Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Active Disjunctive Constraint Acquisition

Grégoire Menguy[1] , **Sébastien Bardin**[1] , **Nadjib Lazaar**[2] , **Arnaud Gotlieb**[3]
[1]CEA LIST, Université Paris Saclay, France
[2]LIRMM, University of Montpellier, CNRS, France
[3]Simula Research Laboratory, Oslo, Norway
{first.name}@cea.fr, arnaud@simula.no, nadjib.lazaar@lirmm.fr

Abstract

Constraint acquisition (CA) is a method for learning users' concepts by representing them as a conjunction of constraints. While this approach works well for many combinatorial problems over finite domains, some applications require the acquisition of disjunctive constraints, possibly coming from logical implications or negations. In this paper, we propose the first CA algorithm tailored to the automatic inference of disjunctive constraints, named DCA. A key ingredient there, is to build upon the computation of maximal satisfiable subsets. We demonstrate experimentally that DCA is faster and more effective than traditional CA with added disjunctive constraints, even for ultra-metric constraints with up to 5 variables. We also apply DCA to precondition acquisition in software verification, where it outperforms the previous CA-based approach PRECA, being 2.5 times faster. Specifically, in our evaluation DCA infers more preconditions in just 5 minutes than PRECA does in an hour, without requiring prior knowledge about disjunction size. Our results demonstrate the potential of DCA for improving the efficiency and scalability of constraint acquisition in the disjunctive case, enabling a wide range of novel applications.

1 Introduction

Constraint programming (CP) (Rossi, Van Beek, and Walsh 2006) has made considerable progress over the last forty years, becoming a powerful paradigm for modelling and solving combinatorial problems. However, modelling a problem as a constraint network still remains a challenging task that requires expertise in the field. As part of knowledge acquisition, several constraint acquisition (CA) systems have been introduced to support the uptake of constraint technology by non-experts. Based on querying an oracle that classifies samples as solutions and non-solutions, an active CA system automatically learns a constraint system that represents a concept the user has in mind. This is an active field of research, with many proposed extensions, for example allowing partial queries (Bessiere et al. 2013; Bessiere et al. 2020), incomplete answers (Tsouros, Stergiou, and Bessiere 2020), arguments (Shchekotykhin and Friedrich 2009), or acquisition of qualitative constraint networks (Belaid et al. 2022).

Handling disjunctions. Still, classical CA is naturally dedicated to the acquisition of conjunctions of constraints. This thwarts its application to broader contexts. Indeed, some problems of interest such as *ultrametric constraints* (Gent et al. 2003), some global constraints (Régin 2010) or recently *precondition acquisition* (Menguy et al. 2022) are disjunctive by essence. As current CA systems lack a principled framework to handle disjunctive behaviours, disjunctive constraints have to be either manually added to the input language of CA or automatically generated (up to a given size) by an enumeration algorithm. For instance, in precondition acquisition, function `mbedtls_aes_setkey_enc` from the `mbedtls` cryptographic library [1] requires adding a disjunctive precondition of 7 atomic constraints. By using an incremental enumeration of all disjunctive constraints, this precondition can hardly be generated in less than two hour of CPU time on a standard machine. Deciding which disjunctions are needed in classical CA systems or finding an efficient generation method of relevant disjunctive constraints is an important but still open problem that can be viewed as a search problem in a lattice-organized search space.

Contributions. In this paper we propose a new CA approach that acquires networks with disjunctive constraints in a principled manner. Our contributions are threefold:

- We propose DCA the first inference framework extending CA to infer conjunctions of disjunctive constraints (Section 3). It elegantly leverages maximal satisfiable subsets (MSS) enumeration (Liffiton and Malik 2013) to render CA more expressive to efficiently handle disjunctions;
- We prove that DCA enjoys good theoretical properties (Section 3.3). Especially, it shares the same guarantees as usual CA, showing that DCA is an appropriate generalization of CA for disjunctive contexts. Notably, DCA always terminates, generates informative queries only, and returns a result that agrees with all tested queries. Moreover, if the target concept can be expressed as a conjunction of disjunctive constraints from the input set of constraints, then DCA infers it (Theorem 1);
- We evaluate[2] in Section 4 our new learning framework DCA over two different benchmarks: a general bench-

[1]https://github.com/Mbed-TLS/mbedtls
[2]Artifacts available: https://github.com/binsec/dca

mark formed of randomly generated disjunctive constraints, ultrametric and domain constraints and a precondition inference benchmark from (Menguy et al. 2022). In particular, we show that DCA is especially well-suited for precondition inference as queries are automatically answered by systematically calling the program under analysis. Regarding the above-mentioned example from `mbedtls` that timeouts in 2h with traditional CA, DCA infers the disjunctive precondition in less than 2min (102sec in average).

To the best of our knowledge, DCA is the first active CA method to handle disjunctive problems. It makes CA more flexible, removing the need to model disjunctive behaviours as a unique constraint – which needs expertise from the user.

2 Background

We now describe the necessary background on constraint acquisition, its applications and MSS enumeration, which will be leveraged throughout this paper.

2.1 Constraint Acquisition (CA)

CA process (Bessiere et al. 2013; Bessiere et al. 2017) can be seen as an interplay between a CA-Agent (learner) and a source of information (*user* or *oracle*[3]). For that, the CA-Agent needs to share some common *vocabulary* to communicate with the user. This vocabulary is a finite set of variables X taking values in a finite domain D. A *constraint* c defined over a subset of variables from the vocabulary is a relation specifying which values of these variables are allowed. A *constraint network* (CN) is a set C of constraints. An example $e \in D^{|X|}$ *satisfies* a constraint c if its projection on c variables is in the solutions set of c, noted, $sol(c)$. An example e *is a solution of* C iff it satisfies all constraints in C. Thus, a CN represents a conjunction of constraints.

In addition, the CA-Agent owns a *language* Γ of bounded arity relations from which it can build constraints on specified sets of variables from the vocabulary. The *constraint bias*, denoted by B, is a set of constraints built from Γ on (X, D), from which the CA-Agent builds a constraint network. We say that a CN C is representable by B iff $C \subseteq B$. A set of constraint $\mathcal{C}$ (e.g., the bias) is said to be complete iff for each constraint $c \in \mathcal{C}$, its negation $c' \equiv \overline{c}$ is in $\mathcal{C}$ too. A *concept* is a Boolean function f over $D^{|X|}$. A *representation* of a concept f is a constraint network C for which $f^{-1}(true)$ equals the solutions set of C. A *membership query* (or simply a *query*) takes an example e and asks the user to classify it. The answer is yes iff e is a solution of the user concept. A query e is, thus, said to be positive (resp. negative) if its answer is yes (resp. no) and is noted e^+ (resp. e^-) to emphasize its classification. For any example e, $\kappa(e)$ denotes the set of all constraints in B rejecting e.

We now define *convergence*. Given a set E of examples labeled by the user yes or no, we say that a network C agrees with E if C accepts all examples labelled yes in E (noted E^+) and does not accept those labelled no (noted E^-). The learning process has *converged* on the network $L \subseteq B$ if **(i)** L agrees with E and **(ii)** for every other $L' \subseteq B$ agreeing with E, we have $L' \equiv L$.

CONACQ is a CA-Agent that submits *membership queries* to a user (Bessiere et al. 2017). CONACQ uses a concise representation of the learner's version space into a clausal formula. Formally, any constraint $c \in B$ is associated with a Boolean atom $a(c)$ stating if c must be in the learned network. CONACQ starts with an empty theory and iteratively expands it by generating and submitting to the user an informative example. An informative example ensures to reduce the version space independently from the user's answer. If no informative example remains, this means that we converged and CONACQ returns the learned network.

2.2 User-based Handling of Disjunctive Constraints

CA has been used in different scenarios, from scheduling (Beldiceanu and Simonis 2012) to robotics (Paulin, Bessiere, and Sallantin 2008). Still, the high number of queries that must be classified by the user and the limitation to conjunctive constraints limit its practical usage.

In prior work, we recently show (Menguy et al. 2022) that CONACQ-like CA is especially well adapted to precondition inference, a program analysis task. Indeed, in such a case, the number of queries is not a limitation, because queries are automatically answered by calling the compiled program over a set of inputs. This led to PRECA, the first contract inference framework based on CA. However, state-of-the-art CA can only infer conjunctions of constraints. Thus, we proposed to let the user include useful disjunctions in the bias directly, so that disjunctive preconditions can be infered. However, deciding which disjunctions shall be included in the bias remains on the shoulder of the user. As an help to the user, given the analyzed program function F, PRECA includes all Horn clauses of size $\leq max(i, 1) + 1$, where i is the number of F integer inputs. While such a heuristics works on simple examples, it does not scale well and cannot handle complex examples, as the number of disjunctive constraints rapidly explodes.

2.3 Maximal Satisfiable Subset (MSS)

Maximal satisfiable subsets are extensively used in knowledge comprehension. The following presents basic definitions which will be used all along the paper.

Definition 1 (MSS). *Given a UNSAT set of constraints $\mathcal{C}$, $M \subseteq \mathcal{C}$ is a maximal satisfiable subset (MSS) of $\mathcal{C}$ iff M is SAT and for all constraints $c \in \mathcal{C} \setminus M$, $M \cup \{c\}$ is UNSAT. We note* $\text{MSS}_{\mathcal{C}}$ *the set of all the MSS of $\mathcal{C}$.*

Example 1. *Let $X = \{x\}$, $D = 0..9$ and $\mathcal{C} = \{x \geq 2, x < 2, x \geq 8, x < 8\}$. $\mathcal{C}$ has 3 MSSes: $\{x < 2, x < 8\}$, $\{x \geq 2, x < 8\}$, and $\{x \geq 2, x \geq 8\}$.*

Enumeration. Multiple MSS enumeration algorithms have been proposed (Bailey and Stuckey 2005; Liffiton and Malik 2013; Liffiton and Sakallah 2005; Van Loon 1981; Gleeson and Ryan 1990). Some are specialized to handle specific constraint types (Van Loon 1981; Gleeson and

[3]The standard CA terminology employs *user*, yet we sometimes use *oracle* to highlight the fact that it can be automated.

Ryan 1990) while others are *constraint-agnostic* (Bailey and Stuckey 2005; Liffiton and Malik 2013; Liffiton and Sakallah 2005). In the former case, enumeration often relies on linear programming constructions. In the latter case, such specific constructs are not available. Thus, approaches like DAA (Bailey and Stuckey 2005) traverse the search space by generating partial candidates through hitting sets computation. On the other hand, MARCO (Liffiton and Malik 2013) maintains a power set map representing the search space as a boolean formula. They generate candidate solutions which are extended into MSSes if they are satisfiable.

3 Disjunctive Constraint Acquisition (DCA)

After having presented how the usual hypothesis and definitions of CA translate into the context of disjunctive CA, we describe DCA in detail, which infers disjunctive constraint network in an active manner. As far as we know, DCA is the first CA method specifically designed for disjunctive scenarios. We then demonstrate that DCA offers the same strong theoretical guarantees as traditional CA methods. Lastly, we propose several optimizations to accelerate inference.

3.1 From CA to DCA

CA assumes that the target concept C_T, given a bias B, is representable by B, i.e., can be expressed as a conjunction of constraints from B. It can then provide clear correctness and termination guarantees (Bessiere et al. 2013; Bessiere et al. 2017). However, this assumption limits its expressivity, making it difficult to infer disjunctive behaviors since the disjunctions must be present in B, significantly increasing its size and hampering inference. In this work, we limit the bias to atomic constraints, without including disjunctive constraints into B, unlike classical CA. During the inference process, the disjunctions will be automatically formed and inferred.

Definition 2 ($\bigvee$-representability)**.** *Let C be a CN and $\mathcal{C}$ be a set of constraints. We say that C is $\bigvee$-representable by $\mathcal{C}$ iff it is composed of constraints that are either in $\mathcal{C}$ or are disjunctions of constraints from $\mathcal{C}$. More formally, C is a CN s.t., for all $c_i \in C$, c_i is in $\mathcal{C}$ or there exists $c_{i1}, ..., c_{ik} \in \mathcal{C}$, s.t., $c_i = c_{i1} \vee ... \vee c_{ik}$*

$\bigvee$-representability generalizes usual *representability* to handle disjunctions of constraints. In the following, we aim to infer a CN C_T that is $\bigvee$-representable by B.

Classical CA not only infers the target concept as a conjunction of constraints but also ensures that it only generates informative (i.e., irredundant) queries, which can prune the search space regardless of their classification. In our scenario, an informative query is defined as follows, given a bias B and a set of queries E:

Definition 3 (Informative query)**.** *Given a bias B and a set of queries E. A query $e \notin E$ is informative if it is not classified the same way by all CN $\bigvee$-representable by B, that agree with E.*

This definition of informativeness matches the one given in classical CA (Bessiere et al. 2017). The only difference is that the considered CN can contain disjunctions. For example, let the bias B be $\{x > 0, x \leq 0, y > 0, y \leq 0\}$ and the set of positive queries $E = \{e_1^+ = (x \leftarrow 1, y \leftarrow 1), e_2^+ = (x \leftarrow 0, y \leftarrow 0)\}$. In classical CA, where only conjunctions of B are allowed, no informative queries are left apart. However, in our case, the query $e_3 = (x \leftarrow 1, y \leftarrow 0)$ is informative as $x \leq 0 \vee y > 0$ classifies it negatively while $x > 0 \vee y \leq 0$ classifies it positively.

3.2 The DCA Framework

To infer disjunctions automatically with good guarantees, we rely on MSS enumeration (Liffiton and Malik 2013).

Observe that, in Example 1, $\text{MSS}_\mathcal{C}$ forms a partition of $D^{|X|}$. That is, all $e \in D^{|X|}$ are solutions of one and only one MSS of $\mathcal{C}$. This is not always the case. For example, the constraint set $\mathcal{C} = \{x > 2, x = 2\}$ has only 2 MSSes: $x > 2$ and $x = 2$, which do not induce a partition of $D^{|X|}$. In fact, such property holds for all complete constraint sets.

Proposition 1 (Partition)**.** *Let $\mathcal{C}$ a complete nonempty set of constraints over a domain $D^{|X|}$. Then $\text{MSS}_\mathcal{C}$ induces a partition of $D^{|X|}$.*

Proof. (sketch.) We prove that all elements of $D^{|X|}$ are a solution of exactly one MSS of $\mathcal{C}$. Let $e \in D^{|X|}$. First, e cannot be a solution of two distinct MSS M_1 and M_2. Otherwise, $M_1 \cup M_2$ would be SAT (contradicts with MSS definition). Second, note that there is always an MSS M s.t., $e \models M$. Indeed, $\mathcal{C}$ being complete, $M = \{c \in \mathcal{C} \mid e \models c\}$ is an MSS accepting e (for each $c \in \mathcal{C}$, either $e \models c$ and $c \in M$ or $e \not\models c$ and $\overline{c} \in M$, so $M \cup \{c\}$ is UNSAT). ☐

Thus, given $e \in D^{|X|}$, there exists a unique $M \in \text{MSS}_\mathcal{C}$ s.t., $e \models M$. We note it $\text{MSS}_\mathcal{C}(e)$ and can be understood as the most precise approximation of e modulo $\mathcal{C}$. When C_T is $\bigvee$-representable by B, MSSes become especially useful. Indeed, being the most precise approximations of elements in $D^{|X|}$, we know that all elements in a MSS share the same classification. Checking only one element per MSS is then enough to deduce the classification of all elements of $D^{|X|}$.

Proposition 2 (MSS classification)**.** *Let $\mathcal{C}$ be a nonempty, complete set of constraints and C_T be the target constraint network $\bigvee$-representable by $\mathcal{C}$. For each $M \in \text{MSS}_\mathcal{C}$, all elements $e \in sol(M)$ share the same classification w.r.t. C_T.*

Proof. (sketch.) Let $M \in \text{MSS}_\mathcal{C}$ and e_1, $e_2 \in sol(M)$. Then, for all, $c \in M, e_1 \models c \wedge e_2 \models c$. Moreover, for all $c \in \mathcal{C} \setminus M, e_1 \not\models c \wedge e_2 \not\models c$ (otherwise $M \cup \{c\}$ would be SAT and M would not be an MSS). As such, we know that for all $c \in \mathcal{C}, e_1 \models c \Leftrightarrow e_2 \models c$. As such, for each disjunction δ in $C_T, e_1 \models \delta \Leftrightarrow e_2 \models \delta$. Thus $e_1 \models C_T \Leftrightarrow e_2 \models C_T$. ☐

From Proposition 2 directly follows that given a nonempty complete constraint set $\mathcal{C}$, checking only one solution of each MSS of $\mathcal{C}$ is enough to know the classification of the full domain. This leads to the DCA algorithm presented in Algorithm 1. It takes a nonempty complete bias and enumerates all its MSS. For each enumerated MSS M, a *membership query* $e \in sol(M)$ is picked to check its classification. If it is not classified positively, the result L is

Algorithm 1: DCA

In : A nonempty complete bias B
Out : A CN L that agrees with all examples
1 **begin**
2 $\quad L \leftarrow \top$
3 $\quad$ **foreach** $M \in \textsc{Mss}_B$ **do**
4 $\quad\quad$ *pick* $e \in sol(M)$
5 $\quad\quad$ **if** ask$(e) \neq yes$ **then**
6 $\quad\quad\quad L \leftarrow L \wedge \neg M$
7 $\quad$ **return** L

updated, at line 6, to remove $sol(M)$ from it – disjunctions are introduced here by adding $\neg M$ i.e., the negation of M. When all MSS have been checked, DCA returns the solution L. The DCA algorithm relies extensively on MSS enumeration. It may rely on any enumeration algorithm ensuring termination, correctness, and completeness, like DAA (Bailey and Stuckey 2005) or MARCO (Liffiton and Malik 2013).

3.3 Theoretical Analysis

We now show the guarantees of DCA: it terminates, asks only informative queries, returns a concept that agrees with all checked queries and is correct under some hypothesis. Then we compare DCA guarantees to usual CA ones.

Proposition 3 (Termination)**.** DCA *terminates.*

Proof. It directly follows the fact that B is finite and MSS enumeration terminates. □

Proposition 4. DCA *generates informative queries only.*

Proof. (sketch.) Let B be the bias, E be the set of already generated queries, and L the inferred concept until now. Let e^* be a newly generated query. The query e^* is associated with an MSS of B: $\textsc{Mss}_B(e^*)$. For each query $e \in E$, $e \not\models \textsc{Mss}_B(e^*)$. Thus L and $L \cup \{\neg\textsc{Mss}_B(e^*)\}$ agree with E but classify e respectively as positive and as negative. □

Proposition 5. DCA *returns a constraint network L that agrees with each classified queries.*

Proof. (sketch.) Let B be a complete bias and E be the generated query set. Then for each $e^- \in E^-$, we know that $\neg\textsc{Mss}_B(e^-)$ has been added to L at line 6. Thus, $e^- \not\models L$. Furthermore, for each $e^+ \in E^+$, we know that $\neg\textsc{Mss}_B(e^+)$ has not been added to L. Moreover, for all other MSS of B different from $\textsc{Mss}_B(e^+)$, noted M, we know that $e^+ \not\models M$ – as the set of MSS induces a partition of the domain. Thus $e^+ \models \neg M$ and $e^+ \models L$. □

Theorem 1 (Correctness)**.** *Given a complete atomic bias B and a target concept C_T $\bigvee$-representable by B.* DCA *returns a network L s.t. $L \equiv C_T$.*

Proof. (sketch.) DCA enumerates all MSS of B, picks a unique query inside it, and updates the candidate solution according to the classification. From Proposition 2, we know that all elements of an MSS of B share the same classification. Thus, at line 5, we not only know the classification of e but also of all elements of $\textsc{Mss}_B(e)$. Moreover, from Proposition 1, we know that the domain is partitioned by the set of MSSes of B. So through MSS enumeration at line 3, DCA deduces the classification of each element of the domain. Moreover, as DCA terminates (Proposition 3) and agrees with all queries (Proposition 5), it is correct. □

DCA vs. CA. Usual constraint acquisition methods (Bessiere et al. 2013; Bessiere et al. 2017) enjoy clear theoretical guarantees. They terminate, generate informative queries only, and return a CN agreeing with all seen queries. Moreover, if the target concept can be represented as a conjunction of constraints from the bias C_T, then constraint acquisition returns a CN L equivalent to C_T. DCA *enjoys the same good theoretical guarantees* adapted to the disjunctive scenario. Still, as DCA relies on a weaker hypothesis, it may ask more queries than standard CA. While it may be an important limitation in some contexts involving human experts, it is not the case when the oracle (user) can be automated (Menguy et al. 2022).

Complexity. Learning constraint networks through membership queries is a well-known challenging task (Bessiere et al. 2017). Our approach addresses this problem while introducing an additional level of complexity with disjunctions. The computational complexity of DCA is primarily determined by the worst-case scenario of exponential-time enumeration of the MSS. Still, we demonstrate in our experimental evaluation that DCA can already handle in an efficient way practical code analysis tasks of significant interest. Still, future directions could focus on reducing such complexity under certain assumptions.

4 Experimental Evaluation

Our experimental evaluation aims to answer the following three Research Questions:

RQ1 *How does* DCA *compare to the classical CA* CONACQ *approach?* We compare CONACQ and DCA to infer disjunctive constraints. As CONACQ cannot automatically infer disjunctions, we added all disjunctions of size up to some threshold into its bias.

RQ2 *Can* DCA *be leveraged for precondition inference?* We apply DCA to the precondition inference application of CA (Menguy et al. 2022). We especially compare DCA to PRECA over a dataset of real-world functions. We also evaluate how the approaches are impacted by the disjunction size present in the target preconditions – including the purely conjunctive case.

RQ3 *How is* DCA *impacted by bias size?* We apply DCA over three bias sizes and compare it to PRECA over the precondition inference use-case.

Implementation We implemented DCA in JAVA, and relied on the CHOCO (Prud'homme, Fages, and Lorca 2014) constraint solver for testing the satisfiability of the learnt constraint network and MINISAT SAT solver (Eén 2006) for the generation of informative queries. For MSS enumeration, DCA leverages the MARCO (Liffiton and Malik 2013) algorithm which proved to be faster than DAA (Bailey and Stuckey 2005) in our context.

4.1 Experimental Design

In order to respond to the three raised RQs, we performed two experiments on two distinct benchmarks. The first benchmark is formed of simple disjunctive constraints composed of random constraints, one global constraint and ultrametric constraints. The second benchmark is extracted from the precondition inference application.

Disjunctive Constraint Benchmark (DCB). This benchmark is composed of three different constraint families, namely random, domain and ultrametric constraints. We selected these constraints in order to ensure a sufficient level of diversity in the benchmark.

- *Random.* As a baseline, we randomly generated disjunctive constraints named $\mathtt{RAND}_{n,d}$ with $n \in \{2,3,4\}$ being the number of variables and $d \in \{2,3,4\}$ the maximum disjunction size considered. These disjunctive constraints are composed of the atomic constraints $X_i = X_j$, $X_i \neq X_j$, $X_i > X_j$, and $X_i \leq X_j$. For example, $\mathtt{RAND}_{3,2}$ aims to infer the constraint $(X_1 \neq X_2 \vee X_0 = X_2) \wedge (X_1 \leq X_2 \vee X_0 > X_2) \wedge (X_1 > X_2 \vee X_0 \geq X_2)$. For each configuration, we randomly generate one constraint network.
- *Domain Constraint.* Global constraints often capture a combination of disjunctive constraints. To complement our benchmark, we selected the $\mathtt{DOMAIN}(X, [X_1, .., X_n])$ global constraint, noted $\mathtt{DOM}_n$, to explore the acquisition of disjunctive constraints over finite domains. This constraint, which is formally defined in the catalogue of global constraint[4], is true iff $\forall i \in 1..n, X = i$ iff $X_i = 1$. Note that $X \in 1..n$ and $X_i \in 0..1$. This is one of the simplest global constraints that capture disjunctive relations.
- *Ultrametric Constraints.* The ultrametric constraint $\mathtt{UM}_3(X, Y, Z)$ (Moore and Prosser 2008) stands for

  $$X > Y = Z \vee Y > X = Z \vee Z > X = Y \vee X = Y = Z$$

  having 3 variables and 4 disjuncts. In our benchmark, we generalized the ultrametric constraint to a family of constraints as follows $\mathtt{UM}_{k+1} = \bigwedge_{V \in 2^X \wedge |V|=k} \mathtt{UM}_k(V)$, where $k \geq 3$ is the number of variables. So, $\mathtt{UM}_k$ contains k variables and an ever-growing number of constraints with 4 disjuncts.

Precondition Inference Benchmark (PIB). To evaluate DCA on precondition acquisition, our dataset considers 60 real C functions for which preconditions have to be inferred. It includes all functions available in the public repository[5] associated to the (Menguy et al. 2022) publication: all functions from `string.h`, all functions from (Seghir and Kroening 2013; Sankaranarayanan et al. 2008) some functions from the DSA benchmark (https://tinyurl.com/tvzzpvmm), Frama-C WP test suite (https://tinyurl.com/ycxdbjf3), Siemens suite (Hutchins et al. 1994), the book Science of Programming (Gries 2012). We also consider some functions from the mbedtls cryptographic library. Overall, our benchmark PIB extends the one from (Menguy et al. 2022) with functions having highly disjunctive preconditions. In particular, PIB functions have in between 1 and 8 inputs and 39 over 60 have disjunctive preconditions with clauses of size between 2 and 7. Note that 21 functions have conjunctive-only preconditions. We choose to keep them in the evaluation as users do not know at first sight if preconditions are disjunctive or not. It also allows evaluating DCA over fully conjunctive problems to evaluate its overhead.

To answer **RQ3**, we consider different bias configurations presented in Table 1. The *Min bias* configuration considers biases with only constraints and variables requested to express the inferred preconditions The *Avg bias* configuration considers biases with only requested constraints but applied to all combinations of variables. Finally, the *Max bias* configuration considers all possible constraints from a given input language similar to the one presented in (Menguy et al. 2022), applied to all combinations of variables. Thus, given a function under analysis, its minimal bias is a subset of its average bias which is itself a subset of the maximal bias.

	min size	max size	mean size
Min bias	2	32	7.5
Avg bias	2	62	11
Max bias	2	76	18.5

Table 1: Statisitics of the biases (in terms of atomic constraints) used for precondition inference

Setup We ran our experiments with different time budgets (from 1s to 1h, excluding bias generation time). Experiments are done on a machine with 6 Intel Xeon E-2176M CPUs and 32 GB of RAM.

4.2 Experimental Results

We now present results of DCA over our two datasets.

RQ1. We compare DCA and CONACQ over the DCB dataset. Results are summarized in Table 2. It presents for CONACQ and DCA the size of the bias considered ($|B|$), the number of queries generated ($|E|$) and the convergence time. Moreover, the maximum disjunction size for each problem is stated in the $Disj$ column. Observe that CONACQ biases' size explodes while DCA biases do not. Indeed, CONACQ cannot infer disjunctions. Thus, all combinations of disjunctions must be added to the bias, hence increasing its size drastically. On the other hand, DCA naturally infers disjunctions, and its bias only contains atomic constraints. Experiments show that DCA handles more complex cases than CONACQ. Especially, CONACQ cannot handle the $\mathtt{RAND}_{4,3}$, and $\mathtt{RAND}_{4,4}$ examples in 1h while DCA infer the correct concepts in 36s and 37s, respectively. On the domain constraints, CONACQ only handles the simplest case

[4] sofdem.github.io/gccat/gccat/Cdomain_constraint.html
[5] https://zenodo.org/record/6513522#.ZBGqLnWYWV4

$\texttt{DOM}_3$ while DCA handles up to $\texttt{DOM}_9$. Over ultra-metric constraints, DCA handles up to 5 variables while CONACQ can only handle the three variables case. Moreover, even in this case, DCA is 100 times faster, taking 0.5s against 50s for CONACQ. Still, on the conjunctive-only problems ($\texttt{RAND}_{X,1}$), CONACQ is faster than DCA. Especially, over $\texttt{RAND}_{4,1}$, CONACQ infers the correct concept in 1s against 38s for DCA. However, CONACQ performances are highly impacted on disjunctive problems (e.g., moving from $\texttt{RAND}_{4,1}$ to $\texttt{RAND}_{4,2}$, CONACQ is 286× slower), while it has no impact on DCA. Moreover, we observe that even giving the exact disjunction size needed (CONACQ$_{Omniscient}$ column), CONACQ cannot keep pace with DCA.

Conclusion: DCA *is faster and infers more complex constraints than* CONACQ *even if we give it exactly the needed disjunction sizes. As expected,* DCA *is not impacted by the disjunctive behaviors of the target problem, unlike* CONACQ.

		CONACQ			CONACQ$_{omiscient}$			DCA		
	$Disj$	$\|B\|$	$\|E\|$	Time	$\|B\|$	$\|E\|$	Time	$\|B\|$	$\|E\|$	Time
$\texttt{RAND}_{2,1}$	1	6	3	**0.2s**	6	3	0.3s	6	3	0.2s
$\texttt{RAND}_{2,2}$	2	18	3	0.4s	18	3	0.3s	6	3	**0.2s**
$\texttt{RAND}_{2,3}$	3	26	3	0.4s	14	3	0.2s	6	3	**0.2s**
$\texttt{RAND}_{2,4}$	4	26	3	0.4s	6	3	0.2s	6	3	**0.2s**
$\texttt{RAND}_{3,1}$	1	18	7	**0.5s**	18	6	0.3s	18	13	0.9s
$\texttt{RAND}_{3,2}$	2	162	13	5s	162	13	3.7s	18	13	**0.9s**
$\texttt{RAND}_{3,3}$	3	834	13	154s	690	13	43s	18	13	**1s**
$\texttt{RAND}_{3,4}$	4	2850	13	817s	2034	13	140s	18	13	**0.9s**
$\texttt{RAND}_{4,1}$	1	36	14	**1s**	36	15	1s	36	75	38s
$\texttt{RAND}_{4,2}$	2	648	54	286s	648	37	47s	36	75	**39s**
$\texttt{RAND}_{4,3}$	3	7176	-	TO	6564	-	TO	36	75	**36s**
$\texttt{RAND}_{4,4}$	4	56136	-	TO	48996	-	TO	36	75	**37s**
$\texttt{DOM}_3$	3	834	24	297s	690	24	217s	18	24	**0.6s**
$\texttt{DOM}_4$	4	9968	-	TO	7944	-	TO	24	64	**1.2s**
$\texttt{DOM}_5$	5	122026	-	TO	96126	-	TO	30	160	**3.3s**
$\texttt{DOM}_6$	6	-[6]	-	TO	-[6]	-	TO	36	384	**9.7s**
$\texttt{DOM}_7$	7	-	-	ME	-	-	ME	42	896	**44s**
$\texttt{DOM}_8$	8	-	-	ME	-	-	ME	48	2048	**233s**
$\texttt{DOM}_9$	9	-	-	ME	-	-	ME	54	4608	**1690s**
$\texttt{DOM}_{10}$	10	-	-	ME	-	-	ME	60	-	TO
$\texttt{UM}_3$	4	472	13	50s	252	13	13s	12	13	**0.5s**
$\texttt{UM}_4$	4	9968	-	TO	7944	-	TO	24	75	**3s**
$\texttt{UM}_5$	4	87440	-	TO	77560	-	TO	40	541	**200s**
$\texttt{UM}_6$	4	-[6]	-	TO	-[6]	-	TO	60	-	TO

$Disj$ is the size of the disjunctions in the target CN; $|B|$ is the size of the bias (note that CONACQ's bias includes disjunctions of size up to $Disj$, while for DCA, it contains only atomic constraints); $|E|$ is the number of queries generated (note: for DCA, $|E|$ also equals to the number of B's MSS); Time is the time taken to converge to a unique solution excluding bias generation time (if TO, it means that execution timeouts). Finally, ME stands for memory exhaustion. The CONACQ$_{Omniscient}$ column presents CONACQ results when the user knows exactly the disjunctions size present in the target network.

Table 2: DCA and CONACQ over synthetic dataset

RQ2. First, we compare DCA to the original PRECA proposal, which decides the maximum size of disjunctions to include in the bias with a simple ad-hoc heuristic. Results are summarized in Table 3. Over the maximal bias, which considers all constraints from (Menguy et al. 2022), DCA infers 51/60 weakest preconditions in 1h against 45/60 for PRECA. This is due to the fact, that PRECA does not include disjunctions big enough to infer the correct concept. Still, even if we give exactly the disjunction sizes present in the target precondition (PRECA-$Omiscient$), PRECA is not able to infer the preconditions in less than 1h. In a more realistic scenario, where the user enforces a specific disjunction size threshold (PRECA-$|disj| \leq n$), we observe that considering disjunctions bigger than three is counterproductive. Especially, for disjunctions of size 3 to 10, the number of inferred preconditions decreases from 44/60 to 35/60 preconditions in one hour. Overall, DCA is, on average, 2.5× faster than PRECA on functions where both terminate. Especially, it infers in 5mins more preconditions than PRECA in 1h even in the $Omiscient$ scenario. We also evaluate how PRECA behaves depending on the size of disjunctions in the target precondition. Results in Table 4 show that PRECA is as good as DCA over conjunctive-only preconditions (i.e., "No disj" column). However, on disjunctive preconditions, PRECA infers less weakest preconditions and is slower than DCA. For disjunctions of size 7, DCA infers 51/60 preconditions against 35/60 for PRECA, and is on average 7× faster on examples where both terminate. Indeed, including disjunctions in PRECA's bias slows PRECA drastically. On the other side, DCA can infer disjunctions of arbitrary size. In practice, over the PIB dataset, DCA successfully infers preconditions with disjunctions of size up to 7.

Conclusion: DCA *enables to infer more weakest preconditions than* PRECA *while requiring no previous knowledge on disjunction size. Especially, it infers disjunctions of size 7 where* PRECA *can only infer efficiently disjunctions of size up to 3. Moreover,* DCA *is as efficient as* PRECA *over fully conjunctive preconditions.*

RQ3. Finally, we compared PRECA and DCA over multiple biases, described in Table 1. Especially, the minimal bias includes only constraints useful for the inference, applied to exactly the good variables. Thus, the only task is to decide how to combine them with disjunctions and conjunctions. Over the minimal bias, we observe the DCA can handle three additional functions compared to the average and maximal bias. Moreover, we observe that, in 1h, PRECA infers fewer preconditions (48/60) than DCA even over the maximal bias (51/60). We also observe a speed-up for DCA over the average bias, which includes only the needed constraints applied to all variables. Especially, compared to the maximal bias, DCA infers in 1s, 5s and 5 mins, respectively 7, 3, and 2 additional preconditions.

Conclusion: Giving DCA *problem knowledge by feeding it with relevant only constraints is beneficial. In such a case, it infers more preconditions faster. Moreover, even without help, i.e., over the maximal bias,* DCA *outperforms* PRECA *over the minimal bias, i.e., with full problem knowledge.*

5 Discussion

In the following, we discuss the needed building blocks to implement DCA and introduce the notion of background knowledge, usual in CA but which was shown to be useless in practice for DCA. Then we present its limitations.

[6]Here even the bias generation takes more than 1 hour

	Min bias				Avg bias				Max bias			
	1s	5s	5 mins	1h	1s	5s	5 mins	1h	1s	5s	5 mins	1h
PRECA	34/60	45/60	48/60	48/60	32/60	44/60	46/60	46/60	24/60	36/60	44/60	45/60
↳ No disj	21/60	21/60	21/60	21/60	21/60	21/60	21/60	21/60	20/60	21/60	21/60	21/60
↳ $\|disj\| \leq 2$	38/60	43/60	44/60	44/60	35/60	42/60	44/60	44/60	21/60	38/60	44/60	44/60
↳ $\|disj\| \leq 3$	30/60	44/60	48/60	48/60	26/60	43/60	46/60	46/60	18/60	31/60	42/60	44/60
↳ $\|disj\| \leq 4$	30/60	43/60	48/60	48/60	26/60	42/60	45/60	46/60	18/60	29/60	35/60	40/60
↳ $\|disj\| \leq 7$	30/60	43/60	48/60	48/60	27/60	42/60	45/60	45/60	18/60	28/60	35/60	35/60
↳ $\|disj\| \leq 10$	30/60	43/60	48/60	48/60	27/60	42/60	45/60	45/60	17/60	27/60	35/60	35/60
↳ $Omniscient$	38/60	45/60	48/60	48/60	34/60	44/60	46/60	46/60	26/60	40/60	43/60	45/60
DCA	40/60	45/60	51/60	54/60	38/60	45/60	49/60	51/60	31/60	42/60	47/60	51/60

The gray lines represent the default configurations of the evaluated tools. For PRECA, it considers the version presented in (Menguy et al. 2022) with its default heuristic to decide on disjunction size to include in the bias. The PRECA-$Omniscient$ line presents PRECA results when the user knows exactly the disjunctions size present in the target precondition.

Table 3: Number of weakest precondition inferred by PRECA and DCA depending on the time budget

	No disj		$\|disj\| \leq 2$		$\|disj\| \leq 3$		$\|disj\| \leq 4$		$\|disj\| \leq 7$		$\|disj\| \leq 10$	
	#WP	Time	#WP	Time	#WP	Time	#WP	Time	#WP	Time	#WP	Time
PRECA-$disj \leq disj$	21/21	0.6s	44/48	7.8s	44/52	38.6s	40/53	166s	35/60	6s	35/60	7s
DCA	21/21	0.8s	44/48	2.4s	48/53	52s (1.2s)	48/53	51s (1s)	51/60	102s (0.8s)	51/60	102s (0.8s)

#WP represents the number of weakest preconditions inferred, and Time is the mean convergence time over **successful** (i.e., not reaching the timeout) acquisitions, excluding bias generation (in parenthesis is the mean time where both PRECA and DCA succeed, not printed if the same). Here, for each disjunction size, PRECA is launched with the same disjunction size. Thus, for the No disj, PRECA considers only conjunctions of atomic constraints. For the $|disj| \leq 2$, it includes disjunctions of size up to 2 and so forth. On the other hand, DCA needs no disjunction size information.

Table 4: Comparison of DCA and PRECA depending on the size of dsjunctions in the target preconditions (Maximal bias; TO=1h)

Implementing DCA. DCA is a simple and very general approach for constraint acquisition. To implement the framework, it only needs a model generation procedure for boolean formulas and for the underlying theory T. Especially, unlike CONACQ, DCA does not rely on costly pseudo boolean solvers. In our experiments, we respectively rely on the MiniSat and the Choco solvers to find solutions to the boolean formula representing the search space and to generate queries. The solver for theory T is only applied to MSSes of the bias, which are conjunctive only.

Background knowledge in DCA. DCA extends classical CA-based approaches to handle disjunctive problems. In Section 3 we show how each CA core concept (bias, informative queries) translates, except for the background knowledge (Bessiere et al. 2017). Sill, DCA can also include a background knowledge, containing rules over constraints to speed up inference. In our context, such background knowledge is composed of minimal unsatisfiable subsets (MUS) of the bias and enables speeding up MSS enumeration methods like DAA or MARCO. However, our experiments (not reported here) show no impact on acquisition time.

Limitations. While DCA shows overall good theoretical and practical properties, it also comes with a few limitations. First, it returns a CN that is hard to understand for a human user. While it may not be crucial in some contexts – e.g., when applied to automated program analysis – it may be a burden for human users. Still, adding a post process may be enough to simplify the result. Especially, on average, over the PIB dataset (*Max bias*), our post process reduces the size of DCA results (in terms of the number of atomic constraints) by a factor of 40, returning a CN with the same size as our ground-truth (mean size before and after simplifications respectively equal 111 and 2.73). Second, the number of queries can be hard to handle for a human user. However, we recently showed that in some applicative scenarios of interest, CA can be combined with an automated oracle, strongly alleviating the limitation of the number of queries. Third, just as CONACQ-like methods, DCA cannot infer global constraints, but only their specialization over a fixed set of variables. Finally, DCA relies on *membership queries* only. Still, other kinds of queries have been proposed, like partial queries (Bessiere et al. 2013). How to extend DCA to such queries is an interesting direction.

6 Related Work

CA has been extensively covered in the literature. Two primary methodologies have emerged, namely passive learning and active learning.

Passive CA. Passive learning involves systems acquiring constraints from a provided set of examples. Some approaches use positive and negative examples. In particular, the Conacq.1 algorithm (Bessiere et al. 2007) relies on version space learning (Mitchell 1977; Mitchell 1982) to infer a constraint network accepting all positive examples and rejects all negative examples. The Conacq.1 algorithm is general-purpose and does not require any specific problem structure. The Lallouet et al. (Lallouet et al. 2010) proposal also handles positive and negative examples but leverages inductive logic programming to infer the target concept. A limitation of this approach is that the user must provide the entire problem structure, unlike CONACQ.1. Other approaches rely on positive examples only (Beldiceanu and Simonis 2016; Kumar et al. 2019). Among them, a successful method is ModelSeeker (Beldiceanu and Simonis 2016). It uses a global constraints catalog to build the version space and identify global constraints satisfied by particular subsets of variables in all examples, such as rows or columns. However, ModelSeeker can only find constraints from the catalog that hold on the specific structures it can recognize. Orthogonal approaches have also been proposed to perform error-resilient acquisition (Prestwich 2020; Prestwich 2021). Unlike previous approaches, trying to classify all examples, such methods consider that some examples are errors and eliminate them. DCA distinguishes from all these previous approaches, performing active constraint acquisition, which enables to enjoy better guarantees and frees the user from the burden of giving examples himself.

Active CA. Active CA is a specific type of query-directed learning called *exact learning* (Angluin 1988; Angluin 2004). In the formalism introduced by Angluin (1987), there are two types of queries: a membership query, which asks the user to classify a given example as positive or negative, and an *equivalence* query, which asks the user to determine whether the given concept is equivalent to the target concept. However, CA restricts itself to asking membership queries only, as answering equivalence queries is too difficult for the user which is assumed to be not skilled enough to express the target networks themselves (Bessiere et al. 2017). One early example of active CA is the *Matchmaker* agent (Freuder and Wallace 1999). The matchmaker suggests examples to the user as potential solutions to the problem. A negative response from the user comes with a "correction" that indicates why the suggestion (i.e., the example) fails. The correction consists of one or more of the constraints that are violated. However, this approach requires sufficient expertise from the user to express the violated constraints. CONACQ.2 is an active learning approach for CA, which has been presented in the literature (Bessiere et al. 2007; Bessiere et al. 2017). In CONACQ.2, only membership queries are presented to the user, meaning that the user is only required to classify examples as solutions or non-solutions. This makes it less demanding in terms of expertise compared to other active learning approaches that require users to answer equivalence queries. This last approach is closer to ours, and we extensively compared DCA to it in this article. Especially, DCA also performs active learning by asking only membership queries. However, unlike CONACQ.2, which can infer conjunctions of constraints only, DCA can infer conjunctions of disjunctions.

Disjunction in CA. Traditional CA has limited expressivity in handling disjunctions, which can be a part of the concept description. Existing CA approaches can only learn disjunctions that are part of the constraint language forming the learning bias. Beldiceanu and Simonis (2012) proposed ModelSeeker, a system that can capture possible disjunctions by learning the conjunction of global constraints from positive examples. These global constraints can encapsulate the disjunctions, like *Domain* and *Disjunctive*. Approaches based on version space learning, such as CONACQ (Bessiere et al. 2017), QUACQ (Bessiere et al. 2013), require a constraint to capture the given disjunction, leading to an increase in the bias size beyond polynomial. This limitation poses a challenge in effectively handling disjunctions within the concept description. To address this issue, PRECA is proposed as an extension of CONACQ for "precondition inference", which allows disjunctions of limited size in the bias to ensure its polynomial size. Our approach DCA is able to infer automatically the needed disjunctions removing the burden to the user to give the needed disjunctions.

7 Conclusion

We propose DCA the first principled approach for active disjunctive constraint acquisition. It generalizes classical CA, which is restricted to the acquisition of conjunctive constraints or disjunctive constraints provided by the user. To do so, DCA relies on maximal satisfiable subsets enumeration to infer the correct concept with good theoretical properties. Especially if the input set of constraints is expressive enough to represent the target concept, then DCA can surely infer it. We evaluated DCA on two benchmarks composed of random, ultrametric, domain constraints and one real-world application scenario, precondition acquisition. Experiments show that DCA is able to handle both cases efficiently. Notably, it outperforms the state-of-the-art CONACQ and PRECA CA-based frameworks.

Acknowledgments

This work has received support from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952215 (TAILOR project) and No 101076911 (AI4CCAM Project), from ICO, the Occitania Cybersecurity Institute – which is funded by the Occitania Region in France, and from the National Research Agency under France 2030 with reference "ANR-22-PECY-0007". We thank Joao Marques-Silva who provided insight and expertise that greatly assisted the work. We would also like to show our gratitude to the anonymous reviewers for their insightful suggestions and careful reading.

References

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75(2).

Angluin, D. 1988. Queries and concept learning. *Machine learning* 2:319–342.

Angluin, D. 2004. Queries revisited. *Theoretical Computer Science* 313(2):175–194.

Bailey, J., and Stuckey, P. J. 2005. Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *Practical Aspects of Declarative Languages: 7th International Symposium, PADL 2005*, 174–186. Springer.

Belaid, M.-B.; Belmecheri, N.; Gotlieb, A.; Lazaar, N.; and Spieker, H. 2022. Geqca: Generic qualitative constraint acquisition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 3690–3697.

Beldiceanu, N., and Simonis, H. 2012. A model seeker: Extracting global constraint models from positive examples. In *International Conference on Principles and Practice of Constraint Programming (CP 2012)*. Springer.

Beldiceanu, N., and Simonis, H. 2016. ModelSeeker: Extracting Global Constraint Models from Positive Examples. In *Data Mining and Constraint Programming*. Springer.

Bessiere, C.; Coletta, R.; O'Sullivan, B.; and Paulin, M. 2007. Query-driven constraint acquisition. In *IJCAI*.

Bessiere, C.; Coletta, R.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Narodytska, N.; Quimper, C.-G.; and Walsh, T. 2013. Constraint acquisition via partial queries. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

Bessiere, C.; Koriche, F.; Lazaar, N.; and O'Sullivan, B. 2017. Constraint acquisition. *Artificial Intelligence*.

Bessiere, C.; Carbonnel, C.; Dries, A.; Hebrard, E.; Katsirelos, G.; Lazaar, N.; Narodytska, N.; Quimper, C.; Stergiou, K.; Tsouros, D. C.; and Walsh, T. 2020. Partial queries for constraint acquisition. *CoRR* abs/2003.06649.

Eén, N. 2006. The minisat page. http://minisat.se/. Accessed: 2021-12-01.

Freuder, E. C., and Wallace, R. J. 1999. Suggestion strategies for constraint-based matchmaker agents. In *Principles and Practice of Constraint Programming*. Springer.

Gent, I. P.; Prosser, P.; Smith, B. M.; and Wei, W. 2003. Supertree construction with constraint programming. In Rossi, F., ed., *Principles and Practice of Constraint Programming*. Springer.

Gleeson, J., and Ryan, J. 1990. Identifying minimally infeasible subsystems of inequalities. *ORSA Journal on Computing* 2(1):61–63.

Gries, D. 2012. *The science of programming*. Springer Science & Business Media.

Hutchins, M.; Foster, H.; Goradia, T.; and Ostrand, T. 1994. Experiments on the effectiveness of dataflow-and controlflow-based test adequacy criteria. In *Proceedings of 16th International conference on Software engineering*. IEEE.

Kumar, M.; Teso, S.; De Causmaecker, P.; and De Raedt, L. 2019. Automating personnel rostering by learning constraints using tensors. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE.

Lallouet, A.; Lopez, M.; Martin, L.; and Vrain, C. 2010. On learning constraint problems. In *22nd IEEE International Conference on Tools with Artificial Intelligence*. IEEE.

Liffiton, M. H., and Malik, A. 2013. Enumerating infeasibility: Finding multiple muses quickly. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*. Springer.

Liffiton, M. H., and Sakallah, K. A. 2005. On finding all minimally unsatisfiable subformulas. In *Theory and Applications of Satisfiability Testing*. Springer.

Menguy, G.; Bardin, S.; Lazaar, N.; and Gotlieb, A. 2022. Automated program analysis: Revisiting precondition inference through constraint acquisition. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Mitchell, T. M. 1977. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 1*.

Mitchell, T. M. 1982. Generalization as search. *Artificial intelligence* 18(2):203–226.

Moore, N. C. A., and Prosser, P. 2008. The ultrametric constraint and its application to phylogenetics. *J. Artif. Int. Res.* 32(1):901–938.

Paulin, M.; Bessiere, C.; and Sallantin, J. 2008. Automatic design of robot behaviors through constraint network acquisition. In *2008 20th IEEE International Conference on Tools with Artificial Intelligence*.

Prestwich, S. D. 2020. Robust constraint acquisition by sequential analysis. *Frontiers in Artificial Intelligence and Applications* 325:355–362.

Prestwich, S. 2021. Unsupervised constraint acquisition. In *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 256–262. IEEE.

Prud'homme, C.; Fages, J.-G.; and Lorca, X. 2014. Choco documentation. *TASC* 6241.

Rossi, F.; Van Beek, P.; and Walsh, T. 2006. *Handbook of constraint programming*. Elsevier.

Régin, J.-C. 2010. *Global Constraints: A Survey*. 63–134.

Sankaranarayanan, S.; Chaudhuri, S.; Ivančić, F.; and Gupta, A. 2008. Dynamic inference of likely data preconditions over predicates by tree learning. In *Proceedings of the 2008 international symposium on Software testing and analysis*. ACM.

Seghir, M. N., and Kroening, D. 2013. Counterexample-guided precondition inference. In *European Symposium on Programming*. Springer.

Shchekotykhin, K. M., and Friedrich, G. 2009. Argumentation based constraint acquisition. In *ICDM 2009, The Ninth IEEE International Conference on Data Mining*. IEEE Computer Society.

Tsouros, D. C.; Stergiou, K.; and Bessiere, C. 2020. Omissions in constraint acquisition. In *International Conference on Principles and Practice of Constraint Programming (CP)*. Springer.

Van Loon, J. 1981. Irreducibly inconsistent systems of linear inequalities. *European Journal of Operational Research* 8(3):283–288.