Constraint Programming for Data Mining and for Natural Language Processing

Thi-Bich-Hanh Dao

UNIVERSITÉ D'ORLÉANS

Laboratoire d'Informatique Fondamentale d'Orléans

Discipline : Informatique

Habilitation à Diriger des Rercherches

présentée et soutenue publiquement le 14 mars 2018 par

Thi-Bich-Hanh DAO

Constraint Programming for Data Mining and for Natural Language Processing

RAPPORTEURS:

Luc De Raedt	Professeur, Katholieke Universiteit Leuven
Lakhdar Sais	Professeur, Université d'Artois
Christine Solnon	Professeur, INSA Lyon

JURY:

Bruno Crémilleux	Professeur, Université Caen Basse-Normandie
Luc De Raedt	Professeur, Katholieke Universiteit Leuven
François Fages	Directeur de recherche, INRIA Saclay
Jin Kao Hao	Professeur, Université d'Angers et Institut Universitaire de France
Lakhdar Sais	Professeur, Université d'Artois
Christine Solnon	Professeur, INSA Lyon
Christel Vrain	Professeur, Université d'Orléans

Résumé

Ce manuscrit d'*Habilitation à Diriger des Recherches* présente mes travaux sur l'application de la programmation par contraintes au traitement automatique des langues et à la fouille de données. En traitement automatique des langues, nous nous intéressons à l'analyse syntaxique des grammaires de propriété, décrites par des propriétés que doivent satisfaire les énoncés grammaticaux. Nous définissons une sémantique formelle en théorie des modèles et formalisons l'analyse syntaxique comme un problème d'optimisation sous contraintes. Nous développons un modèle en programmation par contraintes, ce qui amène à un analyseur entièrement à base de contraintes. En fouille de données, nous considérons le clustering sous contraintes, qui vise à partitionner les objets en groupes homogènes, étant donnés une mesure de dissimilarité entre objets et un ensemble de contraintes utilisateur à satisfaire. Nous développons un cadre déclaratif qui intègre plusieurs critères d'optimisation principaux de clustering et tous les types de contraintes utilisateur populaires. Nous montrons que sa flexibilité permet de trouver la frontière de Pareto pour des problèmes de clustering bi-objectif sous contraintes. Nous améliorons davantage l'efficacité de l'approche en développant des contraintes globales dédiées aux critères d'optimisation de clustering. Nous explorons plusieurs nouveaux problèmes de clustering avec des contraintes et montre que la programmation par contraintes constitue un cadre flexible et efficace pour les résoudre.

Mots-clefs : problème d'optimisation sous contraintes, programmation par contraintes, analyse syntaxique, clustering sous contraintes, contrainte globale d'optimisation.

Abstract

This manuscript of *Habilitation à Diriger des Recherches* presents my work on the application of constraint programming to natural language processing and to data mining. In natural language processing, we are interested in syntactic analysis of property grammars, defined by constraints that must be satisfied by the grammatical utterances. We introduce model-theoretic semantics and formulate the syntactic analysis as a constraint optimization problem. We develop a model using constraint programming, which leads to a fully constraint-based parser. In data mining, we consider constrained clustering problems that aim at partitioning the objects in homogeneous clusters, given a dissimilarity measure between objects and a set of user-constraints to be satisfied. We develop a declarative framework that integrates several principal clustering optimization criteria and all popular types of user-constraints. We show that the flexibility of the framework allows to find the complete Pareto front of bi-objective constrained clustering problems. We enhance further the approach by developing specific global optimization constraints for principal clustering optimization criteria. We explore several new clustering problems with constraints and show that constraint programming offers a general and efficient framework to solve them.

Keywords: constraint optimization problem, constraint programming, parsing, constrained clustering, global optimization constraint.

Remerciements

Tout d'abord je tiens à remercier Luc De Raedt, Lakhdar Saïs et Christine Solnon, qui m'ont fait l'honneur d'avoir accepté de rapporter mes travaux. Je tiens à remercier également mes examinateurs Bruno Crémilleux, François Fages, Jin Kao Hao et Christel Vrain. S'agissant de scientifiques de tout premier plan, je suis particulièrement touchée et émue d'avoir ce jury « de rêve ».

J'exprime toute ma gratitude à Christel Vrain, avec qui j'ai exploré l'application de la programmation par contraintes à la fouille de données. J'apprécie sa perspicacité, sa persévérance et sa rigueur scientifique. Je garde toujours de bons souvenirs de nos moments tant scientifiques que sportifs au travers du pilates et du yoga. Merci encore pour l'ensemble de ses conseils qui m'a amenée à la soutenance de cette habilitation.

Depuis ma thèse de doctorat, j'ai eu grand plaisir à explorer de nouveaux domaines et à collaborer avec de nombreux chercheurs. Chaque collaboration m'a permis de découvrir de nouvelles thématiques ainsi que de nouvelles méthodes de travail. Pour tous ces moments de partage et de collaboration, je tiens à remercier mes co-auteurs : Yohan Boichut, Alain Colmerauer, Ian Davidson, Khalil Djelloul, Denis Duchier, Khanh-Chuong Duong, Abdel-Ali Ed-Dbali, Thom W. Früwirth, Pierre Gançarski, Tias Guns, Chia-Tung Kuo, Arnaud Lallouet, Thomas Lampert, Andreï Legtchenko, Willy Lesaint, Lionel Martin, Valérie Murat, S. S. Ravi, Yannick Parmentier, Jean-Philippe Prost. Merci pour nos échanges scientifiques et pour votre amitié. J'ai une pensée particulière pour ceux qui ne sont plus parmi nous, et pour les autres j'espère que nous aurons encore d'autres occasions de travailler ensemble.

Je remercie tous mes collègues au LIFO et au Pôle Informatique. J'apprécie votre amitié, chacun à leur façon, qui rend le cadre de travail agréable et convivial.

Je dédie une pensée toute particulière à Alain Colmerauer, mon ex-directeur de thèse. Alain et sa femme Colette m'ont toujours apporté leur soutient et amitié infaillible.

Merci à mes amis pour leur soutient et encouragement. Merci à ma petite famille et ma grande famille. Vous êtes ma source d'amour, de joie et de bonheur, qui m'entoure et m'encourage dans ma vie professionnelle.

Encore merci, merci à tous !

Contents

Chapter 1

Introduction

Contents

1.1 Research Topics

My research topics ranged from constraints solving to application of constraint programming in natural language processing or in data mining. They extend over seventeen years since my PhD thesis defended in 2000 in Marseille and have continued in Orléans. They were the result of several collaborations and explorations of new fields. The initial topic that I developed during and after my PhD thesis was on solving first order logic constraints on tree structures. I was recruited as an Associate Professor (*Maître de Conférences*) in Orléans in 2001. I have integrated the *Laboratoire d'Informatique Fondamentale d'Orléans* (LIFO) and the Constraints and Machine Learning team (*Contraintes et Apprentissage*, CA). Working together with some members of CA team I have progressively reoriented my research topics toward constraint programming, in particular finite domain constraints.

The research fields of CA team include constraint programming, machine learning, data mining and natural language processing. Enjoying this diversity, my recent interests have been on topics that link together constraint programming and data mining or natural language processing. Overall, my principal research can be organized into four main topics as follows:

- Solving constraints in tree structures: We developed methods to solve constraints that are first order logic formulas with embedded quantifications and free variables. The constraints are defined on several domains that are extensions of the domain of trees.

- Learning constraint propagators for finite domain constraints solvers: Finite domain constraint solvers are characterized by constraint propagation and search. We developed a framework using machine learning methods to construct propagators for constraints.

- Constraint-based parsing for property grammars in natural language processing: We introduced model-theoretic semantics for property grammars, which allow to represent the syntactic analysis as a constraint optimization problem. We developed a formalization of a fully constraint-based parser for property grammars using constraint programming.

- Declarative approach using constraint programming for constrained clustering: We developed frameworks using constraint programming, that enable a modeling and an efficient solving of various problems in constrained clustering. Taking advantage of the flexibility of constraint programming we explored further the use of constraints in clustering.

After my Master 2 degree in Computer Science (*DEA Informatique*), I had the pleasure of being welcomed at the *Laboratoire d'Informatique de Marseille*, for my PhD thesis under the supervision of Alain Colmerauer. The studied objects were constraints in the theory of finite or infinite trees. Trees are fundamental in computer science since they model data structures, program schemes or program executions. The execution of programs in Prolog II, III and IV was modeled in term of solving equations and disequations in the algebra of finite and infinite trees [Colmerauer 1982, Colmerauer 1984, Colmerauer 1990]. The theory of finite, rational or infinite trees was proven to be complete and decidable [Maher 1988], however there had been no effective algorithm that solves constraints in the tree structure. Constraints to be solved are first order formulas with embedded and alternate quantifications and free variables. My contributions in PhD thesis were: (1) the design of a normal form of constraints where the solutions are explicit and the design of an algorithm for solving constraints by sub-formula rewrite rules that transforms any first order formula to an equivalent formula, which is either *false* or in the normal form [Dao 2000a]; (2) a study on the expressiveness of tree constraints, where we showed that winning positions in a two-player game can be expressed by tree constraints with embedded quantifications, and tree constraints have a quasi-universal expressive power [Colmerauer & Dao 2000, Colmerauer & Dao 2003].

After my PhD thesis, I continued exploring general constraints in different extended structures from the tree structure. I developed this work in collaboration with Alain Colmerauer and Khalil Djelloul (University Aix-Marseille II). We characterized sufficient properties of theories or combination of theories such that the algorithm for solving constraints in the theory of trees can be generalized to a decision algorithm. Several theories satisfy these properties, as for instance the theory of linear dense order without limit, the theory of the rational numbers with addition and subtraction, the theory of the lists, etc. [Djelloul & Dao 2006c, Djelloul & Dao 2006b, Djelloul & Dao 2006a, Dao 2009]. We considered the tree structure, where trees are labeled by symbols and rational numbers. The trees are evaluated such that all the subtrees labeled only by numbers, addition and subtraction symbols are evaluated and reduced to rational numbers. We developed rewrite rules to solve constraints in the evaluated tree structure [Dao & Djelloul 2006] and extended the Prolog model into a solver of first order constraints [Djelloul *et al.* 2007, Djelloul *et al.* 2008].

During the same period, as soon as my arrival at LIFO in 2001, with Abdel-Ali Ed-Dbali, Lionel Martin, Arnaud Lallouet and Andreï Legtchenko (PhD student), members of CA team, we initiated work on automatic construction of solvers for finite domain constraints. Finite domain constraint solvers are characterized by constraint propagation and search. The propagators are designed to enforce a constraint with a certain level of consistency. The efficiency of the solver strongly depends on the efficiency of the propagators, and the task of finding efficient propagators is considered as one of the smartest skills of the solver designer. Pioneering works had investigated in automatic construction of solvers:

constructing rule-based propagators using systematic search [Apt & Monfroy 1999], constructing propagators by rewrite rules in Constraint Handling Rule frameworks using machine learning techniques [Abdennadher & Rigotti 2002]. In our approach, we developed a framework using machine learning methods to construct propagators for finite domain constraints. The principe of the framework is to consider the behavior of the operator enforcing the desired consistency as the set of examples, find an adequate representation of this operator in a given language [Lallouet *et al.* 2003a, Ed-Dbali *et al.* 2003]. The framework was instantiated to learn propagators in the form of indexicals, where each propagator is of the form `X in minX..maxX`. The learned propagators enforce a consistency weaker than bound-consistency but as close to it as possible [Dao *et al.* 2002]. Consistencies can be partially ordered according to their pruning power. The method was extended to build a range of intermediate consistencies for a given constraint, which are located between bound-consistency and arc-consistency [Lallouet *et al.* 2003b].

In CA team, a new line of research, namely natural language processing had been developed. In this context, in 2009, I started collaborating with Denys Duchier, Willy Lesaint, Yannick Parmentier and Jean-Philippe Prost on a theme that connects constraint programming and natural language processing. This work involved my competences on first order logic and built the bridge for me to explore the strengths of constraint programming. In natural language processing, one important task is to analyze the syntax of utterances according to a grammar. An utterance of a natural language can be well-formed according to the grammatical requirements or can be not completely well-formed, yet showing some form of syntactic structure and properties. The former is referred as an expression, the latter is a quasi-expression. In our work, we considered property grammars, where a grammar is given by a set of properties that must be satisfied by each expression of the language. We aimed at developing a fully constraint-based parser, in the same line with what was done for dependency grammars [Duchier & Debusmann 2001, Duchier 2003, Debusmann *et al.* 2004]. An advantage of a fully constraint-based parser is that informations on different aspects of the language could be integrated within the same framework as soon as they can be expressed by constraints. We introduced model-theoretic semantics of property grammars, which allow to formulate the syntactic analysis as a combinatorial search problem [Duchier *et al.* 2009]. The models are trees of syntactic categories and are designed for being expressed by constraints. We developed a formalization presented as a constraint optimization problem, which naturally leads to an implementation of a fully constraint-based parser for property grammars using constraint programming [Duchier *et al.* 2010a]. This framework allows not only to analyze grammatical utterances but also to find out one of the most appropriate syntactic structures for ungrammatical utterances. We extended the framework in order to integrate other types of properties, which allow to express relations between syntactic constituents by constraints on feature structures [Duchier *et al.* 2011, Duchier *et al.* 2014].

A large research field of CA team is on machine learning and data mining, headed by Christel Vrain. Recently the interest of declarative approaches for data mining has been demonstrated [De Raedt *et al.* 2008, De Raedt *et al.* 2010]. In 2011, Christel Vrain and I started our collaboration that brings together constraint programming and data mining. We initiated work on a declarative approach for dissimilarity-based constrained clustering using constraint programming. Given a set of objects, a clustering task aims at grouping the objects into non-empty, disjoint and homogeneous clusters. The homogeneity is usually

characterized by an objective criterion and prior knowledge can be integrated to a clustering process by user-constraints. Numerous methods have been developed for constrained clustering, however they are usually developed for a specific criterion and for some particular types of user-constraints. It is in this context that the PhD thesis of Khanh-Chuong Duong, that I co-supervised with Christel Vrain, took place. We developed a general and declarative framework that takes into account several objective criteria and various types of user-constraints [Dao *et al.* 2013a]. We worked on improving the efficiency of the framework, either by changing the model [Dao *et al.* 2014b, Dao *et al.* 2017] or by developing dedicated global constraints [Dao *et al.* 2013b, Dao *et al.* 2017, Dao *et al.* 2015a]. The flexibility of the framework enables us to find the exact Pareto front of a bi-objective constrained clustering problem [Dao *et al.* 2017]. This is, to our knowledge, the first approach that considers bi-objective clustering under user-constraints.

We have studied and extended our approach using constraint programming in different aspects. Our framework using constraint programming can be extended to clustering tasks that combine constraints and objectives issued from either dissimilarity-based clustering or conceptual clustering. In this context we explored the use of set constraints in the model and we showed that a model using set variables has a better performance than the one using binary variables [Dao *et al.* 2015b]. We explored new clustering problems and showed that they can be solved by exploiting the flexibility and variety of constraint programming: actionable clustering where experts provide complex constraints that make clustering useful in the domain [Dao *et al.* 2016b], minimal clustering modification problem that allows providing guidance a posteriori after a clustering is found [Kuo *et al.* 2017]. Regarding the well-known minimum sum of squares constrained clustering problem, we extended a repetitive branch-and-bound algorithm for unconstrained clustering to constrained clustering and showed that the combination of this algorithm with constraint programming outperforms all existent exact approaches [Guns *et al.* 2016]. The use of our framework in an application of constrained clustering on time-series images involved improving further our framework as well as identifying challenges to consider. These works have been performed within several collaborations that we have started since 2015, with Ian Davidson (University of California Davis, USA), S. S. Ravi (University at Albany, USA), Tias Guns (Katholieke Universiteit Leuven, Belgique) and Pierre Gançarski (University of Strasbourg, France).

1.2 Outline of The Dissertation

My principal research can be organized into four main topics: (i) solving constraints in tree structures, (ii) learning propagators for finite domain constraints, (iii) constraint programming for property grammar constraint-based parsing in natural language processing and (iv) constraint programming for constrained clustering in data mining. My recent interests have been on the topics that link together constraint programming and data mining or natural language processing. The chapters of this dissertation will develop these topics. My work on solving constraints in tree structures and learning propagators for finite domain constraints will be presented in the appendix. The remaining of the dissertation is organized as follows.

Chapter 2 presents main principles of constraint programming. Work to develop con-

straint programming essentially focuses on modeling using constraints, constraint propagation, development of filtering algorithms for global constraints, development of search strategies.

Chapter 3 presents our work on constraint programming for syntactic analysis of property grammars in natural language processing. We present property grammars that describe a language by a set of properties, which must be satisfied by any expression of the language. We introduce model-theoretic semantics for property grammars, where the interpretations are trees labeled by syntactic categories. A mechanism is defined to identify (strong) models for expressions of the language and is generalized to define loose models for quasi-expressions. We present a formulation of the model-theoretic semantics of property grammars as a constraint optimization problem, which naturally leads to a fully constraint-based parser for property grammars using constraint programming.

Chapter 4 presents our work on a declarative framework using constraint programming for constrained clustering. We present dissimilarity-based constrained clustering and review different approaches. We develop a declarative framework using constraint programming, which is general and flexible for modeling various tasks in constrained clustering. The power of constraint programming resides also in the power of constraint propagation. We present global optimization constraints that are designed for optimization criteria in clustering and show that they improve significantly the efficiency of the framework. We consider bi-objective constrained clustering problems and show that the flexibility of the framework enables us to find the exact Pareto front.

Chapter 5 develops different aspects of using constraint programming to explore problems beyond constrained clustering. On the modeling side, using constraint programming enables the modeling of various and more general constraints for more general clustering tasks. We model clustering problem that combines constraints and objectives on both distance-based and conceptual-based properties, as well as actionable clustering that enables the consideration of constraints that aim the clustering for some purposes. Another clustering problem is explored: starting from a clustering given by a clustering algorithm, which is in general good but which presents some undesired properties, minimal clustering modification aims at finding a clustering which is as close as possible without having the undesired properties. On the effectiveness side, we present an extension of the repetitive branch-and-bound algorithm for the well-known minimum sum of squares unconstrained clustering to constrained clustering. We show that this algorithm can be combined with constraint programming and the result outperforms all existent exact approaches. On the application side, we analyze the use of our framework in the context of an application of constrained clustering on time-series images. This raises new questions and challenges that need to be considered.

Chapter 6 concludes and discusses perspectives on future work. Appendix A reviews my work on the two previous topics: solving first order logic constraints in tree structures and learning constraint propagators in finite domain constraint solvers. My publications are given in List of Publications.

Chapter 2

Constraint Programming

Contents

The research topics that will be developed in this dissertation are based on constraint programming. This chapter presents some basics of constraint programming. We highlight the importance of modeling, of global constraints and search strategies.

2.1 Modeling Using Constraints

Constraint Programming (CP) is a powerful paradigm for solving combinatorial problems, based on a wide range of methods from Artificial Intelligence, Computer Science or Operational Research. Using CP to solve a problem, the user needs to formalize the problem in a Constraint Satisfaction Problem or a Constraint Optimization Problem. CP solvers then search for a solution, or all the solutions, or the best one.

Definition 2.1 *A* Constraint Satisfaction Problem (CSP) *is a triple* $\langle X, Dom, C\rangle$ *where:*

- $X = \langle x_1, x_2, \ldots, x_n\rangle$ *is a n-tuple of variables,*
- $Dom = \langle Dom(x_1), Dom(x_2), \ldots, Dom(x_n)\rangle$ *is a corresponding n-tuple of domains such that* $x_i \in Dom(x_i)$*,*
- $C = \{C_1, C_2, ..., C_e\}$ *is a set of constraints where each constraint* C_i *expresses a condition on a subset of* X*.*

A solution of a CSP is a complete assignment of a value from $Dom(x_i)$ *to each variable* x_i *such that all the constraints of* C *are satisfied. A* Constraint Optimization Problem (COP) *is a CSP with an objective function* F *to be optimized. An optimal solution of a COP is a solution of the CSP that gives the best value for the objective function.*

Modeling using CP. Constraint programming eases problem modeling since a wide range of variables and constraints are available. The variable domains can be discrete or continuous, they can be expressed by a set of values or by ranges limited by an upper and a lower bounds. The value of the variables can be integers, floating point values,

sets or can be taken from structured domains such as graphs. Constraints can be *elementary constraints* including arithmetic relations (e.g. $X \neq Y$, $X + 3Y \geq Z$), or *global constraints* that represent more complex n-ary relations (e.g. *atmost*$(X, 5, 10)$ states that there must be at most 5 variables in the variable array X that have their value equals to 10). Constraints can also be *reified constraints* that associate a $0/1$ variable x with a constraint c, so that x takes the value 1 if the constraint c is satisfied and 0 otherwise. This kind of constraints enables the expression of logical combinations of constraints (e.g. $Y > Z \rightarrow atmost(X, 5, 10)$). For solving a practical problem using constraint programming, the problem must be modeled as a CSP or a COP. This means the need of: (1) defining the variables and their domains, (2) specifying the constraints to fully describe the problem and (3) defining the objective function for optimization problems. A problem can be viewed from different points of view and therefore can correspond to different choices of variables and domains (viewpoints). For solving a practical problem using constraint programming, deciding an appropriate viewpoint as well as expressing sufficient or useful constraints can have a real impact on the effectiveness of the solvers. While sufficient constraints give a complete expression of the problem, redundant but useful constraints help to better prune the search space. Useful constraints can be implied constraints or constraints to break symmetries among the solutions. A survey on different issues in modeling can be found in [Smith 2006].

Soving CSP/COP. In general, solving a CSP is NP-Hard. Nevertheless, methods developed in constraint programming enable to efficiently solve a large number of real applications. They rely on constraint propagation and search. The solving process can be summarized in Algorithm 1. In this algorithm, Happy() succeeds when one solution or all the solutions are found. Constraint_Propagation() propagates the constraints to ensure that each constraint reaches a required local consistency. It is done by reducing the domains of variables for each constraint, until a stable state of all the constraints. If after the propagation, a variable has its domain empty then the constraints cannot be satisfied; the solver backtracks in this case. If all the variable domains become singleton, then a solution is reached. Otherwise, Branching() creates sub-cases and the solving process is applied on each sub-case.

Algorithm 1: Solve(P): CSP Solving

```
1  while ¬ Happy() do
2      Constraint_Propagation()
3      if ∃x ∈ X s.t. Dom(x) = ∅ then
4          return Failure
5      else if Solution() then
6          return Solution
7      else
8          Branching()
9          foreach subcase P_i do
10             Solve(P_i)
```

A constraint optimization problem is a CSP with an objective function F to be optimized. For solving a COP, a branch-and-bound mechanism is usually integrated. At each time a solution s is reached, the value f_s of the objective function on the solution is computed. A new constraint that requires next solutions must be better than s is added, for instance the constraint is $F < f_s$ if F is to be minimized. The solver backtracks to other sub-cases with this added constraint, that implies the found solutions are improved one after other, until no more solution can be found. The last solution found is the best one.

2.2 Constraint Propagation

Constraint propagation is a central concept, perhaps *the* central concept, according to [Rossi *et al.* 2006], in the theory and practice of constraint programming. An extensive survey on constraint propagation can be found in [Bessiere 2006].

Let c be a constraints on the variables $X_c \subseteq X$. Constraint propagation of the constraint c reduces the domain of the variables of X_c, by removing some or all inconsistent values, *i.e.*, values that cannot be part of a solution of c. The reduction ensures local consistency properties that define necessary condition on values or set of values to belong to a solution. Constraint propagation is performed by two main approaches: the rule iteration approach and the algorithmic approach. In the rule iteration approach, reduction rules specify conditions under which domain reductions can be performed for a constraint. In the algorithmic approach, filtering algorithms are designed for constraints to reduce domains.

The propagation of a constraint is performed until no more reduction can be done. The constraint is then locally consistent. All the constraints are propagated until a stable state, the CSP is then locally consistent. Many types of local consistency have been studies in constraint programming, including node consistency, (generalized) arc consistency and path consistency. The simplest type is node consistency which requires that each unary constraint defined on a variable x must be satisfied by all values in $Dom(x)$. Arc consistency was defined initially for binary constraint and has been generalized for arbitrary constraint. Generalized arc consistency is currently the most important local consistency in practice and has received attention of lots of work.

Definition 2.2 *(Generalized arc consistency) Let c be a constraint on the variables* $x_1, \ldots, x_k$ *with respective domains* $Dom(x_1), \ldots, Dom(x_k)$*. That is,* $c \subseteq \Pi_i Dom(x_i)$*. Constraint c is generalized arc consistent (arc consistent, for short) if for every* $1 \leq i \leq k$ *and for every* $v \in Dom(x_i)$*, there exists a tuple* $(d_1, \ldots, d_k) \in c$ *such that* $d_i = v$*. A CSP is arc consistent if each of its constraints is arc consistent.*

The most well known algorithm for arc consistency is AC3 [Mackworth 1977]. It was proposed for binary constraints and was extended to generalized arc consistency. Algorithm 2 describes AC3/GAC3. This algorithm achieves arc consistency in $O(er^3d^{r+1})$ time and $O(er)$ space, where e is the number of constraints, d is the size of the largest domain and r is the greatest arity of the constraints. Several algorithms have been proposed to improve the time complexity: AC4, AC6, AC2001, etc.

Algorithm 2: AC3 / GAC3 [Bessiere 2006]

```
1  function Revise3 (x: variable, c: constraint): Boolean
2      CHANGE ← false
3      foreach v ∈ Dom(x) do
4          if ∄t ∈ c ∩ Π_{X_c}(Dom) with t|_x = v then
5              remove v from Dom(x)
6              CHANGE ← true
7      return CHANGE
8  function AC3 / GAC3: Boolean
9      Q ← {(x, c) | c ∈ C, x ∈ X_c}
10     while Q ≠ ∅ do
11         select and remove (x, c) from Q
12         if Revise(x, c) then
13             if Dom(x) = ∅ then return false;
14             else Q ← Q ∪ {(x', c') | c' ∈ C ∧ c' ≠ c ∧ x, x' ∈ X_{c'} ∧ x ≠ x'} ;
15     return true
```

For float value variable, the domain can be defined by intervals which are expressed by a minimum value $min(x)$ and a maximum value $max(x)$. Bound consistency requires that the minimum value and the maximum value must be part of a solution. A constraint c is bound consistent iff for each variable $x \in X_c$:

- there exists a tuple t satisfying c such that $t|_x = min(x)$ and $min(x') \leq t|_{x'} \leq max(x')$ for all the other $x' \in X_c$,
- there exists a tuple t satisfying c such that $t|_x = max(x)$ and $min(x') \leq t|_{x'} \leq max(x')$ for all the other $x' \in X_c$.

2.3 Global Constraints

Different kinds of constraints are available in constraint programming; they can be elementary constraints expressing arithmetic or logic relations, or global constraints expressing meaningful n-ary relations. A *global constraint* is a constraint that captures a relation between a non-fixed number of variables. One of the best known global constraints is the constraint *alldifferent*$(x_1, \ldots, x_n)$, which imposes the variables x_i to be pairwise different. The power of global constraints is two-fold. First, global constraints ease the task of modeling using constraint programming. They usually represent patterns that occur in applications. Second, global constraints benefit from efficient propagation, performed by a dedicated filtering algorithm. A *filtering algorithm* for a constraint c is an algorithm that filters the domains of variables with respect to c. The filtering algorithm for a global constraint is usually designed in taking advantage of the semantics of the constraint and is therefore much more efficient. A detailed survey on global constraints can be found in [van Hoeve & Katriel 2006], which shows that designing constraint propagation algorithm

for global constraints draws on a wide variety of disciplines including graph theory, linear programming and finite automaton.

Examples of Global Constraints. The strength of constraint programming is based on the power of global constraints and their filtering algorithm. A catalog of global constraints with more than 400 inventoried constraints is maintained in [Beldiceanu *et al.* 2014]. Various relations are expressed by global constraints, for instance the following ones:

- Element constraint: Let y be an integer variable, z a variable with finite domain, and $[x_1, \ldots, x_n]$ an array of variables. The constraint *element*$(y, z, x_1, \ldots, x_n)$ states that z is equal to the y-th variable in x, i.e. $z = x_y$ or $z = x[y]$.

- Cardinality constraint: Let y, z be variables, $[x_1, \ldots, x_n]$ an array of variables, θ be an arithmetic relation, e.g. $\geq$. Cardinality constraints state the relations:

$$\#\{i \in \{1, .., n\} \mid x_i = y\} \ \theta \ z$$

  These constraints include *atmost* (with θ being $\leq$) or *atleast* (with θ being $\geq$).

- Global cardinality constraint: Let $[x_1, \ldots, x_n]$ be an array of assignment variables whose domains are contained in $\{v_1, \ldots, v_m\}$ and let $\{c_{v_1}, \ldots, c_{v_m}\}$ be count variables whose domains are sets of integers. The global cardinality constraint *gcc*$(x_1, \ldots, x_n, c_{v_1}, \ldots, c_{v_m})$ states that each value v_i must be assigned to exactly c_{v_i} assignment variables in $x_1, \ldots, x_n$.

  This constraint can be seen as a generalization of *alldifferent*$(x_1, \ldots, x_n)$, where the domain of each count variable is $\{0, 1\}$.

- Global cardinality constraint with cost: This constraint combines a global cardinality constraint *gcc* and a variant of the *sum* constraint. Let $X = \{x_1, \ldots, x_n\}$ be a set of assignment variables and let $c_{v_1}, \ldots, c_{v_m}$ be count variables. Let w be a function that associates to each pair $(x, d) \in X \times Dom(X)$ a "cost" $w(x, d) \in \mathbb{Q}$. Let z be a "cost" variable and let us assume that z is to be *minimized*. The global cardinality constraint with costs is defined as:

$$\begin{aligned} cost\_gcc(x_1, \ldots, x_n, c_{v_1}, \ldots, c_{v_m}, z, w) = \{(d_1, \ldots, d_n, o_1, \ldots, o_m, d) \mid \\ \forall i \ d_i \in Dom(x_i), d \in Dom(z), \\ (d_1, \ldots, d_n, o_1, \ldots, o_m) \in gcc(x_1, \ldots, x_n, c_{v_1}, \ldots, c_{v_m}), \\ \textstyle\sum_{i=1}^{n} w(x_i, d_i) \leq d\} \end{aligned}$$

  The cost variable z represents here an upper bound on the sum of $w(x_i, d_i)$ for all i. This global constraint is therefore used in a constraint optimization problem, where the cost variable z represents the objective function.

- Set constraints: An example of global constraint on set variables is the *atmostOne* constraint. Let $[x_1, \ldots, x_n]$ be an array of set variables, each variable represents a set of fixed cardinality c. The constraint *atmostOne*$(x_1, \ldots, x_n, c)$ is defined as:

$$\forall 1 \leq i \leq n, \ |x_i| = c \ \text{ and } \forall 1 \leq i < j \leq n, \ |x_i \cap x_j| \leq 1$$

Filtering Algorithms. From a logical point of view, a global constraint is equivalent to a conjunction of elementary constraints, e.g. the constraint *alldifferent*(x_1, x_2, x_3) is equivalent to the conjunction of binary constraints $x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3$. The interesting point is that a global constraint with its filtering algorithm has much more powerful propagation than the conjunction of elementary constraints; the filtering algorithm can detect inconsistency earlier than the set of propagators of the elementary constraints. Moreover, filtering algorithms for global constraints take advantage from operational research or graph theory to achieve generalized arc consistency or bound consistency with a lower complexity. A filtering algorithm of a constraint c can remove all the inconsistent values from the domain of every variables of c or only some of them. It it removes all the inconsistent values, it achieves complete filtering, otherwise it performs partial filtering. The filtering algorithm for the *alldifferent* constraint, based on matching theory, performs a complete filtering to achieve arc consistency.

Example 1 [van Hoeve & Katriel 2006] Consider the following CSP:

$$x_1 \in \{a, b, c, d, e\}, x_2 \in \{b, c\}, x_3 \in \{a, b, c, d\}, x_4 \in \{b, c\}$$
$$\textit{alldifferent}(x_1, x_2, x_3, x_4)$$

This CSP is equivalent to a CSP with 6 elementary constraints $x_i \neq x_j$, for $1 \leq i < j \leq 4$. The arc consistency for each individual constraint $x_i \neq x_j$ cannot remove any value from the domains $Dom(x_i)$ and $Dom(x_j)$, since each value is part of a solution. The filtering algorithm for the constraint *alldifferent* [Régin 1994] maintains the bipartite graph $G = (V, E)$, with $V = X \cup \bigcup_i Dom(x_i)$ and $E = \{(x_i, v) \mid v \in Dom(x_i)\}$. This bipartite graph, which is also called the value graph of X, is given in Figure 2.1 (left). A matching $M \subseteq E$ is a set of disjoint edges, *i.e.* two edges in M cannot share a vertex. Two important observations on the relationship between the constraint *alldifferent*$(x_1, \ldots, x_n)$ and matching were introduced in [Régin 1994]:

- There is a matching of cardinality n (maximum-cardinality) if and only if the constraint *alldifferent*$(x_1, \ldots, x_n)$ is satisfiable.
- An edge (x_i, v) belongs to a matching of cardinality n if and only if the value v is consistent with the constraint.

It is proven in matching theory that given a graph G and a maximum-cardinality matching M in G, an edge e belongs to some maximum-cardinality matching in G iff $e \in M$, or e is on an even-length M-alternating path starting at an M-free vertex, or e is on an even-length M alternating circuit. Using this result, the arc consistency algorithm for *alldifferent*$(x_1, \ldots, x_n)$ was constructed as follows. First, a maximum-cardinality matching M of the value graph G is constructed. This can be done in $O(m\sqrt{n})$ time, where $m = \sum_{i=1}^{n} |Dom(x_i)|$. Next, the even M-alternating paths starting at an M-free vertex and the even M-alternating circuits are identified as below.

Define the directed bipartite graph $G_M = (V, A)$ with arc set $A = \{(x, d) \mid x \in X, \{x, d\} \in M\} \cup \{(d, x) \mid x \in X, \{x, d\} \in E \backslash M\}$. In other words, edges in M are oriented from X to $Dom(X)$ and edges not in M are oriented in reverse direction. The strongly connected components in G_M are computed in $O(n + m)$ time. Arcs between vertices in the same strongly connected component belong to an even M-alternating circuit in G,

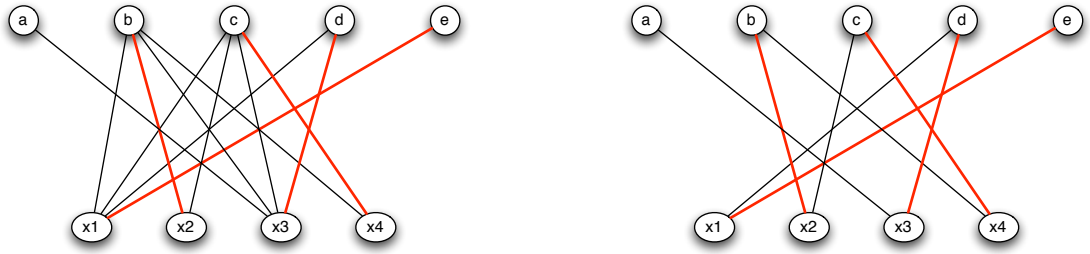


Figure 2.1: Value graph for the *alldifferent* constraint, before (left) and after (right) filtering. Bold edges represent a matching, corresponding to a solution of the *alldifferent* constraint.

and are marked as "used". Next, are searched the arcs that belong to a directed path in G_M, starting at an M-free vertex. This takes $O(m)$ time, using breadth-first search. Arcs belonging to such a path belong to an M-alternating path in G starting at an M-free vertex, and are marked as "used".

For all edges $\{x, d\}$ whose corresponding arc is not marked "used" and that do not belong to M, the value d is arc-inconsistent and therefore is removed from $Dom(x)$. Figure 2.1 (right) shows the value graph after establishing arc consistency. All the remaining edges are part of a maximum-cardinality matching. □

Optimization Constraints. In the context of constraint optimization problems, an optimization constraint is a global constraint that is linked to the objective function. Each solution induces a "cost" and the global constraint exploits this cost to filter not only the variable which represents the objective function, but also other decision variables inside the constraint. The first filtering algorithm for this kind of global constraints is proposed in [Focacci *et al.* 1999]. A well-known example of extension of global constraints to optimization constraints is the constraint *cost_gcc* [Régin 1999], which extends the Global Cardinality Constraint with cost.

2.4 Search

In a CP solver, two steps, constraint propagation and branching, are repeated until a solution is found. Constraints are propagated until a stable state, in which the domains of the variables are reduced as much as possible. If the domains of all the variables are reduced to singletons then a solution is found. If the domain of a variable becomes empty, then there exists no solution with the current partial assignment and the solver backtracks. In the other cases, the solver splits the state into sub-cases, thus leading to new branches in the search tree. The search strategy can be determined by the programmer. A backtracking search for a solution to a CSP can be seen as a depth-first traversal of the search tree. The solver orders branches following the order given by the programmer and explores in depth each branch, activating again constraint propagation [van Beek 2006].

The method of extending a node in the search tree is called a *branching strategy*. Some popular branching strategies are the following. Let x be the unassigned variable which is chosen to branch on.

- Enumeration: a branch is generated for each value in the domain of x.
- Binary branching: a value $v \in Dom(x)$ is chosen and two branches are generated, which correspond to the addition of two constraints $x = v$ and $x \neq v$.
- Domain splitting: $Dom(x)$ is an ordered domain, a value $v \in Dom(x)$ is chosen and two branches are created, which correspond to the addition of two constraints $x \leq v$ and $x > v$.

At each branching decisions must be made as to which variable to branch on and which value to choose. These decisions are referred to as the variable and value ordering. The choice of variables and of values at each branching is extremely important, since it may drastically reduce the search space and therefore computation time. Finding the best ordering is difficult, heuristics are therefore usually used. Various heuristics on variable ordering or value ordering have been designed, they can be static or dynamic. In a static ordering, the choice of variable/value is fixed prior to search, whereas in a dynamic ordering, the choice is determined during the search.

Variable ordering heuristics can be based on the domain sizes or on the structure of the CSP. A well-known heuristic based on the domain sizes is the fail-first principle: "to succeed, try first where you are most likely to fail" [Haralick & Elliott 1980]. A variable with the smallest number of values remaining in its domain is therefore chosen. Different generalizations and improvements of this heuristic have been done. Let the degree of an unassigned variable x be the number of constraints which involve x and at least one other unassigned variable. An heuristic combines the domain size and the degree, such as it chooses the variable with the smallest domain size and breaks any ties by choosing the variable with the highest degree. Another generalization is to divide the domain size of a variable by the degree of the variable and to choose the variable which has the minimal value [Bessière & Régin 1996]. For value ordering, heuristics can be the smallest/largest/median value or on estimating the probability of a solution.

For a constraint optimization problem, a branch-and-bound strategy can be integrated to a depth-first search [Van Hentenryck 1989]: each time a solution, *i.e.* a complete assignment of variables satisfying the constraints, is found, the value of the objective function for this solution is computed and a new constraint is added, expressing that a new solution must be better than this one. Assume that the objective function is represented by a variable y, which is to be minimized. When a solution to the problem is found, its corresponding objective value f is computed and the constraint $y < f$ is added. This process is repeated until the resulting CSP is unsatisfiable, in which case the last solution found has been proven to be optimal. This is therefore important for the effectiveness that constraint propagation operates on the objective variable.

2.5 Summary

This chapter presents some main principles of constraint programming. More detailed on different aspects of constraint programming can be found in [Rossi *et al.* 2006]. Constraint programming systems provide wide services such as constraint propagation and backtracking search. For many applications, the provided services are sufficient. However, some problems require more dedicated services. Most constraint programming systems are

extensible, allowing the user to define new constraint propagators or new search strategies. This flexibility strengthen the power of constraint programming to solve different combinatorial problems.

Chapter 3

Property Grammars Parsing Using Constraints

Contents

In natural language processing, one important task is to analyze the syntax of utterances. An utterance of a natural language can be well-formed according to the grammatical requirements or can be not completely well-formed, yet showing some form of syntactic structure and properties. The former is referred as an expression, the latter is a quasi-expression. In our work, we have considered property grammars, where a grammar describing a language is given by a set of properties that must be satisfied by any expression of the language. We have introduced model-theoretic semantics of property grammars, which represents the syntactic analysis as an optimization problem [Duchier *et al.* 2009]. The models are trees of syntactic categories and are designed for being expressed by constraints. We have developed a formalization presented as a constraint optimization problem, which naturally leads to an implementation of a fully constraint-based parser for property grammars using constraint programming [Duchier *et al.* 2010a]. This framework allows not only to analyze grammatical utterances but also to find out one of the most appropriate syntactic structures for ungrammatical utterances. We have extended the framework in order to integrate other types of properties, which express relations between syntactic constituents by constraints on feature structures [Duchier *et al.* 2011, Duchier *et al.* 2014]. This work has been developed in collaboration with Denys Duchier, Willy Lesaint, Yannick Parmentier and Jean-Phillipe Prost.

The organization of this chapter is as follows. We present property grammars (PG) and PG parsing in Section 3.1. Our model-theoretic interpretation of PG based on trees of syntactic categories and the description of models for expressions and loose models for quasi-expressions are presented in Section 3.2. We present the formalization of PG parsing as a constraint optimization problem using constraint programming in Section 3.3 and we conclude in Section 3.4.

3.1 Problem and Context

3.1.1 Model-Theoretic Syntax and Property Grammars

In natural language processing, two categories of frameworks to describe grammars of natural languages emerged during the 20th century. They are called, such as proposed in [Pullum & Scholz 2001]: *Generative-Enumerative Syntax (GES)* and *Model-Theoretic Syntax (MTS)*. They are based on different sides of logic: GES is developed based on the syntactic side whereas MTS is based on the semantic side.

GES expresses a language as a set of legal strings. A GES grammar provides a set of production rules that enables to enumerate all the elements of the language, the vocabulary being the finite set of terminals. Lots of frameworks are of the GES type, as for instance all the types of phrase structure grammar in the Chomsky Hierarchy. MST frameworks emerged some time later, from developments on the semantic rather than the syntactic side of logic. MTS abstracts away from any specific procedure and focuses on describing syntactic properties of language. In other words, MTS takes a descriptive point of view on syntax, where a grammar is a finite set of axioms in a formal logic with a model-theoretic interpretation. The axioms are referred as *constraints*. The models of the constraints are the expressions that are described by the grammar. A well-formed expression must be a model of the theory, i.e. it satisfies the set of all unordered grammatical constraints.

Pullum and Scholz emphasized that "expressions, not sets of expressions, are the models for an MTS grammars: an individual expression either satisfies or does not satisfy a grammar. An MTS grammar does *not* recursively define a set of expressions; it merely states necessary conditions on the syntactic structure of individual expressions." [Pullum & Scholz 2001]. While the syntactic representation of a string is, in GES, the mere trace of the generative procedure, in MTS it is a model for the grammar, which no information as to how such a model might be obtained. The requirement to be a model for the grammar is to satisfy the set of all the constraints expressing the grammar.

When MTS is compared with GES, the consequences in terms of coverage on linguistic phenomena is significant. Pullum and Scholz have shown that a number of phenomena, which are not accounted for by GES, are well covered in MTS frameworks. Most noticeably, quasi-expressions and graded grammaticality judgements are only covered by MTS. In natural languages there are usually utterances that are not completely well-formed, yet they are partially well-formed and almost like expressions. They are called *quasi-expressions*. Among the quasi-expressions, some are closer to being grammatical than others. Therefore, any framework for describing syntactic structure that can also describe degrees of ungrammaticality for quasi-expressions is to be preferred to one that cannot. MST grammars offer elegant ways to achieve it.

Property Grammars (PG) was initially defined in [Blache 2000]. They are of the MTS

Property	Form	Meaning
Obligation	$A : \triangle B$	at least one node labeled with B
Uniqueness	$A : B!$	at most one node labeled with B
Linearity	$A : B \prec C$	every node labeled with B precedes every node labeled with C
Requirement	$A : B \Rightarrow C$	if $\exists$ a node labeled with B, then $\exists$ one labeled with C
Exclusion	$A : B \not\Rightarrow C$	there are no two daughter nodes labeled resp. with B and C
Constituency	$A : S?$	all children have their labels $\in S$
Agreement	$A : B \rightsquigarrow C$	coreference constraints between categories

Table 3.1: Usual types of properties in Property Grammars

category; a grammar is defined by a collection of statements about the language, which are called properties. A *property* is a constraint that expresses a relationship among syntactic categories. The properties come from linguistic observations, such as word order, co-occurrence, number or gender agreement, etc. In a first approximation, they can be seen as local constraints on categories labeling syntactic trees. A property is of the form $A : \psi$, which specifies for each node labeled with category A, the constraint ψ to be applied on the categories labeling the daughter nodes (these categories are written B, C hereafter). The usual types of properties are given in Table 3.1 (here S is a set of labels). Property grammars are appealing for modeling deviant utterances because they break down the notion of grammaticality into many small constraints (properties) which can be independently violated.

Property grammars are perhaps best understood as the transposition of phrase structure grammars from the GES perspective into the MTS perspective. Let us consider a phrase structure grammar expressed as a collection of rules. As an example, let us consider the context free rules $\texttt{NP} \rightarrow \texttt{D N}$ and $\texttt{NP} \rightarrow \texttt{N}$ describing the relation between a noun and a determiner. They can be translated into the 7 following properties: (1) noun phrases only contain nouns or determiners, (2) in a noun phrase, there is at most one determiner, (3) in a noun phrase, there is at least one noun, (4) in a noun phrase, there is at most one noun, (5) in a noun phrase, a determiner precedes a noun, (6) and (7) determiners and nouns have no daughter nodes labeled with categories (i.e., non-terminal symbols) in a valid syntactic tree. In this manner, rules have become constraints and a phrase structure grammar can be given in model-theoretical semantics by interpretation over syntax tree structures. However these constraints remain very coarse-grained:

$$(1)\ \texttt{NP} : \{\texttt{D}, \texttt{N}\}? \quad (2)\ \texttt{NP} : \texttt{D}! \quad (3)\ \texttt{NP} : \triangle\texttt{N} \quad (4)\ \texttt{NP} : \texttt{N}!$$
$$(5)\ \texttt{NP} : \texttt{D} \prec \texttt{N} \quad (6)\ \texttt{D} : \{\}? \quad (7)\ \texttt{N} : \{\}?$$

In this context, if we only consider syntactic trees whose roots have category NP, there are only two satisfying all the properties:

We notice that these syntactic trees are not lexicalized. In case we want to describe lexicalized trees, we can add some more lexical properties, such as $\texttt{cat}(apple) = \texttt{N}$ which defines the word *apple* as being a noun.

3.1.2 Property Grammars Parsing

Deep parsing with property grammars has been shown to be theoretically exponential in the number of categories of the grammar and the size of the sentence to parse [van Rullen 2005]. Existing approaches usually rely on heuristics to reduce complexity in practice. Among the different approaches, the seminal work is [Blache & Balfourier 2001]. This work was later followed by several ones [Dahl & Blache 2004, Estratat & Henocque 2004, van Rullen 2005, van Rullen *et al.* 2006, Blache & Rauzy 2006], and more recently [Prost 2008].

Most of these works do not rely on a model-theoretic formal semantics of property grammars. They rather apply well-known efficient parsing techniques and heuristics to property grammars. In [Blache & Balfourier 2001], a constraint selection process is used to incrementally build the syntactic tree of a sentence. Hybrid approaches mixing deep and shallow parsing are used in [van Rullen 2005, van Rullen *et al.* 2006]. In [Blache & Rauzy 2006], the authors propose to extend symbolic parsing with probabilities on syntactic categories.

A first attempt to use a constraint satisfaction-based approach is [Dahl & Blache 2004]. In this work, the input property grammar is encoded into a set of rules for the Constraint Handling Rule system [Frühwirth 2009]. The encoding makes it possible to directly interpret the grammar in terms of satisfied/relaxed constraints on syntactic categories. On top of this interpretation, rewriting rules are used to propagate constraint satisfaction/relaxation, and a syntactic tree is built as a side effect. The way a constraint is selected for evaluation is therefore controled, thus the constraint satisfaction problem is not distinguished from its resolution.

Another constraint-based approach is [Estratat & Henocque 2004]. In this work, a grammar is translated into a model in the Object Constraint Language (OCL). This model is interpreted as a configuration problem, which is fed to a configurator. The latter solves the constraints lying in the input model. The result is a valid syntactic structure. In this approach, the OCL-encoding does not allow for relaxed constraints. Hence, it only computes syntactic structures that satisfy the whole set of constraints. In other terms, it cannot make full advantage of the property grammars formalism, which describes natural language in terms of local constraints that can be violated. This feature is particularly useful when dealing with ungrammatical sentences such as those spoken languages often contain.

In his PhD thesis, Jean-Philippe Prost has developed a framework based on first-order logic with model-theoretic semantics for gradience and a parsing algorithm for possibly deviant utterances [Prost 2008]. The parsing algorithm is chart-based algorithm, which uses the dynamic programming approach: optimal sub-trees for parts of the utterance are computed and stored in a structure called a chart, they are used to derive a complete syntactic tree. The formalization is however not entirely satisfactory: among other things,

the models were not trees, but technical devices suggested by the algorithmic approach to parsing.

In our approach, we aim at developing a system that is fully based on constraints. The interest of such a system is that informations from different aspects of the language could be integrated as soon as they can be expressed using constraints. This line of approaches was developed and showed its interest for Dependency Grammar [Duchier & Debusmann 2001, Duchier 2003] and Extensible Dependency Grammar [Debusmann *et al.* 2004].

In our work, we develop model-theoretic semantics that interpret property grammars over syntactic tree structures. The formulation enables considering model construction as a combinatorial search problem. Property grammar parsing can be therefore seen as the search for solutions of a constraint optimization problem. We rely on classic constraint-based techniques such as branch-and-bound and constraint propagation. That is, we clearly distinguish the definition of the constraint based problem from its resolution.

3.2 Model-Theoretic Semantics for Property Grammars

In this section, we develop model-theoretic semantics for property grammars. The interpretations are expressed by syntactic trees. We characterize a mechanism to determine (strong) models and generalize it to define loose models for quasi-expressions. This work was developed in collaboration with Denys Duchier et Jean-Philippe Prost and was published in [Duchier *et al.* 2009].

3.2.1 Domain of Interpretation

Let $\mathcal{L}$ be a finite set of labels representing syntactic categories. We write $\mathcal{P}$ for the set of all possible property literals over $\mathcal{L}$ formed in any of the following ways: $\forall c_0, c_1, c_2 \in \mathcal{L}$,

$$c_0 : c_1 \prec c_2, \quad c_0 : \triangle c_1, \quad c_0 : c_1!, \quad c_0 : c_1 \Rightarrow c_2, \quad c_0 : c_1 \not\Rightarrow c_2, \quad c_0 : s_1?,$$

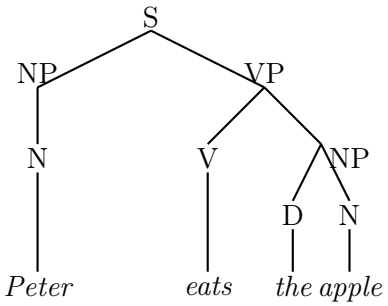
Let $\mathcal{S}$ be a set of elements called *words*. A lexicon is a subset of $\mathcal{L} \times \mathcal{S}$.[1] A *property grammar* G is a pair (P_G, L_G) where P_G is a set of properties (a subset of $\mathcal{P}$) and L_G is a lexicon. The strong semantics of property grammars is given by interpretation over the class of syntax tree structures defined below.

We write $\mathbb{N}_1$ for $\mathbb{N} \setminus \{0\}$. A *tree domain* D is a finite subset of $\mathbb{N}_1^*$ which is closed for prefixes and for left-siblings; in other words it satisfies:

$$\begin{array}{ll} \forall \pi, \pi' \in \mathbb{N}_1^* & \pi\pi' \in D \;\Rightarrow\; \pi \in D \\ \forall \pi \in \mathbb{N}_1^*,\ \forall i, j \in \mathbb{N}_1 & i < j \wedge \pi j \in D \;\Rightarrow\; \pi i \in D \end{array}$$

A node in a tree domain D is therefore characterized by a path. For instance the root is ϵ, its left-first daughter is 1, the first daughter of this daughter is 11, etc. A *syntax tree* $\tau = (D_\tau, L_\tau, R_\tau)$ consists of a tree domain D_τ, a labeling function $L_\tau : D_\tau \to \mathcal{L}$ assigning a category to each node of the tree, and a function $R_\tau : D_\tau \to \mathcal{S}^*$ assigning to each node its surface realization. For instance, the following syntax tree:

[1] We restricted ourselves to the simplest definition sufficient for this presentation.



corresponds to $D_\tau = \{\epsilon, 1, 2, 11, 21, 22, 221, 222\}$, $L_\tau(\epsilon) = \texttt{S}$, $L_\tau(1) = \texttt{NP}$, $L_\tau(2) = \texttt{VP}$, $L_\tau(221) = \texttt{D}$, $R_\tau(\epsilon) = \textit{Peter eats the apple}$ and $R_\tau(22) = \textit{the apple}$.

For convenience, we define the arity function $A_\tau : D_\tau \to \mathbb{N}$ as follows: $\forall \pi \in D_\tau$,

$$A_\tau(\pi) = \max(\{0\} \cup \{i \in \mathbb{N}_1 \mid \pi i \in D_\tau\})$$

In the previous example, we have $A_\tau(\epsilon) = 2$, $A_\tau(1) = 1$ and $A_\tau(22) = 2$.

3.2.2 Instances, Pertinence and Satisfaction

A property grammar G stipulates a set of properties. For example the property $c_0 : c_1 \prec c_2$ is intended to mean that, for a non-leaf node of category c_0, and any two daughters of this node labeled respectively with categories c_1 and c_2, the one labeled with c_1 must precede the one labeled with c_2. Clearly, for each node of category c_0, this property must be checked for every pair of its daughters. Thus, we arrive at the notion of instances of a property, where the property may need to be checked.

Instances. For a grammar G and a syntax tree τ, each property corresponds to a set of instances defined on τ. An instance of a property is a pair composed by the property and a tuple of nodes (paths) to which it is applied. We define the property instances of G on a syntax tree τ as follows:

$$\mathcal{I}_\tau[\![G]\!] = \cup\{\mathcal{I}_\tau[\![p]\!] \mid \forall p \in P_G\}$$

For a property p, by the set $\mathcal{I}_\tau[\![p]\!]$ we generate all the possible tuples of each node with 0, 1 or 2 of its daughters, depending on the definition of p, on which p may need to be checked:

$$\begin{aligned}
\mathcal{I}_\tau[\![c_0 : c_1 \prec c_2]\!] &= \{(c_0 : c_1 \prec c_2)@\langle \pi, \pi i, \pi j\rangle \mid \forall \pi, \pi i, \pi j \in D_\tau,\ i \neq j\} \\
\mathcal{I}_\tau[\![c_0 : \triangle c_1]\!] &= \{(c_0 : \triangle c_1)@\langle \pi\rangle \mid \forall \pi \in D_\tau\} \\
\mathcal{I}_\tau[\![c_0 : c_1!]\!] &= \{(c_0 : c_1!)@\langle \pi, \pi i, \pi j\rangle \mid \forall \pi, \pi i, \pi j \in D_\tau,\ i \neq j\} \\
\mathcal{I}_\tau[\![c_0 : c_1 \Rightarrow c_2]\!] &= \{(c_0 : c_1 \Rightarrow c_2)@\langle \pi, \pi i\rangle \mid \forall \pi, \pi i \in D_\tau\} \\
\mathcal{I}_\tau[\![c_0 : c_1 \not\Rightarrow c_2]\!] &= \{(c_0 : c_1 \not\Rightarrow c_2)@\langle \pi, \pi i, \pi j\rangle \mid \forall \pi, \pi i, \pi j \in D_\tau,\ i \neq j\} \\
\mathcal{I}_\tau[\![c_0 : s_1\texttt{?}]\!] &= \{(c_0 : s_1\texttt{?})@\langle \pi, \pi i\rangle \mid \forall \pi, \pi i \in D_\tau\}
\end{aligned}$$

Pertinence. Since we created instances of all properties in P_G for all nodes in τ, we must distinguish the instances which are truly pertinent from those which are not. An instance is pertinent when the property needs to be checked on the tuple of nodes. For instance, the instance $(c_0 : \triangle c_1)@\langle\pi\rangle$ is pertinent if the label of π is c_0, since the property $c_0 : \triangle c_1$ (at least one daughter node of label c_1) on the node π needs to be checked. The pertinence is defined by the predicate P_τ over instances as follows:

$$\begin{aligned}
P_\tau((c_0 : c_1 \prec c_2)@\langle\pi, \pi i, \pi j\rangle) &\equiv L_\tau(\pi) = c_0 \ \wedge\ L_\tau(\pi i) = c_1 \ \wedge\ L_\tau(\pi j) = c_2 \\
P_\tau((c_0 : \triangle c_1)@\langle\pi\rangle) &\equiv L_\tau(\pi) = c_0 \\
P_\tau((c_0 : c_1!)@\langle\pi, \pi i, \pi j\rangle) &\equiv L_\tau(\pi) = c_0 \ \wedge\ L_\tau(\pi i) = c_1 \ \wedge\ L_\tau(\pi j) = c_1 \\
P_\tau((c_0 : c_1 \Rightarrow c_2)@\langle\pi, \pi i\rangle) &\equiv L_\tau(\pi) = c_0 \ \wedge\ L_\tau(\pi i) = c_1 \\
P_\tau((c_0 : c_1 \not\Rightarrow c_2)@\langle\pi, \pi i, \pi j\rangle) &\equiv L_\tau(\pi) = c_0 \ \wedge\ (L_\tau(\pi i) = c_1 \ \vee\ L_\tau(\pi j) = c_2) \\
P_\tau((c_0 : s_1?)@\langle\pi, \pi i\rangle) &\equiv L_\tau(\pi) = c_0
\end{aligned}$$

Satisfaction. A property when checked on a tuple of nodes can be satisfied or not. A pertinent instance can therefore be satisfied or not. For instance, the instance $(c_0 : \triangle c_1)@\langle\pi\rangle$ is satisfied if one among the daughter nodes of π is labeled by c_1. The satisfaction of instances is defined by the predicate S_τ over instances as follows:

$$\begin{aligned}
S_\tau((c_0 : c_1 \prec c_2)@\langle\pi, \pi i, \pi j\rangle) &\equiv i < j \\
S_\tau((c_0 : \triangle c_1)@\langle\pi\rangle) &\equiv \vee\{L_\tau(\pi i) = c_1 \mid 1 \leq i \leq A_\tau(\pi)\} \\
S_\tau((c_0 : c_1!)@\langle\pi, \pi i, \pi j\rangle) &\equiv i = j \\
S_\tau((c_0 : c_1 \Rightarrow c_2)@\langle\pi, \pi i\rangle) &\equiv \vee\{L_\tau(\pi j) = c_2 \mid 1 \leq j \leq A_\tau(\pi)\} \\
S_\tau((c_0 : c_1 \not\Rightarrow c_2)@\langle\pi, \pi i, \pi j\rangle) &\equiv L_\tau(\pi i) \neq c_1 \ \vee\ L_\tau(\pi j) \neq c_2 \\
S_\tau((c_0 : s_1?)@\langle\pi, \pi i\rangle) &\equiv L_\tau(\pi i) \in s_1
\end{aligned}$$

For a grammar G and a syntax tree τ, we denote by $I^0_{G,\tau}$ for the set of pertinent instances of G in τ, $I^+_{G,\tau}$ for its subset that is satisfied, and $I^-_{G,\tau}$ for its subset that is violated:

$$\begin{aligned}
I^0_{G,\tau} &= \{r \in \mathcal{I}_\tau[\![G]\!] \mid P_\tau(r)\} \\
I^+_{G,\tau} &= \{r \in I^0_{G,\tau} \mid S_\tau(r)\} \\
I^-_{G,\tau} &= \{r \in I^0_{G,\tau} \mid \neg S_\tau(r)\}
\end{aligned}$$

Therefore $|I^+_{G,\tau}|$ is the number of property instances that are satisfied by τ and $|I^-_{G,\tau}|$ the number of property instances violated by τ.

Admissibility. A syntax tree τ is admissible as a candidate model for grammar G iff it satisfies the projection property, i.e. $\forall\pi \in D_\tau$:

$$\begin{aligned}
A_\tau(\pi) = 0 \quad &\Rightarrow \quad \langle L_\tau(\pi), R_\tau(\pi)\rangle \in L_G \\
A_\tau(\pi) \neq 0 \quad &\Rightarrow \quad R_\tau(\pi) = \sum_{i=1}^{A_\tau(\pi)} R_\tau(\pi i)
\end{aligned}$$

where $\sum$ represents here the concatenation of sequences. In other words: leaf nodes must conform to the lexicon, and interior nodes pass upward the ordered realizations of their daughters. Note that the admissibility does not take into account properties of P_G, any syntax tree that respects the projection property is admissible. We write $\mathcal{A}_G$ for the set of admissible syntax trees for grammar G.

3.2.3 Strong and Loose Models

Strong models. A strong model must satisfy all the properties of the grammar. A syntax tree $\tau = (D_\tau, L_\tau, R_\tau)$ is a strong model of a property grammar G if and only if it is admissible and $I^-_{G,\tau} = \emptyset$, i.e. no property instance is violated. We write $\tau : \sigma \models G$ iff τ is a strong model of G with realization σ, i.e. such that $R_\tau(\varepsilon) = \sigma$.

Loose models. Since property grammars are intended to also account for deviant utterances, we must define alternate semantics that accommodate deviations from the *strong* interpretation. The *loose semantics* will allow some property instances to be violated, but will seek syntax trees which maximize the overall *fitness* for a specific utterance.

A syntax tree τ is loosely admissible for utterance σ iff it is admissible and its realization is $\sigma = R_\tau(\epsilon)$. We write $\mathcal{A}_{G,\sigma}$ for the loosely admissible syntax trees for utterance σ:

$$\mathcal{A}_{G,\sigma} = \{\tau \in \mathcal{A}_G \mid R_\tau(\epsilon) = \sigma\}$$

Following [Prost 2008], we define fitness as the ratio of satisfied pertinent instances over the total number of pertinent instances:

$$F_{G,\tau} = I^+_{G,\tau} / I^0_{G,\tau}$$

The loose models for an utterance σ are all loosely admissible models for utterance σ that maximize the fitness:

$$\tau : \sigma \models\!\!\approx G \quad \text{iff} \quad \tau \in \underset{\tau' \in \mathcal{A}_{G,\sigma}}{\operatorname{argmax}}(F_{G,\tau'})$$

Looses models can be used to formalize judgements of acceptability. Given an utterance (expression or quasi-expression), the judgment of acceptability aims at estimating the degree of grammatical acceptability for its model. For modeling natural judgements of acceptability, we hypothesize that an estimate for acceptability of an utterance can be predicted by quantitative factor derivable from the strong/loose model. In [Prost 2008], Jean-Phillippe Prost has proposed scoring functions to quantitatively measure the judgement of acceptability based on the satisfaction and violation of properties. These measures have been shown to well agree with human judgements. We show in [Duchier *et al.* 2009] that these measures can be formulated in terms of property instances.

3.3 Property Grammars Parsing Seen as a Constraint Optimization Problem

In this section, we present a formulation of the model-theoretic semantics of property grammars as a constraint optimization problem. First, we need to define a data structure to represent candidate tree models. Since we do not know *a priori* the number of nodes

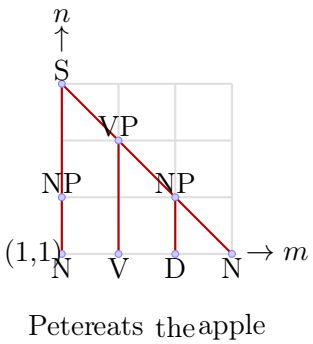


Figure 3.1: Parse tree laid on a grid

in the solution models, we propose to use a grid as a substrate and to enumerate the trees which can be laid out on this grid. We then show how to convert the pertinence and satisfaction of property instances as well as model fitness defined earlier into constraints to be applied on the labels of the nodes of grid-based trees. The model is extended to handle feature-based properties. This work was developed in collaboration with Denys Duchier, Willy Lesaint, Yannick Parmentier and was published in [Duchier *et al.* 2010a, Duchier *et al.* 2010b, Duchier *et al.* 2011, Duchier *et al.* 2012, Duchier *et al.* 2014].

3.3.1 Representing Tree Models Using a Grid

Our approach needs to enumerate candidate tree models, and to retain only those of maximal *fitness*. We use a grid as a substrate and define constraints to enforce that each tree model is laid out on this grid in a unique way. Tree models can therefore be enumerated using this grid.

For an utterance of m words, we know that each tree model has m leaves (property grammars do not use ϵ nodes). Unfortunately, we do not know the maximum depth of each tree model. Due to the intrinsic recursive nature of language, the possibility to find an adequate depth value, i.e. not too big to prevent useless computations, and not too small to avoid missing solutions, is an open question. We may use some heuristics to automatically assign a value n to the tree depth, but we rather parametrize the associated parsing problem with a maximum tree depth n. Fixing this parameter allows us to layout a model over a subset of the nodes of an $n \times m$ grid.

To represent our tree model, we will use a matrix $\mathcal{W}$ such that w_{ij} (with $1 \leq i \leq n$, and $1 \leq j \leq m$) refers to the node located at position (i, j) on the grid (rows and columns are numbered starting from 1, coordinate (1,1) being in the bottom-left corner). Figure 3.1 gives an illustration of such a layout. Let us now present the constraints used to build a tree model on an $n \times m$ grid.

Active nodes. We first need to distinguish among all the nodes of the grid the useful and the useless nodes, i.e. to distinguish between nodes that belong to the candidate tree model τ and nodes that do not.

Let $\mathcal{V}$ be the set of all nodes. A node is active if it is used by the model and inactive otherwise. We introduce two set variables V^+ and V^-, which represent the set of active

nodes and the set of inactive nodes, respectively. Their domain is $\mathcal{P}(\mathcal{V})$. A node has to either be active or inactive, thus we have the constraint:

$$\mathcal{V} = V^+ \uplus V^-$$

where $\uplus$ represents "disjoint union". Following the modeling technique of [Duchier 2003], for each node $w \in \mathcal{V}$, we define the set variables C_w (the set of its children), D_w^+ (the set of its descendants), D_w^* (the set of w *and* its descendants), P_w (the set of its parents), A_w^+ (the set of its ancestors) and A_w^+ (the set of w *and* its ancestors). Their domain is also $\mathcal{P}(\mathcal{V})$. Based on their definition, constraints relating these sets are:

$$D_w^+ = \uplus \{D_{w'}^* \mid w' \in C_w\} \qquad D_w^* = \{w\} \uplus D_w^+$$
$$A_w^+ = \uplus \{A_{w'}^* \mid w' \in P_w\} \qquad A_w^* = \{w\} \uplus A_w^+$$

Disjoint unions are justified by the fact that we are interested in tree models (*i.e.*, we do not allow for cycles). We additionally enforce the duality between ancestors and descendants:

$$w \in P_{w'} \;\Leftrightarrow\; w' \in C_w$$

and that each node has at most one parent:

$$|P_w| \leq 1$$

Inactive nodes have neither parents nor children:

$$w \in V^- \Rightarrow \quad C_w = P_w = \emptyset$$

Since the root of the tree is still unknown, we introduce a set variable R for the set of root nodes. A tree model must have a single root:

$$|R| = 1$$

The root node cannot be a child of any node, the children of two nodes are disjoint and all the children with the root are the active nodes:

$$V^+ = R \uplus (\uplus \{C_w \mid w \in V\})$$

Projection. We define additional constraints to ensure that there is no interleaving branches in a candidate tree model.

We define the projection of a given node w as the set of columns occupied by the tree rooted in w. For each variable $w \in \mathcal{V}$, we introduce a set variable Pr_w to denote the projection of w. As leaf nodes are located on the first row, their projection corresponds to exactly their column:

$$Pr_{w_{1j}} = \{j\} \qquad 1 \leq j \leq m$$

There are no interleaving projections (hence the disjoint union):

$$Pr_{w_{ij}} = \uplus \{Pr_w \mid w \in C_{w_{ij}}\} \qquad 1 < i \leq n, 1 \leq j \leq m$$

There are no holes in the projection of any node (trees are projective):

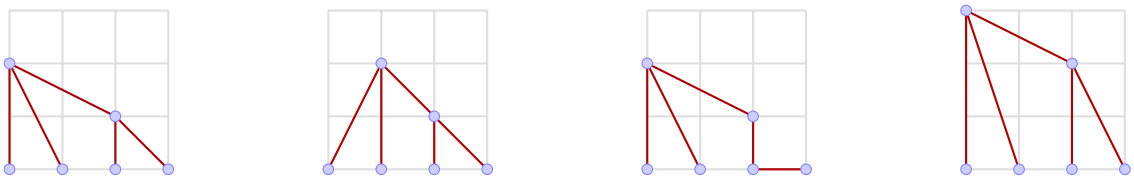
$$\mathsf{convex}(Pr_w) \qquad \forall w \in \mathcal{V}$$

Breaking symmetries. There are many ways of laying out a given tree on a grid. For instance, a four-node and three-leaf tree has among others the following layouts :

In order to have a unique way of laying out a tree, we add specific anti-symmetric constraints (the models satisfying these constraints are called *rectangular trees*) :

1. all leaves are located on the first row of the grid (*i.e.*, the bottom row),
2. the left-most daughter of any node is located on the same column as its mother node (this implies the subtree of a given node n occupies columns on the right of n),
3. every node is above any of its descendant nodes (this implies the subtree of a given node n occupies rows that are below that of n),
4. every internal node must have a daughter node in the row directly below (this implies there are no empty rows below the root node).

As an illustration, among the following trees, only the first one is a rectangular tree (the second tree violates condition 2, the third one condition 3 and the fourth one condition 4) :



How these 4 conditions are represented in our axiomatization ? First, let us write $c(w)$ for the column of node w and $\ell(w)$ for its line:

$$c(w_{ij}) = j \qquad\qquad \ell(w_{ij}) = i$$

(1) The words are linked to the bottom row of the grid, which contains the leaves of the tree. These must all be active:

$$\{w_{1j} \mid 1 \leq j \leq m\} \subseteq V^+$$

(2) Any active node must be placed in the column of the left-most leaf of its subtree:

$$w_{ij} \in V^+ \quad \Leftrightarrow \quad j = \min Pr_{w_{ij}}$$

This stipulation and the fact (3) every node is above of its descendants are translated by constraints on the domains of variables. As mentioned above, the descendants of a node n are on the down-right part of the grid with respect to n. The dual holds, that is the ancestors of a node n are on the upper-left part of the grid with respect to n:

$$D^+_{w_{ij}} \subseteq \{w_{lk} \mid 1 \leq l < i,\ j \leq k \leq m\}$$
$$A^+_{w_{ij}} \subseteq \{w_{lk} \mid i < l \leq n,\ 1 \leq k \leq j\}$$

(4) Any active non-bottom node has at least one child at the level just below:

$$w_{ij} \in V^+ \quad \Leftrightarrow \quad i - 1 \in \{\ell(w) \mid w \in C_{w_{ij}}\} \qquad\qquad 1 < i \leq n$$

Categories. The constraints given above ensure the form of trees. In order to model syntax trees, we also need to assign to each active node a syntactic category. For simplicity, we will assign the category $[f_1{:}\emptyset, \ldots, f_n{:}\emptyset]$ to all and only the inactive nodes:

$$\texttt{cat}(w) = [f_1{:}\emptyset, \ldots, f_n{:}\emptyset] \quad \Leftrightarrow \quad w \in V^-$$

For active nodes, the category will be assigned via property-related constraints, which are introduced in the next section. Finally, words are related to leaves via their category:

$$\texttt{cat}(w_{1j}) = \texttt{cat}(\text{word}_j)$$

where word_j refers to the j^{th} word of the sentence to parse.

3.3.2 Instances of Properties and Optimization Objective

Recall that each property has the form $A : \psi$, which means that for a node of category A, the constraint ψ applies to its children. For example the property $A : B \prec C$ is intended to mean that, for a non-leaf node of category A and any two daughters of this node labeled respectively with categories B and C, then the one labelled with B must precede the one labeled with C. Clearly, for each node of category A, this property must be checked for every pair of its daughters. This corresponds to the notion of instances of a property introduced earlier in Section 3.2.

An instance of a property is a pair of the property and a tuple of nodes to which it is applied. An instance is pertinent if the node where it is instantiated is active (*i.e.*, belongs to V^+) and the parameter nodes of its tuple have the categories stipulated in the property. An instance is satisfied if the property is satisfied. For each instance I we define two boolean variables $P(I)$ and $S(I)$ denoting respectively its *pertinence* and its *pertinence and satisfaction.*

In the following paragraphs, we describe the translation of properties of PG into a set of constraints for our constraint optimization problem. Properties are categorized into 3 types: (1) properties whose instance depends on a single node, (2) properties whose instance depends on a couple of nodes and (3) whose instance depends on a triple of nodes. The only property of type 1 is obligation $A : \triangle B$. The properties of type 2 are requirement $A : B \Rightarrow C$ and constituency $A : S?$. The properties of type 3 are linearity $A : B \prec C$, uniqueness $A : B!$ and exclusion $A : B \not\Leftrightarrow C$. For each type, we present the translation of one property, for the other the reader could see [Duchier *et al.* 2010a].

Properties of type 1. The property obligation $A : \triangle B$ yields instances I of the form:

$$(A : \triangle B)@\langle w_{i_0 j_0} \rangle$$

It is pertinent if $w_{i_0 j_0}$ is an active node labelled with A:

$$P(I) \quad \Leftrightarrow \quad (\, w_{i_0 j_0} \in V^+ \wedge cat(w_{i_0 j_0}) = A \,)$$

It is satisfied if at least one of its children is labelled with B:

$$S(I) \quad \Leftrightarrow \quad (\, P(I) \wedge \bigvee_{w_{ij} \in C_{w_{i_0 j_0}}} cat(w_{ij}) = B \,)$$

Properties of type 2. Let us describe the translation of the property requirement. The property $A : B \Rightarrow C$ yields instances I of the form:

$$(A : B \Rightarrow C)@\langle w_{i_0j_0}, w_{i_1j_1}\rangle$$

It is pertinent only if $w_{i_0j_0}$ is active and $w_{i_1j_1}$ is one of its children and their categories correspond:

$$P(I) \Leftrightarrow \left(\begin{array}{c} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in C_{w_{i_0j_0}} \wedge \\ \texttt{cat}(w_{i_0j_0}) = A \wedge \texttt{cat}(w_{i_1j_1}) = B \end{array} \right)$$

It is satisfied if one of $w_{i_0j_0}$'s children is labelled with C:

$$S(I) \quad \Leftrightarrow \quad (P(I) \wedge \bigvee_{w_{ij} \in C_{w_{i_0j_0}}} \texttt{cat}(w_{ij}) = C)$$

Properties of type 3. Properties of this type are linearity, uniqueness and exclusion. Let us describe linearity. The property linearity $A : B \prec C$ yields instances I of the form:

$$(A : B \prec C)@\langle w_{i_0j_0}, w_{i_1j_1}, w_{i_2j_2}\rangle$$

I is pertinent if $w_{i_0j_0}$ is active, $w_{i_1j_1}$ and $w_{i_2j_2}$ are its children, and each node is labelled with the corresponding category:

$$P(I) \Leftrightarrow \left(\begin{array}{c} w_{i_0j_0} \in V^+ \wedge w_{i_1j_1} \in C_{w_{i_0j_0}} \wedge w_{i_2j_2} \in C_{w_{i_0j_0}} \wedge \\ \texttt{cat}(w_{i_0j_0}) = A \wedge \texttt{cat}(w_{i_1j_1}) = B \wedge \texttt{cat}(w_{i_2j_2}) = C \end{array} \right)$$

Its satisfaction depends on whether the node $w_{i_1j_1}$ precedes $w_{i_2j_2}$ or not. It is thus defined as:

$$S(I) \quad \Leftrightarrow \quad (P(I) \wedge j_1 < j_2)$$

Optimization Objective. As was mentioned in section 3.2, in the loose semantics of property grammars, we want to compute models with the best fitness. We define therefore the fitness as objective function. To account for the loose semantics of property grammars, a property instance counts if it is pertinent, it counts positively if satisfied, negatively otherwise. Let $\mathcal{I}$ be the set of all property instances, $\mathcal{I}^0$ the subset of pertinent instances and $\mathcal{I}^+$ the subset of positive instances. We want to find models which maximize the ratio $|\mathcal{I}^+|/|\mathcal{I}^0|$.

Since for each instance I, the variables $P(I)$ and $S(I)$ are boolean, their reified value is either 0 or 1. We can calculate the cardinality of these sets the following way:

$$|\mathcal{I}^0| = \sum_{I \in \mathcal{I}} P(I) \qquad\qquad |\mathcal{I}^+| = \sum_{I \in \mathcal{I}} S(I)$$

Implementation. The approach described so far has been implemented using the Gecode constraint programming library[2]. This library offers a large catalogue of constraints on integer or float variables as well as set variables.

[2] `http://www.gecode.org/`

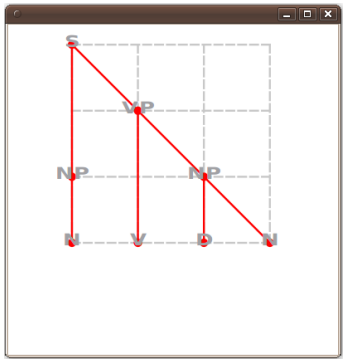


Figure 3.2: Optimal syntactic tree for "*Peter eats the apple*"

As described in Section 3.3.1, we use a $n \times m$ grid as a support. Each node w_{ij} of the grid is identified with an integer $k = (i-1) \times m + j$. The set of nodes $\mathcal{V}$ is defined as $\mathcal{V} = \{1, \ldots, n \times m\}$. V^+ and V^- are two set variables such that $V^+, V^- \subseteq \mathcal{V}$. All the constraints related to these sets are implemented using Gecode's API. The relations on C_w, D_w^+, D_w^* and P_w, A_w^+, A_w^* are encoded using arrays of set variables, whose indexes are nodes of the grid. We also use arrays of set variables to encode property-related constraints. As there are many types of constraints and many instances to consider, the computation of the indexes is slightly more complex than the ones used for tree-shapedness constraints. Definitions of $P(I)$ and $S(I)$ are realized using *reified constraints*. The search for an optimal parse is achieved using the *branch-and-bound* search strategy to maximize the ratio $|\mathcal{I}^+|/|\mathcal{I}^0|$.

Our property grammars parsing system completely explores all the tree candidates and finds one that globally optimizes the fitness. As an example, for the grammatical utterance "*Peter eats the apple*" and grammar having 19 properties handling 6 categories, the search tree has about $450,000$ nodes and 6 intermediary solutions. The optimal syntactic tree is represented in Figure 3.2.

As mentioned above, there are many instances of property to handle, that is to say, many constraints to evaluate. In practice, our parser can relatively quickly find a syntactic tree (in less than a second for the example above), but the proof of optimality can take about a minute on a 2.6 GHz processor with 4 Gb of RAM.

3.3.3 Extension to Property Grammars with Features

In the previous sections, to describe lexicalized trees, we add some lexical properties, such as $\mathsf{cat}(\textit{apple}) = \mathtt{N}$ which defines the word *apple* as being a noun. If we consider the full class of property grammars properties, they are not restricted to atomic syntactic categories, but actually handle feature structures. The properties do not only constrain atomic categories labeling syntactic nodes, but also feature-based labels.

The features are taken from a finite set of features $\mathcal{F} = \{f_1, \ldots, f_n\}$, each feature f_i takes its value in a finite upper semi-lattice D_i. We write $\top_i$ for the greatest element of D_i ($\top_i$ is typically used when feature f_i is unconstrained in a property). Since the syntactic category is mandatory, we suppose that $\mathsf{cat} \in \mathcal{F}$. Attribute-value matrices (AVM) of type $\mathcal{M} = [f_1{:}D_1, \ldots, f_n{:}D_n]$ also form a finite upper semi-lattice, equipped with the usual

"product order" (written $\sqsubseteq$).

The properties are of the following forms, where S_i are AVM expressions:

$$\begin{array}{ccc} S_0 : \triangle S_1 & S_0 : S_1! & S_0 : S_1 \prec S_2 \\ S_0 : S_1 \Rightarrow S_2 & S_0 : S_1 \not\Leftrightarrow S_2 & S_0 : s_1? \\ S_0 : S_1 \rightsquigarrow S_2 & & \end{array}$$

The last property is agreement, that states coreference constraints between categories.

Let $\mathcal{W}$ be a set of elements called *words*. A lexicon is a subset of $\mathcal{W} \times \mathcal{M}$ (that is, a lexicon maps words with AVM types). A property grammar G is a pair (P_G, L_G) where P_G is a set of properties and L_G a lexicon. When describing natural language, the properties of P_G are encapsulated within linguistic constructions, which typically describe syntactic constituents. As an illustration, consider Figure 3.3 containing an extract of the property grammars for French of [Prost 2008].

NP (Noun Phrase)	
obligation	$\triangle(\texttt{N} \sqcup \texttt{Pro})$
uniqueness	$\texttt{D}!$
	$\texttt{N}!$
	$\texttt{PP}!$
	$\texttt{Pro}!$
linearity	$\texttt{D} \prec \texttt{N}$
	$\texttt{D} \prec \texttt{Pro}$
	$\texttt{D} \prec \texttt{AP}$
	$\texttt{N} \prec \texttt{PP}$
requirement	$\texttt{N} \Rightarrow \texttt{D}$
	$\texttt{AP} \Rightarrow \texttt{N}$
exclusion	$\texttt{N} \not\Leftrightarrow \texttt{Pro}$
agreement	$\texttt{N}_{\begin{bmatrix}\text{gend} & \boxed{1} \\ \text{num} & \boxed{2}\end{bmatrix}} \rightsquigarrow \texttt{D}_{\begin{bmatrix}\text{gend} & \boxed{1} \\ \text{num} & \boxed{2}\end{bmatrix}}$

VP (Verb Phrase)	
obligation	$\triangle\texttt{V}$
uniqueness	$\texttt{V}_{[\text{mode:past-part}]}!$
	$\texttt{NP}!$
	$\texttt{PP}!$
linearity	$\texttt{V} \prec \texttt{NP}$
	$\texttt{V} \prec \texttt{Adv}$
	$\texttt{V} \prec \texttt{PP}$
requirement	$\texttt{V}_{[\text{mode:past-part}]} \Rightarrow \texttt{V}_{[\text{aux:+}]}$
exclusion	$\texttt{Pro}_{[\text{case:acc}]} \not\Leftrightarrow \texttt{NP}$

Figure 3.3: Extract of a Property Grammar for French

In this figure, the NP construction describes noun phrases. It can be read as follows. In a noun phrase, there must be either a noun or a pronoun. If there is a determiner, a noun, a prepositional phrase or a pronoun, it must be unique. The determiner (if any) precedes the noun, pronoun, prepositional and adjective phrase (if any). A noun must come with a determiner, so does an adjective phrase with a noun. There cannot be both a noun and a pronoun. There must be gender and number agreements between the noun and the determiner.[3]

In [Duchier *et al.* 2011, Duchier *et al.* 2012, Duchier *et al.* 2014], we described how the model-theoretic semantics in Section 3.2 and the model using constraints can be extended to handle feature-based properties. Features are taken into account in the definitions of property instances, of instance pertinence and of instance satisfaction. This account can be directly integrated into our model using constraints. For more details we refer the readers to [Duchier *et al.* 2014].

[3]The $\boxed{i}$ notation is the conventional graphical representation for coreferences, *i.e.*, references to the same (constrained) term.

3.4 Summary

Property grammars describe a language by sets of constraints that must be satisfied by the elements of the language. This category enables the analysis of quasi-expressions, which are utterances not completely grammatical yet represent some grammatical structures. The satisfaction of grammatical constraints makes constraint programming a candidate to achieve fully constraint based parsing. In our work, we proposed model-theoretic semantics of property grammars, that formulate syntactic analysis as a combinatorial search problem on syntactic trees. We formalized property grammars parsing by the search for solution of a constraint optimization problem. We showed that this formalization can be extended to property grammars taking into account features structures.

The implementation is still however a prototype and we do not have any benchmark. In this work we have been mainly interested in exploring the logical consequences of representation choices made in property grammars, i.e. without using any heuristic to reduce complexity. In order to deal with the challenging exponential of property grammars parsing, we could consider a decomposition of the constraint optimization problem that models the constraint-based parsing task into several sub-problems. We need therefore to determine how to divide in partial parses and to aggregate the results. We could also consider the parallelization of the search space exploration. Another point is by the branch-and-bound mechanism, our system finds only the first solution that maximizes the fitness. However, there may exist several loose models that have the same value of fitness. An extension to be considered is an enumeration of all loose models having the best fitness.

Chapter 4

Declarative Approach for Constrained Clustering

Contents

Clustering is an important task in Data Mining and many algorithms have been designed for it. It has been extended to constrained clustering, so as to integrate previous knowledge to make the clustering task either easier or more accurate. User knowledge is expressed by user-constraints, which can be instance-level or cluster-level constraints. Classic clustering algorithms are usually designed for a specific criterion and need to be adapted to integrate each kind of constraints. In our approach, we have developed a declarative and general framework for constrained clustering using constraint programming. The framework allows the user to specify several clustering tasks by integrating different types of user-constraints and by choosing an optimization criterion among several ones [Dao *et al.* 2013a]. The CP model has been improved so that the number of clusters does not need to be set beforehand, but only a lower bound and an upper bound are needed [Dao *et al.* 2014b, Dao *et al.* 2017]. To further exploit the strength of constraint programming, global optimization constraints have been developed for several popular optimization criteria in clustering [Dao *et al.* 2013b, Dao *et al.* 2015a, Dao *et al.* 2017]. We showed that the flexibility of the framework enables multi-objective constrained clustering

[Dao *et al.* 2017]. This work has been developed during the PhD thesis of Khanh-Chuong Duong [Duong 2014] that I co-supervised with Christel Vrain.

The chapter is organized as follows. In Section 4.1 we present clustering under user-constraints. In Section 4.2 we present the framework using CP that enable the modeling of different clustering objectives. To enhance the power of constraint programming global optimization constraints were developed for several well-known clustering objectives, they are presented in Section 4.3. The flexibility of the framework enables multi-objective constrained clustering, which is presented in Section 4.4.

4.1 Problem and Context

4.1.1 Dissimilarity-Based Partition Clustering

Given a set of objects, a clustering task aims at grouping the objects together into homogeneous groups, such that the objects in the same group are similar and the objects in different groups are different. The groups are called clusters and the set of groups is a clustering. The homogeneity of the clusters is usually formalized by an optimization criterion and the clustering task usually corresponds to a search of a partition that optimizes the given criterion. Different types of clustering exist: clustering by partitioning, hierarchical clustering, overlapping clustering, etc. [Aggarwal & Reddy 2013]. We are interested here in clustering by partitioning, based on a dissimilarity measure.

Let us consider a set $\mathcal{O}$ of N objects $\{o_1, \ldots, o_N\}$ and let us assume that there exists a dissimilarity measure $d(o_i, o_j)$ between each couple of objects $o_i, o_j \in \mathcal{O}$. *Dissimilarity based partition clustering* aims at finding a partition Δ of the objects in $\mathcal{O}$ into K clusters $C_1, \ldots, C_K$ such that: (1) for all $k \in [1, K]$[1], $C_k \neq \emptyset$, (2) $\cup_k C_k = \mathcal{O}$, (3) for all $k \neq k'$, $C_k \cap C_{k'} = \emptyset$, and (4) a criterion is optimized. Different optimization criteria exist, the most popular are below [Hansen & Jaumard 1997].

- Minimizing the maximal diameter of the clusters. The diameter of a cluster is the maximal dissimilarity between two objects in the cluster. The *maximal diameter* D of a partition Δ is the maximal one among the cluster diameters:

$$D(\Delta) = \max_{k \in [1,K]} \max_{o_i, o_j \in C_k} d(o_i, o_j)$$

- Maximizing the minimal split between clusters. The *minimal split* S between clusters of a partition Δ is the smallest dissimilarity between two objects in different clusters:

$$S(\Delta) = \min_{k < k' \in [1,K]} \min_{o_i \in C_k, o_j \in C_{k'}} d(o_i, o_j)$$

- Minimizing the within-cluster sum of dissimilarities WCSD:

$$WCSD(\Delta) = \sum_{k \in [1,K]} \frac{1}{2} \sum_{o_i, o_j \in C_k} d(o_i, o_j)$$

  For this criterion, the dissimilarity $d(o_i, o_j)$ is usually the squared Euclidean distance between o_i and o_j.

[1]For discrete values, $[1, K]$ denotes the set of integers from 1 to K.

- Minimizing the within-cluster sum of squares WCSS. In an Euclidean space WCSS is the sum of squared Euclidean distances between each object o_i and the centroid m_k of the cluster that contains o_i:

$$WCSS(\Delta) = \sum_{k\in[1,K]} \sum_{o_i \in C_k} ||o_i - m_k||^2,$$

  Note that when the dissimilarity $d(o_i, o_j)$ is the squared Euclidean distance, $d(o_i, o_j) = ||o_i - o_j||^2$, the sum of squares for each cluster C_k is equal to the sum of dissimilarities within the cluster C_k divided by the size of C_k. For a partition Δ, this gives:

$$WCSS(\Delta) = \sum_{k\in[1,K]} \frac{1}{2|C_k|} \sum_{o_i,o_j \in C_k} d(o_i, o_j).$$

The clustering task that minimizes the maximal diameter is also called *complete-link* non-hierarchical clustering. The task that maximizes the minimal split is also called *single-link* non-hierarchical clustering. All of these criteria except the minimal split are NP-Hard. Finding a partition maximizing the minimal split between clusters is Polynomial [Delattre & Hansen 1980]. This problem however becomes NP-Hard with the presence of user constraints [Davidson & Ravi 2007]. As for the maximal diameter criterion, the problem is Polynomial if we aim at finding a best clustering with 2 clusters ($K = 2$), but as soon as we need clusterings with at least 3 clusters ($K \geq 3$) the problem becomes NP-Hard [Hansen & Delattre 1978]. The NP-Hardness of the WCSS criterion in general dimension even with $K = 2$ is shown in [Aloise *et al.* 2009]. Because of the problem hardness, classic algorithms always search for a local optimum. For instance, the K-means algorithm finds a local optimum for the WCSS criterion, the K-median algorithm finds a local optimum for the sum of stars criterion and the FPF algorithm (Furthest Point First) [Gonzalez 1985] for the diameter criterion.

Let us notice another type of partitional clustering which is based on similarity given by weighted graphs. The similarity between the objects is defined by an undirected graph where the vertices are the objects and the edges have non-negative weights. *Spectral clustering* aims to find a partition of the vertices of the graph such that the edges between different groups have a low weights and the edges within a group have high weight. Given a cluster C_i, a cut measure $\text{cut}(C_i)$ is defined by the sum of the weights of the edges that link an object in C_i and an object not in C_i. The two most common optimization criteria are [Luxburg 2007]:

- Minimizing the ratio cut, which is defined by the sum of $\frac{\text{cut}C_i}{|C_i|}$.
- Minimizing the normalized cut, which is defined by the sum of $\frac{\text{cut}C_i}{\text{vol}(C_i)}$, where $\text{vol}(C_i)$ measures the weight of the edges within C_i.

These criteria are also NP-Hard. Spectral clustering algorithms solve relax versions of these problems: relaxing the normalized cut leads to normalized spectral clustering and relaxing the ratio cut leads to unnormalized spectral clustering. Also based on a similarity graph between objects, *correlation clustering* [Bansal *et al.* 2004] aims at finding a partition that agree the most possible with the similarities. This criterion is also NP-Hard [Bansal *et al.* 2004, Giotis & Guruswami 2006]. Another type of problems that considers

a graph of objects is *community detection*, which aims at grouping the nodes into groups with dense connections internally and sparser connection between groups. Several criteria are defined for characterizing the maximization (minimization) of internal (external) connections. Spectral clustering is therefore applied when the criterion is defined by the ratio or the normalized cut. In what follows, we will consider dissimilarity based partitional clustering.

4.1.2 Clustering Under User Constraints

In practice, the user can have some requirements for, or some prior knowledge about, the final solution. In many applications it is therefore desirable to have the clustering process take user constraints into consideration. For instance, the user can have some information on the label of a subset of objects [Wagstaff & Cardie 2000]. Another example is in a problem of determining the location for K package delivery service stations in a city, each station should serve at least 5000 ordinary customers and at least 100 high-value customers [Han *et al.* 2006]. Because of the inherent hardness of the optimization criteria, classic algorithms always find a local optimum. Several optima may exist, some of them may be closer to the user requirement. It is therefore important to integrate user prior knowledge into the clustering process. Prior knowledge is expressed by user constraints to be satisfied by the clustering solution. User constraints can be stated on instances or on clusters [Basu *et al.* 2008].

Instance-level constraints are the most widely used type of the constraints and were first introduced in [Wagstaff & Cardie 2000]. Two kinds of instance-level constraints exist: *must-link* and *cannot-link* constraints:

- A must-link (ML) constraint between two objects o_i and o_j states that they must be in the same cluster: $\forall k \in [1, K],\ o_i \in C_k \Leftrightarrow o_j \in C_k$.
- A cannot-link (CL) constraint on two objects o_i and o_j states that they cannot be in the same cluster: $\forall k \in [1, K],\ \neg(o_i \in C_k \wedge o_j \in C_k)$.

In *semi-supervised clustering*, a small amount of labeled data is available to aid the clustering process. Instance-level constraints can be inferred from class labels: if two objects have the same label then they are linked by a must-link constraint, otherwise by a cannot-link constraint. Supervision by instance-level constraints is however more general and more realistic than class labels. Using knowledge, even class labels may be unknown, a user can specify whether pairs of points belong to the same cluster or not, as for instance in clustering GPS data for lane finding [Wagstaff *et al.* 2001].

Cluster-level constraints state requirements on the clusters. Requirements can be on the size, on the breadth, on the density, etc. of the clusters. We can find the following ones:

- A *capacity (size) constraint* expresses a maximal or a minimal limit on the number of objects in each cluster. A minimal capacity constraint states that each cluster must have at least α objects: $\forall k \in [1, K],\ |C_k| \geq \alpha$. A maximal capacity contraints requires that each cluster must have at most β objects: $\forall k \in [1, K],\ |C_k| \leq \beta$.
- Considering the diameter of the clusters, a *maximum diameter constraint* gives an upper bound γ on the diameter of each cluster: $\forall k \in [1, K], \forall o_i, o_j \in C_k,\ d(o_i, o_j) \leq \gamma$.

As for the split between clusters, a *minimum split constraint* states that the clusters must be separated by at least δ: $\forall k, k' \in [1,k], k' \neq k, \forall o_i \in C_k, \forall o_j \in C_{k'}, d(o_i, o_j) \geq \delta$. Note that although the diameter or split constraints state requirements on the clusters, they can be expressed by a conjunction of cannot-link constraints or must-link constraints, respectively [Davidson & Ravi 2005].

- An ϵ-*constraint*, introduced in [Davidson & Ravi 2005], demandes that each object o_i have in its neighborhood of radius ϵ at least one other object in the same cluster: $\forall k \in [1,K], \forall o_i \in C_k, \exists o_j \in C_k, o_j \neq o_i, d(o_i, o_j) \leq \epsilon$. This constraint tries to capture the density notion, used in density based clustering DBSCAN [Ester *et al.* 1996]. We can generalize this constraint to the requirement that each object o_i has in its neighborhood of radius ϵ at least m objects in the same cluster with o_i.

4.1.3 Different Approaches for Constrained Clustering

Clustering under user constraints has been introduced in [Wagstaff & Cardie 2000] and constraint-based clustering has been termed in [Tung *et al.* 2001]. Several works have been done since to extend classic clustering algorithms to handle user constraints. Most of them consider instance-level constraints. The extension is done by enforcing pairwise constraints, using them to guide the search process or by learning a distance metric from pairwise constraints before and/or during searching. A survey on partitional and hierarchical clustering with instance level constraints can be found in [Davidson & Basu 2007]. We review below different approaches for constrained clustering. They are either approximative or exact approaches; they are designed as algorithmic methods or declarative frameworks; they can integrate instance-level constraints or both instance-level and cluster-level constraints.

K-means based methods. In this type of approach, the clustering algorithm or the objective function is modified so that user constraints are used to guide the algorithm toward a more appropriate data partitioning. The extension is done either by enforcing pairwise constraints or by using pairwise constraints to define penalties in the objective function. The first work proposed a modified version of COBWEB [Fisher 1987] that tends to satisfy all the pairwise constraints [Wagstaff & Cardie 2000]. Subsequent work extended the K-means algorithm to instance level constraints. The K-means algorithm starts with initial assignment seeds and assigns objects to clusters in several iterations. At each iteration, the centroids of the clusters are computed and the objects are reassigned to the closest centroids. The algorithm converges to a solution which is a local optimum of the within-cluster sum of squares (WCSS or distortion). To integrate ML and CL constraints, the COP-KMeans algorithm reassigns the objects in each iteration in such a way that no constraint is violated [Wagstaff *et al.* 2001]. However, this greedy behavior without backtracking means that the algorithm may fail to find a solution that satisfies all the constraints even when such a solution exists. Two variants of K-means, the Seed-KMeans and Constrained-KMeans algorithms, allow the use of labeled objects as seeds [Basu *et al.* 2002]; the difference between the two being the possibility of changing the class centers or not. In both methods, it is assumed that there is at least one seed per cluster and that the number of cluster is known. The seeds are used to overcome the sensitivity of the K-means algorithm for the initial parameterization.

Penalty based methods. Other methods uses penalties as a trade-off between finding a best clustering and satisfying as many constraints as possible. Considering a subset of instances whose label is known, the clustering objective function is modified to incorporate a dispersion measure and an impurity measure [Demiriz *et al.* 1999]. The impurity measure is based on Gini Index to measure misplaced known labels. The CVQE (constrained vector quantization error) method [Davidson & Ravi 2005] penalizes constraint violations using distance. If a must-link constraint is violated then the penalty is the distance between the two centroids of the clusters containing the two instances that should be together. If a cannot-link constraint is violated then the penalty is the distance between the cluster centroid the two instances are assigned to and the distance to the nearest cluster centroid. These two penalty types together with the distortion measure define a new differentiable objective function. An improved version, LCVQE (linear-time CVQE) [Pelleg & Baras 2007], avoids checking all possible assignments for cannot-link constraints and its penalty calculations takes into account coordinates of the involved instances in the violated constraint. The method PCK-Means [Basu *et al.* 2004a] formulated the goal of pairwise constrained clustering as minimizing a combined objective function, defined as the sum of the total squared distances between the points and their cluster centroids WCSS, and the cost incurred by violating any pairwise constraints. The cost can be uniform but can also take into account the metric of the clusters, as in the MPCK-Means version that integrates both constraints and metric learning. Lagrangian constrained clustering [Ganji *et al.* 2016] also formulates the objective function as a sum of distortion and the penalty of violating cannot-link constraints (must-link constraints are used to aggregate instances into super-instances so they are all satisfied). This method uses a Lagrangian relaxation strategy of increasing penalties for constraints which remain unsatisfied in subsequent clustering iterations. A local search approach using Tabu search was developed to optimize the objective function, which is the sum of the distortion and the weighted cost incurred by violating pairwise constraints [Hiep *et al.* 2016].

Metric based and hybrid approaches. Metric-based approaches investigate how a better distance metric can be learned from the constraints: learning from must-link constraints [Bar-Hillel *et al.* 2005] or both must-link and cannot-link constraints[Klein *et al.* 2002, Xing *et al.* 2003]. Hybrid approaches integrated both constraints enforcing and metric learning in a single framework: MPCK-Means [Bilenko *et al.* 2004], HMRF-KMeans [Basu *et al.* 2004b], semi-supervised kernel K-means [Kulis *et al.* 2005]. An uniform framework that integrates both constraint-based and metric-based methods was defined in [Bilenko *et al.* 2004]. This framework represents PCK-Means when considering a constraint-based factor and MPCK-Means when considering both constraint-based and metric-based factors. Semi-supervised HMRF K-means [Basu *et al.* 2004b] is a probabilistic framework based on Hidden Markov Random Fields, where the semi-supervised clustering objective minimizes both the overall distortion measure of the clusters and the number of violated must-link and cannot-link constraints. A K-means like iterative algorithm is used for optimizing the objective, where at each step the distortion measure is re-estimated to respect user-constraints. Semi-supervised kernel K-means [Kulis *et al.* 2005] is a weighted kernel-based approach, that generalizes HMRF K-means. The method can perform semi-supervised clustering on data given either as vectors or as a graph. It can be used on a wide class of graph clustering objectives such as minimizing the normalized cut or ratio

cut. The framework can be therefore applied on semi-supervised spectral clustering.

Algorithmic methods beyond pairwise constraints. Suitable constraints are added into the mathematical program formulation of the k-Means algorithm to extend the algorithm to the problem of partitioning objects into clusters where the number of elements in each cluster is fixed [Ng 2000]. In order to avoid local solution with empty clusters or clusters having very few points, k minimal capacity constraints are added to the formulation of the clustering optimization problem [Bradley *et al.* 2000]. This work considers the K-means algorithm and the constraints are enforced during the assignment step at each iteration. Considering cluster size constraints, a scalable algorithm starts by finding an initial solution that satisfies user-constraints and then refines the solution by performing confined object movement under constraints [Tung *et al.* 2001]. A framework to generate balanced clusters, i.e. clusters of comparable sizes was proposed in [Banerjee & Ghosh 2006], and a minimal size constraint was integrated to K-means algorithm [Demiriz *et al.* 2008]. Considering two types of constraints, the minimum number of objects in a cluster and minimum variance of a cluster, [Ge *et al.* 2007] proposed an algorithm that generates clusters satisfying them both. This algorithm is based on a CD-Tree data structure, which organizes data points in leaf nodes such that each leaf node approximately satisfies the significance and variance constraint and minimizes the sum of squared distances.

Exact methods for unconstrained clustering. Exact methods have also been investigated for clustering without user constraints. They are based on graph theory, branch-and-bound search or dynamic programming. For the diameter criterion, exact algorithms were developed using graph coloring problem [Hansen & Delattre 1978] or branch-and-bound search [Brusco & Stahl 2005]. A repetitive branch-and-bound algorithm was proposed in [Brusco 2006], which can be used for the diameter, the sum of dissimilarities WCSD and the sum of squares WCSS criteria. For the WCSS criterion, several methods were developed, based on branch-and-bound search [Koontz *et al.* 1975, Brusco 2003, Brusco 2006], dynamic programming [Jensen 1969, B.J. van Os 2004], integer linear programming using column generation [du Merle *et al.* 1999, Aloise *et al.* 2012], cutting plane algorithm [Xia & Peng 2005] or semidefinite optimization [Aloise & Hansen 2009].

Declarative approaches. Recently, approaches using generic optimization frameworks for Data Mining problems have shown their interest in the capacity of modeling and solving [De Raedt *et al.* 2008, De Raedt *et al.* 2010, Cambazard *et al.* 2010, Guns *et al.* 2011, Jabbour *et al.* 2013, Rojas *et al.* 2014, Bessiere *et al.* 2009]. For constrained clustering, several works have been developed using integer linear programming (ILP), SAT, constraint programming (CP) or mathematical programming. These approaches enable the modeling of different types of user constraints and the search of an exact solution that is a global optimum and that satisfies all the user constraints. For dissimilarity-based constrained clustering setting with two clusters ($K = 2$), a SAT framework has been proposed [Davidson *et al.* 2010]. This framework integrates different constraints (must-link, cannot-link, maximum diameter, minimum split) and the optimization criteria of diameter and split. Using ILP, [Mueller & Kramer 2010] has proposed an approach that takes a set of cluster candidates as input and constructs a clustering by selecting a subset of the candidates. This approach allows different constraints on the clusters as well as constraints

on set of clusters. Instance-level constraints can be enforced while selecting the cluster candidates. However, since the set of cluster candidates must be given, the application of this approach to general clustering setting is limited. Indeed, having a good set of clusters candidates is difficult, since the number of cluster candidates is exponential on the number of objects. This approach is experimented in conceptual clustering setting, where the candidates correspond to frequent patterns. Another approach using ILP and column generation has been developed for the sum of squares WCSS criterion [Babaki *et al.* 2014]. This approach integrates instance-level constraints and anti-monotonic cluster-level constraints, but is limited only to the WCSS criterion.

Generic optimization frameworks have also been investigated in several works for other clustering settings. Conceptual clustering considers objects described by categorical attributes and aims at associating to each cluster a definition expressed by a pattern. A CP framework has been developed for the K-pattern set mining problem that can be used for conceptual clustering [Guns *et al.* 2013]. This framework integrates constraints on patterns or groups of patterns as well as different optimization criteria. Recently another CP framework has been developed that exploit further the use of set constraints [Chabert & Solnon 2017]. A SAT based framework has also been proposed, which provides a query language to formalize conceptual clustering tasks [Métivier *et al.* 2012a]. The elements of the language are translated into SAT clauses and solved by a SAT solver. In the same manner as [Mueller & Kramer 2010], an approach using ILP based on the choice of clusters to compose a clustering has been developed in [Ouali *et al.* 2016]. This approach uses closed frequent pattern instead of frequent pattern as clusters candidates, and presents a formulation of several tasks of clustering, as for instance conceptual clustering, co-clustering or soft-co-clustering.

Regarding clustering problems on graph, a framework using mathematical programming for spectral clustering has been developed in [Wang & Davidson 2010, Wang *et al.* 2014]. This framework allows to consider different types of constraints and also to specify a lower bound on the satisfaction of the constraints. It is extended to integrate logical combinations of constraints [Zhi *et al.* 2013], which are translated into linear equations or linear inequations. Also based on a similarity graph between objects, correlation clustering aims at finding a partition that agree the most possible with the similarities. A MaxSAT framework has been developed for constrained correlation clustering [Berg & Jarvisalo 2013, Berg & Järvisalo 2017]. In this model, hard-clauses guarantee a well defined partition and soft-clauses are used to encode the cost function. For the problem of community detection, recently a model using constraint programming has been proposed [Ganji *et al.* 2017]. A SAT-based approach has been developed for overlapping community detection [Jabbour *et al.* 2017].

Different from partition clustering, hierarchical clustering constructs a hierarchy of partitions, represented by a dendrogram. A framework developed in [Gilpin *et al.* 2013] allows to model hierarchical clustering using ILP. Another SAT framework allows to integrate different types of user constraints [Gilpin & Davidson 2011].

In our work, we develop a declarative approach using constraint programming for constrained clustering problems. We study not only the modeling using constraint programming, but also enhancing the efficiency of the approach.

4.2 A Declarative Framework Using Constraint Programming

We have developed a general and declarative framework based on constraint programming for constrained clustering. The framework allows to model different problems of constrained clustering, by integrating several optimization criteria (diameter, split, WCSD, WCSS) and various types of user constraints. Let the dataset be a set of N objects that are also called points. Without loss of generality let us assume that the points are indexed and are named by their index, which ranges from 1 to N. Let $d(i, j)$ be a dissimilarity for each pair of points i, j.

4.2.1 A CP Model for Constrained Clustering

This subsection describes the first model [Dao *et al.* 2013a, Dao *et al.* 2013d, Dao *et al.* 2013c].

Variables and constraints. A CP model is characterized by the set of variables and their domains and the set of constraints. In this model, a partition is represented by two levels: the representative points that represent the clusters and the assignment of each point to a cluster representative. The variables are therefore defined as follows:

- For each cluster $k \in [1, K]$, among all the points of the cluster, the one with the smallest index is considered as the representative point [2]. An integer variable $I[k]$[3] with $Dom(I[k]) = [1, N]$ is introduced, where $I[k]$ is the index of the representative point of the cluster k.

- For each $i \in [1, N]$, let $G[i]$ be a variable with $Dom(G[i]) = [1, N]$, where $G[i]$ is the representative point of the cluster that contains point i.

The constraints can be organized into three categories: (1) constraints to guarantee a partition, (2) constraints to express user-constraints and (3) constraints to express the optimization criterion. When no optimization criterion is specified, the model corresponds to finding all the partitions that satisfy the user-constraints. To define a partition, the following constraints are used:

- Each representative belongs to its cluster: $\forall k \in [1, K]$, the constraint $G[I[k]] = I[k]$ is put. This constraint is expressed by an `element` CP constraint.

- Each point is assigned to a representative: $\forall i \in [1, N]$, $\bigvee_{k \in [1,K]}(G[i] = I[k])$. This relation can be expressed by N cardinality constraints in CP: $\forall i \in [1, N], \quad \#\{k \mid I[k]{=}G[i]\} = 1$.

- The representative of a cluster is the point in this cluster with the minimal index; in other words, the index i of a point is greater or equal to the index of its representative given by $G[i]$: $\forall i \in [1, n], \quad G[i] \leq i$.

[2] It allows to have a single representation of a cluster. It must not be confused with the notion of representative in the medoid approach.

[3] To denote the k-th element of an array I, we will use two notations $I[k]$ and I_k.

- In order to break symmetries among the clusters, the following conditions are expressed. The representative of the first cluster is the first point: $I[1] = 1$. The representatives are sorted in increasing order: $\forall k \in [1, K-1],\ I[k] < I[k+1]$.

To represent user-constraints in clustering, CP constraints are used. A must-link constraint on points i, j is expressed by $G_i = G_j$ and a cannot-link constraint by $G_i \neq G_j$. Cluster level constraints are naturally expressed by CP constraints, for instance a minimal cluster size constraint is expressed by K cardinality CP constraints: $\forall k \in [1, K], \#\{i \mid G[i] = I[k]\} \geq \alpha$. For each $k \in [1, K]$, this constraint states that the value of $I[k]$ must appears at least α times in the array G. A density constraint states that each point must have in its neighborhood of radius ϵ at least $MinPts$ points belonging to the same cluster as itself. So, for each $i \in [1, N]$, the set of points in its ϵ-neighborhood is computed and a constraint is put on its cardinality:

$$\#\{j \mid d(i,j) \leq \epsilon, G[j]{=}G[i]\} \geq MinPts$$

Optimization criteria. To represent the optimization criterion, a float value variable is introduced for each potential criterion: D (diameter), S (split) and W (WCSD). Constraints are used to enforce the relation behind each criterion. For instance, for the diameter criterion, since D represents the maximal diameter of the clusters, any two points i, j that have $d(i, j) > D$ must be in different clusters. Since the value of D is still unknown, this relation can be expressed using $O(N^2)$ reified constraints:

$$\forall i < j \in [1, N], \quad (d(i,j) > D) \rightarrow (G[i] \neq G[j])$$

The relation that defines the sum WCSD is defined by $W = \sum_{i,j \in [1,N]} (G[i] == G[j]) d(i,j)^2$, where $G_i == G_j$ is 1 if the variables G_i, G_j have the same value and 0 otherwise.

Model improvements. The variables $I[k]$ is the smallest index of the points in cluster k. The way points are indexed is therefore really important. Points are then ordered and indexed, so that points that are probably representatives have small index. In order to achieve this, we rely on FPF (Furthest Point First) algorithm [Gonzalez 1985]. This algorithm starts by choosing a point furthest from the others, marks it as the first head, links all the points to it and iterates until all the points are marked. At each iteration, it chooses the point i that is furthest to its head, marks it as a new head and links to it all the unmarked points that are closer to i than to their head. The order where the points are marked gives the new order of the points.

Search strategy. The variables $I[k]$ for $k \in [1, K]$ are instantiated before the variables $G[i]$ for $i \in [1, N]$. This means that cluster representations are first determined, allowing constraint propagation to assign some points to clusters. When all the variables $I[k]$ are instantiated, the variables $G[i]$ whose domains are not singletons are instantiated. Variables $I[k]$ are chosen from $I[1]$ to $I[K]$. Since the representative is the one with the minimal index in the cluster, values for instantiating each $I[k]$ are chosen in an increasing order. Variables $G[i]$ are chosen so that the ones with the smallest remaining domain are chosen first. All values in $Dom(G[i])$ are examined and the value j which corresponds to the smallest $d(i, j)$ is chosen and two alternatives are created $G[i] = j$ and $G[i] \neq j$.

4.2.2 An Improved CP Model

The first model is based on the representative points to represent the clusters. This implies that the number of clusters K must be set beforehand, because the number of variables I_k depends on K. A second model has been developed, which is different from the first one on the choice of variables. In this model, the number of clusters K is not fixed, only bounds are needed $K_{min} \leq K \leq K_{max}$. Moreover, dedicated global optimization constraints are developed for each optimization criterion (these constraints are described in Section 4.3). This model was developed in [Dao *et al.* 2014b] and was improved in [Dao *et al.* 2014a, Dao *et al.* 2017].

Two constants K_{min} and K_{max} must be given, which define the minimal and the maximal number of clusters. The objectif is to find a partition of K clusters, where $K_{min} \leq K \leq K_{max}$, such that a given criterion is optimized. The optimization criterion can be:

- minimizing the maximal diameter of the clusters,
- maximizing the minimal split between clusters,
- minimizing the within-cluster sum of dissimilarities WCSD,
- minimizing the within-cluster sum of squares WCSS.

Variables. In this model, the clusters are identified by their index. For a clustering of K clusters, the indices are from 1 to K. To define the assignment of points to clusters, we introduce integer variables $G_1, \ldots, G_N$, each one having the domain the set of integers $\{1, .., K_{max}\}$. An assignment $G_i = k$ means point i is grouped into the cluster k. To represent the optimization criterion, a float value variable is introduced for each criterion: D for the diameter, S for the split, W for the WCSD and V for the WCSS. Their domain is defined by: $Dom(D) = Dom(S) = [\min_{i,j}(d(i,j)), \max_{i,j}(d(i,j))]$, $Dom(W) = Dom(V) = [0, \infty]$.

Constraints. A complete assignment of $G_1, \ldots, G_N$ defines naturally a partition. However, a partition can correspond to several different assignments, by permutation of cluster indices or by changing a cluster index to an unused value. In order to break this kind of symmetries, the clusters are created and indexed such as the first created cluster has the index 1 and a number c is used to index a new cluster only if the number $c-1$ has already been used. A direct method to express this condition is by using the constraints $G_1 = 1$ and $G_i \leq \max_{j \in [1,i-1]}(G_j) + 1$, for $i \in [2, N]$. However, using a global constraint that enforces this relations will yield more interaction and propagation. The global constraint *precede* [Law & Lee 2004] eventually achieves this: $precede([G_1, \ldots, G_N], [1, \ldots, K_{max}])$. This constraint ensures $G_1 = 1$ and moreover, for $i \in [2, N]$, if $G_i = c$ where $1 < c \leq K_{max}$, then there must exist $j < i$ such that $G_j = c - 1$.

To enforce that there must be at least K_{min} clusters, each number from 1 to K_{min} must be used at least once in the assignment of $G_1, \ldots, G_N$. In presence of the constraint *precede*, one needs only to impose that the value K_{min} is used at least once. This means $\#\{i \mid G_i = K_{min}\} \geq 1$, which can be expressed by a CP *atleast* constraint: $atleast(1, [G_1, \ldots, G_N], K_{min})$.

The domain of each variable G_i is the set of integers $[1, K_{max}]$, therefore there will be at most K_{max} clusters. If one needs exactly K clusters, it is sufficient to set $K_{min} = K_{max} = K$.

Clustering user-constraints are expressed in the same way as in the first model. The only difference is on the minimal size constraint, where the variables I_k do no more exist. To express this constraint, we impose that for each $i \in [1, N]$, the value taken by G_i must appear at least α times in the array G, *i.e.* $\#\{j \mid G_j = G_i\} \geq \alpha$. The minimal size helps also to upper bound the number of clusters: $G_i \leq \lfloor n/\alpha \rfloor$, for $i \in [1, N]$.

Optimization criteria. In the first model [Dao *et al.* 2013a], a large number of reified constraints are used to express the relation defined by the optimization criterion. For instance, to define that D is the maximal diameter, $O(N^2)$ reified constraints are used. The number of constraints in the model is therefore important but the link between the constraints is not well exploited. In the second model, we have developed a global optimization constraint for each criterion that summarizes the corresponding relation. These constraints as well as their filtering algorithm are presented in Section 4.3. These constraints are:

- *diameter(*$[G_1, \ldots, G_N], D, d$*)*: D is the maximal diameter of the clusters formed by an assignment of $G_1, \ldots, G_N$, using the dissimilarity measure d;
- *split(*$[G_1, \ldots, G_N], S, d$*)* : S is the minimal split between clusters;
- *wcsd(*$[G_1, \ldots, G_N], W, d$*)* : W is the within-cluster sum of dissimilarities;
- *wcss(*$[G_1, \ldots, G_N], V, d$*)* : V is the within-cluster sum of squares.

When the optimization criterion is specified, the corresponding value is optimized in the objective function. For instance, if the sum of squares is minimized, the objective function will be *minimize* V.

Search strategies. Several search strategies are defined and depending on the objective function a strategy is chosen. For the diameter and split criteria, at each branching point, a variable G_i with the smallest domain is chosen. All values in $Dom(G_i)$ are examined and the number of the closest cluster to i is chosen. The distance between a point i and a cluster k is defined as the maximum distance $d(i, j)$ where G_i is already instantiated to k. If the cluster k is empty (no point G_j such that $G_j = k$), the distance between i and cluster k is zero. This means the creation of a new cluster is favored if there are unused cluster numbers. Moreover, the smallest remaining number is chosen. The closest cluster k to the point i is chosen and two alternatives are created $G_i = k$ and $G_i \neq k$.

For the WCSD and WCSS criteria, a mixed strategy is used. In order to have a good upper bound for the objective variable, a greedy search is used. This means at each branching point, the variable G_i and the value k that increase the objective function as little as possible are chosen. After finding the first solution, the strategy changes to a "first-fail", which tends to detect failures quickly. In this strategy, a value s_{ik} for each point i and each cluster k is defined as the added amount if i is assigned to k. For each unassigned variable G_i, let $s_i = \min_{k \in Dom(G_i)}(s_{ik})$, that is the minimal added amount when i is assigned to a cluster. The unassigned variable G_i having the greatest value s_i

Dataset	# Objects	# Attributes	# Classes
Iris	150	4	3
Wine	178	13	3
Glass	214	9	7
Ionosphere	351	34	2
User Knowledge	403	5	4
Breast Cancer	569	30	2
Synthetic Control	600	60	6
Vehicle	846	18	4
Yeast	1484	8	10
Multiple Features	2000	6	10
Image Segmentation	2000	19	7
Waveform	5000	40	3

Table 4.1: Properties of datasets

is chosen, and for this variable, the value k with the smallest value s_{ik} is chosen. Two alternatives are created, corresponding to $G_i = k$ and $G_i \neq k$.

Experiment results The experiments in [Dao *et al.* 2017] were conducted considering different aspects: comparison to existent approaches with different criteria, analysis of bounds on the number of clusters, analysis of search strategies. We give here some details on the results. The two models are implemented using Gecode solver version 4.2.1. The experiments are performed on a 3.4 GHz Intel Core i5 processor with 8 GB RAM running Ubuntu. Experiments have been done on twelve datasets taken from the UCI repository [Bache & Lichman 2014]. Table 4.1 summarizes information of these datasets.

In case of minimizing the maximal diameter without user-constraints, the model was compared to the repetitive branch-and-bound approach (RBBA) [Brusco & Stahl 2005] and the algorithm based on graph coloring (GC) [Delattre & Hansen 1980]. The RBBA program has been obtained from the author's website[4]. To our knowledge, this was the best exact algorithm for the maximal diameter and WCSD criteria without constraints. No implementation of GC was available so we coded it ourselves in C++ using a well-known available graph coloring program [Mehrotra & Trick 1995]. The timeout was set to 1 hour and the Euclidean distance was used to compute the dissimilarity between objects. The value K was set to the ground truth number of clusters.

Table 4.2 shows the results of the experiments. The symbol - is used when the search is not completed after the timeout and the symbol * is used to mark memory run out. All the algorithms are exact and they find the same value for the optimum diameter. It is clear that our CP models outperform the existent approaches. The second model (CP2) is the most efficient in all cases.

Considering the WCSD criterion, the second model is compared to RBBA, which to our knowledge, is the best exact algorithm for this criterion without user constraints. Both approaches can only find the optimal solution for the Iris dataset (RBBA 3249 sec, CP2 4125 s). Our model can handle different kinds of user constraints. A set of 120

[4] `http://mailer.fsu.edu/~mbrusco/`

Datasets	D_{opt}	RBBA	GC	CP1	CP2
Iris	2.58	1.4	1.8	< 0.1	< 0.1
Wine	458.13	2	2.3	0.3	< 0.1
Glass	4.97	8.1	42	0.9	0.2
IonoSphere	8.60	–	0.6	0.4	0.3
User Knowledge	1.17	–	3.7	75	0.2
Breast Cancer	2377.96	–	1.8	0.7	0.5
Synthetic Control	109.36	–	–	56.1	1.6
Vehicle	264.83	–	–	14.3	0.9
Yeast	0.67	–	–	2389.9	5.2
Multi Features	1594.96	–	–	$*$	10.4
Image Segmentation	436.40	–	–	589.2	5.7
Waveform	15.60	–	–	$*$	50.1

Table 4.2: Runtime in seconds with the minimization of the maximal diameter and without user-constraints

instance-level constraints has been generated from the dataset Iris. The constraints were generated following the method described in [Wagstaff & Cardie 2000]: two points are chosen randomly from the dataset, if they belong to the same cluster in the real partition, a must-link constraint is generated, otherwise a cannot-link constraint is generated. The first test is without user-constraints, the second one considers the first 30 constraints, the third one takes into account the first 60 constraints and so on. Figure 4.1 (left) reports the total time needed to solve the dataset with these user-constraints. When there are 30 constraints, the solver takes more computation time. The reason is that, with user-constraints, the optimal value of WCSD is higher and the propagation of the WCSD constraint is weaker. However, when more user-constraints are integrated, the propagation of must-link and cannot-link constraints is stronger and enables to quickly instantiate variables. As a result, the solver takes only 94s for solving the problem with 60 constraints, and less than 10s when there are 90 or more constraints.

We have also evaluated the quality of the partitions found. For measuring the quality of a partition, we consider the Adjusted Rand Index (ARI). It measures the similarity between two partitions, in this case, the real partition P of the dataset and the partition P' found by our model. It is defined by:

$$ARI = \frac{2(ab - cd)}{(a+d)(d+b) + (a+c)(c+b)}$$

where a is the number of pairs of points that are in the same cluster in P and in P', b is the number of pairs of points that are in different clusters in P and in P', c is the number of pairs of points that are in the same cluster in P, but in different clusters in P' and d is the number of pairs of points that are in different clusters in P, but in the same cluster in P'. The results of this experiment are reported in Figure 4.1 (right). The figure shows that the ARI value of the optimal partition improves when more are more constraints are considered.

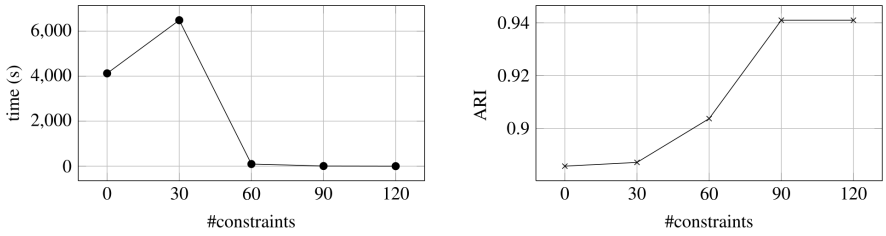


Figure 4.1: WCSD with user-constraints on Iris: computation time (left), Adjusted Rand Index (right)

4.3 Global Optimization Constraints for Clustering

CP solvers offer a rich catalogue of elementary and global constraints. The relation of the optimization criterion can be expressed using existent CP constraints. For instance, in the first model, reified constraints are used to express the relations of maximal diameter and minimal split. The WCSD and WCSS criteria can also be expressed by composing different existent constraints. However, a decomposition into several constraints does not allow to fully exploit the interaction between the variables, i.e. changes on decision variables may not have impact on the criterion since the constraints are considered separately. Moreover, a large number of constraints is usually required, e.g. $O(N^2)$ reified constraints for the diameter or WCSD. One of the strength of CP is the modularity, where each global constraint represents a relation, and we have the possibility of adding new global constraint to define new relation. A filtering algorithm is associated to each global constraint, which enforces the relation expressed by the constraint. The filtering algorithm exploits the relation to remove inconsistent values from the domain of the variables.

We have developed for each optimization criterion a dedicated global constraint and its filtering algorithm. Each constraint links the variable representing the criterion and the decision variables $G_1, \ldots, G_N$ representing the partition. Since the variable representing the criterion is the objective function, this kind of global constraints is also called global optimization constraint. During the branch-and-bound search, when a new solution is reached, the value of the objective function at this solution is computed and is used to narrow the domain of the objective variable. Considering directly and globally the variable of the objective function and the decision variables, we can capture more interactions between them in order to have better propagation. Note that these global constraints can also be used without optimization. For instance, *diameter(*$[G_1, \ldots, G_N], D, d$*)* and $D \leq \gamma$ allow to express a constraint on the maximal diameter of the clusters.

The next subsections will present the global constraints for the maximal diameter of the clusters and the minimal split between clusters [Dao *et al.* 2017], for the within-cluster sum of dissimilarities WCSD [Dao *et al.* 2013b] and for the within-cluster sum of squares WCSS [Dao *et al.* 2015a]. The constraints for diameter or split achieve the same level of propagation with respect to the reified constraints, but with much less time. The constraints for WCSD and WCSS, on the other hand, have dramatically better performance either on propagation level or on the execution time.

4.3.1 Maximal Diameter and Minimal Split

The relation that D is the maximal diameter of the clusters formed by the decision variables $G_1, \ldots, G_N$, considering a dissimilarity measure d, is expressed by:

$$\forall i < j \in [1, N], \quad (D < d(i,j) \ \rightarrow \ G_i \neq G_j) \wedge (G_i = G_j \rightarrow D \geq d(i,j)). \tag{4.1}$$

Using existent CP constraints, this relation can be expressed using $O(N^2)$ reified constraints. We have developed a global constraint *diameter*$([G_1, \ldots, G_N], D, d)$ enforcing this relation. The filtering algorithm is presented in Algorithm 3. In this algorithm, $Dom(D)$ is represented by $[D.lb, D.ub)$, where $D.lb$ is the lower bound, which initially can be the minimum dissimilarity between two points, and $D.ub$ is the upper bound, which can be the maximum dissimilarity between two points or the D value of the last found solution. The bound $D.ub$ is strict since by branch-and-bound, the next solution must be strictly better than the previous one. The filtering algorithm is active when the domain of some variables is changed. The relation (4.1) is used to narrow the variable domains in the following situations:

- The upper bound $D.ub$ has been reduced (e.g. a solution has been found). In this case, for each couple $i < j$, if $D.ub \leq d(i,j)$, we can conclude that $D < d(i,j)$ and by (4.1) conclude that $G_i \neq G_j$. The relation $G_i \neq G_j$ is however useful to domain reduction only when one of the variables has already been instantiated. Therefore, Algorithm 3 memorizes the instantiated variables (lines 2–4) and uses them to filter (line 10). The lower bound $D.lb$ can also be revised (line 11).

- Some variables G_i have been instantiated. In this case, for each couple $i < j$ such that G_i and G_j are instantiated by the same value, we deduce $D \geq d(i,j)$ and revise $D.lb$ (line 11).

Notice that as soon as the domain of a variable becomes empty, a failure is invoked by the solver. The worst case complexity is $O(N^2)$. The algorithm is active when the upper bound of D has been reduced or a variable G_i has been instantiated. However, because of its complexity, the filtering is executed after the other constraints whose filtering has a lower cost.

The constraint *split*$(\mathcal{G}, S, d)$ enforces that S is the minimal split between the clusters formed by $G_1, \ldots, G_N$. It represents the relation:

$$\forall i < j \in [1, N], \quad (S > d(i,j) \ \rightarrow \ G_i = G_j) \wedge (G_i \neq G_j \rightarrow S \leq d(i,j)). \tag{4.2}$$

The filtering algorithm is presented in Algorithm 4, where $Dom(S) = (S.lb, S.ub]$. The lower bound $S.lb$ is either the minimum dissimilarity between two points or the S value of the last found solution. This algorithm is active when the lower bound $S.lb$ has been increased or some variables G_i have been instantiated. If $S.lb$ is changed, for each couple $i < j$, if $S.lb \geq d(i,j)$, then by (4.2) we have $G_i = G_j$. This relation is propagated by enforcing $Dom(G_i) = Dom(G_j)$. If some decision variables have been instantiated and $G_i \neq G_j$, by (4.2) we have $S \leq d(i,j)$ that changes the upper bound of S. The complexity is also $O(N^2)$.

Algorithm 3: Filtering for constraint *diameter*$(\mathcal{G}, D, d)$

```
1  stack ← ∅;
2  if D.ub has been changed then
3      for i ← 1 to n where G_i is instantiated do
4          stack ← stack ∪ {i};
5  else
6      foreach i that G_i has just been instantiated do
7          stack ← stack ∪ {i};
8  foreach i ∈ stack do
9      for j ← 1 to n do
10         if d(i, j) ≥ D.ub then delete G_i from Dom(G_j) ;
11         if G_j is instantiated ∧ G_i = G_j then D.lb ← max(D.lb, d(i, j)) ;
```

Algorithm 4: Filtering for constraint *split*$(\mathcal{G}, S, d)$

```
1  stack ← ∅;
2  if S.lb has been changed then stack ← {1, ..., n} ;
3  else
4      foreach i that Dom(G_i) has just been changed do
5          stack ← stack ∪ {i};
6  foreach i ∈ stack do
7      for j ← 1 to n do
8          if d(i, j) ≤ S.lb then
9              Dom(G_i) ← Dom(G_i) ∩ Dom(G_j);
10             Dom(G_j) ← Dom(G_i);
11         if G_i and G_j are instantiated ∧ G_i ≠ G_j then S.ub ← min(S.ub, d(i, j)) ;
```

	reified constraints	dedicated global constraint
Iris	< 0.1	< 0.1
Wine	< 0.1	< 0.1
Glass	0.4	0.2
IonoSphere	0.3	0.3
User Knowledge	15.4	0.2
Breast Cancer	0.7	0.5
Synthetic Control	23.6	1.6
Vehicle	11.9	0.9
Yeast	574.2	5.2
Multi Features	*	10.4
Image Segmentation	226.7	5.7
Waveform	*	50.1

Table 4.3: Performance (measured in seconds) for different modelings of diameter criterion

Analysis of the dedicated filtering algorithm. Using existent CP constraints, modeling the maximal diameter requires $O(N^2)$ reified constraints. The dedicated global constraint ensures the same level of consistency compared to reified constraints. However, although our dedicated global constraint has a complexity in the worst case of $O(N^2)$, it considers only necessary variables. In order to compare the efficiency of the filtering algorithm, we use the new model (CP2) for both cases: using reified constraints and using the dedicated global constraint to express the diameter criterion. The performance is presented in Table 4.3. We can see that when using reified constraints, the solver cannot find optimal solution with the datasets Wave Form and Multi Features. The reason is that there are too many reified constraints and the computer runs out of memory. Table 4.3 shows that the filtering algorithm boosts the performance and this becomes more and more significant with larger datasets.

4.3.2 Within-Cluster Sum of Dissimilarities

The relation that W is the within-cluster sum of dissimilarities of the clusters formed by $G_1, \ldots, G_N$, using the dissimilarity measure d, is:

$$W = \sum_{1 \leq i < j \leq N} [G_i = G_j] d(i, j) \tag{4.3}$$

where $[G_i = G_j]$ is 1 if G_i and G_j have the same value and 0 otherwise. Optimizing this criterion is NP-Hard since the weighted max-cut problem, which is NP-Complete, is a particular instance of this problem with two clusters. We have developed a global constraint $wcsd([G_1, \ldots, G_N], W, d)$ enforcing this relation [Dao *et al.* 2013b].

Let $Dom(W) = [W.lb, W.ub)$. Given a partial assignment of the variables $G_1, \ldots, G_N$, two steps are achieved in the filtering algorithm:

1. Compute a new lower bound $W.lb$, using the partial configuration of the clusters. If $W.lb \geq W.ub$ then a failure is invoked.

2. Filter the domain of the unassigned decision variables G_i using the lower bound $W.lb$.

The essential of the algorithm is described below. For the details the reader could see [Dao *et al.* 2013b].

Lower bound for WCSD. Given a partial assignment of the variables $G_1, \ldots, G_N$, let $A = \{i \in [1, N] \mid G_i \text{ is assigned}\}$ and $U = \{i \in [1, N] \mid G_i \text{ is unassigned}\}$. According to [Klein & Aronson 1991], (4.3) can be separated into three parts $W = W_1 + W_2 + W_3$, where:

- W_1 is the sum of within-cluster dissimilarities between the assigned points:

$$W_1 = \sum_{i,j \in A, i<j} [G_i = G_j] d(i, j)$$

- W_2 is the sum of within-cluster dissimilarities between an unassigned and an assigned point:

$$W_2 = \sum_{i \in U, j \in A} [G_i = G_j] d(i, j)$$

- W_3 is the sum of within-cluster dissimilarities between the unassigned points:

$$W_3 = \sum_{i,j \in U, i<j} [G_i = G_j] d(i, j)$$

The exact value of W_1 is known but not for W_2 and W_3 because of the unassigned points. However we can compute a lower bound for W_2 and W_3. A lower bound for W_2, denoted by $W_2.lb$ is computed as follows. For each point $i \in U$, each value $k \in Dom(G_i)$ represents the number of a cluster where point i can be assigned to. If i is assigned to cluster k, it will add to W_2 the amount $\sum_{j \in A} [G_j = k] d(i, j)$. A lower bound $W_2.lb$ is then the sum of the minimal amount added by each unassigned point:

$$W_2.lb = \sum_{i \in U} \min_{k \in Dom(G_i)} (\sum_{j \in A} [G_j = k] d(i, j))$$

A lower bound for W_3, denoted by $W_3.lb$, is computed as follows. Let $p = |U|$ and $k = |\cup_{i \in U} Dom(G_i)|$. Let us observe that in the sum W_3, the minimal number of terms $d(i, j)$ is the minimal number of within-cluster connections, considering all the possibilities of grouping p points into k clusters. Let m be the quotient and let m' be the remainder of the division of p by k. Let $f(p, k) = (km^2 + 2mm' - km)/2$. We have proved in [Dao *et al.* 2013b] that whatever be the partition of p points into k clusters, the number of within-cluster connections is at least $f(p, k)$. This value is reached when m' clusters have each one $m+1$ points and $k-m'$ clusters have each one m points. Therefore, a lower bound $W_3.lb$ is the sum of the $f(p, k)$ smallest dissimilarities $d(i, j)$, for $i < j \in U$.

The lower bound of W is therefore revised by $W.lb = \max(W.lb,\ W_1 + W_2.lb + W_3.lb)$. If $W.lb \geq W.ub$, a failure will be invoked by the solver.

n	without filtering		with filtering	
	#nodes	time	#nodes	time
20	667	0.004	375	0.002
25	2887	0.03	599	0.004
30	17183	0.2	867	0.01
35	47901	0.8	1207	0.02
40	1362113	29.7	1663	0.04
45	5687055	145.8	2071	0.06

Table 4.4: Performance of filtering algorithm

Filter decision variable domains. For each unassigned variable G_i, for each value $c \in Dom(G_i)$, with the assumption that point i is assigned to cluster c, the lower bound of W is revised to $W.lb' = W'_1 + W_2.lb' + W_3.lb'$. The revision of W'_1 and W'_2 can be done in a constant time. In order to revise W'_3 in a constant time, we use a weaker lower bound W_4, which is the sum of $f(p-1,k)$ smallest dissimilarities $d(u,v)$, for $u < v \in U$. The used dissimilarities may be linked with i, thus $W'_3.lb \geq W_4$. The interest of using W_4 is that W_4 can be computed once for all unassigned variables G_i, so the revision of W'_3 is done in a constant time. The revised lower bound is $W.lb' = W'_1 + W_2.lb' + W_4$. If $W.lb' \geq W.ub$ then point i cannot be assigned to cluster c. In this case value c is removed from $Dom(G_i)$.

The complexity of the filtering algorithm is $O(N^2 + NK) = O(N^2 + NK_{max})$, since $|Dom(G_i)| \leq K_{max}$. As $K_{max} \leq N$, the complexity is $O(N^2)$.

Experiment results We give here some results of the experiments, more details can be found in [Dao *et al.* 2013b]. The constraint was used in the first model [Dao *et al.* 2013a] and was implemented using Gecode 4.0.0. Expressing directly the relation $W = \sum_{1 \leq i < j \leq n} (G[i] == G[j]) d(i,j)$ using reified constraints and a linear constraint, the propagation is weak. The model without filtering hardly solves dataset with more than 50 samples. Our filtering algorithm takes benefit from both assigned an unassigned points to have a better lower bound and a better filtering. Table 4.4 shows the comparison of performance of our model in two cases: with and without the filtering. In each case, the first column gives the number of nodes in the search tree and the second column reports the total runtime in seconds. The number of samples varies from $n = 20$ to $n = 45$ and the number of clusters K is set to 3.

Our model can find exact solution with different user-constraints as given in Table 4.5. For the dataset Letter Recognition from UCI, only 600 objects of 3 classes are considered from the 20.000 objects in the original dataset, they are composed of the first 200 objects of each class.

4.3.3 Within-Cluster Sum of Squares

In an Euclidean space WCSS is the sum of squared Euclidean distances between each object o_i and the centroid m_k of the cluster that contains o_i. Let $d(i,j)$ be the squared Euclidean distance between points i, j, i.e. $d(i,j) = ||o_i - o_j||^2$. For a partition Δ, WCSS

Dataset	User-constraints	Total time
Wine	separation: $\delta = 1.5\%\ maxD$ minimal capacity: $\alpha = 30$	11.2s
Letter Recognition	# ML constraints = 0.1% total pairs # CL constraints = 0.1% total pairs separation: $\delta = 10\%\ maxD$	11.5s
Vehicle	separation: $\delta = 3\%\ maxD$ diameter: $\gamma = 40\%\ maxD$	1.6s

Table 4.5: Example of combinations of user-constraints

is also defined by:

$$WCSS(\Delta) = \sum_{k \in [1,K]} \frac{1}{2|C_k|} \sum_{o_i, o_j \in C_k} d(i,j).$$

This criterion is popular since the well-known algorithm k-means locally optimizes it. Minimizing the WCSS is however NP-Hard even for $K = 2$ [Aloise *et al.* 2009]. A direct modeling using existent CP constraints is not efficient since the propagation is really weak. A direct modeling can hardly solve a sample of Iris of 14 points with $K = 3$ within 30 minutes.

In [Dao *et al.* 2015a], we have developed a global constraint *wcss(*$[G_1, \ldots, G_N], V, d$*)* that enforces the relation where V is the WCSS of the clusters formed by the decision variables $G_1, \ldots, G_N$, using the distance measure d. Let $Dom(V) = [V.lb, V.ub)$. The filtering algorithm follows the same principle: (1) taking into account the partial assignment of $G_1, \ldots, G_N$, a lower bound $V.lb$ is computed, if $V.lb \geq V.ub$ then a failure is invoked and (2) $V.lb$ is used to filter the domain of unassigned decision variables. The essential of the filtering algorithm is described below.

Lower bound for WCSS. Let $K = \max\{c \mid c \in \bigcup_i Dom(G_i)\}$. The value K is the maximum number of clusters in the partition. Let $C_1, \ldots, C_K$ be the clusters. Let U be the set of the unassigned points and let $q = |U|$. We need to compute a lower bound for V, considering all the possibilities of assigning all the points of U to the clusters $C_1, \ldots, C_K$. The lower bound computation is achieved in two steps:

1. For each $m \in [0, q]$ and $k \in [1, K]$, a lower bound $\underline{V}(C_k, m)$ for the WCSS of the cluster C_k is computed, regarding all the possibilities of assigning any m points of U to C_k. This bound is computed taking into account either the distances between unassigned and assigned points or the distances between unassigned points. For the bound on distances between unassigned points, an estimation based on the smallest distances is used.

2. For each $m \in [0, q]$ and $k \in [2, K]$, a lower bound $\underline{V}(C_1 \ldots C_k, m)$ for the WCSS of the clusters $C_1, \ldots, C_k$ is computed, regarding all the possibilities of assigning any m points of U to k clusters $C_1, \ldots, C_k$.

   An assignment of m points to k clusters corresponds to an assignment of i points to

$k-1$ first clusters and $m-i$ points to the last one. Therefore:

$$WCSS(C_1..C_k, m) \geq \min_{i \in [0,m]} (WCSS(C_1..C_{k-1}, i) + WCSS(C_k, m-i))$$

A lower bound $\underline{V}(C_1..C_k, m)$ can be defined by:

$$\underline{V}(C_1..C_k, m) = \min_{i \in [0,m]} (\underline{V}(C_1..C_{k-1}, i) + \underline{V}(C_k, m-i)) \tag{4.4}$$

This bound is computed using a dynamic program for each $k \in [2, K]$ and $m \in [0, q]$.

A lower bound for V this therefore $\max(V.lb, \underline{V}(C_1 \dots C_K, q))$. The complexity of the first step is $O(Kq^2 \log q + qN)$ and for the second step is $O(Kq^2)$. The complexity of the lower bound computation is therefore $O(Kq^2 \log q + qN)$.

Filtering decision variable domains. For each value $k \in [1, K]$, for each unassigned variable G_i, if $k \in Dom(G_i)$, under the assumption that point i is assigned to cluster C_k, a new lower bound $V.lb'$ is computed.

Let C'_k be the cluster $C_k \cup \{i\}$ and let $\mathcal{C}' = \{C_c \mid c \neq k\} \cup \{C'_k\}$. A new lower bound $V.lb'$ is the value $\underline{V}(\mathcal{C}', q-1)$, since there remains $q-1$ points of $U \backslash \{i\}$ to be assigned. According to (4.4):

$$\underline{V}(\mathcal{C}', q-1) = \min_{m \in [0, q-1]} (\underline{V}(\mathcal{C}' \backslash \{C'_k\}, m) + \underline{V}(C'_k, q-1-m))$$

For all $m \in [0, q-1]$, the bounds $\underline{V}(\mathcal{C}' \backslash \{C'_k\}, m)$ and $\underline{V}(C'_k, m)$ are revised in making use of the informations computed in the first step (the computation of the lower bound $V.lb$). If $\underline{V}(\mathcal{C}', q-1) \geq V.ub$, point i cannot be assigned to cluster k. The value k is then removed from $Dom(G_i)$.

The complexity of filtering decision variable domains is $O(Kq^2)$. The total complexity of the filtering algorithm is therefore $O(Kq^2 \log q + qN)$.

Experiment results. The constraint was used in the second model [Dao *et al.* 2017] and the system was implemented using Gecode 4.2.7. The model was compared to the approach based on Integer Linear Programming and column generation [Babaki *et al.* 2014], which was the state-of-the-art method for WCSS with user-constraints. To generate user constraints, pairs of objects are randomly drawn and either a must-link or a cannot-link constraint is created depending on whether the objects belong to the same class or not. The process is repeated until the desired number for each kind of constraints is reached. For each number of constraints, five different constraint sets are generated for the tests. Table 4.6 presents results when the same number $\#c$ of must-link and cannot-link constraints are added and Table 4.7 for the case when cannot-link constraints are used. Informations given are on the mean execution time μ in seconds, the coefficient of variation σ/μ for the five tests and the percentage of tests for which each system completes the search within the timeout of 30 minutes. We can see that our approach outperforms ILP in all the cases and makes better uses of the user-constraints.

Our approach was compared to the COP-kmeans algorithm [Wagstaff *et al.* 2001], which extends k-means algorithm to must-link and cannot-link constraints. This algorithm is based on a greedy strategy to find a solution that satisfies all the constraints.

#c	CP		ILP	
	μ	σ/μ	μ	σ/μ
25	969.33	51.98 %	-	-
50	43.85	46.67 %	-	-
100	0.41	49.80 %	107	72.35 %
150	0.06	22.60 %	0.8	50.00 %

#c	CP		ILP	
	μ	solved	μ	solved
100	10.32	100 %	-	0 %
125	0.35	100 %	497.60	100 %
150	0.12	100 %	13.98	100 %

Table 4.6: Time in seconds for Iris dataset (left) and Wine dataset (right) with #c must-link and #c cannot-link constraints.

#c	CP		ILP	
	μ	solved	μ	solved
50	1146.86	20 %	-	0 %
100	719.53	80 %	-	0 %
200	1130.33	40 %	-	0 %
300	743.64	60 %	-	0 %

Table 4.7: Iris dataset with #c cannot-link constraints.

When there are only must-link constraints, COP-kmeans always finds a partition satisfying all the constraints, which is a local optimum of WCSS. Nevertheless, when considering also cannot-link constraints, the algorithm may fail to find a solution satisfying all the constraints, even when such a solution exists.

We perform the same tests, but for each set of constraints, COP-kmeans is run 1000 times and we report the number of times COP-kmeans has been able to find a partition. Figure 4.2 shows the percentage of successes when cannot-link constraints are added (left) and when #c must-link and #c cannot-link constraints are added (right). We can see that when the number of constraints increases, COP-kmeans fails more frequently to solve the problem. Our CP model always find a solution satisfying all the constraints. With cannot-link constraints, our model succeeds in proving the optimality for roughly 60 % cases for Iris (Table 4.7). With must-link and cannot-link constraints, it succeeds in all the cases for Iris dataset and in all the cases where $\#c \geq 100$ for Wine dataset, as shown in Table 4.6.

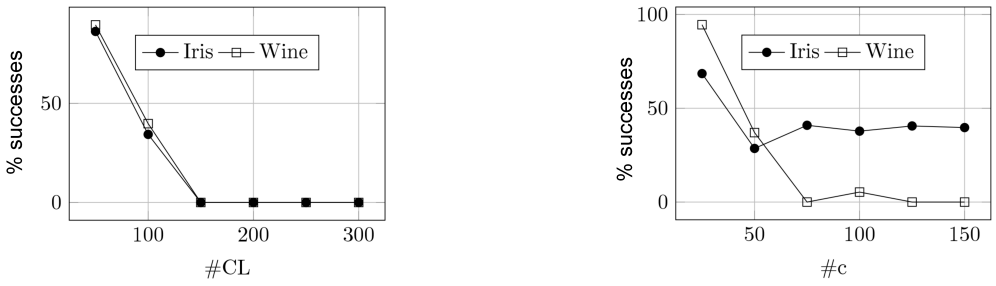


Figure 4.2: COP-kmeans with cannot-link (left), with #c must-link and #c cannot-link constraints (right)

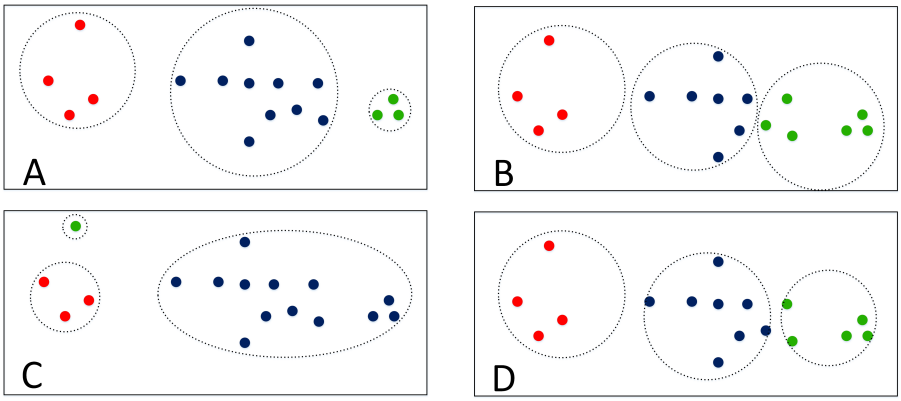


Figure 4.3: Effect with different criteria. A: intuitive groups; B: complete link; C: single link; D: WCSS

4.4 Bi-objective Constrained Clustering

We consider bi-objective clustering problem that simultaneously minimizes the maximal diameter of the clusters and maximizes the minimal split between clusters, under a set of user-constraints. Exploiting the flexibility of handling different kinds of user-constraints of our framework based on CP, we have developed an algorithm to find the exact Pareto front. In our knowledge, it is the first method for bi-objective constrained clustering. This work was published in [Dao *et al.* 2014a, Dao *et al.* 2017].

4.4.1 Bi-objective Clustering

Clustering with the criterion of minimizing the maximal diameter aims at finding homogeneous clusters, but it often suffers from the dissection effect [Cormack 1971], i.e. quite similar objects may be classified in different clusters, in order to keep the diameters small. On the other hand, clustering with the criterion of maximizing the minimal split, which aims at finding well separated clusters, often suffers from the chain effect [Johnson 1967], i.e. a chain of close objects may lead to group very different objects in the same cluster. The popular WCSS criterion, which minimizes the sum of the squared distances between points and the center of their cluster also suffers from undesirable effects. Considering this criterion, objects that should be in a large group may be classified in different clusters in order to keep this sum small. Figure 4.3 gives an illustration of these effects. Image A shows three groups that can be easily identified. Image B shows the obtained solution with the diameter criterion when the number of clusters is set to 3. In this partition, some points are very close but they are classified in two different groups. The partition obtained when considering the split criterion is shown in Image C. Because of the chain effect, the largest group contains points that are very far each from other. The optimal solution with the WCSS criterion is shown in Image D. In this partition, some points that are very close are grouped in different clusters.

A good partition with homogeneous and well-separated clusters should have a minimal diameter and a maximal split. Unfortunately, such a partition in general does not exist, since the two criteria are often conflicting. This problem can be modeled by considering

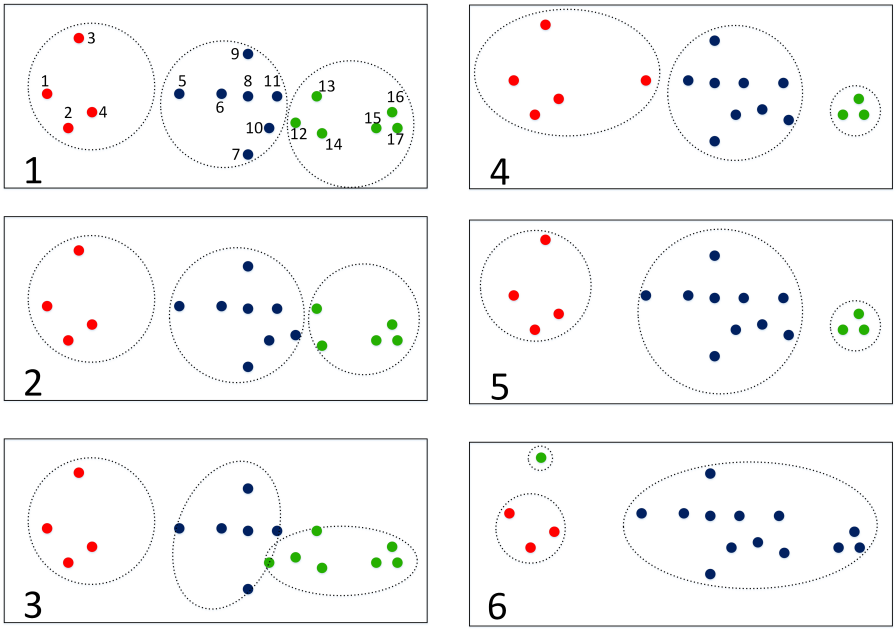


Figure 4.4: Pareto optimal solutions

the bi-criterion of maximizing the minimal split between clusters and minimizing the maximal diameter, as introduced in [Delattre & Hansen 1980]. Considering these two criteria together is natural and allows to capture both the homogeneity and the separation requirements for a good clustering. A general approach for handling two optimization criteria is to find the Pareto optimal solutions. A Pareto optimal solution is a solution such that it is not possible to improve the value of one criterion without degrading the value of the other one. If the user specifies a function on the criteria to optimize, for example $\max(S/D)$ or $\min[\alpha D - (1-\alpha)S]$ with $0 \leq \alpha \leq 1$, the optimal solution will be among the Pareto optima.

Let us consider, for instance, the example given in Figure 4.3. When the number of classes is set to 3, a complete and minimal set of Pareto solutions is given in Figure 4.4. If the ratio S/D is minimized, the optimal solution is solution 5, which is the one that fits the best the intuitive groups. The user can specify conditions on the desired solutions. If for instance it is specified that points 5 and 14 must be in the same cluster, then only solutions 5 and 6 are found. If another condition is added, requiring the size of each group to be at least 2, only solution 5 is found.

A bi-criterion clustering algorithm finding a complete and minimal set of Pareto solutions for different values of the number k of clusters is proposed in [Delattre & Hansen 1980]. When $k = 2$, an exact polynomial algorithm is proposed in [Wang *et al.* 1996, Wang & Chen 2012]. However, to the best of our knowledge, there is no algorithm dealing with this bi-criterion, while supporting various kinds of user constraints.

For the split-diameter bi-criterion optimization without user-constraints, an algorithm finding a complete and minimal set of Pareto optimal solutions, which are partitions with at most k_{max} clusters, is proposed in [Delattre & Hansen 1980]. It is proved that for n points, regardless of the number of classes k, regardless of the partition, the split value

can be found among the edges of the minimum weight spanning tree which is constructed from the matrix of dissimilarities between objects. These values are ordered decreasingly and the split s will take value in this order. On the other hand, the diameter value is one of the dissimilarities between two objects. All the dissimilarities are ordered decreasingly and the diameter d will take value in this order. Each couple (s, d) is considered and in case without conflict will induce a graph. Graph coloring on the induced graph helps to find a partition with minimum number of clusters (this number is the chromatic number of the induced graph). The algorithm finds a complete and minimal set of Pareto optimal solutions. Each solution is a partition with at most k_{max} classes.

In the case of bi-partition ($k = 2$), an exact polynomial algorithm to find Pareto optimal solutions is proposed in [Wang *et al.* 1996, Wang & Chen 2012]. For $k > 2$, [Wang & Chen 2012] also offers a 2-approximation algorithm. These two algorithms are both based on the principle of [Delattre & Hansen 1980]: a spanning tree is built to find the possible values for split and graph coloring tests are used to verify if a dissimilarity can be the maximal diameter. However none of these bi-criterion cluster analysis approaches does support any kind of user-constraints.

Multi-view spectral clustering is an extension of spectral clustering to multi-view datasets. Instead of combining different views into a single objective function, Davidson *et al.* propose in [Davidson *et al.* 2013] a natural formulation that treats the problem as a multi-objective problem and solve it using Pareto optimization.

4.4.2 Bi-objective Optimization and Exact Pareto Front Computation

In a decision problem, several criteria usually need to be optimized together. Although the criteria could be combined together in a single criterion to be used in a mono-objective problem, the simplification in general may not present faithfully the different aspects of the initial problem. Multi-objective optimization problems consider the simultaneous optimization of two or more objectives. A multi-objective optimization problem is defined by [Miettinen 1998]:

$$\begin{array}{ll} \text{minimize} & \{f_1(\overline{x}), \ldots, f_m(\overline{x})\} \\ \text{such that} & \overline{x} \in \mathcal{S} \end{array}$$

where $m \geq 2$ objective functions $f_i : \mathbb{R}^n \to \mathbb{R}$ are optimized. The vector $\overline{x} \in \mathbb{R}^n$ is composed by n decision variables. $\mathcal{S}$ is a non empty set of feasible solutions, which is a subset of the decision space $\mathbb{R}^n$. The vector of the objective functions or the criteria $f(\overline{x})$, also named by $\overline{z}$, denote $(f_1(\overline{x}), \ldots, f_m(\overline{x}))^T$. The criterion space $\mathcal{Z}$ is defined by $\{f(\overline{x}) \mid \overline{x} \in \mathcal{S}\}$.

The objective functions are simultaneously optimized. If they are compatible, one optimal solution will be found where all the objectives reach their optimality. However in general the objective functions are contradictory and there does not exist a solution that optimizes all the functions at the same time. In the multi-objective context, one is usually interested in the criteria space and in comparing the vectors in this space. A relation which is usually used for the comparison is the Pareto dominance.

Definition 4.1 *(Dominance relation) Let $f(\overline{x}_1)$ and $f(\overline{x_2})$ be two vectors of $\mathcal{Z}$. We say $f(\overline{x}_1)$ dominatesf$(\overline{x}_2)$ and denoted by $f(\overline{x}_1) \prec f(\overline{x}_2)$, or $\overline{x}_1$ dominates $\overline{x}_2$, denoted by $\overline{x}_1 \prec \overline{x}_2$, if and only if $f_i(\overline{x}_1) \leq f_i(\overline{x}_2)$ for all $i \in [1, m]$ and at least one inequality is strict.*

The solutions that dominate the others but do not dominate themselves are called Pareto optimal solution. These solutions are such that we cannot improve an element of the vector without deteriorate at least another element.

Definition 4.2 *(Pareto optimality) A vector $\overline{x}$ is Pareto optimal if there does not exist $\overline{x}'$ such that $\overline{x}'$ dominates $\overline{x}$. The est of Pareto optima is denoted by $\mathcal{P}$.*

Definition 4.3 *(Pareto front) The Pareto front $\mathcal{F}$ is the set $\{f(\overline{x}) \mid \overline{x}$ is a Pareto optimal$\}$.*

Different methods were developed to solve multi-objective optimization problems [Miettinen 1998, Collette & Siarry 2002]. One of them is the method ϵ-constraint, introduced in [Haimes *et al.* 1971]. In this method, one of the objective functions is chosen to optimize, a vector of constraints is given which allows to transform the other objective functions into inequality constraints. The problem becomes:

$$\begin{array}{ll} \text{minimize} & f_l(\overline{x}) \\ \text{such that} & f_j(\overline{x}) \leq \epsilon_j, \text{ for all } j \in [1,m], j \neq l \\ & \overline{x} \in S \end{array}$$

It is proven in [Miettinen 1998] that the ϵ-constraint method allows to compute the exact Pareto front, as soon as we have the appropriate value for each ϵ_j. The choice of the value for ϵ_j becomes however very hard for general problems.

We focus now on bi-objective optimization problems, $f(\overline{x}) = (f_1(\overline{x}), f_2(\overline{x}))^T$. Without loss of generality, let us assume that $f_1(\overline{x})$ is to minimize and $f_2(\overline{x})$ to maximize. Let us assume that $\mathcal{S}$ is finite and $0 \leq f_1(\overline{x}) < \infty$ and $0 \leq f_2(\overline{x}) < \infty$ for all $\overline{x} \in \mathcal{S}$. We define two sub-problems

$P_1(f_2 > \epsilon_2)$: $P_2(f_1 \leq \epsilon_1)$:

$$\begin{array}{ll} \text{minimize} & f_1(\overline{x}) \\ \text{such that} & f_2(\overline{x}) > \epsilon_2 \\ & \overline{x} \in S \end{array} \qquad \begin{array}{ll} \text{maximize} & f_2(\overline{x}) \\ \text{such that} & f_1(\overline{x}) \leq \epsilon_1 \\ & \overline{x} \in S \end{array}$$

By choosing for ϵ the value of the objective function on a solution found, the ϵ-constraint method can be achieved by Algorithm 5. This algorithm is similar with the one in [Bérubé *et al.* 2009]. One observation is that several steps may improve the value of

Algorithm 5: Computing exact Pareto front

```
1 P ← ∅
2 ε2 ← 0
3 repeat
4 |   x̄* ← solve P1(f2 > ε2)
5 |   ε2 ← f2(x̄*)
6 |   if x̄* is not dominated in P then
7 |   |   P ← P ∪ {x̄*}
8 until x̄* = NULL;
9 Return P
```

f_2, but the value of f_1 remains unchanged. This implies that the solution $\overline{x}^*$ fount at step

i dominates the solution $\overline{x}^*_{i-1}$ found at step $i-1$, but is itself dominated by the solution $\overline{x}^*_{i+1}$ found at step $i+1$. Therefore we developed Algorithm 6 that optimizes iteratively both objective functions. At each step, one objective function is optimized, the value of the other objective function is used as a new constraint. We showed that this algorithm computes the exact Pareto front.

Algorithm 6: Computing exact Pareto front by successive optimizations

$\mathcal{P} \leftarrow \emptyset$;
$i \leftarrow 1$;
$\overline{y}^*_i \leftarrow$ solve $P_1(f_2 > -1)$;
while $\overline{y}^*_i \neq NULL$ **do**
  $\overline{x}^*_i \leftarrow$ solve $P_2(f_1 \leq f_1(\overline{y}^*_i))$;
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\overline{x}^*_i\}$;
  $i \leftarrow i+1$;
  $\overline{y}^*_i \leftarrow$ solve $P_1(f_2 > f_2(\overline{x}^*_{i-1}))$;
Return $\mathcal{P}$

4.4.3 Diameter-Split Bi-objective Constrained Clustering

We consider a constrained clustering task under a set $\mathcal{C}$ of user constraints that aims at simultaneously minimizing the maximal diameter of the clusters and maximizing the split between clusters. Since our framework using CP allows to choose one among different optimization criteria and to add user constraints, it can be directly used in Algorithm 6 to solve this problem. The integration to solve diameter-split constrained clustering is shown in Algorithm 7. The function *Maximize_Split*($\mathcal{C}$) or *Minimize_Diameter*($\mathcal{C}$) means the use of our CP model with the optimization criterion of maximizing the split or minimizing the diameter, respectively, and with the set of constraints $\mathcal{C}$. It returns an optimal solution which satisfies all the constraints in $\mathcal{C}$, if there exists one, or $NULL$ otherwise.

Algorithm 7: Algorithm computing a complete and minimal set $\mathcal{P} = \{\Delta^S_1, \ldots, \Delta^S_m\}$ of Pareto optimal solutions

Input: $\mathcal{C}$, a set of user constraints
$\mathcal{P} \leftarrow \emptyset$;
$i \leftarrow 1$;
$\Delta^D_i \leftarrow$ Minimize_Diameter($\mathcal{C}$);
while $\Delta^D_i \neq NULL$ **do**
  $\Delta^S_i \leftarrow$ Maximize_Split($\mathcal{C} \cup \{D \leq D(\Delta^D_i)\}$) ;
  $\mathcal{P} \leftarrow \mathcal{P} \cup \{\Delta^S_i\}$;
  $i \leftarrow i+1$;
  $\Delta^D_i \leftarrow$ Minimize_Diameter($\mathcal{C} \cup \{S > S(\Delta^S_{i-1})\}$) ;
Return $\mathcal{P}$

Minimize_Diameter (resp. *Maximize_Split*) searches for a partition minimizing the diameter (resp. maximizing the split) among the partitions satisfying the set of constraints

Dataset	#Sol	bGC	CP2
Iris	8	4.2	< 0.1
Wine	8	0.9	< 0.1
Glass	9	21.5	0.4
Ionosphere	6	1.8	2.6
User Knowledge	16	23.6	12.8
Breast Cancer	7	167.5	1.1
Synthetic Control	6	–	6.7
Vehicle	13	–	5.5
Yeast	–	–	–
Multi Features	15	–	229.1
Image Segmentation	8	–	41.3
Waveform	–	–	–

Table 4.8: Runtime in seconds with bi-criterion Split-Diameter

given as argument. Let us recall that there may exist several partitions optimizing a criterion but our model returns the first one found. Nevertheless it is later possible to apply our model with no optimization criterion but with the constraint that the diameter of the partitions must be this optimum and the algorithm will enumerate all the partitions satisfying this constraint. In this way, given an element (D_i, S_i) in the Pareto front, our model without optimization criteria, but with the constraints $\mathcal{C} \cup \{D = D_i, S = S_i\}$, will enumerate all the partitions Δ that satisfy $\mathcal{C}$ and such that $D(\Delta) = D_i$ and $S(\Delta) = S_i$.

Experiment results. In the case without user-constraints, we compare our method with the bi-criterion clustering algorithm based on graph coloring [Delattre & Hansen 1980] (denoted bGC). Since no implementation of the program was available, we have coded it in C++. To our knowledge, this is the only exact algorithm for $K \in [K_{min}, K_{max}]$. In the experiments, the datasets in Table 4.1 are used and the timeout is set to 1 hour. The number of classes K varies between 2 and the real number of classes. Table 4.8 gives the results of the experiments. The second column (#Sol) gives the number of Pareto optimal solutions found, or equivalently, the number of elements in the complete Pareto front. The following columns give the runtime of each approach in seconds. The two programs are exact and they find the same Pareto front. It is clear that our model is the most efficient in most cases. It takes advantage of the efficient constraint propagation mechanism to reduce the search space. As in the case of GC, the algorithm bGC is limited to datasets with less than 500 points.

For the experiments with user-constraints, we have generated randomly 80 user constraints for dataset Iris. Figure 4.5 presents the Pareto front, for five cases of 0 to 80 user-constraints. The first test is without user-constraints, the second one is with the first 20 user-constraints, the third one with the first 40 constraints and so on. As more and more user-constraints are added, the number of feasible solutions decreases and as a result, the criterion space changes significantly. Since we have generated user-constraints from the real partition, it is obvious that the point (D_r, S_r) corresponding to the real partition must be in the region delimited by each Pareto front. Therefore it must be in

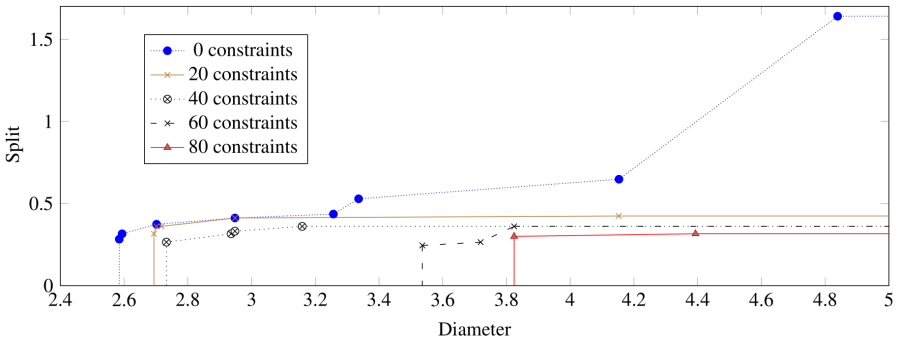


Figure 4.5: Bi-criterion constrained clustering with dataset Iris

the region delimited by the Pareto front with 80 constraints. We can see that without user-constraints, there are many points in the Pareto front but all of them are very far from (D_r, S_r). For that reason, it is useful to enable user-constraints for the task of bi-criterion clustering. Moreover, given an element (D_i, S_i) in the Pareto front, our model can be used to enumerate all Pareto optimal solutions that have the maximum diameter D_i and the minimum split S_i. For example, considering the Pareto front in the case of 80 instance-level constraints, it is composed of two points, which correspond respectively to 8704 and 4352 partitions.

4.5 Summary

We have developed a declarative framework for constrained clustering, using constraint programming. This approach has the advantage of allowing several optimization criteria and various types of user-constraints. Two main principes that make the power of constraint programming are constraint propagation and search. We have exploited these principes by developing dedicated global optimization contraints and filtering algorithms, as well as search strategies. We show that our approach has the best performance compared to existant exact constrained clustering approaches. We show that the flexibility of our framework enables a method for bi-objective constrained clustering problem.

Chapter 5

Beyond Constrained Clustering

Contents

Declarative approach using constraint programming enables the modeling of different clustering tasks as well as the integration of different types of constraints. A CP model for clustering can also be seen as a task inside a more general process. Exploiting these advantages we have explored further the use of constraints in more general clustering problems. This chapter presents different aspects of generalizing the approach using CP beyond constrained clustering. This work enjoys various collaborations: Tias Guns (Katholieke Universiteit Leuven), Ian Davidson, Chia-Tung Kuo (University of California, Davis), S. S. Ravi (University at Albany), Pierre Gançarski, Thomas Lampert (University of Strasbourg), Khanh-Chuong Duong, Willy Lesaint, Nicolas Serrette, Christel Vrain (University of Orléans).

The chapter is organized as follows. On the modeling side, we explore the use of constraint programming in more general clustering tasks. In Section 5.1, a CP framework is presented for clustering problems that combine both dissimilarity-based and conceptual-based constraints or criteria. Sections 5.2 and 5.3 present two other new clustering problems and our approach using CP for them. In Section 5.2, actionable clustering problem is developed where constraints can be given on properties to make clustering useful. Section 5.3 presents minimal clustering modification problem, which takes in input a clustering

and aims at finding a clustering with minimal changes while removing undesirable properties. On the effectiveness side, Section 5.4 addresses the constrained clustering problem with the well-known within cluster sum of squares criterion. We present an extension of RBBA, an unconstrained clustering algorithm, to user-constraints and show that the combination of this algorithm with constraint programming outperforms all existent approaches. On the application side, Section 5.5 analyzes the use of our work in the context of constrained clustering on time-series data.

5.1 Combining Dissimilarity-Based and Conceptual-Based Constraints

Dissimilarity based clustering defines the homogeneity of the clusters based on a dissimilarity measure between each pair of objects. In general, the Euclidean distance is used when all the attributes are numerical. The interpretability of the clusters is however often difficult, since the influence of the attributes is aggregated in the dissimilarity measure. Another clustering setting is conceptual clustering, when the objects are described by qualitative attributes, usually boolean descriptors. The aims is to group all the objects into clusters, such that each cluster can be identified by a set of descriptors that defines all the objects of the cluster. The interpretation of the clusters is therefore straightforward, but numerical attributes must be discretized to be qualitative attributes. The way attributes are discretized has a very strong influence on the result. We aim at exploiting the advantages of both approaches, when data is described by qualitative and/or quantitative attributes. We have generalized the framework using CP to integrate optimization criteria and constraints, which can be either concept based or dissimilarity based. Existent models using CP for conceptual clustering used binary variables. Our model distinguishes itself by taking advantage of set variables and set constraints to express conceptual constraints. This work was developed in collaboration with Willy Lesaint and Christel Vrain [Dao *et al.* 2015b].

5.1.1 Conceptual Constrained Clustering

Let $\mathcal{O}$ be a set of N objects. We aim at partitioning $\mathcal{O}$ into homogeneous clusters. The homogeneity is defined by an optimization criterion and the clustering task searches for a partition $\mathcal{C}$ that optimizes it. For each cluster $c \in \mathcal{C}$, O_c denotes the set of the objects in the cluster.

In conceptual clustering setting, the objects are described by a set of binary attributes $\mathcal{A} = \{a_1, ..., a_m\}$. In the transactional database terminology, objects are called *transactions* and attributes are called *items*. Each object $o \in \mathcal{O}$ is represented by a set $t_o \subseteq \mathcal{A}$ of attributes that the object satisfies (set of items that the transaction contains). The data can be expressed by a $n \times m$ binary matrix such that $\forall o \in \mathcal{O}, \forall a \in \mathcal{A}$, $t_{oa} = 1$ if and only if object o satisfies attribute a.

Two notions *extension* and *intention* are defined. The extension of a set of attributes $A \subseteq \mathcal{A}$ is the set of the objects having all the attributes in A: $\text{ext}(A) = \{o \in \mathcal{O} \mid A \subseteq t_o\}$. The intention of a set of objects $O \subseteq \mathcal{O}$ is the set of the attributes satisfied by all the objects in O: $\text{int}(O) = \{a \in \mathcal{A} \mid \forall o \in O,\ a \in t_o\}$. A cluster c is represented by a couple

(O_c, A_c), where $O_c \subseteq \mathcal{O}$ and $A_c \subseteq \mathcal{A}$. Here O_c is the set of objects in the cluster and A_c the set of attributes that characterizes the cluster. A cluster c is a concept in sense of Formal Concept Analysis if and only if $O_c = \text{ext}(A_c)$ and $A_c = \text{int}(O_c)$. This condition defines a closure property $A = \text{int}(\text{ext}(A))$.

Let us consider for instance a dataset $\mathcal{O}$ of five objects described on the attributes $\mathcal{A} = \{a_1, \ldots, a_4\}$ as follows.

	a_1	a_2	a_3	a_4
o_1	1	1	1	0
o_2	1	1	0	0
o_3	1	1	0	1
o_4	1	0	1	1
o_5	0	1	1	1

The cluster $(\{o_1, o_2\}, \{a_1, a_2\})$ is not closed, since o_3 satisfies also a_1 and a_2. On the other hand, the clusters $(\{o_1, o_2, o_3\}, \{a_1, a_2\})$ and $(\{o_4, o_5\}, \{a_3, a_4\})$ are closed and are therefore concepts. The requirements of conceptual clustering tasks are expressed by constraints and optimization criteria [Guns *et al.* 2013, Guns *et al.* 2011] as follows.

Concept based constraints.

- The *extension constraint* states that for all $c \in \mathcal{C}$, we must have $O_c = \text{ext}(A_c)$. Let us notice that $O_c \subseteq \text{ext}(A_c)$ means that each object in O_c has all the attributes of A_c and $\text{ext}(A_c) \subseteq O_c$ means that no other object of $\mathcal{O} \backslash O_c$ has all these properties. Since $\mathcal{C}$ is a partition, with the extension constraint, there do not exist two clusters c and c' such that $A_{c'} \subseteq A_c$. Indeed, if there exist $c \neq c'$ such that $A_{c'} \subseteq A_c$, so $\text{ext}(A_c) \subseteq \text{ext}(A_{c'})$, we then have $O_c = \text{ext}(A_c) \subseteq \text{ext}(A_{c'}) = O_{c'}$, which contradicts that $O_c \cap O_{c'} = \emptyset$.

- The *intention constraint* states that for all $c \in \mathcal{C}$, we must have $A_c = \text{int}(O_c)$.

- The *closed constraint* requires that each cluster is a concept, in sense of Formal Concept Analysis: $\forall c \in \mathcal{C},\ O_c = \text{ext}(A_c)\ \wedge\ A_c = \text{int}(O_c)$.

- A *cluster size constraint*, also called frequent constraint requires that each cluster must have at least/at most a number of objects: $\forall c \in \mathcal{C}, |O_c| \lessgtr b$.

- A *concept size constraint* requires that each cluster must be characterized by at least/at most a number of attributes: $\forall c \in \mathcal{C}, |A_c| \lessgtr b$.

Concept based optimization criteria. In order to search for a more balanced set of clusters, the following optimization criteria are proposed:

- maximizing the size of the smallest cluster, in order to balance the size of the clusters: maximize$(\min\{|O_c| \mid c \in \mathcal{C}\})$.

- maximizing the size of the smallest concept, in order to balance the size of the concepts: maximize$(\min\{|A_c| \mid c \in \mathcal{C}\})$.

The seminal work that developed a declarative framework for itemset (pattern) mining using CP was [De Raedt *et al.* 2008]. Other frameworks for itemset mining have been developed based on SAT [Métivier *et al.* 2012b, Jabbour *et al.* 2013] or Answer Set Programming [Järvisalo 2011]. Itemset mining has been then generalized to k-pattern set mining, formalizing in this way conceptual clustering. In [Guns *et al.* 2013], k-pattern set mining tasks are modeled in Constraint Programming, as well as most of classical tasks in Data Mining. Conceptual clustering is represented by a k-pattern set mining task with specific constraints. In the same line, a constraint-based language is proposed in [Métivier *et al.* 2012a]. Conceptual clustering tasks can be formulated by queries in this language, which are translated into SAT clauses and solved by a SAT solver. In our knowledge, until our work, no work however had integrated both conceptual and dissimilarity based settings. Recent work has developed efficient formulations for conceptual clustering using integer linear programming [Ouali *et al.* 2016]. Exploiting further the use of set constraints as in our model, a formulation using constraint programming has recently been developed in [Chabert & Solnon 2017] for conceptual clustering and bi-objective conceptual clustering.

5.1.2 Models for Dissimilarity and Conceptual Constrained Clustering

We have developed a framework that enables the consideration of heterogenous data, which can be characterized by both numerical (quantitative) attributes and symbolic (qualitative) attributes. The user can specify among the attributes the ones used to define concepts and the ones used to compute the dissimilarity measure. The framework therefore offers a variety of modeling tasks that are purely conceptual or dissimilarity-based constrained clustering or that raise from conceptual (or relational) clustering in which relational (or conceptual, resp.) constraints are integrated.

The framework is generalized from our CP model for relational constrained clustering to integrate conceptual constraints. We present two models for conceptual constraints: a model using binary variables which is a direct extension of the work by [De Raedt *et al.* 2008, Guns *et al.* 2011] and a new model using set variables and set constraints.

5.1.2.1 Model Using Binary Variables

The model is an extension of the model presented in Subsection 4.2.2. The composition of the clusters is defined by an integer variable G_o for each object $o \in \mathcal{O}$, where $G_o = c$ means object o is assigned to cluster c. To represent the attributes associated to each cluster, in the same way as [Guns *et al.* 2013], we use binary variables $A : \mathcal{C} \times \mathcal{A} \rightarrow \{0, 1\}$. Here $A_{ca} = 1$ means attribute a is in A_c, the set of attributes that characterizes cluster c. Therefore $k \times m$ binary variables are needed, where k is the number of clusters and m the number of attributes.

Conceptual constraints are expressed as follows.

- Constraint on concept size:

$$\forall c \in \mathcal{C}, \quad \sum_{a \in \mathcal{A}} A_{ca} \lesseqgtr b \tag{5.1}$$

where b is the bound on the size. This constraint is expressed using k linear constraints.

- Extension constraint:

$$\forall o \in \mathcal{O}, \forall c \in \mathcal{C}, \quad G_o = c \Leftrightarrow \sum_{1 \leq a \leq m} A_{ca}(1 - t_{oa}) = 0 \tag{5.2}$$

  This is expressed using $n \times k$ reified constraints, each one is linked with a linear constraint on A.

- Intention constraint:

$$\forall c \in \mathcal{C}, \forall a \in \mathcal{A}, \quad A_{ca} \Leftrightarrow \sum_{1 \leq o \leq n} (G_o = c)(1 - t_{oa}) = 0 \tag{5.3}$$

  This constraint is expressed using $m \times k$ reified constraints.

- Closed constraint: By definition, this constraint on $\mathcal{C}$ is expressed by the constraints (5.2) and (5.3), which represent respectively the extension and intention constraints on the clusters of $\mathcal{C}$.

For the optimization criterion that maximizes the minimal concept size, an integer variable T is introduced with the domain $[1, m]$. The value of T is maximized, and the following constraints express that T is the minimal size of the concepts:

$$\forall c \in \mathcal{C}, \ T \leq \sum_{a \in \mathcal{A}} A_{ca}$$

5.1.2.2 Model Using Set Variables

In this model, set variables are used to define the set of attributes associated to each cluster $E : \mathcal{C} \rightarrow \mathcal{P}(\mathcal{A})$. For each cluster $c \in \mathcal{C}$, a set variable E_c is defined. Its value is $E_c \subseteq \mathcal{A}$ and its domain is $\mathcal{P}(\mathcal{A})$.

The data is represented using the initial set representation: each object $o \in \mathcal{O}$ is characterized by the set t_o of the attributes it satisfies. The constraints are expressed as follows.

- Concept size constraint: bound on concept size is expressed naturally by a cardinality constraint

$$\forall c \in \mathcal{C}, \ |E_c| \lesseqgtr b$$

- Extension constraint:

$$\forall c \in \mathcal{C}, \forall o \in \mathcal{O}, \ G_o = c \Leftrightarrow E_c \subseteq t_o$$

  This is expressed by $n \times k$ reified constraint, each one is linked with an inclusion constraint.

- Intention constraint:

$$\forall c \in \mathcal{C},\ E_c = \cap_{G_o=c} t_o$$

  Each constraint $E_c = \cap_{G_o=c} t_o$ is expressed by n reified domain constraints *dom* to construct the set $I_c = \{o \in \mathcal{O} \mid G_o = c\}$ and a set version *element* constraint to state the relation $E_c = \cap(\langle t_1, ..., t_n \rangle [I_c])$. The last relation means the intersection is done with the set t_i such that $i \in I_c$.

For the optimization criterion that maximizes the minimal concept size, a variable T of domain $[1, m]$ is introduced. Its value is maximized and the following constraints link it to the size of the concepts:

$$\forall c \in \mathcal{C},\ T \leq |E_c|$$

5.1.2.3 Experiment Results

The models was implemented using Gecode version 4.2.1. The dataset *Automobile* available on the UCI repository[1] was used. After removing incomplete instances, this dataset contains 193 objects (automobile models) and 23 attributes. Among the attributes, 22 attributes (9 symbolic et 13 numerical) represent technical features of the vehicles. For instance, a vehicle is characterized by its type of engine (gas oil or gas), the drive wheels (4 wheels, 2 front or 2 rear), the horse-power (from 48 to 288), etc. The last attribute is the price (from 5118 to 45400). This attribute is chosen to define the dissimilarity measure. The technical features are discretized into 64 binary attributes (for each numerical attribute, two attributes are created to divide the vehicles according to their position relative to the median).

Comparison between binary and set models. The models were compared with three cases of constraints: intention, extension and closed. The optimization criterion was minimizing the maximal diameter. The performances are reported in Table 5.1 (timeout was set to 3 hours, the character _ means the search was not completed after timeout). The results highlight the efficiency of the set model. In the binary model, the use of reified constraints linked with linear constraints does not give sufficient propagation. The use of inclusion constraint in set model offers much better propagation, the number of explored nodes is much less important.

Benefits of the unified framework. Experiments was done on this dataset with conceptual clustering, relational clustering and combinations of both. The aim is to interpret the obtained concepts for 2 or 3 clusters according to the price. Test (e) and (f) are in conceptual setting, with the *closed* constraint and maximizing the minimal cluster size (e) or maximizing the minimal concept size (f). Test (g) is in relational setting, with minimizing the maximal cluster diameter. Test (h) combines the *closed* constraint and minimizing the maximal diameter. Results are given in Table 5.2. We can observe that tests (e) and (f) in conceptual setting give cluster of balanced size, but while one would hope that prices are correlated with technical features, the clusters have large distribution on price. Test (g) in relational clustering gives clusters with small diameter but of unbalanced sizes and unbalanced concept sizes. The difference between cluster size can be explained by very

[1] `http://archive.ics.uci.edu/ml`

Constraint	k	Binary model		Set model	
		time(s)	#nodes	time(s)	#nodes
intention	2	0.10	2338	0.14	2877
	3	0.63	15109	0.55	13986
extension	2	0.03	79	0.03	52
	3	8.40	145098	0.08	979
	4	6960	>1M	0.55	8636
	5	_	_	2.49	36264
closed	2	0.03	24	0.03	47
	3	13.50	72494	0.15	963
	4	9900	>1M	1.29	8585
	5	_	_	5.47	36057

Table 5.1: Performance of binary and set models

Test	method	clust. size			concept size			diam.
(e)	conceptual	63	66	64	3	3	3	38615
(f)	conceptual	68	61	64	3	3	3	38615
(g)	relational	161	20	12	1	5	10	13226
(h)	combination	72	108	13	4	2	5	33550

Table 5.2: Results with $k = 3$

important differences between the prices in top-of-the-range cars. Note that using all the attributes for computing the dissimilarities does not provide more balanced cluster size. Test (h) provides more balanced clusters. They are more representative of the different ranges of the vehicles than the conceptual tests.

Tests (i) and (j) in Figure 5.1 highlight the interest of the unified framework thanks to user constraints and a better use of dissimilarities. Test (i) added to (h) two user constraints: a cluster size of at least 40 and a concept size of at least 2. Compared to test (h), it obtains more balanced cluster sizes. In Test (j), we consider that a top-of-the-range vehicle is both expensive and powerful. The dissimilarity is computed on price and horse-power. Using the same constraint as test (i), the obtained results show a better distribution of the objects and well defined concepts.

Test	clust. sizes			concept sizes			diam.
(i)	57	95	41	2	2	4	36479
(j)	42	56	95	7	3	2	38411

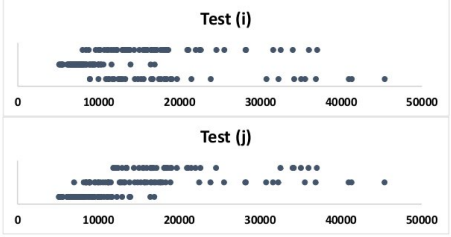


Figure 5.1: Cluster size and distribution of objects/cluster for (i) and (j)

5.2 Actionable Clustering

Constrained clustering has been developed to integrate user-constraints into clustering tasks. User-constraints can be instance-level or cluster-level constraints. Instance-level must-link and cannot-link constraints are usually generated from the labels of a few known instances, while cluster-level constraints are usually on the size or on the diameter/separation of the clusters. However, in many domains experts can provide complex constraints that are not generated from a ground truth, rather they capture what makes the clustering useful in the domain. We have characterized new types of user-constraints for this focus and we term clustering with these constraints *actionable clustering*. We show that constraint programming offers a natural formulation for these kinds of constraints. This work was presented in [Dao *et al.* 2016b] and was developed in collaboration with Ian Davidson, Khanh-Chuong Duong and Christel Vrain.

5.2.1 Constraints Categorization

Previous work on constrained clustering is most suitable for the *semi-supervised* setting where a few instances are labeled and the instance-level must-link and cannot-link constraints can be generated from them [Basu *et al.* 2008]. The data is then clustered under small numbers of these constraints with the number of clusters equaling the number of different labels. Performance is typically measured in terms of prediction: how well the clustering found matches the ground truth clustering induced by the labels. However, in many domains experts can provide complex constraints that are not generated from a ground truth, rather they capture what makes the clustering useful in the domain.

Consider if you wish to cluster your ego network so as to find several useful groups each of which you can invite to a different diner party. You may require that each cluster must contain equal number of males and females, and the difference in term of age of a cluster is at most 10 and that each person in a cluster should have at least 3 other people sharing the same hobby. A clustering algorithm may find a useful grouping that results in a successful party but is unlikely to unless we somehow encode what is required.

Existing instance-level constraints cannot be used to specify this type of guidance. Consider the constraint that the number of males and females in each cluster should be approximately equal. Since instance-level constraints are specified before the algorithm execution they typically cannot constraint any property that dynamically changes during the algorithm execution such as the cluster composition. Constraining cluster level properties has probably not been well studied as it is challenging to do so in procedural languages but can be elegantly performed in constraint programming, which we use in our approach.

Up to this point we have discussed finding clusters which are actionable since they meet a particular set of requirements. However, it is also likely that a clustering is actionable because it *does not* contain a set of properties. This idea of using negative feedback was first explored in the alternative clustering literature [Qi & Davidson 2009, Dang & Bailey 2010]. There the problem was given a good (according to the objective function value) clustering Π which is not actionable (perhaps because it is trivial or inappropriate), find an alternative clustering Π' such that Π' has a good objective function value but Π and Π' are different using some sort of measure such as the Rand index.

However, that work is limited in that *how* Π' is different to Π is not controlled. Instead with actionable clustering, if the existing clustering has undesirable properties such all females in one cluster, we can explicitly require that females be equally distributed (i.e. $\forall i, j$ $\texttt{CountFemale}(\pi_i) \approx \texttt{CountFemale}(\pi_j)$ where π_i is the i^{th} cluster).

The Actionable Clustering Problem. Consider the classic clustering problem. We are given a dataset $\mathcal{X}$ where each instance $x \in \mathcal{X}$ is described by a vector of features f. Typical objective functions include minimizing the vector quantization error (k-means), minimizing graph cuts (spectral methods) if the data is represented as a graph and a similarity measure is used, and optimizing cluster properties such as minimizing the maximum cluster diameter. In our work we present the novel extension that each instance is further described by a set of properties from which the definitions of what is actionable/interesting is given. In our formulation to separate the features and properties we use the notation: x_i^f and x_i^p to represent the features and properties of the i^{th} instance respectfully. The feature vectors $\mathcal{X}^f$ are used to calculate the clustering objective function value and the constraints are enforced on the property vectors $\mathcal{X}^p$. However, there is nothing stopping the same attribute of an instance being in both vectors and used as both a feature and property.

Formally the actionable clustering problem is formulated as finding a partition that optimizes an objective function on $\mathcal{X}^f$ and that satisfies all the constraints on $\mathcal{X}^p$. The type of constraints we explore can be divided into four categories: i) cardinality, ii) density, iii) geometric and iv) complex logical combination of these constraints. They are not the only relevant ones but the most pragmatic in clustering: cardinality constraints are useful for categorical attributes, density constraints for relational information and geometric constraints for real value attributes. It is important to note that these constraints can be applied simultaneously for multiple different properties on multiple clusters.

i) Cardinality constraints place a requirement on a count of the elements in a cluster having a property. They may be as simple as each cluster should contain at least one female to more complex variations such as the number of males must be no greater than two times the number of females.

ii) Density constraints relate to a cardinality constraint in that it provides requirements on a count of a property except not for an entire cluster but rather a subset of instances in the cluster. For example, we may require each person have at least 10 people in his/her cluster sharing the same hobby.

iii) Geometric constraints place an upper or lower bound on some geometric property of a cluster or cluster combination. Examples include that the maximum diameter of a cluster with respect to the age property is 10 years. This would prevent clusters containing individuals with a wide range of ages.

iv) Complex logic constraints express logic combinations of constraints, which can be instance-level or cluster-level constraints. For instance, we may require that any cluster having more than 2 professors should have more than 10 PhD students.

5.2.2 A CP Formulation for Actionable Clustering

Our framework using CP for constrained clustering (cf. Section 4.2) can be extended to integrate these categories of constraints. We present below schemes to express these constraints using CP constraints.

Cardinality constraints. Cardinality constraints allow to express requirements on the number of instances that satisfy some conditions in each cluster. The condition can be for instance being more than 20 years old and the cardinality constraint can state that each cluster must have more than 30 persons being more than 20 years old. The minimal capacity constraint is then a special case of a cardinality constraint.

Given a condition, the set C of the instances that satisfy it can be computed and the number of instances of C that are in a cluster k can be captured using the CP cardinality constraint and a variable Y_k:

$$\#\{i \in C \mid G_i = k\} = Y_k \tag{5.4}$$

The constraint $\sum_{k=1}^{K} Y_k = |C|$ enforces the link between the variables Y_k. Cardinality constraints are then expressed by arithmetic constraints on Y_k. Let us illustrate this by an example.

- *In each cluster, the number of teachers must be no less than half the number of students.* Let C_t and C_s be the sets of instances that are teachers and students, respectively. For $k \in [1, K]$, constraints similar to Equation (5.4) are put with the variables T_k and S_k to capture the number of teachers or students in the cluster k. These variables are linked by the constraint $2T_k \geq S_k$. The number of new variables is $2K$ and the number of constraints is $3K + 1$.

Density constraints. Density constraints provide bounds on the occurrence of some properties on a subset of instances in each cluster. For instance, each person being more than 20 years old should have in his/her cluster more than 5 persons sharing the same hobby. Density constraints allow a more general form than the basic ϵ-ball count constraint [Davidson & Ravi 2007]. To express this constraint, for each instance $i \in [1, N]$ which is eligible (eg. more than 20 years old), the set of neighborhood instances $NI(i)$ (eg. persons having the same hobby) is determined. The number of instances of $NI(i)$ in the same cluster as i can be captured using the variable Z_i and:

$$\#\{j \in NI(i) \mid G_j = G_i\} = Z_i \tag{5.5}$$

Arithmetic conditions are then stated on Z_i to express density constraints. Let us take the following example.

- *In the same cluster, each person should have at least* 5 *persons having the same hobby.* For each instance i, we compute the set $NI(i) = \{j \in [1, n] \mid \mathit{hobby}(i) = \mathit{hobby}(j)\}$. The fact that there must be at least 5 other persons of $NI(i)$ in the same cluster as i means the value of G_i must be taken at least 6 times by the elements in $NI(i)$. Therefore the constraint of Equation (5.5) is put as well as the constraint $Z_i \geq 6$.

In these cases, the number of new introduced variables is N and the number of CP constraints is $2N$. Let us notice that the computation of the neighborhoods is done only once before putting CP constraints.

Geometric constraints. Geometric constraints allow to set bounds on some geometric properties inside each cluster, or between the clusters. For instance, each cluster must have a difference in age of at most 20. These properties therefore can be used to define a dissimilarity measure and the *diameter* or *split* can be used to express these constraints.

A geometric constraint can also place a bound on the sum of all the values on some properties inside each cluster, or a condition on the ranges of some properties of the clusters. For instance, age ranges of the clusters should or should not overlap, or the total sum of age in each cluster must not exceed some value.

- *The average age in each cluster must not exceed 50.* To express this constraint, for each instance i and each cluster k, we introduce a boolean variable $B_{ik} \in \{0, 1\}$ ($0{:}false$ and $1{:}true$). A reified constraint [2] $B_{ik} \leftrightarrow (G_i = k)$ is put, ie. B_{ik} represents whether or not instance i is in the cluster k. For each $k \in [1, K]$, the sum of age S_k and the cardinality C_k are linked by:

$$\sum_{i \in [1,N]} age(i) B_{ik} = S_k \quad \text{and} \quad \#\{i \in [1, N] \mid G_i = k\} = C_k$$

  The bound is therefore expressed by: $S_k \leq 50 C_k$. In this case, $N \times K$ boolean variables (B_{ik}), K float point value variables (S_k) and K integer value variable (C_k) are introduced. This case is expressed by $3K$ CP constraints.

- *Constraints on the property ranges of the clusters.* We consider constraints that state conditions on the ranges of the clusters on a property p. To capture the range of a cluster, for each cluster k, we introduce the variables Min_k and Max_k that represent the minimal and the maximal values on the property p of the elements in the cluster k. Let m_p be the maximal value of the property p for all the instances. The minimal and maximal values are linked by the following constraints:

$$Min_k = \min_{i \in [1,n]}(p(i) B_{ik} + m_p(1 - B_{ik}))$$
$$Max_k = \max_{i \in [1,n]}(p(i) B_{ik})$$

  The constraint that the ranges on p of the clusters should not overlap can be expressed by putting, for any two clusters k, k',

$$(Min_k > Max_{k'}) \vee (Min_{k'} > Max_k)$$

  A constraint stating that the range of a cluster k must be included in the range of another cluster k' can be expressed by:

$$Min_k \geq Min_{k'} \text{ and } Max_k \leq Max_{k'}$$

  This requires $N \times K$ boolean variables and $2K$ float point value variables. The number of constraints is linear on K.

[2] A reified constraint on a constraint c, stated by $B \leftrightarrow c$, links the truth value of a constraint c to a boolean variable B: B is 1 if the constraint c is satisfied, 0 if c cannot be satisfied, or $B \in \{0, 1\}$ if the satisfaction of c has not yet been determined.

Complex logic constraints. Complex logic constraints can be used to enhance the expressivity power of formulating knowledge. This can be done in CP using reified and Boolean constraints as shown by the following examples.

- *Two instances* $3, 9$ *are in the same cluster if the instances* $11, 15$ *are in different clusters.* Two Boolean variables B_1, B_2 are introduced with the constraints: $B_1 \leftrightarrow (G_{11} \neq G_{15})$, $B_2 \leftrightarrow (G_3 = G_9)$ and $B_1 \leq B_2$.
- *Any cluster having more than 5 professors must have at least 10 PhD students.* For each $k \in [1, K]$, let P_k and S_k be the variables that capture the number of professors and students in the cluster k, using cardinality constraints such as in Equation (5.4). Two Boolean variables BP_k and BS_k are introduced and linked by $BP_k \leftrightarrow (P_k \geq 5)$, $BS_k \leftrightarrow (S_k \geq 10)$, and $BP_k \leq BS_k$.

5.2.3 Analyzing the Use of Constraints

The framework was implemented using the CP solver library Gecode 4.3.3. The objective function in the experiments was to minimize the maximal diameter of the clusters. All experiments were performed on a 3.4GHz Intel Core i5 processor with 8Gb of RAM under Ubuntu 14.04. We give here some elements to analyze the interest of constraints in clustering.

Improving Semi-Supervised Clustering Results. In the semi-supervised learning setting typically labels on a subset of instances are used to generate instance-level constraints such as must-link or cannot-link constraints. Here we explore if the labels can be better exploited by inferring more complex constraints on the clusters. We illustrate this point on seven UCI datasets (their properties are given in Table 4.1, Section 4.2). Rather than using the labelled data to only generate must-link or cannot-link constraints, we use the labels to provide upper and lower bound estimates on the cluster sizes. In these experiments all the objects of the datasets are considered and the number K of clusters is set to the true number of classes for each dataset. Performance is typically measured in terms of prediction: how well the found clustering matches the ground truth clustering. To measure the accuracy of a clustering P compared to the ground truth clustering P^*, we use the Rand Index [Rand 1971] which is defined by $RI = (a + b)/(a + b + c + d)$, where a and b are the numbers of pairs of instances for which P and P^* are in agreement (a, or b, is the numbers of pairs of instances that are in the same class, or respectively in different classes, in both P and P^*), c and d are the numbers of pairs of instances for which P and P^* disagree (same class in P but different classes in P^* and vice versa). This index varies from 0 to 1 and the better the partitions are in agreement, the closer RI to 1.

Instance-level constraints can decrease the quality of the found clustering compared to the ground truth clustering, as was reported in an earlier work [Davidson *et al.* 2006]. We consider the datasets Iris, Wine, Breast Cancer. All the attributes are considered as features ($\mathcal{X}^f$) in order to compute pairwise Euclidean distance and they are also considered as properties ($\mathcal{X}^p$). We generate a number of randomly created (from labels) instance-level constraints as is standard [Basu *et al.* 2008]: two instances are randomly taken, whether their labels are the same or not a must-link or a cannot-link constraint is stated, and this is repeated until required the number of instance-level constraints is reached. On those same

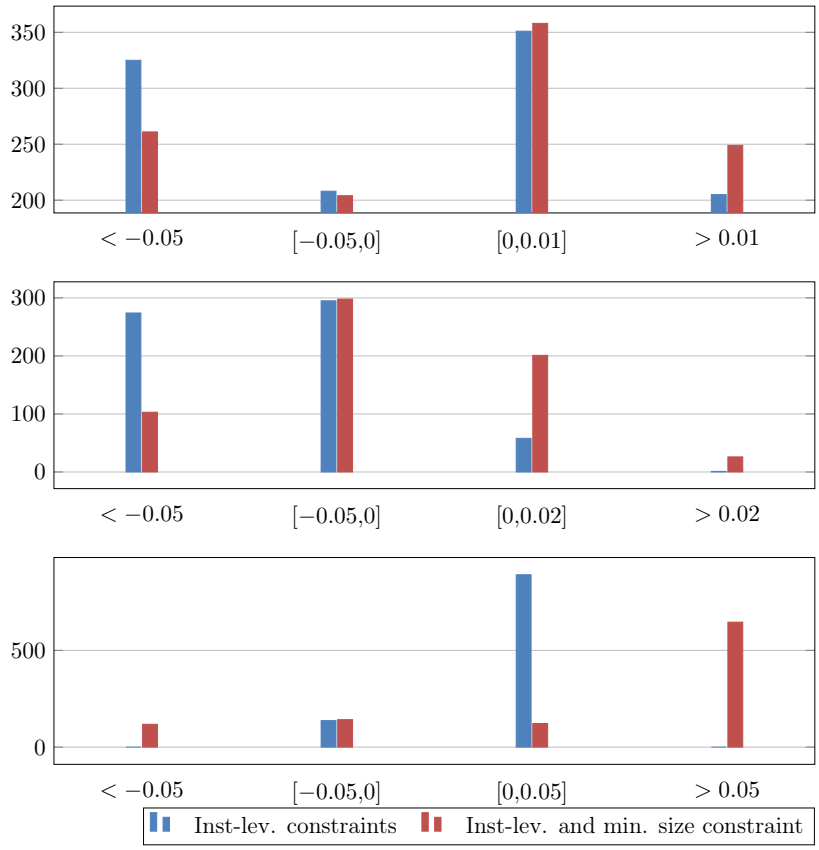


Figure 5.2: The frequency distribution (y-axis) over the 1000 experiments by the amount of increase or decrease in accuracy over not using any constraints (x-axis). Ordering of figures are for Iris, Wine and Breast Cancer each one with 30 instance-level constraints. As we can see instance-level + cardinality constraints (red-bar) produces more increases and less decreases in accuracy.

taken instances we generate a minimum cardinality constraint for all clusters as follows: let the whole dataset be of N instances, the labeled sample be of e instances, and the smallest cluster size observed on the labeled sample be m, then for the whole dataset, the smallest cluster size is set to $0.9\frac{m}{e}N$. This simulates a user guess at how big the smallest cluster should be based on the less frequent occurring label.

We compare two cases: first, clustering with a set of randomly generated instance-level constraints and second, clustering with the very same instance-level constraints and a minimum cardinality constraint as described above. We perform 1000 experiments and analyze the distribution over all experiments of the accuracy decrease or increase over not using any constraints. Figure 5.2 shows that adding instance-level constraints decreases the quality in a large number of cases which agrees with [Davidson *et al.* 2006]. On the other hand, the improvement is more stable when using a minimum cardinality constraint along with instance-level constraints. This is most like because enforcing a cardinality constraint prevents skewed cluster sizes which can yield poor performance.

To analyze the use of cardinality constraints, four cases are considered for the datasets Breast cancer, Synthetic control, Multiple feature and Image segmentation. In the first case, there is no user constraint. In the second case, 20 instance-level constraints are

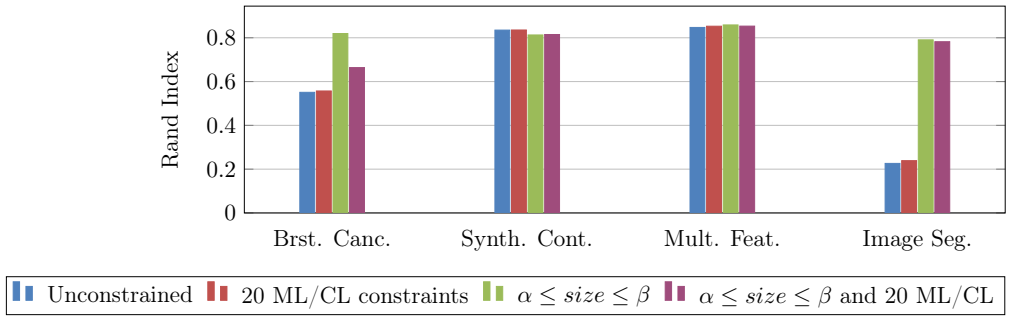


Figure 5.3: Rand Index in different cases: unconstrained, with instance-level constraints and/or minimal and maximal size constraints

randomly generated and added. In the third case, the constraints requiring that the cluster size must be between α and β are added, where α and β are respectively 150 and 400 for Breast cancer, 50 and 150 for Synthetic control, 50 and 350 for Multiple feature and 200 and 400 for Image segmentation. In the fourth case both cluster size constraints and 20 instance-level constraints are added. For the second and the fourth cases, we make 100 runs and report the average Rand Index of all the runs. We can observe different behaviors with cluster size constraints here. While for Synthetic control, the constraints slightly decrease the Rand Index, for Multiple Feature, they bring slight improvement. The most significant improvement is observed for Image Segmentation. This dataset is composed by 2100 objects of 7 classes with 300 objects per class. In the unconstrained case, the clustering found has the Rand Index 0.226314 and is very unbalanced, with two clusters of 1 object and with a large cluster of 1972 objects. The situation is not improved with 20 random must-link or cannot-link constraints, the clusterings found are always unbalanced. Adding cluster size constraints such that the clusters must have their size between 200 and 400, the clustering found has the Rand Index grow to nearly 0.80. On Breast cancer we can see that with only size constraints, the Rand Index is over 0.80, while the same constraint together with 20 instance-level constraints gives the Rand Index only about 0.66.

Guided Alternative Clustering. In the area of alternative clustering one tries to find an equally good alternative to a given clustering [Qi & Davidson 2009, Dang & Bailey 2010, Truong & Battiti 2015]. However, it is somewhat unrefined in that no guidance can be given to specify how the clustering is to differ from the given clustering. To address this issue we consider the UCI Pen Digit dataset where each instance corresponds to a single digit and has 16 attributes, which represent the 8 x, y positions of the pen as the digit is being written. All the 16 attributes are considered as features ($\mathcal{X}^f$) in order to compute pairwise Euclidean distances and are also considered as properties ($\mathcal{X}^p$). We use 1000 random instances of the dataset. We aim to find alternative ways that people write digits and we consider the simplest case where the number of clusters $k = 2$. This effectively forms a dichotomy of the two ways people in the data set write their digits.

For each cluster, the centroid is computed, which is considered as the representative of the cluster and can be easily visualized to show the underlying digit prototype the cluster represents. In the case of minimizing the maximal cluster diameter and without

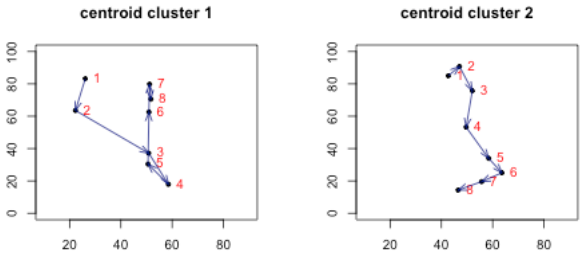


Figure 5.4: The centroids of the clustering found without any constraints. Time=1.16s, maximal cluster diameter=263.58. Arrows indicate pen movement.

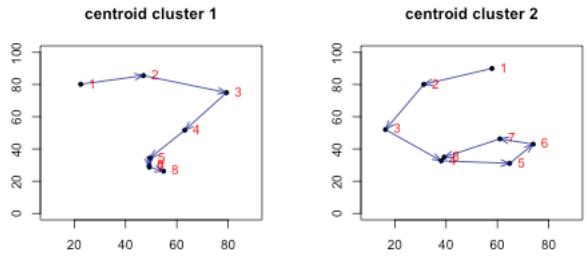


Figure 5.5: The centroids of the clustering found with a diameter constraint on the horizontal value of the third time step. Time=0.02s, maximal cluster diameter=291.50. Arrows indicate pen movement.

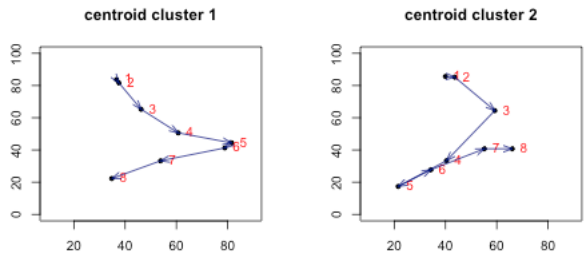


Figure 5.6: The centroids of the clustering found with a diameter constraint on the horizontal value of the fifth time step. Time=0.02s, maximal cluster diameter=269.68. Arrows indicate pen movement.

any constraints, the centroids of the found clusters, which represent two types of writing, are represented in Figure 5.4. The clustering found is the *global optimum* of the clustering algorithm objective function. However the centroids are not really meaningful unless of saying that in one way people write from up to down then up again, and in the other way only from up to down. Instead we wish to find an actionable alternative by adding a diameter constraint on the horizontal position of the pen at the 3^{rd} time step. This effectively means that all digits in the same cluster must have a similar horizontal pen location at the third time step. We obtain a different clustering whose centroids are shown in Figure 5.5 but whose quality, in term of the objective function, is comparable to the first clustering found in Figure 5.4. These centroids have the 3^{rd} positions respectively on the right and on the left. The centroids can give an interpretation such that in one way people write from left to right and in the other way like a spiral from right to left and in both ways from up to down. By adding a diameter constraint on the horizontal position of the pen at the 5^{th} time step, we obtain another very different clustering whose centroids are shown in Figure 5.6. Again the centroids have the 5^{th} position either on the right or on the left, and the quality of this clustering is comparable to the initial clustering in Figure 5.4. This is an example of guided alternative clustering, which unlike earlier work [Davidson & Qi 2008, Dang & Bailey 2015] did not find an arbitrary alternative clustering, rather we find one with specific properties. Adding constraints can deteriorate the quality in term of the objective function, however the flexibility of our framework allows to control the gap between the constrained case and the unconstrained case by means of constraints. For instance let the maximal diameter of the clusters in the unconstrained case be D_{opt}, one can require an actionable alternative clustering with both a diameter constraint on the horizontal position of the pen at the 3^{rd} time step and another constraint stating that the maximal diameter of the clusters does not exceed $1.2D_{opt}$.

Computational Effect of Constraints. To address the computational effect of constraints, we report times taken by different cases considered previously. The total runtimes (stating constraints and search) are presented in Table 5.3, where +1800 means the solver did not complete the search after 30 minutes. Cluster size constraints can give large variations in runtime. One explanation is that the efficiency of a CP framework depends on the power of constraint propagation. For extended cardinality constraint filtering the domains of all variables to arc-consistency [Quimper *et al.* 2004] is NP-hard. However, when setting an upper bound and a lower bound on the count variables, efficient filtering algorithms have been developed [Régin 1996], which help pruning the search space.

For other kinds of constraints we consider the census dataset available at UCI[3]. This dataset has 48,842 instances, each one is described by 14 attributes with 6 continuous and 8 symbolic. We choose 5 continuous attributes (age, capital-gain, capital-loss, hours-per-week and fnlwgt) as features ($\mathcal{X}^f$) to compute distances on. All of the 14 attributes are used as properties. We generate 5 samples each one of 1000 instances and for each sample we conduct experiments with 5 use-cases described in Table 5.4. In each use-case, the number of clusters is set to 2 and to 3. Tables 5.5 and 5.6 give average runtime across five samples for the use cases with $K = 2$ and $K = 3$ respectively.

We can see from Table 5.5 that the run-time to find the best solution in the uncon-

[3]https://archive.ics.uci.edu/ml/datasets/Adult

Dataset	N	K	case	time (s)
Breast cancer	569	2	unconstrained	0.30
			$150 \leq size \leq 400$	0.70
Yeast	1484	10	unconstrained	3.15
			$10 \leq size \leq 500$	11.56
Synthetic Control	600	6	unconstrained	0.49
			$50 \leq size$	+1800
			$size \leq 150$	2.58
			$50 \leq size \leq 150$	3.02
Multiple Feature	2000	10	unconstrained	10.30
			$50 \leq size$	76.80
			$size \leq 350$	+1800
			$50 \leq size \leq 350$	71.04
Image Segmentation	2100	7	unconstrained	3.29
			$200 \leq size$	86.85
			$size \leq 400$	+1800
			$200 \leq size \leq 400$	92.00

Table 5.3: Total runtime in seconds for (un)constrained cases with cardinality requirements.

1	*No constraints.*
2	*Cardinality constraint.* A cardinality constraint is added, which requires that in each cluster, the ratio of $\#female_c/\#male_c$ for the cluster c is between a half and twice the ratio of females and males in the sample.
3	*Density constraint.* A constraint is added, stating each person of age between 20 and 50 must have at least 10% of people with the same work occupation in the same cluster.
4	*Diameter Geometric constraint.* A constraint is added, which states that the difference in age in each cluster must not exceed $2(\max(age) - \min(age))/3$.
5	*Complex logic constraint.* A constraint is added, which states that a cluster having more than 20 persons younger than 20 should have more than 30 persons older than 45, and each cluster has at least 100 persons.

Table 5.4: The five use cases for testing the computation time effect of constraints.

Sample	UC1	UC2	UC3	UC4	UC5
1	0.42	0.41	2.51	0.42	0.49
2	0.32	0.69	1.71	0.32	0.35
3	0.44	0.54	2.94	0.41	0.34
4	0.44	0.31	0.71	0.32	0.34
5	0.36	0.48	2.16	0.32	0.32

Table 5.5: The runtime (seconds) for use cases (see Table 5.4) across five samples, for $K = 2$.

Sample	UC1	UC2	UC3	UC4	UC5	UC3+4
1	0.32	1.93	+1800	0.40	0.36	4.14
2	0.44	2.79	6.82	0.44	0.45	2.72
3	0.50	2.23	+1800	0.48	0.40	+1800
4	0.44	0.33	+1800	0.86	0.34	1.19
5	0.33	2.30	+1800	0.36	0.32	2.91

Table 5.6: The runtime (seconds) for use cases (see Table 5.4) across five samples, for $K = 3$. Note how using UC3 and UC4 together mitigates the increases of just using UC3.

strained setting is under 0.5 second (use case 1) for all the 5 samples. Use cases 2, 4 and 5 take comparable run-time. Use case 3, which is expressed by a large number of CP cardinality constraints, is the most difficult among all the use cases. This trend is confirmed with $K = 3$ (Table 5.6), for the solver does not complete the search after the timeout of 30 min for 4 of the 5 samples.

One explanation for this variety of run times is that some constraints when added help the solver to prune the search tree at the top levels which has a large effect. Some other constraints, for instance, the cardinality constraint, however are useful in pruning the search tree only in more deeper levels towards the leaf nodes reducing down the benefits of pruning. On the other hand, constraints such as the diameter geometric constraint are useful in general. We have combined the diameter geometric constraint of use case 4 with the constraint of use case 3. The run-times of the combination reported in the last column of Table 5.6 have almost dropped for most of the samples.

5.3 Minimal Clustering Modification

The clustering problems presented so far take in input a dataset and find a clustering according to some criteria and to some constraints. Consider now the situation where you have already a clustering of a dataset. This clustering can be obtained by your favorite clustering algorithm applied to the dataset and it is a good clustering but there are a few undesirable properties. You would like to remove the undesirable properties meanwhile do not change much the clustering. One adhoc way to fix this is to re-run the clustering algorithm and hope to find a better variation. Instead, we propose to not run the algorithm again but *minimally* modify the existing clustering to remove the undesirable properties. This section presents the minimal clustering modification problem where we are given an initial clustering produced from *any* algorithm. This work has been developed by Chia-Tung Kuo (PhD student, University of California, Davis), in a joint work with Ian Davidson (University of California, Davis), S. S. Ravi (University at Albany), Christel Vrain and myself. The result was presented in [Kuo *et al.* 2017].

5.3.1 Problem Formulation

Let us take an example of situation where you wish to cluster your ego-network (those people you have a direct friendship link to) into k groups and invite each group to a separate dinner party. For each person you know their interests, location, gender and age. After applying your favorite clustering algorithm you have k very cohesive clusters

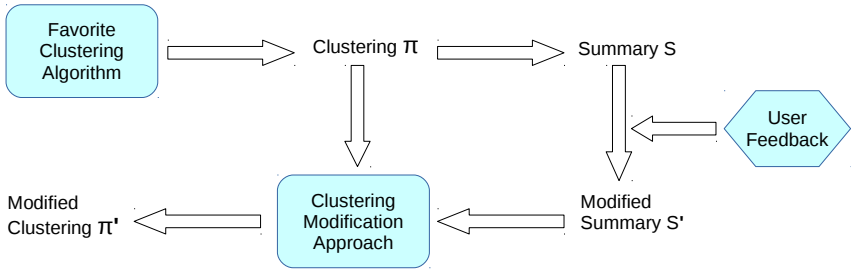


Figure 5.7: A schematic workflow for our clustering modification setting. The initial clustering Π can be produced in any manner.

except perhaps the range of ages for some cluster is too large or one cluster has too many females compared to males. Heuristically trying to move points from one cluster to another until a satisfactory result is found is not a viable approach as we show the intractability of re-clustering data to reduce cluster diameter (see Theorems 5.1 and 5.2). Simply removing data points to get desirable clusters undermines the intended use of clustering in the first place. For example we cannot just leave out some friends from the dinner parties. A more principled approach is to add constraints and apply a constrained clustering algorithm with the hope that the resultant clustering is similar to the previous yet free of the undesirable properties. However, with this approach there is no guarantee that the resultant clustering found will be similar to the original.

An alternative we explore is to start with the initial clustering and minimally modify it while removing the undesirable properties. Our proposed approach is based on the assumption that the given initial clustering is already of fairly good quality and suitability but the user is interested in finding a variation that is similar yet satisfies some additional properties. Though in this work the initial clustering is given by an algorithm, in practice it may be generated from an existing solution to a clustering problem or a set partition induced in any manner.

We can view this proposed work as being related to but quite different to constrained clustering. Constrained clustering allows domain experts to inject human guidance into clustering *a priori* before the clustering algorithm begins. This work instead allows providing guidance *a posteriori* after the clustering is found. This has the advantage of allowing feedback to be injected for *any* clustering algorithm. In short, we provide a principled way to generate a new minimally modified clustering while retaining most of the original solution. Alternative Clustering [Qi & Davidson 2009, Dang & Bailey 2010] addresses the problem of finding a clustering that is *different* from the given one yet whose quality is comparable to the original one. Our setting is different in that the user finds the given clustering acceptable yet only wants it to be minimally modified to satisfy some additional criteria as opposed to targeting a completely distinctive one.

Our approach is as follows and shown in Figure 5.7. We start with an initial clustering, Π, the output of any clustering algorithm or any other process that produces a set partition; this initial clustering has a corresponding clustering summary of properties S. Note this summary need not be with respect to the features used in the clustering algorithm. For example, Facebook data can be clustered based only on the friend-network topology and can be summarized based on user profile information. The user then modifies S,

leading to a desired modified summary S'. Our approach then looks for a Π' that satisfies the modified summary S' but is also a minimal modification of Π. The general problem is:

Problem 1 ***Minimal Clustering Modification (MCM).***
Input: Initial clustering Π *of* k *blocks and its* k*-part summary* S*. The user modifies parts of* S *to obtain a user-desired summary* S'*.*
Output: A modified clustering Π' *of* k *blocks similar to* Π *but also satisfying summary* S'*, formally:*

$$\begin{array}{ll} \underset{\Pi'}{\textit{minimize}} & d(\Pi, \Pi') \\ \textit{subject to} & \Pi' \textit{ satisfies } S' \end{array} \tag{5.6}$$

where $d(\Pi, \Pi')$ *is a distance measure between 2 clusterings.*

User feedback can be intra-cluster or inter-cluster as follows.

- Intra-Cluster Level Summaries for Feedback:
  - Shrink diameter: e.g. reduce the diameter of cluster i with respect to feature `age` to 10 years.
  - Balance categorical feature values: e.g. make the number of females and males equal in cluster i.
  - Upper/lower bounds on feature values per cluster: e.g. every cluster should contain at least 10% female and no more than 90% males.
- Inter-Cluster Level Summaries for Feedback:
  - Widen/shrink the distance between two clusters based on a feature: e.g. the distance between any two instances in cluster i and cluster j should be at least 5 years with respect to age.
  - Keep a cluster, merge two clusters, split a cluster, etc.

We have established intractability results on the general re-clustering problem under one particular modification: namely, cluster diameters. Such results support our choice of using CP as one cannot expect to find a solution efficiently with any particular algorithm.

A general statement of the **diameter-based reclustering problem** is as follows.

Given: A set $P = \{p_1, p_2, \ldots, p_n\}$ of n objects, an integer k $(1 \leq k \leq n)$, a k-clustering $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ of P, a list of $\ell \geq 1$ attributes, with respect to which the diameter of each cluster is computed, and numbers δ_i, $1 \leq i \leq \ell$.

Requirement: Is there another clustering $\mathcal{C}_1$ of P, also with k clusters, such that the maximum diameter of any cluster in $\mathcal{C}_1$ along attribute i is at most δ_i, $1 \leq i \leq \ell$? If so, find one such clustering.

We present complexity results for two versions of this reclustering problem. The first version shows the problem is NP-complete even if the diameter needs to be reduced along just two dimensions. The second version shows when the number of dimensions along which the diameter needs to be reduced is not a constant, the problem is NP-complete even for just three clusters.

Diameter reduction along two dimensions. Suppose the goal of reclustering is to reduce the diameters along $\ell = 2$ dimensions; that is, along the two chosen dimensions, the maximum diameter of any cluster must be at most δ_1 and δ_2 respectively. The following theorem shows that the reclustering problem is computationally intractable.

Theorem 5.1 *The reclustering problem where the maximum diameter must be reduced along two dimensions is NP-complete.*

Diameter reduction along many dimensions. When the number of dimensions ℓ along which the diameter must be reduced is large, we can show that the reclustering problem is NP-complete even when the number of clusters is just three. This result is shown below.

Theorem 5.2 *Suppose the number of dimensions along which the maximum diameter must be reduced is ℓ. Let δ_i denote the bound on the diameter along dimension i, $1 \leq i \leq \ell$. The reclustering problem is NP-complete for any $k \geq 3$.*

5.3.2 A CP Model for Minimal Clustering Modification

5.3.2.1 Variables and Constants

A clustering of k clusters on n data points is represented by a list of n cluster indices in $\{1, \ldots, k\}$, one for each data point. The difference between two clusterings Π and Π', $d(\Pi, \Pi')$, is measured by the number of positions in which the two lists differ; formally $d(\Pi, \Pi') = \sum_{i=1}^{n} \mathbb{I}[\Pi[i] \neq \Pi'[i]]$ where $\Pi[i]$ is the i-th entry in Π and $\mathbb{I}[\cdot]$ is the indicator function. Such a choice eliminates the ambiguity of permutations of the cluster indices. It is important to note that since the given clustering Π is desirable, we wish to minimally change its composition. Having an objective focused on some measure of *clustering quality* difference can result in a fundamentally different clustering.

Let Π be the given clustering. The *feature-wise diameters* for the clusters are represented as a $k \times f$ matrix $\mathcal{D}$ where $\mathcal{D}[i, j]$ records the diameter for the i-th cluster with respect to the j-th feature. Analogously $\mathcal{D}'$ is defined as the diameters the user desires on the modified clustering Π'. For convenience *all pairwise distances* are pre-computed with respect to each feature and are denoted by a 3-dimensional array D where $D[t, i, j]$ is the distance between the i-th instance and the j-th instance with respect to the t-th feature. Finally let X denote the $n \times f$ data feature matrix.

The model aims at finding a modified clustering Π' that is minimally different from Π and that satisfies the requirements of the user. The modified clustering Π' is represented by n variables of domain $\{1, \ldots, k\}$.

5.3.2.2 Encoding Objective and Summary

The model was designed and developed using CP platform Numberjack [Hebrard *et al.* 2010] due to its simple interface and its use of state-of-the-art integer linear programming (ILP) solvers. ILP solvers such as Gurobi[4], which is used in our experiments, can easily exploit multi-core architectures.

[4] `http://www.gurobi.com`

Objective. The number of instances moved from the initial clustering can be encoded straightforwardly with auxiliary variables, z, recording where Π and Π' disagree.

$$\forall i \in \{1, \ldots, n\}, \qquad z[i] = \mathbb{I}[\Pi'[i] \neq \Pi[i]]$$

The objective is therefore:

$$\texttt{minimize} \ \sum_{i=1}^{n} z[i]$$

Diameter Summary. In order to succinctly measure the modified clustering's (Π') diameters, we define a *cluster membership* matrix as a $k \times n$ binary matrix C, where each row indicates the membership of the corresponding cluster. This is enforced by the following constraints.

$$\forall c \in \{1, \ldots, k\}, \ \forall i \in \{1, \ldots, n\}, \qquad C[c, i] = \mathbb{I}[\Pi'[i] = c]$$

Now we describe how we encode constraints to enforce the desired diameters. A straightforward encoding follows the definition of diameter: we require each pair of instances in the same cluster to have feature-wise distance smaller than or equal to the specified diameter, shown as follows.

$$\forall c \in \{1, \ldots, k\}, \ \forall t \in \{1, \ldots, f\}, \qquad \max_{i,j=1,\ldots,n} \{C[c, i]C[c, j]D[t, i, j]\} \leq \mathcal{D}'[c, t]$$

Note $C[c, i] = 1$ if the i-th instance is in cluster c. Thus $C[c, i]C[c, j] = 1$ if and only if the i-th instance and the j-th instance are both in cluster c. One significant drawback of this encoding, however, is that `max` is taken over n^2 variables. This makes the encoding and solving very inefficient (both memory and CPU) when n is large.

A more efficient encoding was developed where each constraint involves at most n variables. The crucial observation is that these diameters are defined feature-wise, as opposed to the classical notion where a single diameter encompasses all dimensions. Accordingly, instead of requiring each pair in the same cluster to obey this cluster's diameter, we require just the difference between the maximum and the minimum value of the feature in a cluster to obey such diameter. Specifically we pre-compute the feature-wise minimums $M_l[t]$ and maximums $M_u[t]$ of the data for each feature t as follows.

$$\forall t \in \{1, \ldots, f\}, \qquad M_l[t] \leftarrow \min_{i=1,\ldots,n} \{X[i, t]\}$$
$$\forall t \in \{1, \ldots, f\}, \qquad M_u[t] \leftarrow \max_{i=1,\ldots,n} \{X[i, t]\}$$

For each cluster c and for each feature t, we define two variables $L[c, t]$ and $H[c, t]$ that correspond to the lowest value and the highest value of feature t in cluster c respectively. This relation is defined by the following constraints:

$$L[c, t] = \min_{i=1,\ldots,n} \{C[c, i](X_u[i, t] - M_u[t])\} + M_u[t]$$
$$H[c, t] = \max_{i=1,\ldots,n} \{C[c, i](X_u[i, t] - M_l[t])\} + M_l[t]$$

The model for the case where the user provides a set of desired feature-wise diameters $\mathcal{D}'$ as feedback is therefore:

$$
\begin{aligned}
&\underset{z,C,L,H}{\texttt{minimize}} && \sum_{i=1}^{n} z[i] \\
&\texttt{subject to} && \\
& && \forall c \in \{1,\ldots,k\},\ \forall i \in \{1,\ldots,n\}, \quad C[c,i] = \mathbb{I}[\Pi'[i] = c] \\
& && \forall i \in \{1,\ldots,n\}, \quad z[i] = \mathbb{I}[\Pi'[i] \neq \Pi[i]] \\
& && \forall c \in \{1,\ldots,k\},\ \ \forall t \in \{1,\ldots,f\}, \\
& && \quad L[c,t] = \underset{i=1,\ldots,n}{\texttt{min}} \{C[c,i](X[i,t] - M_u[t])\} + M_u[t] \\
& && \quad H[c,t] = \underset{i=1,\ldots,n}{\texttt{max}} \{C[c,i](X[i,t] - M_l[t])\} + M_l[t] \\
& && \quad H[c,t] - L[c,t] \leq \mathcal{D}'[c,t]
\end{aligned}
\tag{5.7}
$$

In this model, the numbers of variables and constraints are linear in the number of instances n, the number of clusters k and the number of features f. In addition the variables also have rather small domains. Figure 5.8 provides a tabulation of the numbers of variables, their domain sizes and the associated constraints used to encode the model (5.7). Note the domain size r for variables L and H arises from the discretization of continuous values and the choice of r typically involves a tradeoff between precision and model complexity as is the case in all other discretization problems.

Vars.	Number	Domain size
Π'	n	k
z	n	2
C	nk	2
L	nk	r
H	nk	r

(a) Numbers and domain sizes of variables used in our model.

Constraints	Number	#. Vars. involved
Bind z	n	2
Bind C	nk	2
Bind L	kf	$n+1$
Bind H	kf	$n+1$
$H - L \leq \mathcal{D}'$	kf	2

(b) Number of constraints and the numbers of variables involved in each constraint.

Figure 5.8: Complexity of encoding (5.7) model.

Encoding other feedback/summary. As mentioned earlier CP is flexible in encoding other types of feedback as constraints. A list of common constraints conforming to feedback introduced earlier and their encodings are presented in Table 5.7. It is also worth mentioning that these constraints need not apply to all features or clusters. The ranges of the indices for the constraints in Table 5.7 (i.e. c, t, etc) could be determined at the user's discretion.

Constraints	Encoding
Diameters	$\forall c = 1, \ldots, k, \quad \forall t = 1, \ldots, f, \quad H[c,t] - L[c,t] \leq \mathcal{D}'[c,t]$
Splits	$\forall c_1, c_2 = 1, \ldots, k \text{ where } c_1 \neq c_2, \forall t = 1, \ldots, f,$ $\min_{i,j=1,\ldots,n} \{C[c_1,i]C[c_2,j]D[t,i,j]\} \geq \mathcal{S}'[c_1,c_2,t]$
Bound cluster size	$l_c \leq \sum_{i=1}^{n} C[c,i] \leq u_c$
Keep a cluster	$\sum_i C[c,i] \geq 1$ for cluster c to be kept
Remove a cluster	$\sum_i C[c,i] = 0$ for cluster c to be merged (i.e. effectively empty a cluster)
Upper bound moves	$\sum_{i=1}^{n} z[i] \leq u_z$
Balance binary features	$\forall c = 1, \ldots, k, \ \forall t = 1, \ldots, f,$ $p_l \sum_{i=1}^{n} C[c,i] \leq \sum_{i=1}^{n} C[c,i]X[i,t] \leq p_u \sum_{i=1}^{n} C[c,i]$

Table 5.7: Common feedbacks and their encodings. Note we assume H and L are properly encoded auxiliary variables as in (5.7). $\mathcal{S}'$ is a user-desired $k \times k \times f$ matrix of the desired splits between clusters; p_l and p_u are the user-desired lower and upper bounds on the counts of 1's (True) in binary features for each cluster.

5.3.3 Empirical Evaluation

The model was developed using CP platform Numberjack and the chosen backend solver was Gurobi. Experiments were developed on two real world data sets (social network and medical imaging) to explore the benefits of using modification and also on UCI data sets to explore scalability issues [Kuo *et al.* 2017]. We present below some experiments on real world data sets.

Social Network Modification. We apply our proposed approach to a network data set: *Facebook-egonets* from Stanford SNAP Data sets [Leskovec & Krevl 2014]. This data set consists of 4039 Facebook users where the friendships among them are known and for each person a list of binarized categorical features such as gender[5]. We run (normalized) spectral clustering algorithm [Luxburg 2007] on this graph to find an initial 4-way clustering; a hard clustering is then obtained from the best of 10 runs of k-means on the spectral embedding. Note that spectral clustering *only utilizes the friendship graph topology, but not the node features.* The clustering found is of very low cut cost but a summary of the initial clustering shown in Figure 5.9(a) shows a widely differing composition compared to the population averages.

Our aim now is to minimally modify the original clustering to correct for gender and language imbalance by constraining them to be close to the population averages. We choose the upper and lower bounds according to the averages in the initial summary and set bounds $[0.36, 0.4]$ for gender and $[0.13, 0.15]$ for language so that these two features are "balanced" across clusters. We find a minimum of 69 nodes need to be moved between clusters and the summary for the resulting modified clustering is presented in Figure 5.9(b).

An important comparison is against another clustering satisfying the same summary of "balanced" features but without enforcing the objective of "minimal modification". This simulates re-running the clustering algorithm from the beginning and enforcing the balancing constraints. One often found clustering simply puts most instances in one cluster,

[5]The feature values are anonymized so, for example, it is un- known if 1 is male or female.

resulting in 4015 instances in cluster 1 and 8 instances in each of clusters 2, 3 and 4, leading to a total of 1074 swaps across clusters from the initial clustering. We also report the normalized cut costs (the objective of normalized spectral clustering) on the three clusterings: *initial*, *satisfying summary+minimally modified*, *only satisfying summary*. Their cut costs are, respectively, 0.97, 1.34 and 3.04. Note a constraint on the cut cost could be additionally included if it was desired to keep it below a bound.

	Initial clustering				
	C1	C2	C3	C4	Population
Gender	1096 (0.37)	37 (0.54)	169 (0.49)	230 (0.36)	1532 (0.38)
Language	402 (0.13)	5 (0.07)	64 (0.19)	78 (0.12)	549 (0.14)
Size	2988	69	345	637	4039

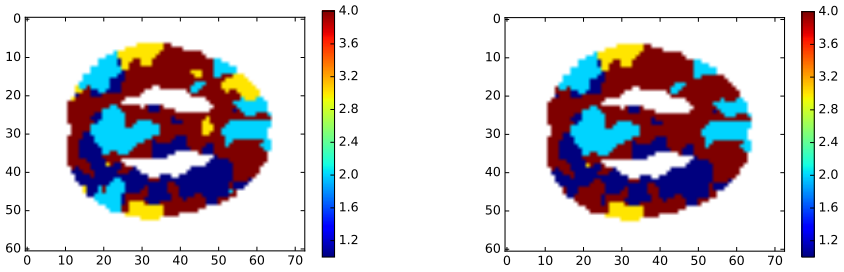
(a) Initial clustering summary

	Modified clustering				
	C1	C2	C3	C4	Population
Gender	1124 (0.37)	22 (0.39)	117 (0.40)	269 (0.40)	1532 (0.38)
Language	408 (0.14)	7 (0.13)	43 (0.15)	91 (0.13)	549 (0.14)
Size	3014	56	293	676	4039

(b) Modified clustering summary

Figure 5.9: Summaries for the initial and modified clusterings. "Gender" and "Language" record the numbers of instances that have this feature being 1; the numbers in the brackets give the ratios. Size is the number of instances in the cluster.

Spatial Region Modification In this experiment, we apply our approach to a fMRI brain imaging data which allows exploring modification based on spatial information. The fMRI scans used here were collected and pre-processed at UC Davis Alzheimer's Disease Center and they were recorded while the subjects were at resting state. We work on one particular slice in the mid-brain so that each scan consists of 2D snapshots over time and each slice has a total of 1730 voxels/nodes whose blood oxidation levels are measured at an interval of 3ms over 200+ time steps.



(a) Initial (b) Modified

Figure 5.10: Initial and modified clusterings on the fMRI scan. The modification asks for the yellow cluster's x-diameter and the cyan cluster's y-diameter to be shrunk. The color-coded cluster numbers match the numbering in Figure 5.11.

We start off by constructing a 1730 node completely connected graph where the edge

weights are the absolute value of Pearson correlations between the voxels measured over time. Such correlation measure has been widely used in the neuroscience community [Friston 2011]. As before we create an initial clustering by running normalized spectral clustering on this graph and then selecting the best result from 10 runs of k-means on the spectral embedding. The initial clustering is shown in Figure 5.10(a).

Often in practice we like clusters to represent compact regions in the brain; however this initial clustering is generated based on correlations and does not take into account *any spatial coordinates* in the brain. Accordingly we look for a modified clustering with tighter diameters in x-y spatial coordinates, specifically, x-diameter ≤ 15 for cluster 3 and y-diameter ≤ 30 for cluster 2 (yellow and cyan in Figure 5.10) while keeping the diameters for the other clusters. Our CP model returns a new clustering that moves a total of 109 voxels where 36 voxels were moved from cluster 2 to cluster 4 and 63 voxels were moved from cluster 3 to cluster 4. We present the summaries of the initial and the modified clusterings in Figure 5.11. Note this is a *global* optimal solution.

Cluster index	C1 (Blue)	C2 (Cyan)	C3 (Yellow)	C4 (Red)
x-diameter	46	49	48	52
y-diameter	37	41	45	44
size	390	369	152	819

(a) Initial summary

Cluster index	C1 (Blue)	C2 (Cyan)	C3 (Yellow)	C4 (Red)
x-diameter	46	49	**15**	52
y-diameter	37	**26**	45	44
size	390	323	89	928

(b) Modified summary

Figure 5.11: Summaries for the initial and modified clusterings for the fMRI scan experiment. Size is the number of voxels in the cluster.

5.4 Repetitive Branch-and-Bound using CP for WCSS

One of the most used criterion is minimizing the Within-Cluster Sum of Squares (WCSS), which is defined by the sum of the squared Euclidean distances from each object to the centroid of the cluster to which it belongs. Minimum sum-of-squares clustering (MSSC) has been proven to be NP-Hard [Aloise *et al.* 2009] and has been studied in numerous work. The well-known k-means algorithm as well as other dedicated heuristic algorithms find a local optimal for this criteria [Steinley 2006]. They have been also extended to integrate must-link and cannot-link constraints: COP-kmeans [Wagstaff *et al.* 2001], PCK-Means [Basu *et al.* 2004a], CVQE [Davidson & Ravi 2005], LCVQE [Pelleg & Baras 2007], etc. However they can either fail to find a solution that satisfies all the constraints even when such a solution exists or do not guarantee the satisfaction of all the constraints. On the other hand, general and declarative frameworks using generic optimization tools have been proposed, based either on Integer Linear Programming with column generation [Babaki *et al.* 2014] or our work using constraint programming (cf. Section 4.3.3).

For unconstrained MSSC, Brusco [Brusco 2006] proposed a simple yet effective method: the Repetitive Branch-and-Bound Algorithm (RBBA). In what follows, we show how the

idea of clustering with RBBA can be extended to user-constraints and can be combined with the ideas of clustering with constraint programming. We show that the methodology can be combined with a CP framework to obtain an efficient method that can easily incorporate user constraints to obtain a constraint-based method that is generic yet more efficient then other exact constrained methods. This work was published in [Dao *et al.* 2016a, Guns *et al.* 2016] and was developed in collaboration with Tias Guns, Khanh-Chuong Duong and Christel Vrain.

5.4.1 Repetitive Branch-and-Bound Algorithm

Let $\mathcal{O}$ be a set of N points. Let Δ be a partition of $\mathcal{O}$ into at most K clusters. For any subset S of $\mathcal{O}$, let Δ_S denote the projection of Δ onto the objects in S and $WCSS(\Delta_S)$ the WCSS value of Δ_S. Let $WCSS^*(S) = \min_\Delta(WCSS(\Delta_S))$. Let us note that in Δ_S some clusters of Δ may become empty.

Lower Bound Inequalities Without User-Constraints. The bounds used in RBBA rely on the following result [Koontz *et al.* 1975]. Let S be a subset of $\mathcal{O}$, and let S_1 and S_2 be such that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$ (non-overlapping). We have:

$$WCSS(\Delta_S) \geq WCSS(\Delta_{S_1}) + WCSS(\Delta_{S_2}) \tag{5.8}$$

Since $WCSS^*(S_2) = \min_\Delta(WCSS(\Delta_{S_2}))$, so $WCSS^*(S_2)$ is the smallest WCSS value for all partitions of S_2 into at most K clusters. Hence we have:

$$WCSS(\Delta_{S_2}) \geq WCSS^*(S_2) \tag{5.9}$$

and hence [Brusco 2006]:

$$WCSS(\Delta_S) \geq WCSS(\Delta_{S_1}) + WCSS^*(S_2) \tag{5.10}$$

Equation (5.10) can be used during a branch-and-bound search for an optimal partition of S as follows. Let us suppose that we have previously built a partition of S, thus giving an upper bound for $WCSS^*(S)$, that we have currently built a partial solution Δ_{S_1} and that we know an optimal solution of $WCSS^*(S_2)$. If $WCSS(\Delta_{S_1}) + WCSS^*(S_2)$ is greater than the actual upper bound, then the partial solution Δ_{S_1} can never lead to a better solution than the current upper bound. This process is used in the Repetitive Branch-and-Bound Algorithm presented below, where the value $WCSS^*(S_2)$ is computed for each increasing set S_2 using branch-and-bound.

Repetitive Branch-and-Bound Algorithm. The Repetitive Branch-and-Bound Algorithm (RBBA) [Brusco 2006] is presented in Algorithm 8.

Points in $\mathcal{O}$ are first ordered following an heuristic by *OrderPoints($\mathcal{O}$)*. Different heuristics can be used for ordering points.We assume that according to the ordering, points are named by their index $i \in [1, N]$. $\mathcal{O}_n$ is composed of the *last* n points according to this order.

In this algorithm, Δ_n indicates any partition of $\mathcal{O}_n$ into at most K clusters and Δ_n^* denotes the optimal partition of $\mathcal{O}_n$ into at most K clusters. This algorithm starts with the set $\mathcal{O}_K$ of the last K points and *Init($\mathcal{O}_K$)* creates Δ_K^* by putting each point alone in a

Algorithm 8: RBBA *input: objects $\mathcal{O}$, number clusters K*

```
1  OrderPoints(O)
2  O_K ← {o_{N−K+1}, ..., o_N}
3  Δ*_K ← Init(O_K)
4  W_n ← 0, ∀n ∈ {1, ..., K}
5  for n = K + 1 to N do
6      O_n ← O_{n−1} ∪ {o_{N−n+1}}
7      Δ_n ← Greedy_Extension(O_n, Δ*_{n−1})
8      U_n ← WCSS(Δ_n)
9      Δ*_n ← BaB_Search(O_n, U_n, W)
10     W_n ← WCSS(Δ*_n)
```

cluster. The optimal value $WCSS(\Delta_n^*)$ is stored in W_n for each n, and the first K values $W_1, \ldots, W_K$ are 0 (each point in its own cluster).

The algorithm next iterates by adding to the set $\mathcal{O}_n$ one point each time, from the point $N-K$ down to the first point. Here $\mathcal{O}_n$ represents this set of last n points $o_{N-n+1}, \ldots, o_N$. Function *Greedy_Extension($\mathcal{O}_n, \Delta_{n-1}^*$)* greedily finds a partition Δ_n for $\mathcal{O}_n$, by adding the new point to the previous best partition Δ_{n-1}^* so that the value WCSS is minimally increased. The value $WCSS(\Delta_n)$ constitutes an upper bound U_n for $WCSS(\Delta_n^*)$. *BaB_Search($\mathcal{O}_n, U_n, W$)* is a branch-and-bound algorithm which searches for a global optimal partition Δ_n^* on the set of points $\mathcal{O}_n$, using U_n as an upper bound and exploiting Equation (5.10) with the W_i values ($i < n$) as lower bounds. Let $o_m = o_{N-n+1}$ be the new point added at this step. The branch-and-bound search considers the points in $\mathcal{O}_n$ in the order $o_m, o_{m+1}, \ldots, o_N$ and tries to assign them to clusters.

Let us consider an arbitrary step when a point number p ($m \leq p < N$) is assigned to a cluster. Let S_1 be the set of points $\{o_m, ..., o_p\}$ and S_2 be $\{o_{p+1}, ..., o_N\}$. All the points in S_1 have already been assigned and hence $WCSS(\Delta_{S_1})$ is known. All the points in S_2 are currently unassigned, however, $WCSS^*(S_2)$ has been computed in a previous step of RBBA and stored in $W_{|S_2|}$; U_n is the current upper bound. Equation (5.10) is used and if $WCSS(\Delta_{S_1}) + WCSS^*(S_2) \geq U_n$, we cannot extend Δ_{S_1} to a solution having WCSS better than U_n. Therefore *BaB_Search* will not continue to extend Δ_{S_1} and the branch is pruned. When $p = N$, the partition Δ is complete and U_n is set to $WCSS(\Delta)$. When the entire search space is explored, the last complete partition found is the optimal solution.

This algorithm takes advantage of the optimal solutions previously computed to provide lower bounds in the branch-and-bound search. Also important are the upper bounds found by the greedy extension, they are often tight (meaning that the greedy extension is the optimal partitioning). Because of these tight bounds, even though the algorithm runs the branch-and-bound search N times, it is nevertheless one of the best exact algorithms for minimum sum-of-squares clustering. A similar search method was proposed for valued (soft) CSPs with an additive objective function, called Russian Doll Search [Verfaillie *et al.* 1996].

5.4.2 Extension of RBBA to User-Constraints

Lower Bound Inequalities With User-Constraints. We have studied the conditions under which Equation (5.10) is still valid in the presence of a set of user constraints $\mathcal{C}$ on $\mathcal{O}$. Given a set of points $S \subseteq \mathcal{O}$ and a set of constraints C on S, $\mathcal{S}(S, C)$ denotes the set of all partitions Δ_S of S satisfying C. We denote by $WCSS^*(S, \mathcal{C})$ the optimal WCSS of $S \subseteq \mathcal{O}$ under constraint set $\mathcal{C}$, that is, $WCSS^*(S, \mathcal{C}) = \min(\{WCSS(\Delta_S) | \Delta_S \in \mathcal{S}(S, \mathcal{C})\})$. We denote by $WCSS(\Delta_S, \mathcal{C})$ the WCSS value of a partition Δ_S under the condition that it satisfies the constraint set $\mathcal{C}$.

One can see from this that Equation (5.9) still holds when considering a set of constraints $\mathcal{C}$: $WCSS(\Delta_S, \mathcal{C}) \geq WCSS^*(S, \mathcal{C})$. Indeed, any $\Delta_S \in \mathcal{S}(S, \mathcal{C})$ will have a score equal or worse than the optimal one satisfying $\mathcal{C}$.

The main question is then under what conditions Equation (5.8), and hence (5.10), holds in the presence of constraints. Given a set of constraints $\mathcal{C}$ on S which Δ_S satisfies, the set of constraints $\mathcal{C}_{S_i}$ that can be put on S_1 and S_2 such that Equation (5.10) is still valid must be carefully defined. For instance, a cannot-link constraint on one point in S_1 and one point in S_2 is undefined on both S_1 and on S_2, or a minimal size constraint satisfied on Δ_S can be no longer satisfied neither on S_1, nor on S_2.

In general, given a set of C of constraints put on objects of S, we can restrict the set C_{S_i} with $S_i \subseteq S$ to those constraints for which all objects in the constraint are in the set S_i. If a partition Δ_S satisfies a set of constraints C, then its projection onto S_i (Δ_{S_i}) will satisfy the subset of constraints C_{S_i}. Therefore:

$$WCSS(\Delta, C) \geq WCSS(\Delta_{S_1}, C_{S_1}) + WCSS^*(S_2, C_{S_2}) \tag{5.11}$$

Many *cluster-level* constraints involve all variables and hence with this approach cannot be considered until the very end. However, for two constraint sets C_1 and C_2 such that $C_1 \subseteq C_2$, then $\mathcal{S}(S, C_2) \subseteq \mathcal{S}(S, C_1)$ and therefore $WCSS^*(S, C_1) \leq WCSS^*(S, C_2)$. Hence, including more constraints can lead to tighter lower bounds.

In order to incorporate some cluster-level constraints, we distinguish those that are anti-monotonic from those that are not. A constraint c is said to be anti-monotonic if when satisfied by a partition Δ_S, it is satisfied by all the projections Δ_{S_i}, with $S_i \subseteq S$. As an example, a maximal size constraint is anti-monotonic whereas a minimal size constraint is not.

Let C_a be the anti-monotonic constraints in C. Then, since Δ_{S_2} satisfies the constraints on C_{S_2} and the anti-monotonic constraints of C, and similarly for S_1, we have:

$$WCSS(\Delta, C) \geq WCSS(\Delta_{S_1}, C_{S_1} \cup C_a) + WCSS^*(S_2, C_{S_2} \cup C_a) \tag{5.12}$$

RBBA with User Constraints. Let $\mathcal{C}$ be the set of all constraints on $\mathcal{O}$. We assume that the set $\mathcal{C}$ is satisfiable on $\mathcal{O}$, ie. there exists a partition Δ of $\mathcal{O}$ that satisfies $\mathcal{C}$. The extension of RBBA to incorporate user constraints is presented in Algorithm 9.

After ordering points, Algorithm 9 constructs an initial partition Δ_K of *at most* K clusters taking constraints $\mathcal{C}_K = \mathcal{C}_{\mathcal{O}_k}$ into account. It does so by putting each point that can be in its own cluster in a separate cluster (if there is a must-link, the two points must be put in the same cluster). Among all such partitions, the one with smallest $WCSS(\Delta_K)$ is chosen. Since $\mathcal{C}$ is satisfiable on $\mathcal{O}$, the partition Δ_K^* must exist.

At each step n, for the set $\mathcal{O}_n$ of the last n points, Algorithm 9 searches in the solution space $\mathcal{S}(\mathcal{O}_n, \mathcal{C}_n)$. There are different options for the constraint set $\mathcal{C}_n$. As discussed in the

previous section, $\mathcal{C}_n$ can be $\mathcal{C}_{\mathcal{O}_n}$ or $\mathcal{C}_{\mathcal{O}_n} \cup \mathcal{C}_a$. We note that the more constraints that are considered at one step, the tighter the lower bound for the next step would be. At the last step, when $\mathcal{O}_N = \mathcal{O}$, the full set of user constraints $\mathcal{C}$, anti-monotonic or not, will be considered.

Feasible_Extension tries to extend the best partition of the previous step Δ^*_{n-1} to a partition Δ_n of $\mathcal{O}_n$ that satisfies $\mathcal{C}_n$. *Constrained_BaB($\mathcal{O}_n, \mathcal{C}_n, U_n, W$)* performs a branch-and-bound search to find an optimal partition among all the partitions that satisfy the set of constraints $\mathcal{C}_n$. It uses U_n as the initial upper bound and W for the lower bounds, in the same way as *BAB_Search* in Algorithm 8.

Algorithm 9: Extended RBBA
input: objects $\mathcal{O}$, number clusters K, constraint set $\mathcal{C}$

```
1  OrderPoints(O)
2  O_K ← {o_{N−K+1}, ..., o_N}
3  Δ*_K ← Init(O_K, C_K)
4  W_K ← WCSS(Δ*_K)
5  for n = K + 1 to N do
6      O_n ← O_{n−1} ∪ {o_{N−n+1}}
7      Δ_n ← Feasible_Extension(O_n, C_n, Δ*_{n−1})
8      if Δ_n exists then
9          U_n ← WCSS(Δ_n)
10     else
11         U_n ← ∞
12     Δ*_n ← Constrained_BaB(O_n, C_n, U_n, W)
13     W_n ← WCSS(Δ*_n)
```

Ordering of Points. Algorithms 8 and 9 start by ordering points and they do branch-and-bound for an increasing set of points following this order. Different orders can be used. In RBBA [Brusco 2006], the nearest-neighbor separation heuristic is used: at each step of the ordering, the two points that have the smallest distance among all pairs of points are withdrawn from the set of points and are placed at opposite ends in the ordering. Another ordering we use is based on the furthest-point-first (FPF) algorithm [Gonzalez 1985], which is used in our models (cf. Section 4.2).

5.4.3 A Framework Using CP

Our CP model that defines a partition and user-constraints, such as in Subsection 4.2.2, is used to achieved the constrained branch-and-bound search at each step. To make further use of the computed bounds in the previous steps, we have developed a global constraint that expresses the relation $V = sumSquares(G, d, W)$, where G is the array of variables defining a partition, d is the distance between each pair of points and W contains the previous WCSS* values (as per Algorithm 9).

A Novel Sum-of-Squares Constraint. The filtering algorithm for the sum-of-squares constraint $V = sumSquares(G, d, W)$ is given in Algorithm 10. Because of the variable order, at any time the propagator is called, there is an index p ($1 \leq p < n$) such that $G_1, .., G_p$ are instantiated and $G_{p+1}, ..., G_n$ are not.

Algorithm 10 enforces bound consistency for V by first computing a lower bound for V, using the lower bound given by Equation (5.12). This algorithm exploits also W to do a look ahead to filter the domain of G_{p+1}. For each $k \in Dom(G_{p+1})$, that is, all clusters k not forbidden for this point because of another constraint, if point $p+1$ is assigned to the cluster k, the lower bound for V is revised, using the same Equation (5.12) with the minimal WCSS value for the last $n-p-1$ points.

Algorithm 10: Filtering of: "$V = sumSquares(G, d, W)$"

input: V, G, d, W with $G_1, ..., G_p$ assigned, G_{p+1} unassigned
1 `// computation of lower bound for` V
2 **for** $k = 1$ **to** K **do**
3 $\quad sum[k] \leftarrow 0$; $size[k] \leftarrow 0$; $s[k] \leftarrow 0$
4 **for** $i = 1$ **to** p **do**
5 $\quad k \leftarrow G_i.val()$
6 $\quad size[k] \leftarrow size[k] + 1$
7 $\quad$ **for** $j = i + 1$ **to** p **do**
8 $\quad\quad$ **if** $G_j.val() == k$ **then**
9 $\quad\quad\quad sum[k] \leftarrow sum[k] + d(i, j)^2$
10 $V_1 \leftarrow 0$
11 **for** $k = 1$ **to** K **do**
12 $\quad V_1 \leftarrow V_1 + \ sum[k]/size[k]$
13 **if** $V_1 + W_{n-p} \geq V.ub$ **then**
14 $\quad$ return Failure
15 **else**
16 $\quad V.lb \leftarrow \max(V.lb, V_1 + W_{n-p})$
17 `// look ahead to filter` $Dom(G_{p+1})$
18 **for** $i = 1$ **to** p **do**
19 $\quad s[G_i.val()] \leftarrow s[G_i.val()] + d(i, p+1)^2$
20 **foreach** k *in* $Dom(G_{p+1})$ **do**
21 $\quad V_1' \leftarrow V_1 - sum[k]/size[k] + (sum[k] + s[k])/(size[k] + 1)$
22 $\quad$ **if** $V_1' + W_{n-p-1} \geq V.ub$ **then**
23 $\quad\quad$ remove k from $Dom(G_{p+1})$

The complexity of this algorithm is $O(p^2)$, due to the computation of *sum* and *size*. It can be reduced to $O(p)$ when the arrays *sum* and *size* are stored and computed incrementally over different propagation runs.

Improvements. Several improvements have been proposed, such as agglomerating all the points related by must-link constraints to the same super-points, or using the same model with a greedy strategy to find a good feasible clustering quickly [Guns *et al.* 2016]. We emphasize here the interest of constraint propagation in CP. A constraint solver can additionally reason over *partial* solutions, namely over the domain of a set of variables. A constraint solver is guaranteed not to reject a partial solution that can be extended to a full solution, while it can reject partial solutions that provably can not satisfy a constraint (such as an anti-monotonic constraint and more). Exploiting this property we have proposed two different ways to define of constraint set $\mathcal{C}_n$: local and full.

Let $\mathcal{C}$ be the set of all user constraints on the whole set of points $\{o_1, \ldots, o_N\}$. There may be instance-level constraints (must-link or cannot-link constraints) or cluster-level constraints (cardinality, density constraints etc.). At each step n, *Constrained_ BaB* finds a clustering that minimizes the WCSS value and that satisfies the set of constraints $\mathcal{C}_n$. Let $\mathcal{O}_n$ be the set of points to cluster at step n. The *local* model is defined on $\mathcal{O}_n$, with the set $\mathcal{C}_n = C_{\mathcal{O}_n}$, the set of user constraints on a (sub)set of the elements of $\mathcal{O}_n$. One can see that for $n = N$, $\mathcal{O}_N = \mathcal{O}$ and hence we will consider the set $C_{\mathcal{O}} = C$ of all constraints. The *full* model considers *all* constraints at every steps. At each iteration $n \leq N$, *Constrained_ BaB* operates on the full set of N variables and all the user constraints in $\mathcal{C}$ are considered in the model. However, since we are interested in finding a best clustering on the last n points of G only, the constraint *sumSquares* is defined only on the last n variables $G_{N-n+1}, \ldots, G_N$. The branching is also on these n variables only. The interest of such a *full* model is that it can allow to prune earlier cases that cannot be extended to a full solution.

Experiment results. CPRBBA has been compared to other state-of-the-art exact clustering approaches: original RBBA[6] [Brusco 2006], our CP model with one phase branch-and-bound search [Dao *et al.* 2015a] and CCCG-0.5.1[7] [Babaki *et al.* 2014] using Integer Linear Programming and column generation. Both unconstrained and constrained settings are considered. We report here some results from [Guns *et al.* 2016].

Table 5.8 reports results in cases without user-constraints. We can see that both RBBA and CPRBBA are better than the recent CPClustering and CCCG methods in case no constraints are added. Considering user-constraints, Table 5.9 reports results on 5 random samples of #c must-link and #c cannot-link constraints. We can observe here that CPRBBA outperforms existant methods in all the cases (CPClus is faster on average in one case – wine, 100 ML and 100 CL constraints – caused by CPRBBA not finding the optimal solution within the timeout on for one constraint set, two for -local). The observation is the same with a cluster minimal size constraint, as shown in Table 5.10.

Multi-Objective WCSS-Split Constrained Clustering. We have used this approach to experiment multi-objective constrained clustering, where the homogeneity is expressed by the WCSS and the separatedness is expressed by the split between clusters. Algorithm 5 is used to find the Pareto front for this bi-objective problem under a set of user-constraints $\mathcal{C}$. In this algorithm, constrained single objective optimization (WCSS) is iterated, each time with a condition on the best value of the other objective (minimal

[6] `http://www.psiheart.net/QuantPsych/monograph.html`

[7] `https://dtai.cs.kuleuven.be/CP4IM/cccg/`

	N	K	CCCG	CPClustering	RBBA	CPRBBA
ruspini	75	4	1800+	0.41	**0.01**	**0.01**
soybean	47	4	1800+	1.21	**0.38**	1.28
hatco	100	2	1800+	1.74	**0.03**	0.05
hatco	100	3	1800+	186.18	0.29	**0.20**
hatco	100	4	1800+	1800+	53.95	**7.52**
hatco	100	5	1800+	1800+	1800+	**1636.41**
iris	150	3	1800+	583.19	**1.14**	1.33
wine	178	3	1800+	1800+	**7.86**	53.57
seeds	210	3	1800+	1800+	542.74	**170.67**
breast	569	2	1800+	1800+	1800+	1800+

Table 5.8: Runtimes in seconds of different exact methods

	#c	cccg	cpclus	cprbba-local	cprbba-full
iris	10	1800+ (5)	468.48 (0)	**1.13 (0)**	1.22 (0)
iris	25	1800+ (5)	204.89 (0)	**1.41 (0)**	1.56 (0)
iris	50	1800+ (5)	205.98 (0)	**0.27 (0)**	0.34 (0)
iris	100	279.80 (0)	0.09 (0)	**0.01 (0)**	**0.01 (0)**
iris	150	0.20 (0)	0.02 (0)	0.02 (0)	**0.01 (0)**
wine	10	1800+ (5)	1800+ (5)	**1029.67 (2)**	1033.67 (2)
wine	25	1800+ (5)	1800+ (5)	**724.33 (2)**	724.68 (2)
wine	50	1800+ (5)	1800+ (5)	749.87 (2)	**749.46 (2)**
wine	100	1800+ (5)	**21.58 (0)**	1132.24 (2)	361.90 (1)
wine	150	172.80 (0)	0.08 (0)	3.83 (0)	**0.04 (0)**
wine	250	0.20 (0)	0.02 (0)	**0.01 (0)**	**0.01 (0)**

Table 5.9: Runtimes averaged over 5 random samples of #c must-link and #c cannot-link constraints; between brackets number of runs that timed-out (counted as 1800 seconds in average).

	K	min size	cpclus.	cprbba-local	cprbba-full
ruspini	4	17	1.08	**0.02**	1.17
ruspini	4	18	270.00	**9.00**	24.06
soybean	4	10	1.28	**1.39**	1.78
soybean	4	11	1800+	**1563.12**	1652.13
iris	3	38	564.86	**1.32**	1.67
iris	3	42	693.38	9.23	**2.45**
iris	3	46	933.23	341.23	**18.46**
iris	3	50	1508.77	1800+	**294.75**

Table 5.10: Runtime in seconds for clustering with minimum (top) and maximum (bottom) size constraint

Use case	time (s)	#sols	#c/s
unconstrained	1.11	10	1
20 ML/CL	13.68	7	1
40 ML/CL	9.66	8	1
size minimal 38	1.6	7	1
size minimal 40	1.8	4	1
20 ML/CL, size min 40	13.80	7	1
40 ML/CL, size min 40	9.75	8	1

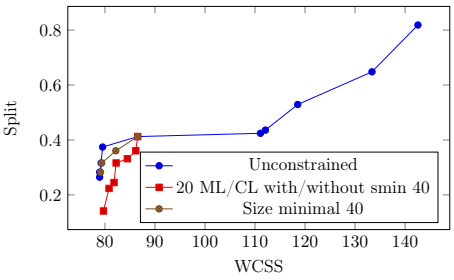


Figure 5.12: Results and Pareto fronts for bi-objective WCSS-Split on Iris dataset

split) found so far. This minimal-split constraint can in turn be translated into must-link constraints.

Figure 5.12 shows the results for different use cases on the Iris dataset and the exact Pareto fronts for four of these cases (the two cases for 20 ML/CL constraints with and without the minimal size constraint have the same Pareto front). We can see here the interest of being able to handle user-constraints during the optimization process. Indeed, in this dataset, each ground truth cluster is of size 50, whereas in the unconstrained use case, the Pareto solutions can give clusterings with unbalanced clusters. For instance, the last point in the Pareto front corresponds to a clustering with clusters of size 2, 50 and 98. The constrained cases have the last Pareto solution with WCSS=86.5396 and Split=0.412311. This solution is common to all the 4 cases, and the only corresponding clustering has clusters of size 49, 50, 51.

5.5 Constrained Clustering for Time-Series Data

Constrained clustering enables the exploitation of expert knowledge. Nevertheless, the application of constrained clustering on time-series analysis is relatively unknown. This is partly due to the unsuitability of the Euclidean distance metric, which is typically used in data mining, to time-series data. In January 2017, I have started a collaboration with Pierre Gançarski (ICube, University of Strasbourg) on constrained clustering for time-series images. This collaboration involves Thomas Lampert, Baptiste Lafabregue (ICube), Chirstel Vrain and Nicolas Serrette, Master student whose internship was in the context of the collaboration and was under our supervision. We explore the application on time-series data of different approaches on dissimilarity-based constrained clustering, including our constraint programming based approach. A survey on the approaches and on experimental study of the available approaches has been submitted to Journal of Data Mining and Knowledge Discovery [44].

Time-series data clustering. Data in many applications is being stored in the form of time-series data. Time-series data is a type of temporal data, where each time-series is consisting of a large number of data points. Most of times-series clustering are classified into two categories [Keogh & Lin 2005]:

- Whole time-series clustering: clustering of a set of individual time-series with respect to their similarity. Here objects are time-series.

- Subsequence clustering: given a single time-series, individual time-series are extracted via a sliding window. Clustering is then performed on the extracted subsequences.

[Keogh & Lin 2005] represented that subsequence clustering is "meaningless". A typical goal in time-series analysis is to cluster the data using the full time-series therefore we focus on whole time-series clustering.

Time-series clustering relies on distance measure to a high extent. While distance between static objects is exactly match based, for time-series objects, distance is calculated approximately. Different measures can be applied to compute distance among time-series, the most common methods in time-series clustering are Euclidean distance and DTW [Aghabozorgi *et al.* 2015]. Euclidean distance relies on a fixed mapping between points in two times-series and is proper to finding similar time-series in time. According to [Aghabozorgi *et al.* 2015], clustering of time-series that are correlated (e.g. to cluster time-series of share price related to many companies to find which shares changes together and how they are correlated) is categorized as clustering based on similarity in time. Dynamic Time Warping (DTW) [Sakoe & Chiba 1971, Sakoe & Chiba 1978], on the other hand is a elastic dissimilarity measure that finds an optimal alignment between two time-series by non-linearly warping them. It is appropriated for finding similar time-series in shape, which is a more general case of similarity in time. Various work has been developed for time-series clustering, for a survey we refer to the work by [Aghabozorgi *et al.* 2015].

A review on distance-based constrained clustering. In many time-series applications, there are expert knowledge that need to be considered during the clustering process. An example of applications is to provide a topology of changes, which are extractable from time-series of images. This application is the objective of the project A2CNES, that involves the laboratories ICube (*Laboratoire des sciences de l'ingénieur, de l'informatique et de l'imagerie*, University of Strasbourg), Live (*Laboratoire Image, Ville, Environnement*, University of Strasbourg) and the CNES (*Centre nationale d'études spatiales*). In this application, expert knowledge is represented by thematic constraints. These constraints once transformed to user-constraints need to be considered by the clustering process. We have therefore explored the use of different constrained clustering methods for time-series. This work has been performed by the following tasks:

- A review of constrained clustering methods: a wide range of partitioning methods using distance/similarity were considered. The methods range from algorithmic approaches to declarative approaches, from using the constraints to guide the search process to using them to learn a metric before and/or during searching, and from constructing a clustering directly from the dataset to constructing a clustering from a set of given clusterings.

- An adaptation of methods (if necessary) to using DTW in time-series: the methods are selected from implementations that are publicly available. While some algorithms can be directly applied to use DTW, others require modification the algorithm itself to integrate the measure.

- An evaluation of the methods on publicly available time-series data. Nine datasets are taken from the UCR time-series classification archive [Chen *et al.* 2015]. Clus-

tering tasks with instance-level must-link and cannot-link constraints were experimented.

For a review of constrained clustering methods, a wide range of partitioning clustering methods that use distance/similarity were considered. They can be categorized as follows.

- K-means based methods. In this type of approach, a classic clustering algorithm or the objective function is modified so that user-constraints are used to guide the algorithm towards a more appropriate clustering. Most of the work consider instance level must-link and cannot-link constraints. The extension is done either by enforcing pairwise constraints or by using them to define penalties in the objective function.

- Metric learning methods. User-constraints are used to learn a metric between the objects. The metric is then given to a clustering algorithm.

- Hybrid methods, which integrate both constraint enforcing and metric learning into a single framework.

- Spectral constrained clustering. Spectral clustering considers weighted graph that represents similarity between the objects and aims to find a partition of the graph such that the edges between different groups have a very low weight and the edges within a group have high weight. User-constraints can be taken into account either as "hard" or "soft" constraints.

- Declarative approaches, developed using general optimization tools such as SAT, constraint programming or integer linear programming. These approaches allow a large range of user-constraints and objective functions.

- Ensemble and collaborative clustering. These approaches take in input several given clusterings (or clustering algorithms) and construct a consensus clustering. In ensemble clustering, the result clustering is constructed by using a consensus function. In collaborative clustering, the input clustering algorithms collaborate together by communicating and changing their parameters until a consensus. Constraints can be integrated in the consensus function or to guide the collaborative process.

Among all the methods, 14 methods whose implementation is available were studied for the application with time-series data. Some of them can be directly applied to time-series using DTW (spectral clustering, declarative approach using constraint programming), the others need modifications inside the algorithm to integrate the measure (K-means based methods). Ten of these methods were not appropriate because of the restriction on the number of clusters, the cost inferred by an intensive use of DTW computation, or the used heuristic that can not be directly extended to DTW. Four methods were finally chosen for analysis: spectral clustering [Kamvar *et al.* 2003], spectral clustering with kernel matrix learning [Li & Liu 2009], collaborative clustering [Forestier *et al.* 2010] and our approach using constraint programming. All of these methods can handle instance level must-link and cannot-link constraints, our method can integrate cluster-level constraints.

Dataset	# classes	# data points	# time points
UCR dataset			
ECG5000	5	4500	140
ElectricDevices	7	7711	96
FacesUCR	14	2050	131
InsectWingbeatSound	11	1980	256
MALLAT	8	2345	1024
StarLightCurves	3	8236	1024
TwoPatterns	4	4000	128
UWaveGestureLibraryAll	8	3582	945
UWaveGestureLibraryX	8	3582	315
Real dataset			
Sud	12	9869	11

Table 5.11: Time-series datasets used in the experimentations.

Application to time-series data. Nine datasets of the UCR repository [Chen *et al.* 2015] were chosen, so that they represent typical time-series clustering problems. One real data was used that represents series of 11 images of size 1000×1000 of an agricultural zone in the South-East of France. The images were taken by the satellite SPOT4 and provided by CESBIO (*Centre d'Études Spatiales de la Biosphère*). For this time-series data, we have a partial ground truth, which consists of 11843 labeled pixels. The dataset was sampled into a dataset having 9869 labeled instances. The properties of the datasets are described in Table 5.11.

The chosen methods were experimented with must-link and cannot-link constraints. To generate a constraint, a pair of instances was randomly drawn from the instances, and depending on their labels a must-link or a cannot-link constraint was generated. Different sizes of constraint sets were defined, they correspond to 5%, 10%, 15% and 50% of the number of instances. For each size, 10 constraint sets were randomly generated. The quality of the solution was measured using the Adjusted Rand Index [Hubert & Arabie 1985] and the Normalized Mutual Information [Fred & Jain 2003].

Our framework was used with the criterion of minimizing the maximal diameter of the clusters. The application was accomplished by Nicolas Serrette during his Master internship (April - September 2017). In order to handle these datasets with large number of instances, several restrictions and improvements have been made. We observed that when minimizing the maximal diameter of the clusters, the search can be stopped before all the instances are assigned without changing the optimal value. Indeed, the diameter of a cluster is the maximal dissimilarity between two instances within the cluster. Let d be the dissimilarity measure, $D.lb$ be the lower bound of the diameter and x be an instance that has not been assigned to a cluster. If $\max\{d(x,i)\} \leq D.lb$ then x can be assigned to any cluster without changing the value of the diameter D, that is x can be ignored. That means if we detect that all the unassigned instances can be ignored, the search for the optimal value of diameter can be stopped. This is of course valid if each ignored instance is not concerned by any must-link or cannot-link constraint. After the

search, the unassigned instances are assigned into a cluster by a post-process. Different choices were experimented to assign the remaining instances:

- Closest cluster: The distance from an instance x to a cluster c is defined as the largest distance from x to each instance in the cluster. Instance x is then assigned to the cluster having the smallest distance.
- K-nearest neighbors: For each remaining instance x, the K nearest assigned neighbors are considered, and x is assigned to the cluster that is the most representative among these neighbors.
- Nearest medoid: Using the assigned instances, the medoid of each cluster is identified. The medoid of a cluster is the instance having the minimal sum of distances to the other instances of the cluster. Each remaining instance is assigned to the cluster corresponding to the nearest medoid.
- K-medoid: A K-means like algorithm is used to assign the remaining instance, but instead of computing the centroid, a medoid is used for each cluster. This algorithm does not change however the instances already assigned by the search.

The experiments show that depending on the dataset, the number of ignored instances varies a lot. It can be about ten instances for dataset MALLAT or InsectWingsBeat, or up to more than 5000 instances for ElectricDevices (which represents 67% of the instances). Comparing the different choices of post-process assignment, we observe that on average the closest cluster choice gives less good results and the K-medoid choice gives better result. It requires however the most in complexity to be achieved.

The numbers of instances and of clusters in the datasets are usually large. In order to handle the large number of tests, a timeout was also set to 30 minutes. The solution found so far was return at timeout. Therefore for our approach will not guarantee the optimal solution, but it always guarantee that all constraints are satisfied. In analyzing the results, some observations were made concerning our approach as follows:

- The ARI and NMI indexes, excepting two datasets MALLAT and StarLightCurves, are low – under 50%. This may come from three factors: (a) the criterion of minimizing the diameter may not be appropriate, (2) the DTW metric for the dissimilarity may not correspond the best for the data and (3) the ground truth used to generate constraints may contain errors. The last point was clearly observed for the real dataset Sud, where for the sake of simplification, some areas with agricultural roads or small plots have been declared as a single area of a single crop type.
- Must-link and cannot-link constraints, when added, do not always improve the quality of the result measured by ARI or NMI. It is evident that when the constraints are enough to describe exactly the ground truth partition then the result will be the ground truth partition. However this case is unrealistic since the number of pairwise constraint is about $\frac{1}{2}N(N-1)$ for N instances, which can be huge for some thousands of instances. The experiments was conducted with up to $\frac{1}{2}N$ pairwise constraints, which represents a very small part compared to all the possible pairwise constraints. The deterioration can be observed for all the approaches, but is mostly significant for CPClustering, which among the four selected methods, always

satisfies all the constraints. This observation agrees with that also made before on pairwise constraints with the minimization of WCSS in [Davidson *et al.* 2006], and in our work with the minimization of the maximal cluster diameter (cf. Subsection 5.2.3).

From this analysis some questions and challenges have been identified:

1. Pairwise constraints when added to the clustering process can deteriorate the quality of the solution. Measuring the usefulness of constraints is therefore beneficial. To measure the usefulness of a constraint set, two scores *informativeness* and *coherence* have been proposed in [Davidson *et al.* 2006]. When used together, these scores can give significant insight on the usefulness of a constraint set to improve the quality of the solution. However the coherence is defined for Euclidean distance and it is not obvious how this can be extended to DTW.

2. A large number of constraints may have an impact that is not always positive on the efficiency of the used method as well as the quality of the solution. It would be therefore useful to sample the constraint set into a smaller one without loss of quality. This could be done using a measure on the usefulness of constraints. Sampling the set of instances into a smaller but relevant one is also a challenge to be considered.

3. The experiments were restricted with must-link and cannot-link constraints. In practice, thematic constraints can be extremely broad and have to be translated into actionable constraints. Several kinds of actionable constraints are available. Nevertheless, generating actionable constraints from thematic constraints is not always straightforward. A thematic constraints on a set of data points can rapidly lead to a significant increase in both the number and the scope of the constraints. An example is a constraint that states two sets of points are "of different natures". Depending on the context this constraint can correspond to a disjunction of several sets of constraints.

4. The expert can provide constraints that are not always exact. A consequent is that the constraints can lead to over constrained problem or unsatisfied problem. It would be useful to enable the user to add or to remove constraints, or to rank the constraints.

5. The expert can provide constraints that are only partial or that can change depending on the context. An iterative process that takes into account expert feedback to improve the result will be useful.

5.6 Summary

We have explored the use of constraints in different clustering tasks and approach using constraint programming. On the modeling side, using constraint programming enables the modeling of more general clustering tasks. We have generalized constrained clustering problem to combinations of dissimilarity based and conceptual based clusterings. User constraints have been also categorized so that they can be stated on properties in order to make clustering actionable. Guidance by constraints up to now is provided *a priori*

before the clustering process. We have defined and developed a framework for a new problem, minimal clustering modification, which allows providing guidance *a posteriori*. This has the advantage of allowing feedback to be injected for the result of any clustering algorithm. On the effectiveness side, we have improved the efficiency of our constraint programming approach for constrained clustering with the WCSS criterion (within cluster sum of squares). This makes our approach so far the best in performance compared to existent methods on exact constrained clustering for WCSS. On the application side, the use of our framework in an application of constrained clustering on time-series images involved improving further our framework as well as identifying challenges to consider.

Chapter 6

Conclusion

The declarative approach of constraint programming enables its application to a wide range of problems. Using constraint programming, the problem must be formulated as a constraint satisfaction or a constraint optimization problem. In this way, solving the problem becomes exploring the search space to find one or all the solutions. We have developed approaches using constraint programming for problems issued from natural language processing and from data mining.

In natural language processing, one important task is to analyze the syntax of utterances according to a grammar. We have considered property grammars, where a grammar is given as a set of properties that must be satisfied by the grammatical utterances. We introduced model-theoretic semantics of property grammars, which allow to formulate the syntactic analysis as a combinatorial search problem. We developed a formalization as a constraint optimization problem and modeled it using constraint programming, which leads to an implementation of a fully constraint-based parser for property grammars. One main consequence is that the modeling of the problem is decoupled from its solving. This approach will enable the consideration of constraints coming from different aspects of a language, such as dependency between constituents.

In data mining, a large range of problems are represented as combinatorial problems. We have considered constrained clustering problems, where the objective is to find a partition of the objects that optimizes a criterion and that satisfies user-constraints. We have developed a declarative framework that integrates several principal optimization criteria and all popular types of user-constraints. We showed that the flexibility of the framework allows using it to find the complete Pareto front of bi-objective constrained clustering problems, which in our knowledge, is the first method in bi-objective clustering that integrates user-constraints. We explored new clustering problems and showed that the flexibility and variety of constraint programming can be exploited to solve them. These problems extend constrained clustering by allowing experts to provide complex constraints that make a clustering useful in the domain. They allow providing feedback a posteriori after a clustering is found.

The efficiency of our approach using constraint programming is one issue we studied in order to make it valuable among the existent approaches. We have improved the model by a better choice of variables and constraints. One key power of constraint programming resides in global constraints with their filtering algorithm. Exploiting further this power, we have developed specific global optimization constraints for principal clustering optimization criteria. This work makes our approach outperform the existent ones. Regarding the most popular minimum sum-of-squares constrained clustering, we showed that the constraint programming model can be combined within a more general process to reach even better performance.

My research perspectives are extension of my work on declarative approaches for data

mining. The objective is to develop efficient declarative methods for data mining problems. This is represented by: (a) the scope and the use of user-constraints and objective functions, (b) the scalability of the declarative approach and (c) the integration of human in the data mining process.

Scope and use of constraints and objective functions. Our approaches searches for solutions that optimizes a criterion and that satisfies all the user-constraints. We have observed that when user-constraints are instance-level must-link or cannot-link constraints, increasing the number of constraints does not always improve the quality of the solution. The constraints can also be incorrect or conflicting since the information can be incomplete or inexact, or the data can contain noises. Moreover, a large number of constraints usually yields a more difficult problem. One solution would be sampling the large set of constraints to a smaller but relevant one. One idea is using the measures for the usefulness of each constraint proposed in [Davidson *et al.* 2006] to construct a relevant constraint set, when the Euclidean distance is used. When the dissimilarity measure is not a distance, as for instance DTW, a new measure of usefulness should be developed.

Extension of the scope of constraints or objective functions is also important to better exploit user knowledge. Firstly, user knowledge or thematic constraints need to be expressed by actionable constraints or objective function. Thematic constraints can be on instances, on clusters, but also can be on subsets of instances that yield the need of complex constraints. Secondly, user knowledge can be defined on complex data. The attributes that describe data are usually used to compute distances as in usual distance based clustering or are discretized to binary for instance to define concept as in conceptual clustering. In order to fully exploit complex data, we should be able to consider numerical or symbolic information, with properties of instances or with relational information between the instances. And thirdly, expressing user feedback on clusterings will need the development of new types of relations and constraints on a meta level. This will extend the scope of the constraints to involve complex but meaningful constraints as well as objective functions. We continue exploring clustering problems that exploit complex data.

Scalability. The flexibility and the expressiveness are strong points of a declarative approach, but its limit is the scalability, since an exhaustive exploration of the search space is achieved in order to prove the optimality. One important direction is therefore the scalability.

We have shown in our work that changing the model can give significant improvement and have shown the interest of set constraints when combining distance-based and conceptual-based constraints and objectives. In conceptual clustering, recent work [Chabert & Solnon 2017] has investigated further on set constraints in both modeling types to construct a clustering : one model is finding an assignment of instances such as in our model and the other is an orthogonal model using a set covering formulation, that composes a clustering by choosing the clusters among the cluster candidates, such as in [Mueller & Kramer 2010, Ouali *et al.* 2016]. In the dissimilarity-based clustering setting the approach using set covering formulation may generate an exponential number of clusters candidates, but in a more special case, for instance in an interactive process with user feedback on the composition of the clustering, this approach can be considered.

Moreover, we may exploit the strength of channeling constraint to combine this two points of view in modeling.

Constraint programming offers a large range of constraints to model new clustering or meta-clustering constraints or objective functions. To better exploit relations that are complex or that are not fully expressed using existing tools, the design of global (optimization) constraints will always be considered. Appropriate search strategies can also improve significantly the performance. We have extended the repetitive branch-and-bound algorithm to user-constraints and show that combining with constraint programming the obtained system has significant improvement. The order is however static and needs to be computed before the search. We plan to study dynamic order that would better improve the bounds. Another direction is to consider stopping the search before a complete variables assignment. For instance when minimizing the maximal diameter we have observed that without user constraints some instances can be ignored in order to stop the search before all instances are assigned. This observation has been extended with instance-level constraints, such that there must be no constraints related to an ignored instance. When consider cluster level constraints, the problem of deciding whether the search can be stopped could be considered as a constraint satisfaction sub problem. If the subproblem has a solution then the search can be stopped before assigning all the variables. Depending on the constraints, the cost of deciding a constraint satisfaction problem can be much less than the cost of continuing the assignment with backtracking in the optimization problem. For instance, an assignment clustering problem with minimum cluster size constraints is polynomial, since it can be transformed to a minimum cost flow problem [Bradley *et al.* 2000].

The development of global optimization constraints is on the line of integrating constraint programming and operations research concepts. The integrations of constraint programming, artificial intelligence and operations research techniques have shown the interest of combining the strength of each paradigm in various applications [Milano & Hentenryck 2011]. Depending on the choice of variables for modeling, clustering problems have been modeled either entirely as an integer linear program or entirely as a constraint programming problem. To better exploit the strength of constraint programming on global constraints and search, the strength of mixed integer linear programming on relaxation methods and cutting planes, integrated modeling needs to be studied. This kind of models would allow to express and to exploit parts that are represented by global constraints or by linear constraints. Another direction that needs also to be studied is the use of large neighborhood search in order to find a good solution.

Integration of human in the process. Constraint-based clustering finds clusters that satisfy user-specified preferences or constraints. To enable domain experts who are usually non-data mining experts, to use a knowledge discovery system, it is important to have them involved in the process. Han wrote in [Han *et al.* 2006]:

> User input regarding important dimensions or the desired results will serve as crucial hints or meaningful constraints for effective clustering. In general, we contend that knowledge discovery would be most effective if one could develop an environment for human-centered, exploratory mining of data, that is, where the human user is allowed to play a key role in the process. Foremost, a user

> should be allowed to specify a focus – directing the mining algorithm toward the kind of "knowledge" that the user is interested in finding. Clearly, user-guided mining will lead to more desirable results and capture the application semantics.

This motivates the idea of an interactivity which is based on an iterative process such as the Mine (a solution), Interact (with the user), Learn (her preference), Repeat (the process with updated constraints) framework [van Leeuwen 2014]. The idea of facilitating the integration of user in the loop has also been addressed by the development of a declarative language for constraint-based mining [Guns *et al.* 2017], or of query based approaches for constrained clustering using SAT [Métivier *et al.* 2012b] or SQL [Adam *et al.* 2013]. An interactive process in clustering will require efficiently integrating user feedback as well as exploiting or learning user feedback expression. An interactive clustering algorithm has been designed in [Awasthi *et al.* 2017], which enables local change feedback such as splitting/merging clusters or must-link/cannot-link constraints. Interactivity by visual exploration approaches have been developed to easy the user for giving feedback [Boudjeloud-Assala *et al.* 2016, Puolamäki *et al.* 2016].

Given a clustering task specified by an optimization criterion and user-constraints, a clustering algorithm usually returns a single solution (exact or approximative). The result can give a model or a view on the data, however it may not entirely fit the user. User feedback is therefore taken into account to improve the result. User feedback can be considered with several scenarios:

- the given clustering may in general satisfy the requirements but the user may wish to explore some "similar" clusterings,
- the user may be interested in a clustering of similar quality but very different in term of partition, which is called alternative clustering [Bae & Bailey 2006, Davidson & Qi 2008, Kontonasios & Bie 2015]
- the user may be interested only in some clusters and require them in a new clustering, or may want to modify some clusters by merging or splitting them [Awasthi *et al.* 2017],
- the user may wish to keep the composition of a subset of instances.

This will require constraints on instances, on clusters but also on clusterings. This can be seen as an extension of meta-clustering problems [Caruana *et al.* 2006] with consideration of user constraints on different levels. Furthermore, in order to develop a scalable method, we may need to consider the case where the data is divided in several parts. Each part is clustered and the assembly is achieved in an interactive way, where constraints are not only on clusters but can be also on clustering. Considering clusterings, we will need to determine a measure that enables the comparison of clusterings, an objective that formulates the search of a clustering. The objective can be defined based on a measure or can also represent a semantically meaningful function. Exploiting the flexibility of constraint programming and enhancing its effectiveness to enable an efficient consideration of various constraints as well as objective functions, will make the approach using constraint programming a good candidate to integrate human in the process.

Appendix A

Other Research Topics

Contents

This chapter presents the other research topics that I worked on. Section A.1 presents work on solving first order logic constraints in tree structures. I developed this work during and after my PhD thesis, in collaboration with Alain Colmerauer and Khalil Djelloul (University Aix-Marseille II). Upon my arrival at CA team (Contraintes et Apprentissage), I started a joint work with Abdel-Ali Ed-Dbali, Lionel Martin, Arnaud Lallouet and Andreï Legtchenko (University of Orléans). We initialized a topic on learning solver for finite domain constraints. Section A.2 presents our work on this topic.

A.1 Solving Constraints in Tree Structures

The tree algebra plays a fundamental role in Computer Science: it models data structures, program schemes or program executions. Term unification, developed in 1965 by A. Robinson [Robinson 1965], corresponds to solving a conjunction of equations in the algebra of finite trees. B. Courcelle studied the properties of infinite trees in the context of recursive programs [Courcelle 1983]. A. Colmerauer described the execution of programs in Prolog II, III and IV in term of solving equations and disequations in the algebra of finite and infinite trees [Colmerauer 1982, Colmerauer 1984, Colmerauer 1990]. Solving conjunctions of equations in finite trees theory was also the subject of various work [Huet 1976, Jaffar 1984, Ramachandran & Van Hentenryck 1993].

The properties of the algebra of finite, rational or infinite trees have been axiomatized in a complete first order theory [Maher 1988]. This theory is defined on an infinite set of function symbols F and an unique relation symbol, which is the equality $=$. Each function symbol $f \in F$ has an arity which is positive or nul. A function symbol having arity nul is called a constant. Let V be an infinite set of variables. A *term* is an expression which is either x, with $x \in V$, either $f(t_1, \ldots, t_n)$, where $f \in F$ is of arity n and $t_1, \ldots, t_n$ are terms.

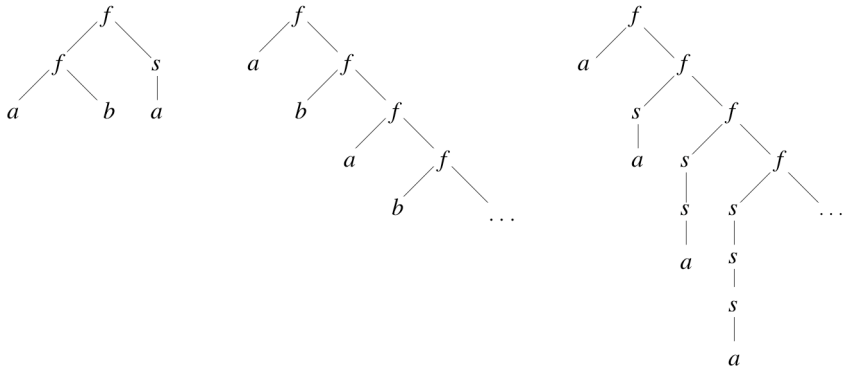
Definition A.1 *The theory of finite, rational and infinite trees is the infinite set of propositions in one of the three following forms [Maher 1988]:*

$$\begin{array}{ll} \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_m & \neg(f(x_1, \ldots, x_n) = g(y_1, \ldots, y_m) \\ \forall x_1 \ldots \forall x_n \forall y_1 \ldots \forall y_n & f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \ \rightarrow \bigwedge_i x_i = y_i \\ \forall x_1 \ldots \forall x_n \exists! y_1 \ldots \exists! y_m & \bigwedge_i y_i = t_i[x_1, \ldots, x_n, y_1, \ldots, y_m] \end{array}$$

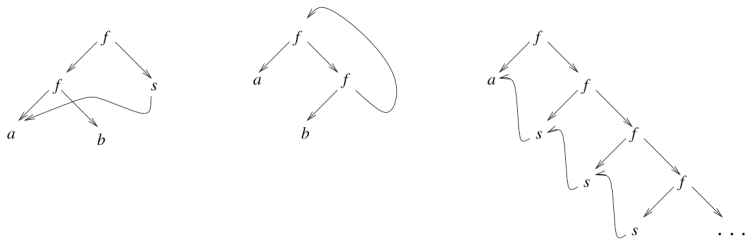
where $f, g \in F$ and $f \neq g$ and $t_i[x_1, \ldots, x_n, y_1, \ldots, y_m]$ is a term formed by an element in F and variables taken from $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$.

These forms are also called the axiom schemes of the theory. M. Maher proved that this theory T is complete, i.e. for all proposition p, either $T \models p$, either $T \models \neg p$. This theory is called tree theory since the algebra of finite, rational and infinite trees is a model of this theory, as explained below.

The interpretation domain is the set of trees, whose nodes are labeled by elements of F. To each element $f \in F$ of arity n is associated an operation called construction operation $(a_1, \ldots, a_n) \mapsto b$, where $a_1, \ldots, a_n$ are trees and b is the tree whose root is labeled by f and possesses as direct subtrees the trees $a_1, \ldots, a_n$. A constant f is interpreted by a tree having only one node labeled by f. A term $f(t_1, \ldots, t_n)$ is interpreted by a tree whose root is labeled by f and the direct subtrees are trees that interpret $t_1, \ldots, t_n$. A *finite tree* is a tree having a finite set of nodes. An infinite tree has an infinite number of nodes. For instance, let a, b be constants, f be of arity 2 and s of arity 1. Among the three following trees, the first one is finite and the two other are infinite:



A rational tree is a tree having a finite number of subtrees. For instance, the two first one are rational and the last one is infinite non rational. Indeed, their subtrees are identified with the following schemes:



The first tree can be expressed by:

$$x = f(y, z) \wedge y = f(u, v) \wedge z = s(u) \wedge u = a \wedge v = b$$

the second by:

$$x = f(u, y) \wedge y = f(v, x) \wedge u = a \wedge v = b$$

The third one cannot however be presented by a finite conjunction of equations.

A.1.1 Solving Constraints in The Tree Theory

This subsection presents work during my PhD thesis [Dao 2000b], supervised by Alain Colmerauer (University Aix-Marseille II). The objective was to construct an algorithm to solve first order constraints in the tree theory, i.e. in all the models of the theory. First order constraints are of one of the following forms:

$$s = t,\ true,\ false,\ \neg(\varphi),\ (\varphi \wedge \psi),\ (\varphi \vee \psi),\ (\varphi \rightarrow \psi),\ (\varphi \leftrightarrow \psi),\ \exists x\varphi,\ \forall x\varphi$$

where x is a variable, s, t are terms and φ and ψ are constraints. These constraints therefore can be stated with any logical connection and can have nested and alternated quantifiers. One example of quantified constraints is in two player games, where we want to find the positions where the first player can win after at most k moves.

Example 2 Let $move(x, y)$ be a predicate that defines the possibles moves from a position x to a position y. Let $winning_k(x)$ and $losing_k(x)$ be the predicates stating that the player who plays at the position x can win, no matter the way the other plays (or always lose, respectively) after at most k moves. These predicates can be defined as follow:

$$\begin{array}{lcl} winning_0(x) & \leftrightarrow & false \\ winning_{k+1}(x) & \leftrightarrow & \exists y\ (move(x, y) \wedge losing_k(y)) \\ losing_k(x) & \leftrightarrow & \forall y\ (move(x, y) \rightarrow winning_k(y)) \end{array}$$

Trees can be used to represent the positions in a game and constraints in the tree structure can be used to represent the relation $move(x, y)$. Let us consider a two player game where we start with a non negative integer n and in turn each player subtracts 1 or 2 without making n negative. The first player who cannot play loses. The integers here are interpreted by trees such that 0 is represented by 0 and $i > 0$ by $s^i(0)$. The predicate $move(x, y)$ can be defined as follow:

$$move(x, y) \stackrel{def}{=} x = s(y) \vee x = s(s(y)) \vee (\neg(x = 0) \wedge \neg\exists u\,(x = s(u)) \wedge x = y)$$

The constraint $winning_k(x)$, once developed, is represented by a first order constraint with nested and alternated quantifiers and with the free variable x. The solutions for $winning_1(x)$ are:

$$x = s(0)\ \vee\ x = s(s(0))$$

and the solutions for $winning_2(x)$ are:

$$x = s(0)\ \vee\ x = s(s(0))\ \vee\ x = s(s(s(0))))\ \vee\ x = s(s(s(s(0)))))$$

□

Solving constraints by rewriting. I developed a method for eliminating quantifiers and defined a normal form for the constraints, where the solutions can be deduced explicitly. Using this quantifier elimination method, solving a constraint becomes transforming it to an equivalent formula, which is either $false$ (there is no solution for the constraint) or a formula in the normal form (the solution are deduced directly). The transformation is achieved by sub-formula rewriting. I proposed a set of eleven sub-formula rewrite rules. Each rewrite rule transforms a sub-formula to an equivalent formula. To transform a constraint, the rewrite rules are applied as many times as possible on sub-formulas of the constraint and they can be applied independently. I proved that their application always terminates and the final formula, which is equivalent to the initial one, is either $false$ or in the normal form [Dao 2000a]. I implemented in C++ a tree constraints solver using these rewrite rules. The solve is capable to solve complex constrains, for instance winning position constraints having up to 160 nested and alternated quantifiers.

Expressiveness of tree constraints. We showed that constraints in tree structure have a quasi-universal expressive power [Colmerauer & Dao 2000, Colmerauer & Dao 2003]. Let $\alpha(k)$ be a function defined for each integer $k \geq 0$ such that $\alpha(0) = 1$ and $\alpha(k+1) = 2^{\alpha(k)}$. This function increases in a stunning way, since $\alpha(0) = 1$, $\alpha(1) = 2$, $\alpha(2) = 4$, $\alpha(3) = 16$, $\alpha(4) = 65536$ and $\alpha(5) = 2^{65536}$. We constructed a family of strongly expressive tree constraints: by a tree constraint of size proportional to k, we define a tree having exactly $\alpha(k)$ nodes. Using this family we can express for instance the multiplication table computed by a Prolog machine executing up to $\alpha(k)$ instructions. By replacing the Prolog machine with a Turing machine, we showed the quasi-universality of tree constraints, i.e. the ability to concisely describe trees which the most powerful machine will never have time to compute. As such we also rediscover the following result by S. Vorobyov [Vorobyov 1996]: the complexity of an algoirthm, deciding whether a tree constraint without free variables is true, cannot be bounded above by a function obtained from finite composition of simple function including exponentiation.

Extension with finite trees. Let $finite(x)$ a predicate expressing x is a finite tree. We proposed an extension $T \cup \text{FINITE}$ of the tree theory T, having as model the tree algebra. We proposed a set of 16 rewrite rules to solve constraints in this theory. The rewrite rules transform the constraint to solve in an equivalent formula, which is either $false$ or a formula in the normal form. Therefore if the initial constraint does not have free variables, the final formula will be either $false$ or $true$. In the same way such that the 11 rewrite rules confirm the completeness of the tree theory T, the 16 rewrite rules show that the theory $T \cup \text{FINITE}$ is complete. For the theory of finite trees, we proposed 12 rewrite rules to solve first order constraints.

A.1.2 Solving Constraints in Extended Tree Theories

This subsection describes work after my PhD thesis. This work was developed in collaboration with Khalil Djelloul (University Aix-Marseille II).

Solving constraints in decomposable theories. In generalizing the quantifier elimination method that I had developed, we identified sufficient properties for a theory so that

the quantifier elimination can be applied. This category of theories is called decomposable theories. We showed that several fundamental theories are decomposable, for instance the theory of dense linear order without limit, the theory of rational numbers with addition and subtraction, the theory of queues or the theory of lists. We generalized the algorithm for solving constraints in the tree theory to solve constraints in a decomposable theory [Djelloul & Dao 2006c].

Solving constraints in evaluated trees structure. Let $\boldsymbol{Q}$ be the set of rational numbers. We studied the tree structure, whose trees have nodes labeled by elements of $\boldsymbol{Q} \cup F \cup \{+, -\}$. The trees are evaluated such that the subtrees labeled only by elements of $\boldsymbol{Q} \cup \{+, -\}$ are evaluated and reduced to an element of $\boldsymbol{Q}$. This structure reflects the essential of Prolog III and IV, which are modeled by a combination of trees and rational numbers, booleans and intervals [Colmerauer 1984, Colmerauer 1990]. We defined a normal form for constraints where the solutions are explicit. For solving constraints in this structure, we developed a set of rewrite rules, which transforms the initial constraint to either $false$ or a formula in the normal form [Dao & Djelloul 2006].

Combination of theories with the tree theory. Let T be a decidable theory. We studied the combination of the theory T with the tree theory, denoted by T^*, in the case where the signatures of these theories can overlap. We identified the properties that need to be satisfied by the theory T, in order to make T^* complete. We proposed a decision procedure for T^*, which is presented by a set of 6 sub-formula rewrite rules [Djelloul & Dao 2006b]. Considering T the theory of queues, we developed an algorithm for solving constraints in the theory combining trees with queues [Dao 2009].

Extension into first order of Prolog model. The execution of Prolog programs corresponds to the resolution of conjunctions of equations and disequations in the finite and infinite trees structure. In order to extend the Prolog model into a solver of general first order constraints in this structure, we extended the tree theory with the relation $finite(t)$, stating that t must be a finite tree. We developed an algorithm for solving constraints in this theory and implemented it using C++ and the formalism of CHR [Djelloul *et al.* 2007, Djelloul *et al.* 2008].

A.2 Learning Finite Domain Constraint Solver

Constraint programming solvers are based on constraint propagation and search. Constraint propagation is achieved for each constraint of the problem. Propagating a constraint c consists in removing some inconsistant values from the domain of the variables of c. The propagation of a constraint c is accomplished by the propagators associated to c. This means in CP solvers, each constraint scheme is associated with a set of propagators or with a filtering algorithm. The propagators or the filtering algorithm are designed to enforce the constraint with a certain level of consistency. Different types of consistency exist and the propagators can have difference effects on the variable domain. For instance, propagators enforcing arc-consistency remove arc-consistent

values and therefore need to consider each value, while propagators enforcing bound-consistency move only upper and lower bounds of the domain. The capacity of removing inconsistent values reduces the search space. However since the solver interleaves constraint propagation with search, the propagators need also be fast enough. The efficiency of the solver therefore depend strongly in the efficiency of the propagators. The task of finding efficient propagators which actually define a local consistency is considered as one of the smartest skills of the solver designer. Pioneering work had investigated on automatic construction of solvers: constructing rule-based propagators using systematic search [Apt & Monfroy 1999], constructing propagators by rewrite rules in CHR frameworks using machine learning techniques[Abdennadher & Rigotti 2002, Abdennadher & Rigotti 2004].

An efficient technique for computing consistencies is to use a data representation for the CSP and a set of operators whose common fixed point models the expected consistency. The operators are then applied via chaotic iteration until reaching their common fixed point [Apt 1999]. In our approach we developed a framework using Machine Learning techniques for learning a finite domain constraint solver. Given a CSP, solver learning consists in an automatic generation of propagators for each constraint to enforce a desired local consistency. The principe of the learning framework is considering the behavior of the operator enforcing the desired consistency as a set of examples, in order to find an adequate representation of this operator in a given language. The contributions presented below were published in [Dao *et al.* 2002, Lallouet *et al.* 2003a, Lallouet *et al.* 2003b, Ed-Dbali *et al.* 2003] and enjoyed a collaboration with Abdel-Ali Ed-Dbali, Lionel Martin, Arnaud Lallouet et Andreï Legtchenko.

A.2.1 Theoretical Framework

Let V be a set of variables and $D = (D_X)_{X \in V}$ their finite domains. A *constraint* c is a pair (W, T) where $W \subseteq V$ is the arity of the constraints and $T \subseteq \prod_{X \in W} D_X$ is the set of solutions of c. A *CSP* is a set of constraints. For $W \subseteq V$, a *search state* s is a tuple $(s_X)_{X \in W}$ where $\forall X \in W$, $s_X \subseteq D_X$. A *singletonic* search state $(\{v_X\})_{X \in W}$ represents a single tuple. The *search space* is $S_W = \prod_{X \in W} \mathcal{P}(D_X)$. The set S_W ordered by point-wise inclusion $\subseteq$ forms a complete lattice.

A *consistency* for a constraint $c = (W, T)$ is an operator $f : S_W \to S_W$ such that:

- f is monotonic, i.e $s \subseteq s' \Rightarrow f(s) \subseteq f(s')$, in order to ensure the confluence of the reduction mechanism,
- f is contracting, i.e $\forall s \in S_W$, $f(s) \subseteq s$, in order to reduce variable's domains,
- f is correct w.r.t c, i.e $\forall s \in S_W$, every solutions of c which are present in s remain in $f(s)$,
- f represents c, i.e for every singletonic search state s which does not represent a solution of c, $f(s)$ is an empty state (at least one element of the tuple $f(s)$ is the empty set).

Operators which satisfy the first three conditions are called *pre-consistencies* for c. As an example of consistency, if we suppose that each variable domain D_X is ordered by a total

ordering $\leq$ and for $A \subseteq D_X$, we denote by $[A]$ the set $\{a \in D_X \mid \min(A) \leq a \leq \max(A)\}$, then the bound-consistency bc_c is defined by $\forall s \in S_W$, $\forall X \in W$, $bc_c(s)_X = s_X \cap [T_X]$, with T_X the projection of T on X.

Let cs_c be the consistency to be learned. Our aim is to build a consistency f which behaves like cs_c as much as possible. Thus f must be contracting, monotonic, *correct w.r.t* cs_c ($\forall s \in S_W$, $cs_c(s) \subseteq f(s)$) and *singleton complete w.r.t* cs_c ($f(s) \subseteq cs_c(s)$ for any singletonic search state s). However, singleton completeness is difficult to get and even not always possible to express in a given language. In order to transform a pre-consistency into a consistency, let us define a consistency id_c such that $\forall s \in S_W$, $id_c(s)$ is an empty state if s is a non-solution singletonic state, and $id_c(s) = s$ otherwise. Thus $f \circ id_c$ and $id_c \circ f$ are consistencies for c if f is a pre-consistency for c. Therefore by adding id_c in the set of operators, processed by a chaotic iteration mechanism [Apt 1999], we only need to build pre-consistencies for c. On the other hand, the correctness condition must be ensured for every $s \in S_W$ which is generally huge. We showed that: If f is a monotonic and contracting operator such that $f(s) = s$ for every singletonic state s which represents a solution of c, then f is a pre-consistency for c. Therefore, by considering monotonic operators, we can reduce the search space to a sample set E which is a subset of S_W and which contains all singletonic search states. Let $\mathcal{L}$ be the language in which operators are expressed and l be an operator in this language. In order to find the best possible expression, we need to compare two consistencies. This is usually done with a distance. Let d be such a distance between two consistencies. The learning problem is formulated as follows:

$$\begin{array}{rl} \texttt{minimize} & d(cs_s, l), \\ \texttt{subject to} & \forall s \in E,\ cs_c(s) \subseteq l(s) \subseteq s, \end{array}$$

where $E \subseteq S_W$, E contains all singletonic search states of S_W and l is a monotonic operator. Following the machine learning vocabulary, cs_c represents the example space and $\mathcal{L}$ the hypothesis space.

A.2.2 Learning Indexicals

To instantiate our theoretical framework, we have to define strong language biases in order to limit the combinatorial explosion.

The first question is the language in which operators are expressed. The language of indexicals [van Hentenryck *et al.* 1991] is chosen, motivated by the ease of integration of the user-defined indexicals in Gnu-Prolog [Diaz & Codognet 2001]. In this language, an operator is written `X in r`, where `X` represents the domain of the variable X and `r` is an expression representing a subset of D_X. If we denote x the unary constraint representing X's domain, then the indexical represents the operator $x \mapsto x \cap r$.

Then comes the choice of consistency. We learn the bound-consistency, since it allows to limit the example space to intervals instead of arbitrary subsets.

For each variable we learn a reduction indexical and define an indexical for id_c. The reduction indexical for X is of the form `X in minX .. maxX` where `minX`, `maxX` are in some predefined forms. In order to be monotonic, the bound `minX` must be anti-monotonic and `maxX` monotonic. This can be ensured by syntactic conditions on the sign of the coefficients for each expression. Let $L = \{\texttt{minY}, \texttt{maxY} \mid Y \neq X\}$. In practice, the predefined form for

the expression can be linear:

$$\alpha_0 + \sum_{t \in L} \alpha_t t$$

quadratic:

$$\alpha_0 + \sum_{t,t' \in L} \alpha_{tt'} tt'$$

or rational form:

$$\alpha_0 + \sum_{t \in L} \frac{\alpha_t}{t+1}$$

The coefficients α_i are therefore to be learned. The indexicals for id_c could be implemented in two ways: by using Gnu-Prolog indexicals for predefined constraints in which each instance of `min` and `max` is simply replaced by `val`, or by a direct code using `val` and C operators.

As distance between two consistencies, we use the global error on the example space E. By considering that f must be correct w.r.t cs_c, this distance is:

$$\sum_{s \in E} |f(s) \backslash cs_c(s)|$$

A.2.3 Intermediate Consistency

Consistencies can be partially ordered according to their pruning power. This pruning power, however, should be put into balance with the complexity of enforcing them. For instance, the pruning power of path-consistency is great, but the price to pay is so high that the consistency is not used. Similarly, for many CSPs, bound-consistency is preferred to arc-consistency even if it does not remove values from the middle of variable domains. In Subsection A.2.2, a consistency weaker than bound-consistency but as close to it as possible was constructed. This subsection extends the method to build a range of consistencies for a given constraint. These intermediate consistencies are located between bound- and arc-consistencies.

This set of consistencies is provided by a new solver learning method based on a clustering of the constraint's tuples, a sampling of the search space and a repair technique which is able to fix a too weak operator. This consistency is called multibound-consistency. The idea is to isolate disjunctive chunks of the constraint and to apply bound consistency to each chunk. A constraint, given by the set of its solution tuples, is partitioned into clusters using a clustering algorithm. A clustering of a constraint $c = (W, T)$ is a set of constraints $CL = \{c_1 = (W, T_1), \ldots, c_n = (W, T_n)\}$ such that $\{T_1, \ldots, T_n\}$ forms a partition of T. The agglomerative complete-link clustering algorithm is used.

The consistency obtained consists in applying the bound-consistency on each separate cluster as follows. Let bc_{c_i} be the bound consistency for a constraint $c_i \in CL$. The multibound-consistency is defined by the operator:

$$\forall s \in S_W,\ mb_c(s) = s \cap \bigcup_{c_i \in CL} bc_{c_i}(s)$$

Note that when each cluster only holds one tuple, we get arc-consistency, and when all tuples are in a single cluster, we get bound-consistency. Since the clustering wraps some

holes, it is necessary to compose the learned operator with id_c in order to get a consistency. The multibound-consistency operator is obviously monotonic. The pruning power and the computational cost of this consistency is directly related to the number of allowed clusters. The ratio between filtering and search can be finely tuned by choosing a level of consistency in this set, instead of just bound- and arc-consistencies.

List of Publications

International Journal Articles

[1] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *Constrained Clustering by Constraint Programming*, Artificial Intelligence, Vol 244, pages 70–94, 2017.

[2] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier, *Model-Theory and Implementation of Property Grammars with Features*, Journal of Logic and Computation, Vol 24, No 2, pages 491-509, 2014.

[3] Khalil Djelloul, Thi-Bich-Hanh Dao, Thom Fruehwirth, *Theory of Finite or Infinite Trees Revisited*, Journal of Theory and Practice of Logic Programming, Vol 8, No 4, Pages 431-489, 2008.

[4] Alain Colmerauer, Thi-Bich-Hanh Dao, *Expressiveness of Full First-Order Constraints in the Algebra of Finite or Infinite Trees*, Journal of Constraints, Kluwer Academic Publishers, Vol 8, No 3, pages 283-302, 2003.

National Journal Articles

[5] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *Un nouveau modèle pour la classification non supervisée sous contraintes*, Revue d'Intelligence Artificielle, Vol. 28, No. 5, pages 523-545, 2014.

[6] Ali Ed-Dbali, Thi-Bich-Hanh Dao, Arnaud Lallouet, Andreï Legtchenko, *Apprentissage de solveurs de contraintes sur les domaines finis*, Technique et Science Informatique, Vol. 22, No. 1, pages 125-138, 2003.

International Conference Papers

[7] Chia-Tung Kuo, S. S. Ravi, Thi-Bich-Hanh Dao, Christel Vrain, Ian Davidson, *A Framework for Minimal Clustering Modification via Constraint Programming.* In the 31st AAAI Conference on Artificial Intelligence AAAI-17, 2017.

[8] Thi-Bich-Hanh Dao, Christel Vrain, Khanh-Chuong Duong, Ian Davidson, *A Framework for Actionable Clustering Using Constraint Programming.* In the 22nd European Conference on Artificial Intelligence ECAI, pages 453-461, 2016.

[9] Tias Guns, Thi-Bich-Hanh Dao, Christel Vrain, Khanh-Chuong Duong, *Repetitive Branch-and-Bound Using Constraint Programming for Constrained Minimum Sum-of-Squares Clustering.* In the 22nd European Conference on Artificial Intelligence ECAI, pages 462-470, 2016.

[10] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *Constrained Minimum Sum of Squares Clustering by Constraint Programming*, In 21st International Conference

on Principles and Practice of Constraint Programming CP, Cork, Ireland, August 31 - September 4, 2015, Proceedings. Lecture Notes in Computer Science 9255, Springer, pages 557-573, 2015.

[11] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *A Declarative Framework for Constrained Clustering*, In European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases ECMLPKDD, Prague, Czech Republic, September 23-27, Proceedings, Part III, Lecture Notes in Computer Science 8190, Springer, pages 419-434, 2013.

[12] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *A Filtering Algorithm for Constrained Clustering with Within-Cluster Sum of Dissimilarities Criterion*. In IEEE 25th International Conference on Tools with Artificial Intelligence ICTAI, Herndon, VA, USA, November 4-6, 2013, pages 1060-1067, 2013.

[13] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier, *Model-Theory of Property Grammars with Features*, Proceedings of the 12th International Conference on Parsing Technologies, IWPT 2011, October 5-7, Dublin City University, Dubin, Ireland, pages 75-79, 2011.

[14] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier, Willy Lesaint, *Property Grammar Parsing Seen as a Constraint Optimization Problem*, in Formal Grammar - 15th International Conference, FG 2010, Copenhagen, Denmark, August 2010. Revised Selected Papers. Lecture Notes in Computer Science 7395, Springer, pages 82-96, 2012.

[15] Denys Duchier, Jean-Philippe Prost, and Thi-Bich-Hanh Dao, *A model-theoretic framework for grammaticality judgements*. Formal Grammar - 14th International Conference, FG 2009, Bordeaux, France, July 25-26, 2009, Revised Selected Papers. Lecture Notes in Computer Science 5591, Springer, pages 17-30, 2011.

[16] Khalil Djelloul, Thi-Bich-Hanh Dao, and Thom Fruehwirth, *Toward a first-order extension of Prolog's unification using CHR*. In Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, pages 58-64, 2007.

[17] Thi-Bich-Hanh Dao, Khalil Djelloul. *Solving First-Order Constraints in the Theory of the Evaluated Trees*. In 22nd International Conference on Logic Programming, ICLP 2006, Proceedings. Lecture Notes in Computer Science 4079 Springer, pages 423–424, 2006.

[18] Khalil Djelloul, Thi-Bich-Hanh Dao, *Solving first-order constraints in the theory of finite or infinite trees: introduction to the decomposable theories*. In Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), pages 7-14, 2006.

[19] Khalil Djelloul, Thi-Bich-Hanh Dao, *Extension into trees of first order theories*. In Artificial Intelligence and Symbolic Computation, 8th International Conference, AISC 2006, Beijing, China, September 20-22, 2006, Proceedings. Lecture Notes in Computer Science 4120, Springer, pages 53-67, 2006.

[20] Arnaud Lallouet, Andrei Legtchenko, Thi-Bich-Hanh Dao, Ali Ed-Dbali, *Intermediated (Learned) Consistencies*, Principles and Practice of Constraint Programming - CP

2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings. Lecture Notes in Computer Science 2833, Springer, pages 889-893, 2003.

[21] Arnaud Lallouet, Thi-Bich-Hanh Dao, Ali Ed-Dbali, *Language, Definition and Optimal Computation of CSP Approximations*, Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference, May 12-14, 2003, St. Augustine, Florida, USA. AAAI Press, pages 182-186, 2003.

[22] Arnaud Lallouet, Andrei Legtchenko, Thi-Bich-Hanh Dao, Ali Ed-Dbali, *Finite Domain Constraint Solver Learning*, IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003. Morgan Kaufmann, pages 1379-1380, 2003.

[23] Thi-Bich-Hanh Dao, Arnaud Lallouet, Andrei Legtchenko, Lionel Martin, *Indexical-Based Solver Learning*, Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings. Lecture Notes in Computer Science 2470, Springer, pages 541-555, 2002.

[24] Alain Colmerauer, Thi-Bich-Hanh Dao, *Expressiveness of Full First Order Constraints in the Algebra of Finite or Infinite Trees*, Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings. Lecture Notes in Computer Science 1894, Springer, pages 172-186, 2000.

Workshop Papers

[25] Yohan Boichut, Thi-Bich-Hanh Dao, Valérie Murat, *Characterizing Conclusive Approximations by Logical Formulae*, In Proceedings of 5th International Workshop on Reachability Problems, RV 2011, Lecture Notes in Computer Science 6945, pages 72-84, 2011.

[26] Thi-Bich-Hanh Dao, Khalil Djelloul, *Solving First-Order Constraints in the Theory of the Evaluated Trees*. Recent Advances in Constraints, 11th Annual ERCIM International Workshop on Constraint Solving and Contraint Logic Programming, CSCLP 2006, Caparica, Portugal, June 26-28, 2006, Revised Selected and Invited Papers. Lecture Notes in Computer Science 4651, Springer, pages 108-123, 2006.

[27] Khalil Djelloul and Thi-Bich-Hanh Dao. *Complete First-Order Axiomatization of Finite or Infinite M-extended Trees*. In 20th Workshop on Logic Programming, Vienna, Austria, February 22–24, 2006. INFSYS Research Report 1843-06-02 Technische Universität Wien, Austria 2006

[28] Arnaud Lallouet, Andrei Legtchenko, Thi-Bich-Hanh Dao, Ali Ed-Dbali, *Learning Approximate Consistencies* , Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2003, Budapest, Hungary, June 30 - July 2, 2003, Selected Papers. Lecture Notes in Computer Science 3010, Springer, pages 87-106, 2003.

National Conference Papers

[29] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Tias Guns, Christel Vrain, *Branch-and-bound répétitif et programmation par contraintes pour le clustering sous contraintes*. In Douzième Journées Francophones de Programmation par Contraintes JFPC 2016.

[30] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *Clustering avec la minimisation de la somme des carrés par la programmation par contraintes*. In Onzième Journées Francophones de Programmation par Contraintes JFPC 2015.

[31] Thi-Bich-Hanh Dao, Willy Lesaint, Christel Vrain, *Clustering conceptuel et relationnel en programmation par contraintes*. In Onzième Journées Francophones de Programmation par Contraintes JFPC 2015.

[32] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *Classification non supervisée mono et bi-objectif par la programmation par contraintes*. In Dixièmes Journées Francophones de Programmation par Contraintes JFPC 2014.

[33] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *Un modèle général pour la classification non supervisée sous contraintes d'utilisateurs*. In Neuvièmes Journées Francophones de Programmation par Contraintes JFPC 2013.

[34] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain, *Une approche en programmation par contraintes pour la classification non supervisée*, Extraction et gestion des connaissances EGC, Actes, 29 janvier - 01 février 2013, Toulouse, France. Revue des Nouvelles Technologies de l'Information RNTI-E-24, Hermann-Éditions, pages 55-66, 2013.

[35] Denys Duchier, Thi-Bich-Hanh Dao, and Yannick Parmentier, *Analyse syntaxique par contraintes pour les grammaires de propriétés à traits*. In Huitième Journées Francophones de Programmation par Contraintes JFPC 2012.

[36] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier, and Willy Lesaint. *Une modélisation en CSP des grammaires de propriétés*. In Sixièmes Journées Francophones de Programmation par Contraintes JFPC 2010.

[37] Thi-Bich-Hanh Dao. *Un algorithme de décision dans l'algèbre des arbres finis ou infinis et des queues*. In Actes des Cinquièmes Journées Francophones de Programmation par Contraintes JFPC 2009.

[38] Khalil Djelloul, Thi-Bich-Hanh Dao, and Thom Fruehwirth. *Extension au premier ordre de l'unification des termes par CHR*. In Troisièmes Journées Francophones de Programmation par Contraintes JFPC 2007.

[39] Khalil Djelloul and Thi-Bich-Hanh Dao. *Résolution de contraintes du premier ordre dans la théorie des arbres évalués*. In Journées Francophones de Programmation par Contraintes JFPC 2006.

[40] Thi-Bich-Hanh Dao and Khalil Djelloul. *Complétude des extensions en arbres de théories*. In Journées Francophones de Programmation par Contraintes JFPC 2006.

[41] Arnaud Lallouet, Andrei Legtchenko, Thi-Bich-Hanh Dao, Ali Ed-Dbali, *Apprentissage de solveur de contraintes sur les domaines finis*, Journées Francophones de Programmation en Logique avec Contraintes (JFPLC 2003), Amiens, France, du 17 au 19 Juin 2003. Hermes/LavoisierJFPLC, pages 125-138, 2003.

[42] Thi-Bich-Hanh Dao, *Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis*, Programmation en logique avec contraintes, JFPLC 2000, 28-30 Juin 2000, Marseille, France. Hermes, pages 225-240, *Prix du meilleur article*, 2000.

PhD Thesis

[43] Thi-Bich-Hanh Dao, *Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis*, Université Aix-Marseille 2, 2000.

Submissions

[44] Thomas Lampert, Thi-Bich-Hanh Dao, Baptiste Lafabregue, Nicolas Serrette, Germain Forestier, Bruno Crémilleux, Christel Vrain, Pierre Gançarski, *Constrained Distance Based Clustering for Time-Series: A Comparative and Experimental Study*, submitted to Data Mining and Knowledge Discovery.

Bibliography

[Abdennadher & Rigotti 2002] Slim Abdennadher and Christophe Rigotti. *Automatic generation of rule-based solvers for intensionally defined constraints.* International Journal on Artificial Intelligence Tools, vol. 2, no. 11, pages 283–302, 2002.

[Abdennadher & Rigotti 2004] Slim Abdennadher and Christophe Rigotti. *Automatic generation of rule-based constraint solvers over finite domains.* Transaction on Computational Logic, vol. 2, no. 5, 2004.

[Adam *et al.* 2013] Antoine Adam, Hendrik Blockeel, Sander Govers and Abram Aertsen. *SCCQL : A Constraint-Based Clustering System.* In Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III, pages 681–684, 2013.

[Aggarwal & Reddy 2013] Charu C. Aggarwal and Chandan K. Reddy. Data clustering: Algorithms and applications. Chapman & Hall/CRC, 2013.

[Aghabozorgi *et al.* 2015] S. Aghabozorgi, A. Shirkhorshidi and T. Wah. *Time-series clustering – A decade review.* Information Systems, vol. 53, pages 16–38, 2015.

[Aloise & Hansen 2009] Daniel Aloise and Pierre Hansen. *An branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering.* Pesquisa Operacional, vol. 29, no. 3, pages 503–516, 2009.

[Aloise *et al.* 2009] Daniel Aloise, Amit Deshpande, Pierre Hansen and Preyas Popat. *NP-hardness of Euclidean Sum-of-squares Clustering.* Mach. Learn., vol. 75, no. 2, pages 245–248, May 2009.

[Aloise *et al.* 2012] Daniel Aloise, Pierre Hansen and Leo Liberti. *An improved column generation algorithm for minimum sum-of-squares clustering.* Mathematical Programming, vol. 131, no. 1-2, pages 195–220, 2012.

[Apt & Monfroy 1999] K. R. Apt and E. Monfroy. *Automatic generation of constraint propagation algo- rithms for small finite domains.* In International Conference on Principles and Practice of Constraint Programming, pages 58–72, 1999.

[Apt 1999] Krzysztof R. Apt. *The Essence of Constraint Propagation.* Theoretical Computer Science, vol. 221, no. 1-2, pages 179–210, 1999.

[Awasthi *et al.* 2017] Pranjal Awasthi, Maria-Florina Balcan and Konstantin Voevodski. *Local algorithms for interactive clustering.* Journal of Machine Learning Research, vol. 18, pages 3:1–3:35, 2017.

[Babaki *et al.* 2014] Behrouz Babaki, Tias Guns and Siegfried Nijssen. *Constrained Clustering using Column Generation.* In Proceedings of the 11th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pages 438–454, 2014.

[Bache & Lichman 2014] K. Bache and M. Lichman. *UCI Machine Learning Repository.* `http://archive.ics.uci.edu/ml`, 2014.

[Bae & Bailey 2006] Eric Bae and James Bailey. *COALA: A Novel Approach for the Extraction of an Alternate Clustering of High Quality and High Dissimilarity.* In Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China, pages 53–62, 2006.

[Banerjee & Ghosh 2006] Arindam Banerjee and Joydeep Ghosh. *Scalable Clustering Algorithms with Balancing Constraints.* Data Mining and Knowledge Discovery, vol. 13, no. 3, pages 365–395, 2006.

[Bansal *et al.* 2004] Nikhil Bansal, Avrim Blum and Shuchi Chawla. *Correlation Clustering.* Mach. Learn., vol. 56, no. 1-3, pages 89–113, June 2004.

[Bar-Hillel *et al.* 2005] Aharon Bar-Hillel, Tomer Hertz, Noam Shental and Daphna Weinshall. *Learning a Mahalanobis Metric from Equivalence Constraints.* Journal of Machine Learning Research, vol. 6, pages 937–965, 2005.

[Basu *et al.* 2002] S. Basu, A. Banerjee and R. Mooney. *Semi-supervised clustering by seeding.* In Proceedings of the International Conference on Machine Learning, pages 19–26, 2002.

[Basu *et al.* 2004a] Sugato Basu, A. Banjeree, ER. Mooney, Arindam Banerjee and Raymond J. Mooney. *Active Semi-Supervision for Pairwise Constrained Clustering.* In In Proceedings of the 2004 SIAM International Conference on Data Mining (SDM-04), pages 333–344, 2004.

[Basu *et al.* 2004b] Sugato Basu, Mikhail Bilenko and Raymond J. Mooney. *A Probabilistic Framework for Semi-Supervised Clustering.* In In Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-04), pages 59–68, 2004.

[Basu *et al.* 2008] Sugato Basu, Ian Davidson and Kiri Wagstaff. Constrained clustering: Advances in algorithms, theory, and applications. Chapman & Hall/CRC, 1 édition, 2008.

[Beldiceanu *et al.* 2014] Nicolas Beldiceanu, Mats Carlsson and Jean-Xavier Rampon. *Global Constraint Catalog.* SICS and EMN Technical Report, `http://sofdem.github.io/gccat/`, 2014.

[Berg & Jarvisalo 2013] J. Berg and M. Jarvisalo. *Optimal Correlation Clustering via MaxSAT.* In Proceedings of the 13th IEEE International Conference on Data Mining Workshops, pages 750–757, 2013.

[Berg & Järvisalo 2017] Jeremias Berg and Matti Järvisalo. *Cost-optimal constrained correlation clustering via weighted partial Maximum Satisfiability.* Artif. Intell., vol. 244, pages 110–142, 2017.

[Bérubé *et al.* 2009] Jean-François Bérubé, Michel Gendreau and Jean-Yves Potvin. *An exact epsilon-constraint method for bi-objective combinatorial optimization problems: Application to the Traveling Salesman Problem with Profits.* European Journal of Operational Research, vol. 194, no. 1, pages 39–50, 2009.

[Bessière & Régin 1996] Christian Bessière and Jean-Charles Régin. *MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems.* In Proceedings of the Second International Conference on Principles and Practice of Constraint Programming, Cambridge, Massachusetts, USA, August 19-22, 1996, pages 61–75, 1996.

[Bessiere *et al.* 2009] Christian Bessiere, Emmanuel Hebrard and Barry O'Sullivan. *Minimising Decision Tree Size as Combinatorial Optimisation.* In Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, pages 173–187, 2009.

[Bessiere 2006] Christian Bessiere. *Constraint Propagation.* In Handbook of Constraint Programming, pages 29–83. 2006.

[Bilenko *et al.* 2004] M. Bilenko, S. Basu and R. J. Mooney. *Integrating constraints and metric learning in semi-supervised clustering.* In Proceedings of the 21st International Conference on Machine Learning, pages 11–18, 2004.

[B.J. van Os 2004] J.J. Meulman B.J. van Os. *Improving Dynamic Programming Strategies for Partitioning.* Journal of Classification, 2004.

[Blache & Balfourier 2001] Philippe Blache and Jean-Marie Balfourier. *Property Grammars: a Flexible Constraint-Based Approach to Parsing.* In International Workshop on Parsing Techniques, Beijing, China, 2001.

[Blache & Rauzy 2006] Philippe Blache and Stéphane Rauzy. *Mécanismes de contrôle pour l'analyse en Grammaires de Propriétés.* In Actes, Traitement Automatique des Langues Naturelles (TALN), pages 415–424. P. Mertens, C. Fairon, A. Dister et P. Watrin eds., 2006.

[Blache 2000] Philippe Blache. *Constraints, Linguistic Theories and Natural Language Processing.* In Lecture Notes in Artificial Intelligence, Vol. 1835. Springer-Verlag, 2000.

[Boudjeloud-Assala *et al.* 2016] Lydia Boudjeloud-Assala, Philippe Pinheiro, Alexandre Blansché, Thomas Tamisier and Benoît Otjacques. *Interactive and iterative visual clustering.* Information Visualization, vol. 15, no. 3, pages 181–197, 2016.

[Bradley *et al.* 2000] P. Bradley, K. Bennett and A. Demiriz. *Constrained K-Means Clustering.* Rapport technique MSR-TR-2000-65, Microsoft Research, 2000.

[Brusco & Stahl 2005] Michael Brusco and Stephanie Stahl. Branch-and-Bound Applications in Combinatorial Data Analysis (Statistics and Computing). Springer, 1 édition, July 2005.

[Brusco 2003] M.J. Brusco. *An enhanced branch-and-bound algorithm for a partitioning problem.* British Journal of Mathematical and Statistical Psychology, pages 83–92, 2003.

[Brusco 2006] M.J. Brusco. *A repetitive branch-and-bound procedure for minimum within-cluster sum of squares partitioning.* Psychometrika, pages 347–363, 2006.

[Cambazard *et al.* 2010] Hadrien Cambazard, Tarik Hadzic and Barry O'Sullivan. *Knowledge Compilation for Itemset Mining.* In Proceedings of the 19th European Conference on Artificial Intelligence, pages 1109–1110, 2010.

[Caruana *et al.* 2006] Rich Caruana, Mohamed Farid Elhawary, Nam Nguyen and Casey Smith. *Meta Clustering.* In Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China, pages 107–118, 2006.

[Chabert & Solnon 2017] Maxime Chabert and Christine Solnon. *Constraint Programming for Multi-criteria Conceptual Clustering.* In Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, pages 460–476, 2017.

[Chen *et al.* 2015] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen and G. Batista. *The UCR Time Series Classification Archive*, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[Collette & Siarry 2002] Yann Collette and Patrick Siarry. Optimisation multiobjectif. Eyrollles, 2002.

[Colmerauer & Dao 2000] Alain Colmerauer and Thi-Bich-Hanh Dao. *Expressiveness of Full First Order Constraints in the Algebra of Finite or Infinite Trees.* In Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings, pages 172–186, 2000.

[Colmerauer & Dao 2003] Alain Colmerauer and Thi-Bich-Hanh Dao. *Expressiveness of Full First-Order Constraints in the Algebra of Finite or Infinite Trees.* Constraints, vol. 8, no. 3, pages 283–302, 2003.

[Colmerauer 1982] Alain Colmerauer. *Prolog and infinite trees.* Logic Programming, pages 231–251, 1982.

[Colmerauer 1984] Alain Colmerauer. *Equation and disequations on finite and infinite trees.* In Proceedings of the International Conference on the Fifth Generation of Computer Systems, pages 85–99, 1984.

[Colmerauer 1990] Alain Colmerauer. *An introduction to Prolog III.* Communication of the ACM, vol. 33, no. 7, pages 68–90, 1990.

[Cormack 1971] R. Cormack. *A review of classification.* Journal of the Royal Statistical Society. Series A (General), vol. 134, no. 3, pages 321–367, 1971.

[Courcelle 1983] B. Courcelle. *Fundamental Properties of Infinite Trees.* Theoretical Computer Science, vol. 25, no. 2, pages 95–169, 1983.

[Dahl & Blache 2004] Veronica Dahl and Philippe Blache. *Directly Executable Constraint Based Grammars.* In Actes des Journées Francophones de Programmation Logique et par Contraintes 2004 (JFPLC-04), Angers, France, 2004.

[Dang & Bailey 2010] Xuan Hong Dang and James Bailey. *Generation of Alternative Clusterings Using the CAMI Approach.* In SDM, volume 10, pages 118–129. SIAM, 2010.

[Dang & Bailey 2015] Xuan Hong Dang and James Bailey. *A framework to uncover multiple alternative clusterings.* Machine Learning, vol. 98, no. 1-2, pages 7–30, 2015.

[Dao & Djelloul 2006] Thi-Bich-Hanh Dao and Khalil Djelloul. *Solving First-Order Constraints in the Theory of the Evaluated Trees.* In Logic Programming, 22nd International Conference, ICLP 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings, pages 423–424, 2006.

[Dao *et al.* 2002] Thi-Bich-Hanh Dao, Arnaud Lallouet, Andrei Legtchenko and Lionel Martin. *Indexical-Based Solver Learning.* In Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings, pages 541–555, 2002.

[Dao *et al.* 2013a] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *A Declarative Framework for Constrained Clustering.* In Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, pages 419–434, 2013.

[Dao *et al.* 2013b] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *A Filtering Algorithm for Constrained Clustering with Within-Cluster Sum of Dissimilarities Criterion.* In Proceedings of the 25th International Conference on Tools with Artificial Intelligence, pages 1060–1067, 2013.

[Dao *et al.* 2013c] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Un modèle général pour la classification non supervisée sous contraintes utilisateur.* In Neuvième Journées Francophones de Programmation par Contraintes, 2013.

[Dao *et al.* 2013d] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Une approche en programmation par contraintes pour la classification non supervisée.* In Extraction et gestion des connaissances (EGC'2013), Actes, 29 janvier - 01 février 2013, Toulouse, France, pages 55–66, 2013.

[Dao *et al.* 2014a] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Classification non supervisée mono et bi-objectif sous contraintes utilisateur par la programmation par contraintes.* In Dixième Journées Francophones de Programmation par Contraintes, 2014.

[Dao *et al.* 2014b] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Un nouveau modèle pour la classification non supervisée sous contraintes.* Revue d'Intelligence Artificielle, vol. 28, no. 5, pages 523–545, 2014.

[Dao *et al.* 2015a] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Constrained Minimum Sum of Squares Clustering by Constraint Programming.* In Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings, pages 557–573, 2015.

[Dao *et al.* 2015b] Thi-Bich-Hanh Dao, Willy Lesaint and Christel Vrain. *Clustering conceptuel et relationnel en programmation par contraintes.* In Onzième Journées Francophones de Programmation par Contraintes, 2015.

[Dao *et al.* 2016a] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Tias Guns and Christel Vrain. *Branch-and-bound répétitif et programmation par contraintes pour le clustering sous contraintes.* In Douzième Journées Francophones de Programmation par Contraintes, 2016.

[Dao *et al.* 2016b] Thi-Bich-Hanh Dao, Christel Vrain, Khanh-Chuong Duong and Ian Davidson. *A Framework for Actionable Clustering using Constraint Programming.* In Proceedings of the 22nd European Conference on Artificial Intelligence, pages 453–461, 2016.

[Dao *et al.* 2017] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Constrained Clustering by Constraint Programming.* Artificial Intelligence, vol. 244, pages 70–94, 2017.

[Dao 2000a] Thi-Bich-Hanh Dao. *Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis.* In Actes des Journées Francophones de Programmation Logique et par Contraintes 2000 (JFPLC-00), Hermes, pages 225–240, Marseille, France, 2000.

[Dao 2000b] Thi-Bich-Hanh Dao. *Résolution de contraintes du premier ordre dans la théorie des arbres finis ou infinis.* Ph.D. Thesis, Université Aix-Marseille II, 2000.

[Dao 2009] Thi-Bich-Hanh Dao. *Un algorithme de décision dans l'algèbre des arbres finis ou infinis et des queues.* In Cinquième Journées Francophones de Programmation par Contraintes, 2009.

[Davidson & Basu 2007] Ian Davidson and Sugato Basu. *A survey of clustering with instance level constraints.* ACM Transactions on Knowledge Discovery from Data, vol. 77, no. 1, pages 1–41, 2007.

[Davidson & Qi 2008] Ian Davidson and Zijie Qi. *Finding alternative clusterings using constraints.* In Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, pages 773–778. IEEE, 2008.

[Davidson & Ravi 2005] Ian Davidson and S. S. Ravi. *Clustering with Constraints: Feasibility Issues and the k-Means Algorithm.* In Proceedings of the 5th SIAM International Conference on Data Mining, pages 138–149, 2005.

[Davidson & Ravi 2007] Ian Davidson and S. S. Ravi. *The Complexity of Non-hierarchical Clustering with Instance and Cluster Level Constraints.* Data Mining Knowledge Discovery, vol. 14, no. 1, pages 25–61, 2007.

[Davidson *et al.* 2006] Ian Davidson, Kiri L. Wagstaff and Sugato Basu. *Measuring Constraint-Set Utility for Partitional Clustering Algoirthms.* In PKDD, pages 115–126, 2006.

[Davidson *et al.* 2010] Ian Davidson, S. S. Ravi and Leonid Shamis. *A SAT-based Framework for Efficient Constrained Clustering.* In Proceedings of the 10th SIAM International Conference on Data Mining, pages 94–105, 2010.

[Davidson *et al.* 2013] Ian Davidson, Buyue Qian, Xiang Wang and Jieping Ye. *Multi-objective Multi-view Spectral Clustering via Pareto Optimization.* In Proceedings of the 13th SIAM International Conference on Data Mining, pages 234–242, 2013.

[De Raedt *et al.* 2008] Luc De Raedt, Tias Guns and Siegfried Nijssen. *Constraint programming for itemset mining.* In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 204–212, 2008.

[De Raedt *et al.* 2010] L. De Raedt, T. Guns and S. Nijssen. *Constraint Programming for Data Mining and Machine Learning.* In Proc. of the 24th AAAI Conference on Artificial Intelligence, 2010.

[Debusmann *et al.* 2004] Ralph Debusmann, Denys Duchier, Alexander Koller, Marco Kuhlmann, Gert Smolka and Stefan Thater. *A Relational Syntax-Semantics Interface Based on Dependency Grammar.* In COLING 2004, 20th International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2004, Geneva, Switzerland, 2004.

[Delattre & Hansen 1980] Michel Delattre and Pierre Hansen. *Bicriterion Cluster Analysis.* IEEE Transactions on Pattern Analysis and Machine Intelligence, no. 4, pages 277–291, 1980.

[Demiriz *et al.* 1999] A. Demiriz, K. Bennett and M. Embrechts. *Semi-supervised clustering using genetic algorithms.* In Proceedings of the Conference on Artificial Neural Networks in Engineering, pages 809–814, 1999.

[Demiriz *et al.* 2008] Ayhan Demiriz, Kristin P Bennett and Paul S Bradley. *Using assignment constraints to avoid empty clusters in k-means clustering.* Constrained Clustering: Advances in Algorithms, Theory, and Applications, 2008.

[Diaz & Codognet 2001] Daniel Diaz and Philippe Codognet. *Design and Implementation of the Gnu-Prolog System.* Journal of Functional and Logic Programming, vol. 2001, no. 6, 2001.

[Djelloul & Dao 2006a] Khalil Djelloul and Thi-Bich-Hanh Dao. *Complete First-Order Axiomatization of Finite or Infinite M-extended Trees.* In 20th Workshop on Logic Programming, Vienna, Austria, February 22–24, 2006, pages 111–119, 2006.

[Djelloul & Dao 2006b] Khalil Djelloul and Thi-Bich-Hanh Dao. *Extension of First-Order Theories into Trees.* In Artificial Intelligence and Symbolic Computation, 8th International Conference, AISC 2006, Beijing, China, September 20-22, 2006, Proceedings, pages 53–67, 2006.

[Djelloul & Dao 2006c] Khalil Djelloul and Thi-Bich-Hanh Dao. *Solving first-order constraints in the theory of finite or infinite trees: introduction to the decomposable theories.* In Proceedings of the 2006 ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006, pages 7–14, 2006.

[Djelloul *et al.* 2007] Khalil Djelloul, Thi-Bich-Hanh Dao and Thom W. Frühwirth. *Toward a first-order extension of Prolog's unification using CHR: a CHR first-order constraint solver over finite or infinite trees.* In Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15, 2007, pages 58–64, 2007.

[Djelloul *et al.* 2008] Khalil Djelloul, Thi-Bich-Hanh Dao and Thom W. Frühwirth. *Theory of finite or infinite trees revisited.* TPLP, vol. 8, no. 4, pages 431–489, 2008.

[du Merle *et al.* 1999] O. du Merle, P. Hansen, B. Jaumard and N. Mladenovic. *An Interior Point Algorithm for Minimum Sum-of-Squares Clustering.* SIAM Journal on Scientific Computing, vol. 21, no. 4, pages 1485–1505, 1999.

[Duchier & Debusmann 2001] Denys Duchier and Ralph Debusmann. *Topological Dependency Trees: A Constraint-Based Account of Linear Precedence.* In Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference, July 9-11, 2001, Toulouse, France., pages 180–187, 2001.

[Duchier *et al.* 2009] Denys Duchier, Jean-Philippe Prost and Thi-Bich-Hanh Dao. *A Model-Theoretic Framework for Grammaticality Judgements.* In Formal Grammar - 14th International Conference, FG 2009, Bordeaux, France, July 25-26, 2009, Revised Selected Papers, pages 17–30, 2009.

[Duchier *et al.* 2010a] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier and Willy Lesaint. *Property Grammar Parsing Seen as a Constraint Optimization Problem.* In Formal Grammar - 15th and 16th International Conferences, FG 2010, Copenhagen, Denmark, August 2010, FG 2011, Ljubljana, Slovenia, August 2011, Revised Selected Papers, pages 82–96, 2010.

[Duchier *et al.* 2010b] Denys Duchier, Thi-Bich-Hanh Dao, Yannick Parmentier and Willy Lesaint. *Une modélisation en CSP des grammaires de propriétés.* In Sixième Journées Francophones de Programmation par Contraintes, 2010.

[Duchier *et al.* 2011] Denys Duchier, Thi-Bich-Hanh Dao and Yannick Parmentier. *Model-Theory of Property Grammars with Features.* In Proceedings of the 12th International Conference on Parsing Technologies, IWPT 2011, October 5-7, 2011, Dublin City University, Dubin, Ireland, pages 75–79, 2011.

[Duchier *et al.* 2012] Denys Duchier, Thi-Bich-Hanh Dao and Yannick Parmentier. *Analyse syntaxique par contraintes pour les grammaires de propriétés.* In Huitième Journées Francophones de Programmation par Contraintes, 2012.

[Duchier *et al.* 2014] Denys Duchier, Thi-Bich-Hanh Dao and Yannick Parmentier. *Model-theory and implementation of property grammars with features.* Journal of Logic and Computation, vol. 24, no. 2, pages 491–509, 2014.

[Duchier 2003] Denys Duchier. *Configuration of labeled trees under lexicalized constraints and principles.* Journal of Research on Language and Computation, vol. 1, no. 3/4, pages 307–336, 2003.

[Duong 2014] Khanh-Chuong Duong. *Constrained Clustering by Constraint Programming.* Ph.D. Thesis, Université d'Orléans, 2014.

[Ed-Dbali *et al.* 2003] AbdelAli Ed-Dbali, Thi-Bich-Hanh Dao, Arnaud Lallouet and Andrei Legtchenko. *Apprentissage de solveurs de contraintes sur les domaines finis.* Technique et Science Informatiques, vol. 22, no. 1, pages 125–138, 2003.

[Ester *et al.* 1996] Martin Ester, Hans P. Kriegel, Jorg Sander and Xiaowei Xu. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.* In Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pages 226–231, 1996.

[Estratat & Henocque 2004] Mathieu Estratat and Laurent Henocque. *Parsing languages with a configurator.* In Proceedings of the European Conference for Artificial Intelligence ECAI'2004, pages 591–595, Valencia, Spain, August 2004.

[Fisher 1987] DouglasH. Fisher. *Knowledge Acquisition Via Incremental Conceptual Clustering.* Machine Learning, vol. 2, no. 2, pages 139–172, 1987.

[Focacci *et al.* 1999] F. Focacci, A. Lodi and M. Milano. *Cost-Based Domain Filtering.* In Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming, pages 189–203, 1999.

[Forestier *et al.* 2010] G. Forestier, P. Gançarski and C. Wemmert. *Collaborative clustering with background knowledge.* Data & Knowledge Engineering, vol. 69, no. 2, pages 211–228, 2010.

[Fred & Jain 2003] Ana L. N. Fred and Anil K. Jain. *Robust Data Clustering.* In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings, 2003.

[Friston 2011] Karl J Friston. *Functional and effective connectivity: a review.* Brain connectivity, vol. 1, no. 1, pages 13–36, 2011.

[Frühwirth 2009] Thom Frühwirth. Constraint Handling Rules. Cambridge University Press, 2009. ISBN 9780521877763.

[Ganji *et al.* 2016] M. Ganji, J. Bailey and P. Stuckey. *Lagrangian Constrained Clustering.* In Proceedings of the SIAM International Conference on Data Mining, pages 288–296, 2016.

[Ganji *et al.* 2017] Mohadeseh Ganji, James Bailey and Peter J. Stuckey. *A Declarative Approach to Constrained Community Detection.* In Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings, pages 477–494, 2017.

[Ge *et al.* 2007] R. Ge, M. Ester, W. Jin and I. Davidson. *Constraint-driven clustering.* In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 320–329, 2007.

[Gilpin & Davidson 2011] Sean Gilpin and Ian N. Davidson. *Incorporating SAT solvers into hierarchical clustering algorithms: an efficient and flexible approach.* In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1136–1144, 2011.

[Gilpin *et al.* 2013] Sean Gilpin, Siegfried Nijssen and Ian N. Davidson. *Formalizing Hierarchical Clustering as Integer Linear Programming.* In Proceedings of the 27th AAAI Conference on Artificial Intelligence, pages 372–378, 2013.

[Giotis & Guruswami 2006] Ioannis Giotis and Venkatesan Guruswami. *Correlation Clustering with a Fixed Number of Clusters.* Theory of Computing, vol. 2, no. 1, pages 249–266, 2006.

[Gonzalez 1985] T. Gonzalez. *Clustering to minimize the maximum intercluster distance.* Theoretical Computer Science, vol. 38, pages 293–306, 1985.

[Guns *et al.* 2011] Tias Guns, Siegfried Nijssen and Luc De Raedt. *Itemset mining: A constraint programming perspective.* Artificial Intelligence, vol. 175, pages 1951–1983, 2011.

[Guns *et al.* 2013] Tias Guns, Siegfried Nijssen and Luc De Raedt. *k-Pattern set mining under constraints.* IEEE Transactions on Knowledge and Data Engineering, vol. 25, no. 2, pages 402–418, 2013.

[Guns *et al.* 2016] Tias Guns, Thi-Bich-Hanh Dao, Christel Vrain and Khanh-Chuong Duong. *Repetitive Branch-and-Bound using Constraint Programming for Constrained Minimum Sum-of-Squares Clustering.* In Proceedings of the 22nd European Conference on Artificial Intelligence, pages 462–470, 2016.

[Guns *et al.* 2017] Tias Guns, Anton Dries, Siegfried Nijssen, Guido Tack and Luc De Raedt. *MiningZinc: A declarative framework for constraint-based mining.* Artif. Intell., vol. 244, pages 6–29, 2017.

[Haimes *et al.* 1971] Yacov Y. Haimes, Leon S. Lasdon and David A. Wismer. *On a Bicriterion Formulation of the Problems of Integrated System Identification and System Optimization.* IEEE Transactions on Systems, Man, and Cybernetics, no. 1, pages 296–297, 1971.

[Han *et al.* 2006] Jiawei Han, Micheline Kamber and Jian Pei. Data Mining: Concepts and Techniques, Second Edition (The Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann, 2 édition, January 2006.

[Hansen & Delattre 1978] Pierre Hansen and Michel Delattre. *Complete-Link Cluster Analysis by Graph Coloring.* Journal of the American Statistical Association, vol. 73, no. 362, pages 397–403, 1978.

[Hansen & Jaumard 1997] Pierre Hansen and Brigitte Jaumard. *Cluster Analysis and Mathematical Programming.* Mathematical Programming, vol. 79, no. 1-3, pages 191–215, 1997.

[Haralick & Elliott 1980] Robert M. Haralick and Gordon L. Elliott. *Increasing Tree Search Efficiency for Constraint Satisfaction Problems.* Artif. Intell., vol. 14, no. 3, pages 263–313, 1980.

[Hebrard *et al.* 2010] Emmanuel Hebrard, Eoin O'Mahony and Barry O'Sullivan. *Constraint Programming and Combinatorial Optimisation in Numberjack.* In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 7th International Conference, CPAIOR 2010, pages 181–185, 2010.

[Hiep *et al.* 2016] T. Hiep, N. Duc and B. Trung. *Local Search Approach For The Pairwise Constrained Clustering Problem.* In Proceedings of the Symposium on Information and Communication Technology, pages 115–122, 2016.

[Hubert & Arabie 1985] L. Hubert and P. Arabie. *Comparing partitions.* Journal of classification, vol. 2, no. 1, pages 193–218, 1985.

[Huet 1976] Gérard Huet. Résolution d'équations dans les langages d'ordre $1, 2, \ldots, \omega$. Thèse d'Etat, Université Paris 7, 1976.

[Jabbour *et al.* 2013] Saïd Jabbour, Lakhdar Sais and Yakoub Salhi. *The Top-k Frequent Closed Itemset Mining Using Top-k SAT Problem.* In Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, pages 403–418, 2013.

[Jabbour *et al.* 2017] Saïd Jabbour, Nizar Mhadhbi, Badran Raddaoui and Lakhdar Sais. *A SAT-Based Framework for Overlapping Community Detection in Networks.* In Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part II, pages 786–798, 2017.

[Jaffar 1984] J. Jaffar. *Efficient unification over infinite terms.* New Generation Computing, vol. 2, no. 3, pages 207–219, 1984.

[Järvisalo 2011] Matti Järvisalo. *Itemset Mining as a Challenge Application for Answer Set Enumeration.* In Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings, pages 304–310, 2011.

[Jensen 1969] Robert E. Jensen. *A dynamic programming algorithm for cluster analysis.* Journal of the Operations Research Society of America, vol. 7, pages 1034–1057, 1969.

[Johnson 1967] StephenC. Johnson. *Hierarchical clustering schemes.* Psychometrika, vol. 32, no. 3, pages 241–254, 1967.

[Kamvar *et al.* 2003] S. Kamvar, D. Klein and C. Manning. *Spectral Learning.* In Proceedings of the International Joint Conference on Artificial Intelligence, pages 561–566, 2003.

[Keogh & Lin 2005] E. Keogh and J. Lin. *Clustering of time-series subsequences is meaningless: implications for previous and future research.* Knowledge and Information Systems, vol. 8, no. 2, pages 154–177, 2005.

[Klein & Aronson 1991] Gary Klein and Jay E. Aronson. *Optimal clustering: A model and method.* Naval Research Logistics, vol. 38, no. 3, pages 447–461, 1991.

[Klein *et al.* 2002] Dan Klein, Sepandar D. Kamvar and Christopher D. Manning. *From Instance-level Constraints to Space-Level Constraints: Making the Most of Prior Knowledge in Data Clustering.* In Proceedings of the 19th International Conference on Machine Learning, pages 307–314, 2002.

[Kontonasios & Bie 2015] Kleanthis-Nikolaos Kontonasios and Tijl De Bie. *Subjectively interesting alternative clusterings.* Machine Learning, vol. 98, no. 1-2, pages 31–56, 2015.

[Koontz *et al.* 1975] W. L. G. Koontz, P. M. Narendra and K. Fukunaga. *A Branch and Bound Clustering Algorithm.* IEEE Trans. Comput., vol. 24, no. 9, pages 908–915, 1975.

[Kulis *et al.* 2005] Brian Kulis, Sugato Basu, Inderjit Dhillon and Raymond Mooney. *Semi-supervised graph clustering: a kernel approach.* In ICML '05: Proceedings of the 22nd international conference on Machine learning, pages 457–464, New York, NY, USA, 2005. ACM.

[Kuo *et al.* 2017] Chia-Tung Kuo, S. S. Ravi, Thi-Bich-Hanh Dao, Christel Vrain and Ian Davidson. *A Framework for Minimal Clustering Modification via Constraint Programming.* In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA., pages 1389–1395, 2017.

[Lallouet *et al.* 2003a] Arnaud Lallouet, Thi-Bich-Hanh Dao, Andrei Legtchenko and AbdelAli Ed-Dbali. *Finite Domain Constraint Solver Learning.* In IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003, pages 1379–1380, 2003.

[Lallouet *et al.* 2003b] Arnaud Lallouet, Andrei Legtchenko, Thi-Bich-Hanh Dao and AbdelAli Ed-Dbali. *Intermediate (Learned) Consistencies.* In Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings, pages 889–893, 2003.

[Law & Lee 2004] Yat Chiu Law and Jimmy Ho-Man Lee. *Global Constraints for Integer and Set Value Precedence.* In Mark Wallace, editeur, Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming, pages 362–376, 2004.

[Leskovec & Krevl 2014] Jure Leskovec and Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. `http://snap.stanford.edu/data`, June 2014.

[Li & Liu 2009] Z. Li and J. Liu. *Constrained Clustering by Spectral Kernel Learning*. In IEEE International Conference on Computer Vision, pages 421–427, 2009.

[Luxburg 2007] Ulrike Luxburg. *A Tutorial on Spectral Clustering*. Statistics and Computing, vol. 17, no. 4, pages 395–416, 2007.

[Mackworth 1977] Alan K. Mackworth. *Consistency in Networks of Relations*. Artif. Intell., vol. 8, no. 1, pages 99–118, 1977.

[Maher 1988] Michael Maher. *Complete axiomatization of the algebra of finite, rational and infinite trees*. Rapport technique, IBM T. J. Watson Research Center, 1988.

[Mehrotra & Trick 1995] Anuj Mehrotra and Michael A. Trick. *A Column Generation Approach For Graph Coloring*. INFORMS Journal on Computing, vol. 8, pages 344–354, 1995.

[Métivier *et al.* 2012a] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari and Samir Loudni. *Constrained Clustering Using SAT*. In Proceedings of the 11th International Symposium on Advances in Intelligent Data Analysis, pages 207–218, 2012.

[Métivier *et al.* 2012b] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari and Samir Loudni. *Constrained Clustering Using SAT*. In Advances in Intelligent Data Analysis XI - 11th International Symposium, IDA 2012, Helsinki, Finland, October 25-27, 2012. Proceedings, pages 207–218, 2012.

[Miettinen 1998] Kaisa Miettinen. Nonlinear Multiobjective Optimization. Kluwer Academic Publishers, 1998.

[Milano & Hentenryck 2011] Michela Milano and Pascal Van Hentenryck. Hybrid optimization - the ten years of cpaior, volume 45 of *Springer Optimization and its Applications*. Springer, 2011.

[Mueller & Kramer 2010] Marianne Mueller and Stefan Kramer. *Integer Linear Programming Models for Constrained Clustering*. In Proceedings of the 13th International Conference on Discovery Science, pages 159–173, 2010.

[Ng 2000] M. Ng. *A Note on Constrained K-Means Algorithms*. Pattern Recognition, vol. 33, no. 3, pages 515–519, 2000.

[Ouali *et al.* 2016] A. Ouali, S. Loudni, Y. Lebbah, P. Boizumault, A. Zimmermann and L. Loukil. *Efficiently finding conceptual clustering models with integer linear programming*. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 647–654, 2016.

[Pelleg & Baras 2007] Dan Pelleg and Dorit Baras. *K-Means with Large and Noisy Constraint Sets*. In Machine Learning: ECML 2007, volume 4701 of *Lecture Notes in Computer Science*, pages 674–682. Springer Berlin Heidelberg, 2007.

[Prost 2008] Jean-Philippe Prost. *Modelling Syntactic Gradience with Loose Constraint-based Parsing*. Cotutelle Ph.D. Thesis, Macquarie University, Sydney, Australia, and Université de Provence, Aix-en-Provence, France, 2008.

[Pullum & Scholz 2001] G. Pullum and B. Scholz. *On the distinction between model-theoretic and generative-enumerative syntactic frameworks*. In 4th International Conference on Logical Aspects of Computational Linguistics, pages 17–43. Springer Verlag, 2001.

[Puolamäki *et al.* 2016] Kai Puolamäki, Bo Kang, Jefrey Lijffijt and Tijl De Bie. *Interactive Visual Data Exploration with Subjective Feedback*. In Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part II, pages 214–229, 2016.

[Qi & Davidson 2009] ZiJie Qi and Ian Davidson. *A principled and flexible framework for finding alternative clusterings*. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 717–726. ACM, 2009.

[Quimper *et al.* 2004] Claude-Guy Quimper, Alejandro López-Ortiz, Peter van Beek and Alexander Golynski. *Improved Algorithms for the Global Cardinality Constraint*. In Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, Proceedings, pages 542–556, 2004.

[Ramachandran & Van Hentenryck 1993] V. Ramachandran and P. Van Hentenryck. *Incremental algorithm for constraint solving and entailment over rational trees*. In Proceedings of the 13rd Conference on Foundations of Software Technology and Theoretical Computer Science, pages 205–217, 1993.

[Rand 1971] William M. Rand. *Objective Criteria for the Evaluation of Clustering Methods*. Journal of the American Statistical Association, vol. 66, no. 336, pages 846–850, 1971.

[Régin 1994] Jean-Charles Régin. *A Filtering Algorithm for Constraints of Difference in CSPs*. In Proceedings of the 12th National Conference on Artificial Intelligence (Vol. 1), pages 362–367, 1994.

[Régin 1996] Jean-Charles Régin. *Generalized Arc Consistency for Global Cardinality Constraint*. In Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1, AAAI'96, pages 209–215, 1996.

[Régin 1999] Jean-Charles Régin. *Arc Consistency for Global Cardinality Constraints with Costs*. In Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming, pages 390–404, 1999.

[Robinson 1965] J.A. Robinson. *A machine-oriented logic based on the resolution principle*. JACM, vol. 12, no. 1, pages 23–41, 1965.

[Rojas *et al.* 2014] Willy Ugarte Rojas, Patrice Boizumault, Samir Loudni, Bruno Crémilleux and Alban Lepailleur. *Mining (Soft-) Skypatterns Using Dynamic CSP*. In Proceedings of the 11th International Conference on Integration of AI and OR Techniques in Constraint Programming, pages 71–87, 2014.

[Rossi *et al.* 2006] Francesca Rossi, Peter van Beek and Toby Walsh, editeurs. Handbook of Constraint Programming. Foundations of Artificial Intelligence. Elsevier B.V., Amsterdam, Netherlands, August 2006.

[Sakoe & Chiba 1971] H. Sakoe and S. Chiba. *A dynamic programming approach to continuous speech recognition.* In Proceedings of the International Congress on Acoustics, volume 3, pages 65–69, 1971.

[Sakoe & Chiba 1978] H. Sakoe and S. Chiba. *Dynamic programming algorithm optimization for spoken word recognition.* IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 26, no. 1, pages 43–49, 1978.

[Smith 2006] Barbara M. Smith. *Modelling.* In Handbook of Constraint Programming, pages 377–406. 2006.

[Steinley 2006] Douglas Steinley. *k-means clustering: A half-century synthesis.* British Journal of Mathematical and Statistical Psychology, vol. 59, no. 1, pages 1–34, 2006.

[Truong & Battiti 2015] Duy Tin Truong and Roberto Battiti. *A flexible cluster-oriented alternative clustering algorithm for choosing from the Pareto front of solutions.* Machine Learning, vol. 98, no. 1-2, pages 57–91, 2015.

[Tung *et al.* 2001] Anthony K. H. Tung, Jiawei Han, Laks V.S. Lakshmanan and Raymond T. Ng. Constraint-based clustering in large databases, pages 405–419. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[van Beek 2006] Peter van Beek. *Backtracking Search Algorithms.* In Handbook of Constraint Programming, pages 85–134. 2006.

[van Hentenryck *et al.* 1991] P. van Hentenryck, V. Saraswat and Y. Deville. *Constraint processing in cc(FD).* draft, 1991.

[Van Hentenryck 1989] Pascal Van Hentenryck. Constraint satisfaction in logic programming. The MIT Press, 1989.

[van Hoeve & Katriel 2006] Willem-Jan van Hoeve and Irit Katriel. *Global Constraints.* In Handbook of Constraint Programming, pages 169–208. 2006.

[van Leeuwen 2014] Matthijs van Leeuwen. *Interactive Data Exploration Using Pattern Mining.* In Interactive Knowledge Discovery and Data Mining in Biomedical Informatics - State-of-the-Art and Future Challenges, pages 169–182. 2014.

[van Rullen *et al.* 2006] Tristan van Rullen, Philippe Blache and Jean-Marie Balfourier. *Constraint-Based Parsing as an Efficient Solution: Results from the Parsing Evaluation Campaign EASy.* In Proceedings of the Language and Resources Evaluation Conference, Genoa, Italy, 2006.

[van Rullen 2005] Tristan van Rullen. *Vers une analyse syntaxique à granularité variable.* PhD thesis, Université de Provence, Aix-Marseille 1, France, 2005.

[Verfaillie *et al.* 1996] Gérard Verfaillie, Michel Lemaître and Thomas Schiex. *Russian Doll Search for Solving Constraint Optimization Problems.* In Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, pages 181–187, 1996.

[Vorobyov 1996] Sergei Vorobyov. *An Improved Lower Bound for the Elementary Theory of Trees.* In Proceedings of the 13th International Conference on Automated Deduction CADE 96, pages 275–287, 1996.

[Wagstaff & Cardie 2000] K. Wagstaff and C. Cardie. *Clustering with instance-level constraints.* In Proceedings of the 17th International Conference on Machine Learning, pages 1103–1110, 2000.

[Wagstaff *et al.* 2001] Kiri Wagstaff, Claire Cardie, Seth Rogers and Stefan Schrödl. *Constrained K-means Clustering with Background Knowledge.* In Proceedings of the 18th International Conference on Machine Learning, pages 577–584, 2001.

[Wang & Chen 2012] Jiabing Wang and Jiaye Chen. *Clustering to Maximize the Ratio of Split to Diameter.* In Proceedings of the 29th International Conference on Machine Learning, 2012.

[Wang & Davidson 2010] Xiang Wang and Ian Davidson. *Flexible constrained spectral clustering.* In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 563–572, 2010.

[Wang *et al.* 1996] Y. Wang, H. Yan and C. Sriskandarajah. *The weighted sum of split and diameter clustering.* Journal of Classification, vol. 13, no. 2, pages 231–248, 1996.

[Wang *et al.* 2014] Xiang Wang, Buyue Qian and Ian Davidson. *On constrained spectral clustering and its applications.* Data Mining and Knowledge Discovery, vol. 28, no. 1, pages 1–30, 2014.

[Xia & Peng 2005] Y. Xia and J. Peng. *A cutting algorithm for the minimum sum-of-squared error clustering.* In SDM, 2005.

[Xing *et al.* 2003] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan and Stuart Russell. *Distance Metric Learning, With Application To Clustering With Side-Information.* In ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 15, pages 505–512. MIT Press, 2003.

[Zhi *et al.* 2013] Weifeng Zhi, Xiang Wang, Buyue Qian, Patrick Butler, Naren Ramakrishnan and Ian Davidson. *Clustering with Complex Constraints - Algorithms and Applications.* In Proceedings of the 27th AAAI Conference on Artificial Intelligence, 2013.

Résumé

Ce manuscrit d'*Habilitation à Diriger des Recherches* présente mes travaux sur l'application de la programmation par contraintes au traitement automatique des langues et à la fouille de données. En traitement automatique des langues, nous nous intéressons à l'analyse syntaxique des grammaires de propriété, décrites par des propriétés que doivent satisfaire les énoncés grammaticaux. Nous définissons une sémantique formelle en théorie des modèles et formalisons l'analyse syntaxique comme un problème d'optimisation sous contraintes. Nous développons un modèle en programmation par contraintes, ce qui amène à un analyseur entièrement à base de contraintes. En fouille de données, nous considérons le clustering sous contraintes, qui vise à partitionner les objets en groupes homogènes, étant donnés une mesure de dissimilarité entre objets et un ensemble de contraintes utilisateur à satisfaire. Nous développons un cadre déclaratif qui intègre plusieurs critères d'optimisation principaux de clustering et tous les types de contraintes utilisateur populaires. Nous montrons que sa flexibilité permet de trouver la frontière de Pareto pour des problèmes de clustering bi-objectif sous contraintes. Nous améliorons davantage l'efficacité de l'approche en développant des contraintes globales dédiées aux critères d'optimisation de clustering. Nous explorons plusieurs nouveaux problèmes de clustering avec des contraintes et montre que la programmation par contraintes constitue un cadre flexible et efficace pour les résoudre.

Mots-clefs : problème d'optimisation sous contraintes, programmation par contraintes, analyse syntaxique, clustering sous contraintes, contrainte globale d'optimisation.

Abstract

This manuscript of *Habilitation à Diriger des Recherches* presents my work on the application of constraint programming to natural language processing and to data mining. In natural language processing, we are interested in syntactic analysis of property grammars, defined by constraints that must be satisfied by the grammatical utterances. We introduce model-theoretic semantics and formulate the syntactic analysis as a constraint optimization problem. We develop a model using constraint programming, which leads to a fully constraint-based parser. In data mining, we consider constrained clustering problems that aim at partitioning the objects in homogeneous clusters, given a dissimilarity measure between objects and a set of user-constraints to be satisfied. We develop a declarative framework that integrates several principal clustering optimization criteria and all popular types of user-constraints. We show that the flexibility of the framework allows to find the complete Pareto front of bi-objective constrained clustering problems. We enhance further the approach by developing specific global optimization constraints for principal clustering optimization criteria. We explore several new clustering problems with constraints and show that constraint programming offers a general and efficient framework to solve them.

Keywords: constraint optimization problem, constraint programming, parsing, constrained clustering, global optimization constraint.