *in* C *is a pair* $\langle var(c), rel(c)\rangle$*, where* $var(c)$ *is a sequence of variables of* X*, called the* constraint scope *of* c*, and* $rel(c)$ *is a relation over* $D^{|var(c)|}$*, called the* constraint relation *of* c*. For each constraint* c*, the tuples of* $rel(c)$ *indicate the allowed combinations of simultaneous value assignments for the variables in* $var(c)$*. The* arity *of a constraint* c *is given by the size* $|var(c)|$ *of its scope.*

For the sake of clarity, we will often take examples using binary constraints, that is, constraints with a scope involving two variables. With a slight abuse of notation, we use c_{ij} to refer to the binary relation that specifies which pairs of values are allowed for the sequence $\langle x_i, x_j\rangle$. For example, $\leq_{12}$ denotes the constraint specified on $\langle x_1, x_2\rangle$ with relation "less than or equal to".

The *complement* of a constraint c is the constraint denoted $\overline{c}$ such that $var(\overline{c}) = var(c)$ and $rel(\overline{c}) = D^{|var(c)|} \setminus rel(c)$. In other words, $\overline{c}$ and c describe complementary relations defined on the same scope. For example, $>_{12}$ is the complement of $\leq_{12}$.

Given a vocabulary $\langle X, D\rangle$, an *assignment* is a vector $\boldsymbol{x} = \langle \mathtt{x}_1, \cdots, \mathtt{x}_n\rangle$ in $D^{|X|}$. An assignment $\boldsymbol{x}$ maps to each variable $x_i \in X$ a corresponding domain value $\mathtt{x}_i \in D$. An assignment $\boldsymbol{x}$ *satisfies* a constraint c if the projection of $\boldsymbol{x}$ onto the scope $var(c)$ is a member of $rel(c)$. An assignment $\boldsymbol{x}$ *violates* a constraint c, or equivalently, the constraint c *rejects* $\boldsymbol{x}$, if $\boldsymbol{x}$ does *not* satisfy c. An assignment $\boldsymbol{x}$ *satisfies* a constraint network C if $\boldsymbol{x}$ satisfies every constraint c in C. Such an assignment is called a *solution* of C. A *non-solution* of C is an assignment that violates at least one constraint from C.

The set of all solutions of a constraint network C is denoted $sol(C)$. C is *satisfiable* if $sol(C) \neq \varnothing$, and *unsatisfiable* otherwise. If $sol(C) \subseteq sol(C')$ for two constraint networks C and C', then we say that C *entails* C'. Finally, if $sol(C) = sol(C')$, then we say that C and C' are *equivalent*.

2.2 Constraint Language and Bias

Borrowing from [14], a constraint language is a set of relations that restricts the type of constraints that are allowed when modeling a network.

Definition 3 (Constraint Language)**.** *A* constraint language *is a set* $\Gamma = \{r_1, \cdots, r_t\}$ *of* t *relations over some subset of* $\mathbb{Z}$*.*

In this study, we shall concentrate on constraint languages of *fixed* arity; for such languages, the arity of any relation occurring in a constraint language Γ is bounded by a constant k. Based on this assumption, global constraints will *not* be considered in this paper. Recall that global constraints, such as *alldifferent*, are relations defined for any (unbounded) arity. Though our framework does not prevent their use, the fact that a single global constraint leads to an exponential number of possible constraints, one for each subset of X for a vocabulary $\langle X, D\rangle$, makes most learning tasks analysed in this paper non-polynomial.

Definition 4 (Extension)**.** *Given a prefixed vocabulary* $\langle X, D\rangle$*, the* extension *of* Γ *on* $\langle X, D\rangle$ *is the set* $\mathbf{B}_\Gamma$ *of all constraints* c *for which* $var(c)$ *is a tuple of variables of* X*, and there exists a relation* r_i *in* Γ *such that* $rel(c) = r_i \cap D^{|var(c)|}$*.*

We are now ready to define the *bias*[1] for the acquisition problem, that is, the set of all possible constraints that are candidate for being in the constraint network to be learned.

Definition 5 (Bias)**.** *Given a prefixed vocabulary* $\langle X, D\rangle$*, the* bias *for the learning task is a set* $\mathbf{B} \subseteq \mathbf{B}_\Gamma$*.*

The bias $\mathbf{B}$ can be equal to the extension $\mathbf{B}_\Gamma$ of Γ on the vocabulary. But it can also be a strict subset if we have some initial information on the problem to be elicited. With these notions in hand, any constraint network C defined over Γ is simply a subset of $\mathbf{B}_\Gamma$. Note that the extension of a constraint language of bounded arity k is always polynomial in the input dimension. Indeed, the size of $\mathbf{B}_\Gamma$ is bounded by $n^k t$ in the general case, and by $n^2 t$ in the setting of binary constraint networks. By contrast, if Γ was containing global constraints, the size of $\mathbf{B}_\Gamma$ would no longer be polynomial. A single global constraint gives rise to 2^n possible constraints in $\mathbf{B}_\Gamma$.

Example 1. Consider the binary constraint language $\Gamma = \{\leq, \neq, \geq\}$ over $\mathbb{Z}$. Given the vocabulary defined by $X = \{x_1, x_2, x_3\}$ and $D = \{1, 2, 3, 4\}$, we observe that the constraint network $C = \{\leq_{12}, \geq_{12}, \leq_{23}, \neq_{23}\}$ is indeed a subset of the extension $\mathbf{B}_\Gamma = \{\leq_{12}, \neq_{12}, \geq_{12}, \leq_{13}, \neq_{13}, \geq_{13}, \leq_{23}, \neq_{23}, \geq_{23}\}$. ♢

2.3 Concept Learning

In inductive learning, it is generally assumed that learning algorithms operate over some concept class which captures the space of concepts that the learner

[1] This is called the *declarative bias* in [31].

can potentially generate over all possible sets of examples. In the setting of constraint acquisition, the concept class is defined according to a bias $\mathbf{B}$, which specifies the constraints that are allowed for modeling the target constraint network.

Given a prefixed vocabulary $\langle X, D\rangle$, a *concept* is a Boolean function over $D^{|X|}$, that is, a map that assigns to each assignment $\boldsymbol{x}$ a value in $\{0, 1\}$. Given two concepts f and g, and a set A of assignments, we shall write $f \subseteq g$ if $f^{-1}(1) \subseteq g^{-1}(1)$, and $f \subseteq A$ if $f^{-1}(1) \subseteq A$. A *representation* of a concept f is a constraint network C for which $f^{-1}(1) = sol(C)$. A concept f is said to be *representable* by a bias $\mathbf{B}$ if there is a subset C of $\mathbf{B}$ such that C is a representation of f. We denote by f_C a concept represented by a set of constraints C. The *concept class* of $\mathbf{B}$, denoted $\mathbf{C_B}$, is the set of all concepts that are representable by $\mathbf{B}$. For instance, the concept class $\mathbf{C_B}$ defined over the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$ is the set of all Boolean functions that are representable by inequality and disequality constraints.

An *example* is a pair $e = \langle \boldsymbol{x}(e), y(e)\rangle$ where $\boldsymbol{x}(e)$ is an assignment and $y(e)$ is a value in $\{0, 1\}$. If $y(e) = 1$, e is called a positive example and, if $y(e) = 0$, e is called a *negative example*. A *training set* is a set $E = \{e_1, \cdots, e_m\}$ of examples. A concept f is *consistent* with an example e if $f(\boldsymbol{x}(e)) = y(e)$. By extension, a concept f is consistent with a training set E if f is consistent with every example in E.

Example 2. Given again the vocabulary defined by $X = \{x_1, x_2, x_3\}$ and $D = \{1, 2, 3, 4\}$ and the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$, let us examine the concepts f_1 and f_2 represented by the networks $C_1 = \{=_{12}, <_{23}\}$ and $C_2 = \{=_{12}, \leq_{23}\}$, respectively. Here, $=_{12}$ is an abbreviation of $\{\leq_{12}, \geq_{12}\}$ and $<_{23}$ is an abbreviation of $\{\leq_{23}, \neq_{23}\}$. Notice that $f_1, f_2 \in \mathbf{C_B}$. Suppose that we are given the training set E specified in the following table.

	x_1	x_2	x_3	y
e_1	2	2	4	1
e_2	1	3	3	0
e_3	1	1	1	0

We can easily verify that f_1 is consistent with E whereas f_2 is inconsistent with E because e_3 is not consistent with f_2. ♢

Based on these notions, we are now in position to introduce the useful notion of version space, adapted from [27] and [24].

Definition 6 (Version Space/Collapsing/Convergence)**.** *Given a bias* $\mathbf{B}$ *and a training set* E*, the* version space *of* E *with respect to* $\mathbf{B}$*, denoted* $\mathbf{C_B}(E)$*, is the set of all concepts in* $\mathbf{C_B}$ *that are consistent with* E*. We say that the version space of* E *has* collapsed *if* $\mathbf{C_B}(E) = \varnothing$*. We say that the version space of* E *has* converged *if* $|\mathbf{C_B}(E)| = 1$*, i.e.* $\mathbf{C_B}(E)$ *is reduced to a singleton.*

Borrowing the terminology of [18, 22], any version space is a "convex poset", which basically means that for any subset $F = \{f_1, f_2, \cdots, f_p\}$ of concepts in

$\mathbf{C_B}$ such that $f_1 \subseteq f_2 \subseteq \cdots \subseteq f_p$, if the extreme elements f_1, f_p are members of $\mathbf{C_B}(E)$, then the whole chain F must be included in $\mathbf{C_B}(E)$. This in turn means that any finite version space can be characterized by its minimal and maximal elements, with respect to set inclusion.

Definition 7 (Maximally Specific/General)**.** *Given a bias* $\mathbf{B}$ *and a training set* E*, a concept* $f \in \mathbf{C_B}(E)$ *is* maximally specific *(resp.* maximally general*) if there is no* $f' \in \mathbf{C_B}(E)$ *such that* $f' \subset f$ *(resp.* $f \subset f'$*). A constraint network* $C \subseteq \mathbf{B}$ *is* maximally specific *(resp.* maximally general*) in* $\mathbf{C_B}(E)$*, if* C *is a representation of a maximally specific (resp. maximally general) concept* $f \in \mathbf{C_B}(E)$*.*

Example 3. Consider again Example 2. The network $C_1 = \{=_{12}, <_{23}\}$ is maximally specific in $\mathbf{C_B}(E)$. Any other constraint from $\mathbf{B}$ either is implied by C_1 or rejects the positive example e_1 from E. The network $C_2 = \{\neq_{23}\}$ is maximally general in $\mathbf{C_B}(E)$. We cannot remove any constraint from C_2 and still reject all negative examples. The network $C_3 = \{\geq_{12}, \neq_{13}\}$ is also a maximally general constraint network of $\mathbf{C_B}(E)$. $\diamondsuit$

3 Constraint Acquisition

In this section, we introduce several constraint acquisition problems that arise from the conjunction of concept learning and constraint programming. We provide the complexity of each of these problems.

3.1 Interaction Between User and Learner

The problem of acquiring a constraint network from examples can be viewed as an interactive process between two protagonists: the human *user* and a virtual *learner*. The user has in mind a target problem but does not know how to represent this problem into a constraint network. An interaction between the learner and the user is performed to acquire a network representing the target problem.

There are two natural forms of interaction which emerge from the paradigm of constraint learning. In *passive* acquisition, the user provides classified examples and the learner is passive because it has no control on the series of supplied examples. By contrast, in *active* acquisition, the learner is active because it can guide the exploration of its version space by asking queries to the user.

In machine learning, Angluin [1] defines several types of queries. An *equivalence query* requests the user to decide whether a given concept is equivalent to the target, and in case of negative answer, to provide an example showing the discrepancy between the given concept and the target. A *membership query* requests the user to classify a given example as positive or negative. In the setting of constraint acquisition, asking a user equivalence queries is unreasonable, especially because our starting assumption is that the user is not able to articulate the constraints of the target network directly. (Equivalence queries

will be useful to characterize complexity of learning.) Membership queries are, however, a reasonable interaction process where the user is just asked to be able to recognize elements of her target concept.

3.2 Problems Related to Constraint Acquisition

The basic problem related to passive acquisition is to find whether the version space of the examples supplied by the user has collapsed, or not. If the version space has not collapsed, the learner is naturally required to provide a concept consistent with the examples.

Definition 8 (Consistency Problem)**.** *Given a bias* $\mathbf{B}$ *and a training set* E*, the* consistency problem *is to determine whether* $\mathbf{C_B}(E) \neq \varnothing$*. If the answer is "yes", then a representation* C *of some concept in* $\mathbf{C_B}(E)$ *must be returned.*

In the consistency problem, the learner only picks a concept in its version space and returns a representation of it to the user. A more informative task is to determine whether the version space has converged, or not. If this is indeed the case, the user knows that she has an exact characterization of her target problem, and hence, she does not need to supply more examples to the learner.

Definition 9 (Convergence Problem)**.** *Given a bias* $\mathbf{B}$ *and a training set* E*, the* convergence problem *is to determine whether* $\mathbf{C_B}(E)$ *has converged. If the answer is "yes", then a representation* C *of the target concept in* $\mathbf{C_B}(E)$ *must be returned.*

Recall that in passive acquisition, the learner cannot control the dynamics of its version space. By contrast, in *active* acquisition, the learner is allowed to present queries to the user. Depending on the type TQ of query, the user is not requested the same information. With the equivalence query $EQ(C)$, the learner asks the user whether the network C is equivalent to its target problem or not. If not, the user must return a counter-example to the learner that is, a solution of C that is not solution of her target problem, or a solution of her target problem that is not solution of C. With the membership query $MQ(\boldsymbol{x})$, the learner asks the user what is the label of $\boldsymbol{x}$. The user answers 1 if $\boldsymbol{x}$ is a solution of the target problem, and 0 otherwise. Starting from a bias, an initial training set, and a type TQ of query (among membership query and equivalence query), the goal is to find a polynomial sequence of queries leading to a network representing the target problem.

Definition 10 (Identification Problem/Learnability)**.** *Given a bias* $\mathbf{B}$*, a training set* E *on a vocabulary* $\langle X, D\rangle$*, and a type* $TQ \in \{membership, equivalence\}$ *of queries, the* identification problem *for a target concept* f *representable by* Γ *is to find a sequence* $\langle q_1, \cdots, q_m\rangle$ *of queries of type* TQ *leading to the detection of a constraint network* C *representing* f*. If the length* m *of the sequence is polynomial in the number* $|\mathbf{B}|$ *of constraints in the bias, we say that* f *is* learnable *by queries of type* TQ*.*

3.3 Complexity of Constraint Acquisition

We now give the complexity of the constraint acquisition problems defined in the previous subsection.

Theorem 1. *The consistency problem can be solved in polynomial time.*

Proof. Given a bias $\mathbf{B}$ and a training set E, we compute the set C of constraints from $\mathbf{B}$ that are not violated by any positive example in E. This is done by traversing all positive examples from E and removing from $\mathbf{B}$ all constraints violated by such an example. We then check whether there exists a negative example in E satisfying C. This is done by traversing the negative examples and performing $|C|$ constraint checks for each of them. If a negative example satisfying C is found, we return "collapse", otherwise we return "yes" and C as a witness. □

One might be tempted to believe that a version space has converged precisely when there is exactly one constraint network representing the version space. The convergence problem would then be tractable. It would be sufficient to compute the maximal subset C of $\mathbf{B}$ satisfying all positive examples, to check that it rejects all negative examples, and to prove that for every $c \in C$, there exists a negative example solution of $C \setminus \{c\}$. However, in most constraint languages, constraint relations are *interdependent*, as illustrated in the following example.

Example 4. As in Example 2, consider the vocabulary $X = \{x_1, x_2, x_3\}$ and $D = \{1, 2, 3, 4\}$ and the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$. The target network is $C = \{\leq_{12}, \leq_{13}, \leq_{23}\}$. Suppose we are given the training set E specified in the following table.

	x_1	x_2	x_3	y
e_1	1	2	3	1
e_2	4	4	4	1
e_3	1	2	1	0
e_4	3	1	3	0

At the beginning $\mathbf{B} = \{\leq_{12}, \neq_{12}, \geq_{12}, \leq_{13}, \neq_{13}, \geq_{13}, \leq_{23}, \neq_{23}, \geq_{23}\}$. After processing example e_1 we know that constraints $\geq_{12}, \geq_{13}$, and $\geq_{23}$ cannot belong to the target network C because they are violated by e_1, which is positive. After processing e_2 similarly the remaining possible constraints in the bias $\mathbf{B}$ are: $\leq_{12}, \leq_{13}$, and $\leq_{23}$. By processing e_3 we know that the constraint $\leq_{23}$ is in the target network as it is the only one rejecting e_3. Similarly, $\leq_{12}$ is in the target network as it is the only one rejecting e_4. Hence, there exists two constraint networks consistent with the training data E, namely $C = \{\leq_{12}, \leq_{23}\}$ and $C' = \{\leq_{12}, \leq_{13}, \leq_{23}\}$. However, the version space $\mathbf{C_B}(E)$ contains a single concept and thus has converged because C and C' have the same solutions. ♢

On this example we see that any concept may have multiple representations because of the interdependency of constraints (i.e., the constraint $\leq_{13}$ is redundant with $\leq_{12}$ and $\leq_{23}$). Consequently, even if the learner's version space has

converged, it may still have more than one representative network. As shown by the following result, the convergence problem is much harder than the consistency problem.

Theorem 2. *The convergence problem is coNP-complete.*

Proof. First, observe that the problem is in coNP. A polynomial certificate is composed of the vocabulary $\langle X, D\rangle$, the bias $\mathbf{B}$, the set E of examples, and optionally two networks C_1 and C_2 and an assignment $\boldsymbol{x}$ over $D^{|X|}$. The certificate is valid if and only if the version space $\mathbf{C_B}(E)$ is either empty or includes the two concepts f_1 and f_2, associated with C_1 and C_2, and the assignment $\boldsymbol{x}$ belongs to $f_1 \setminus f_2$. By Theorem 1, checking whether $\mathbf{C_B}(E) = \varnothing$ takes polynomial time. Checking whether $f_1, f_2 \in \mathbf{C_B}(E)$ also takes polynomial time. For each positive example e^+ in E we just need to check whether e^+ belongs to $sol(C_1)$ and to $sol(C_2)$, which is done by checking whether e^+ satisfies all constraints in C_1 and all constraints in C_2. For each negative example e^- in E we need to check whether e^- belongs neither to $sol(C_1)$ nor to $sol(C_2)$, which is done by checking whether there exists a constraint in C_1 and a constraint in C_2 rejecting e^-. Finally, checking whether $f_1(\boldsymbol{x}) = 1$ and $f_2(\boldsymbol{x}) = 0$ can be done by testing whether $\boldsymbol{x}$ satisfies all constraints in C_1 and $\boldsymbol{x}$ violates at least one constraint in C_2.

We now prove that the convergence problem is complete for coNP. Recall that 3-Col is the problem of deciding if a graph (N, U) is 3-colorable, where $|N| = n$. Let $\langle X, D\rangle$ be the vocabulary formed by the variables $X = \{x_1, \ldots, x_n\}$, where x_i represents the node i in N, and the domain $D = \{0, 1, \mathsf{r}, \mathsf{g}, \mathsf{b}\}$, where r, g and b are the *colors*. The bias is $\mathbf{B} = \mathbf{B}_\Gamma$, with $\Gamma = \{r_{\neq}, r_{01}, r_4\}$, where $r_{\neq}$ is the binary relation $D^2 \setminus \{(\mathsf{r}, \mathsf{r}), (\mathsf{g}, \mathsf{g}), (\mathsf{b}, \mathsf{b})\}$, r_{01} is the binary relation $(D \times \{0\}) \cup (\{0\} \times D) \cup \{(1, 1)\}$, and r_4 is the quaternary relation formed by the set $\{0, \mathsf{r}, \mathsf{g}, \mathsf{b}\}^4 \cup \{0, 1\}^4$ from which we remove all 4-tuples containing two 0s and two different colors, all 4-tuples containing no 1s and exactly one 0, and the tuple $(1, 1, 1, 1)$.

Any constraint network C over $\mathbf{B}$ is composed of binary constraints $c_{\neq}(x_i, x_j)$ such that $rel(c_{\neq}) = r_{\neq}$, and/or binary constraints $c_{01}(x_i, x_j)$ such that $rel(c_{01}) = r_{01}$, and/or quaternary constraints $c_4(x_i, x_j, x_k, x_l)$, such that $rel(c_4) = r_4$.

We now build the training set E as follows:

- for every 4-tuple i, j, k, l in $\{1, \cdots, n\}$, let e^-_{ijkl} be the negative example, where $\boldsymbol{x}(e^-_{ijkl})$ is the n-tuple containing 1 in positions i, j, k, l, and 0s everywhere else,
- for every pair $\{i, j\} \in U$ with $i < j$, let e^-_{ij} be the negative example where $\boldsymbol{x}(e^-_{ij})$ is the n-tuple containing only 0s except in positions i and j where it contains b,
- for every pair $\{i, j\} \notin U$ with $i < j$, let e^+_{ij} be the positive example where $\boldsymbol{x}(e^+_{ij})$ is again the n-tuple containing only 0s except in positions i and j where it contains b.

Note that $c_4(x_i, x_j, x_k, x_l)$ is the only constraint that rejects the example e^-_{ijkl}. We are thus guaranteed that all c_4 for any quadruple of variables belong to all representations of consistent concepts. Moreover, for each pair $\{i, j\} \in U$, $c_{01}(x_i, x_j)$ and $c_{\neq}(x_i, x_j)$ are the only constraints that reject the example e^-_{ij}. So we know that any representation of a consistent concept contains either $c_0(x_i, x_j)$ or $c_{\neq}(x_i, x_j)$. Finally, because of e^+_{ij} we know that any representation of a concept in the version space excludes $c_0(x_i, x_j)$ and $c_{\neq}(x_i, x_j)$ for each pair $\{i, j\} \notin U$. Hence, the constraint network S containing all constraints c_4 in $\mathbf{B}$, and the constraints $c_{\neq}(x_i, x_j)$ and $c_{01}(x_i, x_j)$ for all pairs $\{i, j\} \in U$, is guaranteed to be the maximally specific network of E with respect to Γ.

Based on these considerations, the version space has not converged if and only if there exists a constraint network C such that f_C is in the version space and $sol(S) \subset sol(C)$.

Let $\boldsymbol{x}$ be a member of $sol(C) \setminus sol(S)$. Suppose that $\boldsymbol{x}$ contains a 1. We know that all c_4 constraints belong to C. So, $\boldsymbol{x}$ contains only 0s and 1s. Yet, because $r_{\neq}$ and r_{01} accept all tuples in $\{0, 1\}^2$, $\boldsymbol{x}$ would necessarily be solution of S, a contradiction. So $\boldsymbol{x}$ contains only 0s and colors. Obviously, $\boldsymbol{x}$ must contain less that $n-1$ 0s because any tuple with a number of 0s greater than or equal to $n-1$ is a solution of S. Suppose that $\boldsymbol{x}$ contains $n-2$ 0s, and colors at positions i and j. Then x_i and x_j must have the same color in $\boldsymbol{x}$ because otherwise any c_4 involving x_i and x_j would be violated. If the pair $\{i, j\}$ is not in U, then $\boldsymbol{x}$ is already a solution of S because it satisfies the same constraints as e^+_{ij}. So, $\{i, j\}$ must be an edge in U, but since C rejects any negative example e^-_{ij}, either $c_{01}(x_i, x_j)$ or $c_{\neq}(x_i, x_j)$ belongs to C. In any case, this implies that x_i and x_j cannot take the same colors, a contradiction. So $\boldsymbol{x}$ must have at least three entries i, j, k taking colors. But if there is a 0 at an arbitrary position l of $\boldsymbol{x}$, the c_4 involving x_i, x_j, x_k and the variable x_l taking 0 rejects the tuple. As a result, $\boldsymbol{x}$ contains only colors. Since all c_{01} are violated by $\boldsymbol{x}$, it follows that all $c_{\neq}(x_i, x_j)$ with $\{i, j\} \in U$ are satisfied by $\boldsymbol{x}$, and hence, $\boldsymbol{x}$ is a 3-coloring of (N, U).

Conversely, any 3-coloring $\boldsymbol{x}$ of (N, U) is not a solution of S but is a solution of any network C containing all c_4 and all $c_{\neq}(x_i, x_j)$ for $\{i, j\} \in U$. Clearly, the concept of C is consistent with E. So, if (N, U) is 3-colorable, the version space has not converged.

To summarize, the version space has converged if and only if the problem of coloring (N, U) with three colors has no solution. Our construction contains $O(n^4)$ constraints and $O(n^4)$ examples, and hence, it is polynomial in the size of (N, U). Therefore, the convergence problem is coNP-complete. □

Theorem 3. *Given any bias* $\mathbf{B}$ *and any target concept* f *representable by* $\mathbf{B}$*,* f *is learnable by equivalence queries.*

Proof. The following algorithm guarantees that we find a network representing the target concept with a polynomial number of equivalence queries. Remember that $EQ(C)$ returns an assignment in $(f \cup sol(C)) \setminus (f \cap sol(C))$.

```
1 C ← B
2 e ← EQ(C)
3 while e ≠ true do
4     for c ∈ C | x(e) ⊭ c do  C ← C \ {c}
5     e ← EQ(C)
6 return C
```

Soundness. The algorithm is sound because it returns C only when $EQ(C)$ has returned true, which means that C is a representation of the target concept f. *Termination in polynomial number of queries.* Let C^f be any network representing f over Γ, that is, $sol(C^f) = f^{-1}(1)$. The property "$C^f \subseteq C$" is true when we start the loop in Line 3 because $C^f \subseteq \mathbf{B} = C$. Hence, if the first call to $EQ(C)$ does not return *true*, it necessarily returns an example e such that $y(e) = 1$ and $\boldsymbol{x}(e) \in f^{-1}(1) \setminus sol(C)$. Constraints removed from C in Line 4 reject e and thus do not belong to C^f, so "$C^f \subseteq C$" remains true. The next example e' returned by $EQ(C)$ will again be such that $y(e') = 1$ and $\boldsymbol{x}(e') \in f^{-1}(1) \setminus sol(C)$, and "$C^f \subseteq C$" is an invariant of the loop. In addition, there is at least one constraint in C rejecting e in Line 4 otherwise $EQ(C)$ would have returned *true*. Therefore C strictly decreases in size each time we go through the loop and the algorithm terminates in $O(|\mathbf{B}|)$ number of equivalence queries. □

Theorem 4. *There exist biases* $\mathbf{B}$ *with fixed arity constraints, training sets* E*, and target concepts* f *representable by* $\mathbf{B}$ *that are not learnable by membership queries, even if* E *contains a positive example.*

Proof. Let Γ be the language containing only two binary constraints, given by $c_1 = \{(0,0),(1,1),(2,2)\}$ and $c_2 = \{(0,1),(1,0),(2,2)\}$. Let $\mathbf{B}$ be the bias with $2n$ variables, domain $\{0,1,2\}$, and language Γ, and let $\mathbf{C_B}$ be the class of concepts representable by this bias. Let G be the subset of $\mathbf{C_B}$ composed of all concepts representable by networks composed of n constraints from Γ, one constraint for each pair (x_i, x_{n+i}) of variables. Every network representing a concept in G can be associated with a vector t of length n constructed by setting $t[i]$ to 0 if $c_1(x_i, x_{n+i})$ and $t[i]$ to 1 if $c_2(x_i, x_{n+i})$. Thus there is a bijection between G and $\{0,1\}^n$ and G is isomorphic to $\{0,1\}^n$. As a result, the size of G is 2^n. Suppose that the positive example defined over the assignment $(2, 2, \cdots, 2)$ has already been given by the user. This does not eliminate any concept from G. Given any assignment $\boldsymbol{x} \in \{0,1,2\}^{2n}$, a membership query answered 0 will eliminate at most one concept from G (when $\boldsymbol{x} \in \{0,1\}^{2n}$). Therefore, whatever the strategy of generation of the queries, there exists a target concept f in G requiring at least $2^n - 1$ queries for identifying it. □

4 The Passive Conacq Algorithm

We are now in position to present the CONACQ architecture. In this section, we concentrate on *passive* constraint acquisition, where the learner is presented a pool of examples and must determine the state of its version space. We provide a compact representation of version spaces, next we analyse the complexity of some common operations on versions spaces in this representation, and we finally describe the CONACQ.1 algorithm.

4.1 Representing Version Spaces

The basic building block of our learning architecture is a clausal representation of version spaces for constraint biases. Given a prefixed vocabulary $\langle X, D\rangle$ and a bias $\mathbf{B}$, the version space $\mathbf{C_B}(E)$ of a training set E is encoded into a clausal theory T, where each model of T is a representation of some concept in $\mathbf{C_B}(E)$. The size of T is linear in the number of examples, and its structure allows us to check satisfiability in polynomial time.

Formally, any constraint $c \in \mathbf{B}$ is associated with a Boolean atom, denoted $a(c)$. As usual, a *positive literal* is an atom $a(c)$, and a *negative literal* is the negation $\neg a(c)$ of an atom. A *clause* α is a disjunction of literals; α is a *Horn clause* (resp. *dual Horn clause*) if it includes at most one positive (resp. negative) literal, and α is a *unit clause* if it includes exactly one (positive or negative) literal. A *clausal theory* T is a conjunction of clauses, and a *Horn theory* (resp. *dual Horn theory*) is a conjunction of Horn (resp. dual Horn) clauses. The *size* $|T|$ of a clausal theory T is given by the sum of the sizes of its clauses, where the size of a clause is the number of its literals. Given a clause α in T, *constraints*(α) is the set $\{c_i \mid a(c_i) \in \alpha\}$.

Any Boolean assignment I in $\{0,1\}^{|\mathbf{B}|}$ is called an *interpretation*, and we write $I[a(c)]$ to denote the (Boolean) value of the atom $a(c)$ under I. An interpretation I is a *model* of a clausal theory T if T is true in I according to the standard propositional semantics. The set of models of T is denoted *models*(T). A clausal theory T is *satisfiable* if *models*$(T) \neq \varnothing$, and *unsatisfiable* otherwise. Given two clausal theories T and T', we say that T *entails* T', and write $T \models T'$, if *models*$(T) \subseteq$ *models*(T').

The *transformation* φ assigns to each interpretation I over $\{0,1\}^{|\mathbf{B}|}$ a corresponding constraint network $\varphi(I)$ in $\mathbf{C_B}$ defined by the set of all constraints $c \in \mathbf{B}$ such that $I[a(c)] = 1$. Clearly, φ is *bijective*. In the following $\varphi^{-1}(C)$ denotes the *characteristic model* of C obtained by setting each atom $a(c)$ from $\mathbf{B}$ to 1 if $c \in C$ and to 0 otherwise. Given a set $\mathcal{C}$ of constraint networks, $\varphi^{-1}(\mathcal{C})$ denotes the set of characteristic models of the networks in $\mathcal{C}$.

It is important to keep in mind that the negative literal $\neg a(c)$ does *not* represent the complement $\overline{c}$ of the constraint c. Instead, $\neg a(c)$ denotes the *absence* of c in the network. Thus, $\neg a(c)$ captures a weaker form of negation than $\overline{c}$. For instance, given two variables x_i and x_j, the constraint $\overline{\leq_{ij}}$ is $>_{12}$, but the literal $\neg a(\leq_{12})$ simply specifies the absence of $\leq_{12}$ in the network.

Definition 11 (Clausal Representation)**.** *Given a bias* $\mathbf{B}$ *and a training set* E*, the* clausal representation *of* $\mathbf{C_B}(E)$ *is the dual Horn formula defined by:*

$$T = \bigwedge_{e \in E, y(e)=1} \left(\bigwedge_{c \in \kappa(\boldsymbol{x}(e))} \neg a(c) \right) \wedge \bigwedge_{e \in E, y(e)=0} \left(\bigvee_{c \in \kappa(\boldsymbol{x}(e))} a(c) \right)$$

where $\kappa(\boldsymbol{x})$ *is the set of constraints* c *in* $\mathbf{B}$ *such that* $\boldsymbol{x}$ *violates* c*.*

Clearly, the clausal encoding of a version space can be performed incrementally: on each incoming example e, encode e as a set of clauses using $\kappa(\boldsymbol{x}(e))$. If e is positive, we must discard from the version space all concepts that reject $\boldsymbol{x}(e)$. This is done by expanding the theory with a unit clause $\neg a(c)$ for each constraint c in $\kappa(\boldsymbol{x}(e))$. Dually, if e is negative, we must discard from the version space all concepts that accept $\boldsymbol{x}(e)$. This is done by expanding the theory with the clause consisting of all literals $a(c)$ in $\kappa(\boldsymbol{x}(e))$. The resulting theory T is indeed a dual Horn formula because each clause contains at most one negative literal.

Example 5. We wish to acquire a constraint network specified over the variables $\{x_1, x_2, x_3, x_4\}$, the domain $\{1, 2, 3, 4\}$, and the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$. Suppose that the target network contains only one constraint, namely $x_1 \neq x_4$. The following table illustrates how the clausal theory T is expanded after processing each example of some training set E.

$\boldsymbol{x}(e)$	$y(e)$	Clauses added to T
$\langle 1,2,3,4 \rangle$	1	$\neg a(\geq_{12}) \wedge \neg a(\geq_{13}) \wedge \neg a(\geq_{14}) \wedge \neg a(\geq_{23}) \wedge \neg a(\geq_{24}) \wedge \neg a(\geq_{34})$
$\langle 4,3,2,1 \rangle$	1	$\neg a(\leq_{12}) \wedge \neg a(\leq_{13}) \wedge \neg a(\leq_{14}) \wedge \neg a(\leq_{23}) \wedge \neg a(\leq_{24}) \wedge \neg a(\leq_{34})$
$\langle 1,1,1,1 \rangle$	0	$a(\neq_{12}) \vee a(\neq_{13}) \vee a(\neq_{14}) \vee a(\neq_{23}) \vee a(\neq_{24}) \vee a(\neq_{34})$
$\langle 1,2,2,1 \rangle$	0	$a(\geq_{12}) \vee a(\geq_{13}) \vee a(\neq_{14}) \vee a(\neq_{23}) \vee a(\leq_{24}) \vee a(\leq_{34})$

The fourth clause can be reduced to $a(\neq_{14}) \vee a(\neq_{23})$ using unit propagation over T, and hence the third clause can be removed from T because it is subsumed by the reduced fourth clause. Thus, models of T are all interpretations that set $a(\neq_{14})$ or $a(\neq_{23})$ to true and that falsify all $a(\leq_{ij})$ and all $a(\geq_{ij})$. ♢

The next theorem establishes a one-to-one correspondence between the representations of concepts in the version space and the models of the clausal theory.

Theorem 5. *Let* $\langle X, D \rangle$ *be a prefixed vocabulary,* $\mathbf{B}$ *a bias, and* E *a training set. Let* T *be the clausal representation of* $\mathbf{C_B}(E)$*. Then,*

$$I \in \mathit{models}(T) \text{ if and only if } f_{\varphi(I)} \in \mathbf{C_B}(E)$$

Proof. Consider a representation $C = \varphi(I)$ such that $I \in \mathit{models}(T)$ but $f_C \notin \mathbf{C_B}(E)$. We show that this leads to a contradiction. Obviously, f_C must be inconsistent with at least one example e in the training set. If e is positive then $f_C(\boldsymbol{x}(e)) = 0$, which implies that $\boldsymbol{x}(e) \notin \mathit{sol}(C)$. So, there is at least

one constraint in C that is included in $\kappa(\boldsymbol{x}(e))$. It follows that I violates the clause set $\bigwedge_{c\in\kappa(\boldsymbol{x}(e))}\neg a(c)$, and hence, I cannot be a model of T. Dually, if e is negative then $f_C(\boldsymbol{x}(e)) = 1$, which implies that $\boldsymbol{x}(e) \in sol(C)$. So, there is no constraint in C that belongs to $\kappa(\boldsymbol{x}(e))$. It follows that I violates the clause $\bigvee_{c\in\kappa(\boldsymbol{x}(e))} a(c)$, and hence, I cannot be a model of T.

Consider now a representation $C = \varphi(I)$ such that $f_C \in \mathbf{C_B}(E)$ but $I \notin models(T)$. Again, we show that this leads to a contradiction. If $I \notin models(T)$, then there is at least one example e in E such that I falsifies the set of clauses generated from e. If e is positive then I must violate the conjunction $\bigwedge_{c\in\kappa(\boldsymbol{x}(e))}\neg a(c)$. So C must include at least one member of $\kappa(\boldsymbol{x}(e))$, which implies that at least one constraint in C is violated by $\boldsymbol{x}(e)$. Therefore, $\boldsymbol{x}(e) \notin sol(C)$, and hence, f_C cannot be consistent with e. Dually, if e is negative then I must violate the disjunction $\bigvee_{c\in\kappa(\boldsymbol{x}(e))} a(c)$. So C must exclude all constraints c in $\kappa(\boldsymbol{x}(e))$, which implies that no constraint in C is violated by $\boldsymbol{x}(e)$. Therefore, $\boldsymbol{x}(e) \in sol(C)$, and hence, f_C cannot be consistent with e. □

4.2 Operations on the Clausal Representation of Version Spaces

The key interest of the clausal encoding of version spaces is to exploit the algorithmic properties of dual Horn theories.

To this point, recall that *unit propagation* is the process of applying unit resolution (a resolution step in which at least one resolvant is a unit clause) on a clausal theory T until no further unit resolution steps can be carried out, or until the empty clause is obtained. If T is Horn or dual Horn, then unit propagation is enough for deciding the satisfiability of T in linear ($\mathcal{O}(|T|)$) time, by either returning a model I of T, or deriving the empty clause [17]. Interestingly, the "unit implicates" of Horn theories and dual Horn theories can also be computed efficiently. For a clausal formula T, let $unit^+(T)$ (resp. $unit^-(T)$) be the set of positive (resp. negative) literals ℓ such that $T \models \ell$, and let $unit(T) = unit^+(T) \cup unit^-(T)$. A clausal theory T is *reduced* (under unit propagation) if there is no clause in T that properly contains a unit clause in $unit(T)$, or that contains the complementary literal of a unit clause in $unit(T)$.

As shown in [13] (Chapter 5.3), if T is Horn or dual Horn, then it can be reduced in linear time, using unit propagation. Furthermore, the set $unit^+(T)$ can be computed in linear time if T is a Horn theory, and the set $unit^-(T)$ can be computed in linear time if T is a dual Horn theory.

With these notions in hand, we shall consider in the following a prefixed vocabulary $\langle X, D\rangle$, a bias $\mathbf{B}$, and a training set E. Based on Definition 11, the size of the dual Horn theory T encoding $\mathbf{C_B}(E)$ is given by $\sum_e |\kappa(\boldsymbol{x}(e))|$. Since $|\kappa(\boldsymbol{x}(e))|$ contains at most $|\mathbf{B}|$ constraints, it follows that $|T|$ is bounded by $|E| \cdot |\mathbf{B}|$.

We first show that the consistency problem is solvable in linear time.

Proposition 1. *The consistency problem can be solved in $\mathcal{O}(|E| \cdot |\mathbf{B}|)$ time.*

Proof. By Theorem 5, we know that if T includes at least one model I then $\mathbf{C_B}(E)$ covers the concept $f_{\varphi(I)}$. Conversely, if $\mathbf{C_B}(E)$ covers at least one concept f, then T includes all models I for which $f = sol(\varphi(I))$. It follows that T is satisfiable if and only if $\mathbf{C_B}(E)$ is not empty. Since the satisfiability of T can be decided in $\mathcal{O}(|T|)$ time, the result follows. □

In the *update* operation, we need to compute a new representation of the version space formed by the addition of a new example to the training set.

Proposition 2. *The* update *operation takes* $\mathcal{O}(|\mathbf{B}|)$ *time.*

Proof. Adding a new example e means expanding T with the encoding of e. The cost of this expansion is dominated by the construction of $\kappa(\boldsymbol{x}(e))$. Checking whether a constraint is satisfied or violated by an example e is in $\mathcal{O}(1)$. Thus, the number of such checks is bounded by $|\mathbf{B}|$, which concludes. □

Consider a pair of training sets E_1 and E_2. The *intersection* operation requires computing a representation of the version space $\mathbf{C_B}(E_1) \cap \mathbf{C_B}(E_2)$. This operation is interesting when we acquire a network from examples supplied by multiple users.

Proposition 3. *The* intersection *operation takes* $\mathcal{O}((|E_1| + |E_2|) \cdot |\mathbf{B}|)$ *time.*

Proof. Let T_1 and T_2 be the representations of the version spaces $\mathbf{C_B}(E_1)$ and $\mathbf{C_B}(E_2)$, respectively. The representation of the version space $\mathbf{C_B}(E_1) \cap \mathbf{C_B}(E_2)$ is simply obtained by $T_1 \wedge T_2$, which cost is in the sum of the sizes of the two theories, that is, $\mathcal{O}((|E_1| + |E_2|) \cdot |\mathbf{B}|)$. □

Given a pair of training sets E_1 and E_2, we may wish to determine whether $\mathbf{C_B}(E_1)$ is a *subset* of $\mathbf{C_B}(E_2)$, or whether $\mathbf{C_B}(E_1)$ is *equal* to $\mathbf{C_B}(E_2)$.

Proposition 4. *The* subset *and* equality *tests take* $\mathcal{O}(|E_1| \cdot |E_2| \cdot |\mathbf{B}|^2)$ *time.*

Proof. Let T_1 and T_2 be the clausal theories of $\mathbf{C_B}(E_1)$ and $\mathbf{C_B}(E_2)$, respectively. If $\mathbf{C_B}(E_1) \subseteq \mathbf{C_B}(E_2)$ then we must have $models(T_1) \subseteq models(T_2)$ because otherwise there would be a model I in $models(T_1) \setminus models(T_2)$, implying by Theorem 5 that $f_{\varphi(I)} \in \mathbf{C_B}(E_1)$ but $f_{\varphi(I)} \notin \mathbf{C_B}(E_2)$, a contradiction. Conversely, if $models(T_1) \subseteq models(T_2)$ then we must have $\mathbf{C_B}(E_1) \subseteq \mathbf{C_B}(E_2)$ because otherwise there would exist a concept f in $\mathbf{C_B}(E_1) \setminus \mathbf{C_B}(E_2)$ implying, again by Theorem 5, that for every representation C of f, $\varphi^{-1}(C)$ is a model of T_1, but not a model of T_2, a contradiction. Thus, deciding whether $\mathbf{C_B}(E_1) \subseteq \mathbf{C_B}(E_2)$ is equivalent to deciding whether $models(T_1) \subseteq models(T_2)$, which is equivalent to state whether T_1 entails T_2. By Lemma 5.6.1 from [13], the entailment problem of two dual Horn formulas T_1 and T_2 can be decided in $\mathcal{O}(|T_1| \cdot |T_2|)$ time. It follows that the subset operation takes $\mathcal{O}(|\mathbf{B}|^2 \cdot |E_1| \cdot |E_2|)$ time. For the equality operation, we simply need to check whether T_1 entails T_2 and T_2 entails T_1. □

The *membership* test is to determine whether a given constraint network is associated, or not, to a consistent concept in the version space.

Proposition 5. *The* membership *test takes* $\mathcal{O}(|E| \cdot |\mathbf{B}|)$ *time.*

Proof. Let $C \subseteq \mathbf{B}$ be a constraint network. By Theorem 5, determining whether $f_C \in \mathbf{C_B}(E)$ is equivalent to check whether $\varphi^{-1}(C) \in \mathit{models}(T)$, which can be done in $\mathcal{O}(|T|)$ time. □

Apart from the membership test, we might also be interested to check whether a given assignment is classified in the same way by all concepts in the version space. Specifically, an assignment $\boldsymbol{x}$ is predictable by $\mathbf{C_B}(E)$, if $\mathbf{C_B}(E)$ is not empty, and $f(\boldsymbol{x}) = 1$ for all $f \in \mathbf{C_B}(E)$, or $f(\boldsymbol{x}) = 0$ for all $f \in \mathbf{C_B}(E)$. The *prediction* test suggested in [24], is to determine whether an assignment is predictable, or not.

Proposition 6. *The* prediction *test takes* $\mathcal{O}(|E| \cdot |\mathbf{B}|)$ *time.*

Proof. Given an assignment $\boldsymbol{x}$, let T_0 (resp. T_1) be the clausal theory obtained from T by updating $\mathbf{C_B}(E)$ with the example $\langle \boldsymbol{x}, 0\rangle$ (resp. $\langle \boldsymbol{x}, 1\rangle$). T_0 is unsatisfiable if and only if $\mathbf{C_B}(E \cup \{\langle \boldsymbol{x}, 0\rangle\}) = \varnothing$. This condition holds if and only if $f(\boldsymbol{x}) = 1$ for all $f \in \mathbf{C_B}(E)$. Analogously, T_1 is unsatisfiable if and only if $f(\boldsymbol{x}) = 0$ for all $f \in \mathbf{C_B}(E)$. Note that $\mathbf{C_B}(E) = \varnothing$ if and only if both T_0 and T_1 are unsatisfiable. So, $\boldsymbol{x}$ is predictable if and only if exactly one of T_0 or T_1 is unsatisfiable. Since by Proposition 2 the update of $\mathbf{C_B}(E)$ with $\langle \boldsymbol{x}, 0\rangle$ (or $\langle \boldsymbol{x}, 1\rangle$) takes $\mathcal{O}(|\mathbf{B}|)$ time, the prediction test requires only two satisfiability tests over dual Horn formulas, which takes $\mathcal{O}(|E| \cdot |\mathbf{B}|)$ time. □

We close the list of operations be examining the boundary elements of the version space, that is, the maximally specific concepts and the maximally general ones.

Proposition 7. *If* $\mathbf{C_B}(E) \neq \varnothing$*, then the maximally specific concept is unique and a representation of it can be computed in* $\mathcal{O}(|E| \cdot |\mathbf{B}|)$ *time.*

Proof. Suppose that $\mathbf{C_B}(E) \neq \varnothing$, and consider the network $C = \{c \in \mathbf{B} \mid \neg a(c) \notin \mathit{unit}^-(T)\}$. Note that C can be constructed in $\mathcal{O}(|E| \cdot |\mathbf{B}|)$ time by computing $\mathit{unit}^-(T)$ using unit propagation, and taking the atoms in $\mathbf{B}$ which do not occur in $\mathit{unit}^-(T)$. Now, consider any concept $f' \in \mathbf{C_B}$ such that $f' \subset f_C$. Any representation C' of f' must include at least one constraint c in $\mathbf{B} \setminus C$. This implies that $\neg a(c) \in \mathit{unit}^-(T)$, which in turn implies that $\phi^{-1}(C')$ does not satisfy T. Therefore, $f' \notin \mathbf{C_B}(E)$, and hence, f_C is the unique maximally specific concept in $\mathbf{C_B}(E)$. □

Unfortunately, a dual property for maximally general concepts cannot be derived, due to the fact that their number can grow exponentially with the size of $\mathbf{B}$. Indeed, as observed in [23], even in the very restricted case where the domain is $\{0, 1\}$ and $\mathbf{B}$ is a set of n unary constraints c with $\mathit{rel}(c) = \{1\}$, there are training sets E such that the number of maximally general concepts in $\mathbf{C_B}(E)$ is exponential in n. A tractable result can yet be derived when the clausal representation T is reduced to a monomial (i.e., conjunction of unit clauses).

Proposition 8. *If* $\mathbf{C_B}(E) \neq \varnothing$ *and* T *is a monomial, then the maximally general concept is unique and a representation of it can be computed in* $\mathcal{O}(|E| \cdot |\mathbf{B}|)$ *time.*

Proof. Suppose again that $\mathbf{C_B}(E) \neq \varnothing$, and consider the network $C = \{c \in \mathbf{B} \mid a(c) \in \mathit{unit}^+(T)\}$. Because T is a monomial, it is both Horn and dual Horn, and hence, C can be constructed in $\mathcal{O}(|T|)$ time by deriving $\mathit{unit}^+(T)$ via unit propagation. Now, consider any concept $f' \in \mathbf{C_B}$ such that $f_C \subset f'$. Here, any representation C' of f' must exclude at least one constraint $c \in C$. This implies that $a(c) \notin \mathit{unit}^+(T)$, which in turn implies that $\phi^{-1}(C')$ violates T. Therefore, $f' \notin \mathbf{C_B}(E)$, and hence, f_C is the unique maximally general concept. □

4.3 The Algorithm

The Conacq.1 algorithm is presented in Algorithm 1. It takes as input a bias $\mathbf{B}$ and a training set E, and returns as output a clausal theory T that encodes the version space $\mathbf{C_B}(E)$. The algorithm starts from the empty theory (Line 1) and iteratively expands it by encoding each example in the training set (Line 2). If e is negative, we must discard from the version space all concepts that accept $\boldsymbol{x}(e)$. This is done by expanding the theory with the clause consisting of all literals $a(c)$ in $\kappa(\boldsymbol{x}(e))$ (Line 4). Dually, if e is positive, we must discard from the version space all concepts that reject $\boldsymbol{x}(e)$. This is done by expanding the theory with a unit clause $\neg a(c)$ for each constraint c in $\kappa(\boldsymbol{x}(e))$ (Line 5). After encoding the example, a "collapse" message is returned if the theory is no longer satisfiable (Line 6). When all examples in the training set are processed, Conacq.1 calls a convergence procedure to determine whether $\mathbf{C_B}(E)$ is reduced to a singleton set, or not (Line 7). Finally Conacq.1 returns the resulting theory encoding $\mathbf{C_B}(E)$ together with the flag for convergence (Line 8).

As stated by Theorem 2, the convergence problem is coNP-complete. A naive strategy for implementing the Convergence procedure is to start from the interpretation I encoding the maximally specific network (as detailed in Proposition 7), and to explore the other models of T in order to find an interpretation I' for which the constraint networks $\phi^{-1}(I)$ and $\phi^{-1}(I')$ are not equivalent. Yet, due to the exponential number of models of dual Horn theories, and the complexity of constraint network equivalence, such an implementation will be too expensive in most cases. The next section proposes two ways to improve the way convergence is handled in Conacq.1.

5 Improvements for Convergence Testing

A natural way to alleviate the computational barrier related to the convergence test is to use the notion of *local consistency*, which is ubiquitous in Constraint Programming. The idea of local consistency can be summarized as an explicitation of inconsistent combinations of values or combinations of constraints among subsets of variables. Such information can be exploited as a *background*

Algorithm 1: The CONACQ.1 Algorithm

Input: a bias $\mathbf{B}$ and a training set E
Output: a clausal theory T encoding $\mathbf{C_B}(E)$, a Boolean value v saying if convergence is reached

$T \leftarrow \varnothing$
foreach *example* $e \in E$ **do**
  $\kappa(\boldsymbol{x}(e)) \leftarrow \{c \in \mathbf{B} \mid \boldsymbol{x}(e) \text{ violates } c\}$
  if $y(e) = 0$ **then** $T \leftarrow T \wedge \left(\bigvee_{c \in \kappa(\boldsymbol{x}(e))} a(c)\right)$
  if $y(e) = 1$ **then** $T \leftarrow T \wedge \bigwedge_{c \in \kappa(\boldsymbol{x}(e))} \neg a(c)$
  if T *is unsatisfiable* **then return** *"collapse"*
$v \leftarrow$ CONVERGENCE(T)
return (T, v)

knowledge for improving the learning process and speeding up the convergence test. Taking a central part in Inductive Logic Programming [30], the background knowledge is a set of clauses that impose restrictions on the possible representations of learned concepts. In our constraint acquisition setting, the background knowledge is a set of rules, each encoding in a declarative way a form of local consistency. By combining the clausal representation of a version space with some background knowledge, the convergence problem can be solved using *backbone tests*, a powerful SAT technique.

In this section, we first examine the concept of background knowledge adapted to constraint acquisition, and then we turn to the technique of backbone tests.

5.1 Background Knowledge

Recall that a Horn clause is *definite* if it contains exactly one positive literal. Intuitively, a rule is a definite Horn clause that captures, in form of implication, a local consistency property between some constraints defined over the bias.

Definition 12 (Rule)**.** *Given a vocabulary* $\langle X, D\rangle$*, and a bias* $\mathbf{B}$*, a* rule *is a definite Horn clause* $\bigwedge_{i=1}^{l} a(c_i) \rightarrow a(c)$ *such that* $\{c_1, \cdots, c_l, c\} \subseteq \mathbf{B}$*. The rule is* correct *for* $\mathbf{B}$ *if* $\{c_1, \cdots, c_l, \overline{c}\}$ *is unsatisfiable.*

Definition 13 (Background Knowledge)**.** *Given a bias* $\mathbf{B}$*, a* background knowledge K *for* $\mathbf{B}$ *is a set of correct rules for* $\mathbf{B}$*.*

Based on some background knowledge K, any candidate constraint network C can be "saturated" by K, in order to yield an equivalent network C' including all additional constraints forced by rules whose body is satisfied by C.

Definition 14 (Saturation)**.** *Given a bias* $\mathbf{B}$*, a background knowledge* K *for* $\mathbf{B}$*, and a constraint network* $C \subseteq \mathbf{B}$*, the* saturation *of* C *with* K*, denoted* $sat_K(C)$*,*

is the network $C \cup \{c \mid a(c) \in unit^+(K \cup C)\}$. *A network* $C \subseteq \mathbf{B}$ *is* saturated *by* K *if* $C = sat_K(C)$.

Example 6. Consider the vocabulary composed of variables $\{x_1, x_2, x_3\}$ and domain $\{1, 2, 3, 4, 5\}$, and the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$. The following set of rules is background knowledge for $\mathbf{B}$.

$$K = \begin{cases} a(\leq_{ij}) \leftrightarrow a(\geq_{ji}), \forall i, j \\ a(\neq_{ij}) \leftrightarrow a(\neq_{ji}), \forall i, j \\ a(\leq_{ij}) \wedge a(\leq_{jk}) \rightarrow a(\leq_{ik}), \forall i, j, k \\ a(\geq_{ij}) \wedge a(\geq_{jk}) \rightarrow a(\geq_{ik}), \forall i, j, k \\ a(\leq_{ij}) \wedge a(\geq_{ij}) \wedge a(\circ_{jk}) \rightarrow a(\circ_{ik}) \text{ with } \circ \in \{\leq, \neq, \geq\}, \forall i, j, k \end{cases}$$

Consider the constraint network $C = \{\leq_{12}, \leq_{23}\}$. The saturation of C with K is $C^* = \{\leq_{12}, \leq_{23}, \leq_{13}\}$. ♢

Informally, the background knowledge K is "complete" if all local consistencies between constraints in the bias can be derived by K.

Definition 15 (Subsumed Rule)**.** *Given a bias* $\mathbf{B}$*, background knowledge* K *for* $\mathbf{B}$*, and a correct rule* R *for* $\mathbf{B}$ *such that* $R \notin K$*,* R *is* subsumed *by* K *if and only if* $sat_{K \cup \{R\}}(C) = sat_K(C)$ *for every constraint network* $C \subseteq \mathbf{B}$.

Definition 16 (Complete Background Knowledge)**.** *Given a bias* $\mathbf{B}$*,* background knowledge K *for* $\mathbf{B}$ *is* complete *if any correct rule for* $\mathbf{B}$ *is either in* K *or subsumed by* K.

In presence of complete background knowledge, the equivalence between constraint networks can be identified by saturation.

Lemma 1. *Let* $\mathbf{B}$ *be a bias, and* K *be complete background knowledge for* K*. Then, for any* $C, C' \subseteq \mathbf{B}$*,* C *is equivalent to* C' *if and only if* $sat_K(C) = sat_K(C')$.

Proof. If $sat_K(C) = sat_K(C')$ then, because K is a set of correct rules for $\mathbf{B}$, it follows that C and C' are equivalent. Conversely, suppose that $C = \{c_1, \cdots, c_l\}$ is equivalent to C'. For any constraint $c' \in C'$, because $C \cup \{c'\}$ is entailed by C, the rule R given by $a(c_1) \wedge \cdots \wedge a(c_l) \rightarrow a(c')$ is correct. And, since K is complete, R is either in K or subsumed by K, which implies that $c' \in sat_K(C)$. By applying the same strategy to all constraints in the symmetric difference $(C' \setminus C) \cup (C \setminus C')$, we get that $C' \setminus C \subseteq sat_K(C)$ and $C \setminus C' \subseteq sat_K(C')$. This, together with the fact that $C \subseteq sat_K(C)$ and $C' \subseteq sat_K(C')$ implies that $sat_K(C) = sat_K(C \cup C') = sat_K(C')$, as desired. □

Recall that a Boolean formula is uniquely satisfiable if it has a single model. Based on this notion and the above result, a useful property can be derived from complete forms of background knowledge.

Proposition 9. *Let* K *be complete background knowledge for some bias* $\mathbf{B}$*. Then, for any training set* E *and its associated clausal representation* T*,* $\mathbf{C_B}(E)$ *has converged if and only if* $K \wedge T$ *is uniquely satisfiable.*

Proof. Suppose that $\mathbf{C_B}(E)$ has converged to the unique concept f^*, and let $\mathcal{C}_{f^*}$ be the equivalence class of all representations of f^* in $\mathbf{B}$. By Theorem 1, we know that $\phi^{-1}(C) \models T$ for every $C \in \mathcal{C}_{f^*}$. By Lemma 1, we also know that there exists exactly one representation $C^* \in \mathcal{C}_{f^*}$ such that $C^* = sat_K(C)$ for all $C \in \mathcal{C}_{f^*}$. Since C^* is saturated, it follows that $\phi^{-1}(C^*)$ satisfies every rule of K, and hence, $\phi^{-1}(C^*) \models T \wedge K$. Now, consider any network $C \in \mathcal{C}_{f^*}$ such that $C \neq C^*$. Since C is a proper subset of $sat_K(C)$, $\phi^{-1}(C)$ violates every rule R with body $\{a(c) \mid c \in C\}$ and head in $\{a(c') \mid c' \in sat_K(C) \setminus C\}$. However, because K is complete, R is subsumed by K, and by contraposition, $\phi^{-1}(C)$ is not a model of K. Consequently, $\phi^{-1}(C^*)$ is the unique model of $T \wedge K$.

Conversely, suppose that $K \wedge T$ is uniquely satisfiable, and let I^* be the unique model of $K \wedge T$, with associated network $C^* = \phi(I^*)$. Since $I^* \models T$, we know that $f^* = f_{C^*}$ is a member of $\mathbf{C_B}(E)$. Now consider any consistent concept $f \in \mathbf{C_B}(E)$. For any representation C of f, the interpretation $I = \phi^{-1}(C)$ must be a model of T. If I is distinct from I^*, it cannot be a model of $T \wedge K$, and hence, C must be a proper subset of $C' = sat_K(C)$. For this saturated network C', we know that $I' = \phi^{-1}(C')$ is a model of K. Since, in addition, K is a set of correct rules for $\mathbf{B}$, C' is a representation of f, which implies that I' is also a model of T. Consequently, $I' \models T \wedge K$, implying that $I' = I^*$ by unique satisfiability of $T \wedge K$, which in turn implies that $f = f^*$. □

Building complete background knowledge is often too expensive, both in time and space as it requires generating a set of rules potentially exponential in space (all combinations of constraints that imply another one). This is not surprising as it is closely related to the concept of relational consistency in constraint networks [16]. However, by analogy with "levels of consistency" used in Constraint Programming, it is possible to compute approximations by bounding the number of constraints in the body of a rule. For instance, in all the experiments we have performed with Conacq, we only generate the rules that contain two constraints in the body because we found that in practice many rules have a small length. Take for instance the rule $x \leq y \wedge y \leq z \rightarrow x \leq z$. It can be detected by brute force generation of the $|D|^3$ assignments on (x, y, z) that satisfy the body of the rule, and testing that the head is always satisfied. Once detected, such a rule must be put in K in the form of a clause $a(\leq_{ij}) \wedge a(\leq_{jk}) \rightarrow a(\leq_{ik})$ for all triplets of such constraints in $\mathbf{B}$. In our example, there are $|n^3|$ such rules. In general, given a rule of length l associated with a language Γ containing t relations of maximum arity k, and a vocabulary involving n variables, we can generate up to n^{kl} rules to be put in K.

With these notions in hand, we can identify a case where checking convergence in Conacq.1 is polynomial. The method described in Algorithm 2 provides and implementation of the Convergence procedure in situations where the theory T is reduced to a monomial after unit propagation. Recall that in such situations, the maximally specific and the maximally general concepts are both unique (Propositions 7 and 8). We first take the maximally specific constraint network S formed by all constraints whose associated atom is not negated in T (Line 1). Next, we take the "saturated" maximally general net-

Algorithm 2: Procedure `Restricted Convergence`

Input: a satisfiable monomial theory T and a background knowledge K
Output: a Boolean reporting convergence or not

1 $S \leftarrow \{c \in \mathbf{B} \mid \neg a(c) \notin \mathit{unit}^-(T)\}$
2 $G \leftarrow \{c \in \mathbf{B} \mid a(c) \in \mathit{unit}^+(T \wedge K)\}$
3 **return** $S = G$

work G formed by all constraints whose associated atom occurs positively in $T \wedge K$ (Line 2). The convergence is established by simply testing whether S and G are equal, or not (Line 3).

Theorem 6 (Restricted Convergence)**.** *Given a bias* $\mathbf{B}$*, background knowledge* K *for* $\mathbf{B}$*, and a training set* E*, if* K *is complete for* $\mathbf{B}$ *and the clausal representation* T *of* $\mathbf{C_B}(E)$ *is reduced to a monomial by unit propagation, then the convergence problem can be solved in* $\mathcal{O}(|\mathbf{B}| + |K|)$ *time.*

Proof. First, we examine the correctness of the restricted convergence procedure. Since T is a monomial, we know that the maximally specific concept f_S and the maximally general concept f_G of $\mathbf{C_B}(E)$ are unique (Propositions 7 and 8). So, $\mathbf{C_B}(E)$ has converged if and only if $f_S = f_G$. Let S (resp. G) be a representation of f_S (resp. f_G) over $\mathbf{B}$. By Lemma 1, a sufficient condition to establish the equality $f_S = f_G$ is to show that $\mathit{sat}_K(S) = \mathit{sat}_K(G)$. From this perspective, consider the networks S and G constructed by Algorithm 2. By Proposition 7, S is a representation of f_S. Clearly $S = \mathit{sat}_K(S)$ because otherwise, there would be a constraint $c \in \mathit{sat}_K(S) \setminus S$ such that $\neg a(c) \in \mathit{unit}^-(T)$, which in turn would imply that $f_S \neq f_{\mathit{sat}_K(S)}$, contradicting the fact that K is a set of correct rules. By Proposition 8, the network $G' = \{c \mid a(c) \in \mathit{unit}^+(T)\}$ is a representation of f_G, and by construction, $G = \mathit{sat}_K(G')$. Thus, $\mathbf{C_B}(E)$ has converged if and only if $S = G$, which is precisely what Algorithm 2 returns.

Now, let us turn to the complexity of the procedure. Since T is a monomial, $\mathit{unit}^-(T)$ can be computed in $\mathcal{O}(|T|)$ time, and since $T \wedge K$ is a Horn formula, $\mathit{unit}^+(T \wedge K)$ can be computed in $\mathcal{O}(|T + K|)$ time. The result follows using the fact that $|T|$ is bounded by $|\mathbf{B}|$. □

We note in passing that the above result can be derived from Proposition 9. Indeed, using the fact that $T \wedge K$ is a Horn theory when T is a monomial, the unique satisfiability test can be evaluated in $\mathcal{O}(|T| + |K|)$ time, using a directed hypergraph representation of the Horn formula [13]. However, in our setting, the restricted convergence procedure is much simpler to implement, requiring only unit propagation for computing the sets S and G.

Example 7. Consider the theory T generated in Example 4. After unit propagation, T contains two positive literals $a(\leq_{12})$ and $a(\leq_{23})$. The maximally specific network S is precisely $\{\leq_{12}, \leq_{23}, \leq_{13}\}$ and the maximally general network G is $\{\leq_{12}, \leq_{23}\}$. Using the background knowledge K presented in Example 6,

we derive that S is equal to the saturation of G with K. Hence, we infer that the version space has converged. ♢

5.2 Backbone Detection

In general, the theory T returned by CONACQ.1 cannot be reduced to a simple monomial by unit propagation. Even if the theory includes conjunctions of disjunctions, the version space may have converged because each maximally general concept is equivalent to the maximally specific concept. This general case is illustrated in the following example.

Example 8. Consider the variables $\{x_1, x_2, x_3\}$, the domain $\{1, 2, 3, 4, 5\}$, and the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$. Suppose that the target network is $C = \{=_{12}, =_{13}, =_{23}\}$. To acquire this concept, the learner is given the set E of 7 examples illustrated in the following table.

	$\langle x_1, x_2, x_3 \rangle$	$y(e)$	unit propagated clauses in T
e_1	$\langle 1, 1, 1 \rangle$	1	$\neg a(\neq_{12}) \wedge \neg a(\neq_{13}) \wedge \neg a(\neq_{23})$
e_2	$\langle 2, 2, 3 \rangle$	0	$a(\geq_{13}) \vee a(\geq_{23})$
e_3	$\langle 4, 4, 1 \rangle$	0	$a(\leq_{13}) \vee a(\leq_{23})$
e_4	$\langle 1, 2, 2 \rangle$	0	$a(\geq_{12}) \vee a(\geq_{13})$
e_5	$\langle 3, 1, 1 \rangle$	0	$a(\leq_{12}) \vee a(\leq_{13})$
e_6	$\langle 2, 4, 2 \rangle$	0	$a(\geq_{12}) \vee a(\leq_{23})$
e_7	$\langle 2, 1, 2 \rangle$	0	$a(\leq_{12}) \vee a(\geq_{23})$

This training set is sufficient to infer that the version space has converged. However, all positive clauses in T contain two positive literals. It follows that, even when using the complete background knowledge K given in Example 6, unit propagation on $T \wedge K$ is not sufficient to detect convergence. ♢

In the above example, unit propagation on $T \wedge K$ is not sufficient to infer that any concept in the version space is equivalent to the target constraint network $\{=_{12}, =_{13}, =_{23}\}$. The powerful notion of *backbone* of a propositional formula can be used here. A literal belongs to the backbone of a formula if it belongs to all models of the formula [28]. In the setting of our framework, we say that an atom $a(c)$ is in the backbone of a theory T with respect to background knowledge K if $a(c)$ is entailed by $T \wedge K$, or equivalently, if $T \wedge K \wedge \neg a(c)$ is unsatisfiable. Once the literals in the backbone are detected, they can be exploited to prove convergence.

The general version of convergence testing is described in Algorithm 3. As in Algorithm 2, we start from the maximally specific concept S of the version space (Line 1). But this time we cannot generate a unique maximally general concept. So, we use the theory T and the background knowledge K and we determine whether each constraint in S lies in the backbone of $T \wedge K$ (Lines 2-3). If this is indeed the case, we have converged (Line 4).

Algorithm 3: Procedure `General Convergence`

Input: a satisfiable theory T and background knowledge K
Output: a Boolean reporting convergence or not

```
1 S ← {c ∈ B | ¬a(c) ∉ unit⁻(T)}
2 foreach c ∈ S do
3     if T ∧ K ∧ ¬a(c) is satisfiable then return no
4 return yes
```

Theorem 7 (General Convergence)**.** *Given a bias* $\mathbf{B}$*, background knowledge* K *for* $\mathbf{B}$*, and a training set* E*, if* K *is complete for* $\mathbf{B}$*, then the convergence problem can be solved using* $\mathcal{O}(|\mathbf{B}|)$ *backbone tests.*

Proof. Suppose that K is complete, and let T be the clausal theory encoding $\mathbf{C_B}(E)$. By Proposition 9, we know that $\mathbf{C_B}(E)$ has converged if and only if $T \wedge K$ is uniquely satisfiable. If T is satisfiable, then testing whether $T \wedge K$ is uniquely satisfiable can be done by checking whether for each constraint $c \in \mathbf{B}$, the positive literal $a(c)$ or negative literal $\neg a(c)$ over c is entailed by $T \wedge K$. However, since we already know that all negative literals in $unit^-(T)$ are entailed by $T \wedge K$, we only need to check the constraints $c \in \mathbf{B}$ for which $\neg a(c) \notin unit^-(T)$. Furthermore, we also know that for the constraint network $S = \{c \in \mathbf{B} \mid \neg a(c) \notin unit^-(T)\}$, the corresponding interpretation $\phi^{-1}(S)$ is a model of $T \wedge K$, because S is an encoding of the maximally specific concept $f_S \in \mathbf{C_B}(E)$ (as stated in Proposition 7), and because S is saturated (as already shown in the proof of Theorem 6). Therefore, for each $c \in \mathbf{B}$ such that $a(c) \notin unit^-(T)$, we know that $T \wedge K \wedge a(c)$ is satisfiable. So, for each of these constraints c, we only need to check the satisfiability of $T \wedge K \wedge \neg a(c)$, which is precisely what the general convergence procedure performs, using at most $|\mathbf{B}| - |unit^-(T)|$ backbone tests. □

From a computational point of view, we must keep in mind that the theory T, coupled with the background knowledge K is neither a Horn (T is dual Horn) nor a dual Horn (K is Horn) formula. So, testing whether a literal is in the backbone of T with respect to K is coNP-complete. However, as shown in our experiments, backbone detection can be very fast under this representation scheme because both theories T and K efficiently propagate unit clauses.

Example 9. Consider again the scenario of Example 8. After processing the examples e_1, e_2 and e_4, we remark that $a(\geq_{13})$ is in the backbone of $T \wedge K$ because K includes the rule $a(\geq_{12}) \wedge a(\geq_{23}) \rightarrow a(\geq_{13})$. Analogously, after processing the examples e_3 and e_5, $a(\leq_{13})$ is in the backbone of T, because K includes the rule $a(\leq_{12}) \wedge a(\leq_{23}) \rightarrow a(\leq_{13})$. It follows that $=_{13}$ is in the target concept. After processing the example e_6, $a(\leq_{23})$ is in the backbone of $T \wedge K$ because K includes the rules $a(\geq_{12}) \wedge a(=_{13}) \rightarrow a(\geq_{32})$ and $a(\geq_{32}) \rightarrow a(\leq_{23})$. We similarly deduce that $a(\geq_{12})$ is in the backbone. Dually, after processing

the example e_7, $a(\leq_{12})$ and $a(\geq_{23})$ are in the backbone of $T \wedge K$. We have converged on the concept $\{=_{12}, =_{13}, =_{23}\}$. $\diamondsuit$

6 The Active Conacq.2 Algorithm

As specified in Section 3, constraint acquisition is the problem of identifying a representation of some target concept using a constraint language and user-supplied information taking the form of examples. In the *passive* acquisition setting examined in Section 4, the information given by the user is a training set E over which the learner has no control. By contrast, in the *active* acquisition setting, the learner is allowed to ask *membership queries*, that is, to select an assignment $\boldsymbol{x}$ and to ask the user what is the label of $\boldsymbol{x}$. The user answers *yes* if $\boldsymbol{x}$ is a solution of the target problem, and *no* otherwise. For many concept classes, the use of membership queries in conjunction with equivalence queries is known to dramatically accelerate the learning process [11, 12]. Though equivalence queries are not considered in constraint acquisition, and membership queries alone are not powerful enough to guarantee convergence in a polynomial number of queries (Theorem 4), the use of membership queries in conjunction with a given training set can substantially improve the acquisition process.

In this section, we will assume that the constraint acquisition problem is *realizable*, and membership queries are *answered correctly*. Namely, the target concept f^* is representable by a satisfiable constraint network over $\mathbf{B}$, and the answer y to any query $q = \boldsymbol{x}$ is consistent with f^*, i.e. $y = f^*(\boldsymbol{x})$. The more general, yet challenging, "agnostic" constraint acquisition setting, with possible omissions and errors in answers to membership queries, is deferred to future research.

Intuitively, a membership query $q = \boldsymbol{x}$ is "informative" or "irredundant" if, whatever being the user's answer y, the resulting example $\langle \boldsymbol{x}, y \rangle$ added to the current training set E will ensure to reduce the learner's version space. The task of finding such queries is, however, far from easy. In this section, we begin by a formal characterization of the notion of informative membership query, and next, we show that the problem of finding such a query is NP-hard. We then present the Conacq.2 algorithm, an active version of Conacq, which relies on different strategies for efficiently generating informative queries.

6.1 Informative Queries

Let $\mathbf{B}$ be a constraint bias, and E a set of examples. Formally, a membership query $q = \boldsymbol{x}$ is *informative* (or *irredundant*) with respect to $\mathbf{C_B}(E)$ if and only if $\boldsymbol{x}$ is *not* predictable by $\mathbf{C_B}(E)$. In other words, q is informative if and only if $\boldsymbol{x}$ is not classified in the same way by all concepts in the version space.

Example 10. [Redundant Query] Consider the vocabulary defined over the variables $\{x_1, x_2, x_3\}$, the domain $\{1, 2, 3, 4\}$, and the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$. Given the positive example $e^+ = \langle 1, 2, 3 \rangle$, the associated clausal theory T will contain $\neg a(\geq_{12})$, $\neg a(\geq_{13})$, and $\neg a(\geq_{23})$. Asking the user to classify $\boldsymbol{x} = \langle 1, 2, 4 \rangle$ is

redundant because all constraints rejecting it are already forbidden by T. In other words, any concept in the version space classifies $\boldsymbol{x}$ as positive. $\diamondsuit$

The next property is a variant of Proposition 6 which will be useful in active constraint acquisition. Given an assignment $\boldsymbol{x}$ over $D^{|X|}$, we denote by $\kappa_{[T]}(\boldsymbol{x})$ the subset obtained by removing from $\kappa(\boldsymbol{x})$ all constraints that appear as negated literals in T, that is, $\kappa_{[T]}(\boldsymbol{x}) = \kappa(\boldsymbol{x}) \setminus \{c_i \mid \neg a(c_i) \in \mathit{unit}^-(T)\}$. With a slight abuse of terminology, we say that $\kappa_{[T]}(\boldsymbol{x})$ is *subsumed* by a clause $\alpha \in T$ if $\mathit{constraints}(\alpha) \subseteq \kappa_{[T]}(\boldsymbol{x})$.

Proposition 10. *Given a constraint bias* $\mathbf{B}$*, and a training set* E*, any membership query* $q = \boldsymbol{x}$ *is informative if and only if* $\kappa_{[T]}(\boldsymbol{x})$ *is neither empty nor subsumed by any clause in* T*, where* T *is the reduced clausal theory of* $\mathbf{C_B}(E)$*.*

Proof. By Proposition 6, $\boldsymbol{x}$ is not predictable by $\mathbf{C_B}(E)$ if and only if both T_0 and T_1 are satisfiable, where T_0 and T_1 are the clausal encodings of $\mathbf{C_B}(E \cup \{\langle \boldsymbol{x}, 0\rangle\})$ and $\mathbf{C_B}(E \cup \{\langle \boldsymbol{x}, 1\rangle\})$, respectively. Let α be the positive clause $\bigvee\{a(c) \mid c \in \kappa(\boldsymbol{x})\}$. Then, T_0 is satisfiable if and only if $T \wedge \alpha$ is satisfiable, which is equivalent to state that $\kappa_{[T]}(\boldsymbol{x})$ is nonempty. Moreover, T_1 is satisfiable if and only if T does not entail α, which is equivalent to state that $\kappa_{[T]}(\boldsymbol{x})$ does not cover any clause in the reduced theory T. $\square$

Example 11. [Irredundant Query] Consider again Example 10 in which example e^+ has been processed, yielding $T = \{\neg a(\geq_{12}), \neg a(\geq_{13}), \neg a(\geq_{23})\}$. The query $q = \langle 1, 2, 2\rangle$ is irredundant because $\kappa_{[T]}(q)$ is neither empty nor a superset of any clause in T, and $\kappa_{[T]}(q) = \kappa(q) \setminus \{\geq_{12}, \geq_{13}, \geq_{23}\} = \{\neq_{23}\}$. On the one hand, if the query q is classified as positive, the clauses $\neg a(\geq_{12})$, $\neg a(\geq_{13})$ and $\neg a(\neq_{23})$ are added to T because $\kappa(q) = \{\geq_{12}, \geq_{13}, \neq_{23}\}$, which yields the theory $T \wedge \neg a(\neq_{23})$. On the other hand, if the query q is classified as negative, the clause $a(\geq_{12}) \vee a(\geq_{13}) \vee a(\neq_{23})$ is added to T, which by unit reduction yields the theory $T \wedge a(\neq_{23})$. Regardless of the classification of q, something new was learned. $\diamondsuit$

Though irredundant queries ensure that a nonempty portion of the version space is eliminated, some queries are "more" informative than others, by reducing larger portions of the hypothesis space. Technically, our Conacq architecture does not have access to the size of version spaces, but instead to a logical representation of them, for which the number of models is an approximation of the number of consistent concepts. From this viewpoint, the quality of a membership query is assessed here by the number of models it eliminates from the clausal representation of the version space. By $\|T\|$, we denote the number of distinct interpretations which satisfy a theory T, i.e. $\|T\| = |\mathit{models}(T)|$.

Definition 17. *Let* $\mathbf{B}$ *be a constraint bias,* E *a set of examples, and* T *be the clausal encoding of* $\mathbf{C_B}(E)$*. Given a real* θ *in* $[0, 1]$*, a membership query* $q = \boldsymbol{x}$ *is called* weakly *(resp.* strongly*)* θ-informative *with respect to* $\mathbf{C_B}(E)$ *if*

$$0 < \|T_y\| \leq (1 - \theta)\|T\|$$

for some (resp. all) $y \in \{0,1\}$*, where* T_y *is the encoding of* $\mathbf{C_B}(E \cup \{\langle \boldsymbol{x}, y\rangle\})$.

In other words, a membership query $q = \boldsymbol{x}$ is weakly θ-informative if, for at least one of the possible answers $y \in \{0,1\}$, the resulting theory T_y obtained by updating the version space with $\langle \boldsymbol{x}, y\rangle$ eliminates at least $\theta\|T\|$ models from the original encoding T of $\mathbf{C_B}(E)$. By contrast, $q = \boldsymbol{x}$ is strongly θ-informative if, whatever being the user's answer y, $\|T\|$ will be reduced by a factor of at least θ. Since $\|T_y\|$ must be positive, any (weakly or strongly) θ-informative query with $\theta \in [0,1]$ is irredundant.

Proposition 11. *Let* $\mathbf{B}$ *be a constraint bias,* E *be a set of examples, and* T *be the clausal encoding of* $\mathbf{C_B}(E)$*. Then, for any membership query* $q = \boldsymbol{x}$ *such that* $\kappa_{[T]}(\boldsymbol{x})$ *does not cover any clause in* T*, if* $|\kappa_{[T]}(\boldsymbol{x})| = t$ *then* q *is weakly* $(1 - 1/2^t)$*-informative, and if* $|\kappa_{[T]}(\boldsymbol{x})| = 1$ *then* q *is strongly* $1/2$*-informative.*

Proof. Let y be the answer to the query $q = \boldsymbol{x}$. If $y = 1$ then, by construction, $T_1 = T \wedge \{\neg a(c) : c \in \kappa_{[T]}(\boldsymbol{x})\}$. Since $\kappa_{[T]}(\boldsymbol{x})$ and the set of constraints occurring in T are disjoint, it follows that $\|T_1\| = 1/2^t\|T\|$, and hence, q is weakly $(1-1/2^t)$-informative. Furthermore, if $t = 1$ then let c be the unique constraint occurring in $\kappa_{[T]}(\boldsymbol{x})$. Because $T_0 = T \wedge a(c)$, $T_1 = T \wedge \neg a(c)$, and c does not occur in T, it follows that $\|T_0\| = \|T_1\| = 1/2\|T\|$, and hence, q is strongly $1/2$-informative. □

In light of the above result, we shall examine two strategies for generating membership queries, namely, the *optimistic* strategy, and the *optimal-in-expectation* strategy.

Optimistic Strategy. Intuitively, an *optimistic* query captures a large information gain when it is classified "in our favor", but conveys very little information when it is classified otherwise. Given $\boldsymbol{x} \in D^{|X|}$, the larger $|\kappa_{[T]}(\boldsymbol{x})|$, the more optimistic the query $q = \boldsymbol{x}$. By denoting $t = |\kappa_{[T]}(\boldsymbol{x})|$, we know by Proposition 11 that q is weakly $(1 - 1/2^t)$-informative. Specifically, if q is classified as positive, the resulting example prunes a large portion of $models(T)$ by assigning $|\kappa_{[T]}(\boldsymbol{x})|$ literals in T to 0; if q is classified as negative, it will only prune a tiny portion of $models(T)$ by just expanding T with the clause $\bigvee\{a(c) \mid c \in \kappa_{[T]}(\boldsymbol{x})\}$. The next example illustrates the acquisition process with an optimistic strategy.

Example 12. [Optimistic Queries] Suppose we wish to acquire the constraint network specified in Example 5, namely, the target network involves four variables, $\{x_1, \ldots, x_4\}$ defined over the domain $D = \{1,2,3,4\}$, a single constraint $x_1 \neq x_4$. Using the context defined in Example 5, the bias is $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$, and the training set is given by the three examples $e_1^+ = \langle 1,2,3,4\rangle, e_2^+ = \langle 4,3,2,1\rangle$ and $e_3^- = \langle 1,1,1,1\rangle$. The unique positive clause in T is $a(\neq_{12}) \vee a(\neq_{13}) \vee a(\neq_{14}) \vee a(\neq_{23}) \vee a(\neq_{24}) \vee a(\neq_{34})$. All other atoms in T are fixed to 0 because of e_1^+ and e_2^+. Here, $q_4 = \langle 1,1,1,3\rangle$ and $q_5 = \langle 1,1,3,1\rangle$ are two optimistic queries for T. $\kappa_{[T]}(q_4) = \{\neq_{12}, \neq_{13}, \neq_{23}\}$ and $\kappa_{[T]}(q_5) = \{\neq_{12}, \neq_{14}, \neq_{24}\}$ are both of size 3. According to the target network, q_4 will be classified positive by the user. Clauses $\neg a(\neq_{12})$, $\neg a(\neq_{13})$ and $\neg a(\neq_{23})$ are then added to T and its number of models is divided by 2^3. Consider q_5 instead of q_4: according to the target

Table 1: The *Optimal-in-expectation* query generation strategy of Example 13.

E	y	$\kappa_{[T]}(q)$	T
e_1, e_2, e_3	1,1,0		$T_{12} \wedge (a(\neq_{12}) \vee a(\neq_{13}) \vee a(\neq_{14}) \vee a(\neq_{23}) \vee a(\neq_{24}) \vee a(\neq_{34}))$
$q_4 = \langle 1,1,2,3\rangle$	1	$\{\neq_{12}\}$	$T_{12} \wedge (\neg a(\neq_{12})) \wedge (a(\neq_{13}) \vee a(\neq_{14}) \vee a(\neq_{23}) \vee a(\neq_{24}) \vee a(\neq_{34}))$
$q_5 = \langle 2,1,1,3\rangle$	1	$\{\neq_{23}\}$	$T_{12} \wedge (\neg a(\neq_{12})) \wedge (\neg a(\neq_{23})) \wedge (a(\neq_{13}) \vee a(\neq_{14}) \vee a(\neq_{24}) \vee a(\neq_{34}))$
$q_6 = \langle 2,3,1,1\rangle$	1	$\{\neq_{34}\}$	$T_{12} \wedge (\neg a(\neq_{12})) \wedge (\neg a(\neq_{23})) \wedge (\neg a(\neq_{34})) \wedge (a(\neq_{13}) \vee a(\neq_{14}) \vee a(\neq_{24}))$
$q_7 = \langle 1,3,1,2\rangle$	1	$\{\neq_{13}\}$	$T_{12} \wedge (\neg a(\neq_{12})) \wedge (\neg a(\neq_{23})) \wedge (\neg a(\neq_{34})) \wedge (\neg a(\neq_{13})) \wedge (a(\neq_{14}) \vee a(\neq_{24}))$
$q_8 = \langle 2,1,3,1\rangle$	1	$\{\neq_{24}\}$	$T_{12} \wedge (\neg a(\neq_{12})) \wedge (\neg a(\neq_{23})) \wedge (\neg a(\neq_{34})) \wedge (\neg a(\neq_{13})) \wedge (\neg a(\neq_{24})) \wedge (a(\neq_{14}))$

network, it is classified negative by the user. The clause $a(\neq_{12}) \vee a(\neq_{14}) \vee a(\neq_{24})$ is added to T and its number of models only decreases of $1/2^3$. This example reveals how unbalanced optimistic queries can be. $\diamondsuit$

Optimal-in-expectation Strategy. As observed in [12], a membership query is optimal if it reduces the size of the version space in half *regardless of how the user classifies it*. The aforementioned optimistic queries are clearly not optimal in this sense. A query $q = \boldsymbol{x}$ is called *optimal-in-expectation* if $\kappa_{[T]}(\boldsymbol{x}) = 1$. Since we are guaranteed that one literal will be fixed in T, whatever being the choice of y, q is strongly $1/2$-informative. The next example illustrates a sequence of optimal-in-expectation queries that are sufficient for establishing the version space convergence.

Example 13. [Optimal-in-Expectation Queries] Consider again the target network $x_1 \neq x_4$, using a vocabulary involving four variables $x_1, \ldots, x_4$, the domain $D = \{1,2,3,4\}$, and the bias $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$. Using the training set $E = \{e_1^+, e_2^+, e_3^-\}$, the unique positive clause in T is $\alpha = a(\neq_{12}) \vee a(\neq_{13}) \vee a(\neq_{14}) \vee a(\neq_{23}) \vee a(\neq_{24}) \vee a(\neq_{34})$. All other atoms in T are set to 0. Using T_{12} as an abbreviation of $\neg a(\leq_{12}) \wedge \ldots \wedge \neg a(\leq_{34}) \wedge \neg a(\geq_{12}) \wedge \ldots \neg a(\geq_{34})$, we can write $T = T_{12} \wedge \alpha$. Table 1 captures a sequence of queries which are optimal-in-expectation for $\mathbf{C}_{\mathbf{B}}(E)$. The first column specifies q, the second column indicates the user response y, the third column defines $\kappa_{[T]}(q)$, and the final column captures the update of T. For the query q_4, we know that $a(\neq_{12})$ will be fixed to a unit clause in the resulting theory, whatever being the user response. By repeating this invariant to all other queries, we are ensured that the version space will converge after processing q_8. $\diamondsuit$

In Example 13, we found a sequence of optimal-in-expectation queries for establishing convergence. However, it is not always possible to generate such queries, due to the interdependency between constraints in the bias. Example 14 illustrates the impossibility to generate an optimal-in-expectation query.

Example 14. [No Possible Optimal-in-Expectation Query] Consider the acquisition problem, using $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$, and with $x_1 = x_2 = x_3$ as a target network. After processing an initial positive example (for instance $e_1^+ = \langle 2,2,2\rangle$), the possible constraints in the version space are $\leq_{12}, \leq_{13}, \leq_{23}, \geq_{12}, \geq_{13}, \geq_{23}$. Here, every membership query q has either a $\kappa_{[T]}(q)$ of size 3 (if no variables equal), or a $\kappa_{[T]}(q)$ of size 2 (if two variables equal), or a $\kappa_{[T]}(q)$ of size 0 (if all three

variables equal). Therefore, no query with a $\kappa_{[T]}(q)$ of size 1 can be generated. Interdependency between constraints prevents us from generating such examples. ♢

Detecting whether a query strategy is feasible, or not, is intrinsically related to the "query generation" problem, examined below.

6.2 Query Generation Problem

The membership queries q of interest in this study only differ by the number $t = |\kappa_{[T]}(q)|$ of constraints which are not rejected by the clausal theory T. For optimal-in-expectation queries, we search for an assignment such that $t = 1$, whereas for optimistic queries, we search for an assignment with a larger t. When T already contains a non-unary clause α, we may want to shrink α, thus searching for an assignment such that $\kappa_{[T]}(q) \subset \mathit{constraints}(\alpha)$. The query generation problem is formulated as follows.

Definition 18 (Query Generation Problem)**.** *Given a bias* $\mathbf{B}$*, a training set* E*, an integer* t*, (and optionally a non-unary positive clause* α *in* T*), the* query generation problem *is to find a query* q *such that* $\kappa_{[T]}(q)$ *does not cover any clause in* T *(resp.* $\kappa_{[T]}(q) \subset \mathit{constraints}(\alpha)$*) and* $|\kappa_{[T]}(q)| = t$*. Deciding whether such a query* q *exists is called the* query existence problem*.*

Proposition 12 (Intractability of query generation)**.** *The query existence problem is NP-complete for any value of* t*, and thus the generation problem is NP-hard for any value of* t*.*

Proof. Membership. Given a bias $\mathbf{B}$, a theory T, (optionally a non-unary positive clause α in T), and an expected size t, checking that a given query q is a solution to the query generation problem is polynomial. Building $\kappa(q)$ is linear in $|\mathbf{B}|$. From $\kappa(q)$, building $\kappa_{[T]}(q)$, checking that $\kappa_{[T]}(q)$ does not cover any clause in T (resp. checking that $\kappa_{[T]}(q) \subset \mathit{constraints}(\alpha)$), and checking that $\kappa_{[T]}(q)$ is of size t are all linear in $|\kappa(q)| \cdot |T|$.

Completeness. We reduce the problem of coloring a graph with three colors (3Col) to the problem of the existence of a query q with $\kappa_{[T]}(q) = t$. Let (N, E) be a graph, where N is a set of n vertices and E a set of edges. v_i refers to the ith vertex and e_{ij} to the edge between vertices v_i and v_j. We transform the instance of graph coloring problem into the following query generation problem. The vocabulary is composed of a set of variables $X = \{x_i \mid v_i \in N\} \cup \{x_{n+1}, \ldots, x_{n+t+1}\}$, and a domain D such that $D(x_i) = \{1, 2, 3\}, \forall i \in [1..n]$, and $D(x_i) = \{1\}, \forall i \in [n + 1..n + t + 1]$. The constraint bias $\mathbf{B}$ is the set $\{\neq_{ij} \mid e_{ij} \in E\} \cup \{\neq_{i,i+1} \mid i \in [n + 1..n + t]\}$. (The clausal theory T could optionally contain the clause $\alpha = \bigvee_{\neq_{ij} \in \mathbf{B}} a(\neq_{ij})$. This could have happened after receiving the negative example $\langle 1, 1, \ldots, 1, 1\rangle$.) Suppose we want to generate a query q with t (and optionally α) as input. Such a query will necessarily be an assignment on X that violates $\neq_{i,i+1}$ for all $i \in \{n + 1, \cdots, n + t\}$ because $D(x_i) = D(x_{i+1}) = \{1\}$. Hence, the query will be any assignment that satisfies

Algorithm 4: The CONACQ.2 Algorithm

Input: a bias $\mathbf{B}$, background knowledge K , a strategy *Strategy*
Output: a clausal theory T encoding the target network

1 $T \leftarrow \varnothing$; $converged \leftarrow false$; $N \leftarrow \varnothing$
2 **while** $\neg converged$ **do**
3 $\quad q \leftarrow$ QUERYGENERATION($\mathbf{B}, T, K, N,$ *Strategy*)
4 $\quad$ **if** $q = nil$ **then** $converged \leftarrow true$
5 $\quad$ **else**
6 $\quad\quad$ **if** ASK(q) $= no$ **then** $T \leftarrow T \wedge \left(\bigvee_{c \in \kappa(q)} a(c)\right)$
7 $\quad\quad$ **else** $T \leftarrow T \wedge \bigwedge_{c \in \kappa(q)} \neg a(c)$
8 **return** T

all other constraints in $\mathbf{B}$ because we want the query to violate exactly t constraints. Now, by construction, an assignment on $X \setminus \{x_{n+1}, \ldots, x_{n+t+1}\}$ that satisfies all constraints in $\mathbf{B}$ corresponds to a 3-coloring of (N, E). Thus, the query existence problem has a solution if and only if the graph is 3-colorable. Our transformation is polynomial in the size of (N, E). Therefore, the query existence problem is NP-complete and the query generation problem is NP-hard. □

6.3 Description of Conacq.2

Based on a computational analysis of query generation, we are now ready to examine an active version of CONACQ.

As presented in Algorithm 4, the CONACQ.2 algorithm takes as input a constraint bias $\mathbf{B}$, background knowledge K for $\mathbf{B}$, and a query generation strategy used by function QUERYGENERATION (Algorithm 5) for the generation of queries. Each time we discover an interdependency among constraints that is not captured by the background knowledge K, we encode the interdependency as a logical nogood that is stored in a set N to avoid repeatedly discovering it. The algorithm returns a clausal theory T that encodes a constraint network representing the target concept.

CONACQ.2 starts from an empty theory T (Line 1) and iteratively expands it by an example $\langle \boldsymbol{x}, y \rangle$ formed by the query $q = \boldsymbol{x}$ generated in Line 3, and the user response y supplied in Line 6. The query generation process is implemented by the function QUERYGENERATION, which takes as input the bias $\mathbf{B}$, the current clausal theory T, the given background knowledge K, the current set of nogoods N, and the given strategy *Strategy*. If there exist irredundant queries, QUERYGENERATION returns an irredundant query q following *Strategy* as much as possible, that is, with $|\kappa_{[T]}(q)|$ as close as possible to the specified t. If there is no irredundant query, this means that we have reached convergence and we return the theory encoding the target network (Lines 4 and 8). Otherwise, the

query q is supplied to the user, who answers by *yes* (1) or *no* (0). If q is classified as negative by the user, we must discard from the version space all concepts that accept q. This is done by expanding the theory with the clause consisting of all literals $a(c)$ with c in $\kappa(q)$ (Line 6). Dually, if q is classified as positive, we must discard from the version space all concepts that reject q. This is done by expanding the theory with a unit clause $\neg a(c)$ for each constraint c in $\kappa(q)$ (Line 7).

6.4 Implementing our Strategies

The technique we propose to generate a query q is based on the following basic idea: define a constraint network expressing the strategy, that is, whose solutions are queries following the strategy. Based on Section 6.1, optimistic strategies and optimal-in-expectation strategies are both characterized by the number $t = |\kappa_{[T]}(q)|$ of constraints which are not inconsistent with (any concept of) the version space. To generate a query q that violates a number t of such constraints, we build a constraint network that forces t of those constraints to be violated. To be able to build this constraint network, we assume that for any constraint $c \in \mathbf{B}$, the complement $\overline{c}$ of c is available for building the constraint network. Thanks to the bounded arity assumption, the relation associated with $\overline{c}$ does not require an exponential space. As seen in Example 14 and Proposition 12, it may be the case that, due to interdependency between constraints, there does not exist any network in the version space that has a solution s with $|\kappa_{[T]}(s)| = t$. We then must allow for some *flexibility* ϵ in the number of constraints rejecting a query.

We implement the query generation problem in function QueryGeneration (Algorithm 5). The goal in function QueryGeneration is to find a good query following the given strategy. The algorithm starts by initializing the query q to *nil* and the clause α to the empty set (Line 1). Then, it enters the main loop in Line 2 if T is not a monomial. (Otherwise it immediately returns any irredundant query generated in Line 19.) The loop starts by testing whether a clause α is currently processed (i.e., α non-empty) or not. If α is empty and there still exist non-unary clauses in T that have not yet been marked, Line 4 reads such a (necessarily positive) clause in T. (The meaning of marked clauses is explained later.) We should bear in mind that a positive clause in T represents the set of constraints that reject a negative example already processed. So, we are sure that at least one of the constraints in *constraints*(α) must belong to the target network. Line 5 initializes the flexibility parameter ϵ to 0. ϵ measures how much we accept to deviate from the strategy. That is, a generated query q has to be such that $|t - \kappa_{[T]}(q)| \leq \epsilon$. The expected number t of constraints in the $\kappa_{[T]}(q)$ of the query q we will generate is set to the value corresponding to the strategy we use. For instance, in the optimal-in-expectation strategy, t is always set to 1.

Once ϵ, α, and t are set, if α is non-empty the purpose of the loop is to find a query q that will allow us to reduce the size of α. In Line 6 the Boolean *splittable* is computed. *splittable* will tell us if ϵ is small enough to produce a query that

Algorithm 5: QUERYGENERATION

Input: the bias $\mathbf{B}$, the clausal theory T, background knowledge K, a nogood set N, and a strategy *Strategy*
Output: a query q

1 $q \leftarrow nil$; $\alpha \leftarrow \varnothing$
2 **while** (T *is not a monomial*) *and* ($q = nil$) **do**
3 **if** ($\alpha = \varnothing$) *and* (T *contains non-unary unmarked clauses*) **then**
4 read a non-unary unmarked clause α in T
5 $\epsilon \leftarrow 0$; Assign t depending on *Strategy*
6 $splittable \leftarrow (\alpha \neq \varnothing) \wedge ((t + \epsilon < |\alpha|) \vee (t - \epsilon > 0))$
7 $F \leftarrow$BUILDFORMULA $(splittable, T, \alpha, t, \epsilon)$
8 **if** $models(F \wedge K \wedge N) = \varnothing$ **then**
9 **if** *splittable* **then** $\epsilon \leftarrow \epsilon + 1$
10 **else** $T \leftarrow T \cup \{a(c) \mid a(c) \in \alpha\} \setminus \{\alpha\}$; $\alpha \leftarrow \varnothing$
11 **else**
12 select $I \in models(F \wedge K \wedge N)$
13 **if** $Sol(\varphi(I)) = \varnothing$ **then**
14 **foreach** $CS \in ConflictSets(\varphi(I))$ **do**
15 $N \leftarrow N \cup \{\bigvee_{c \in CS} \neg a(c)\}$
16 **else**
17 select any q in $sol(\varphi(I))$
18 **if** ($\neg splittable$) *and* ($\alpha \neq \varnothing$) **then** mark α
19 **if** $q = nil$ **then** $q \leftarrow$ IRREDUNDANTQUERY(T)
20 **return** q

will reduce the size of α. In Line 7, the function BUILDFORMULA is called (see Algorithm 6). If *splittable* is true, this function returns a pseudo-Boolean formula F such that any solution s of any network corresponding to a model of F has a $\kappa_{[T]}(s)$ of size $t \pm \epsilon$ subsuming α. If α is non-splittable, constraints outside α can belong to $\kappa_{[T]}(s)$. If α is empty, some constraints not yet decided as member or non-member of the target network must belong to $\kappa_{[T]}(s)$. The formula F is defined on the atoms of T plus all atoms $a(\overline{c})$ such that $c \in \mathbf{B}$ and $\overline{c} \notin \mathbf{B}$. Once the formula F is generated, it is solved in Line 8. If $F \wedge K \wedge N$ is unsatisfiable, and α is still splittable, we must increase the flexibility ϵ (Line 9). If $F \wedge K \wedge N$ is unsatisfiable and α is non-splittable, this means that our current background knowledge K and nogoods N are sufficient to prove that α cannot be split because of the interdependency among its own constraints and independently of constraints outside α. Then, all literals corresponding to constraints of α are put in T, α is removed from T and is reset (Line 10) so that at the next iteration, a new clause will be selected.

In Line 12 a model I of $F \wedge K \wedge N$ is selected. If the constraint network $\varphi(I)$ has no solution (Line 13), this means that an interdependency between constraints of $\varphi(I)$ was not caught by K and the current stored nogoods N.

Algorithm 6: BUILDFORMULA

Input: a Boolean *splittable*, the clausal theory T, a clause α, an expected size t and the allowed flexibility ϵ
Output: a formula F

1 **if** $\alpha \neq \varnothing$ **then**
2 $F \leftarrow T \setminus \{\{\neg a(\overline{c})\} \mid c \in \mathit{constraints}(\alpha)\}$
3 **foreach** $c \in \mathbf{B} \mid \{\neg a(c)\} \notin T$ **do**
4 **if** $\mathit{splittable} \wedge c \notin \mathit{constraints}(\alpha) \wedge \overline{c} \notin \mathit{constraints}(\alpha)$ **then**
5 $F \leftarrow F \wedge (a(c))$
6 **if** $c \in \mathit{constraints}(\alpha)$ **then** $F \leftarrow F \wedge (a(c) \vee a(\overline{c}))$
7 **if** *splittable* **then**
8 $\mathit{lower} \leftarrow \mathit{max}(|\alpha| - t - \epsilon, 1)$
9 $\mathit{upper} \leftarrow \mathit{min}(|\alpha| - t + \epsilon, |\alpha| - 1)$
10 **else** $\mathit{lower} \leftarrow 1; \mathit{upper} \leftarrow |\alpha| - 1$
11 $F \leftarrow F \wedge \mathit{atLeast}(\mathit{lower}, \alpha) \wedge \mathit{atMost}(\mathit{upper}, \alpha)$
12 **else** $F \leftarrow T \setminus \{\{\neg a(\overline{c})\} \mid c \in \mathbf{B}, a(c) \text{ unset}\} \cup \{\bigvee_{c \in \mathbf{B}, a(c) \text{ unset}} a(\overline{c})\}$
13 **return** F

Lines 14-15 extract some conflicting sets of constraints (not necessarily all)[2] from $\varphi(I)$ and add the corresponding nogood clauses to N to avoid repeatedly generating models I' with this hidden inconsistency in $\varphi(I')$. Finally, if $\varphi(I)$ has solutions, such a solution q is selected in Line 17 and returned as the query to be asked to the user (Line 20). If α was non-splittable (and non-empty), Line 18 marks it as non-splittable to avoid selecting it again in a later call to QUERYGENERATION.

We now present function BUILDFORMULA (Algorithm 6). As already said above, BUILDFORMULA takes as input a Boolean *splittable*, a theory T, a clause α, an expected size t and an allowed flexibility ϵ. If α is not empty, the formula F is initialized to T minus the literals discarding the negation $\overline{c}$ of constraints c occurring in $\mathit{constraints}(\alpha)$ (Line 2). The idea is to build a formula F that only allows for a subset of the constraints in α to be violated. To monitor this number, we must be able to force a constraint or the negation of a constraint in $\mathit{constraints}(\alpha)$. This is why we remove the literals $\neg a(\overline{c})$ for constraints c in $\mathit{constraints}(\alpha)$. For each literal $a(c)$ not already negated in T (Line 3), if we are in the splittable mode and if c and $\overline{c}$ are both outside $\mathit{constraints}(\alpha)$, we force $a(c)$ to be true (Line 5). Hence, we force the constraint c to belong to the network $\varphi(I)$ for any model I of F, so that any solution s of $\varphi(I)$ will be rejected only by constraints in $\mathit{constraints}(\alpha)$ or by constraints already negated in T (so no longer in the version space). Thus, $\kappa_{[T]}(s) \subseteq \mathit{constraints}(\alpha)$. This will ensure that the query generated cannot extend α. If non-splittable, we do not force constraints outside $\mathit{constraints}(\alpha)$ to be satisfied.

[2]If we do not return all the conflict sets in $\varphi(I)$, the only negative effect is the possibility to generate again a network $\varphi(I')$ containing one of these missed conflict sets.

We now have to force the size of $\kappa_{[T]}(s)$ to be in the right interval. In the splittable mode, if $a(c)$ belongs to α we add the clause $(a(c) \vee a(\overline{c}))$ to F to ensure that either c or its complementary constraint $\overline{c}$ is in the resulting network (Line 6). $\overline{c}$ is required because $\neg a(c)$ only expresses the absence of the constraint c. $\neg a(c)$ is not sufficient to force c to be violated. We now just add two pseudo-Boolean constraints that ensure that the number of constraints from $\mathit{constraints}(\alpha)$ violated by solutions of $\varphi(I)$ will be in the interval $[t-\epsilon \,..\, t+\epsilon]$. This is done by forcing at least $|\alpha| - t - \epsilon$ and at most $|\alpha| - t + \epsilon$ literals from α to be set to true (lines 8–9 and Line 11). The 'min' and 'max' ensure we avoid trivial cases (i.e., no literal or all literals from α set to true). In the non-splittable mode, we just set the lower and upper bounds to the smallest and greatest non-trivial values (Line 10).

If α is empty, this means that we just want to generate a formula whose models will represent networks whose solutions are irredundant queries. The formula F is thus built in such a way that it satisfies T and violates at least one of the constraints not yet decided as member or not of the target network. For such constraints c we do not put $\neg a(\overline{c})$ in F to be able to force violation of one such constraint c. Line 13 returns F.

We finally present function IrredundantQuery (Algorithm 7). Function IrredundantQuery returns an irredundant query if one exists. This means that IrredundantQuery answers the convergence problem when it returns *nil*. As we want our technique to work whatever the background knowledge K is complete or not, we cannot apply Theorem 7 and thus cannot work at the level of the logical theory T to prove convergence. Fortunately, IrredundantQuery is called only when T is a monomial, which makes the test much easier. Line 1 builds the network C_L of constraints that have been learned as members of the target network. In lines 2 to 4, the function iteratively tries to randomly generate an irredundant query during a short amount of time that is sufficient in most cases to find one. If the time limit is reached, we enter a systematic way of generating such an irredundant query. Line 5 stores in the set *RuledOut* all constraints that have been ruled out of the version space. Then, for each constraint c that is neither ruled out nor set in C_L, Line 7 tries to find an assignment that satisfies C_L whereas violating c. If such an assignment s exists, it is returned as an irredundant query (Line 8). If no such assignment exists, this means that c is implied by C_L and $a(c)$ can be added to T without changing the version space (Line 9). If there does not exist any constraint that can be violated whereas satisfying C_L this means that T has converged. The function returns *nil* (Line 10).

Lemma 2. QueryGeneration *terminates and returns an irredundant query if there is one, nil otherwise.*

Proof. Termination. Termination is based on the fact that the loop of Line 2 cannot run forever. The loop starts by selecting an unmarked clause α if it exists (Line 4). Once a given α selected (or α still being the empty set), the loop necessarily executes one of the lines 9, 10, 15, or 17. We show that none

Algorithm 7: Function IrredundantQuery

Input: a monomial T
Output: an irredundant query q if it exists

1 $C_L \leftarrow \{c \in \mathbf{B} \mid a(c) \in T\}$
2 **while** *no more than 0.1 second has elapsed* **do**
3 $s \leftarrow$ any assignment in $sol(C_L)$
4 **if** $(\kappa_{[T]}(s) \neq \varnothing)$ **then return** s
5 $RuledOut \leftarrow \{c \in \mathbf{B} \mid \neg a(c) \in T\}$
6 **foreach** $c \in \mathbf{B} \setminus (C_L \cup RuledOut)$ **do**
7 $s \leftarrow$ any assignment in $sol(C_L \cup \{\overline{c}\})$
8 **if** $s \neq nil$ **then return** s
9 **else** $T \leftarrow T \cup \{a(c)\}$
10 **return** *nil*

of these instructions can be repeated forever for a given α.

Let us first suppose $\alpha \neq \varnothing$. As α is of finite size and ϵ is bounded above by $|\alpha|$ (Line 6), Line 9 cannot be executed an infinite number of times. Line 10 resets α to $\varnothing$ so it can be executed only once. In Line 15, a nogood is added to the set N of nogoods. This nogood is necessarily new, otherwise I would not have been a model of $F \wedge K \wedge N$. As nogoods are defined on a finite number of atoms (atoms $a(c)$ and $a(\overline{c})$ for each c in $\mathbf{B}$), we cannot add an infinite number of nogoods without making N unsatisfiable, and thus no longer entering Line 15. Line 17 is executed only once as it breaks the condition of the loop.

Let us now consider the case where $\alpha = \varnothing$. We show that lines 9 and 10 cannot be executed. If $\alpha = \varnothing$, function BuildFormula has executed its Line 12. When entering BuildFormula, T was not a monomial. Hence, there exists at least one atom $a(c)$ which is unset (that is, $T \wedge K \wedge N$ has two models I and I' with $I[a(c)] = 0$ and $I'[a(c)] = 1$).[3] Thus, the only clause in F which is not in T (added in Line 12) is non-empty. Furthermore, the atoms of this clause are unset in F by construction. As a result, the extra clause added to F in Line 12 cannot lead to inconsistency of $F \wedge K \wedge N$. We are thus guaranteed that $F \wedge K \wedge N$ is satisfiable and QueryGeneration goes directly to Line 12. Lines 15 and 17 cannot be executed an infinite number of times for the same reason as the case $\alpha \neq \varnothing$.

Finally, we have to prove that a given α cannot be selected several times in Line 4. Once a given α selected, the only way to stop processing it is by executing Line 10 or Line 17. If Line 10 is executed, we know α was not empty. Adding its literals to T makes it subsumed by T, and thus it can no longer appear in T. If Line 17 is executed, we have two cases. The first case is that we are in the splittable mode. By construction of F, the query q is such that $\kappa_{[T]}(q) \subset constraints(\alpha)$. This means that whatever the answer of the user, a

[3] We have to keep in mind that $T \wedge K \wedge N$ is maintained backboned.

clause will be generated by CONACQ.2 that will subsume α, thus discarding α from T. The second case, non-splittable mode, means that Line 18 is executed and α is marked, so that it will never be selected again in Line 4.

Correctness. QUERYGENERATION returns a query q generated in Line 17 or the result of IRREDUNDANTQUERY in Line 19. If a query q has been generated in Line 17, by construction of formula F in function BUILDFORMULA, q satisfies at least one constraint of each positive clause of T. In addition, it violates at least one constraint of α if α is not empty, or it violates at least one unset constraint. Thus, q is irredundant. If a query q has been generated in Line 19, it is irredundant by construction (see function IRREDUNDANTQUERY). If no irredundant query has been found by IRREDUNDANTQUERY, this means that none exists, and *nil* is returned. □

Theorem 8. CONACQ.*2 is correct and terminates.*

Proof. Correctness. At each non-terminal execution of the main loop, (Line 2), a new example (q, y) is generated, where q is the query returned by BUILDFORMULA, and y is the user's response given at Line 6. Since we took the assumption that the constraint acquisition setting is realizable, and membership queries are answered correctly, (q, y) is consistent with the the target concept, and hence, $\mathbf{C_B}(E)$ cannot collapse. By Definition 6, we also know that the clauses generated at Line 6 or at Line 7 yield a correct representation of $\mathbf{C_B}(E)$. So, by Theorem 5, the invariant of the main loop is that T is always satisfiable.

The set N only contains nogoods that represent inconsistent sets of constraints. Hence, N cannot lead to the deletion of any concept from the version space, except the inconsistent concept (which, by assumption, is not the target concept.) Thus, if K is a correct background knowledge, for any concept f_C consistent with the examples, $\varphi^{-1}(C)$ is a model of $T \wedge K \wedge N$. Now, by Lemma 2, Line 4 sets the Boolean *converged* to true if and only if there does not exist any irredundant query. So, the version space represented by T has converged.

Termination. By Lemma 2 we know that the query q generated by BUILDFORMULA is irredundant. So, by Proposition 10, the clauses added to T in lines 6 and 7 are not subsumed by T. As T involves a finite number of atoms, we cannot add new clauses forever without making T unsatisfiable. However, as an invariant of the main loop, we know that T is always satisfiable. This implies that the number of iterations of the main loop is finite. □

Now that we have presented the algorithm generating queries, we see that implementing our strategies is just a matter of setting t to the right value. In the optimal-in-expectation strategy, t must be set to 1 for every query. This corresponds to *near-misses* in machine learning ([35]), dividing by 2 the size of the version space whatever the answer from the user. In the optimistic strategy, we can set t to any value greater than 1 and smaller than the size of the clause α we want to shrink.

6.5 Example of the Query Generation Process

We now illustrate the query generation process on a small example.

Example 15. [Query Generation] We want to acquire the constraint network involving variables, $x_1, \ldots, x_3$, with domains $D(x_1) = D(x_2) = D(x_3) = \{1, 2, 3, 4\}$ and constraints $C_T = \{x_1 \leq x_2, x_2 \neq x_3\}$, with $\mathbf{B} = \mathbf{B}_{\leq,\neq,\geq}$ and $K = \varnothing$. In the table below, the first column reports the queries generated by the query generation procedure, the second column reports the classification of the query, and the third column is the update of T.

E	y	T
$q_1 = \langle 2, 2, 4\rangle$	1	$\neg a(\neq_{12}), \neg a(\geq_{13}), \neg a(\geq_{23})$
$q_2 = \langle 2, 1, 4\rangle$	0	$a(\leq_{12}), \neg a(\neq_{12}), \neg a(\geq_{13}), \neg a(\geq_{23})$
$q_3 = \langle 1, 1, 1\rangle$	0	$a(\leq_{12}), \neg a(\neq_{12}), \neg a(\geq_{13}), \neg a(\geq_{23}), a(\neq_{13}) \vee a(\neq_{23})$
$q_4 = \langle 1, 2, 2\rangle$	0	$a(\leq_{12}), \neg a(\neq_{12}), \neg a(\geq_{13}), \neg a(\geq_{23}), a(\neq_{13}) \vee a(\neq_{23}), a(\geq_{12}) \vee a(\neq_{23})$
$q_5 = \langle 1, 2, 3\rangle$	1	$a(\leq_{12}), \neg a(\neq_{12}), \neg a(\geq_{12}), \neg a(\geq_{13}), a(\neq_{23}), \neg a(\geq_{23})$
$q_6 = \langle 1, 2, 1\rangle$	1	$a(\leq_{12}), \neg a(\neq_{12}), \neg a(\geq_{12}), \neg a(\neq_{13}), \neg a(\geq_{13}), \neg a(\leq_{23}), a(\neq_{23}), \neg a(\geq_{23})$
$q_7 = \langle 2, 3, 1\rangle$	1	$a(\leq_{12}), \neg a(\neq_{12}), \neg a(\geq_{12}), \neg a(\leq_{13}), \neg a(\neq_{13}), \neg a(\geq_{13}), \neg a(\leq_{23}), a(\neq_{23}), \neg a(\geq_{23})$

At each execution of the main loop of Conacq.2, Line 3 calls the function QueryGeneration for producing a new query.

- At the beginning T is empty. The query generation process directly goes to Line 19 of QueryGeneration and calls the function IrredundantQuery. As T is empty, C_L is also empty and an arbitrary assignment is returned (Line 4). Let $q_1 = \langle 2, 2, 4\rangle$ be this first query. We have $\kappa(q_1) = \{\neq_{12}, \geq_{13}, \geq_{23}\}$. As q_1 is classified positive by the user, we add the clauses $\neg a(\neq_{12})$, $\neg a(\geq_{13})$, and $\neg a(\geq_{23})$ to T in Line 7 of Conacq.2.

- At the second call to QueryGeneration, T is still a monomial so we again directly call IrredundantQuery, which will return a query non-redundant with q_1. Let $q_2 = \langle 2, 1, 4\rangle$ be that second query. We have $\kappa(q_2) = \{\leq_{12}, \geq_{13}, \geq_{23}\}$, and $\kappa_{[T]}(q_2) = \{\leq_{12}\}$. As q_2 is classified negative by the user, we add the clause $a(\leq_{12})$ to T in Line 6 of Conacq.2.

- Again T is a monomial and by the same process, the query $q_3 = \langle 1, 1, 1\rangle$ is returned. We have $\kappa(q_3) = \{\neq_{12}, \neq_{13}, \neq_{23}\}$, and $\kappa_{[T]}(q_3) = \{\neq_{13}, \neq_{23}\}$. As q_3 is classified negative by the user, we add the clause $a(\neq_{13}) \vee a(\neq_{23})$ to T.

- At this point T is no longer a monomial. QueryGeneration selects the non-unary clause $\alpha = (a(\neq_{13}) \vee a(\neq_{23}))$ in Line 4 and calls function BuildFormula in Line 7 with the Boolean *splittable* set to true. In Line 2, function BuildFormula builds a formula F that contains all clauses of T. (Note that it is here irrelevant to remove literals $a(\overline{c})$ as our language does not contain any negation of constraints.) In addition, F forces all atoms which are neither already negated nor belonging to α to be true (Line 5). For each constraint c involved in α, F forces either $a(c)$ or $a(\overline{c})$. This will be used by the cardinality constraint in Line 11 to ensure that the number of constraints satisfying or rejecting a solution

of a network built from a model of F follows the strategy. In our case, whatever the strategy is, $t = 1$ because $|\alpha| = 2$. Thus, F is equal to $\{a(\leq_{12}), a(\geq_{12}), a(\leq_{13}), a(\leq_{23}), a(\neq_{13}) \vee a(=_{23}), a(=_{13}) \vee a(\neq_{23}), exactly(1, \alpha)\}$. A model of F is returned in Line 12 of QueryGeneration. But whatever this model is, $\varphi(I)$ is inconsistent because we cannot have $x_1 = x_2$, and x_3 equal to one of the variables x_1, x_2 and different from the other. (Observe that if K had contained the rules $a(\leq_{ij}) \wedge a(\geq_{ij}) \wedge a(\neq_{ik}) \rightarrow a(\neq_{jk})$, $F \wedge K$ would have been unsatisfiable.) After two similar executions of the loop, Line 15 of QueryGeneration has added the conflict sets $\{a(\leq_{12}), a(\geq_{12}), a(\neq_{13}), a(=_{23})\}$ and $\{a(\leq_{12}), a(\geq_{12}), a(=_{13}), a(\neq_{23})\}$ to N. We loop to Line 2, selecting again α and entering BuildFormula. The same formula is produced, but this time Line 8 of QueryGeneration detects that $F \wedge N$ is inconsistent. ϵ is incremented and we loop again. *splittable* is flipped to false and BuildFormula is called. When *splittable* is false, we do not force all atoms outside α to be satisfied. Thus, $F \wedge N$ is satisfiable and Line 12 of QueryGeneration produces a model I that necessarily contains either $\{a(\leq_{12}), a(\neq_{13}), a(=_{23})\}$ or $\{a(\leq_{12}), a(=_{13}), a(\neq_{23})\}$. Suppose I contains the former. $\varphi(I)$ has solutions, and one of them, $q_4 = \langle 1, 2, 2\rangle$, is returned. Before returning q_4, α is marked in Line 18 of QueryGeneration, not to be selected again for splitting it. q_4 is classified as negative and a new clause $a(\geq_{12}) \vee a(\neq_{23})$ is added to T.

- At the next loop, the clause $a(\neq_{13}) \vee a(\neq_{23})$ being marked, this is the clause $\alpha = a(\geq_{12}) \vee a(\neq_{23})$ which is selected. BuildFormula works as usual, forcing all non-set literals to true except those in α, for which exactly one has to be satisfied. F is satisfiable, and its unique model leads to a satisfiable network. Let $q_5 = \langle 1, 2, 3\rangle$ be the query generated. $\kappa_{[T]}(q_5) = \{\geq_{12}\}$. q_5 is classified positive. As a result, $a(\geq_{12})$ is set to false. By unit propagation, $a(\neq_{23})$ is set to true and C_L is equal to the target network.

- T is again a monomial. The two next loops of QueryGeneration call IrredundantQuery, which produces two, necessarily positive (because $C_L = C_T$), queries, $q_6 = \langle 1, 2, 1\rangle$ and $q_7 = \langle 2, 4, 1\rangle$, leading to the unit literals $\neg a(\neq_{13})$, $\neg a(\leq_{23})$ and $\neg a(\leq_{13})$.

- At this point IrredundantQuery is called a last time but cannot produce any irredundant query as all constraints in $B \setminus C_L$ have been ruled out (Line 6). Conacq.2 returns convergence.

$\diamondsuit$

7 Experimental Evaluation

We made some experiments to evaluate and compare the algorithms presented in this paper. We implemented the passive Conacq.1 and the active Conacq.2

algorithms. We first present the benchmark problems we used for our experiments and we give a brief description on how we obtain a background knowledge for each problem instance. Then, we report the results of acquiring these problems with CONACQ.1 and CONACQ.2. We evaluate the impact of the background knowledge and of the different strategies proposed in Section 6. Our tests were conducted on an Intel Core i7 @ 2.9 GHz with 8 Gb of RAM.

7.1 Benchmark Problems

Random. We generated binary random target networks $rand\_n\_d\_m$ with n variables, domains of size d, and m binary constraints.
Schur's lemma problem. (prob015 in [21]) We considered the Schur's lemma problem. The problem is to put n balls labeled 1 to n into three boxes so that for any triple of balls (x, y, z) with $x + y = z$, not all are in the same box.
Golomb Rulers. (prob006 in [21]) The problem is to find a ruler where the distance between any two marks is different from that between any other two marks. The target network is encoded with n variables corresponding to the n marks, and constraints of varying arity. For our experiments, we selected the instances from 4-marks to 8-marks rulers.
Sudoku. We used the Sudoku logic puzzle with 4×4 and 9×9 grids. The grid must be filled with numbers from 1 to 4 (resp. 1 to 9) in such a way that all rows, all columns and the 4 (resp. 9) non-overlapping 2×2 (resp. 3×3) squares contain the numbers 1 to 4 (resp. 1 to 9). The target network of the Sudoku has 16 variables (resp. 81 variables) with domains of size 4 (resp. 9) and 56 (resp. 810) binary $\neq$ constraints on rows, columns and squares.

For all these problems we used biases containing the basic arithmetic binary relations $\leq$, $<$, $\neq$, $=$, $\geq$, and $>$, plus, in some cases, quaternary constraints $|x_i - x_j| \neq |x_k - x_l|$, $|x_i - x_j| = |x_k - x_l|$, $allEqual(x_i, x_j, x_k, x_l)$, $notAllEqual(x_i, x_j, x_k, x_l)$, $allDiff(x_i, x_j, x_k, x_l)$, $notAllDiff(x_i, x_j, x_k, x_l)$, and all ternary constraints obtained when two indices among $\{i, j, k, l\}$ are the same. This gives us biases of size 80 for the smallest problems to more than 6000 for the largest. For each problem, the bias used is able to express the problem.

7.2 Background Knowledge Generation

We implemented a generator for the background knowledge to be used in CONACQ. Generating a complete background knowledge is generally too expensive in time and space as it requires generating a set of rules potentially exponential in space (see Section 5.1). Given a language Γ, our generator, called $gen\_rules(\#Vars, UB, maxSize)$, generates all first order rules involving a number $\#Vars$ of variables, with domains $[0, .., UB]$, and $maxSize$ constraints. For instance, with $\Gamma = \{\leq, \neq, \geq\}$, $gen\_rules(3, UB, 3)$ would produce the rule $a(\leq_{ij}) \wedge a(\leq_{jk}) \rightarrow a(\leq_{ik})$ whatever UB is. To ensure that generated rules are correct, the set $[0, .., UB]$ must be a superset of the domain D for all vocabularies $\langle X, D\rangle$ on which we want to apply the rule. For instance, the rule

Table 2: Background knowledge generation using *gen_rules* generator

Problems	K:	$\#Vars$	UB	$maxSize$	$\#rules$	time (sec.)
rand_3_5_2	K_1:	3	10	3	212	7.95
rand_5_6_10	K_2:	4	10	3	752	218.41
schurs_6	K_3:	4	10	2	56	8.83
schurs_8	K_4:	4	10	3	104	31.13
schurs_10	K_5:	5	10	3	230	371.77
golomb_4	K_6:	3	20	3	32	7.98
golomb_6	K_7:	4	40	3	48	320.95
golomb_8	K_8:	5	70	3	48	2192.89
sudoku_4x4	K_9:	4	10	3	54	8.13
sudoku_9x9	K_{10}:	4	10	4	114	15.62

$a(\neq_{ij}) \wedge a(\neq_{jk}) \rightarrow a(=_{ik})$ generated by $gen\_rules(3,1,3)$ is no longer correct when applied on the domain $D = \{0,1,2\}$. Given a vocabulary $\langle X, D\rangle$, the background knowledge K is obtained by taking each rule in *gen_rules* and by generating all ground instances of the rule on all subsets of $\#Vars$ variables in X. Such a K is correct background knowledge.

Table 2 reports the obtained background knowledge K_i for each problem instance using our $gen\_rules(\#Vars, UB, maxSize)$ generator. The last column reports the CPU time consumed to generate K_i. Take *schurs_6* as an example. $gen\_rules(4,10,2)$ takes 8.83 seconds to produce K_3, a background knowledge of 56 ground instances of rules.

7.3 Conacq.1: Passive Learning

Table 3 reports the evaluation of passive learning using Conacq.1. For each problem instance, we ran 100 times the learning process. All number reported in the table are averages of these 100 runs. For each run, we randomly generated a training set E of 100 examples. The line denoted by (1+100) represents the case where we process an extra positive example before processing E. The gray line represents the case where we use a background knowledge (K_i of Table 2). Each line of the table reports the size $|E|$ of the training set (smaller than 100 when convergence was reached before processing all examples), the number of positive examples $|E^+|$ and negative examples $|E^-|$, and the size of the target network C_T. We also report the sizes of the learned network C_L and the most specific network S after all examples have been processed. S contains all constraints c from the bias such that $\neg a(c)$ does not belong to T. The column S/C_T reports the ratio of solutions of S to C_T. $|E_C|$ represents the number of examples needed to reach convergence. For obtaining $|E_C|$ we have run Conacq.1 with 900 additional examples after the 100 first examples have been processed. The last column reports the average time in seconds needed to process an example

(average over the 100 first examples).

The first observation is the short time needed by CONACQ.1 to process an example from the training set. If we except *schurs_10* that is more expensive, it goes from 0.1 seconds for the smallest problems to 2 seconds for the largest (the sudoku 9×9 and its 810 constraints). These CPU times include backbone detection in the theory T representing the version space each time an example is processed.

The second thing we observe in Table 3 is that without a background knowledge, CONACQ.1 is never able to converge to the target network. This is not surprising as we only work on the clausal theory T. If the target network contains an implied constraint c, taking any model of T and flipping $a(c)$ from 0 to 1 or 1 to 0 will remain a model, thus preventing convergence (see Algorithm 3). Nevertheless, we observe that the most specific network S is often quite close or equivalent to the target network ($S/C_T \approx 100\%$). For instance, on *golomb_4*, only two constraints were learned after 100 examples, but the most specific network S contains 24 constraints that are equivalent to the target network C_T ($S/C_T = 100\%$). Providing a positive example at the beginning increases even more this behavior, especially on problems which are critically constrained.

The third observation we can make in Table 3 is that providing CONACQ.1 with some background knowledge often helps a lot to reach convergence. Take for instance *schurs_6*. Without background knowledge, CONACQ.1 is able to learn a C_L of 6 constraints in 1+100 examples, and the most specific network S has 9 constraints. By adding a background knowledge, we reach convergence after 70 examples. In this case the three missing constraints were implied by C_L. On *schurs_8* the scenario is almost the same. After 1+100 examples, CONACQ.1 with some background knowledge has learned 8 more constraints than without. It has not yet converged but will do after 123 examples. Half of our problems have converged before 1000 examples.

7.4 Conacq.2: Active Learning

We present some results on active learning using CONACQ.2. In our implementation of CONACQ.2 we made the following choices. In Line 4 of Algorithm 5, we always select the clause α of *minimum* size. In Line 14 of Algorithm 5, the function $ConflictSets$ returns a single conflict set. In the optimistic strategy, the parameter t is set to $\lfloor |\alpha|/2 \rfloor$. The following two subsections are respectively devoted to active learning without background knowledge and to active learning using some background knowledge.

7.4.1 Conacq.2 Without Background Knowledge

Table 4 displays results on active learning using CONACQ.2 without any background knowledge. We give a comparison between the two query generation strategies, *optimal-in-expectation* and *optimistic*, described in Section 6. For each strategy and for each problem instance, we report the size $|C_T|$ of the target network, the total number of queries $\#q$, and the numbers $\#yes$ and $\#no$

Table 3: Passive learning using CONACQ.1

Instance	$\|E\|$	$\|E^+\|$	$\|E^-\|$	$\|C_T\|$	$\|C_L\|$	$\|S\|$	S/C_T	$\|E_C\|$	time
rand_3_5_2	100	13	87	2	0	5	100%	—	0.14
	1+100	14	87	2	0	5	100%	—	0.13
K_1	1+20	4	17	2	5	5	100%	21	0.04
rand_5_6_10	100	0	100	10	0	60	0%	—	0.26
	1+100	1	100	10	0	30	36%	—	0.16
K_2	1+100	1	100	10	0	30	36%	818	0.94
schurs_6	100	55	45	6	6	9	100%	—	0.11
	1+100	56	45	6	6	9	100%	—	0.11
K_3	1+69	42	28	6	9	9	100%	70	0.11
schurs_8	100	31	69	12	11	22	51%	—	0.16
	1+100	32	69	12	11	21	55%	—	0.16
K_4	1+100	32	69	12	19	21	55%	123	0.81
schurs_10	100	5	95	20	0	66	1%	—	8.48
	1+100	6	95	20	0	36	2%	—	8.12
K_5	1+100	6	95	20	0	36	2%	323	10.97
golomb_4	100	1	99	18	2	24	100%	—	0.16
	1+100	2	99	18	2	24	100%	—	0.16
K_6	1+100	2	99	18	2	24	100%	>1000	0.22
golomb_6	100	0	100	110	0	240	0%	—	0.27
	1+100	1	100	110	0	120	100%	—	0.19
K_7	1+100	1	100	110	0	120	100%	>1000	0.37
golomb_8	100	0	100	385	0	798	0%	—	0.17
	1+100	1	100	385	0	399	100%	—	0.27
K_8	1+100	1	100	385	0	399	100%	>1000	0.99
sudoku_4x4	100	0	100	56	0	240	0%	—	0.42
	1+100	1	100	56	0	120	8%	—	0.59
K_9	1+100	1	100	56	0	120	8%	>1000	0.76
sudoku_9x9	100	0	100	810	0	6480	0%	—	1.72
	1+100	1	100	810	0	3240	0%	—	1.96
K_{10}	1+100	1	100	810	0	3240	0%	>1000	2.03

Table 4: Active learning using CONACQ.2 without background knowledge

		Optimal-in-expectation				**Optimistic**			
	$\|C_T\|$	$\#q$	$\#yes$	$\#no$	time	$\#q$	$\#yes$	$\#no$	time
rand_3_5_2	2	**11**	3	8	0.12	12	4	8	0.17
rand_5_6_10	10	**20**	3	17	0.37	68	3	65	0.24
schurs_6	6	53	30	23	0.16	**37**	19	18	0.14
schurs_8	12	134	33	101	4.71	**110**	28	82	4.73
schurs_10	20	342	57	285	7.13	**263**	48	215	6.98
golomb_4	18	**95**	3	92	0.05	108	2	106	0.05
golomb_6	110	**337**	1	336	0.51	370	1	369	0.76
golomb_8	385	**993**	1	992	11.24	1030	1	1029	17.13
sudoku_4 × 4	56	>1000	—	—	—	>1000	—	—	—
sudoku_9 × 9	810	>1000	—	—	—	>1000	—	—	—

of positive and negative answers. We also report the time in seconds needed to generate and to process a query. (We obviously do not count the time needed by the user to reply. This effort is measured by $\#q$.)

The first observation that we can draw from Table 4 is that generating a query generally takes more time than processing an example in CONACQ.1. Nevertheless, these times remain reasonable for a non-optimized implementation. They go from 0.05 to 11 seconds for the optimal-in-expectation strategy.

The second observation is that the number of queries asked by CONACQ.2 is dramatically reduced compared to CONACQ.1 (regardless of the query generation strategy that is used). For all instances of Random, Schur's Lemma and Golomb rulers, CONACQ.2 converges to the target network in less than 1000 queries whereas CONACQ.1 without background knowledge could not converge. For instance, for *golomb_4*, CONACQ.1 has learned only 2 constraints with 100 examples, whereas CONACQ.2 converges to the target network with only 95 queries. Unfortunately, we also observe that CONACQ.2, like CONACQ.1, is not able to learn Sudoku puzzles with 1000 queries.

When we compare the two query generation approaches among themselves, we see that the optimistic one is the best on under-constrained networks (e.g., *Schur's Lemma*). This confirms that when the probability of a *yes* answer is high enough, choosing a query that violates several constraints is a good strategy. However, on critically constrained problems, where the density of solutions is low, the optimal-in-expectation strategy becomes the best.

Figure 1 shows the impact of CONACQ.1 and CONACQ.2 on the version space for *golomb_4*. At the initial state, the learned network C_L is empty and the most specific S contains the whole bias (i.e., all possible constraints). Providing CONACQ.1 and CONACQ.2 with a positive example at the beginning divides the version space by half. CONACQ.1 learns 4 constraints when processing the 526 first examples. Thereafter, the version space remains stable in size ($|S| = 24, |C_L| = 4$) until we have processed the 1000 examples in the training set. In CONACQ.2, the learning process is much more efficient. S is reduced to the target network in 75 queries and convergence is proved in 95 queries.

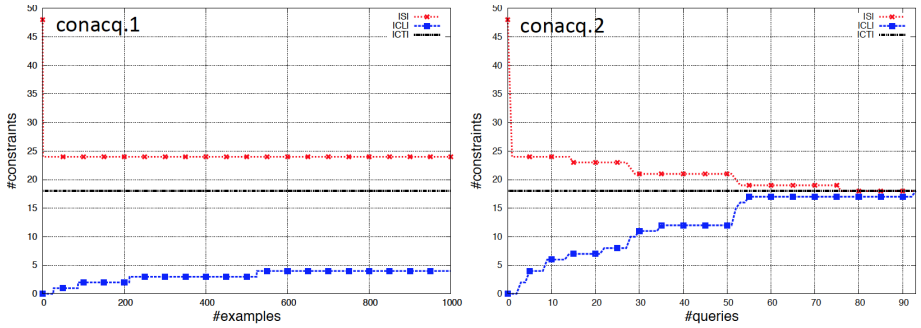

Figure 1: CONACQ.1 and CONACQ.2 learning *golomb_4*.

7.4.2 Conacq.2 with Background Knowledge

Table 5 reports our experiments on CONACQ.2 using a background knowledge. The strategy for query generation is optimal-in-expectation. In addition to the size of the learned network $|C_L|$, the number of queries ($\#q$, $\#yes$, $\#no$) and the average time needed for generating and processing a query, we report the background knowledge K_i of Table 2 we used.

The first observation is that, in CONACQ.2, the use of a background knowledge does not reduce the number of queries required for convergence. This is explained by the test of convergence in CONACQ.2, which is no longer done on the clausal theory T but on the actual constraint networks. Hence, implied constraints become irrelevant. (The small variations in number of queries between CONACQ.2 with and without background knowledge are due to the difference in queries generated.)

The second information we can draw from Table 5 is that the use of background knowledge speeds up significantly the generation of queries. It is, for instance, between 3 and 4 times faster on *schurs_10* or *golomb_8*. The reason for the speed up in generating queries is that rules in K prevent some of the fails in Line 13 of function QUERYGENERATION. Instead of repeatedly generating unsatisfiable constraint networks, a rule allows to directly detect unsatisfiability in the clausal formula (Line 8), avoiding the generation of a potentially large number of nogoods.

Finally, we observe that despite background knowledge, we are still unable to learn the sudoku puzzles. The reason for this bad behavior is probably that our background knowledges K_9 and K_{10} do not contain rules useful for learning the cliques of disequalities of the sudoku.

7.5 Discussion

In this section we have provided basic experiments on a bench of toy problems. Not surprisingly, CONACQ.1 often requires a huge number of examples to converge to the target network. The good news are that even when CONACQ.1

Table 5: Active learning using CONACQ.2 with background knowledge

instances	$\|C_T\|$	K	$\#q$	$\#yes$	$\#no$	time
$rand\_3\_5\_2$	2	K_1	10	3	7	0.11
$rand\_5\_6\_10$	10	K_2	21	2	19	0.15
$schurs\_6$	6	K_3	54	28	26	0.07
$schurs\_8$	12	K_4	128	30	98	1.32
$schurs\_10$	20	K_5	337	55	282	2.10
$golomb\_4$	18	K_6	98	3	95	0.05
$golomb\_6$	110	K_7	328	2	326	0.55
$golomb\_8$	385	K_8	994	1	993	3.15
$sudoku\_4 \times 4$	56	K_9	>1000	—	—	—
$sudoku\_9 \times 9$	810	K_{10}	>1000	—	—	—

has not converged, the most specific network in the version space is often quite close to the target network. CONACQ.2, by its active behavior, reduces a lot the number of examples needed to converge to the target network. Once the user has produced the effort to learn her target network, we can wonder about the re-usability of this network. By definition of our framework, the network will be usable as long as the user wants to solve instances of the problem on the same set of variables and with domains included in the domain of the vocabulary used to learn. For instance, once the sudoku network has been learned, any sudoku grid can be solved with this network, by simply specifying the singleton domains associated with the clues. However, one can object that even if re-usable, these numbers of queries remain too large if a human user is in the loop. This weakness of our basic constraint acquisition framework has led to several subsequent works based on or inspired by the CONACQ approach. These works are briefly discussed in the next section. Their common denominator is to use as much background knowledge as possible or to allow for more complex queries asked to the user. Background knowledge is a well-known technique in machine learning consisting in using properties of the problem to learn to reduce the bias as much as possible so that the version space to explore becomes small enough. Complex queries is another way to speed up convergence by allowing a more informative communication between the user and the learner so as to capture more information Complex queries is another way to speed up convergence by allowing a more informative communication between the user and the learner so as to capture more information.

8 Related Work

It is instructive to compare our constraint acquisition approaches with the classical learning approaches advocated in the machine learning literature. Notably, in the *Probably Approximately Correct (PAC)* learning introduced by Valiant

[36], the learner is merely passive and interacts with an oracle that supplies examples independently at random according to a fixed distribution. In this setting, an ϵ-good hypothesis is a concept for which the probability of misclassifying an example supplied by the oracle is at most ϵ. Very roughly, the goal of a PAC learning algorithm is to provide, with high probability $(1-\delta)$, an ϵ-good hypothesis. In our passive acquisition, the user can be viewed as an example oracle. Yet, the essential difference is that she is not required to supply examples according to a *fixed* distribution. The distribution is allowed to change during the acquisition process.

In [4], Beldiceanu and Simonis have proposed ModelSeeker, a passive constraint acquisition system. ModelSeeker is devoted to problems having a regular structure, such as matrix models. In ModelSeeker the bias contains global constraint from the global constraints catalog ([3]) whose scopes are the rows, the columns, or any other structural property ModelSeeker can capture. ModelSeeker also provides an efficient ranking technique that returns the best candidate constraints representing the pattern occurring on a particular set of variables (e.g., variables composing a row). As opposed to our passive learning approach, ModelSeeker handles positive examples only. Its very specific bias allows it to quickly find candidate models when the problem has a good structure. The counterpart is that it misses any constraint that does not belong to one of the structural patterns it is able to handle. Finally, its efficiency also comes from the fact that it does not prove convergence. It provides candidate constraints. It is the user who selects the constraints that fit the best the target problem.

In [25], Lallouet et al. have proposed a passive constraint acquisition system based on inductive logic programming. Their system is able to learn constraints from a given language that classify correctly the examples. To overcome the problem of the huge space of possible candidate constraint networks, their system requires the structure of the constraint network to be put in the background knowledge. They illustrate their approach on graph coloring problems. The positive/negative examples (i.e., correct and wrong colorations) are provided with their logical description using a set of given predicates. The background knowledge already contains all edges of the graph. These assumptions on the provided background knowledge make the approach questionable.

In [6], Bessiere et al. have proposed QuAcq, an active learner which, in addition to membership queries, is able to ask the user to classify partial queries. A partial query is an assignment that does not involve all variables. This is thus an extra condition on the capabilities of the user: Even if she is not able to articulate the constraints of her problem, she is able to decide if partial assignments of variables violate some requirements or not. The significant advantage of using partial queries is that our Theorem 4 about the impossibility to guarantee the existence of a polynomial sequence of queries to converge no longer holds. For instance, given a negative example, Conacq produces a non-unary positive clause on the candidate constraints for rejecting that example. Using partial queries, QuAcq is able, from the same negative example, to return exactly a constraint of the target network in a number of (partial) queries logarithmic

in the number of variables. The overall number of queries to converge is thus polynomial.

In [33], the approach used in CONACQ.2 has been extended to allow the user to provide *arguments*. Given a target network C, an argument arg is a set of constraints and/or variable assignments that the user labels as positive (i.e., $C \cup arg$ is satisfiable), negative (i.e., $C \cup arg$ is unsatisfiable), sufficient (i.e., $arg \models C$), or necessary (i.e., $C \models arg$). Such arguments allow the constraint acquisition process to converge more rapidly (with less examples) than CONACQ.2. However, it puts more of the effort of constraint acquisition on the shoulders of the user. For instance, deciding that an argument is positive is NP-complete, deciding that an argument is negative/sufficient/necessary is coNP-complete. In CONACQ.2, classifying a query as positive/negative is linear in the number of constraints in the target network C.

Finally, we would like to point out to the reader that what we call constraint acquisition in this paper is *not* comparable to works presented for instance in [10, 26]. These two papers propose techniques, which, given a model for a constraint problem, learn implied (global) constraints that can enhance the model (in [10]) or replace some simpler constraints of the model (in [26]). The goal is not to acquire a constraint model for the problem the user has in mind, but instead, to acquire a *better* model than the one the user proposed. In this case, better means a model with (global) constraints that are expected to propagate more or faster during search. This does not mean that CONACQ cannot be viewed as a reformulation tool. We could indeed initialize CONACQ with a bias containing only constraints for which we know efficient propagators, and let it interact with the user's model instead of the user directly. The learned network would then hopefully be more efficient to solve than the original one.

9 Conclusion

In this paper we have presented the basic architecture for acquiring constraint networks from examples classified by the user. We have formally defined the main constraint acquisition problems related to passive and active acquisition. We have closed several central complexity results that were still open. For instance, we have shown that consistency of the version space is polynomial to decide whereas convergence is intractable. We have also shown that constraint networks are not learnable in general with a polynomial number of membership queries. We have then proposed CONACQ, a system for acquiring constraint networks that uses a clausal representation of the version space. CONACQ is presented in a passive version (CONACQ.1), where the learner is only provided a pool of examples, and an active version (CONACQ.2), where the learner asks membership queries to the user. The clausal representation of the version space allows CONACQ to perform operations on the version space efficiently. For instance, the clausal representation is used to implement query generation strategies in active acquisition. Finally, we have compared experimentally the passive and active versions of CONACQ on a set of toy problems.

Acknowledgments.

This work was supported by a Ulysses Travel Grant from Enterprise Ireland and CNRS (Grant Number FR/2003/022). This work also received support from Science Foundation Ireland under Grant 00/PI.1/C075, from Agence Nationale de la Recherche under projects CANAR (ANR-06-BLAN-0383-02) and BR4CP (ANR-11-BS02-008), and from the European Union under project ICON (FP7-284715). The Insight Centre for Data Analytics at UCC is supported by Science Foundation Ireland through Grant No. SFI/12/RC/2289.

References

[1] D. Angluin. Queries revisited. *Theoretical Computer Science*, 313:175–194, 2004.

[2] K. Apt. *Principles of Constraint Programming.* Cambridge University Press, 2003.

[3] N. Beldiceanu, M. Carlsson, S. Demassey, and T. Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62, 2007.

[4] N. Beldiceanu and H. Simonis. A model seeker: Extracting global constraint models from positive examples. In *Proceedings of the Seventeenth International Conference on Principles and Practice of Constraint Programming (CP'12), LNCS 7514, Springer–Verlag*, pages 141–157, Quebec City, Canada, 2012.

[5] C. Bessiere, R. Coletta, E. Freuder, and B. O'Sullivan. Leveraging the learning power of examples in automated constraint acquisition. In *Proceedings CP'04*, pages 123–137, Toronto, Canada, 2004.

[6] C. Bessiere, R. Coletta, E. Hebrard, G. Katsirelos, N. Lazaar, N. Narodytska, C.G. Quimper, and T. Walsh. Constraint acquisition via partial queries. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI'13)*, pages 475–481, Beijing, China, 2013.

[7] C. Bessiere, R. Coletta, F. Koriche, and B. O'Sullivan. A SAT-based version space algorithm for acquiring constraint satisfaction problems. In *Proceedings of the European Conference on Machine Learning (ECML'05), LNAI 3720, Springer–Verlag*, pages 23–34, Porto, Portugal, 2005.

[8] C. Bessiere, R. Coletta, F. Koriche, and B. O'Sullivan. Acquiring constraint networks using a SAT-based version space algorithm. In *Proceedings AAAI'06*, pages 1565–1568, Boston MA, 2006. Nectar paper.

[9] C. Bessiere, R. Coletta, B O'Sullivan, and M. Paulin. Query-driven constraint acquisition. In *Proceedings IJCAI'07*, pages 44–49, Hyderabad, India, 2007.

[10] C. Bessiere, R. Coletta, and T. Petit. Learning implied global constraints. In *Proceedings IJCAI'07*, pages 50–55, Hyderabad, India, 2007.

[11] A. Blum and S. Rudich. Fast learning of k-term dnf formulas with queries. *Journal of Computer and System Sciences*, 51(3):367–373, 1995.

[12] N. H. Bshouty, S. A. Goldman, T. R. Hancock, and S. Matar. Asking questions to minimize errors. *J. Comput. Syst. Sci.*, 52(2):268–286, 1996.

[13] H. K. Büning and T. Lettman. *Propositional Logic: Decution and Algorithms*. Cambridge Tracts in Theoretical Computer Science, 48. Cambridge, 1999.

[14] D. Cohen and P. Jeavons. Tractable constraint languages. In R. Dechter, editor, *Constraint Processing*. Elsevier, 2003.

[15] R. Dechter. *Constraint Processing*. Morgan Kaufmann Publishers, San Francisco, CA, 2003.

[16] R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173(1):283–308, 1997.

[17] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[18] P. Flach. *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge, 2012.

[19] E. Freuder. Modeling: The final frontier. In *1st International Conference on the Practical Applications of Constraint Technologies and Logic Programming*, pages 15–21, London, UK, 1999. Invited Talk.

[20] A. M. Frisch, C. Jefferson, B. Martínez Hernández, and I. Miguel. The rules of constraint modelling. In *19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 109–116, Edinburgh, Scotland, 2005. Professional Book Center.

[21] I.P. Gent and T. Walsh. Csplib: a benchmark library for constraints. http://www.csplib.org/, 1999.

[22] C.A. Gunter, T-H. Ngair, P. Panangaden, and D. Subramanian. The common order-theoretic structure of version spaces and atms's. In *9th National Conference on Artificial Intelligence (AAAI)*, pages 500–505, 1991.

[23] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177–221, 1988.

[24] H. Hirsh, N. Mishra, and L. Pitt. Version spaces and the consistency problem. *Artificial Intelligence*, 156(2):115–138, 2004.

[25] A. Lallouet, M. Lopez, L. Martin, and C. Vrain. On learning constraint problems. In *Proceedings of the 22nd IEEE International Conference on Tools for Artificial Intelligence (IEEE-ICTAI'10)*, pages 45–52, Arras, France, 2010.

[26] K. Leo, C. Mears, G. Tack, and M. Garcia de la Banda. Globalizing constraint models. In *Proceedings of the Eighteenth International Conference on Principles and Practice of Constraint Programming (CP'13), LNCS 8124, Springer–Verlag*, pages 432–447, Uppsala, Sweden, 2013.

[27] T. Mitchell. Generalization as search. *AI Journal*, 18(2):203–226, 1982.

[28] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Ttroyansky. Determining computational complexity from characteristic 'phase transition'. *Nature*, 400:133–137, 1999.

[29] M. Paulin, C. Bessiere, and J. Sallantin. Automatic design of robot behaviors through constraint network acquisition. In *Proceedings of the 20th IEEE International Conference on Tools for Artificial Intelligence (IEEE-ICTAI'08)*, pages 275–282, Dayton OH, 2008.

[30] L. De Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer, 2008.

[31] L. De Readt and L. Dehaspe. Clasual discovery. *Machine Learning*, 26:99–146, 1997.

[32] F. Rossi, P. van Beek, and T. Walsh. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier, 2006.

[33] K.M. Shchekotykhin and G. Friedrich. Argumentation based constraint acquisition. In *Proceedings of the Ninth IEEE International Conference on Data Mining (ICDM'09)*, pages 476–482, Miami, Florida, 2009.

[34] B. Smith. Modelling. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 21. Elsevier, 2006.

[35] B. D. Smith and P. S. Rosenbloom. Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version spaces. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI'90)*, pages 848–853, Boston, MA, 1990. AAAI Press / The MIT Press.

[36] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.