Learning and Using Structures for Constraint Acquisition

Abderrazak Daoudi

Collège Doctoral du Languedoc-Roussillon

THÈSE

Pour obtenir le grade de

Docteur

Délivré par l'**Université de Montpellier**

Préparée au sein de l'école doctorale **I2S**
Et de l'unité de recherche **LIRMM**

Spécialité : **Informatique**

Présentée par **Abderrazak DAOUDI**

Learning and Using Structures for Constraint Acquisition

Soutenue le 10 Mai 2016 devant le jury composé de

M. Mohammed BENKHALIFA	Professeur	U. Mohammed V, Maroc	Président
M. Christian BESSIERE	DR CNRS	U. de Montepellier, France	Directeur de thèse
M. El Houssine BOUYAKHF	Professeur	U. Mohammed V, Maroc	Directeur de thèse
M. Redouane EZZAHIR	Professeur	U. Ibn Zohr, Maroc	Rapporteur
Mme Souhila KACI	Professeur	U. de Montepellier, France	Examinateur
Mme Élise VAREILLES	HDR	École des Mines d'Albi, France	Rapporteur

UNIVERSITÉ MONTPELLIER

Université Mohammed V
Faculté des Sciences de Rabat

Collège doctoral du languedoc-roussillon
Université de Montpellier

N° d'ordre: .....

THÈSE DE DOCTORAT

présentée par

Abderrazak DAOUDI

Discipline : Sciences de l'ingénieur

Spécialité : Informatique

LEARNING AND USING STRUCTURES FOR CONSTRAINT ACQUISITION

Soutenue le 10 Mai 2016 devant le jury composé de :

M. Mohammed BENKHALIFA	Professeur	U. Mohammed V, Maroc	Président
M. Christian BESSIERE	DR CNRS	U. de Montepellier, France	Directeur de thèse
M. El Houssine BOUYAKHF	Professeur	U. Mohammed V, Maroc	Directeur de thèse
M. Redouane EZZAHIR	Professeur	U. Ibn Zohr, Maroc	Rapporteur
Mme Souhila KACI	Professeur	U. de Montepellier, France	Examinateur
Mme Élise VAREILLES	HDR	École des Mines d'Albi, France	Rapporteur

To my parents
To my sisters and brothers

Contents

Acknowledgments

The research work presented in this thesis has been conducted in the Laboratory of Informatics Applied Mathematics, Artificial Intelligence and Pattern Recognition (LIMIARF), Faculty of Sciences, University Mohammed V of Rabat, Morocco, and the Laboratory of Informatics, Robotics and Microelectronics of Montpellier (LIRMM), University of Montpellier, France.

This thesis has been done in collaboration between University Mohammed V of Rabat, Morocco, and University of Montpellier, France, under the financial support of the scholarship Averroès funded by the European Commission.

Firstly, I would like to express my deepest gratitude to my supervisors Professor El Houssine Bouyakhf and Professor Christian Bessiere. I will always be grateful for all the support, guidance, encouragement they have given me during these three years. It was a real pleasure for me to work with them. They are so highly motivated, enthusiastic and passionate about science. Their guidance, suggestions and relevant feedback helped me a lot to progress and to continue my research in the right direction.

I would like to express my sincere gratitude to the jury for the interest according to my work. I gratefully acknowledge Mr. Mohammed Benkhalifa, Professor at Faculty of sciences, University Mohammed V of Rabat, Morocco, for accepting to chair the jury of my dissertation. I would like to thank Ms. Souhila Kaci, Professor at University of Montpellier, France, for accepting to exanimate my thesis. I am most grateful to my reviewers Ms. Elise Vareilles, Associate Professor (HDR) at Ecole des Mines d'Albi, France, and Mr. Redouane Ezzahir, Associate Professor (PH) at University of Ibn Zohr, Morocco. I thank them all for their comments and fruitful discussions.

The published research works of this thesis have been done in collaboration with so highly motivated, smart and enthusiastic coauthors. I want to thank them for

their hard work and teamwork. I cannot thank my coauthors without giving my special gratitude to Younes Mechqrane, Nadjib Lazaar and Remi Coletta.

My thanks go also to the staffs of the LIRMM and LIMIARF Laboratories. I would like to thank all past and present colleagues at the LIRMM and LIMIARF for the joyful and pleasant working environment. I especially acknowledge the members of the Coconut team: Gilles, Joel, Phillipe, Natasa, Gaelle, Olivier, Nicolas, Mehdi and Robin, and the members of IA team of LIMIARF: Imade, Saida, Yousra, Amine, Mehdi, Zakarya and Ghizlan.

During this thesis, I have been blessed with a cheerful group of friends. I would like to particularly thank Amine, Nawfal, Adil, Mohamed, Abdelhak, Youssef K., Youssef M., Mouad, Lhassan, Salim, Jaouad, Younes and Ahmed.

Last but not the least; I would like to thank my family: my parents, my sisters and brothers for their unconditional support, encouragement and endless love.

Dr. Abderrazak Daoudi
Montpellier, May 23, 2016

Abstract

Constraint Programming is a general framework used to model and to solve complex combinatorial problems. However, modeling a problem as a constraint network requires significant expertise in the field. Such level of expertise is a bottleneck to the broader uptake of the constraint technology. To alleviate this issue, several constraint acquisition systems have been proposed to assist the non-expert user in the modeling task. Nevertheless, in these systems the user is only asked to answer very basic questions. The drawback is that when no background knowledge is provided, the user may need to answer a large number of such questions to learn all the constraints. In this thesis, we show that using the structure of the problem under consideration may improve the acquisition process a lot. To this aim, we propose several techniques. Firstly, we introduce the concept of generalization query based on an aggregation of variables into types. Secondly, to deal with generalization queries, we propose a constraint generalization algorithm, named GenAcq, together with several strategies. Thirdly, to make the build of generalization queries totally independent of the user, we propose the algorithm Mine&Ask, which is able to learn the structure, during the constraint acquisition process, and to use the learned structure to generate generalization queries. Fourthly, toward a generic concept of query, we introduce the recommendation query based on the link prediction on the current constraint graph. Fifthly, we propose a constraint recommender algorithm, called Predict&Ask, that asks recommendation queries, each time the structure of the current graph has been modified. Finally, we incorporate all these new generic techniques into QuAcq algorithm leading to three boosted versions, g-QuAcq, m-QuAcq, and p-QuAcq. To evaluate all these techniques, we have made experiments on several benchmarks. The results show that the extended versions improve drastically the basic QuAcq.

Keywords: *Artificial Intelligence (AI), Constraint Programming (CP), Constraint Satisfaction Problems (CSP), Constraint Acquisition, Learning, Graph Community Detection, Recommendation Systems.*

Résumé

La Programmation par Contraintes est un cadre général utilisé pour modéliser et résoudre des problèmes combinatoires complexes. Cependant, la modélisation d'un problème sous forme d'un réseau de contraintes nécessite une bonne expertise dans le domaine. Ce niveau d'expertise est un obstacle majeur pour une large diffusion de la programmation par contraintes. Pour remédier à ce problème, plusieurs systèmes d'acquisition de contraintes ont été proposés pour aider l'utilisateur dans la tâche de modélisation. Dans ces systèmes, l'utilisateur ne répond qu'à des questions très simples. L'inconvénient est que lorsqu'aucune connaissance de base n'est fournie, l'utilisateur peut avoir besoin de répondre à un grand nombre de questions pour apprendre toutes les contraintes. Dans cette thèse, nous montrons que l'utilisation de la structure du problème peut améliorer considérablement le processus d'acquisition. Pour ce faire, nous proposons plusieurs techniques. Tout d'abord, nous introduisons le concept de requête de généralisation basée sur une agrégation de variables sous forme de types. Deuxièmement, pour faire face aux requêtes de généralisation, nous proposons un algorithme de généralisation de contraintes, nommé GENACQ, ainsi que plusieurs stratégies. Troisièmement, pour rendre la construction de requêtes de généralisation totalement indépendante de l'utilisateur, nous proposons l'algorithme MINE&ASK, qui est en mesure d'apprendre la structure au cours du processus d'acquisition, et d'utiliser la structure apprise pour générer des requêtes de généralisation. Quatrièmement, pour aller vers un concept générique de requête, nous introduisons la requête de recommandation basée sur la prédiction de liens dans le graphe de contraintes apprises jusqu'à présent. Cinquièmement, nous proposons un algorithme de recommandation de contraintes, appelé PREDICT&ASK, qui demande à l'utilisateur de classifier des requêtes de recommandation chaque fois que la structure du graphe courant a été modifiée. Enfin, nous intégrons toutes ces nouvelles techniques dans l'algorithme QUACQ, menant à trois nouvelles versions, à savoir G-QUACQ, M-QUACQ, et P-QUACQ. Pour évaluer toutes ces techniques, nous avons fait des expérimentations sur plusieurs jeux de données. Les résultats montrent que les versions étendues améliorent considérablement le QUACQ de base.

Mots clefs : *Intelligence Artificielle, Programmation par contraintes, Problèmes de satisfaction de contraintes, Acquisition de contraintes, Détection de communautés, Systèmes de recommandation.*

CHAPTER

1 Introduction

Constraint Programming (CP) is a very active research area within Artificial Intelligence (AI), and its importance has increased dramatically within the last years. CP is a highly successful technology for solving a wide range of combinatorial problems, such as scheduling, resource allocation, and design. Nowadays, a number of companies, like ILOG and Dash Optimization, sell constraint programming toolkits, which are used by companies as diverse as Amazon.com, British Airways, Cisco, Ford, HP, SNCF, and Volvo.

Constraint programming is a declarative style of modeling combinatorial problems. The user identifies the decision variables, their possible domain of values, and specifies constraints over the allowed values. For instance, the constraint $X_1+X_2 \leq X_3$ specifies that any combination of values for variables X_1, X_2 and X_3 has to be such that the sum of X_1 and X_2 less than or equals X_3. Sophisticated AI search techniques like constraint propagation, which allow to prune irrelevant parts of the search tree, and chronological backtracking can then be used to find solutions.

Unfortunately, modeling a combinatorial problem as a constraint network remains limited to specialists in the field. However, it is well known that modeling a combinatorial problem in the constraint formalism requires significant expertise in constraint programming [Freuder, 1999; Puget, 2004]. Such a level of knowledge prevents non-expert users 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 novices.

One answer to the task of modeling is constraint acquisition, which is an active research field that lies in the conjunction of two fields, namely Constraint Programming and Machine Learning. The idea behind constraint acquisition is to assist

a non-expert user in modeling her problem as a constraint network. Several constraint acquisition systems have been introduced in the last decade [Bessiere et al., 2005, 2007; Lallouet et al., 2010; Beldiceanu and Simonis, 2012; Bessiere et al., 2013]. Unfortunately, Most of these constraint acquisition systems interact with the user by asking her to classify an example as positive or negative. Such queries do not use the structure of the problem and can thus lead the user to answer a large number of queries. Hence, it can be hard to put constraint acquisition in practice.

The propose of this thesis is to provide new approaches to constraint acquisition in order to make it more efficient in practice. However, we show that learning and using the structure of the problem under consideration may accelerate the process of learning a lot. To this end, we introduce two new concept of queries that use the structure of the problem. The first one, called *generalization query*, is an opportunistic query based on the aggregation of variables into types. Generalization query asks the user whether or not a learned constraint can be generalized on other variables of the same types as those of the learned constraint. The second query, called *recommendation query*, based on the analysis of the partial constraint graph learned so far, by using techniques borrowed from data mining and link prediction in dynamic graphs. Recommendation query asks the user whether or not a predicted constraint may belongs to the target network.

We propose several algorithms that manage these new concept of queries. First, we propose GenAcq algorithm, which is a generic algorithm that asks generalization queries. Second, we present Mine&Ask algorithm, which is able to learn types during the constraint acquisition process and to use the extracted types to build generalization queries. Third, we introduce Predict&Ask, which is a constraint recommender algorithm that uses the structure, of the constraint graph learned so far, to predict new constraints that may belong to the target network, and to ask the user to classify recommendation queries. Finally, all these generic algorithms are incorporated into QuAcq leading to three boosted versions, which are g-QuAcq (i.e, QuAcq+GenAcq), m-QuAcq (i.e, QuAcq+Mine&Ask), and p-QuAcq (i.e, QuAcq+ Predict&Ask). To show the efficiency of our new techniques, we experimentally evaluate their benefit on several benchmark problems. The results show that g-QuAcq, m-QuAcq and p-QuAcq dramatically improve the basic QuAcq algorithm in terms of number of queries.

The organization of this thesis is as follows. In chapter 2, we present the essential material to understand the technical presentation of this thesis. We present also the state of the art related to the constraint acquisition. In chapter 3, we introduce the concept of *generalization query* based on an aggregation of variables into types. We present a constraint generalization algorithm that can be plugged into any constraint acquisition system. We propose several strategies to make our approach more efficient in terms of number of queries. Finally, we experimentally

compare the recent QuAcQ system to an extended version boosted by the use of our generalization functionality. The results show that the extended version dramatically improves the basic QuAcQ. This chapter is based on the research previously published in the following papers:

— C. Bessiere, R. Coletta, A. Daoudi, N. Lazaar, Y. Mechqrane, and E.H. Bouyakhf. *Boosting constraint acquisition via generalization queries*. In ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applica- tions of Intelligent Systems (PAIS 2014), pages 99–104, 2014.

— C. Bessiere, R. Coletta, A. Daoudi, N. Lazaar, Y. Mechqrane, and E.H. Bouyakhf. Acquisition de contraintes par requêtes de généralisation. In Dixièmes Journées Francophones de Programmation par Contraintes (JFPC'14), Angers, France, 2014.

— C. Bessiere, R. Coletta, A. Daoudi, E. Hebrard, G. Katsirelos, N. Lazaar, Y. Mechqrane, N. Narodytska, C. Quimper, and T. Walsh. *New Approaches to Constraint Acquisition*. In ICON Book, Lecture Notes in Artificial Intelligence, April 2016.

In chapter 4, we present a new algorithm that is able to learn the structure of the problem during the constraint acquisition process. The idea is to infer potential types by analyzing the structure of the current constraint network and to use the extracted types to ask generalization queries. Our approach gives good results although no knowledge on the types is provided. Chapter 4 is based on the research previously published in the following paper:

— A. Daoudi, N. Lazaar, Y. Mechqrane, C. Bessiere, and E.H. Bouyakhf. *Detecting types of variables for generalization in constraint acquisition*. In 27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015, pages 413–420, 2015.

In chapter 5, we propose Predict&Ask, an algorithm based on the prediction of missing constraints in the partial network learned so far. Such missing constraints are directly asked to the user through *recommendation* queries, a new, more informative kind of queries. Predict&Ask can be plugged in any constraint acquisition system. We experimentally compare the QuAcQ system to an extended version boosted by the use of our recommendation queries. The results show that the extended version improves the basic QuAcQ. Chapter 5 based on the the following paper:

— A. Daoudi, Y. Mechqrane, C. Bessiere, N. Lazaar, and E.H. Bouyakhf. *Constraint acquisition using Recommendation Queries*. In IJCAI 2016, New York City, USA.

In chapter 6, we conclude this thesis and give some perspective to constraint acquisition.

CHAPTER

2 Background

Preamble

In this chapter, we present the essential material to understand the technical presentation of this thesis. We then present the state of the art related to the constraint acquisition.

Contents

2.1 Constraint Programming

This section introduces the formalism of constraint programming (CP), which can represent many academic and industrial problems. More details can be found in the text books [Lecoutre, 2013; Rossi et al., 2006a; Dechter, 2003].

Constraint programming is a declarative paradigm. The basic idea underlying CP is to model a combinatorial problem as a constraint network. That is, to specify a set of variables, a set of domain values, and a set of constraints. Each constraint is a rule that impose a limitation on the values that a variable, or a combination of variables, may be assigned. A solution of the constraint network is an assignment of variables to domain values that satisfies all constraints in the network. The Constraint Satisfaction Problem (CSP) is, therefore, the problem of determining whether

a solution exists, finding one or all solutions, finding whether or not a partial instantiation can be extended to a full solution. The CSP is not known to admit polynomial running time algorithms to solve its instances; hence, CSP is NP-hard.

After this brief overview of CP, we first start with formal definition of the central concepts.

2.1.1 Basic Definitions

Constraint satisfaction problems include two important components, namely variables with associated domains and constraints.

Definition 2.1 (Variable and Domain)**.** ***Variables*** *are objects or items that can take on a variety of values. The set of possible values for a given variable is called its* ***domain****. In our context, the pair* (X, D) *is called the vocabulary, with* X *is a finite set* $\{x_1, ..., x_n\}$ *of variables, and* $D = \{D(x_1), ..., D(x_n)\}$ *are a finite subsets of* $\mathbb{Z}$ *named the domains.*

The second component of a constraint satisfaction problem is the set of constraints themselves.

Definition 2.2 (Constraint)**.** *Constraints are rules that impose a limitation on the values that a variable, or a combination of variables, may be assigned. Formally speaking, a constraint* c *is a pair* $(var(c), rel(c)$*, 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 purpose of clarity, we limit ourselves to binary constraints; that is, constraints with a scope involving only two variables. In the following, 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 instance, $\neq_{12}$ denotes the constraint specified on $\langle x_1, x_2 \rangle$ with the relation "not equal".

A model that includes variables, their domains, and constraints is called a constraint network.

Definition 2.3 (Constraint Network)**.** *A constraint network over a given vocabulary* (X, D) *is a finite set* C *of constraints.*

In real problems, variables often represent components of the problem that can be classified in various types. For instance, in a school time-tabling problem, variables can represent teachers, students, rooms, courses, or time-slots. Such types are often known by the user.

Definition 2.4 (Variable Type)**.** *A type* T_i *is a subset of variables defined by the user as having a common property. A variable* x *is of type* T_i *if and only if* $x \in T_i$*. A scope*

$var = (x_1,\ldots,x_k)$ *of variables is said to* belong *to a sequence of types* $s = (T_1,\ldots,T_k)$ *(denoted by* $var \in s$*) if and only if* $x_i \in T_i$ *for all* $i \leq k$*. Consider* $s = (T_1, T_2,\ldots,T_k)$ *and* $s' = (T'_1, T'_2,\ldots,T'_k)$ *two sequences of types. We say that* s' covers s *(denoted by* $s \sqsubseteq s'$*) if and only if* $T_i \subseteq T'_i$ *for all* $i \in 1..k$*. A relation* r holds *on a sequence of types* s *if and only if* $(var, r) \in C$ *for all* $var \in s$*. A sequence of types* s *is* maximal *with respect to a relation* r *if and only if* r *holds on* s *and there does not exist* s' *covering* s *on which* r *holds.*

When a variable is assigned a value from its domain, we say that the variable has been instantiated.

Definition 2.5 (Assignment/Example)**.** *Given a vocabulary* (X, D)*. Let* $Y = \{x_1,...,x_k\}$ *be a subset of* X*. An assignment is a vector* $e_y = \{v_1,...,v_k\}$ *in* $D^{|Y|}$*. This assignment is partial if and only if* $Y \neq X$*, complete otherwise (dented by* e*).* e_y *is rejected by a constraint* c *(i.e.,* $e_y \not\models c$*) if and only if* $var(c) \subseteq var(e_y)$ *and the projection* $e_y[var(c)]$ *of* e_y *on* $var(c)$ *is not in* c*. Otherwise we say that* e_y *is accepted by* c*.*

A solution of the constraint network is an assignment of variables to domain values that satisfies all constraints of the network.

Definition 2.6 (Solution of a Constraint Network)**.** *A solution of a Constraint Network* C *is an assignment of each variable of the* constraint network *to a value in its domain such that all the constraints are simultaneously satisfied. Formally speaking, a complete assignment* e *of* X *is a solution of* C *if and only if for all* $c \in C$*,* c *does not reject* e*. We denote by* $sol(C)$ *the set of solutions of* C*.*

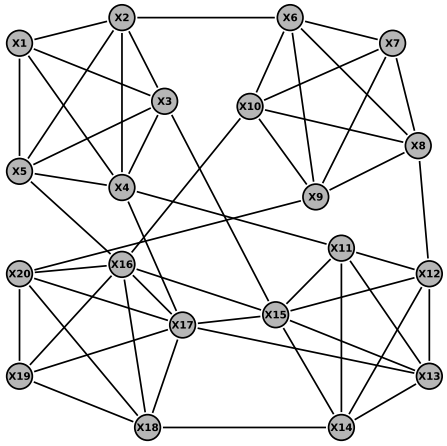

Figure 2.1 – An example of constraint graph

Graph concept is very useful in capturing the structure of a constraint problem. A constraint network can be represented by a graph called a primal constraint

graph, where each vertex represents a variable and the arcs connect all vertices whose variables belong to a constraint scope (see Figure 2.1).

Definition 2.7 (Constraint Graph)**.** *A constraint graph can be associated with a constraint graph* $G = (V, E)$*, where vertices* V *represent variables* X*, and edges* E *represent constraints* C*.*

2.1.2 Examples of Constraint Networks

In this section, we present some common examples of problems that can be intuitively modeled as constraint networks.

Queens Problem

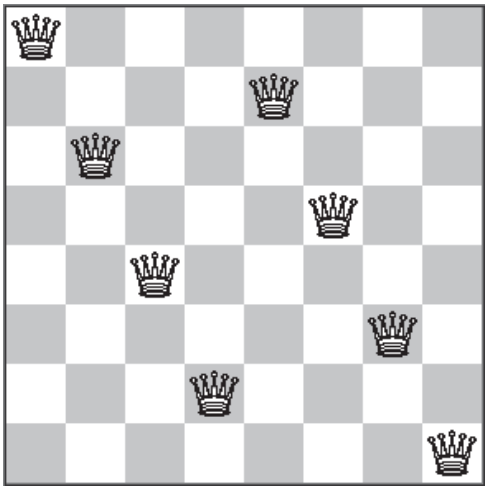

Figure 2.2 – An example of 8-Queens problem

The Description. The classic example used to illustrate a constraint satisfaction problem is the n-queens problem (see Figure 2.2). The problem is to place n queens on a $n \times n$ chessboard such that the placement of no queen constitutes an attack on any other.
The Model. One possible constraint network formulation of the problem is as follows. There is a variable for each column of the chessboard.
Variables: $X = \{x_1, \ldots, x_n\}$;
Domains: $D_i = \{1, \ldots, n\}$;
Constraints:

— $\forall i, j \in \{1, \ldots, n\}\ x_i \neq x_j$ (One queen each row.)

— $\forall i, j \in\in 1, \ldots, n\ |x_i - x_j| \neq |i - j|$ (One queen diagonally.)

Graph-coloring Problem

The Description. Another example of constraint network is the graph-coloring problem. In this problem, the goal is to color the nodes of a graph so that there is not a pair of linked nodes colored with the same color. Each node has a finite number of possible colors. This problem can be modeled as a constraint network by representing each node of the graph as a variable. The domain of each variable is defined by the possible colors that this variable can take. There exits a constraint between each pair of linked variables that forbids these variables to have the same color.

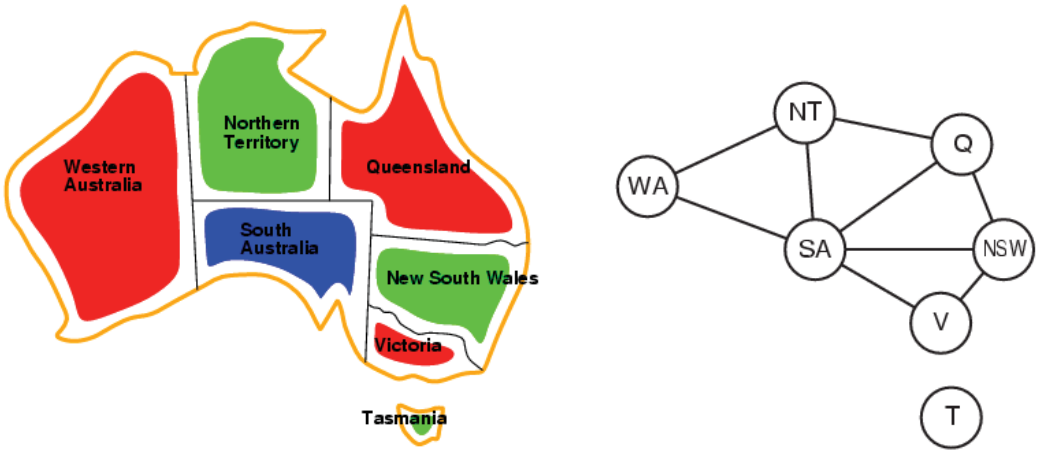

Figure 2.3 – An example of graph-coloring problem

The Model. The graph-coloring problem of Figure 2.3 is modeled as follows.
Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$;
Domains: $D_i = \{red, green, blue\}$;
Constraints: adjacent regions must have different colors.

Sudoku Problem

The Description. Sudoku, originally called Number Place, is a logic-based combinatorial number-placement puzzle (Figure 2.4). The task is to fill a 9×9 grid with digits so that each column, each row, and each block contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a unique solution.
The Model. The Sudoku puzzle can be modeled as follows.
Variables: $X = \{x_{11}, \ldots, x_{99}\}$;
Domains: $D = \{D_{ij} = \{1, \ldots, 9\} \mid \forall i, j \in 1, \ldots, 9\}$;
Constraints:

— $\forall i \in 1, \ldots, 9$ $\mathrm{AllDifferent}(x_{ij} \mid j \in 1, \ldots, 9)$;

— $\forall j \in 1,\ldots,9$ AllDifferent$(x_{ij} \mid i \in 1,\ldots,9)$;

— $\forall i,j \in 1,\ldots,9$ AllDifferent$(x_{(3i+k)(3j+q)} \mid k,q \in 1,\ldots,3)$.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 2.4 – An example of sudoku problem.

2.1.3 Solving Methods

Constraint satisfaction problems are solved using search techniques. The most used techniques are variants of backtracking, constraint propagation, and local search.

Backtracking Search

Backtracking is a recursive algorithm [Knuth, 1968; Rossi et al., 2006b]. It maintains a partial assignment of the variables. Initially, all variables are unassigned. At each step, a variable is chosen, and all possible values are assigned to it consecutively. For each value, the consistency of the partial assignment with the constraints is checked, and in case of consistency, a recursive call is performed. When all values have been tried, the algorithm backtracks. In this basic backtracking algorithm, consistency is defined as the satisfaction of all constraints whose variables are all assigned. In the literature, several variants of backtracking have been proposed. Backmarking [Gaschnig, 1977] improves the efficiency of checking consistency. Backjumping [Prosser, 1993] allows saving part of the search by backtracking "more than one variable" in some cases. Constraint learning [Dechter, 1990] infers and saves new constraints that can be later used to avoid exploring part of the tree search. Look-ahead [Haralick and Elliott, 1980] is also often used in

backtracking to anticipate the effects of choosing a variable or a value, thus, sometimes determining in advance when a subproblem is satisfiable or unsatisfiable.

Propagation

Constraint propagation techniques are methods used to modify the structure of a constraint satisfaction problem [Bessiere, 2006]. More accurately, they are methods that enforce a form of local consistency, which are conditions related to the consistency of a group of variables and/or constraints. Constraint propagation has various uses. First, it change a problem into equivalent one, which is usually simpler to solve. Second, it may prove satisfiability or unsatisfiability of problems. This is not guaranteed to happen in general; however, it always happens for some forms of constraint propagation and for some certain kinds of problems. The most used form of local consistency are arc consistency, hyper-arc consistency, and path consistency. The most popular constraint propagation method is the $AC-3$ algorithm [Mackworth, 1977], which enforces arc consistency.

Local search

Local search methods [Hoos and Tsang, 2006] are incomplete satisfiability algorithms. They may find a solution of a problem, but they may fail even if the problem has a solution. They work by iteratively improving a complete assignment over the variables. At each step, it changes the values of some variables, with the overall purpose of increasing the number of constraints satisfied by this assignment. based in that principle, the min-conflicts algorithm [Minton et al., 1992] is a local search algorithm dedicated for solving CSPs. In practice, local search appears to work well when these changes are also affected by random choices. Integration of backtracking search with local search has been developed, leading to hybrid algorithms [Wallace and Schimpf, 2002].

2.2 Constraint Language and Basis

In this section, we present two essential components for the constraint acquisition problem, which are constraint language and basis. A constraint language is a set of relations that restricts the type of constraints that are allowed when modeling a constraint satisfaction problem as a constraint network.

Definition 2.8 (Constraint Language)**.** *A constraint language is a set* $\Gamma = \{r_1, \ldots, r_t\}$ *of* t *relations over some subset of* $\mathbb{Z}$*.*

We are now ready to define the basis for the constraint acquisition problem. The basis is the set of all possible constraints that are candidate for being in the target constraint network.

Definition 2.9 (Basis)**.** *Given a vocabulary* (X, D) *and a constraint language* Γ*. The basis for the constraint acquisition problem is the set* B *of all constraints* c *for which* $var(c)$ *is a sequence of variables of* X*, and there exists a relation* r_i *in* Γ *such that* $rel(c) = r_i \cap D^{|var(c)|}$ *.*

It is worth to notice that the building of a basis B using a constraint language of bounded arity k is polynomial in the input dimension; the size of B is bounded by n^{kt} in the general case, and by n^{2t} in the case of binary constraint networks. However, if Γ contains global constraints, th size of B is no longer polynomial; as one global constraint gives rise to 2^n possible constraints in B.

Example 2.1. *Consider the binary constraint language* $\Gamma = \{\neq, \geq, \leq\}$ *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* $C = \{\neq_{12}, \geq_{12}, \leq_{23}\}$ *is indeed a subset of the basis* $B = \{\neq_{12}, \geq_{12}, \leq_{12}, \neq_{13}, \geq_{13}, \leq_{13}, \neq_{23}, \geq_{23}, \leq_{23}\}$.

2.3 Concept Learning

Inducing general functions from specific training examples is a main issue of machine learning [Mitchell, 1997]. Concept Learning consists on acquiring the definition of a general concept from positive and negative training examples of the target concept. Concept Learning can be seen as a problem of searching through a predefined space of potential hypotheses for the hypothesis that is consistent with the training examples. The hypothesis space has a general-to-specific ordering of hypotheses, and the search can be efficiently organized by taking advantage of a naturally occurring structure over the hypothesis space.

In our setting, which is acquiring constraints from examples, a *general concept* is a Boolean function over $D^X = \Pi_{x_i \in X} D(x_i)$; that is, a map that assigns to each example $e \in D^X$ a value in $\{0,1\}$. A representation of a general concept f is a constraint network C for which $f^{-1}(1) = sol(C)$. A general concept f is said to be representable by a basis B if there is a subset C of B such that C is a representation of f. We call *target concept* the concept f_T that returns 1 for e if and only if e is a solution of the problem the user has in mind. The target constraint is represented by a *target constraint network* denoted by C_T.

Definition 2.10 (Membership query)**.** *A* membership query $Ask(e)$*, with* $var(e) \subseteq X$*, is a classification question asked to the user, where* e *is an assignment in* $D^{var(e)} = \Pi_{x_i \in var(e)} D(x_i)$*. A set of constraints* C accepts *an assignment* e *if and only if there does not exist any constraint* $c \in C$ *rejecting* e*. The answer to* $Ask(e)$ *is* yes *if and only if* C_T *accepts* e*.*

2.4 Constraint Acquisition

In Constraint Programming, it is well known that the modeling task is a bottleneck [Freuder, 1999; Puget, 2004]. This bottleneck has a negative effect on the

uptake of CP technology on a large scale, especially in the industrial field. An answer to the modeling task is constraint acquisition, which lies in the conjunction of two research areas, namely Constraint Programming and Machine Learning. The idea behind that is to assist a non-expert user in modeling her problem as a constraint network. Several constraint acquisition systems have been proposed in the last decade; we will survey the existing systems in the following section. In this section, we formally define constraint acquisition.

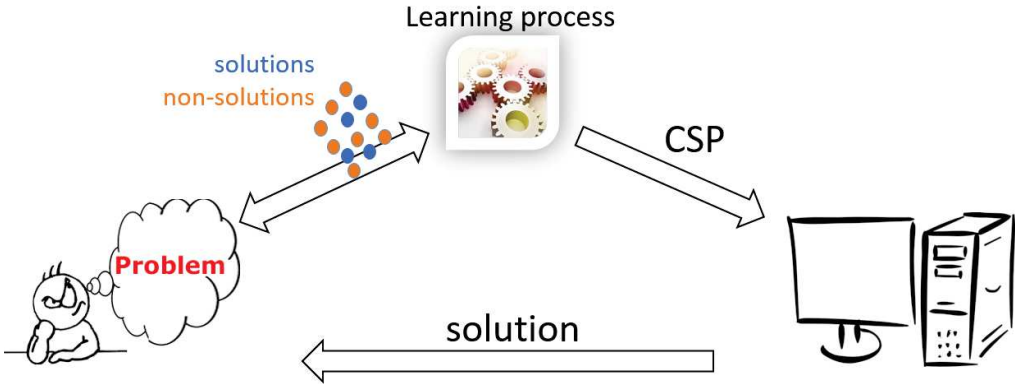

Figure 2.5 – Constraint acquisition.

Constraint acquisition can be seen as an interplay between the user and the learner (see Figure 2.5). The learner has a combinatorial problem in her mind, noted C_T. We suppose that the common knowledge between the user and the learner is the vocabulary (X, D). In one hand, the user has in her disposal a set of examples that represent both solutions and non-solutions of her problem. In the other hand, the learner owns a library of constraints L and a basis B that we suppose capturing the problem of the user. The objective of the learner is to induce a constraint network C_L that best fits the training examples proposed by the user, and which is equivalent to C_T (i.e, $C_L \equiv C_T$). That is, a constraint network that accepts the positive examples and rejects the negative ones.

In the literature, there are two forms of constraint acquisition. The first form is a passive learning, and the second form is an active learning.

2.4.1 Passive Learning

In a passive constraint acquisition context, there is no interplay between the user and the learner. The user provides a set of positive and negative examples, and the learner tries to find a set of constraints that classifies the training examples.

Definition 2.11 (Passive constraint acquisition problem)**.** *Given a vocabulary* (X, D)*, an unknown constraint network* C*, a basis* B*, a constraint language* L*, a set of solu-*

tions E^+, and an other set of non-solutions E^- of the problem the user has in mind, the constraint acquisition problem is, therefore, to find C such that:

- $C \subseteq B$
- *For each $e \in E^+$, e satisfies all constraints in C*
- *For each $e \in E^-$, e rejected by at least one constraint in C*

2.4.2 Active Learning

By contrast, in an active setting, the user dialogs with the learner. The learner proposes informative examples, and the user classifies them as positive or negative. The benefit of such kind of learning is that it provides less of a burden on the user.

Definition 2.12 (Active constraint acquisition problem)**.** *Given a basis B and an unknown user classification function f, the active constraint acquisition problem is to find a converging sequence $Q = \{q_1, \ldots, q_m\}$ of queries. That is, a sequence such that q_{i+1} is a query relative to B and C_L, where C_L is the constraint network learned so far, i.e the learned constraint network after classifying q_i. After the classification of the query q_m, the learner converges on the target constraint network, i.e $C_L \equiv C_T$.*

The general context underlying this thesis is the active constraint acquisition. Our objective is to make this kind of learning more efficient in practice despite the context of use.

2.4.3 Constraint Acquisition as a Search

Constraint acquisition can be seen as a form of search through a search space. The search space can be represented as a lattice (see Figure 2.6). Each node of the lattice is a possible constraint network. The top $\top$ of the lattice represents the general constraint network, which accepts all examples provided by the user. The specific born of the lattice is represented by all constraints of the basis B. The search space is explored via examples classified by the user. In the case of a positive example, we prune the search space by removing all constraints that reject the example. Hence, we generalize our learned constraint network, and we then go up through the lattice. In the case of a negative example, we learn at least one constraint that explains the rejection of that example. That is, we specify our learned constraint network, by going dawn through the lattice.

2.5 Existing Constraint Acquisition Systems

In this section, we survey the existing constraint acquisition systems proposed in the last years.

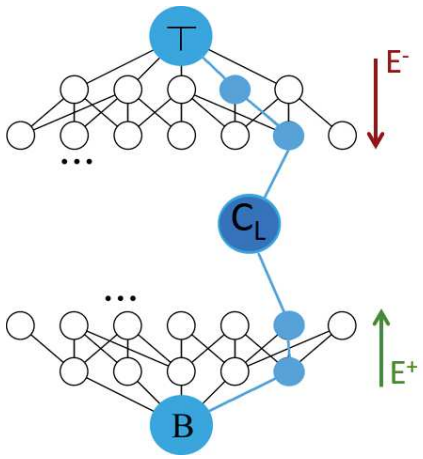

Figure 2.6 – Search space in constraint acquisition

2.5.1 CONACQ.1

CONACQ.1 is a SAT-based algorithm presented in [Bessiere et al., 2004, 2005] for acquiring constraint networks based on version space [Winston, 1992; Mitchell, 1997]. In CONACQ.1, Bessiere et al. made the assumption that the only thing the user is able to provide is examples of solutions and non-solutions of the target problem. Based on these examples, the CONACQ.1 system learns a set of constraints that correctly classifies all examples given so far. This type of learning is called *passive* learning.

Description of CONACQ.1

The set of the training examples provided to CONACQ.1 by the user is composed of negative and positive examples.

Definition 2.13 (Training Data)**.** *The set* E^f *of training examples provided to* CONACQ.1 *is composed of a set* E *of instances and a classification function* $f : E \longrightarrow \{0,1\}$*. Given an element* e *of* E*,* $f(e) = 1$ *if and only if* e *is a solution of the problem that the user has in mind. That is,* e *is a positive example noted* e^+*. If* e *is a non-solution of the problem, therefore,* $f(e) = 0$*, and we say that the example is negative, noted* e^-*.*

We say that a constraint network C is consistent with the training data E^f if and only if each example $e^+ \in E^f$ belongs to the set of solutions of C, and each $e^- \in E^f$ is a non-solution of C. Given a basis B and a set of training data E^f, the problem of constraint acquisition consists on finding a constraint network C admissible for B and consistent with the training data E^f.

CONACQ.1 is a constraint acquisition system based on the version space introduced by Mitchel in [Mitchell, 1997]. The version space is a supervised machine

learning technique that aims at acquiring a target concept from positive and negative examples. Formally speaking, given a set of hypotheses H and a set of training examples E, the version space is the set of hypotheses of H that are consistent with the training examples. In the setting of constraint acquisition, the version space $V_B(E^f)$, of a constraint acquisition problem, is the set of all constraint networks admissible by B and consistent with the training examples E^f. In the SAT-based formulation of CONACQ.1, the version space is encoded by the clausal theory K, where each model of this theory is a constraint network of $V_B(E^f)$.

If B is the basis, a literal is either an atom b_{ij} of B or its negation $\neg b_{ij}$. It is wroth to stress that $\neg b_{ij}$ is not a constraint, but it point out that b_{ij} does not belong to the learned constraint network. A clause is a disjunction of literals, and the clausal theory K is a conjunction of clauses.

Definition 2.14 (Interpretation)**.** *Let B be a basis. An interpretation I over B is a function that assigns to each literal b_{ij} in B a value $I(b_{ij})$ in $\{0,1\}$*

Definition 2.15 (Transformation)**.** *Let B be a basis. A transformation is a function ϕ that assigns to each interpretation I of B the constraint network $\phi(I)$ admissible by B and such that:*

$$C_{ij} \in \phi(I) \text{ iff } C_{ij} = \bigcap\{b_{ij}^p \in B \mid I(b_{ij}^p) = 1\}$$

An interpretation I is a model of the clausal theory K if K is true in I according to the standard propositional semantic. The set of all the models of K is noted $Models(K)$.

The construction of K is as follows. For each training example e provided by the user, CONACQ.1 builds the set $k(e)$, of literals corresponding to the constraints $b_{ij} \in B$, that rejects e. Then, CONACQ.1 adds iteratively a set of clauses such that, for each interpretation I of $Models(K)$, the network $\phi(I)$ classifies correctly the the training examples given so far, as well as the example e. The update of the clausal theory differs according to the learning class of e:

— If e is a negative example, then one of the constraints in $k(e)$ explains the rejection of e. We then add to K the disjunction $\{\bigvee_{b_{ij} \in k(e)} b_{ij}\}$ of elements of $k(e)$.

— If e is a positive example, then CONACQ.1 is ensured that no one of the constraints of $k(e)$ belongs to the target network. Consequently, K is updated by adding the conjunction of the unitary clauses $\neg b_{ij}$ of $k(e)$.

The analysis of positive instances of E^f allows to simplify, with unitary propagation, the clauses that explain the rejection of negative examples. The correction of the algorithm CONACQ.1 is formulated by the theorem 2.1.

Theorem 2.1 (Correctness of CONACQ.1)**.** *Let E^f be a set of training examples and B a basis. The clausal theory built by CONACQ.1 for E^f and B is such that:*

$$V_B(E^f) = \{\phi(m) \mid m \in Models(K)\}$$

Proof. The proof of the theorem 2.1 is given in [Bessiere et al., 2005]. ∎

CONACQ.1 has numerous advantages. Firstly, the formulation is generic; therefore, it can use any SAT solver as a basis for version space learning. Secondly, it can exploit powerful SAT concepts, such as unit propagation and backbone detection to improve learning rate. Finally, CONACQ.1 can easily incorporate domain-specific knowledge in constraint programming to improve the quality of the learned network. Specifically, for handling redundant constraints in constraint acquisition, two generic techniques have been developed. The first is based on the notion of redundancy rules, which can deal with some forms of redundancy. The second technique, based on backbone detection, is far more powerful.

Despite its distinct number of advantages, CONACQ.1 has also some limitations. Firstly, It cannot acquire global constraints. Secondly, it may require a number exponential of examples to get the target network [Bessiere and Koriche, February, 2012]. Thirdly, and finally, CONACQ.1 needs positive examples to learn constraints; why modeling a problem if we have already, at our disposal, solutions for it?

2.5.2 CONACQ.2

To overcome the limitations reproached to CONACQ.1, Bessiere et al. have proposed an active learning version, called CONACQ.2 [Bessiere et al., 2007]. The system CONACQ.2 proposes examples to the user to classify as solutions or non-solutions. Such questions are called membership queries [Angluin, 1987].

Description of CONACQ.2

In order to generate queries, the authors proposed several approaches. First, a random affectation to the variables, which is a baseline approach. Second, a query with a small $K(e)$. Finally, a query with a large $k(e)$. It is worth to notice that using these approaches could generate useless queries. In fact, a query does not contain information if it is rejected by the version space. We then talk about redundant queries.

Definition 2.16 (A redundant query)**.** *A redundant query is a query that do not allow us to learn new informations. Let e be an example, we have tow cases of redundant queries:*

- *Either $k(e) = \varnothing$ and in this case we automatically detect that e is a solution;*
- *Or all constraints rejecting e are already in K; then, any constraint network in the version space accepts e.*

Hence, redundant queries are useless, and it is not necessary asking the user to classify them.

To make CONACQ.2 more efficient in terms of number of queries, the authors propose some optimizations. They propose to break some clauses of K. That is, they propose to compute a query with $k(e)$ containing a strict subset of literals present in

the clause that they wish to break. If this query is classified as negative, they get a smaller clause that contains only literals existing in $k(e)$. Otherwise, they decrease the size of the clause by setting all literals of $k(e)$ to false. Note that if one seeks a query with a $k(e)$ of size 1, and if the query is classified by the user as negative; then, the clause will be simplified to a single variable, which must therefore be true.

The last optimization, proposed by the authors, concerns a case occurring if they try to break clauses. This case arises when they force the not fixed variables to be true in order to such variables do not appear in $k(e)$. In this case, it is possible that no query exists. They propose to extract a conflict set S of literals and to add the clause $\bigvee_{b_{ij} \in S} \neg b_{ij}$ forcing at least one of the variables to be false. In practice, there are several methods to produce such conflict sets. One method is, for each variable $b_{ij} \in k(e)$, to test whether to fix it to true; then, K becomes inconsistent. In this case, this variable must be false.

Even that CONACQ.2 is active and generates only informative examples, it remains exponentially large in terms of number of queries; it may require a large number of queries to get the target constraint network. Thus, it is hard to put CONACQ.2 in practice. An other weakness of CONACQ.2 is that it is not able to learn global constraints that are not decomposable.

2.5.3 MODELSEEKER

In [Beldiceanu and Simonis, 2012], Beldiceanu and Simonis proposed *Model Seeker*, another passive learning approach. Positive examples are provided by the user. The system arranges these examples as a matrix, or any other data structure, and identifies constraints in the global constraints catalog ([Beldiceanu et al., 2007]) that are satisfied by rows or columns of all examples.

Description of MODELSEEKER

The general schema of MODELSEEKER is shown in Figure 2.7. Here, we will give a general overview of the different steps of the system. MODELSEEKER takes as input positive examples of the problem that we want to model as a constraint network.

Transformation In the first step, MODELSEEKER tries to convert the format of input examples to other more appropriate representations. For instance, it tries to replace a boolean format with finite domain values. MODELSEEKER also converts different graph representations into the successor variable form used by the global constraints in the catalog.

Candidate Generation The second step (*Sequence Generation*) tries to group the variables of the example into regular subsets. For instance, it interprets the input vector as a matrix and creates subsequences for all rows and all columns. In the *Argument Creation* step, MODELSEEKER creates call patterns for constraints from the subsequences. it can try each subsequence on its own, or combine pairs of subsequences, or use all subsequences together in a single collection. It also tries to add additional arguments based on functional dependencies of constraints, described

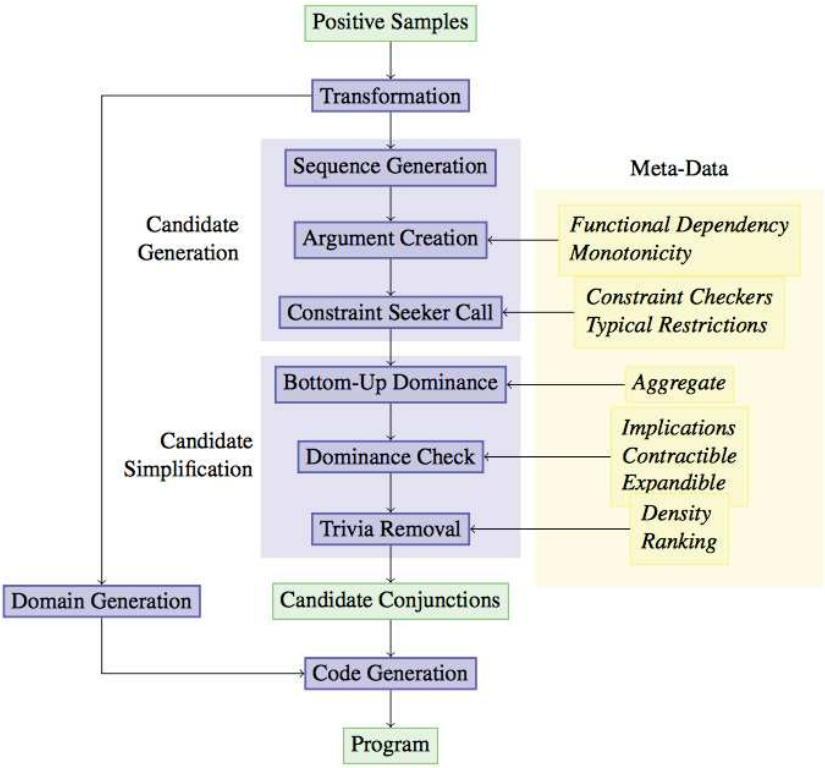

Figure 2.7 – General schema of MODELSEEKER.

as meta-data in the global constraint catalog. For each call pattern, MODELSEEKER then calls the Constraint Seeker to discover matching constraints, which satisfy all subsequences of each example. Only the highest ranking candidates are retained for further analysis.

Candidate Simplification After the Constraint Seeker step, MODELSEEKER potentially has a very large list of possible candidate global constraints (up to 2 000 global constraints for some examples). The next step is to reduce this list as much as possible. first, MODELSEEKER applies a *Dominance Check* to remove all candidate constraints that are implied by other constraints in the candidate list. The dominance check is the core of modeling system, helping to remove useless constraints from the candidate list. In the last step before the final candidate list output, the system *removes trivial constraints* and simplifies some constraint pattern. At the end of this step, MODELSEEKER performs a ranking of the remaining candidates based on the constraint and sequence generator used.

Code Generation As a side effect of the initial transformation in the first step, MODELSEEKER *generates potential domains* for the variables of the problem under

consideration. In the default case, the system just creates variable domains using all values occurring in the examples. But, for some graph-based transformations a more accurate domain definition is used. At the last step, which is *Code Generation*, given the candidate list and domains for the variables, It can be easily to generate code for a model using the constraints. MODELSEEKER produces SICStus Prolog code using the call format of the catalog. The generated code can then be used to validate the provided examples or to find solutions for the learned constraint network.

MODELSEEKER remains the only system that deals with global constraints. The major drawback of MODELSEEKER is that it does not handle all global constraints in the catalog [Beldiceanu et al., 2007]. It deals only with 67 global constraints. Another reproach to that system is that it learns only from positive examples; hence, it makes the system hard to put in practice, as the provide of such examples by a novice is not an obvious task.

2.5.4 QUACQ

QUACQ is a recent active learner system that is able to ask the user to classify *partial* queries [Bessiere et al., 2013]. Using partial queries and given a negative example, QUACQ is able to find a constraint of the problem the user has in mind in a number of queries logarithmic in the size of the example. This key component of QUACQ allows it to always converge on the target set of constraints in a polynomial number of queries. However, even that good theoretical bound can be hard to put in practice. For instance, QUACQ requires the user to classify more than 8000 examples to get the complete Sudoku model.

Description of QUACQ

QUACQ takes as input a basis B on a vocabulary (X, D). It asks partial queries of the user until it has converged on a constraint network C_L equivalent to the target network C_T, or collapses. When a query is answered *yes*, constraints rejecting it are removed from B. When a query is answered no, QUACQ enters a loop (functions `FindScope` and `FindC`) that will end by the addition of a constraint to C_L.

QUACQ (see Algorithm 1) initializes the network C_L it will learn to the empty set (line 2). If C_L is unsatisfiable (line 6), the space of possible networks collapses because there does not exist any subset of the given basis B that is able to correctly classify the examples already asked of the user. In line 8, QUACQ computes a complete assignment e satisfying C_L but violating at least one constraint from B. If such an example does not exist (line 10), then all constraints in B are implied by C_L, and we have converged. If we have not converged, we propose the example e to the user, who will answer by *yes* or no. If the answer is *yes*, we can remove from B the set $\kappa_B(e)$ of all constraints in B that reject e (line 12). If the answer is no, we are sure that e violates at least one constraint of the target network C_T. We then call the function `FindScope` to discover the scope S of one of these violated constraints (line

Algorithm 1: QUACQ: Acquiring a constraint network C_T with partial queries

```
1  C_L ← ∅;
2  while true do
3      if sol(C_L) = ∅ then return "collapse";
4      choose e in D^X accepted by C_L and rejected by B ;
5      if e = nil then return "convergence on C_L";
6      if ASK(e) = yes then B ← B \ κ_B(e) ;
7      else
8          S ← FindScope(e, ∅, X, false);
9          c_S ← FindC(S);
10         if c_S = nil then return "collapse";
11         else C_L ← C_L ∪ {c_S};
```

20). `FindC` will return a constraint of C_T whose scope is in S (line 9). If no constraint is returned (line 22), this is again a condition for collapsing as we could not find in B a constraint rejecting one of the negative examples. Otherwise, the constraint returned by `FindC` is added to the learned network C_L (line 28).

The recursive function `FindScope` (see Algorithm 2) takes as parameters an example e, two sets R and Y of variables, and a Boolean ask_query. An invariant of `FindScope` is that e violates at least one constraint whose scope is a subset of $R \cup Y$. When `FindScope` is called with $ask\_query =$ **false**, we already know whether R contains the scope of a constraint that rejects e (line 3). If $ask\_query =$ **true** we ask the user whether $e[R]$ is positive or not (line 4). If *yes*, we can remove all the constraints that reject $e[R]$ from the basis, otherwise we return the empty set (line 5). We reach line 6 only in case $e[R]$ does not violate any constraint. We know that $e[R \cup Y]$ violates a constraint. Hence, as Y is a singleton, the variable it contains necessarily belongs to the scope of a constraint that violates $e[R \cup Y]$. The function returns Y. If none of the return conditions are satisfied, the set Y is split in two balanced parts (line 7) and we apply a technique similar to QUICKXPLAIN ([Junker, 2004a]) to elucidate the variables of a constraint violating $e[R \cup Y]$ in a logarithmic number of steps (lines 8–10).

The function `FindC` (see Algorithm 3) takes as parameter Y, the scope on which `FindScope` has found that there is a constraint from the target network C_T. `FindC` first removes from B all constraints with scope Y that are implied by C_L because there is no need to learn them (line 3). The set Δ is initialized to all candidate constraints (line 4). In line 6, an example e is chosen in such a way that Δ contains both constraints rejecting e and constraints satisfying e. If no such example exists (line 7), this means that all constraints in Δ are equivalent wrt $C_L[Y]$. Any of them is returned except if Δ is empty (lines 8-9). If a suitable example was found, it is proposed to the user for classification (line 10). If classified positive, all constraints rejecting it are removed from B and Δ (line 11). Otherwise we test whether the

Algorithm 2: Function `FindScope`: returns the scope of a constraint in C_T

```
1  function FindScope(in e: example, R,Y: scopes, ask_query: Boolean): scope;
2  begin
3      if ask_query then
4          if ASK(e[R]) = yes then B ← B \ κ_B(e[R]) ;
5          else return ∅;
6      if |Y| = 1 then return Y;
7      split Y into <Y_1, Y_2> such that |Y_1| = ⌈|Y|/2⌉ ;
8      S_1 ← FindScope(e, R ∪ Y_1, Y_2, true);
9      S_2 ← FindScope(e, R ∪ S_1, Y_1, (S_1 ≠ ∅));
10     return S_1 ∪ S_2;
```

Algorithm 3: Function `FindC`: returns a constraint of C_T with scope Y

```
1  function FindC(in Y: scope): constraints;
2  begin
3      B ← B \ {c_Y | C_L ⊨ c_Y};
4      Δ ← {c_Y | (c_Y ∈ B[Y])};
5      while true do
6          choose e in sol(C_L[Y]) such that ∃c_Y, c'_Y ∈ Δ, e ⊨ c_Y and e ⊭ c'_Y;
7          if e = nil then
8              if Δ = ∅ then return nil;
9              else pick c_Y in Δ; return c;
10         if ASK(e) = yes then
11             B ← B \ κ_B(e); Δ ← Δ \ κ_B(e);
12         else
13             if ∃c_S ∈ κ_B(e) | S ⊊ Y then
14                 return FindC(FindScope(e, ∅, Y, false));
15             else Δ ← Δ ∩ κ_B(e) ;
```

example e does not violate constraints with scope strictly included in Y (line 13). If yes, we recursively call `FindScope` and `FindC` to find that smaller arity constraint before the one having scope Y (line 14). If no, we remove from Δ all constraints accepting that example e (line 15) and we continue the loop of line 5.

Complexity of QuAcq

The complexity of QuAcq, in terms of queries, is given by Theorem 2.2.

Theorem 2.2 (Complexity). *Given a basis B built from a language Γ of bounded arity*

constraints, and a target network C_T, QuAcq *uses* $\mathcal{O}(C_T.(\log|X|+|\Gamma|))$ *queries to find the target network or to collapse and* $\mathcal{O}(|B|)$ *queries to prove convergence.*

Proof. The proof of the theorem 2.2 is given in [Bessiere et al., 2013]. ∎

Illustrative Example

Table 2.1 – The example

call	R	Y	ASK	return
0	∅	X_1,X_2,X_3,X_4,X_5	×	X_3,X_4
1	X_1,X_2,X_3	X_4,X_5	yes	X_4
1.1	X_1,X_2,X_3,X_4	X_5	no	∅
1.2	X_1,X_2,X_3	X_4	×	X_4
2	X_4	X_1,X_2,X_3	yes	X_3
2.1	X_4,X_1,X_2	X_3	yes	X_3
2.2	X_4,X_3	X_1,X_2	no	∅

We illustrate the behavior of QuAcq on a simple example. Consider the set of variables $X_1,\ldots,X_5$ with domains $\{1..5\}$, a language $\Gamma=\{=,\neq\}$, a basis $B=\{=_{ij},\neq_{ij}|\ i,j\in 1..5, i<j\}$, and a target network $C_T=\{=_{15},\ \neq_{34}\}$. Suppose the first example generated in line 8 of QuAcq is $e_1=(1,1,1,1,1)$. The trace of the execution of `FindScope`$(e_1,\varnothing,X_1\ldots X_5,$**false**$)$ is in Table 2.1. Each line corresponds to a call to `FindScope`. Queries are always on the variables in R. '×' in the column ASK means that the previous call returned ∅, so the question is skipped. The queries in lines 1 and 2.1 in the table permit `FindScope` to remove the constraints $\neq_{12},\neq_{13},\neq_{23}$ and $\neq_{14},\neq_{24}$ from B. Once the scope (X_3,X_4) is returned, `FindC` requires a single example to return $\neq_{34}$ and prune $=_{34}$ from B. Suppose the next example generated by QuAcq is $e_2=(1,2,3,4,5)$. `FindScope` will find the scope (X_1,X_5) and `FindC` will return $=_{15}$ in a way similar to the processing of e_1. The constraints $=_{12},=_{13},=_{14},=_{23},=_{24}$ are removed from B by a partial positive query on $X_1,\ldots,X_4$ and $\neq_{15}$ by `FindC`. Finally, examples $e_3=(1,1,1,2,1)$ and $e_4=(3,2,2,3,3)$, both positive, will prune $\neq_{25},\neq_{35},=_{45}$ and $=_{25},=_{35},\neq_{45}$ from B respectively, leading to convergence.

CHAPTER

3 Constraint Acquisition with Generalization Queries

Preamble

In this chapter, we introduce the concept of generalization query *based on an aggregation of variables into types. We present a constraint generalization algorithm that can be plugged into any constraint acquisition system. We propose several strategies to make our approach more efficient in terms of number of queries. Finally we experimentally compare the recent* QUACQ *system to an extended version boosted by the use of our generalization functionality. The results show that the extended version dramatically improves the basic* QUACQ*.*

Contents

3.1 Introduction

Constraint programming (CP) is used to model and to solve combinatorial problems in many application areas, such as resource allocation or scheduling. However, building a CP model requires some expertise in constraint programming. This prevents the use of this technology by a novice and thus this has a negative effect on the uptake of constraint technology by non-experts.

Several techniques have been proposed for assisting the user in the modeling task. In [Freuder and Wallace, 2002], Freuder and Wallace proposed the match-

maker agent, an interactive process where the user is able to provide one of the constraints of her target problem each time the system proposes a wrong solution. In [Lallouet et al., 2010], Lallouet et al. proposed a system based on inductive logic programming that uses background knowledge on the structure of the problem to learn a representation of the problem correctly classifying the examples. In [Bessiere et al., 2004, 2005], Bessiere et al. made the assumption that the only thing the user is able to provide is examples of solutions and non-solutions of the target problem. Based on these examples, the *Conacq.1* system learns a set of constraints that correctly classifies all examples given so far. This type of learning is called *passive* learning. In [Beldiceanu and Simonis, 2012], Beldiceanu and Simonis proposed *Model Seeker*, another passive learning approach. Positive examples are provided by the user. The system arranges these examples as a matrix and identifies constraints in the global constraints catalog ([Beldiceanu et al., 2007]) that are satisfied by rows or columns of all examples.

By contrast, in an active learner like *Conacq.2*, the system proposes examples to the user to classify as solutions or non solutions [Bessiere et al., 2007]. Such questions are called membership queries [Angluin, 1987]. CONACQ introduces two computational challenges. First, how does the system generate a useful query? Second, how many queries are needed for the system to converge to the target set of constraints? It has been shown that the number of membership queries required to converge to the target set of constraints can be exponentially large [Bessiere and Koriche, February, 2012].

QUACQ is a recent active learner system that is able to ask the user to classify *partial* queries [Bessiere et al., 2013]. Using partial queries and given a negative example, QUACQ is able to find a constraint of the problem the user has in mind in a number of queries logarithmic in the size of the example. This key component of QUACQ allows it to always converge on the target set of constraints in a polynomial number of queries. However, even that good theoretical bound can be hard to put in practice. For instance, QUACQ requires the user to classify more than 8000 examples to get the complete Sudoku model.

In this chapter, we propose a new technique to make constraint acquisition more efficient in practice by using variable types. In real problems, variables often represent components of the problem that can be classified in various types. For instance, in a school time-tabling problem, variables can represent teachers, students, rooms, courses, or time-slots. Such types are often known by the user. To deal with types of variables, we introduce a new kind of query, namely, *generalization query*. We expect the user to be able to decide if a learned constraint can be generalized to other scopes of variables of the same type as those in the learned constraint. We propose an algorithm, GENACQ for *generalized acquisition*, that asks such generalization queries each time a new constraint is learned. We propose several strategies and heuristics to select the good candidate generalization query. We plugged our generalization functionality into the QUACQ constraint acquisition system, leading to the G-QUACQ algorithm. We experimentally evaluate the benefit of our technique

on several benchmark problems. The results show that G-QUACQ dramatically improves the basic QUACQ algorithm in terms of number of queries.

The rest of this chapter is organized as follows. Section Section 3.2 describes the generalization algorithm. In Section 3.3, several strategies are presented to make our approach more efficient. Section 3.4 presents the experimental results we obtained when comparing G-QUACQ to the basic QUACQ and when comparing the different strategies in G-QUACQ. Section 3.5 concludes the chapter and gives some directions for future research.

3.2 GENACQ Algorithm

In this section we present GENACQ, a *generalized acquisition* algorithm, The idea behind this algorithm is, given a constraint c learned on $var(c)$, to generalize this constraint to sequences of types s covering $var(c)$ by asking generalization queries $AskGen(s,r)$. A generalization query $AskGen(s,r)$ is answered *yes* by the user if and only if for every sequence var of variables covered by s the relation r holds on var in the target constraint network C_T.

Algorithm 4: GENACQ (c, NonTarget)

```
2  Table ← {s | var(c) ∈ s} \ {var(c)}
4  G ← ∅
6  #NoAnswers ← 0
8  foreach s ∈ Table do
10     if ∃(s', r) ∈ NegativeQ | rel(c) ⊆ r ∧ s' ⊑ s then
12        Table ← Table \ {s}
14     if ∃c' ∈ NonTarget | rel(c') = rel(c) ∧ var(c') ∈ s then Table ← Table \ {s} ;
17 while Table ≠ ∅ ∧ #NoAnswers < cutoffNo do
19     pick s in Table
21     if AskGen(s, rel(c)) = yes then
23        G ← G ∪ {s} \ {s' ∈ G | s' ⊑ s}
25        Table ← Table \ {s' ∈ Table | s' ⊑ s}
27        #NoAnswers ← 0
28     else
30        Table ← Table \ {s' ∈ Table | s ⊑ s'}
32        NegativeQ ← NegativeQ ∪ {(s, rel(c))}
34        #NoAnswers++
36 return G;
```

3.2.1 Description of GenAcQ

The algorithm GenAcQ (see Algorithm 4) takes as input a target constraint c that has already been learned and a set NonTarget of constraints that are known not to belong to the target network. It also uses the global data structure NegativeQ, which is a set of pairs (s,r) for which we know that r does not hold on all sequences of variables covered by s. c and NonTarget can come from any constraint acquisition mechanism or as background knowledge. NegativeQ is built incrementally by each call to GenAcQ. GenAcQ also uses the set Table as local data structure. Table will contain all sequences of types that are candidates for generalizing c.

In line 2, GenAcQ initializes the set Table to all possible sequences s of types that contain $var(c)$. In line 3, GenAcQ initializes the set G to the empty set. G will contain the output of GenAcQ, that is, the set of maximal sequences from Table on which $rel(c)$ holds. The counter #NoAnswers counts the number of consecutive times generalization queries have been classified negative by the user. It is initialized to zero (line 6). #NoAnswers is not used in the basic version of GenAcQ but it will be used in the version with cutoffs. (In other words, the basic version uses cutoffNo $= +\infty$ in line 17).

The first loop in GenAcQ (line 8) eliminates from Table all sequences s for which we already know the answer to the query $AskGen(s,rel(c))$. In lines 10-12, GenAcQ eliminates from Table all sequences s such that a relation r entailed by $rel(c)$ is already known not to hold on a sequence s' covered by s (i.e., (s',r) is in NegativeQ). This is safe to remove such sequences because the absence of r on some scope in s' implies the absence of $rel(c)$ on some scope in s (see Lemma 3.1). In lines 14-15, GenAcQ eliminates from Table all sequences s such that we know from NonTarget that there exists a scope var in s such that $(var,rel(c)) \notin C_T$.

In the main loop of GenAcQ (line 17), we pick a sequence s from Table at each iteration and we ask a generalization query to the user (line 21). If the user says *yes*, s is a sequence on which $rel(c)$ holds. We put s in G and remove from G all sequences covered by s, so as to keep only the maximal ones (line 23). We also remove from Table all sequences s' covered by s (line 25) to avoid asking redundant questions later. If the user says *no*, we remove from Table all sequences s' that cover s (line 30) because we know they are no longer candidate for generalization of $rel(c)$ and we store in NegativeQ the fact that $(s,rel(c))$ has been answered *no*. The loop finishes when Table is empty and we return G (line 16).

3.2.2 Completeness and Complexity

We analyze the completeness and complexity of GenAcQ in terms of number of generalization queries.

Lemma 3.1. *If* $AskGen(s,r) = no$ *then for any* (s',r') *such that* $s \sqsubseteq s'$ *and* $r' \subseteq r$*, we have* $AskGen(s',r') = no$.

Proof. Assume that $\mathrm{AskGen}(s,r) = \mathrm{no}$. Hence, there exists a sequence $var \in s$ such that $(var,r) \notin C_T$. As $s \sqsubseteq s'$ we have $var \in s'$ and then we know that $(var,r) \notin C_T$. As $r' \subseteq r$, we also have $(var,r') \notin C_T$. As a result, $\mathrm{AskGen}(s',r') = \mathrm{no}$. ∎

Lemma 3.2. *If* $\mathrm{AskGen}(s,r) = yes$ *then for any* s' *such that* $s' \sqsubseteq s$, *we have* $\mathrm{AskGen}(s',r) = yes$.

Proof. Assume that $\mathrm{AskGen}(s,r) = yes$. As $s' \sqsubseteq s$, for all $var \in s'$ we have $var \in s$ and then we know that $(var,r) \in C_T$. As a result, $\mathrm{AskGen}(s',r) = yes$. ∎

Proposition 3.1 (Completeness)**.** *When called with constraint* c *as input, the algorithm* GENACQ *returns all maximal sequences of types covering* $var(c)$ *on which the relation* $rel(c)$ *holds.*

Proof. All sequences covering $var(c)$ are put in $Table$. A sequence in $Table$ is either asked for generalization or removed from $Table$ in lines 12, 15, 25, or 30. We know from Lemma 3.1 that a sequence removed in line 12, 15, or 30 would necessarily lead to a *no* answer. We know from Lemma 3.2 that a sequence removed in line 25 is subsumed and less general than another one just added to G. ∎

Proposition 3.2. *Given a learned constraint* c *and its associated* $Table$, GENACQ *uses* $O(|Table|)$ *generalization queries to return all maximal sequences of types covering* $var(c)$ *on which the relation* $rel(c)$ *holds.*

Proof. For each query on $s \in Table$ asked by GENACQ, the size of $Table$ strictly decreases regardless of the answer. As a result, the total number of queries is bounded above by $|Table|$. ∎

3.2.3 Illustrative Example

Let us take the Zebra problem to illustrate our generalization approach. The Lewis Carroll's Zebra problem has a single solution. The target network is formulated using 25 variables, partitioned in 5 types of 5 variables each. The types are thus *color*, *nationality*, *drink*, *cigaret*, *pet*, and the trivial type X of all variables. There is a clique of $\neq$ constraints on all pairs of variables of the same non trivial type and 14 additional constraints given in the description of the problem.

Figure 3.1 shows the variables of the Zebra problem and their types. In this example, the constraint $x_2 \neq x_5$ has been learned between the two color variables x_2 and x_5. This constraint is given as input of the GENACQ algorithm. GENACQ computes the $Table$ of all sequences of types covering the scope (x_2,x_5). $Table = \{(x_2,color), (x_2,X),(color,x_5),(color,color),(color,X),(X,x_5),(X,color), (X,X)\}$. Suppose we pick $s = (X,x_5)$ at line 19 of GENACQ. According to the user's answer (*no* in this case), the $Table$ is reduced to $Table = \{(x_2,color),(x_2,X),(color,x_5),(color,color),(color,X)\}$. As next iteration, let us pick $s = (color,color)$. The user will answer *yes* because there is indeed a clique of $\neq$ on the color variables. Hence, $(color,color)$ is added to G and the $Table$ is reduced to $Table = \{(x_2,X),(color,X)\}$. If we pick (x_2,X),

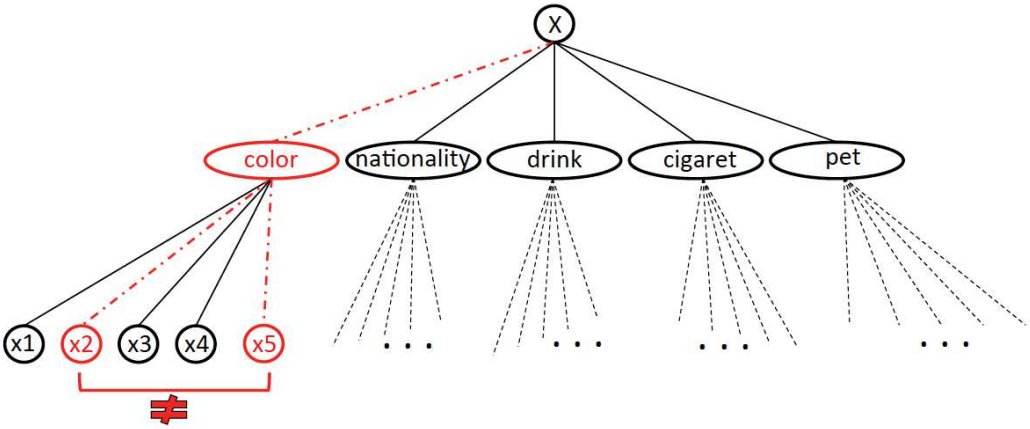

Figure 3.1 – Variables and types for the Zebra problem.

the user answers *no* and we reduce the Table to the empty set and return $G = \{(\text{color},\text{color})\}$, which means that the constraint $x_2 \neq x_5$ can be generalized to all pairs of variables in the sequence $(\text{color},\text{color})$, that is, $(x_i \neq x_j) \in C_T$ for all $(x_i,x_j) \in (\text{color},\text{color})$.

3.2.4 Using Generalization in QuAcq

GenAcq is a generic technique that can be plugged into any constraint acquisition system. In this section we present g-QuAcq, a constraint acquisition algorithm obtained by plugging GenAcq into QuAcq, the constraint acquisition system presented in [Bessiere et al., 2013],

g-QuAcq is presented in Algorithm 5. We do not give the code of functions `FindScope` and `FindC` as we use them exactly as they appear in [Bessiere et al., 2013]. But let us say a few words on how they work. Given sets of variables S_1 and S_2, `FindScope`$(e,S_1,S_2,false)$ returns the subset of S_2 that, together with S_1 forms the scope of a constraint in the basis of possible constraints B that rejects e. Inspired from a technique used in QuickXplain [Junker, 2004b], `FindScope` requires a number of queries logarithmic in $|S_2|$ and linear in the size of the final scope returned. The function `FindC` takes as parameter the negative example e and the scope returned by `FindScope`. It returns a constraint from C_T with the given scope that rejects e. For any assignment e, $\kappa_B(e)$ denotes the set of all constraints in B rejecting e.

g-QuAcq has a structure very similar to QuAcq. It initializes the set NonTarget and the network C_L it will learn to the empty set (line 2). If C_L is unsatisfiable (line 6), the space of possible networks collapses because there does not exist any subset of the given basis B that is able to correctly classify the examples already asked of the user. In line 8, QuAcq computes a complete assignment e satisfying C_L but

Algorithm 5: G-QuAcq =QuAcq +GenAcq

```
2  C_L ← ∅, NonTarget ← ∅;
4  while true do
6      if sol(C_L) = ∅ then return"collapse";
8      choose e in D^X accepted by C_L and rejected by B
10     if e = nil then return "convergence on C_L";
12     if Ask(e) = yes then
14         B ← B \ κ_B(e)
16         NonTarget ← NonTarget ∪ κ_B(e)
18     else
20         c ← FindC(e, FindScope(e, ∅, X, false))
22         if c = nil then return "collapse";
24         else
26             G ← GenAcq(c, NonTarget)
28             foreach s ∈ G do
30                 C_L ← C_L ∪ {(var, rel(c)) | var ∈ s}
```

violating at least one constraint from B. If such an example does not exist (line 10), then all constraints in B are implied by C_L, and we have converged. If we have not converged, we propose the example e to the user, who will answer by *yes* or *no* (line 12). If the answer is *yes*, we can remove from B the set $\kappa_B(e)$ of all constraints in B that reject e (line 14) and we add all these ruled out constraints to the set NonTarget to be used in GenAcq (line 16). If the answer is *no*, we are sure that e violates at least one constraint of the target network C_T. We then call the function `FindScope` to discover the scope of one of these violated constraints. `FindC` will select which one with the given scope is violated by e (line 20). If no constraint is returned (line 22), this is again a condition for collapsing as we could not find in B a constraint rejecting one of the negative examples. Otherwise, we know that the constraint c returned by `FindC` belongs to the target network C_T. This is here that the algorithm differs from QuAcq as we call GenAcq to find all the maximal sequences of types covering $var(c)$ on which $rel(c)$ holds. They are returned in G (line 26). Then, for every sequence of variables var belonging to one of by these sequences in G, we add the constraint $(var, rel(c))$ to the learned network C_L (line 28).

3.3 Strategies

GenAcq learns the maximal sequences of types on which a constraint can be generalized. The order in which sequences are picked from `Table` in line 19 of Algorithm 4 is not specified by the algorithm. As shown on the following example, different orderings can lead more or less quickly to the good (maximal) sequences

on which a relation r holds. Let us come back to our example on the Zebra problem (Section 3.2.3). In the way we developped the example, we needed only 3 generalization queries to empty the set $Table$ and converge on the maximal sequence $(color, color)$ on which $\neq$ holds:

1. $AskGen((X, x_5), \neq) = no$
2. $AskGen((color, color), \neq) = yes$
3. $AskGen((x_2, X), \neq) = no$

Using another ordering, GenAcq needs 8 generalization queries:

1. $AskGen((X, X), \neq) = no$
2. $AskGen((X, color), \neq) = no$
3. $AskGen((color, X), \neq) = no$
4. $AskGen((X, x_5), \neq) = no$
5. $AskGen((x_2, X), \neq) = no$
6. $AskGen((x_2, color), \neq) = yes$
7. $AskGen((color, x_5), \neq) = yes$
8. $AskGen((color, color), \neq) = yes$

If we want to reduce the number of generalization queries, we may wonder which strategy to use. In this section we propose two techniques. The first idea is to pick sequences in the set $Table$ following an order given by a heuristic that will try to minimize the number of queries. The second idea is to put a cutoff on the number consecutive negative queries we accept to face, leading to a non complete generalization startegy: the output of GenAcq will no longer be guaranteed to be the *maximal* sequences.

3.3.1 Query Selection Heuristics

We propose some query selection heuristics to decide which sequence to pick next from $Table$. We first propose *optimistic* heuristics, which try to take the best from positive answers:

— **max_CST:** This heuristic selects a sequence s maximizing the number of possible constraints (var, r) in the basis such that var is in s and r is the relation we try to generalize. The intuition is that if the user answers *yes*, the generalization will be maximal in terms of number of constraints.

— **max_VAR:** This heuristic selects a sequence s involving a maximum number of variables, that is, maximizing $|\bigcup_{T \in s} T|$. The intuition is that if the user answers *yes*, the generalization will involve many variables.

Dually, we propose *pessimistic* heuristics, which try to take the best from negative answers:

— **min_CST:** This heuristic selects a sequence s minimizing the number of possible constraints (var, r) in the basis such that var is in s and r is the relation we try to generalize. The intuition is to maximize the chances to receive a *yes* answer. If, despite this, the user answers *no*, a great number of sequences are removed from $Table$ (see Lemma 3.1).

— **min_VAR:** This heuristic selects a sequence s involving a minimum number of variables, that is, minimizing $|\bigcup_{T \in s} T|$. The intuition is to maximize the chances of a *yes* answer while focusing on smaller sets of variables than `min_CST`. Again, a *no* answer leads to a great number of sequences removed from Table.

As a baseline for comparison, we define a random selector.

— **random:** It picks randomly a sequence s in Table.

3.3.2 Using Cutoffs

The idea here is to exit GenAcq before having proved the maximality of the sequences returned. We put a threshold cutoffNo on the number of consecutive negative answers to avoid using queries to check unpromising sequences. The hope is that GenAcq will return near-maximal sequences of types despite not proving maximality. This cutoff strategy is implemented by setting the variable cutoffNo to a predefined value. In lines 27 and 34 of GenAcq, a counter of consecutive negative answers is respectively reset and incremented depending on the answer from the user. In line 17, that counter is compared to cutoffNo to decide to exit or not.

3.4 Experimentations

We made some experiments to evaluate the impact of using our generalization functionality GenAcq in the QuAcq constraint acquisition system. We implemented GenAcq and plugged it in QuAcq, leading to the G-QuAcq version. We first present the benchmark problems we used for our experiments. Then, we report the results of several experiments. The first one compares the performance of G-QuAcq to the basic QuAcq. The second reports experiments evaluating the different strategies we proposed (query selection heuristics and cutoffs) on G-QuAcq. The third evaluates the performance of our generalization approach when our knowledge of the types of variables is incomplete.

3.4.1 Benchmark Problems

Zebra problem. As introduced in section 3.2.3, the Lewis Carroll's Zebra problem is formulated using 5 types of 5 variables each, with 5 cliques of $\neq$ constraints and 14 additional constraints given in the description of the problem. We fed QuAcq and G-QuAcq with a basis B of 4450 unary and binary constraints taken from a language with 24 basic arithmetic and distance constraints.
Sudoku. The Sudoku model is expressed using 81 variables with domains of size 9, and 810 $\neq$ binary constraints on rows, columns and squares. In this problem, the types are the 9 rows, 9 columns and 9 squares, of 9 variables each. We fed QuAcq and G-QuAcq with a basis B of 6480 binary constraints from the language $\Gamma = \{=, \neq\}$.
Latin Square. The Latin square problem consists of an $n \times n$ table in which each element occurs once in every row and column. For this problem, we use 25 variables

with domains of size 5 and 100 binary $\neq$ constraints on rows and columns. Rows and columns are the types of variables (10 types). We fed QuAcQ and G-QuAcQ with a basis of constraints based on the language $\Gamma = \{=, \neq\}$.

Radio Link Frequency Assignment Problem. The RLFAP problem is to provide communication channels from limited spectral resources [Cabon et al., 1999]. Here we build a simplified version of RLFAP that consists in distributing all the frequencies available on the base stations of the network. The constraint model has 25 variables with domains of size 25 and 125 binary constraints. We have five stations of five terminals (transmitters/receivers), which form five types. We fed QuAcQ and G-QuAcQ with a basis of 1800 binary constraints taken from a language of 6 arithmetic and distance constraints

Purdey. Like Zebra, this problem has a single solution. Four families have stopped by Purdey's general store, each to buy a different item and paying differently. Under a set of additional constraints given in the description, the problem is how can we match family with the item they bought and how they paid for it. The target network of Purdey has 12 variables with domains of size 4 and 30 binary constraints. Here we have three types of variables, which are *family*, *bought* and *paid*, each of them contains four variables.

3.4.2 Results

For all our experiments we report, the total number #Ask of standard queries asked by the basic QuAcQ, the total number #AskGen of generalization queries, and the numbers #no and #yes of negative and positive generalization queries, respectively, where #AskGen = #no+#yes. The time overhead of using G-QuAcQ rather than QuAcQ is not reported. Computing a generalization query takes a few milliseconds.

Our first experiment compares QuAcQ and G-QuAcQ in its baseline version, G-QuAcQ +rand, on our benchmark problems. Table 3.1 reports the results. We observe that the number of queries asked by G-QuAcQ is dramatically reduced compared to QuAcQ. This is especially true on problems with many types involving many variables, such as Sudoku or Latin square. G-QuAcQ acquires the Sudoku with 260 standard queries plus 166 generalization queries, when QuAcQ acquires it in 8645 standard queries.

Let us now focus on the behavior of our different heuristics in G-QuAcQ. Table 3.2(top) reports the results obtained with G-QuAcQ using `min_VAR`, `min_CST`, `max_VAR`, and `max_CST` to acquire the Sudoku model. (Other problems showed similar trends.) The results clearly show that `max_VAR`, and `max_CST` are very bad heuristics. They are worse than the baseline `random`. On the contrary, `min_VAR` and `min_CST` significantly outperform `random`. They respectively require 90 and 132 generalization queries instead of 166 for `random`. Notice that they all ask the same number of standard queries (260) as they all find the same maximal sets of sequences for each learned constraint.

Table 3.1 – QuAcq vs g-QuAcq.

	QuAcq	g-QuAcq +random	
	#Ask	#Ask	#AskGen
Zebra	638	257	67
Sudoku	8645	260	166
Latin square	1129	117	60
RFLAP	1653	151	37
Purdey	173	82	31

Table 3.2 – g-QuAcq with heuristics and cutoff strategy on Sudoku.

	cutoff	#Ask	#AskGen	#yes	#no
random	$+\infty$	260	166	42	124
min_VAR			90	21	69
min_CST			132	63	69
max_VAR			263	63	200
max_CST			247	21	226
min_VAR	3	260	75	21	54
	2		57	21	36
	1		**39**	21	18
min_CST	3	626	238	112	126
	2	679	231	132	99
	1	837	213	153	60

At the bottom of Table 3.2 we compare the behavior of our two best heuristics (min_VAR and min_CST) when combined with the cutoff strategy. We tried all values of the cutoff from 1 to 3. A first observation is that min_VAR remains the best whatever the value of the cutoff is. Interestingly, even with a cutoff equal to 1, min_VAR requires the same number of standard queries as the versions of g-QuAcq without cutoff. This means that using min_VAR as selection heuristic in Table, g-QuAcq is able to return the maximal sequences despite being stopped after the first negative generalization answer. We also observe that the number of generalization queries with min_VAR decreases when the cutoff becomes smaller (from 90 to 39 when the cutoff goes from $+\infty$ to 1). By looking at the last two columns we see that this is the number #no of negative answers which decreases. The good performance of min_VAR + cutoff=1 can thus be explained by the fact that min_VAR selects first queries that cover a minimum number of variables, which increases the chances to have a *yes* answer. Finally, we observe that the heuristic min_CST does not have the same nice characteristics as min_VAR. The smaller the cutoff, the more standard queries are needed, not compensating for the saving in number of generalization queries (from 260 to 837 standard queries for min_CST when the cutoff

Table 3.3 – G-QuAcq with `random`, `min_VAR`, and cutoff=1 on Zebra, Latin square, RLFAP, and Purdey.

	#Ask	#AskGen	#yes	#no
Zebra				
Random	257	67	10	57
min_VAR		48	5	43
min_VAR +cutoff=1		**23**	5	18
Latin square				
Random	117	60	16	44
min_VAR		34	10	24
min_VAR +cutoff=1		**20**	10	10
RLFAP				
Random	151	37	16	21
min_VAR		41	14	27
min_VAR +cutoff=1		**22**	14	8
Purdey				
Random	82	31	5	26
min_VAR		24	3	21
min_VAR +cutoff=1		**12**	3	9

goes from $+\infty$ to 1). This means that with `min_CST`, when the cutoff becomes too small, GenAcq does not return the maximal sequences of types where the learned constraint holds.

In Table 3.3, we report the performance of G-QuAcq with `random`, `min_VAR` and `min_VAR` +cutoff=1 on all the other problems. We see that `min_VAR` +cutoff=1 significantly improve the performance of G-QuAcq on all problems. As in the case of Sudoku, we observe that `min_VAR` +cutoff=1 does not lead to an increase in the number of standard queries. This means that on all these problems `min_VAR` +cutoff=1 always returns the maximal sequences while asking less generalization queries with negative answers.

From these experiments we see that G-QuAcq with `min_VAR` +cutoff=1 leads to tremendous savings in number of queries compared to QuAcq: 257+23 instead of 638 on Zebra, 260+39 instead of 8645 on Sudoku, 117+20 instead of 1129 on Latin square, 151+22 instead of 1653 on RLFAP, 82+12 instead of 173 on Purdey.

In our last experiment, we show the effect on the performance of G-QuAcq of a lack of knowledge on some variable types. We took again our 5 benchmark problems in which we have varied the amount of types known by the algorithm. This simulates a situation where the user does not know that some variables are from the same type. For instance, in Sudoku, the user could not have noticed that variables are arranged in columns. Figure 3.2 shows the number of standard queries and generalization queries asked by G-QuAcq with `min_VAR` +cutoff=1 to learn the

RLFAP model when fed with an increasingly more accurate knowledge of types. We observe that as soon as a small percentage of types is known (20%), G-QUACQ reduces drastically its number of queries. Table 3.4 gives the same information for all other problems.

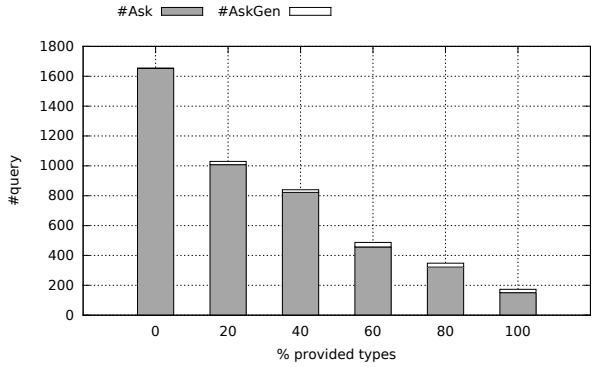

Figure 3.2 – G-QUACQ on RLFAP when the percentage of provided types increases.

Table 3.4 – G-QUACQ when the percentage of provided types increases.

	% of types	#Ask	#AskGen
Zebra	0	638	0
	20	619	12
	40	529	20
	60	417	27
	80	332	40
	100	257	48
Sudoku 9×9	0	8645	0
	33	3583	232
	66	610	60
	100	260	39
Latin Square	0	1129	0
	50	469	49
	100	117	20
Purdey	0	173	0
	33	111	8
	66	100	10
	100	82	12

3.5 Conclusion

We have proposed a new technique to make constraint acquisition more efficient by using information on the types of components the variables in the problem represent. We have introduced generalization queries, a new kind of query asked to the user to generalize a constraint to other scopes of variables of the same type where this constraint possibly applies. Our new technique, GenAcQ, can be called to generalize each new constraint that is learned by any constraint acquisition system. We have proposed several heuristics and strategies to select the good candidate generalization query. We have plugged GenAcQ into the QuAcQ constraint acquisition system, leading to the G-QuAcQ algorithm. We have experimentally evaluated the benefit of our approach on several benchmark problems, with and without complete knowledge on the types of variables. The results show that G-QuAcQ dramatically improves the basic QuAcQ algorithm in terms of number of queries.

The drawback of generalization queries is that they require types of variables of the learned constraint to be generated. To overcome this weakness, and in order to make the build of such queries totally independent of the user, we propose in the next chapter to learn types of variables during the constraint acquisition process, and then to use the learned types to generate generalization queries.

CHAPTER

4 Detecting Types of Variables for Constraint Generalization

Preamble

In this chapter, we propose a new algorithm that is able to learn types during the constraint acquisition process. The idea is to infer potential types by analyzing the structure of the current constraint network and to use the extracted types to ask generalization queries. Our approach gives good results although no knowledge on the types is provided.

Contents

4.1 Introduction

Constraint programming (CP) is a paradigm that allows effective solving of combinatorial problems in many areas, such as planning and scheduling. However, modeling a combinatorial problem using constraints requires significant expertise in CP [Freuder, 1999].

To alleviate this issue, several constraint acquisition systems have been introduced [Bessiere et al., 2005, 2007; Beldiceanu and Simonis, 2012; Lallouet et al., 2010; Bessiere et al., 2014b; Shchekotykhin and Friedrich, 2009; Bessiere et al., 2013]. Most of these systems interact with the user by asking her to classify simple

examples. The drawback is that when the problem has a lot of constraints, the user may have to classify a large number of such questions to learn them all.

To make constraint acquisition systems more efficient in practice, an opportunistic kind of query that uses the structure of the problem has been introduced in [Bessiere et al., 2014a] with the GenAcq algorithm. This kind of query, named "generalization query", is based on an aggregation of variables into types. The user provides the variable types and based on these types, generalization queries ask the user whether or not a learned constraint can be generalized to other scopes of variables of the same type as those on the learned constraint. By using such queries over existing constraint acquisition systems, the number of queries needed to converge on the target constraint network can be significantly reduced.

Nevertheless, the aggregation of variables into types may not always be a straightforward task for the user especially when the problem under consideration has a hidden structure. In this chapter, we propose to learn the potential types during the constraints acquisition process. The idea is to analyze the structure of the partial constraint network learned so far in order to detect potential types and to build generalization queries.

Indeed, when one looks more closely at the constraint network of a given problem, the variables of the same type are often tightly connected with similar constraints whereas the variables of different types are connected in a weaker way. To illustrate this point, let us consider the well known Lewis Carroll's Zebra problem. The constraint network of this problem is usually formulated using 25 variables, partitioned into 5 types of 5 variables each. The types are color, nationality, drink, cigaret and pet. There is a clique of $\neq$ constraints on all pairs of variables of the same type and 14 additional constraints, among which 3 are unary, given in the description of the problem. Figure 4.1 shows the constraint network of the Zebra problem. In this example, it is clear that types have dense internal links but there are only a lower density of external links between different types.

The idea of detecting tightly connected sub-graphs arose in the study of networks such as social networks [Wasserman and Faust, 1994] and biochemical networks [Ito et al., 2001]. An important characteristic that commonly occurs in such networks is *community* structure. Informally, a network or a graph is said to have community structure, if the nodes of the network can be easily grouped into (potentially overlapping) sets of nodes such that the groups have more internal edges than outgoing edges. Each such group is called a community.

Given the similarity between the structure of a type and that of a community, we propose in this chapter to detect potential types by finding communities in the constraint network during the constraint acquisition process. Several methods for community finding have been proposed in the literature. We have considered three different techniques in this chapter. The first one is based on the concept of modularity [Newman and Girvan, 2004] which provides information on the strength of division of a network into communities. Networks with high modularity have dense connections between the nodes within communities but sparse connections between nodes in different communities. The second technique exploits the notion of edge

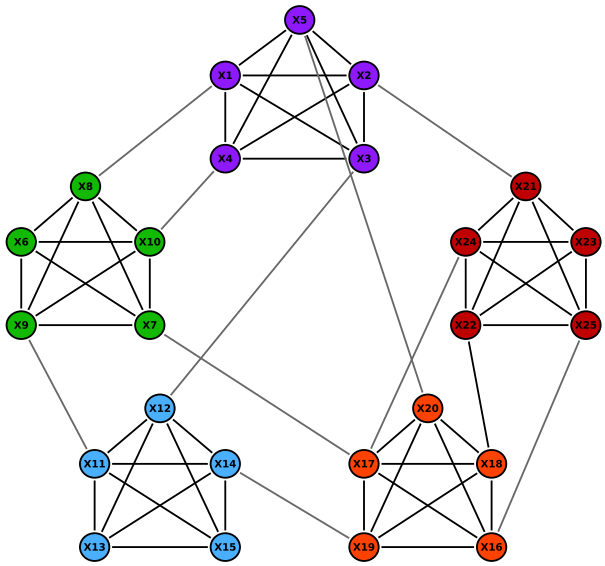

Figure 4.1 – The constraint graph of the Zebra problem.

betweenness [Girvan and Newman, 2002], which is a measure that assigns a score to each edge. The edges that lie “between" many pairs of nodes have high scores which enables their easier identification. Removing these edges will leave behind just the communities themselves. The third technique is more straightforward. It is based on the assumption that the variables of the same type will tend to form quasi-cliques during the constraint acquisition process. That is, this technique finds sub-graphs with an edge density exceeding a threshold parameter.

In this chapter we propose an algorithm, named MINE&ASK, that makes use of the extracted potential types to ask the user to classify generalization queries. We plugged MINE&ASK into the QUACQ constraint acquisition system, to obtain the boosted version M-QUACQ algorithm. We experimentally evaluate the benefit of our technique on several benchmark problems. The results show that M-QUACQ improves the basic QUACQ algorithm in terms of number of queries although no knowledge on the types is provided.

The outline of this chapter is as follows. The algorithm MINE&ASK is presented in Section 4.2. We illustrate the idea behind our approach with an example in section 4.3. Section 4.4 gives more details on the techniques used to extract potential types. The experimental results we obtained when comparing MINE&ASK with the different techniques to the basic QUACQ are given in section 4.5. Section 4.6 concludes the chapter.

4.2 MINE&ASK Algorithm

In this section we present the MINE&ASK algorithm. The idea behind this algorithm is to mine the partial graph of the current constraint network in order to get potential types and then ask the user to classify generalization queries.

4.2.1 Description of MINE&ASK

Algorithm 6: MINE&ASK

Input: C: a set of constraints, r: a relation, $mine \in \{\texttt{modularity}, \texttt{betweenness}, \gamma\texttt{-clique}\}$: a mine strategy, GQ_{max}: the maximum number of generalization queries
Output: L: a set of learned constraints

1 $L \leftarrow \varnothing$; $\#GQ \leftarrow 0$
2 $X' \leftarrow \bigcup var(c)$ s.t. $c \in C \wedge rel(c) = r$
3 $G_C \leftarrow G(X', E)$, $E = \{\{x,y\} \mid x,y \in var(c) \wedge x \neq y \wedge c \in C \wedge rel(c) = r\}$
4 $Table \leftarrow \{Y \mid Y \in component(G_C) \wedge \neg isClique(G_C(Y))\}$
5 **while** $Table \neq \varnothing \wedge \#GQ \leq GQ_{max}$ **do**
6 pick Y in $Table$
7 $generalized \leftarrow false$
8 **if** $(\nexists (Y', r') \in NegativeQ \mid Y' \subseteq Y \wedge r \subseteq r') \wedge (\nexists var \in Y^{|r|} \mid (var, r) \notin B)$ **then**
9 **if** $Sol(C_L \cup \{(var, r) \mid var \in Y^{|r|}\}) \neq \varnothing$ and $AskGen(Y, r) = Yes$ **then**
10 $L \leftarrow L \cup \{(var, r) \mid var \in Y^{|r|}\}$
11 $generalized \leftarrow true$
12 **else** $NegativeQ \leftarrow NegativeQ \cup \{(Y, r)\}$;
13 $\#GQ{+}{+}$
14 **if** $\neg generalized$ **then**
15 $Table \leftarrow Table \cup mine(G(Y))$
16 **return** L;

The algorithm MINE&ASK takes as argument the set of constraints C learned so far, a relation r, the operator *mine* that corresponds to the strategy used for extracting potential types and the maximum number of generalization queries GQ_{max} that we are allowed to ask. The algorithm uses a global data structure *NegativeQ*, which is a set of pairs (Y, r) for which we know that r does not hold. MINE&ASK also uses the local data structure *Table* which contains all potential types that are candidates for generalization.

MINE&ASK starts by initializing L to the empty set and the number of generalization queries #GQ to zero (line 1). The set L will contain the output of MINE&ASK, that is all learned constraints. In line 3, we build the constraint graph $G(X', E)$, noted G_C and restricted to the relation r. This step is important because it helps to

reveal the structure of the network. To illustrate what this means, let us consider the example given in Figure 4.2(a). This problem consists of 12 variables and 66 binary constraints which means that the constraint network of this problem is a complete graph (i.e. clique). Two relations are used in the constraints namely $\neq$ and $\geq$. Although the constraint network is a clique, no learned constraint can be generalized to the other scopes in this clique as the links connecting the variables of the clique come from different relations. On the contrary, as shown in Figure 4.2(b), when we focus on the only $\neq$ relation, useful types can be easily detected.

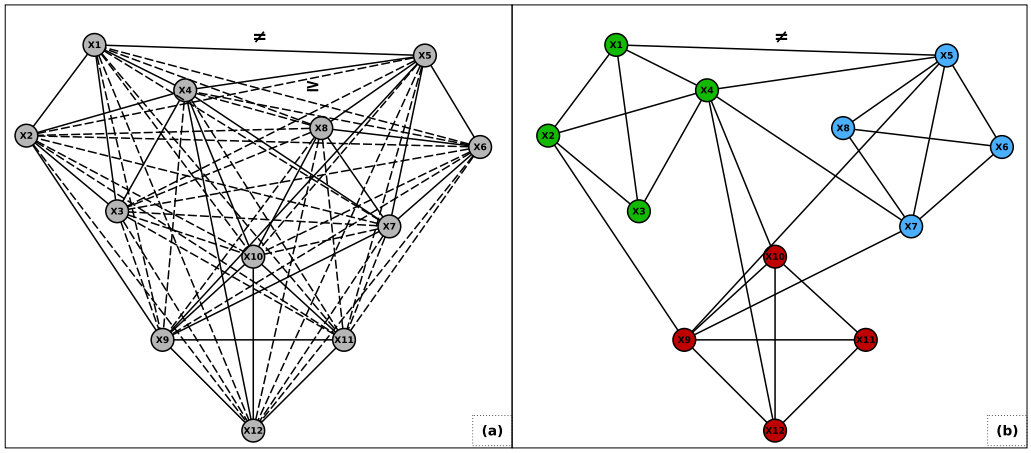

Figure 4.2 – A constraint network with a hidden structure

Afterwards, we put in $\mathtt{Table}$ all connected components of G whilst excluding the already formed cliques (line 4). At each iteration, MINE&ASK picks a potential type Y from this table (line 6) and asks a generalization query on (Y,r) (line 9) if the answer cannot be deduced. That is, the answer to an $\mathtt{AskGen}(Y,r)$ is negative if the user already classifies as negative a query on a sub-type of Y (line 8). A negative answer can also be deduced in the case of the unsatisfiability of the resulting network where in such case the answer cannot be positive (line 9). Such satisfiability tests allow us to avoid asking unnecessary queries. For instance, trying to generalize a non-commutative relation to a clique, or asking if a clique of difference applies on a set of 4 variables with the same domain of size 3.

Now, if the answer of the user to $\mathtt{AskGen}(Y,r)$ is *yes*, this means that r holds on the type Y. Thus, we add all inferred constraints on Y to the set L (line 10). In the case of a negative answer, we add (Y,r) to $\mathtt{NegativeQ}$ with intent to avoid asking redundant queries afterwards. When no generalization happened on (Y,r) (line 15), this means that Y is not a type on which r can be generalized. Here, we call the operator $\mathtt{mine}$ to extract potential types from the subgraph $G(Y)$. The resulting potential types are added to $\mathtt{Table}$ to be taken into account later. The main loop (line

5) terminates when all potential types in Table are processed, or when the number of generalization queries exceeds a given threshold GQ_{max}.

4.2.2 M-QUACQ Algorithm

MINE&ASK is a generic technique that can be plugged into any constraint acquisition system. In this section we present M-QUACQ (Algorithm 7) where we plugged MINE&ASK into the QUACQ system [Bessiere et al., 2013].

M-QUACQ initializes the constraint network C_L to the empty set (line 1). When C_L is unsatisfiable (line 3), the space of possible networks collapses because there does not exist any subset of the given basis B that is able to correctly classify the examples already asked to the user. In line 4, M-QUACQ computes a complete assignment e satisfying C_L and violating at least one constraint from B. If such an example does not exist (line 5), then all constraints in B are implied by C_L, and the algorithm has converged. Otherwise, we propose the example e to the user, who will answer by *yes* or *no* (line 6). If the answer is *yes*, we can remove from B the set $\kappa_B(e)$ of all constraints in B that reject e (line 7). If the answer is *no*, we are sure that e violates at least one constraint of the target network C_T. We then call the function `FindScope` to discover the scope of one of these violated constraints. Here, `FindScope` acts in a dichotomous manner and asks a number of queries logarithmic in the size of the example. Afterwards, `FindC` will select which constraint with the given scope is violated by e (line 9). If no constraint is returned (line 10), this is a condition for collapsing as we could not find in B a constraint rejecting one of the negative examples. Otherwise, we know that the constraint c returned by `FindC` belongs to the target network C_T, then we add it to the learned network C_L (line 12). Note that `FindScope` and `FindC` functions are used exactly as they appear in [Bessiere et al., 2013]. Afterwards, we call MINE&ASK to mine the learned constraint network C_L in order to extract potential types and to ask generalization queries. M-QUACQ updates C_L by adding all learned constraints (line 13).

4.3 An Illustrative Example

In this section we illustrate on an example the idea of extracting potential types during the constraint acquisition process. Let us consider the example given in Figure 4.3. The **part (a)** of Figure 4.3 shows the constraint network of the problem that the user has in mind. This problem consists in 15 variables and 39 binary constraints. Two relations are used, noted r_1 and r_2 in Figure 4.3. The **part (b)** of Figure 4.3 shows the constraint network learned, at a given point, using QUACQ system. Suppose that the last constraint learned using QUACQ was $((x_1,x_2),r_1)$. Now, we want to extract potential types on which the relation r_1 can be generalized. To this end, MINE&ASK restricts the constraint network to the constraints that use r_1 (**part (c)** in Figure 4.3). Suppose now that MINE&ASK algorithm finds three potential types $T_1 = \{x_1,\dots x_5\}$, $T_2 = \{x_6,\dots x_{10}\}$ and $T_3 = \{x_{11},\dots x_{15}\}$. According to what the user has in mind (**part (a)** of Figure 4.3), a generalization query on T_3 will be

Algorithm 7: M-QuAcq = QuAcq + Mine&Ask

Input: $mine \in \{\texttt{modularity}, \texttt{betweenness}, \gamma\texttt{-clique}\}$: a mine strategy, GQ_{max}: the maximum number of generalization queries
Output: C_L: a set of learned constraints

1 $C_L \leftarrow \varnothing$;
2 **while** *true* **do**
3 **if** $sol(C_L) = \varnothing$ **then return** "collapse";
4 choose e in D^X accepted by C_L and rejected by B
5 **if** $e = nil$ **then return** "convergence on C_L";
6 **if** $Ask(e) = yes$ **then**
7 $B \leftarrow B \setminus \kappa_B(e)$;
8 **else**
9 $c \leftarrow \texttt{FindC}(e, \texttt{FindScope}(e, \varnothing, X, \textbf{false}))$;
10 **if** $c = nil$ **then return** "collapse";
11 **else**
12 $C_L \leftarrow C_L \cup \{c\}$;
13 $C_L \leftarrow C_L \cup$ Mine&Ask$(C_L, mine, rel(c), GQ_{max})$;
14 **return** C_L;

classified as negative whereas the ones on T_1 and T_2 will be classified as positive. Nine constraints will be, in one shot, added to the current constraint network (see **part (d)** of Figure 4.3).

4.4 Extraction of Potential Types

Mine&Ask extracts variable types by finding communities in the current graph of learned constraints. The way in which the operator $mine$ finds communities at line 15 of Algorithm 6 is described in this section.

4.4.1 Optimizing Modularity

One of the most effective approaches for detecting communities in networks is based on the optimization of the measure known as modularity [Newman and Girvan, 2004]. Given a partition of vertices of a network into disjoint communities, modularity reflects the concentration of edges within communities compared with random distribution of links between all nodes regardless of communities.

More formally, let $G = (X, E)$ be a graph, with $X = \{x_1, \ldots, x_n\}$ the set of vertices and let A be the adjacency matrix of G. That is, $A_{ij} = 1$ if there exists an edge between vertices x_i and x_j and $A_{ij} = 0$ otherwise. Suppose the vertices are divided into communities such that vertex x_i belongs to community $c(x_i)$ and let $deg(x_i)$ denotes the degree of x_i. Then the modularity Q is given by the following formula:

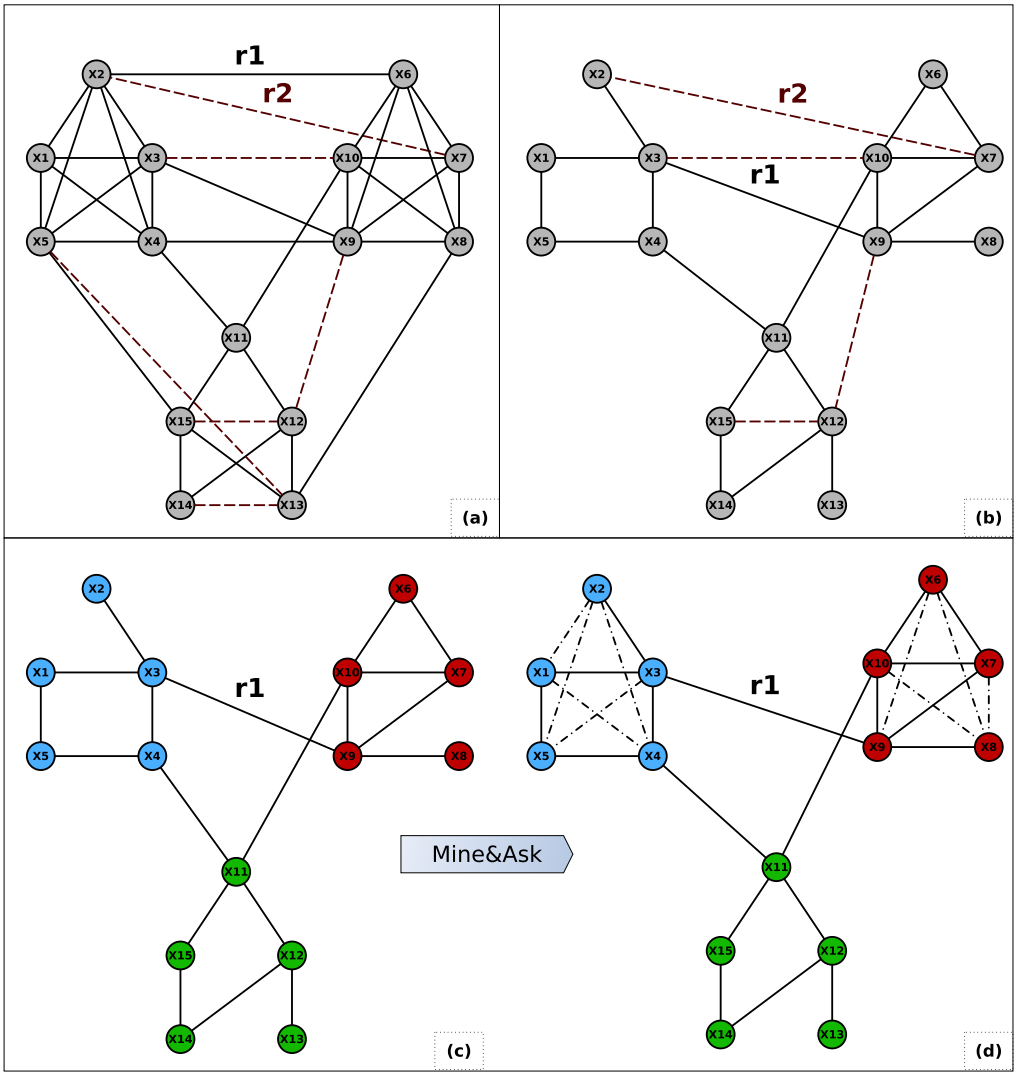

Figure 4.3 – Illustrative Example

$$Q = \sum_{i,j} [\frac{A_{ij}}{2m} - \frac{deg(x_i) \times deg(x_j)}{4m^2}] \delta(c(x_i), c(x_j))$$

where m is the total number of edges in the network, and $\delta(c(x_i), c(x_j)) = 1$ if x_i and x_j belong to the same community (i.e., $c(x_i) = c(x_j)$) and 0 otherwise.

High values of the modularity correspond to good partitions of a network into communities [Newman and Girvan, 2004]. Hence one should be able to find such good partitions by searching through the possible candidates for ones with high

modularity. Unfortunately, finding the global maximum modularity over all possible divisions is NP-hard, but reasonably good solutions can be found with approximate optimization techniques. In this chapter, we have used the algorithm introduced in [Aaron et al., 2004] and implemented in the `igraph` software package [Csardi and Nepusz, 2006]. This algorithm uses a greedy optimization where, starting with a partition where each vertex is the unique member of a community, it repeatedly joins together the two communities whose fusion produces the largest increase in Q.

4.4.2 Edge Betweenness Centrality

Recently, the concept of edge betweenness was introduced [Girvan and Newman, 2002] as a measure that provides information on edges centrality in networks. This measure can be implemented in several ways but the most common way is the one based on shortest paths. Formally speaking, let x_i and x_j be two nodes in the network. Let σ_{ij} denotes the number of shortest paths between nodes x_i and x_j and $\sigma_{ij}(e)$ denotes the number of shortest paths between x_i and x_j which go through the edge e. The Betweenness centrality of e, denoted by $B(e)$, is defined as follows:

$$B(e) = \sum_{ij} \frac{\sigma_{ij}(e)}{\sigma_{ij}}$$

If two communities are joined by only a few inter-community edges, then all paths through the network from vertices of one community to vertices of the other must pass through one of those few edges. Thus, the edge betweenness scores for inter-community edges are expected to be larger than the ones for intra-community edges. The betweeness based algorithm to find community structure is used as it appears in [Girvan and Newman, 2002]. The idea behind this algorithm is to iteratively calculate the betweenness score for each edge and to remove the one with the highest score. That is, removing edges with high betweenness scores allows us to isolate the communities. This algorithm is also available in the `igraph` software package [Csardi and Nepusz, 2006].

4.4.3 Quasi-cliques Detection

Mining the constraint network for dense subgraphs may be a possible way for discovering communities. Cliques are the densest form of subgraphs. A graph is a clique if there is an edge between every pair of the vertices. This requirement is not desirable in our case because the idea is to anticipate the formation of complete cliques in order to be able to infer some constraints. Therefore, instead of mining cliques, our goal is to extract $\gamma-cliques$ (i.e. quasi-cliques), which are sub-graphs with an edge density exceeding a given threshold parameter $\gamma \in [0,1]$.

Definition 4.1. *($\gamma-$clique) Let $G=(X,E)$ be a graph with X the set of vertices, E the set of edges, and a parameter $\gamma \in [0,1]$. A $\gamma-$clique is a subset of vertices $K \subseteq X$ such*

that the induced subgraph $G(K)$ *is a connected component and* $|E \cap K \times K| \geq \gamma \frac{q(q-1)}{2}$*, with* $q = |K|$.

Algorithm 8: `FindQCliques` (G, A, B, K, γ)

```
1  if A = ∅ then
2      report K as a quasi-clique;
3  while A ≠ ∅ do
4      choose x ∈ A;
5      K' ← K ∪ {x};
6      A' ← {y | y ∈ X \ K' ∧ K' ∪ {y} is a γ−clique};
7      A' ← A' \ A' ∩ B;
8      FindQCliques(G, A', B, K', γ) ;
9      A ← A \ {x};
10     B ← B ∪ {x};
```

We propose an incomplete recursive algorithm (Algorithm 8) for finding $\gamma-cliques$ in an undirected graph G. This algorithm is an adaptation of the basic form of the well-known Bron-Kerbosch's algorithm [Bron and Kerbosch, 1973] for finding maximal cliques in a graph. Algorithm 8 is based on the recursive function `FindQCliques` that takes as arguments an undirected graph G, a set A of candidates, a set B of vertices to exclude from consideration to avoid generating the same quasi-clique several times, the quasi-clique K being constructed and a parameter $\gamma \in]0,1[$ which specifies the minimum edge density of quasi-cliques. The recursion is initiated by setting B and K to be the empty set and A to be the vertex set of the graph. Each time a new element is added to the current quasi-clique (line 5), we calculate a new set A' of candidates. A vertex y is an element of A' if and only if when it is added to the current quasi-clique we obtain a new quasi-clique (line 6). This condition is not a necessary condition to lead to a new quasi-clique. Consequently, Algorithm 8 is incomplete. When the candidate set becomes empty (line 1), a new quasi-clique is reported and a backtrack to the last choice is performed. Then, the last choice is added to the set B to exclude it from consideration in future quasi-cliques.

4.5 Experimentations

We performed some experiments to evaluate the impact of using MINE&ASK in constraint acquisition. We implemented MINE&ASK and plugged it in QUACQ system, leading to the M-QUACQ version. We first present the benchmark problems we used for our experiments. Then, we report the results of acquiring these problems with the basic version of QUACQ [Bessiere et al., 2013], our version M-QUACQ and G-QUACQ version. The G-QUACQ version includes the generalization process with

GenAcq algorithm and the user provides all variable types [Bessiere et al., 2014a]. The experiments evaluate also the different ways in which our approach extracts potential types, namely the `modularity`, the `betweenness` and the γ-`clique`. Our tests were conducted on an Intel Core i5-3320M CPU @ 2.60GHz × 4 with 4 Gb of RAM.

4.5.1 Benchmark Problems

Zebra problem. The Lewis Carroll Zebra problem is formulated using 5 types of 5 variables each, with 5 cliques of $\neq$ constraints and 14 additional constraints given in the description of the problem. We fed QuAcq, G-QuAcq and M-QuAcq with a basis B of 4450 unary and binary constraints taken from a language with 24 basic arithmetic and distance constraints.
Latin Square. The Latin square problem consists of an $n \times n$ table in which each element occurs once in every row and column. For this problem, we use 36 variables with domains of size 6 and 180 binary $\neq$ constraints on rows and columns. Rows and columns are the types of variables (10 types). We fed QuAcq, G-QuAcq and M-QuAcq with a basis B of 1260 constraints based on the language $\Gamma = \{=, \neq\}$.
Purdey. Like Zebra, this problem has a single solution. Four families have stopped by Purdey's general store, each to buy a different item and paying differently. Under a set of additional constraints given in the description, the problem is how can we match family with the item they bought and how they paid for it. The target network of Purdey has 12 variables with domains of size 4 and 30 binary constraints. Here we have three types of variables, which are *family*, *bought* and *paid*, each of them contains four variables. We fed QuAcq, G-QuAcq and M-QuAcq with a basis B of 396 constraints based on the language $\Gamma = \{=, \neq\}$.
PlaceNumPuzzle. The PlaceNumPuzzle problem is to place numbers 1 through N on nodes of a given graph such that each number appears exactly once and no connected nodes have consecutive numbers. For this problem, we use 25 variables with domains of size 25 and 64 binary constraints. The problem has three types which are the cliques of the graph. We fed QuAcq, G-QuAcq and M-QuAcq with a basis B of 1260 binary constraints taken from a language of 4 arithmetic and distance constraints.
Murder. Someone was murdered last night, and you are summoned to investigate the murder. The objects found on the spot that do not belong to the victim include: a pistol, an umbrella, a cigarette, a diary, and a threatening letter. There are also witnesses who testify that someone had argued with the victim, someone left the house, someone rang the victim, and some walked past the house several times about the time the murder occurred. The suspects are: Miss Linda Ablaze, Mr. Tom Burner, Ms. Lana Curious, Mrs. Suzie Dulles, and Mr. Jack Evilson. Each suspect has a different motive for the murder, including: being harassed, abandoned, sacked, promotion and hate. Other clues are given below. Under a set of additional clues given in the description, the problem is who was the Murderer? And what was the motive, the evidence-object, and the activity associated with each suspect.

The target network of Murder has 20 variables with domains of size 5 and 53 binary constraints. Here we have four types of variables, which are *suspect*, *motive*, *object*, and *activity*, each of them contains five variables. We fed QUACQ, G-QUACQ and M-QUACQ with a basis B of 380 constraints based on the language $\Gamma = \{=, \neq\}$.
Sudoku. The Sudoku model is expressed using 81 variables with domains of size 9, and 810 $\neq$ binary constraints on rows, columns and squares. In this problem, the types are the 9 rows, 9 columns and 9 squares, of 9 variables each. We fed QUACQ, G-QUACQ and M-QUACQ with a basis B of 6480 binary constraints from the language $\Gamma = \{=, \neq\}$.

4.5.2 Results

For all our experiments we report, the total number #Ask of standard queries asked by the basic QUACQ, the total number #AskGen of generalization queries, and the numbers #no and #yes of negative and positive generalization queries, where #AskGen = #no + #yes. The time overhead of using M-QUACQ rather than QUACQ is not reported since computing a generalization query takes a few milliseconds.

First of all, it should be noted that the parameter γ specifying the minimum edge density of quasi-cliques may have an influence on the performance of M-QUACQ. Indeed, the lower γ, the greater the number of extracted quasi-cliques. This means that the probability that extracted types do not correspond to real types increases when γ is small and therefore, the number of negative answers to generalization queries may become important. This phenomenon can be more or less controlled by adjusting the value of γ. That being said, the value of γ was fixed to 0.8 after a few preliminary tests.

Now, with the results reported in table 4.1, we aim at comparing the performance of the basic QUACQ, G-QUACQ where the types are provided by the user and the three versions of M-QUACQ where types are learned during the acquisition process.

Not surprisingly, we notice that M-QUACQ is always better than QUACQ but still less efficient than G-QUACQ. Furthermore, the number of queries (#Ask + #AskGen) that were asked using M-QUACQ is often closer to the number of queries asked using G-QUACQ. For instance, to learn the *PlaceNumPuzzle* QUACQ needs 3746 queries. Providing the types to G-QUACQ reduced the number of queries to 390. Now, using our approach, M-QUACQ needed only 662 queries although no knowledge on types was provided.

In addition, Table 4.1 reports the performance of the three versions of M-QUACQ. What is noticeable about the three different ways for extracting potential types is that `modularity` clearly outperforms the two other techniques and improves significantly the performance of M-QUACQ on all considered problems. Indeed, the combination of M-QUACQ with `modularity` leads to tremendous savings in the number of queries compared to QUACQ: 627+35 (-82%) instead of 3746 on PlaceNumPuzzle, 272+12 (-41%) instead of 483 on Murder, 410+14 (-39%) instead of 694 on Zebra, 140+8 (-28%) instead of 205 on Purdey, 7963+57 (-28%) instead of 9593 on Sudoku.

Table 4.1 – M-QUACQ with `modularity`, `betweenness` and `γ-clique` on PlaceNumPuzzle, Murder, Zebra, Purdey and Sudoku.

	QUACQ	G-QUACQ		M-QUACQ			
Strategies	#Ask	#Ask	#AskGen	#Ask	#AskGen	#no	#yes
Latin Square							
modularity	2058	129	68	987	61	26	35
betweenness				1674	22	5	17
γ-clique				1172	35	1	34
PlaceNumPuzzle							
modularity	3746	351	39	627	35	4	31
betweenness				655	33	2	31
γ-clique				688	33	2	31
Murder							
modularity	483	230	55	272	12	2	10
betweenness				272	12	2	10
γ-clique				342	13	3	10
Zebra							
modularity	694	257	67	410	14	0	14
betweenness				410	14	0	14
γ-clique				410	14	0	14
Purdey							
modularity	205	93	39	140	8	0	8
betweenness				140	8	0	8
γ-clique				140	8	0	8
Sudoku							
modularity	9593	260	166	7963	57	20	37
betweenness				8960	50	18	32
γ-clique				9461	117	104	13

We also notice that the `betweenness` outperforms the `γ-clique` on all problems except for the Latin Square. This can be explained by the fact that the types overlap (rows and columns) in Latin Square and that `γ-clique` is likely more able to detect overlapping types than `betweenness`.

In conclusion we can say that when `modularity` is used to extract the types, the algorithm M-QUACQ performs very well and is very close to G-QUACQ although no knowledge on types is provided. Furthermore, since `modularity` was introduced to find communities in very large networks, we think that the performance of M-QUACQ with `modularity` can be more significant when the size of the problem increases. For instance, Figure 4.4 shows the gain of M-QUACQ with `modularity` compared to QUACQ on the Latin Square when fed with an increasing number of variables. It is clear in this figure that the gain significantly increases with the size of the problem.

4.6 Conclusion

We have proposed MINE&ASK, a generalization based algorithm that is able to mine partial graphs of constraint networks and to generalize, on potential types, constraints learned by any constraint acquisition system. MINE&ASK acts when no knowledge is provided on the variable types. We have detailed and tested three tech-

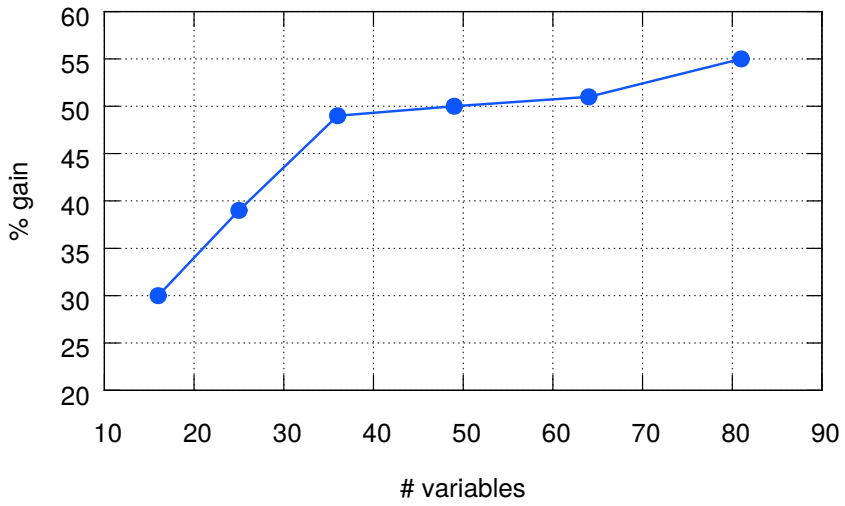

Figure 4.4 – The gain of M-QUACQ on Latin Square when the number of variables increases.

niques to extract potential types, namely the `modularity`, the `betweenness` and the `γ-clique` techniques. We have plugged our MINE&ASK into the QUACQ constraint acquisition system, leading to the M-QUACQ algorithm. We have experimentally evaluated the benefit of our approach on several benchmark problems. The results show that M-QUACQ significantly improves the basic QUACQ algorithm and they are quite close to the results when variable types are provided.

The advantage of such an approach is that, despite we lose in performance, there is no need for the user to provide the types. Nevertheless, generalization queries do not work for all problems. They work only for high structured problems that their variables can be grouped into types. In the following chapter, we introduce a generic kind of query based on the link prediction on dynamic graphs. The advantage of such queries is that they work for a wide range of problems, and they do not need any background knowledge from the user.

CHAPTER

5 Constraint Acquisition with Recommendation Queries

Preamble

In this chapter, we propose PREDICT&ASK*, an algorithm based on the prediction of missing constraints in the partial network learned so far. Such missing constraints are directly asked to the user through* recommendation *queries, a new, more informative kind of queries.* PREDICT&ASK *can be plugged in any constraint acquisition system. We experimentally compare the* QUACQ *system to an extended version boosted by the use of our recommendation queries. The results show that the extended version improves the basic* QUACQ*.*

Contents

5.1 Introduction

Constraint programming (CP) allows effective solving of combinatorial problems in many areas, such as planning and scheduling. However, modeling a combinatorial problem using constraints is a fastidious task that requires significant expertise in CP [Freuder, 1999].

To make constraint programming accessible to novices, several constraint acquisition systems have been introduced in the last decade [Bessiere et al., 2005, 2007; Beldiceanu and Simonis, 2012; Lallouet et al., 2010; Shchekotykhin and Friedrich, 2009]. These systems either need an expert to validate the learned model or need an exponential number of queries to converge on the target constraint network [Bessiere et al., 2015]. Recently, a system polynomial in terms of queries, called QUACQ, has been proposed [Bessiere et al., 2013]. QUACQ iteratively generates *partial* queries (that is, partial assignments of the variables) and asks the user to classify them. When the answer of the user is *yes*, QUACQ reduces the search space by removing constraints that reject the positive example. In the case of a negative answer, QUACQ focuses on a constraint in a number of queries logarithmic in the size of the example. This key component allows QUACQ to converge on the target network in a polynomial number of queries. Despite this good theoretical bound, QUACQ may require a lot of queries to learn the target constraint network, especially when the problem is highly structured and involves a large number of constraints. For instance, the user has to classify more than 8000 queries to get the Sudoku model.

The next challenge to constraint acquisition is to reduce the dialog length between the user and the learner or, in other words, to reduce the number of asked queries to get the target model. This chapter presents a generic approach to constraint acquisition which is centered on the following question: Given the constraint graph learned so far, can we infer which new constraints are more likely to belong to the target constraint network? We formalize this question as a link prediction problem in the partial constraint graph learned so far. Furthermore, we introduce a new concept of query, called *recommendation* query. Borrowing techniques from the link prediction field, a recommendation query asks the user whether or not a predicted constraint belongs to the target constraint network. To deal with recommendation queries, we propose a new constraint recommender algorithm called PREDICT&ASK, which we plugged into the QUACQ constraint acquisition system leading to the P-QUACQ algorithm. We experimentally evaluated the benefit of our approach on several benchmark problems. The results show that P-QUACQ significantly improves the basic QUACQ algorithm in terms of number of queries.

The rest of the chapter is organized as follows. Section 5.2 describes the constraint recommender algorithm. We illustrate the idea behind our constraint recommender algorithm through an example in Section 5.3. In Section 5.4, several predictor techniques are presented. Section 5.5 presents the experimental results we obtained when comparing P-QUACQ to the basic QUACQ. Section 5.6 presents the related work. Section 5.7 concludes the chapter.

5.2 PREDICT&ASK Algorithm

In this section, we present our constraint recommender PREDICT&ASK algorithm. The idea behind this algorithm is to predict missing constraints in the partial net-

work learned so far, and then to recommend the predicted constraints to the user through a new kind of query, recommendation queries. A *recommendation query* $\mathrm{AskRec}(c)$ asks the user whether or not the constraint c belongs to the target constraint network C_T. It is answered yes if and only if c belongs to C_T.

5.2.1 Description of PREDICT&ASK

The algorithm PREDICT&ASK takes as argument the set of constraints C learned so far, a relation r, and the predictor $score$ that corresponds to the strategy used to assign a cost to a candidate constraint for recommendation. The algorithm uses the local data structure Δ which contains all constraints that are candidate for recommendation.

PREDICT&ASK starts by initializing L to the empty set (line 1). The set L will contain the output of PREDICT&ASK, that is all constraints learned by prediction plus recommendation query. In line 4, we build the constraint graph $G = (Y, E)$ restricted to the relation r. The counter #No counts the number of consecutive times recommendation queries have been classified negative by the user. It is initialized to zero at line 5. We put in Δ all constraints that are candidate for recommendation.

In the main loop of PREDICT&ASK (line 7), for each iteration, we pick a constraint from Δ such that its score is maximum (line 8). A constraint with a high score means

Algorithm 9: PREDICT&ASK

Input: C: a set of constraints,
r: a relation,
$score \in \{AA, LHN\}$: a prediction strategy
Output: L: a set of predicted constraints

```
1  L ← ∅;
2  Y ← ⋃var(c)  s.t. (c ∈ C ∧ rel(c) = r)
3  E ← {(x,y) | c ∈ C ∧ rel(c) = r ∧ var(c) = (x,y)}
4  G ← (Y,E)
5  #No ← 0
6  Δ ← {((x,y),r) ∈ B | (x,y) ∈ Y² \ E}
7  while Δ ≠ ∅ ∧ #No < α do
8      pick ((x,y),r) in Δ that maximizes score((x,y),G)
9      if AskRec((x,y),r) = yes then
10         L ← L ∪ {((x,y),r)}
11         E ← E ∪ (x,y)
12         #No ← 0
13     else
14         B ← B \ {((x,y),r') | r ⊆ r'}
15         #No++
16 return L;
```

that it is likely that this constraint belongs to the target constraint network. Hence, PREDICT&ASK asks a recommendation query on $((x,y),r)$ (line 9). If the user says 'yes', $((x,y),r)$ is a constraint of the target network. Hence, we put $((x,y),r)$ in L (line 10). We also add the edge (x,y) to E to be taken into account in the next iteration when computing the *score*. In line 12, we reinitialize #No to zero. If the user says 'no', we remove from B the constraint $((x,y),r)$ (line 14) and we increment #No (line 15). The loop ends when Δ is empty or when #No reaches the given threshold α, and we return L (line 16).

5.2.2 Using Recommendation in QUACQ

PREDICT&ASK is a generic constraint recommender algorithm that can be plugged into any constraint acquisition system. In this section, we present P-QUACQ (Algorithm 10) where we incorporate PREDICT&ASK into the QUACQ system.

P-QUACQ initializes the constraint network C_L to the empty set (line 1). When C_L is unsatisfiable (line 3), the space of possible networks collapses because there does not exist any subset of the given basis B that is able to correctly classify the examples already asked to the user. In line 4, P-QUACQ computes a complete assignment e satisfying C_L and violating at least one constraint from B. If such an example does not exist (line 5), then all constraints in B are implied by C_L, and the algorithm has converged. Otherwise, we propose the example e to the user, who will answer by *yes* or *no* (line 6). If the answer is *yes*, we can remove from B the set $\kappa_B(e)$ of all constraints in B that reject e (line 7). If the answer is *no*, we are sure that e violates at least one constraint of the target network C_T. We then call the function `FindScope` to discover the scope of one of these violated constraints. Here, `FindScope` acts in a dichotomous manner and asks a number of queries logarithmic in the size of the example. `FindC` selects which constraint with the given scope is violated by e (line 9). If no constraint is returned (line 10), this is a condition for collapsing as we could not find in B a constraint rejecting one of the negative examples. Otherwise, we know that the constraint c returned by `FindC` belongs to the target network C_T, we then add it to the learned network C_L (line 12). Note that `FindScope` and `FindC` functions are used exactly as they appear in [Bessiere et al., 2013]. Afterwards, we call PREDICT&ASK to mine the learned constraint network C_L in order to predict and recommend missing constraints that may belong to the target network. P-QUACQ updates C_L by adding all learned constraints (line 13).

5.2.3 Complexity Analysis

Let us now give the theoretical upper bound of the new constraint acquisition system P-QUACQ.

Theorem 5.1. *Given a constraint basis B built from a language Γ of bounded arity, and a target network C_T, P-QUACQ uses $O(C_T.(\log|X|+\Gamma)+|B|)$ queries to prove convergence or to collapse.*

Algorithm 10: P-QUACQ = QUACQ + PREDICT&ASK

Input: $score \in \{AA, LHN\}$: a predictor strategy
Output: C_L: a set of learned constraints

```
1  C_L ← ∅ ;
2  while true do
3      if sol(C_L) = ∅ then return "collapse";
4      choose e in D^X accepted by C_L and rejected by B
5      if e = nil then return "convergence on C_L";
6      if Ask(e) = yes then
7          B ← B \ κ_B(e);
8      else
9          c ← FindC(e, FindScope(e, ∅, X, false));
10         if c = nil then return "collapse";
11         else
12             C_L ← C_L ∪ {c};
13             C_L ← C_L ∪ PREDICT&ASK(C_L, rel(c), score);
```

Proof. By construction, P-QUACQ inherits the correctness of QUACQ, and thus, it always finishes by proving convergence or collapsing. As for its complexity, P-QUACQ asks partial queries (line 6 of P-QUACQ) and recommendation queries (line 9 of PREDICT&ASK). By construction, the number of partial queries in P-QUACQ is bounded above by the number of partial queries of pure QUACQ, that is, $O(C_T.(\log|X|+\Gamma)+|B|)$ [Bessiere et al., 2013]. Concerning recommendation queries, we know that they are asked on constraints that are in B and not in C_L (Algorithm 9, lines 6 and 8). Furthermore, a recommendation query cannot be asked twice on the same constraint as, whatever the answer, the constraint is put in C_L (*yes* answer, Algorithm 9, line 10 and Algorithm 10, line 13) or removed from B (*no* answer, Algorithm 9, line 14). As a result the number of recommendation queries asked by PREDICT&ASK is in $O(|B|)$ and the number of queries asked by P-QUACQ is in $O(C_T.(\log|X|+\Gamma)+|B|)$. □

5.3 An Illustrative Example

In this section, we illustrate our constraint recommender algorithm PREDICT&ASK through an example. Figure 5.1(a) shows the constraint network of the problem that the user has in mind. This problem involves 10 variables and 21 binary constraints. Two relations are used, noted r_1 and r_2 in Figure 5.1. Figure 5.1(b) shows the constraint network partially learned by QUACQ. Suppose that the last constraint learned using QUACQ was $((x_1,x_2),r_1)$. At that point, we want to recommend potential constraints on which the relation r_1 may hold. PREDICT&ASK builds a partial network limited to the relation r_1 (Figure 5.1(c)), and

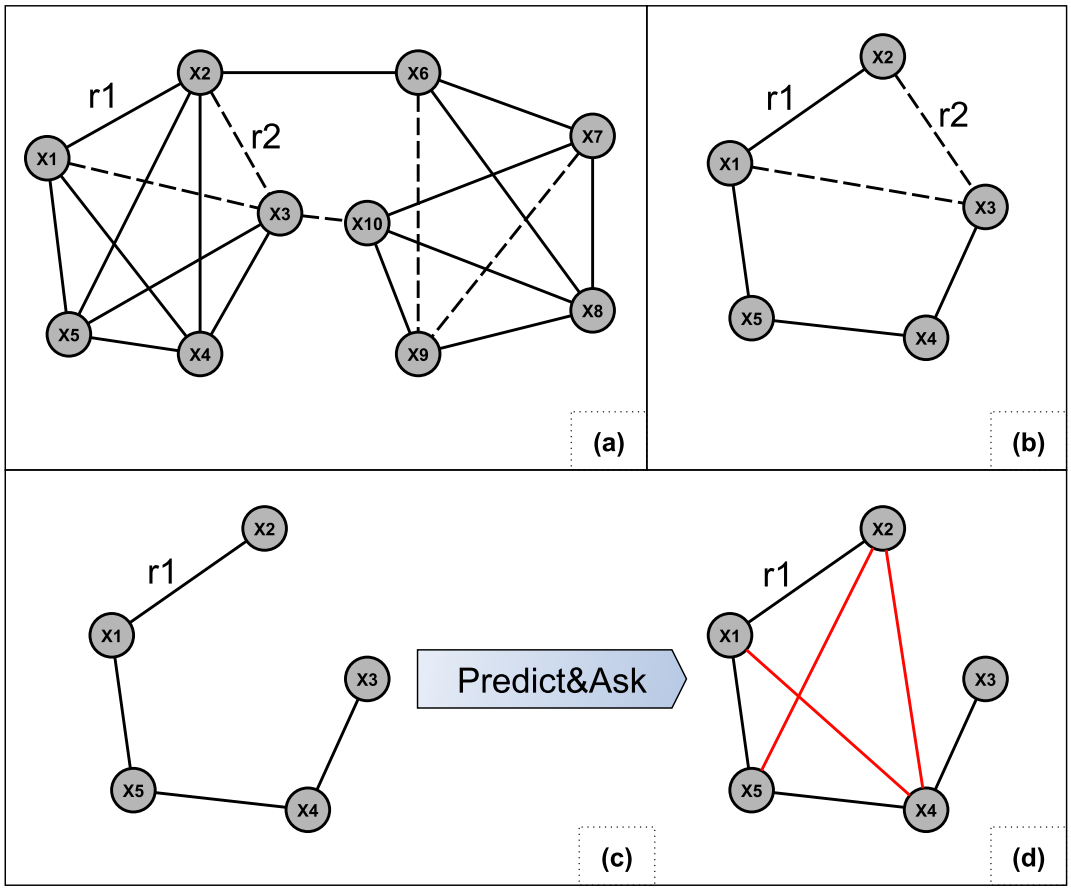

Figure 5.1 – PREDICT&ASK on the illustrative example.

then computes the set Δ of all candidate constraints that may belong to the target network. $\Delta = \{((x_1,x_3),r_1),((x_1,x_4),r_1),((x_2,x_3),r_1),\ ((x_2,x_4),r_1),((x_2,x_5),r_1),((x_3,x_5),r_1)\}$. Then, PREDICT&ASK assigns to each candidate constraint in Δ a score. We sort the elements of Δ in decreasing order of their score. Suppose that we have the following order $\langle((x_1,x_4),r_1),((x_2,x_4),r_1),((x_2,x_5),r_1),\ ((x_2,x_3),r_1),((x_3,x_5),r_1),((x_1,x_3),r_1)\rangle$. Suppose that $\alpha = 1$, which means that we have to exit PREDICT&ASK after one negative answer. We pick the first constraint $((x_1,x_4),r_1)$ in Δ, and we ask the user the recommendation query $AskRec((x_1,x_4),r_1)$, which will be answered yes, as the constraint $((x_1,x_4),r_1)$ belongs to the target network. The other questions are as follows:

— $AskRec((x_2,x_4),r_1) = yes \quad (\#No = 0)$
— $AskRec((x_2,x_5),r_1) = yes \quad (\#No = 0)$
— $AskRec((x_2,x_3),r_1) = no \quad (\#No = 1 \Rightarrow exit)$

At the end, thanks to PREDICT&ASK three (out of four) constraints are added to the current constraint network (see Figure 5.1(d)).

5.4 Prediction Strategies

The way PREDICT&ASK computes the *score* has not been detailed in Section 5.2. In this section, we present the two techniques that we have used to predict missing constraints. Bessiere et al. (2014a) have shown that when a constraint network has some structure, variables of the same given *type* are often involved in constraints with the same relation. Hence, we expect that when variable types are not known in advance, predicting type similarity or type proximity of variables could be done by prediction link techniques.

Link prediction in dynamic graphs is an important research field in data mining. Link prediction can be used for recommendation systems [Li and Chen, 2009], security domain [Krebs, 2002], social networks [Liben-Nowell and Kleinberg, 2003], and many other fields. Several techniques have been proposed in the literature for link prediction. All these techniques compute and assign a score to pairs of nodes (x, y), based on the input graph and then produce a ranked list in a decreasing order of scores. They can be viewed as computing a measure of proximity or similarity between nodes x and y, with respect to the network topology. Most of these techniques are based either on node neighborhood or on path ensemble [Lu and Zhou, 2010]. In our experiments we selected one link prediction technique representative of node-neighborhood-based techniques (Adamic/Adar –AA), and one representative of path-ensemble-based techniques (Leicht-Holme-Newman Index –LHN). Both of these techniques have a time complexity in $O(n^3)$. We will see in our experiments that this never takes more than a few milliseconds.

5.4.1 Adamic-Adar Index (AA)

Adamic and Adar (2003) proposed a measure in the context of deciding when two personal home pages are strongly "related". They compute features of the pages and define the similarity index between two pages to be:

$$\sum_{z:\mathcal{Z}} \frac{1}{\log(\text{frequency}(z))}$$

where $\mathcal{Z}$ is the set of features shared by x and y. This refines the simple counting of common features by weighting rarer features more heavily. This suggests the measure

$$\text{score}(x,y) = \sum_{z \in N(x) \cap N(y)} \frac{1}{\log|N(z)|}$$

where $N(x)$ denotes the neighborhood of x, that is, the set of variables with whom it shares a constraint.

5.4.2 Leicht-Holme-Newman Index (LHN)

Leicht-Holme-Newman (2006) proposed to compute vertex similarity, or proximity, based on the concept that two nodes are similar when their neighbors are similar. This index can be expressed into a matrix form as:

$$S = 2m\lambda_1 D^{-1}(I - \frac{\phi A}{\lambda_1})^{-1} D^{-1}$$

where m is the number of links in the network, λ_1 is the largest Eigenvalue of the adjacency matrix A, D is a diagonal degree matrix, I is the identity matrix, and ϕ ($0 < \phi < 1$) is a free parameter that assigns higher weights to shorter paths if it is closer to 0 and to longer paths if it is closer to 1 [Lu and Zhou, 2010]. In all our experiments we have set ϕ to 0.5 to assign the same weight to both shorter and longer paths.

5.5 Experimental Evaluation

We made experiments to evaluate the impact of using PREDICT&ASK in constraint acquisition. We first present the benchmark problems we used for our experiments. Then, we report the results of acquiring these problems with the basic version of QUACQ, with a brute-force algorithm using only recommendation queries (denoted by ONLYREC), and with our P-QUACQ using the Adamic/Adar (AA) and Leicht-Holme-Newman (LNH) indexes to recommend constraints to the user. ONLYREC makes a brute-force use of recommendation queries: it asks recommendation queries on constraints from B and removes redundant constraints from B each time a new constraint is learned, until convergence is reached. Our tests were conducted on an Intel Core i5-3320M CPU @ 2.60GHz × 4 with 4 Gb of RAM.

5.5.1 Benchmark Problems

Radio Link Frequency Assignment Problem. The RLFAP is to provide communication channels from limited spectral resources [Cabon et al., 1999]. Here we build a simplified version of RLFAP that consists in distributing all the frequencies available on the base stations of the network. The constraint model has 36 variables with domains of size 36, and 210 binary constraints. We fed QUACQ and P-QUACQ with a basis of 1800 binary constraints taken from a language of 6 arithmetic and distance constraints.

Vessel Loading. Supply vessels transport containers from site to site. The deck area is rectangular. Containers are cuboid, and are laid out in a single layer. All containers are positioned parallel to the sides of the deck. The contents of the containers determine their class. Certain classes of containers are constrained to be separated by minimum distances either along the deck or across the deck. The constraint model has 25 variables with domains of size 25, and 210 binary constraints.

We fed QuAcq and p-QuAcq with a basis of 2610 binary constraints taken from a language of 6 arithmetic and distance constraints.

Murder. Someone was murdered last night, and you are summoned to investigate the murder. The objects found on the spot that do not belong to the victim include: a pistol, an umbrella, a cigarette, a diary, and a threatening letter. There are also witnesses who testify that someone had argued with the victim, someone left the house, someone rang the victim, and some walked past the house several times about the time the murder occurred. The suspects are: Miss Linda Ablaze, Mr. Tom Burner, Ms. Lana Curious, Mrs. Suzie Dulles, and Mr. Jack Evilson. Each suspect has a different motive for the murder, including: being harassed, abandoned, sacked, promotion and hate. Under a set of additional clues given in the description, the problem is who was the Murderer? And what was the motive, the evidence-object, and the activity associated with each suspect. The target network of Murder has 20 variables with domains of size 5, and 53 binary constraints. We fed QuAcq and p-QuAcq with a basis $\mathbb{B}$ of 1140 binary constraints based on the language $\Gamma = \{=, \neq, \geq, <, \leq, >\}$.

Zebra problem. The Lewis Carroll's Zebra problem is formulated using 25 variables, with 5 cliques of $\neq$ constraints and 14 additional constraints given in the description of the problem. We fed QuAcq and p-QuAcq with a basis $\mathbb{B}$ of 4450 unary and binary constraints taken from a language with 24 basic arithmetic and distance constraints.

5.5.2 Results

We compare QuAcq, OnlyRec, and p-QuAcq. For p-QuAcq we report results when predicting links with AA or LHN, without cutoff (i.e, $\alpha = +\infty$) and also with four values for the cutoff α (from 1 to 4). We also report results when predicting links with a Random strategy, which serves as baseline selector as it randomly picks a candidate constraint from Δ, and recommends it to the user. For all our experiments we report the number of (standard) queries asked by the basic QuAcq, the number of (recommendation) queries asked by OnlyRec, and the number of queries asked by p-QuAcq. For p-QuAcq we report the number #Ask of standard queries, the number #AskRec of recommendation queries, the numbers #no and #yes of negative and positive recommendation queries (i.e., #AskRec = #no + #yes), and the total number #query of queries (i.e., #query = #Ask + #AskRec). The time overhead of computing scores and generating recommendation queries is not reported because it takes a few milliseconds.

Table 5.1 reports the results of acquiring the RLFAP problem. We first observe that the number of queries asked by p-QuAcq is always significantly lower than with QuAcq or OnlyRec, whatever the way we predict links in p-QuAcq. We also observe that AA and LHN outperform Random for all values of α, which means that their predictions are correlated to the probability of having a constraint at the selected link. When predicting links with AA, we observe that cutoffs hurt the acquisition: the smaller the cuttoff, the greater the number of queries required

Table 5.1 – P-QuAcq on RLFAP.

	α	#query	#Ask	#AskRec	#no	#yes
QuAcq	–	1653	1653	–	–	–
OnlyRec	–	1575	–	1575	–	–
P-QuAcq +Random	+∞	964	560	404	322	82
	4	1017	676	341	268	73
	3	1129	817	312	250	62
	2	1281	1013	268	220	48
	1	1553	1370	183	161	22
P-QuAcq +AA	+∞	964	560	404	322	82
	4	970	643	327	250	77
	3	1010	719	291	220	71
	2	1028	784	244	178	66
	1	1229	1055	174	128	46
P-QuAcq +LHN	+∞	964	560	404	322	82
	4	886	564	322	240	82
	3	859	580	279	197	82
	2	**851**	624	227	148	79
	1	1052	878	174	114	60

for convergence. On the contrary, P-QuAcq +LHN is better with cutoff: it reaches its lower number of queries to learn the RLFAP network when $\alpha = 2$ (851 queries instead of 1653 for basic QuAcq and 1575 for OnlyRec). The good performance of LHN on RLFAP can be explained by the structure of that problem. The RLFAP structure contains bicliques and cliques. The constraints that belong to the same clique can be easily predicted by both the neighborhood-based method, AA, or the path-ensemble-based method, LHN. However, constraints in bicliques cannot be predicted by AA because variables of the same constraint do not share any neighbor (see Figure 5.2a).

Table 5.2 reports the results on the Vessel Loading problem. The structure of this problem is quite similar to the structure of RLFAP. Thus, the results follow the same trend as on the RLFAP (P-QuAcq +LHN with $\alpha = 2$ is the best). However, we see that as opposed to the RLFAP, P-QuAcq +AA benefits from the cutoffs.

Table 5.3 reports the results on the Murder problem. The structure of that problem is essentially composed of cliques, as we can see in Figure 5.2b. In this case, we observe that P-QuAcq +AA with a cutoff equal to 1 is the best. It requires 367 queries to get the model instead of 585 queries for QuAcq and 1050 for OnlyRec. The good performance of the AA predictor can be explained by the fact that neighborhood-based predictors are effective in detecting cliques.

Table 5.4 reports the results on the Zebra problem. Again, P-QuAcq with AA or LHN predictors outperforms QuAcq, OnlyRec, and P-QuAcq +Random. When comparing AA to LHN, we observe that, interestingly, P-QuAcq +AA and P-QuAcq +LHN give exactly the same results for all values of the cutoff α. This can be explained by the fact that only the relation ≠ shows a structure in the network, and that structure is such that all cliques are isolated, as illustrated in Figure 5.2c.

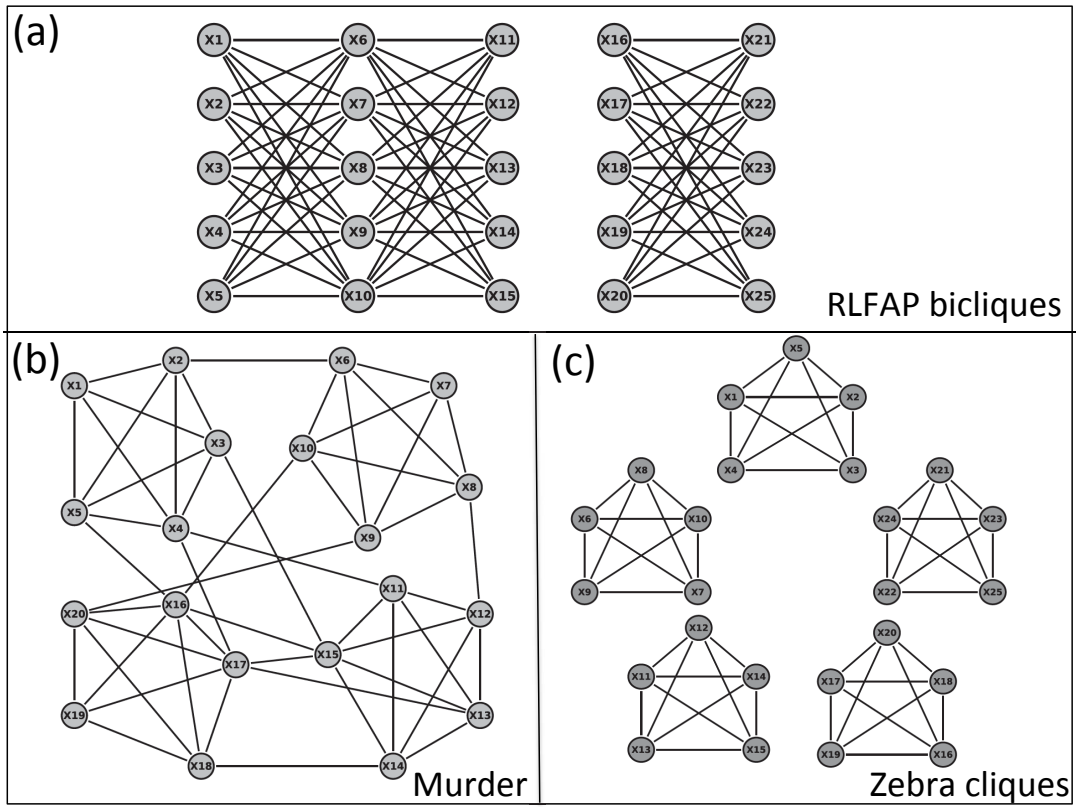

Figure 5.2 – Constraint graphs of our problems.

Such a structure is perfectly well detected both by AA and LHN. This explains that they behave the same and that the shorter the cutoff, the better.

This experimental analysis clearly shows that the use of prediction strategies with recommendation queries can significantly reduce the number of queries asked to the user. The brute-force use of recommendation queries (OnlyRec) is always close to the worst case (i.e., close to $|B|$ queries). The AA prediction strategy seems to be particularly well-suited to problem containing cliques of constraints, whereas the LHN can be highly efficient to predict biclique structures.

5.6 Related Work

Several papers have already proposed to use the structure of the constraint graph to decrease the number of examples needed to learn the target constraint network. Beldiceanu and Simonis (2012) have proposed ModelSeeker, a passive constraint acquisition system devoted to problems having a regular structure. ModelSeeker

Table 5.2 – p-QuAcq on Vessel Loading.

	α	#query	#Ask	#AskRec	#no	#yes
QuAcq	–	2252	2252	–	–	–
OnlyRec	–	2595	–	2595	–	–
p-QuAcq +Random	+∞	1505	864	641	501	140
	4	1655	1159	496	378	118
	3	1686	1212	474	361	113
	2	1758	1368	390	289	101
	1	1934	1658	276	196	80
p-QuAcq +AA	+∞	1505	864	641	501	140
	4	1385	889	496	357	139
	3	1240	852	388	266	122
	2	1195	882	313	194	119
	1	1270	1033	237	128	109
p-QuAcq +LHN	+∞	1505	864	641	501	140
	4	1381	892	489	350	139
	3	1213	831	382	259	123
	2	**1137**	827	310	187	123
	1	1217	988	229	116	113

Table 5.3 – p-QuAcq on Murder.

	α	#query	#Ask	#AskRec	#no	#yes
QuAcq	–	585	585	–	–	–
OnlyRec	–	1050	–	1050	–	–
p-QuAcq +Random	+∞	433	267	166	138	28
	4	453	315	138	115	23
	3	496	380	116	100	16
	2	497	406	91	78	13
	1	523	467	56	49	7
p-QuAcq +AA	+∞	433	267	166	138	28
	4	414	282	132	105	27
	3	404	292	112	86	26
	2	386	301	85	60	25
	1	**367**	313	54	30	24
p-QuAcq +LHN	+∞	433	267	166	138	28
	4	448	309	139	115	24
	3	428	310	118	94	24
	2	439	349	90	70	20
	1	459	401	58	43	15

learns global constraints from the global constraints catalog ([Beldiceanu et al., 2007]) whose scopes are the rows, the columns, or any other structural property ModelSeeker can capture. The counterpart is that it misses any constraint that does not belong to one of the structural patterns it is able to capture. Bessiere et al. (2014a) introduced a new concept of query, called generalization query. By using some background knowledge, namely types of variables, a generalization query asks the user whether or not a learned constraint can be generalized to other scopes of variables of the same types as those of the learned constraint. The drawback of such queries is that they require types of variables to be provided by the user.

Table 5.4 – P-QuAcq on Zebra.

	α	#query	#Ask	#AskRec	#no	#yes
QuAcq	–	694	694	–	–	–
OnlyRec	–	4142	–	4142	–	–
P-QuAcq +Random	+∞	675	423	252	221	31
	4	679	496	183	163	20
	3	660	505	155	132	23
	2	711	593	118	106	12
	1	693	625	68	60	8
P-QuAcq +AA	+∞	675	423	252	221	31
	4	602	431	171	141	30
	3	576	434	142	112	30
	2	524	417	107	77	30
	1	**498**	428	70	40	30
P-QuAcq +LHN	+∞	675	423	252	221	31
	4	602	431	171	141	30
	3	576	434	142	112	30
	2	524	417	107	77	30
	1	**498**	428	70	40	30

To overcome this weakness, Daoudi et al. (2015) have proposed to learn types of variables during the constraint acquisition process, and then to use the learned types to generate generalization queries. The advantage of such an approach is that there is no need for the user to provide the types. Of course learning the types requires extra queries that were not needed when types are given for free at the beginning of the learning process. In addition, generalization queries do not work on all problems. They work only for the problems for which variables can be grouped into types.

By contrast, in our work, recommendation queries are generic and do not require any background knowledge to be generated. By using techniques borrowed from link prediction in dynamic graphs, we infer constraints that are more likely to belong to the target constraint network, and that are validated by asking recommendation queries to the user.

5.7 Conclusion

We have proposed a new kind of queries, called recommendation queries. To deal with these queries, we have proposed a generic constraint recommender algorithm, Predict&Ask, which uses techniques borrowed from link prediction to predict constraints that are likely to belong to the target network. Finally, we have plugged Predict&Ask into QuAcq to have a boosted version called P-QuAcq. Our experiments on several benchmark problems show that our new technique outperforms the basic QuAcq. An interesting direction would be to use a reinforcement learning to decide on the use of neighborhood-based predictions or path-ensemble-based predictions.

CHAPTER

6 Conclusion & Perspectives

We have proposed a new technique to make constraint acquisition more efficient by using information on the types of components the variables in the problem represent. We have introduced generalization queries, a new kind of query asked to the user to generalize a constraint to other scopes of variables of the same type where this constraint possibly applies. Our new technique, GENACQ, can be called to generalize each new constraint that is learned by any constraint acquisition system. We have proposed several heuristics and strategies to select the good candidate generalization query. We have plugged GENACQ into the QUACQ constraint acquisition system, leading to the G-QUACQ algorithm. We have experimentally evaluated the benefit of our approach on several benchmark problems, with and without complete knowledge on the types of variables. The results show that G-QUACQ dramatically improves the basic QUACQ algorithm in terms of number of queries.

The drawback of generalization queries is that they require types of variables of the learned constraint to be generated. To overcome this weakness, and in order to make the build of such queries totally independent of the user, we have proposed to learn types of variables during the constraint acquisition process, and then to use the learned types to generate generalization queries. We have proposed MINE&ASK, a generalization based algorithm that is able to mine partial graphs of constraint networks and to generalize, on potential types, constraints learned by any constraint acquisition system. MINE&ASK acts when no knowledge is provided on the variable types. We have detailed and tested three techniques to extract potential types, namely the `modularity`, the `betweenness` and the γ`-clique` techniques. We have plugged our MINE&ASK into the QUACQ constraint acquisition system, leading to the M-QUACQ algorithm. We have experimentally evaluated the benefit of our approach on several benchmark problems. The results show that M-QUACQ signif-

icantly improves the basic QuAcQ algorithm and they are quite close to the results when variable types are provided.

The advantage of such an approach is that, despite we lose in performance, there is no need for the user to provide the types. Nevertheless, generalization queries do not work on all problems; they work only for high structured problems that their variables can be grouped into types. Toward a generic concept of query, we have introduced a new kind of query, recommendation queries based on the link prediction in dynamic graphs. The advantage of such queries is that they work for a wide range of problems, and they do not need any background knowledge from the user. To deal with that kind of queries, we have proposed a generic constraint recommender algorithm, named Predict&Ask, which uses techniques borrowed from the link prediction in dynamic graphs to predict constraints that are more likely to belong to the target network, and then asks the user to classify recommendation queries. Finally, We have plugged Predict&Ask into QuAcQ system to have a boosted version called p-QuAcQ. To evaluate the benefit of recommendation queries, we have made experiments on several benchmarks. The results show that our new technique outperform drastically the basic QuAcQ.

Our framework opens up two general areas for further work. First, the framework can be applied as it stands to other disciplines, such as criteria weight elicitation and preference learning. Second, the framework can be extended and improved in several ways. First, we plan to learn the structure of the problem in order to choose the appropriate prediction technique. We know that the performance of a prediction technique depends on the topology of the constraint network. We think that a multi-armed bandit technique may be useful for the choice of the appropriate technique during the acquisition process. Second, we plan to recommend not only one constraint that may belongs to the target network, but also to recommend a part of that network or, why not, the entire network. Third, we plan to incorporate in one system all the proposed techniques. In other words, we will make all our new generic techniques collaborating in order to get the target network as quickly as possible. That is, using at the same time the three kind of queries, namely basic queries, generalization queries, and recommendation queries. The question that we will try to answer is, which query to ask at a given step in the learning process? One answer to that question may be the use of the enforcing learning towards an adaptive constraint acquisition. Forth, so far we deal just with binary constraints; so we plan to extend our framework to deal also with constraints of different arities. And also to be able to learn global constraints. Finally, as a perspective of that work, we plan also to propose a distributed constraint acquisition system to learn problems, which may involve a large number of constraints, by several users; that is, a collaborative learning by users that may have to model the same problem.

Bibliography

Clauset Aaron, Newman Mark EJ, and Moore Cristopher. Finding community structure in very large networks. *Physical review E*, 70:066111, 2004. doi: 10.1103/PhysRevE.70.066111. URL http://link.aps.org/doi/10.1103/PhysRevE.70.066111. Cited page 53.

Lada A. Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003. doi: 10.1016/S0378-8733(03)00009-1. URL http://dx.doi.org/10.1016/S0378-8733(03)00009-1. Cited page 65.

Dana Angluin. Queries and concept learning. *Machine Learning*, pages 319–342, 1987. Cited pages 23 and 32.

Nicolas Beldiceanu and Helmut Simonis. A model seeker: Extracting global constraint models from positive examples. In *CP*, pages 141–157, 2012. Cited pages 8, 24, 32, 45, 60, and 69.

Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey, and Thierry Petit. Global constraint catalogue: Past, present and future. *Constraints*, 12(1):21–62, 2007. Cited pages 24, 26, 32, and 70.

Christian Bessiere. Constraint propagation. In *Handbook of Constraint Programming*, pages 29–83. 2006. doi: 10.1016/S1574-6526(06)80007-6. URL http://dx.doi.org/10.1016/S1574-6526(06)80007-6. Cited page 17.

Christian Bessiere and Frédéric Koriche. Non learnability of constraint networks with membership queries. Technical report, Coconut, Montpellier, France, February, 2012. Cited pages 23 and 32.

Christian Bessiere, Remi Coletta, Eugene C. Freuder, and Barry O'Sullivan. Leveraging the learning power of examples in automated constraint acquisition. In *CP*, pages 123–137, 2004. Cited pages 21 and 32.

Christian Bessiere, Remi Coletta, Frédéric Koriche, and Barry O'Sullivan. A sat-based version space algorithm for acquiring constraint satisfaction problems. In *ECML*, pages 23–34, 2005. Cited pages 8, 21, 23, 32, 45, and 60.

Christian Bessiere, Remi Coletta, Barry O'Sullivan, and Mathias Paulin. Query-driven constraint acquisition. In *IJCAI*, pages 50–55, 2007. Cited pages 8, 23, 32, 45, and 60.

Christian Bessiere, Remi Coletta, Emmanuel Hebrard, George Katsirelos, Nadjib Lazaar, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh. Constraint acquisition via partial queries. In *IJCAI*, 2013. Cited pages 8, 26, 29, 32, 36, 45, 50, 54, 60, 62, and 63.

Christian Bessiere, Remi Coletta, Abderrazak Daoudi, Nadjib Lazaar, Younes Mechqrane, and El-Houssine Bouyakhf. Boosting constraint acquisition via generalization queries. In *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, pages 99–104, 2014a. doi: 10.3233/978-1-61499-419-0-99. URL http://dx.doi.org/10.3233/978-1-61499-419-0-99. Cited pages 46, 55, 65, and 70.

Christian Bessiere, Remi Coletta, and Nadjib Lazaar. Solve a constraint problem without modeling it. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 1–7, 2014b. doi: 10.1109/ICTAI.2014.12. URL http://dx.doi.org/10.1109/ICTAI.2014.12. Cited page 45.

Christian Bessiere, Frédéric Koriche, Nadjib Lazaar, and Barry O'Sullivan. Constraint acquisition. *Artificial Intelligence, In Press*, 2015. Cited page 60.

Coenraad Bron and Joep Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973. Cited page 54.

Bertrand Cabon, Simon de Givry, Lionel Lobjois, Thomas Schiex, and Joost P. Warners. Radio link frequency assignment. *Constraints*, 4(1):79–89, 1999. Cited pages 40 and 66.

Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL http://igraph.org. Cited page 53.

Abderrazak Daoudi, Nadjib Lazaar, Younes Mechqrane, Christian Bessiere, and El-Houssine Bouyakhf. Detecting types of variables for generalization in constraint

acquisition. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 413–420, 2015. doi: 10.1109/ICTAI.2015.69. URL http://dx.doi.org/10.1109/ICTAI.2015.69. Cited page 71.

Rina Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artif. Intell.*, 41(3):273–312, 1990. doi: 10.1016/0004-3702(90)90046-3. URL http://dx.doi.org/10.1016/0004-3702(90)90046-3. Cited page 16.

Rina Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003. ISBN 978-1-55860-890-0. URL http://www.elsevier.com/wps/find/bookdescription.agents/678024/description. Cited page 11.

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. Cited pages 7, 18, 45, and 59.

Eugene C. Freuder and Richard J. Wallace. Suggestion strategies for constraint-based matchmaker agents. *International Journal on Artificial Intelligence Tools*, 11 (1):3–18, 2002. Cited page 31.

John Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, August 1977*, page 457, 1977. Cited page 16.

Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002. Cited pages 47 and 53.

Robert M. Haralick and Gordon L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artif. Intell.*, 14(3):263–313, 1980. doi: 10.1016/0004-3702(80)90051-X. URL http://dx.doi.org/10.1016/0004-3702(80)90051-X. Cited page 16.

Holger H. Hoos and Edward P. K. Tsang. Local search methods. In *Handbook of Constraint Programming*, pages 135–167. 2006. doi: 10.1016/S1574-6526(06)80009-X. URL http://dx.doi.org/10.1016/S1574-6526(06)80009-X. Cited page 17.

Takashi Ito, Tomoko Chiba, Ritsuko Ozawa, Mikio Yoshida, Masahira Hattori, and Yoshiyuki Sakaki. A comprehensive two-hybrid analysis to explore the yeast protein interactome. *Proceedings of the National Academy of Sciences*, 98(8):4569–4574, 2001. doi: 10.1073/pnas.061034498. URL http://www.pnas.org/content/98/8/4569.abstract. Cited page 46.

U. Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. pages 167–172, 2004a. Cited page 27.

Ulrich Junker. Quickxplain: Preferred explanations and relaxations for over-constrained problems. In *AAAI*, pages 167–172, 2004b. Cited page 36.

Donald E. Knuth. *The Art of Computer Programming, Volume I: Fundamental Algorithms*. Addison-Wesley, 1968. Cited page 16.

Valdis E. Krebs. Uncloaking terrorist networks. *First Monday*, 7(4), 2002. URL http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/941. Cited page 65.

Arnaud Lallouet, Matthieu Lopez, Lionel Martin, and Christel Vrain. On learning constraint problems. In *ICTAI*, pages 45–52, 2010. Cited pages 8, 32, 45, and 60.

Christophe Lecoutre. *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*. John Wiley & Sons, 2013. Cited page 11.

Elizabeth Leicht, Petter Holme, and Mark Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006. Cited page 66.

Xin Li and Hsinchun Chen. Recommendation as link prediction: a graph kernel-based machine learning approach. In *Proceedings of the 2009 Joint International Conference on Digital Libraries, JCDL 2009, Austin, TX, USA, June 15-19, 2009*, pages 213–216, 2009. doi: 10.1145/1555400.1555433. URL http://doi.acm.org/10.1145/1555400.1555433. Cited page 65.

David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management, New Orleans, Louisiana, USA, November 2-8, 2003*, pages 556–559, 2003. doi: 10.1145/956863.956972. URL http://doi.acm.org/10.1145/956863.956972. Cited page 65.

Linyuan Lu and Tao Zhou. Link prediction in complex networks: A survey. *CoRR*, abs/1010.0725, 2010. URL http://arxiv.org/abs/1010.0725. Cited pages 65 and 66.

Alan K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8 (1):99 – 118, 1977. ISSN 0004-3702. Cited page 17.

Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artif. Intell.*, 58(1-3):161–205, 1992. doi: 10.1016/0004-3702(92)90007-K. URL http://dx.doi.org/10.1016/0004-3702(92)90007-K. Cited page 17.

Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997. ISBN 978-0-07-042807-2. Cited pages 18 and 21.

Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004. Cited pages 46, 51, and 52.

Patrick Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993. doi: 10.1111/j.1467-8640.1993.tb00310.x. URL http://dx.doi.org/10.1111/j.1467-8640.1993.tb00310.x. Cited page 16.

Jean-Francois Puget. Constraint programming next challenge: Simplicity of use. In *CP*, pages 5–8, 2004. Cited pages 7 and 18.

Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006a. ISBN 978-0-444-52726-4. URL http://www.sciencedirect.com/science/bookseries/15746526/2. Cited page 11.

Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006b. ISBN 978-0-444-52726-4. URL http://www.sciencedirect.com/science/bookseries/15746526/2. Cited page 16.

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. Cited pages 45 and 60.

Mark Wallace and Joachim Schimpf. Finding the right hybrid algorithm - A combinatorial meta-problem. *Ann. Math. Artif. Intell.*, 34(4):259–269, 2002. doi: 10.1023/A:1014450507312. URL http://dx.doi.org/10.1023/A:1014450507312. Cited page 17.

Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994. URL http://scholar.google.com/scholar.bib?q=info:gET6m8icitMJ:scholar.google.com/&output=citation&hl=en&as_sdt=0,5&as_vis=1&ct=citation&cd=0. Cited page 46.

Patrick Henry Winston. *Artificial intelligence (3. ed.)*. Addison-Wesley, 1992. ISBN 978-0-201-53377-4. Cited page 21.

List of Algorithms

List of Figures

List of Tables

Abstract

Most of the existing constraint acquisition systems interact with the user by asking her to classify an example as positive or negative. Such queries do not use the structure of the problem and can thus lead the user to answer a large number of queries. In this thesis, we show that using the structure of the problem may improve the acquisition process a lot. To this end, we introduce two new concept of queries that use the structure. The first one, called generalization query, based on an aggregation of variables into types. The second one, named recommendation query, based on the prediction of missing constraints in the current constraint graph. In addition, we propose several algorithms and strategies to deal with these new kind of queries. We incorporate all our proposed algorithms into QuAcQ constraint acquisition system, leading to three boosted versions, namely g-QuAcQ, m-QuAcQ, and p-QuAcQ. The results show that the extended versions improve drastically the basic QuAcQ.

Keywords: Artificial Intelligence, Constraint Programming, Constraint Acquisition, Graph Community Detection, Recommendation Systems.

Résumé

La plupart des systèmes d'acquisition de contraintes existants interagissent avec l'utilisateur en lui demandant de classer un exemple comme positif ou négatif. Ces requêtes n'utilisent pas la structure du problème ce qui peut entraîner l'utilisateur à répondre à un grand nombre de questions pour apprendre toutes les contraintes. Dans cette thèse, nous montrons que l'utilisation de la structure du problème peut améliorer considérablement le processus d'acquisition. Pour ce faire, nous introduisons deux nouveaux concepts de requêtes utilisant la structure du problème. Le premier concept est celui de requête de généralisation basée sur une agrégation des variables sous forme de types. Le deuxième concept est celui de requête de recommandation basée sur la prédiction des contraintes manquantes dans le graphe courant de contraintes. En outre, nous proposons plusieurs algorithmes et stratégies pour faire face à ces nouveaux types de requêtes. Enfin, nous intégrons ces nouveaux algorithmes dans le système d'acquisition de contraintes QuAcQ, menant à trois nouvelles versions, à savoir g-QuAcQ, p-QuAcQ, et m-QuAcQ. Les résultats montrent que les versions étendues améliorent considérablement le système QuAcQ de base.

Mots clefs : Intelligence Artificielle, Programmation par contraintes, Acquisition de contraintes, Détection de communautés, Systèmes de recommandation.