Constrained clustering by constraint programming

Khanh-Chuong Duong

UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE
MATHEMATIQUES, INFORMATIQUE, PHYSIQUE
THEORIQUE ET INGENIERIE DES SYSTEMES

Laboratoire d'Informatique Fondamentale d'Orléans

THÈSE présentée par :

Khanh-Chuong DUONG

soutenue le : **10 décembre 2014**

pour obtenir le grade de : **Docteur de l'Université d'Orléans**

Discipline/ Spécialité : Informatique

Constrained Clustering by Constraint Programming

THÈSE dirigée par :

Christel Vrain	Professeur, Université d'Orléans
Thi-Bich-Hanh Dao	Maître de Conférences, Université d'Orléans

RAPPORTEURS :

Bruno Crémilleux	Professeur, Université de Caen Basse-Normandie
Luc De Raedt	Professeur, Katholieke Universiteit Leuven, Belgium
Christine Solnon	Professeur, INSA de Lyon

JURY :

Bruno Crémilleux	Professeur, Université de Caen Basse-Normandie
Thi-Bich-Hanh Dao	Maître de Conférences, Université d'Orléans
Yves Deville	Professeur, Université catholique de Louvain, Belgium
Luc De Raedt	Professeur, Katholieke Universiteit Leuven, Belgium
Lakhdar Sais	Professeur, Université d'Artois
Christine Solnon	Professeur, INSA de Lyon
Christel Vrain	Professeur, Université d'Orléans

Résumé

La classification non supervisée, souvent appelée par le terme anglais de *clustering*, est une tâche importante en Fouille de Données, pour lesquelles de nombreuses approches ont été développées. Depuis une dizaine d'années, la classification non supervisée a été étendue pour intégrer des contraintes utilisateur permettant de modéliser des connaissances préalables dans le processus de clustering. Différents types de contraintes utilisateur peuvent être considérés, des contraintes pouvant porter soit sur les clusters, soit sur les instances. Par exemple, des contraintes peuvent exprimer des bornes sur la taille ou le diamètre des clusters. Des contraintes sur des instances expriment que deux objets doivent ou ne doivent pas être dans le même cluster (contraintes *must-link* et *cannot-link*).

Dans cette thèse, nous étudions le cadre de la Programmation par Contraintes (PPC) pour modéliser les tâches de clustering sous contraintes utilisateur. Des avancés récentes en PPC ont rendu ce paradigme puissant pour résoudre des problèmes combinatoires. Les principes de la PPC sont : (1) le programmeur spécifie le problème d'une façon déclarative par un Problème de Satisfaction de Contraintes ; (2) le solveur recherche la (les) solution(s) en intégrant la propagation de contraintes à l'exploration de l'espace de recherche. Utiliser la PPC a deux avantages principaux : la déclarativité, qui permet d'intégrer aisément des contraintes utilisateur et la capacité de trouver une solution optimale qui satisfait toutes les contraintes (s'il en existe).

Nous proposons deux modèles basés sur la PPC pour le clustering sous contraintes utilisateur. Les modèles sont généraux et flexibles, ils permettent d'intégrer des contraintes d'instances *must-link* et *cannot-link* et différents types de contraintes sur les clusters. Ils offrent également à l'utilisateur le choix entre différents critères d'optimisation. Afin d'améliorer l'efficacité, divers aspects sont étudiés. Les expérimentations sur des bases de données classiques et variées montrent qu'ils sont compétitifs par rapport aux approches exactes existantes. Nous montrons que nos modèles peuvent être intégrés dans une procédure plus générale et nous l'illustrons par la recherche de la frontière de Pareto dans un problème de clustering bi-critère sous contraintes utilisateur. A notre meilleure connaissance, notre approche est la première pour les problèmes de classification non supervisée sous contraintes utilisateur, qui intègrent le critère de diamètre des clusters, de la marge entre clusters, de la somme des dissimilarités intra-cluster et du bi-critère marge-diamètre, en présence de contrainte utilisateur.

Mots clés : classification non supervisée, contraintes utilisateur, Programmation par Contraintes, clustering bi-critère.

Abstract

Cluster analysis is an important task in Data Mining with hundreds of different approaches in the literature. Since the last decade, the cluster analysis has been extended to constrained clustering, also called semi-supervised clustering, so as to integrate previous knowledge on data to clustering algorithms. Several kinds of constraints can be considered; they may be put on the clusters, as for instance their sizes, or their diameters, or on instances the expert knows that they must be or cannot be in the same cluster (must-link or cannot-link constraints).

In this dissertation, we explore Constraint Programming (CP) for solving the task of constrained clustering. Among generic optimization tools, CP is a powerful paradigm for solving combinatorial search problems. The main principles in CP are: (1) users specify declaratively the problem in a Constraint Satisfaction Problem; (2) solvers search for solutions by constraint propagation and search. Relying on CP has two main advantages: the declarativity, which enables to easily add new constraints and the ability to find an optimal solution satisfying all the constraints (when there exists one).

We propose two models based on CP to address constrained clustering tasks. The models are flexible and general and supports instance-level constraints and different cluster-level constraints. It also allows the users to choose among different optimization criteria. In order to improve the efficiency, different aspects have been studied in the dissertation. Experiments on various classical datasets show that our models are competitive with other exact approaches. We show that our models can easily be embedded in a more general process and we illustrate this on the problem of finding the Pareto front of a bi-criterion optimization process. To the best of our knowledge, we are the first to find a global optimum for the problem of Constrained Clustering with the criteria of diameter, split and within cluster sum of dissimilarities and to solve the bi-criterion (split-diameter) clustering with user constraints.

Keywords: constrained clustering, bicriterion clustering, constraint programming, modelling.

Remerciements

Je voudrais remercier par ces quelques lignes toutes les personnes sans qui ce travail de thèse n'aurait pu voir le jour. Ma gratitude va d'abord à Madame Christel Vrain et Madame Thi-Bich-Hanh Dao, mes directrice et co-encadrante de thèse, pour leurs conseils avisés, leur soutien constant et leur grande disponibilité même pendant les vacances. Les nombreux échanges avec elles ont en effet joué un rôle décisif dans ma conception des pistes de recherche. Je les remercie par ailleurs de leur compréhension et de leurs encouragements en un moment difficile où je ne savais comment rédiger la thèse.

Je remercie vivement Monsieur Bruno Crémilleux , Monsieur Luc De Raedt et Madame Christine Solnon d'avoir accepté d'être les rapporteurs de ce travail. Leurs nombreuses remarques et suggestions ont beaucoup amélioré la qualité de ce mémoire. Mes remerciements s'adressent également à Monsieur Yves Deville et Monsieur Lakhdar Sais pour l'honneur qu'ils me font en acceptant de siéger dans mon jury. Je tiens à remercier le Laboratoire LIFO de l'Université d'Orléans, qui m'a accueilli et financé pendant toute la période de la réalisation de cette thèse, ainsi que les chercheurs qui m'ont aidé chacun à leur façon, dans la vie académique et quotidienne.

Enfin, mes pensées vont vers mes amis, mes sœurs et tout particulièrement mes parents qui m'ont toujours soutenu moralement durant toutes ces années ainsi que mon épouse sans qui rien de tout cela n'aurait été possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Ce chapitre synthétise le contexte et la problématique de la thèse. Nous présentons brièvement les notions principales dans le mémoire: la classification non supervisée (clustering), le clustering sous contraintes utilisateur et la Programmation par Contraintes (PPC). Dans cette thèse, nous étudions le cadre de la PPC pour modéliser les tâches de clustering sous contraintes utilisateur. Nous proposons deux modèles basés sur la PPC pour le clustering sous contraintes utilisateur. Les modèles sont généraux et flexibles, ils permettent d'intégrer des contraintes d'instances *must-link* et *cannot-link* et différents types de contraintes sur les clusters. Ils offrent également à l'utilisateur le choix entre différents critères d'optimisation. Afin d'améliorer l'efficacité, divers aspects sont étudiés. Le chapitre précise les contributions, les publications et l'organisation du mémoire.

Contents

Cluster analysis, also called clustering, is a fundamental task in Data Mining, which aims at partitioning a given set of objects into *homogeneous* and *well-separated* subsets called classes or clusters. Homogeneity means that the objects inside the same cluster must be similar while well separation expresses that objects in a cluster must be different from the objects belonging to other clusters. The problem has been studied since the 18^{th} century with applications in natural sciences, economics, sociology, geology and many other fields. Cluster analysis is an example of unsupervised learning, as it does not require predefined category labels of objects. It helps to find and understand the nature or the structure of the data. For instance, in marketing, by analysing the buying records of supermarkets, cluster analysis can find groups of customers with similar behaviour. Cluster analysis is a difficult task with many challenges, for instance: how to define the similarities between objects? how to decide the number of clusters? or how to evaluate the clusters? Although there have been hundreds of different approaches in the literature, such a general clustering algorithm does not exist. In cluster analysis, there is usually no single correct result and a clustering algorithm may give good results with some types of data and bad results with other data.

In many real applications, there exists some background knowledge about the data that could be useful in clustering. Since the last decade, cluster analysis has been extended to semi-supervised clustering, so as to integrate previous knowledge on data to clustering algorithms. The background knowledge is usually given in the form of user

constraints. The constraints can be classified into cluster-level constraints, specifying requirements on the clusters, or instance-level constraints, specifying requirements on pairs of objects. Most of the attention has been put on instance-level constraints, also called pairwise constraints, first introduced in [Wagstaff & Cardie 2000]. The instance-level constraints are composed of must-link and cannot-link constraints. A must-link constraint on two data objects specifies that they have to appear in the same cluster, whereas a cannot-link constraint specifies that the two objects must not be in the same cluster. Over the last decade, many works have been done to extend classical algorithms for handling must-link and cannot-link constraints, as for instance extensions of COBWEB [Wagstaff & Cardie 2000], k-means [Wagstaff *et al.* 2001, Bilenko *et al.* 2004], hierarchical clustering [Davidson & Ravi 2005a] or spectral clustering [Lu & Carreira-Perpinan 2008, Wang & Davidson 2010], etc. This is achieved either by modifying the dissimilarity measure, or the objective function, or the search strategy. However, there is no general solution to extend traditional algorithms to different types of constraints.

Recently, there has been a growing interest in developing declarative framework for Data Mining [Raedt *et al.* 2011], including the problem of constrained clustering. Declarative frameworks may be less efficient than classic algorithms for a specific task, but it is more flexible and general, easy to incorporate new knowledge to the task. [Guns *et al.* 2013] proposes a framework in Constraint Programming for k-pattern set mining. [Métivier *et al.* 2012] presents a constraint-based language for expressing queries to discover patterns in Data Mining. A Satisfiability of boolean formula (SAT) framework for constrained clustering is proposed in [Davidson *et al.* 2010] for cluster analysis with 2 clusters. The framework integrates different kinds of user-constraints: must-link, cannot-link, diameter and split constraints. [Mueller & Kramer 2010] proposes an approach to constrained clustering based on Integer Linear Programming. This approach takes a set of candidate clusters as input and constructs a clustering by selecting a suitable subset. It allows different kinds of constraints on clusters or on the set of clusters, but no constraint on individual objects. [Babaki *et al.* 2014] proposes an exact approach for constrained clustering with the criterion of minimizing the within-cluster sum of squares, based on Integer Linear Programming. It allows must-link, cannot-link and all constraints that are anti-monotone.

Among generic optimization tools, Constraint Programming (CP) is a powerful paradigm for solving combinatorial search problems that relies on a wide range of techniques from Artificial Intelligence, Computer Science and Operational Research. The basic idea of Constraint Programming is to solve problems by stating constraints that must be satisfied by the solution. The main principles in CP are: (1) users specify declaratively the problem as a Constraint Satisfaction Problem; (2) solvers search for solutions by constraint propagation and search. We claim that Constraint Programming might be a useful tool for the task of cluster analysis with constraints. Relying on Constraint Programming has two main advantages: the declarativity, which enables to easily add new constraints and the ability to find an optimal solution satisfying all the constraints (when there exists one). In this dissertation, we present our research on Constraint Programming to solve the problem of Constrained Clustering.

1.1 Dissertation Contributions

The main contributions of this dissertation are three folds:

- We propose two models based on Constraint Programming, allowing to find an optimal solution for clustering under constraints, given an optimization criterion.
- We show that such a framework can easily be embedded in a more general process and we illustrate this on the problem of finding the Pareto front of a bi-criterion (split-diameter) optimization process.
- We have proposed new global constraints to improve the performance of the framework.

We presented the first model [Dao *et al.* 2013d, Dao *et al.* 2013c, Dao *et al.* 2013a], based on Constraint Programming, which enables to design clustering tasks by specifying an optimization criterion and some constraints either on the clusters or on pairs of objects. We proposed improvements of the search strategy and a filtering algorithm [Dao *et al.* 2013b] in order to enhance the performance of the model. In our framework, several classical optimization criteria are considered and they can be coupled with different kinds of constraints. Experimental results show that our model dominates the state-of-the-art exact clustering algorithms for the criterion of diameter.

Next, we analysed the limitation of the framework and we proposed a new and improved model in Constraint Programming [Dao *et al.* 2014a, Dao *et al.* 2014c, Dao *et al.* 2014b]. The second model is more flexible, as it does not require to set the number of clusters. Moreover, the model is lighter in terms of the number of variables and constraints. We also developed dedicated global constraints for two criteria. Experiment results show that the second model it not only more general but also much more efficient.

Finally, we studied the problem of constrained clustering with a multi-criterion optimization and demonstrated the use of our model in order to solve the problem of clustering with the bi-criterion of diameter and split [Dao *et al.* 2014a, Dao *et al.* 2014c, Dao *et al.* 2014b].

1.2 List of publications

International Journal Article

- Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Constrained Clustering by Constraint Programming.* Artificial Intelligence, 2014. To appear with a minor revision.

National Journal Article

- Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Clustering sous contraintes en PPC.* Revue d'Intelligence Artificielle, 2014. To appear.

International Conference Papers

- 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

- 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

National Conference Papers

- Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Une approche en PPC pour la classification non supervisée.* In 13e Conférence Francophone sur l'Extraction et la Gestion des Connaissances EGC, 2013

- 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

- 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.

1.3 Dissertation Organization

This dissertation is organized as follows:

Chapter 2 is dedicated to preliminaries on Constraint Programming.

Chapter 3 gives background on the problem of cluster analysis, mainly from an optimization viewpoint. We present different approaches of cluster analysis that are related to our research. Finally, we present the state of the art of constrained clustering.

Chapter 4 brings an overview of recent declarative frameworks for Data Mining, specially those that are related to cluster analysis.

Chapter 5 presents our first model for solving the problem of constrained clustering. The model, based on CP, supports instance-level constraints, specifying requirements on pairs of objects, as well as different cluster-level constraints, specifying requirements on clusters. It also allows users to choose among different optimization criteria. The contributions of this Chapter have been published in [Dao *et al.* 2013d, Dao *et al.* 2013c, Dao *et al.* 2013a, Dao *et al.* 2013b].

Chapter 6 presents our second model for constrained clustering. The model is more flexible as it does not require to set a priori the number of clusters k, but only bounds on k are required. The second model is lighter in terms of the number of variables, as well as the number of constraints. The contributions of this Chapter are in [Dao *et al.* 2014a, Dao *et al.* 2014c, Dao *et al.* 2014b].

Chapter 7 gives the introduction of multi-objective optimization, and demonstrate the use of our model for finding the optimal Pareto solutions of a bi-criterion optimization process. The contributions of this Chapter are in [Dao *et al.* 2014a, Dao *et al.* 2014c, Dao *et al.* 2014b].

In the conclusion the dissertation is summarized and perspectives are discussed.

Chapter 2

Constraint Programming

Dans ce chapitre, nous présentons des connaissances de base en Programmation par Contraintes (PPC): le Problème de Satisfaction de Contraintes, la propagation de contraintes, les contraintes globales et le filtrage, la recherche de solutions et la notion de symétrie. Un Problème de Satisfaction de Contraintes (CSP) est un triplet $\langle X, Dom, C\rangle$ où $X = \langle x_1, x_2, \ldots, x_n\rangle$ est un n-tuple de variables, $Dom = \langle Dom(x_1), Dom(x_2), \ldots, Dom(x_n)\rangle$ est un n-tuple de domaines ($x_i \in Dom(x_i)$), $C = \langle C_1, C_2, ..., C_t\rangle$ est un t-tuple de contraintes où chaque contrainte C_i exprime une condition sur un sous-ensemble de X. Une solution d'un CSP est une affectation complète de valeurs $a_i \in Dom(x_i)$ à chaque variable x_i qui satisfait toutes les contraintes de C. La propagation de contraintes opère sur une contrainte c et enlève du domaine des variables de c des valeurs dont on est certain qu'elles ne pourront pas apparaître dans une solution. Les contraintes globales représentent des relations sur des ensembles de variables et en général, elles peuvent être exprimées par une conjonction de contraintes élémentaires. Cependant, définies d'une façon globale, les contraintes globales bénéficient de mécanismes de propagations beaucoup plus efficaces, qui sont effectuées par des algorithmes de filtrage. Un solveur de contraintes itère une étape de propagation suivie d'une étape de branchement jusqu'à ce qu'une solution soit trouvée.

Contents

This chapter gives an introduction and basic notions of Constraint Programming (CP). A large part of this chapter is based on the books [Apt 2003] and [Rossi *et al.* 2006].

2.1 General Concepts

In this section, we present general concepts related to Constraint Programming. Constraint Programming is a powerful paradigm for solving combinatorial search problems

that relies on a wide range of techniques from Artificial Intelligence, Computer Science and Operational Research. The main principles in CP are: (1) users specify declaratively the problem in a Constraint Satisfaction Problem; (2) solvers search for solutions by constraint propagation and search.

Definition 1 *Let x be a variable. The domain of x, denoted by $Dom(x)$, is a set of possible values that can be assigned to x. For each variable x, only a single value in $Dom(x)$ can be assigned to x.*

Example 1 x_1, x_2, x_3 are variables with the domains: $Dom(x_1) = \{1, 2, 3\}, Dom(x_2) = \{2, 5\}$ and $Dom(x_3) = \{1, 3, 5\}$.

Definition 2 *Let $X = \{x_1, x_2, ..., x_n\}$ be a set of variables. An assignment is a n-tuple $t = (t_1, \ldots, t_n) \in Dom(x_1) \times \ldots \times Dom(x_n)$ such that $x_i = t_i$ for all $x_i \in X, n = |X|$.*

Example 2 The tuple $(1, 2, 5)$ is an assignment of variables given in Example 1. It is equivalent with $x_1 = 1, x_2 = 2$ and $x_3 = 5$.

Definition 3 *Let $X = \{x_1, x_2, ..., x_n\}$ be a set of variables. A constraint C on an ordered tuple of variables, denoted by $Var(C) = (x_{i1}, \ldots, x_{ik}) \subseteq X$, is a subset of the cartesian product of the domains of the variables in $Var(C)$, i.e., $C \subseteq Dom(x_{i1}) \times \ldots \times Dom(x_{ik})$, that specifies the allowed combinations of values for the variables. $Var(C)$ is the scope of the constraint and $|Var(C)|$ is the arity of the constraint C. C is called an unary constraint if it is defined on one variable and is called a binary constraint if it is defined on two variables. In many cases, constraints can be expressed by mathematical formulas on variables.*

Example 3 Given the variables defined in Example 1, we consider two binary constraints: $C_1 \equiv x_1 \neq x_2$ and $C_2 \equiv x_1 + x_3 \leq 6$. The first constraint can be defined as $\{(1, 2), (1, 5), (2, 5), (3, 2), (3, 5)\}$. The second constraint is equivalent to $\{(1, 1), (1, 3), (1, 5), (2, 1), (2, 3), (3, 1), (3, 3)\}$.

Definition 4 *A Constraint Satisfaction Problem (CSP) P 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 $x_i \in Dom(x_i)$,*
- *$C = \langle C_1, C_2, ..., C_t\rangle$ is a t-tuple of constraints where each constraint C_i expresses a condition on a subset of X.*

Example 4 Based on Examples 1 and 3, we have a CSP defined as:

$$x_1 \in \{1, 2, 3\}, x_2 \in \{2, 5\}, x_3 \in \{1, 3, 5\}$$

$$x_1 \neq x_2, x_1 + x_3 \leq 6$$

Definition 5 *Given a CSP* $P = \langle X, Dom, C\rangle$ *with* $X = \{x_1, x_2, ..., x_n\}$, $Dom = \langle Dom(x_1), Dom(x_2), \ldots, Dom(x_n)\rangle$ *and a set of constraints* $C = \{C_1, C_2, ..., C_m\}$, *a tuple* $t = (t_1, \ldots, t_n)$ *satisfies a constraint* C_i *defined on the variables* $x_{i_1}, \ldots, x_{i_k}$ *if the tuple* $(t_{i_1}, \ldots, t_{i_k}) \in C_i$.

Example 5 Given the CSP in Example 4, the tuple (1, 2, 3) satisfies the constraint $x_1 \neq x_2$ since the tuple (1,2) (which is equivalent to the assignment $x_1 = 1, x_2 = 2$) satisfies the constraint $x_1 \neq x_2$.

Definition 6 *Given a CSP* $P = \langle X, Dom, C\rangle$ *with* $X = \{x_1, x_2, ..., x_n\}$, $Dom = \langle Dom(x_1), Dom(x_2), \ldots, Dom(x_n)\rangle$ *and a set of constraints* $C = \{C_1, C_2, ..., C_m\}$, *a tuple* $t = (t_1, \ldots, t_n) \in Dom(x_1) \times \ldots \times Dom(x_n)$ *is a solution of this CSP if it satisfies every constraint in* C.

Example 6 Given the CSP in Example 4, a solution to the problem is the tuple (1,2,3). The assignment of $x_1 = 1, x_2 = 2$ and $x_3 = 3$ satisfies all the constraints in this CSP.

Definition 7 *A constraint optimization problem (COP) is a CSP* $P = \langle X, Dom, C\rangle$ *together with an objective function* $f : Dom(x_1) \times \ldots \times Dom(x_n) \rightarrow \mathbb{R}$ *to be optimized. An optimal solution to a COP is a solution to* P *that optimizes the function* f.

Example 7 Given the CSP in Example 4 and a function $f = x_1+x_2+x_3$ to be maximized, we have a COP defined as:

$$x_1 \in \{1, 2, 3\}, x_2 \in \{2, 5\}, x_3 \in \{1, 3, 5\}$$

$$x_1 \neq x_2, x_1 + x_3 \leq 6$$

$$maximize\ x_1 + x_2 + x_3$$

An optimal solution of this COP is the tuple $(1, 5, 5)$, which is equivalent to the assignment $x_1 = 1, x_2 = 5, x_3 = 5$. Another optimal solution of this COP is the tuple $(3, 5, 3)$.

In general, CSPs are NP-hard. However, techniques used in CP solvers allow to solve efficiently many difficult problems. For solving a practical problem by CP, the problem must be modeled as a CSP or a COP, which means it is needed to precise:

- Definition of variables and theirs domains.
- Specification of the constraints between variables.
- Definition of the objective function on related variables if it is a COP.

CSPs and COPs can be modeled and then solved by a constraint solver such as: Gecode [Gecode Team], Choco [choco Team 2010], Comet [Van Hentenryck & Michel 2005], IBM ILOG CP Optimizer [Laborie 2009], ...

2.2 Constraint Propagation

In general, a constraint solver finds one or all solutions of a CSP by using successively constraint propagation and search. Constraint propagation consists of removing inconsistent values from domains of variables. In consequence, the search space can be significantly reduced.

Definition 8 *Let C be a constraint defined on variables $Var(C) = \{x_{i1}, \ldots, x_{im}\}$. A support for a value $v \in Dom(x_{ij})$ is an assignment $t = (t_{i1}, \ldots, t_{im})$ such that $t_{ij} = v$ and $(t_{i1}, \ldots, t_{im}) \in C$. If there doesn't exist such a support, the value v is inconsistent with the constraint C.*

The constraint propagation phase is proceeded by propagation algorithms, or also called filtering algorithms.

Definition 9 *A filtering algorithm associated with a constraint C is an algorithm that removes inconsistent values of variables involved in the constraint. If the filtering algorithm cannot remove any inconsistent value in the domain of variables, the constraint C is locally consistent.*

A constraint is locally consistent if the filtering algorithm cannot remove any inconsistent values. In constraint propagation, filtering algorithms are applied repeatedly until all constraints are locally consistent, with respect to domains of variables. When this process terminates, the CSP is called locally consistent. Many types of local consistencies have been studied in Constraint Programming, including node consistency, arc consistency, generalized arc consistency and path consistency. The simplest type is the node consistency which requires that every unary constraint C defined on a variable x_i is satisfied by all values in $Dom(x_i)$. In practice, the node consistency can be achieved by reducing the domain of each variable to the values that satisfy all unary constraints. As a result, unary constraints can be handled one time at the beginning and be neglected later.

Definition 10 *A CSP is node consistent if for every variable $x \in X$, every unary constraint defined on x is consistent with the domain of x.*

Example 8 Given the CSP in Example 4 with a new unary constraint $C_3 \equiv x_1 \leq 2$. The value 3 is inconsistent in $Dom(x_1)$ with the constraint C_3. By filtering this value, $Dom(x_1) = \{1, 2\}$ and the CSP is node consistent.

The next level of consistency is arc consistency, which is defined on binary constraints.

Definition 11 *Given a binary constraint C defined on variables x and y, with the domains $Dom(x)$ and $Dom(y)$ respectively. The constraint C is arc consistent (AC) if for every value $a \in Dom(x)$, there exists a value $b \in Dom(y)$ such that the assignment $x = a, y = b$ satisfies the constraint C and vice versa:*

$$\forall a \in Dom(x), \exists b \in Dom(y) : (a, b) \in C$$

$$\forall b \in Dom(y), \exists a \in Dom(x) : (a, b) \in C$$

A CSP is arc consistent if all its binary constraints are arc consistent.

Example 9 The two binary constraints of the CSP given in Example 4 are arc consistent. Considering a new binary constraint: $C_3 \equiv x_1 = x_3$, it is not arc consistent because if we assign the value 1 to the variable x_1, there is no value for x_3 in its domain to satisfy the constraint $x_1 = x_3$.

The notion of arc consistency can be generalized to arbitrary constraints.

Definition 12 *A constraint C is generalized arc consistent (GAC) if for each variable involved in the constraint, for any of its value there exists a value for every other variables such that the assignment of those values to variables satisfies the constraint.*

In another way, a constraint C is generalized arc consistent if every value of every variable involved in the constraint has at least a support. A CSP is generalized arc consistent if all of its constraints are generalized arc consistent.

The basic arc consistency algorithm is presented in Algorithm 1. All the binary constraints are revised each time an inconsistent value is detected and removed. In the literature, many other arc consistency algorithms (AC-2, ..., AC-7) have been proposed to reduce the complexity.

Algorithm 1: Basic Arc Consistency algorithm

```
1  Function Revise(x_i, x_j)
2      DELETED ← false
3      foreach value a_i in Dom(x_i) do
4          if there is no a_j in Dom(x_j) such that (a_i, a_j) satisfies all binary
           constraints on (x_i, x_j) then
5              delete a_i in Dom(x_i)
6              DELETED ← true
7          end
8      end
9      Return DELETED
10 Function ArcConsistency(x_i, x_j)
11     repeat
12         CHANGED ← FALSE
13         foreach arc(x_i, x_j) do
14             CHANGED ← Revise(x_i, x_j) or CHANGED
15         end
16     until not Changed;
```

Example 10 Assume the variables x_1, x_2 and x_3 taking integer values in $[1, 4]$[1] with constraints: $x_1 < x_2$ and $x_2 < x_3$. Suppose that arcs will be revised in the order (x_1, x_2), (x_2, x_1), (x_2, x_3), (x_3, x_2).

- we first select the arc (x_1, x_2) and remove the value 4 from $Dom(x_1)$ because if $x_1 = 4$, there exists no value for x_2 such that $x_1 < x_2$

[1] For a discrete variable, $[i, j]$ denotes the set of integers from i to j.

- arc (x_2, x_1) is revised and the value 1 is removed from $Dom(x_2)$ for the same reason.
- arc (x_2, x_3) is revised and the value 4 is removed from $Dom(x_2)$. Now $Dom(x_1) = [1, 3], Dom(x_2) = [2, 3], Dom(x_3) = [1, 4]$
- arc (x_3, x_2) is revised and the values 1 and 2 are removed from $Dom(x_3)$. Now $Dom(x_1) = [1, 3], Dom(D_2) = [2, 3], Dom(x_3) = [3, 4]$
- all the arcs are revised again: when revising (x_1, x_2), the value 3 is removed from $Dom(x_1)$
- Finally $Dom(x_1) = [1, 2], Dom(x_2) = [2, 3], Dom(x_3) = [3, 4]$

The Arc Consistency techniques reduce significantly the search space in many CSPs but there still exist many other possible inconsistencies.

Example 11 Assume the variables x_1, x_2 and x_3 taking integer values in $\{1, 2\}$ with constraints: $x_1 \neq x_2$, $x_2 \neq x_3$ and $x_3 \neq x_1$. The Arc Consistency techniques can not detect any inconsistencies in this case although there is no solution.

Path Consistency is stronger than Arc Consistency, it detects inconsistencies in every path in the constraint graph. It removes more inconsistencies but it is too costly and is not efficient in most CSPs. Moreover, the Path Consistency is not able to detect all of the inconsistencies.

2.3 Global constraints

A global constraint is a constraint that expresses a relationship between a non-fixed number of variables. This relationship can be expressed by a conjunction of simpler constraints.

Definition 13 *Let $C = \{C_1, \ldots, C_m\}$ be a set of constraints. The constraint C_G, which is equal to the conjunction of all the constraints of C: $C_G = \wedge\{C_1, \ldots, C_m\}$, is a global constraint.*

A typical example of a global constraint is the AllDifferent$(x_1, x_2, \ldots, x_n)$ constraint. This constraint requires that values assigned to the variables $x_1, \ldots, x_n$ must be pairwise distinct.

Definition 14 *Let $X = \{x_1, \ldots, x_n\}$ be a set of variables. The AllDifferent constraint on X is defined as:*

$$AllDifferent(x_1, \ldots, x_n) = \{(d_1, \ldots, d_n) \mid \forall_i : d_i \in Dom(x_i), \forall_{i \neq j} : d_i \neq d_j\}$$

For instance, the constraint AllDifferent$(x_1, x_2, \ldots, x_n)$ can be replaced by $\frac{n(n-1)}{2}$ binary constraints: $x_1 \neq x_2, x_1 \neq x_3, \ldots, x_{n-1} \neq x_n$.

Using global constraints makes a CSP become more intuitive. But more importantly, powerful filtering algorithms can be designed because the set of simple constraints can be considered as a whole at the same time. Many filtering algorithms for global constraints

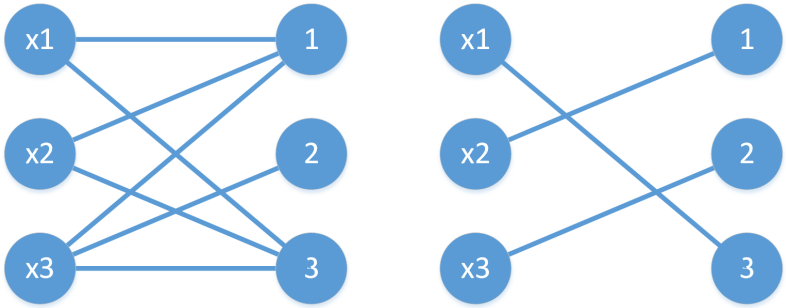

Figure 2.1: graph value (left) and a maximum matching (right) of Example 12

use operational research techniques or graph theory to achieve generalized arc consistency with low complexity. In this section, we present one of the most successful filtering algorithms associated to the well known AllDifferent constraint, proposed by [Régin 1994]. The algorithm is based on graph theory.

Definition 15 *Given a graph $G = (V, E)$, a matching in G is a set $M \subseteq E$ of disjoint edges, i.e no two edges in M share a vertex. A matching M is a maximum matching if it contains the largest possible number of edges.*

Definition 16 *Let $X = x_1, \ldots, x_n$ be a sequence of variables with respective domains $Dom(x_1), \ldots, Dom(x_n)$. The bipartite graph $G = (V, E)$, with $V = X \cup Dom(x_1) \cup \ldots \cup Dom(x_n)$ and $E = \{(x_i, d) | x_i \in X, d \in Dom(x_i)\}$ is called the value graph of X.*

In the value graph G, each variable x_i corresponds to a variable-node while each value v corresponds to a value-node. There is an edge between a variable-node x_i and a value-node v iff the value v is in the domain of x_i.

Example 12 Assume we have 3 variables x_1, x_2 and x_3 with the domains: $x_1 \in \{1, 3\}, x_2 \in \{1, 3\}, x_3 \in \{1, 2, 3\}$. The value graph of those variables and a maximum matching are shown in Figure 2.1.

Let $X = \{x_1, \ldots, x_n\}$ be a set of variables and let G be the value graph of X, [Régin 1994] introduced two important propositions on the relationship between assignments of X and matchings.

- There is a matching of cardinality n if and only if the AllDifferent constraint 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.

From that observation, the filtering algorithm in [Régin 1994] consists of two steps:

- Determine if there exists at least one matching of cardinality $|X|$. If such a matching does not exist, the constraint cannot be satisfied.
- Mark all edges (x_i, v) that could not belong to any matching of cardinality n. For each edge (x_i, v) satisfying the condition, the value v is inconsistent and can be removed from the domain of x_i.

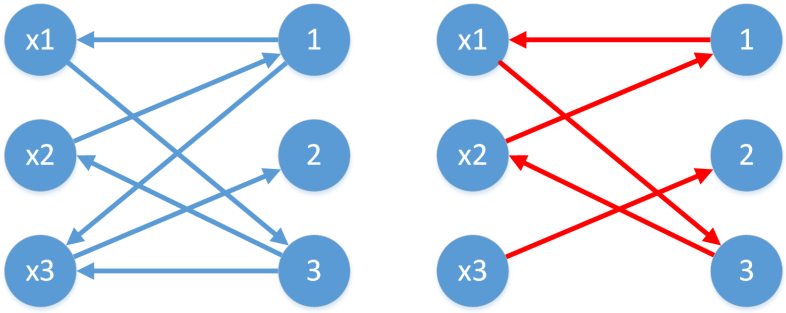

Figure 2.2: residual graph G_M (left) and marked edges (right) in Example 13

The first step can be solved by finding a matching of maximum cardinality M in the value graph. Because there is only edges between variable-nodes and value-nodes, the maximum cardinality of matchings in the value graph is at most $|X|$.

For the second step, [Régin 1994] constructs a residual graph G_M that is a directed version of the graph G, with the direction based on M. Edges that belong to the maximum matching M are oriented from the value-nodes to variable-nodes. Otherwise, edges are oriented from value-nodes to variable-nodes. The filtering algorithm finds all strongly connected components in G_M and marks all edges in those components. A simple depth-first search marks all edges that lie on an even-length path starting at the free nodes. It is proven that those marked edges belong to at least one maximum cardinality matching and the unmarked edges do not belong to the matching M and cannot be a part of any other maximum cardinality matching. Hence, the AllDifferent constraint is made generalized arc consistent by removing all the values $v \in D(x_i)$ such that (x_i, v) is an unmarked edge.

Example 13 Given the value graph in Example 12, the graph G_M and marked edges are presented in Figure 2.2. There are two unmarked edges $(x_3, 1)$ and $(x_3, 3)$. Those edges do not belong to any maximum cardinality matching, therefore we can remove value 1 and 3 from the domain of x_3.

2.4 Search

Definition 17 *Given a CSP P_0, a search tree for P_0 is a rooted tree such that:*

- *its nodes are CSPs,*
- *its root is P_0,*
- *if $P_1, \ldots, P_m$ where $m > 0$ are all direct descendants of P_0, then the union of the solution sets of $P_1, \ldots, P_m$ is equal to the solution set of P_0.*

A node in the search tree is a dead end if it does not lead to a solution. The general idea is that the search tree is built by branching (also called splitting) a CSP into smaller CSPs until each leaf is a dead end or a solved CSP. At each node, the constraint propagation is applied to the CSP until the domains of variables are consistent. There are three cases:

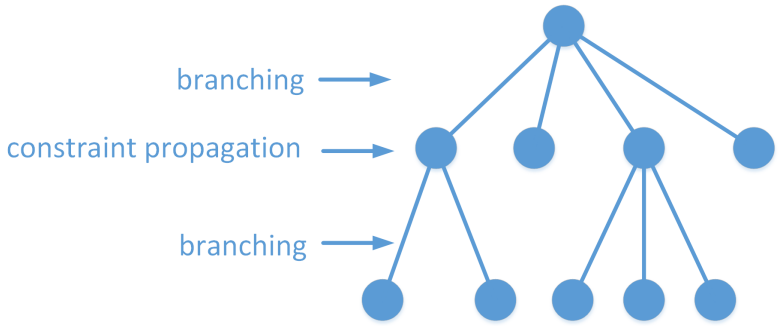

Figure 2.3: A search tree for a CSP

- If all the variable domains are singletons, these singleton domains form a solution and the search continues to explore other branches.
- If any of the variable domains is empty, the CSP has no solution and the search explorers other branches.
- Otherwise, the current CSP is locally consistent and it is split into smaller CSPs either by splitting a constraint or by splitting the domain of a variable. In this thesis, we only consider the latter technique.

The strategy of extending a node in the search tree is often called a branching strategy. An example of a search tree is given in Figure 2.3. Solving a CSP can be complete or incomplete:

- A complete search guarantees to find all existing solutions, or proves that the CSP does not have any solution. A complete search also guarantees to find the optimal solution of a COP. The chronological backtracking search is the most well known complete approach. A search is completed if any leaf in the search tree is either a dead end or a solved CSP.
- An incomplete search does not explore the search space completely, but focuses on branches which have a high probability of containing a solution. It is often effective at finding a solution of CSPs if one exists. In COPs, an incomplete search can be used to find an approximation to an optimal solution. Incomplete searches can not be used to prove that the CSP does not have a solution or to prove the optimality of a solution as the searches does not explore all possible branches of the search tree.

2.4.1 Backtracking search in CP

The naive backtracking algorithm begins with the root node P_0 which is the original CSP. The node is then split following a branching strategy. For example, in the branching strategy of enumeration, the node P_0 is extended by selecting a variable, for example the variable x, and splitting its domain into singleton sets. For each value a_i in the domain of x, a descendant node P_i is generated. P_i is a copy of P_0 except the domain of x is replaced by a singleton: $Dom(x) = \{a_i\}$. The backtracking algorithm then visits a node P_i and the constraint propagation is applied to P_i. There are three cases after that:

- The domain of a variable is reduced to an empty set: P_i is a failed CSP, the search backtracks to visit other non-visited nodes.
- All the variable domains are singletons: these singleton domains form a solution, the search backtracks to visit other non-visited nodes.
- Otherwise: Node P_i is extended and the search visits one of the descendent nodes.

The search is terminated after visiting every generated nodes. Three popular branching strategies often used are:

- Enumeration: A branch is generated for each value in the domain of the variable, as explained above.
- Binary choice points: A variable x and a value $a \in Dom(x)$ are chosen. There are two descendant nodes P_1 and P_2. In P_1, the domain of x is replaced by a singleton $Dom(x) = \{a\}$ whereas in P_2, the domain of x contains all remaining values.
- Domain splitting: Here the domain of the variable is split to create the branching. Suppose that a variable x and a value a is chosen for branching. There are two descendant nodes P_1 and P_2. In P_1, the domain of x contains all values which are less or equal to a, and P_2 contains the remaining values.

Strategies for choosing variables and for choosing values for branching are extremely important, since they can reduce drastically the search tree.

2.4.2 Variable and value ordering heuristics

For branching a node, we have to decide how to select a variable and how to split its domain. The order in which variables and values are selected is extremely important in the search process since correct choices can reduce drastically the number of nodes in the search tree. The variable ordering may be either static or dynamic. In the static ordering, the order of the variables is specified before the search begins whereas in the dynamic ordering, the choice of the variable is considered as the search processes. [Haralick & Elliott 1979] suggests that "*the best search order is the one which minimizes the expected length or depth of each branch*". In order to do that, [Haralick & Elliott 1979] introduced the **Fail-First Principle** that tries to select the variable which can quickly leads to a dead end node. The underlying idea is to detect failures early to avoid useless branches. If the current node P cannot lead to a solution, then the sooner we detect the failure the better. The most common variable ordering heuristic with the Fail-First Principle is the Smallest Remaining Domain method. In this method, the variable with the smallest domain size is selected for branching with the idea that the variable is most likely to lead to a dead-end. This method is based on the assumption that each value in a domain has the same probability to be in a solution, so the variable with the smallest remaining domain size has the smallest chance to be successful. Other heuristics may be used during the search to determine the next variable to be instantiated:

- the variable that participates in a higher number of constraints,
- the variable that has the largest number of constraints with instantiated variables.

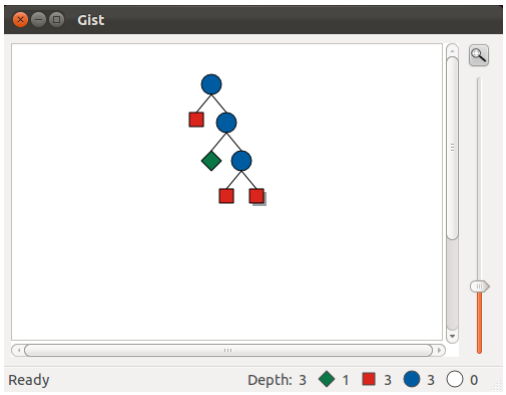

Figure 2.4: Search tree with the strategy: the variable with the smallest domain is selected for instantiation

The order of values for each variable is important too, a correct choice of values can help to reduce the search tree. The example below demonstrates the importance of the variable ordering.

Example 14 Find distinct digits for the letters S, E, N, D, M, O, R, Y such that

		S	E	N	D
+		M	O	R	E
=	M	O	N	E	Y

To model this problem, we can define a variable with the domain [0, 9] for each letter: $X = \{x_S, x_E, x_N, x_D, x_M, x_O, x_R, x_Y\}$.

The following constraints are added:

- $AllDifferent(x_S, x_E, x_N, x_D, x_M, x_O, x_R, x_Y)$
- $x_S \neq 0, x_M \neq 0$
- $1000x_S + 100x_E + 10x_N + x_D + 1000x_M + 100x_O + 10x_R + x_E = 10000x_M + 1000x_O + 100x_N + 10x_E + x_Y$

By using arc consistency techniques, we can reduce the domain of variables to:

$$Dom(x_S) = \{9\}, Dom(x_E) = [4, 7], Dom(x_N) = [5, 8], Dom(x_D) = [2, 8],$$

$$Dom(x_M) = \{1\}, Dom(x_O) = \{0\}, Dom(x_R) = [2, 8], Dom(x_Y) = [2, 8]$$

Figures 2.4 and 2.5 show two search trees with different strategies for the next variable to be instantiated. These search trees are generated by Gist environment of the Gecode solver where a blue circle denotes a stable state but not yet a solution, a red square represents a fail state (there is no solution) and a green diamond is a solution.

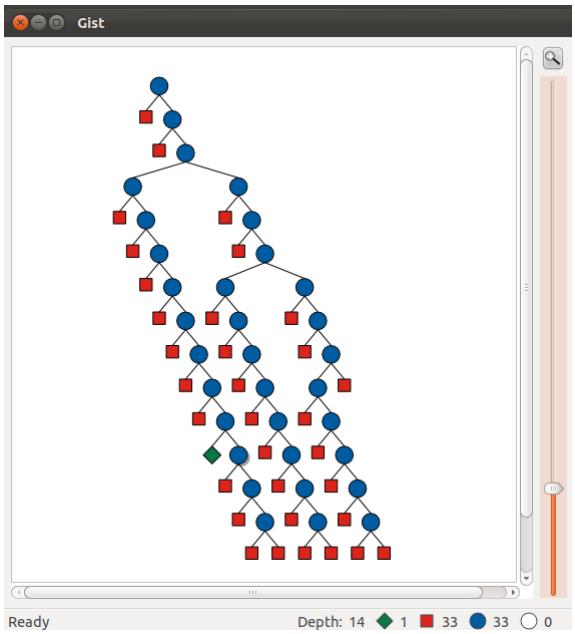

Figure 2.5: Search tree with the strategy: the variable with the biggest domain is selected for instantiation

2.4.3 Search for COPs

To solve a COP, a common approach is to use the branch-and-bound technique. A backtracking search is first used to find a solution which satisfies all the constraints without considering the optimization criterion. When a solution is found, the value of the optimization function is computed and an additional constraint is added to the CSP in order to forbid solutions that are not better than this solution. The solver continues to search for solutions of the new CSP and every time a solution is found, an additional constraint is added to the CSP. When the search space is explored completely, the last solution found is proven optimal.

Let us consider the following example for illustrating a Constraint Optimization Problem and search strategies.

Example 15 Find an assignment of digits to letters such that

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & S & T \\ \hline = & M & O & N & E & Y \end{array}$$

and the value of $MONEY$ is maximized. The optimal solution of this problem is $SEND = 9782$, $MOST = 1094$ and the value $MONEY = 10876$. This problem can be modeled as a Constraint Optimization Problem, using eight variables $x_S, x_E, x_N, x_D, x_M, x_O, x_T, x_Y$, whose domain is the set of digits $[0, 9]$, and a variable V of domain integer, which represents the objective function, which is to be maximized. Constraints that specify the problem are:

- the letters S and M must not be equal to 0: $x_S \neq 0,\ x_M \neq 0$

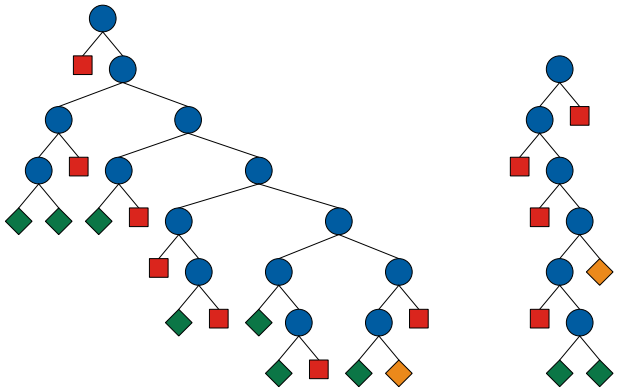

Figure 2.6: Search trees (blue circle: a stable state but not yet a solution, red square: a fail state (there is no solution), green diamond: an intermediary solution, orange diamond: the optimal solution.)

- all the letters are pairwise different: $\texttt{alldifferent}(x_S, x_E, x_N, x_D, x_M, x_O, x_T, x_Y)$
- $(1000x_S + 100x_E + 10x_N + x_D) + (1000x_M + 100x_O + 10x_S + x_T) = 10000x_M + 1000x_O + 100x_N + 10x_E + x_Y$
- $V = 10000x_M + 1000x_O + 100x_N + 10x_E + x_Y$.

The propagation of these constraints leads to a stable state with the domains:

$$Dom(x_S) = \{9\}, Dom(x_E) = \{2,3,4,5,6,7\}, Dom(x_M) = \{1\}, Dom(x_O) = \{0\},$$

$$Dom(x_N) = \{3,4,5,6,7,8\}, Dom(x_D) = Dom(x_T) = Dom(x_Y) = \{2,3,4,5,6,7,8\}$$

Strategies define the way to choose variables and for each chosen variable, the way to choose values. If variables are chosen in the order $x_S, x_E, x_N, x_D, x_M, x_O, x_T, x_Y$ and for each variable, the remaining values in its domain are chosen in an increasing order, the search tree has 29 states with 7 intermediate solutions (a solution which is better than the precedent one). However, if variables are chosen in the order $x_S, x_T, x_Y, x_N, x_D, x_E, x_M, x_O$, the search tree has only 13 states with 2 intermediate solutions. These two search trees are shown in Figure 2.6. For each stable state, the left branch corresponds to the case where the chosen variable is assigned to the chosen value, the right branch is the other case where the chosen value is removed from the domain of the chosen variable. In these search trees, blue circle is a stable state but not yet a solution, red square is a fail state (there is no solution), green diamond is an intermediary solution and the orange diamond is the optimal solution. It is worth to notice that it is an exhaustive search, the returned solution is guaranteed to be optimal. Let us notice that in problems where there are several solutions with the same optimal value, the system returns only a single optimal solution.

2.5 Symmetries in CP

Definition 18 *For any CSP instance $P = (X, D, C)$, a solution symmetry σ of P is a permutation of the set $X \times D$ that preserves the set of solutions to P.[Cohen et al. 2005]*

In other words, a solution symmetry is a bijection mapping on the set of variable-value pairs, that maps solutions to solutions and also non-solutions to non-solutions. As a result, given a solution symmetry σ, a complete assignment A is a solution if and only if $\sigma(A)$ is a solution. The solution symmetry makes some sub-trees in the search space to be equivalent. If two sub-trees are equivalent, they will both have no solution, or every solution in a sub-tree corresponds with a symmetric counterpart in the other sub-tree. As a result, it is not necessary to explore both sub-trees which are symmetrically equivalent. In practice, the most common symmetry types are: variable symmetry and value symmetry. Variable (value) symmetries are symmetries that can be represented by a permutation of only variables (values).

Example 16 The 8-queens problem: Find how to place 8 queens on a 8×8 board, so that they do not attack each other: no two queens share the same row, column or diagonal.

This problem can be modelled as a CSP with 8 variables $x_1 \dots x_8$ for queens, with the domain: $dom(x_i) = \{1, \dots, 8\}$. The variable x_i represents the column position for the queen in row i. With those definitions of variables, no two queens can be on the same row. The constraints for the problem are:

- $\forall i < j \in \{1, \dots, n\} : x_i \neq x_j$: Those constraints guarantee that two queens cannot be on the same column.
- $\forall i < j \in \{1, \dots, n\} : |x_i - x_j| \neq j - i$: Those constraints guarantee that no two queens are on the same diagonal.

With this modelling, there are solution symmetries as following:

1. The variable symmetry $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \leftrightarrow (x_8, x_7, x_6, x_5, x_4, x_3, x_2, x_1)$ that maps x_1 to x_8, x_2 to x_7, ... It represents a flipping horizontal on the board, as illustrated by the solution 0 and 1 in Figure 2.5. The solution 0 is:

   $$x_1 = 1, x_2 = 5, x_3, = 8, x_4 = 6, x_5 = 3, x_6 = 7, x_7 = 2, x_8 = 4.$$

   By switching values of x_1 and x_8, x_2 and x_7,..., we have the solution 1, which is symmetry to the solution 0:

   $$x_1 = 4, x_2 = 2, x_3, = 7, x_4 = 3, x_5 = 6, x_6 = 8, x_7 = 5, x_8 = 1.$$

2. The value symmetry $(1, 2, 3, 4, 5, 6, 7, 8) \leftrightarrow (8, 7, 6, 5, 4, 3, 2, 1)$ that maps value 1 to 8, 2 to 7,... It represents a flipping vertical on the board, as illustrated by the solutions 0 and 2 in Figure 2.5.

3. The variable-value symmetry that maps every $x_i = j$ to $x_j = i$ that represents a flipping diagonal on the board, for example the solutions 0 and 3 in Figure 2.5. Another variable-value symmetry maps every $x_i = j$ to $x_{8-j+1} = 8 - i + 1$, for instance the solution 0 and 4 in Figure 2.5.

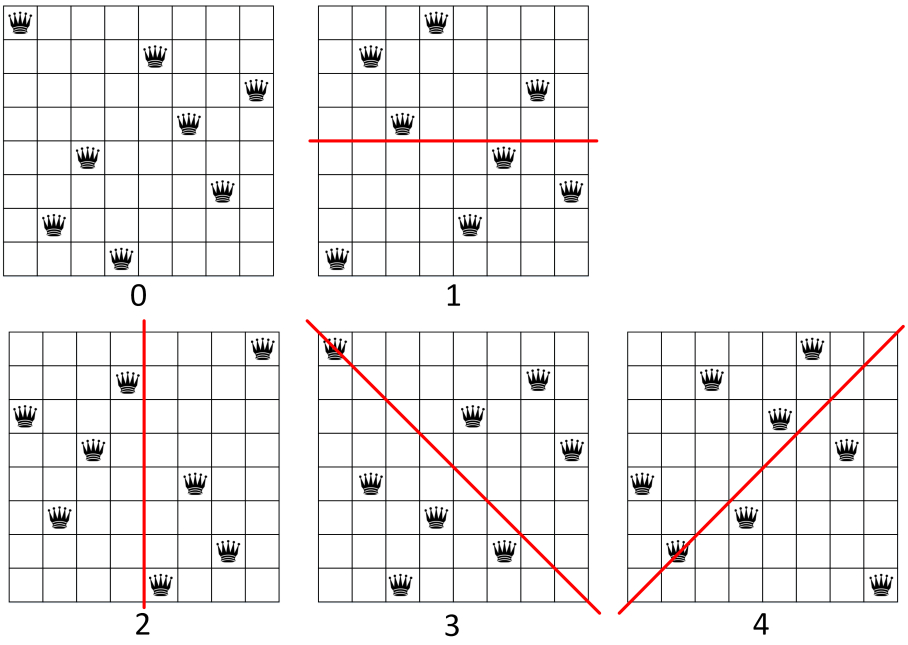

Figure 2.7: Solution symmetries on 8-queens problem

4. Combination of the above symmetries.

Many researches have addressed the importance of symmetry breaking in the search of CSPs. Solution symmetries can be avoided either by adding new constraints to prohibit them or by modifying the search heuristic to prune symmetric states during the search.

2.6 Summary

In this chapter, we present basic preliminaries of Constraint Programming. Constraint Programming is a powerful paradigm in which users specify declaratively the problem by variables and constraint, then the solver finds the solutions by constraint propagation and search.

More details on Constraint Programming can be found in text books [Apt 2003] and [Rossi *et al.* 2006].

Chapter 3

Clustering

Ce chapitre est consacré à l'état de l'art de la classification non supervisée (clustering) et du clustering sous contraintes, notamment les approches concernant nos recherches. Nous présentons les notions principales et différents critères pour le problème de clustering. Ce chapitre introduit ensuite des méthodes fondées sur le partitionnement des données et des méthodes hiérarchiques. Afin de mieux modéliser la tâche d'apprentissage, mais aussi dans l'espoir de réduire la complexité, des contraintes définies par l'utilisateur peuvent être ajoutées. Plusieurs types de contraintes peuvent être considérés; ces contraintes peuvent porter sur les clusters, comme par exemple sur leur diamètre ou sur leur taille, ou porter sur des paires d'objets qui doivent être ou ne doivent pas être dans une même classe. Nous présentons ces contraintes et les approches développées pour intégrer des contraintes utilisateur dans la tâche de clustering.

Contents

Cluster analysis is an important task in Data Mining, which aims at partitioning a given set of objects into *homogeneous* and *well-separated* subsets called classes or clusters. Homogeneity means that the objects inside the same cluster must be similar while well separation expresses that objects in a cluster must be different from the objects belonging to other clusters. The problem has been studied since 18^{th} century with applications in natural sciences, economics, sociology, geology and many other fields. The cluster analysis literature is vast and heterogeneous with hundreds of different algorithms.

Clustering could be seen as an optimization problem, i.e., finding a partition of the objects that optimizes a given criterion. In this chapter, we present the state of the art in clustering, mainly from an optimization viewpoint. We focus more on heuristics and

exact methods with optimization criteria that are closer to our research. We also bring an overview of constrained clustering with several well-known approaches.

3.1 Problem definitions

We consider a dataset of n objects $\mathcal{O} = \{o_1, \ldots, o_n\}$. Each object is described by its values on p attributes, also called variables. We denote the value of the j-th attribute of an object o_i by o_{ij}. Most cluster algorithms do not rely on the description of the objects but on a dissimilarity between them.

Suppose that we have a dissimilarity measure between any two objects $o_i, o_j \in \mathcal{O}$, denoted by d_{ij}. The dissimilarity d_{ij} is typically calculated by using a distance metric defined on the attribute space. A distance metric must satisfy the following properties [Han *et al.* 2011]:

1. Non-negativity: $d_{ij} \geq 0, \forall i, j \in [1, n]$. [1]
2. Identity of indiscernible: $d_{ii} = 0, \forall i \in [1, n]$.
3. Symmetry: $d_{ij} = d_{ji}, \forall i, j \in [1, n]$.
4. Triangle inequality: $d_{ij} \leq d_{it} + d_{tj}, \forall i, j, t \in [1, n]$.

Note that the dissimilarity measure must satisfy the first three properties, but may not satisfy the triangle inequality. In cluster analysis, the most popular distance metric is Euclidean distance and the squared Euclidean distance [Han *et al.* 2011]. Given two object o_i, o_j, the Euclidean distance between them is defined as:

$$d_{ij} = \sqrt{(o_{i1} - o_{j1})^2 + (o_{i2} - o_{j2})^2 + \ldots + (o_{ip} - o_{jp})^2}$$

The squared Euclidean distance which is defined as:

$$d_{ij} = (o_{i1} - o_{j1})^2 + (o_{i2} - o_{j2})^2 + \ldots + (o_{ip} - o_{jp})^2$$

Another popular distance measure is the Manhattan distance [Han *et al.* 2011], defined as:

$$d_{ij} = |o_{i1} - o_{j1}| + |o_{i2} - o_{j2}| + \ldots + |o_{ip} - o_{jp}|$$

Similarly, we denote a similarity measure between two objects o_i and o_j by s_{ij}. The similarity measure is widely used in spectral clustering. It is typically calculated by a Gaussian function of the dissimilarity [Luxburg 2007]:

$$s_{ij} = exp(-\frac{d_{ij}}{2\sigma^2})$$

where σ is a parameter.

There exist other similarity measures, for instance the normalized Pearson correlation, the Jaccard measure, the dice coefficient measure,... [Han *et al.* 2011].

Most clustering methods can be classified into two categories: hierarchical methods and partitioning methods. A partitioning algorithm aims at finding a partition P while a hierarchical clustering aims at finding a set of partitions $P_1, P_2, \ldots, P_l$ of $\mathcal{O}$.

[1] $[1, n]$ denotes the set of integers from 1 to n.

Definition 19 *Let $C_1, C_2, \ldots, C_k$ be subsets of $\mathcal{O}$. $\{C_1, C_2, \ldots, C_k\}$ is a partition of $\mathcal{O}$ into k clusters if:*

- *for all $c \in \{1, \ldots, k\}$, $C_c \neq \emptyset$*
- $\cup_c C_c = \mathcal{O}$
- *for all $c \neq c'$, $C_c \cap C_{c'} = \emptyset$.*

Many partitioning-based clustering methods rely on the choice of a representative for each cluster, it could be a centroid or a medoid. When the objects are vectors of p quantitative attributes, the centroid of a cluster is computed by averaging the data vectors belonging to that cluster.

Definition 20 *The centroid m_c of a cluster C_c is a point defined as:*

$$\forall j \in \{1, \ldots, p\} \quad (m_c)_j = \frac{1}{|C_c|} \sum_{o_i \in C_c} o_{ij}$$

When attributes of objects are qualitative, the centroid point can no longer be calculated and a point, usually a medoid, is chosen to represent each cluster.

Definition 21 *A medoid x_c of a cluster C_c is an object of the cluster whose average dissimilarity to all the objects in the cluster is minimal:*

$$x_c = \underset{o_i \in C_c}{\arg\min} \frac{1}{|C_c|} \sum_{x_j \in C_c} d_{ij}$$

Let us notice that a centroid may not belong to $\mathcal{O}$ whereas a medoid belongs to $\mathcal{O}$.

3.2 Criteria

3.2.1 Criteria for a cluster

Cluster analysis aims at finding clusters, which are *homogeneous* and *well-separated*. These requirements are usually expressed by an optimization criterion and the clustering task can be defined as finding a partition of objects that optimizes a given criterion.

The separation of a cluster C_c is typically measured by the split criterion.

- the split of a cluster $split(C_c)$ is the minimum dissimilarity between an object of cluster C_c and objects belonging to other clusters [Hansen & Jaumard 1997]:

$$split(C_c) = \min_{o_i \in C_c, o_j \notin C_c} d_{ij}$$

Another criterion for the separation is the cut between clusters. This criterion is widely used in the spectral clustering methods and is measured with the similarities between objects.

- the cut of a cluster C_c is the sum of similarities between objects of this cluster and objects belonging to other clusters:

$$cut(C_c) = \sum_{o_i \in C_c} \sum_{o_j \notin C_c} s_{ij}$$

The homogeneity of a cluster C_c can be measured by the following criteria [Hansen & Jaumard 1997]:

- the diameter $diameter(C_c)$ of cluster C_c is the maximum dissimilarity between two of its objects:

$$diameter(C_c) = \max_{o_i, o_j \in C_c} d_{ij}$$

- the radius $radius(C_c)$ of a cluster C_c is measured by the minimum for all objects o_i in the clusters of the maximum dissimilarity between o_i and any other object o_j in C_c:

$$radius(C_c) = \min_{o_i \in C_c} \max_{o_j \in C_c} d_{ij}$$

- the within cluster sum of dissimilarities $wcsd(C_c)$ of a cluster C_c is measured by the sum of dissimilarities between any two of its objects:

$$wcsd(C_c) = \sum_{o_i, o_j \in C_c} d_{ij}$$

  Let us notice that, for this criterion the dissimilarities are usually measured by the squared Euclidean distance.

If the objects o_i are in the p-dimensional Euclidean space, the homogeneity of a cluster C_c can also be measured by:

- the within cluster sum of squares $wcss(C_c)$ of a cluster C_c is the sum of the squared Euclidean distances between each object with the centroid m_c of the cluster:

$$wcss(C_c) = \sum_{o_i \in C_c} ||o_i - m_c||^2$$

  Let us notice that, when squared Euclidean distance is used for measuring the dissimilarities, the wcsd criterion once standardized via the division by the size of each group, is mathematically equivalent to the wcss:

$$wcss(C_c) = \frac{wcsd(C_c)}{|C_c|}$$

- the variance $variance(C_c)$ of a cluster C_c is the within cluster sum of squares divided by the cardinality of C_c:

$$variance(C_c) = \frac{wcss(C_c)}{|C_c|}$$

3.2.2 Criteria for a partition

The separation of a partition P can be measured either by the minimum separation of all clusters C_c with $c \in [1, n]$ or by the sum of the separation. This leads to these criteria:

- the split of a partition:

$$split(P) = \min_{c \in [1,k]} split(C_c)$$

- the sum of splits of a partition:

$$sum\_split(P) = \sum_{c=1}^{k} split(C_c)$$

Another criterion that is often used in spectral clustering is the min cut, which minimizes the cut between clusters.

- the cut of a partition is the sum of similarities of edges between cluster:

$$cut(P) = \frac{1}{2} \sum_{c=1}^{k} cut(C_c)$$

In case $k = 2$, the problem of finding min cut is polynomial and can be solved efficiently [Stoer & Wagner 1997], otherwise it is NP-Hard. In many cases, the criterion of min cut leads to a partition with imbalanced clusters, for example, one cluster with one vertex and one cluster with the rest. The criterion of ratio cut [Hagen & Kahng 1992] and normalized cut [Shi & Malik 2000] resolve this problem by introducing the size and the volume of the clusters in the formula.

- the ratio cut of a partition is measured as:

$$ratio\_cut(P) \quad = \quad \frac{1}{2} \sum_{c=1}^{k} \frac{cut(C_c)}{|C_c|}$$

- the normalized cut is measured as:

$$normalized\_cut(C_1, \ldots, C_k) \quad = \quad \frac{1}{2} \sum_{c=1}^{k} \frac{cut(C_c)}{vol(C_c)}$$

  where $vol(C_c)$ is the sum of similarities in C_c

$$vol(C_c) = \sum_{o_i, o_j \in C_c} s_{ij}$$

The criteria of ratio cut and normalized cut are NP-Hard, even with $k = 2$.

The homogeneity of a partition P can be measured either by the maximum homogeneity or the sum of homogeneity of each cluster. This leads to the following criteria:

- the maximum diameter:

$$diameter(P) = \max_{c \in [1,k]} diameter(C_c)$$

- the sum of diameters:

$$sum\_diameter(P) = \sum_{c=1}^{k} diameter(C_c)$$

- the within cluster sum of dissimilarities:

$$wcsd(P) = \sum_{c=1}^{k} wcsd(C_c)$$

- the within cluster sum of squares:

$$wcss(P) = \sum_{c=1}^{k} wcss(C_c)$$

Let us notice that, for some criteria, for instance the diameter criterion or the split criterion, this may exist different partitions that share the same optimal value. All these criteria lead to NP-Hard problems except the split criterion.

3.3 Partitioning methods

Given a dataset of n objects, a partitioning algorithm searches a partition with exactly k ($k \leq n$) clusters in which each object belongs to exactly one cluster. In general, a partitioning method optimizes a partitioning criterion. From the point of view of optimization, partitioning methods can be classified into heuristics and exact approaches. In this section, we present well-known partitioning methods that are related to our research.

3.3.1 Heuristics

3.3.1.1 The k-means and k-medoid algorithms

The k-means algorithm [MacQueen 1967] is the most well-known method for clustering for it is simple and effective. The algorithm is initialized by randomly picking k objects from $\mathcal{O}$ as the k initial cluster centroids m_1, m_2,..., m_k. Then each object is assigned to the closest centroid, forming a partition of the dataset. After that, the cluster centroids are updated and objects are reassigned again to the closest centroid. This process iterates until there is no change of assignments of objects, or equivalently, until there is no change of centroids. The k-means algorithm always converges to a solution where the centroids are stable.

Algorithm 2: k-Means algorithm

Input: dataset $\mathcal{O} = \{o_1, \ldots, o_n\}$, a number of clusters k
Output: a partition P with k clusters $C_1, C_2, \ldots, C_k$

```
Generate randomly k objects as initial cluster centers: m_1, m_2, ..., m_k
repeat
    foreach o_i ∈ O do
        //o_i is assigned to the closest centroid:
        Assign o_i to C_c such that d(o_i, m_c) = min_{j∈[1,k]} d(o_i, m_j)
    end
    //Recalculate the new centroid of each cluster
    foreach c ∈ [1, k] do
        m_c = (1/|C_c|) Σ_{o_i ∈ C_c} o_i
    end
until no change of cluster centroids;
Return partition P = {C_1, C_2, ..., C_k}
```

The k-means is efficient if clusters are well separated from each other. It can handle large datasets because the complexity of the algorithm is $O(nkt)$ where t is the number of iterations. However, this method can be applied only when the mean value of clusters can be computed. Moreover, this method is sensible to noises and outliers because those values affect the mean value of clusters. To reduce those effects, the k-medoid algorithm is proposed in [Kaufman & Rousseeuw 1987]. In this method, an actual object is chosen in each cluster as a representative object. This differs from k-means where the representative of a cluster is computed as the mean value of the objects of the cluster. It starts by choosing k objects as the initial cluster representatives. Each object is assigned to its nearest representative object. It then chooses a new representative object for each cluster and the method reassigned the objects by taking into account the new representative. The algorithm iterates until each representative object is actually the medoid. It is more robust to noise and outliers as compared to k-means.

3.3.1.2 The Furthest Point First algorithm

The method FPF (Furthest Point First) introduced in [Gonzalez 1985] relies on the maximum diameter criterion. In this method, each cluster has an object as the head and every object in a cluster satisfies the property of being closer to their head than to the other heads. The method starts by choosing an object as the head of the first cluster and assigns all objects to it. In the next iteration, the furthest object from the first head is chosen as the head of the second cluster. Any object which is closer to the second head than to the first one is moved to the second cluster. The algorithm iterates: choosing the object that is the furthest to its head as the head of the new cluster and reassigning objects. It stops after k iterations since k clusters are needed. The time complexity is $O(kn)$. Let d_{opt} be the global optimal, then this heuristic is guaranteed to find a clustering with a maximum diameter d such that $d_{opt} \leq d \leq 2d_{opt}$ if the dissimilarity measure satisfies the triangle inequality. Moreover, [Gonzalez 1985] showed that this approximation ratio is tight: finding d such that $d \leq (2 - \epsilon)d_{opt}$ is NP-hard for all $\epsilon > 0$ when objects are in a

Algorithm 3: k-Medoids algorithm for partitioning

Input: dataset $\mathcal{O} = \{o_1, \ldots, o_n\}$, a number of clusters k
Output: a partition P with k clusters $C_1, C_2, \ldots, C_k$

```
Randomly generate k objects as initial cluster medoids x_1, x_2, ... x_k
repeat
    foreach o_i ∈ O do
        //o_i is assigned to the closest medoid:
        Assign o_i to C_c such that d(o_i, x_c) = min_{j∈[1,k]} d(o_i, x_j)
    end
    Randomly select a non medoid object o_i
    Compute the cost S when swapping the object o_i with a representative x
    if S < 0 then
        o_i replace x as a new medoid
    end
until no change of cluster medoids;
```

three dimensional space.

Algorithm 4: FPF algorithm for partitioning

Input: dataset $\mathcal{O} = \{o_1, \ldots, o_n\}$, a number of clusters k
Output: a partition P with k clusters $C_1, C_2, \ldots, C_k$

```
Randomly select an object, mark it as head_1, the head of cluster 1
foreach i from 2 to k do
    Select the object that is the furthest to its head, mark it as head_i
    Assign every object o to cluster i if o is closer to head_i than to its head
end
```

3.3.1.3 Spectral methods

The main idea of spectral methods is to change the presentation of n objects x_i to y_i by the Laplacian graph. Then the methods use the classic k-means clustering algorithm to partition new objects y_i. Spectral methods use the graph of similarity $G = (V, E)$ where each vertex v_i represents an object o_i, two vertices are connected if the similarity between them is below a given threshold. G is a weighted graph where the weight of edges v_i and v_j, denoted w_{ij}, is the similarity s_{ij} between o_i and o_j. The graph G is undirected and the degree of a vertex v_i is measured as the sum of weights of the edges connected to v_i:

$$degree(i) = \sum_{j=1}^{n} w_{ij}$$

The degree matrix D is a diagonal matrix, where the diagonal contains the degrees of each vertex v_i of G

$$D_{ii} = degree(i), D_{ij} = 0 \quad \forall i \neq j$$

The spectral methods are used to solve the clustering problem with the criteria of min cut, ratio cut or normalized cut. The optimization problems with those criteria are NP-Hard. However, they can be relaxed by using spectral concepts of graph analysis. The relaxation is done by the Laplacian graph.

The unnormalized Laplacian graph is introduced in [Chung 1997] :

$$L = D - W$$

where W is the matrix of (w_{ij}).

The unnormalized Laplacian graph has the following properties:

- For every vector $f = (f_1, \ldots, f_n) \in \mathbb{R}^n$:

$$f^t L f = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (f_i - f_j)^2$$

- L is symmetric and positive semi-definite
- The smallest eigenvalue of L is 0, it corresponds to the constant one vector.
- L has n non-negative eigenvalues

$$0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$$

In the literature, there are different types of Laplacian graph and each Laplacian graph has special properties, suitable for different criteria. The basic spectral clustering algorithm is presented in Algorithm 5.

Algorithm 5: Spectral methods

1 Compute the similarities between objects $s_{ij}, \forall i, j \in [1, n]$
2 Compute the Laplacian graph
3 Compute the first k eigenvectors $u_1, \ldots, u_k$ of the Laplacian graph
4 Construct a matrix $U \in R^{n \times k}$ by those k eigenvectors
5 The new dataset consists of n objects, y_i is the vector corresponding to the i-th row of U
6 Apply the k-means method for the new n objects y_i

[Luxburg 2007] shows how the spectral method works with the criterion of minimizing the ratio cut, in the case $k = 2$, with the unnormalized Laplacian graph .

For a cluster C_c, we define the vector $f = (f_1, f_2, \ldots, f_n)^t$ with:

$$f_i = \sqrt{\frac{|\mathcal{O} \setminus C_c|}{|C_c|}}, \forall i : o_i \in C_c$$

and

$$f_j = -\sqrt{\frac{|C_c|}{|\mathcal{O} \setminus C_c|}}, \forall j : o_j \notin C_c \tag{3.1}$$

Due to the first property of the Laplacian graph , we have:

$$\begin{aligned}
f^t L f &= \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(f_i - f_j)^2 \\
&= \frac{1}{2}\sum_{o_i\in C_c}\sum_{o_j\notin C_c} w_{ij}\left(\sqrt{\frac{|\mathcal{O}\setminus C_c|}{|C_c|}} + \sqrt{\frac{|C_c|}{|\mathcal{O}\setminus C_c|}}\right)^2 \\
&+ \frac{1}{2}\sum_{o_i\notin C_c}\sum_{o_j\in C_c} w_{ij}\left(-\sqrt{\frac{|\mathcal{O}\setminus C_c|}{|C_c|}} - \sqrt{\frac{|C_c|}{|\mathcal{O}\setminus C_c|}}\right)^2 \\
&= \sum_{o_i\in C_c}\sum_{o_j\notin C_c} w_{ij}\left(\sqrt{\frac{|\mathcal{O}\setminus C_c|}{|C_c|}} + \sqrt{\frac{|C_c|}{|\mathcal{O}\setminus C_c|}}\right)^2 \\
&= cut(C_c)\left(\frac{|\mathcal{O}\setminus C_c|}{|C_c|} + \frac{|C_c|}{|\mathcal{O}\setminus C_c|} + 2\right) \\
&= cut(C_c)\left(\frac{|\mathcal{O}\setminus C_c| + |C_c|}{|C_c|} + \frac{|\mathcal{O}\setminus C_c| + |C_c|}{|\mathcal{O}\setminus C_c|}\right) \\
&= n \times ratio_cut(C_c)
\end{aligned}$$

We also have:

$$\begin{aligned}
\sum_{i=1}^{n} f_i &= \sum_{o_i\in C_c}\sqrt{\frac{|\mathcal{O}\setminus C_c|}{|C_c|}} - \sum_{o_i\notin C_c}\sqrt{\frac{|C_c|}{|\mathcal{O}\setminus C_c|}} \\
&= |C_c|\sqrt{\frac{|\mathcal{O}\setminus C_c|}{|C_c|}} - |\mathcal{O}\setminus C_c|\sqrt{\frac{|C_c|}{|\mathcal{O}\setminus C_c|}} \\
&= 0
\end{aligned}$$

The vector f is orthogonal to the constant vector 1, and it satisfies:

$$||f||^2 = \sum_{i=1}^{n} f_i^2 = |C_c|\frac{|\mathcal{O}\setminus C_c|}{|C_c|} + |\mathcal{O}\setminus C_c|\frac{|C_c|}{|\mathcal{O}\setminus C_c|} = n$$

The problem of minimizing the ratio cut is equivalent with:

$$\min_{C_c} f^t L f$$

subject to:

$$\text{f defined in equation 3.1}, f\perp 1, ||f|| = \sqrt{n}$$

This is a discrete optimization problem and it is NP-hard. This problem can be relaxed allowing f_i to take arbitrary values in $\mathbb{R}$. The problem becomes a relaxed optimization problem:

$$\min_{C_c} f^t L f$$

subject to:

$$f\perp 1, ||f|| = \sqrt{n}$$

The solution of this relaxed optimization problem is given by the eigenvector of the second smallest eigenvalue of L. To obtain the partition corresponding to the approximate ratio cut, the vector f must be transformed into a discrete vector. A simple way for the transformation is:

$$o_i \in C_c \;\; if \;\; f_i \geq 0, o_i \notin C_c \;\; if \;\; f_i < 0$$

In the case $k > 2$, a better way for the transformation is to consider f_i as points and then to cluster them by k-means algorithm (as presented in Algorithm 5).

The main advantages of spectral methods are that they are not affected by particular forms of clusters whereas other methods, for example, k-means algorithm tends to find convex sets. It finds clusters based on the connectivity, not on the compactness like k-means. It can solve rather large datasets because it solves a linear problem.

3.3.2 Exact approaches

3.3.2.1 Graph-theoretical algorithm

Graphs are formed by a set of vertices and a set of edges, connecting pairs of vertices. If we consider objects as vertices and using the dissimilarities as the edges between them, the clustering analysis is close to the graph theory. The problem of finding an optimal partition with the criterion of maximum diameter is proved to be NP-Hard and an exact method based on the problem of graph colouring is proposed in [Hansen & Delattre 1978], presented in Algorithm 6. The algorithm is based on the observation that: there are at most $n(n-1)/2$ distinct values of dissimilarities. Suppose that we have a partition $\mathcal{P}$ with k clusters and d_{max} is the maximum diameter. It means that every pair of objects (o_i, o_j) in which $d_{ij} > d_{max}$ must be in different groups. Let $G = (V, E)$ be a graph where V has n vertices corresponding to the n objects $o_1, o_2, \ldots, o_n$, and E contains edges (o_i, o_j) satisfying $d_{ij} > d_{max}$. It is evident that the graph G can be coloured into k colours. Indeed, from the partition $\mathcal{P}$, we can use the same colour for all objects of the same cluster, thus we need at most k colours for k clusters.

Algorithm 6: graph colouring approach for clustering with the diameter criterion

```
Compute the matrix of dissimilarity D = (d_ij)
Build the initial graph G = (V, E) where V contains n objects and E is empty
d_max ← max_{i,j∈[1,n]} d_ij
repeat
    Add edges (o_i, o_j) in which d_ij = d_max to E
    Compute the number of colors N_color needed to colour the graph G
    if N_color ≤ k then
        d_max ← the next largest dissimilarity
    end
until number of colors needed > number of clusters k;
Optimal diameter ← d_max
```

3.3.2.2 Branch-and-bound algorithms

As the problem is defined as assigning n objects into k different clusters, with a criterion to optimize, branch-and-bound approaches can be useful to solve the problem. The performance of a branch-and-bound method depends on how the algorithm calculates the bounds and on the branching strategy. A branch-and-bound algorithm for the criterion of the within cluster sum of squares has been proposed in [Koontz *et al.* 1975]. It was improved later in [Brusco 2006]. For the criterion of within cluster sum of dissimilarities, a branch-and-bound algorithm has been proposed in [Klein & Aronson 1991], and in [Brusco & Stahl 2005]. For the criterion of minimum diameter, a method has been proposed in [Brusco 2003].

The method of branch-and-bound is based on the evaluation of partial (incomplete) partitions. A partial partition Q is an assignment of p objects to k clusters and it remains $n - p$ unassigned objects. It may exist empty clusters in a partial partition.

Depending on the criteria, different bounds can be used to evaluate the partial solution. The basic BAB algorithm which uses a depth-first-search strategy is presented in Algorithm 7, supposing that the objective is to minimize a criterion.

Algorithm 7: Branch-and-bound algorithm for clustering with Depth-First-Search

```
A heuristic is used to obtain a good solution
The upper bound is set to the criterion value of the solution
number of assigned objects p ← 0
DepthFirstSearch(p)
Function DepthFirstSearch(p)
    if p = n then
        Update the optimal solution
        Update the upper bound
        Return
    end
    p ← p + 1
    foreach cluster c ∈ [1, k] do
        Assign object o_p to cluster C_c
        Evaluate the partial solution with p objects
        if the partial solution may be a part of the optimal solution then
            DepthFirstSearch(p)
        end
    end
    Return
```

The real challenge is how to efficiently evaluate the partial solutions in order to prune useless branches in the search tree. To determine whether a partial solution may be a part of the optimal solution, a typical way is to compute a lower bound of the criterion. If the lower bound is greater than the upper bound, then the partial solution cannot be a part of the optimal solution and the search backtracks to explore other branches.

For the criterion of diameter [Brusco 2003] uses three tests to evaluate the partial solution:

- The first test is used to check if there are enough unassigned objects with respect to the remaining empty clusters.
- The second test checks if the diameter of the partial solution is equal to or greater than the maximal diameter of the current best solution, based on assigned objects.
- The third test is based on unassigned objects, it tests if there exists an object that cannot be assigned to a cluster without leading to a new diameter equal to or greater than the maximal diameter of the current best solution.

In any of these three tests return true, there is no need to continue with this partial solution and the search backtracks.

For the criterion of the within cluster sum of dissimilarities The branch-and-bound algorithm in [Klein & Aronson 1991] computes a lower bound of wcsd for the partial solutions. The lower bound of wcsd is computed as a sum of three parts: the first part is the sum of dissimilarities of assigned objects, the second part is the sum of dissimilarities between assigned objects and unassigned ones and the third part is the sum of dissimilarities between unassigned objects.

As the first part is the sum of dissimilarities of assigned objects, it can be computed exactly. The second and the third part cannot be calculated exactly and a lower bound for these parts are measured by heuristics.

[Brusco & Stahl 2005] proposed to improve the lower bound of wcsd by computing the optimal wcsd of unassigned objects for the third part instead of using a heuristic. [Brusco & Stahl 2005] solves the problem in a Repetitive Branch-and-Bound Algorithm (RBBA). Given a dataset of n point $o_1, \ldots, o_n$ and a clustering task dividing the points into k clusters, the RBBA is realized in $n - k$ steps:

- Step 1: Find the optimal wcsd of $k + 1$ objects $\{o_{n-k}, o_{n_k+1}, \ldots, o_n\}$ into k clusters with a branch-and-bound algorithm. The search strategy assigns objects into clusters following the order: $o_{n-k}, \ldots, o_n$.
- Step 2: Find the optimal wcsd of $k + 2$ objects $\{o_{n-k-1}, o_{n_k}, \ldots, o_n\}$.
- ...
- Step $n - k$: Find the optimal wcsd with n objects $\{o_1, \ldots, o_n\}$.

Let us consider the step i that finds the optimal wcsd of $k + i$ objects $\{o_{n-k-i+1}, \ldots, o_n\}$ into k clusters with a branch-and-bound algorithm. The search strategy assigns objects into clusters in the order: $o_{n-k-i+1}, \ldots, o_n$. Considering a partial solution with p assigned objects, it is clear that the assigned objects are $\{o_{n-k-i+1}, \ldots, o_{n-k-i+p}\}$ and the unassigned objects are $\{o_{n-k-i+p+1}, \ldots, o_n\}$. The optimal wcsd of objects $\{o_{n-k-i+p+1}, \ldots, o_n\}$ into k clusters is obviously already computed in one of the previous steps, therefore the third part can be obtained optimally and the lower bound of the partial solution is evaluated better. Figure 3.1 illustrates the steps of RBBA algorithm for solving the wcsd clustering with 7 objects $o_1, \ldots, o_7$ into 2 clusters.

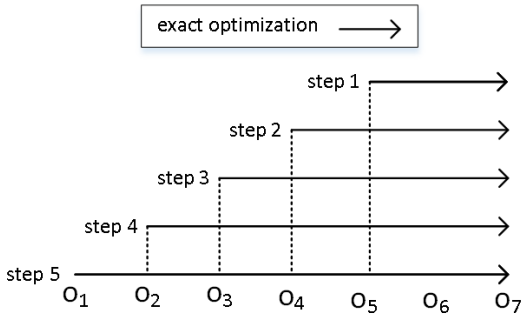

Figure 3.1: Illustration of steps of repetitive branch-and-bound algorithm

For the criterion of within cluster sum of squares The RBBA algorithm can be applied to the wcss criterion [Brusco 2006, Carbonneau *et al.* 2012]. The basic idea is the same. For this criterion, a lower bound of a partial solution can be measured by the sum of wcss of assigned objects and the optimal wcss of unassigned objects. The RBBA solves the problem first with $k+1$ objects, then with $k+2$ objects and so on, until all n objects are considered.

3.4 Hierarchical clustering methods

Hierarchical clustering aims at finding a hierarchy of clusters. Unlike partitioning methods, hierarchical clustering finds a sequence of nested partitions. The result is a tree diagram which is called a dendrogram. There are two types of hierarchical clustering:

- Agglomerative: a "bottom up" approach, it begins with n clusters, each cluster contains one and only one object. At each iteration, two "close" clusters are merged until a stopping criterion is satisfied.

- Divisive: a "top down" approach, it begins with only one cluster that contains all the n objects. It successively splits one cluster into two smaller parts until a stopping criterion is satisfied.

The agglomerative hierarchical clustering algorithms start by a partition P_n with n clusters C_1 to C_n, each cluster C_i contains exactly one object o_i. The basic algorithm is presented in Algorithm 8. At each iteration, the algorithm selects two close clusters based on a criterion and merges them until there is only one cluster. The most popular agglomerative hierarchical clustering algorithms are single-linkage and complete-linkage algorithms. The main difference between those two algorithms is how they calculate the distance between two clusters. In the single-linkage algorithm, the distance between two clusters is the minimum of the distances between any pairs of objects of the two clusters whereas in the complete-linkage algorithm, it is the maximum of the distances of all pairs of objects in the two clusters. The single-link algorithm may suffer from the chain effect [Johnson 1967], *i.e.* a chain of close objects may group very different objects in the same cluster whereas the complete-link cluster may produce compact clusters but not well-separated.

It is proved in [Delattre & Hansen 1980] that at all levels, the single-linkage algorithm provides an optimal partition with respect to the criterion of split, so it can be used to find the exact split with any number of clusters k.

Algorithm 8: Basic agglomerative hierarchical clustering algorithm

```
Construct a partition with n clusters P_n = {C_1, C_2, ..., C_n}, C_i = {o_i}
foreach t ∈ [1, n − 1] do
    Select C_i, C_j ∈ P_{n−t+1} based on a criterion
    C_{n+t} = C_i ∪ C_j
    P_{n−t} = (P_{n−t+1} ∪ C_{n+t}) \ {C_i, C_j}
end
```

In contrast to the agglomerative algorithms, the basic divisive hierarchical clustering, presented in Algorithm 9 starts by a partition P_1 which contains only one cluster C_1 with the whole objects and it divides a cluster based on a criterion, until there are n clusters.

Algorithm 9: Basic divisive hierarchical clustering algorithm

```
Construct a partition with only one cluster P_1 = {C_1}, C_1 = {o_1, o_2, ..., o_n}
foreach t ∈ [1, n − 1] do
    Select C_i ∈ P_t based on a criterion
    Divide C_i into C_{2t} and C_{2t+1}
    P_{t+1} = (P_t ∪ C_{2t} ∪ C_{2t+1}) \ C_t
end
```

3.5 Constrained Clustering

In order to better model the task, but also in the hope of reducing the complexity, user-specified constraints are added, leading to Constrained Clustering that aims at finding clusters that satisfy user-specified constraints. User constraints can be classified into cluster-level constraints, specifying requirements on clusters, or instance-level constraints, specifying requirements on pairs of objects.

Instance-level constraints (also called pairwise constraints) are constraints on pairs of objects. There are two types of instance-level constraints, introduced for the first time in [Wagstaff & Cardie 2000], the must-link and cannot-link constraints. A must-link constraint between two object o_i, o_j, denoted by $ML(o_i, o_j)$, expresses that both objects o_i and o_j must be in the same cluster. In contrast, a cannot-link constraint between o_i and o_j, denoted by $CL(o_i, o_j)$, means that these two objects must not be in the same cluster. According to [Davidson & Basu 2007], the instance-level constraints have the following properties:

- Must-link constraints are transitive: Let CC_a and CC_b be connected components (completely connected subgraphs by must-link constraints), let o_i and o_j be objects

Must-link	P
Cannot-link	NP-Complete
ϵ-constraint	P
δ-constraint	P
Must-link and ϵ constraint	NP-Complete
Must-link and δ constraint	P
ϵ and δ constraints	P

Table 3.1: Complexity of finding a feasibility partition with different constraints [Davidson & Ravi 2007]

in CC_a and CC_b respectively, then:

$$ML(o_i, o_j), o_i \in CC_a, o_j \in CC_b \Rightarrow ML(o_x, o_y), \forall o_x, o_y : o_x \in CC_a, o_y \in CC_b$$

- Cannot-link constraints can be entailed. Let CC_a and CC_b be connected components (completely connected subgraphs by must-link constraints), let o_i and o_j be objects in CC_a and CC_b respectively, then:

$$CL(o_i, o_j), o_i \in CC_a, o_j \in CC_b \Rightarrow CL(o_x, o_y), \forall o_x, o_y : o_x \in CC_a, o_y \in CC_b$$

The idea of must-link and cannot-link constraints might seem simple but in fact, those constraints are powerful in many applications. Many researchers have reported that increasing the number of pairwise constraints may improve the accuracy of the result. The must-link and cannot-link constraints can also be used to express other user-constraints. [Davidson & Ravi 2005b] has introduced ϵ-constraint and δ-constraint which can be handled with pairwise constraints:

- δ-constraint (also called minimum split constraint) expresses that the distance between any pair of points which are in two different clusters must be at least δ. It can be handled by a conjunction of must-link constraints between all pair of objects with distance less than δ.

- ϵ-constraint expresses that: for any cluster C_c containing more than one object, for any object $o_i \in C_c$, there must be another object $o_j \in C_c$ such that the distance between o_i and o_j does not exceed ϵ: $d_{ij} \leq \epsilon$. This constraint tries to capture the notion of density, introduced in DBSCAN [Ester *et al.* 1996]. It is shown in [Davidson & Ravi 2005b] that this constraint is equivalent with a disjunction of must-link constraints. For each point o_i, a set $\mathcal{X}$ of point o_j such that $d_{ij} \leq \epsilon$ is computed, then a disjunction of must-link constraints between o_i and points in $\mathcal{X}$ are generated.

[Davidson & Ravi 2007] show that finding feasibility partition with those constraints is not an easy task. Table 3.1 expresses the complexity of finding a feasibility partition with different constraints.

Another constraint that can be expressed by instance-level constraints is the maximum diameter constraint. This constraint specifies an upper bound γ on the diameter of the

clusters to express that any pair of points which are in the same cluster must be at a distance of at most γ. This constraint can be handled by conjunction of cannot-link constraints between all pair of objects with distance greater than γ.

3.5.1 Clustering algorithms for instance-level constraints

In the last ten years, many works have been done to extend classical algorithms for handling must-link and cannot-link constraints, as for instance an extension of COBWEB [Wagstaff & Cardie 2000], k-means [Wagstaff *et al.* 2001, Bilenko *et al.* 2004], hierarchical non supervised clustering [Davidson & Ravi 2005a] or spectral clustering [Lu & Carreira-Perpinan 2008, Wang & Davidson 2010], ...
Those methods can be classified into two general approaches [Davidson & Ravi 2007]: constraint-based and distance-based methods. In constraint-based approaches, the clustering algorithm is modified to integrate pairwise constraints whereas in distance-based approaches, only the distance measure is modified.

3.5.1.1 Constraint-based methods

[Wagstaff *et al.* 2001] proposed a modified k-means algorithm, called COP-Kmeans to integrate must-link and cannot-link constraints. The major modification is that, at each iteration, objects are updated so that none of the constraints is violated. The drawback of this approach is that it tries to satisfy all the constraints but the algorithm does not provide any backtracking technique. As a result, the algorithm may fail to find a partition even if one exists. It happens when there are many constraints, especially cannot-link constraints.

Algorithm 10: COP-Kmeans algorithm

```
1  Generate randomly k objects as initial cluster centroids
2  repeat
3      foreach object o_i do
4          Assign object o_i to the closest cluster C_c such that:
                                  ∄o_j : o_j ∈ C_c ∩ CL(o_i, o_j)
                                  ∄o_j : o_j ∉ C_c ∩ ML(o_i, o_j)
5          if no such cluster C_c exists then
6              Return FAILED
7          end
8      end
9      Recalculate the new centroid of each cluster as the mean value of the objects in
       each cluster
10 until no change of cluster centroid;
```

To overcome the disadvantage of COP-Kmeans algorithm, an approach is to try not to satisfy all the constraints, but most of them. The CVQE (constrained vector quantization error) algorithm [Davidson & Ravi 2005b] extends the K-means algorithm to incorporate

must-link and cannot-link constraints by modifying the objective function. In K-means algorithm, at each iteration, the centroid m_c is updated as:

$$m_c = \frac{\sum_{o_i \in C_c} o_i}{|C_c|}$$

After that, each object is reassigned to the closest centroid to minimize the vector quantization error (VQE):

$$VQE = \frac{1}{2} \sum_{c \in [1,k]} \sum_{o_i \in C_c} ||m_c - o_i||^2$$

The CVQE algorithm modifies the error function to penalize the violated constraints, [Davidson & Ravi 2005b] defines the constrained vector quantization error $CVQE_c$ of a cluster C_c by:

$$\begin{aligned} CVQE_c &= \frac{1}{2} \sum_{o_i \in C_c} (m_c - o_i)^2 \\ &+ \frac{1}{2} \sum_{o_i \in C_c, ML(o_i,o_j), o_j \in C'_c, c' \neq c} ||m_c - m'_c||^2 \\ &+ \frac{1}{2} \sum_{o_i \in C_c, CL(o_i,o_j), o_j \in C_c} ||m_c - m_{h(c)}||^2 \end{aligned}$$

where $h(c)$ returns index of the cluster (other than c) whose centroid is the closest to m_c.

In this equation, the first part is the VQE criterion as for the k-means algorithm. The second part is the penalty for the violation of must-link constraints. If a constraint $ML(o_i, o_j)$ is violated, the penalty is measured as the square distance between the cluster centroids containing objects of the constraint. The third part is the penalty for cannot-link constraints, it is measured as the squared distances between the centroid containing both objects and the nearest centroid.

The objects that appear in no constraint are always assigned to the nearest centroid. For the other objects, they are assigned to the centroid so as to minimize the CVQE.

The update rule of centroids are:

$$m_c = \frac{\sum_{o_i \in C_c} o_i + \sum_{ML(o_i,o_j), o_j \in C_{c'}, c' \neq c} m_{c'} + \sum_{CL(o_i,o_j), o_j \in C_c} m_{h(c)}}{|C_c| + \sum_{ML(o_i,o_j), o_j \in C_{c'}, c' \neq c} 1 + \sum_{CL(o_i,o_j), o_j \in C_c} 1}$$

The meaning of the updating on centroids is as follows: if a must-link constraint $ML(o_i, o_j)$ is violated, the centroid m_c of the cluster containing o_i moves closer to the centroid m'_c of the cluster containing o_j. Similarly, the violation of a cannot-link constraint $CL(o_i, o_j)$ makes the centroid containing both constrained objects move to the nearest cluster centroid.

[Pelleg & Baras 2007] proposed LCVQE, an improved algorithm of CVQE, by changing the penalty for violating constraints. For the violation of a constraint $ML(o_i \in C_c, o_j \in C_{c'})$, suppose that o_i is assigned before o_j, the penalty is now the squared distance between o_j and m_c. For the violation of a constraint $CL(o_i \in C_c, o_j \in C_c)$, the point that is further to m_c is chosen. The penalty is the squared distance between this point to the nearest centroid to m_c. The value LCVQE of each cluster c is measured between o_i and o_j as:

$$LCVQE_c = \frac{1}{2}\sum_{o_i \in C_c} (m_c - o_i)^2 + \frac{1}{2}\sum_{o_i \in C_c, ML(o_i,o_j), o_j \notin C'_c, c' \neq c} ||m_c - m'_c||^2 + \frac{1}{2}\sum_{o_i \in C_c, CL(o_i,o_j), o_j \in C_c} ||m_c - m_{h(c)}||^2$$

By modifying the formulation, the algorithm LCVQE has better computation time.

3.5.1.2 Distance-based algorithms

Another way to incorporate instance-level constraints is to modify the distance metric or the objective function. The basic idea is that if there is a must link constraint between two objects o_i and o_j, the distance between them might be smaller than usual, so they have a higher chance to be in the same cluster, similarly to the cannot-link constraints.

[Xing *et al.* 2003] proposed an algorithm to learn a distance metric over must-link and cannot-link constraints. The distance metric learning is formulated as an optimization problem. The objective function is the sum of distances of pairs in the must-link constraints. The algorithm searches to minimize this sum, under the condition that the distances of pairs in the cannot-link constraints are greater than a constant. The distance metric is of the form:

$$d(o_i, o_j) = d_A(o_i, o_j) = ||o_i - o_j||_A = \sqrt{(o_i - o_j)^T A (o_i - o_j)}$$

A is semi-definite positive and if $A = I$, we have the Euclidean distance. The problem is now to find the matrix A:

$$\min \sum_{ML(o_i,o_j)} d_A^2(o_i, o_j)$$

with

$$\sum_{CL(o_i,o_j)} d_A(o_i, o_j) \geq 1$$

$$A \succeq 0$$

The optimization problem is convex and differentiable and it is possible to find a global optimum in polynomial time.

3.5.2 Clustering algorithms for cluster-level constraints

Cluster-level constraints specify requirements on clusters. Beside the ϵ-constraint (minimum split constraint), δ-constraint and maximum diameter constraints, other examples of cluster-level constraints are:

- The minimum capacity constraint requires that each cluster has a number of objects greater than a given threshold α: $\forall c \in [1, k],\ |C_c| \geq \alpha$
- The maximum capacity constraint requires each cluster to have a number of objects inferior to a predefined threshold β: $\forall c \in [1, k],\ |C_c| \leq \beta$.

- The balance constraint expresses that all clusters have approximately the same size, or the fraction between the size of smallest and largest cluster must be greater than a given threshold θ: $\frac{\min_{i\in[1,k]}|C_i|}{\max_{j\in[1,k]}|C_j|} \geq \theta$

[Banerjee & Ghosh 2006] proposed a scalable framework for clustering to incorporate the balance constraint. The framework consists of three steps:

- sampling the data into a small representative subset of the points,
- clustering of the sampled data,
- populating the initial clusters with the remaining data.

The first step is for the scalability of the framework. In the second step, any clustering algorithm can be used. The third step consists of populating and refining the clusters. In populating phase, the remaining objects are assigned to the existing clusters such that the constraint is satisfied while ensuring good quality clusters. In refining phase, objects are re-assigned to improve the objective function.

[Demiriz *et al.* 2008] proposed an approach to integrate the minimum capacity constraint to k-means algorithm, to avoid empty clusters or clusters with few points in the result. [Demiriz *et al.* 2008] modifies the cluster assignment step of k-means algorithm to introduce the minimum capacity constraint, then solves the constrained problem by using linear programming or network simplex methods.

3.6 Summary

In this chapter, we made a brief introduction to clustering. We presented important algorithms, mainly from an optimization viewpoint, with a focus on those related to our research. Afterwards, we presented basic notions of constrained clustering and several well-known approaches.

Chapter 4

Declarative Data Mining

Ce chapitre est consacré à l'état de l'art des approches déclaratives en fouille de données, notamment les approches récentes concernant le domaine du clustering. Un bref aperçu des approches déclaratives est donné. Nous introduisons ensuite les travaux utilisant la programmation par contraintes [Khiari *et al.* 2010, Guns *et al.* 2013] pour résoudre le problème de recherche de k motifs fréquents. Ces cadres sont appliquées au problème de clustering conceptuel. Ce chapitre introduit ensuite différents travaux utilisant la satisfiabilité des formules booléennes (SAT) [Davidson *et al.* 2010, Metivier *et al.* 2011, Berg & Jarvisalo 2013] et la programmation linéaire [Babaki *et al.* 2014] pour résoudre la tâche de clustering sous contraintes.

Contents

A large number of approaches have been developed for solving data mining problems. Most researches aim at highly optimized and scalable algorithms, tailored towards specific tasks, in imperative paradigm. Researchers express algorithms in a way that machine understand how to solve the problems. In return, they expect that the machine will find out what they want. Recently, there has been a growing interest in declarative programming for data mining [Raedt *et al.* 2011]. According to [Lloyd 1994]: "A declarative programming involves stating what is to be computed, but not necessarily how it is to be computed". Here, researchers tell the machine what they want and the machine will figure out how to do it. Although declarative framework may be less efficient than tailored algorithms for a specific task, a declarative framework is way more flexible, easy to incorporate new knowledge to the task [Raedt *et al.* 2011]. Table 4.1 presents a comparison between Declarative and Imperative approach.

To solve a problem in a declarative paradigm, users specify it in a modelling language and a solver automatically generates the solutions. Declarative frameworks can be expressed in Constraint Programming, Propositional Satisfiability, Integer Linear Programming or Logic Programming. The real challenge is how to encode the problem effectively.

Declarative Approach	Imperative Approach
Focus on what the problem is	Express steps to solve the problem
High-level modelling	Low-level control
Less fast and less scalable	Fast and efficient
General, Flexible and Reusable	Specific for one problem

Table 4.1: Comparison of Declarative and Imperative Approaches (from a lecture of Tias Guns)

In this Chapter, we present state of the art of recent declarative frameworks for Data Mining, specially researches that are related to cluster analysis.

4.1 Constraint Programming framework for k-pattern set mining

Pattern mining is a data mining method that involves finding all existing patterns of a database that satisfy a set of constraints. A well-known example of pattern mining is finding frequent itemsets in market basket analysis. We consider each product is an item and a group of items is an itemset. Transactions are instances of products that are bought by each customer. An itemset, such as milk and bread, is frequent if these products appear frequently together in a transaction dataset.

[De Raedt *et al.* 2008] introduces a framework in Constraint Programming for itemset mining, which is flexible and allows to combine different mining constraints. [Khiari *et al.* 2010] proposes a generic approach to express and mine n-ary patterns, i.e. partterns involves several local pattern. [Guns *et al.* 2013] introduces the problem of k-pattern set mining that finds a set of k related patterns under constraints. The set of constraints consists of local constraints, *i.e.* constraint on individual pattern, and global constraints, *i.e.* constraints on the overall pattern set.

[Guns *et al.* 2013] proposes a framework in Constraint Programming for k-pattern set mining. The framework is similar to [Khiari *et al.* 2010], but it covers wider range of tasks. The framework can be applied in various well-known mining tasks, including conceptual clustering.

The principle of conceptual clustering is not limited to find a partition of the data, but it has to generate a conceptual description for each cluster. A cluster concept is a set of properties that every object of the cluster has. If objects are transactions, the description of a cluster can be represented by a set of items and the conceptual clustering can be considered as a k-pattern set mining problem under specific constraints.

To solve the problem of k-pattern set mining, [Guns *et al.* 2013] proposes a general framework in Constraint Programming. Considering a dataset of n items and m transactions, the goal is to find k-patterns $\pi_1, \ldots, \pi_k$ that satisfy all local and global constraints. The model uses $k \times n$ binary variables $I[p,i] \in \{0,1\}$ for items and $k \times m$ variables $T[p,t] \in \{0,1\}$ for transactions ($p \in \{1,\ldots,k\}, i \in \{1,\ldots,n\}, t \in \{1,\ldots,m\}$). The variable $I[p,i] = 1$ indicates that item i is in the pattern π_p. Similarly, the variable $T[p,t] = 1$ indicates that transaction t is in the pattern π_p. By using Constraint Programming, the

model is flexible to express different kinds of constraints on items, on transactions, on patterns and on an objective function:

- Coverage constraint: the k patterns are disjoint.
- The k patterns cover all transactions.
- Each pattern is closed: the itemset must be the largest itemset that is contained in all transactions of the pattern.
- Constraint on the size of the pattern which is measured by the size of the itemset.
- Constraint on the difference between the sizes of the patterns.
- Constraint on the number of transactions in each pattern.
- Constraint on the overlap: it allows a percentage of transactions that can be in more than one pattern/cluster.

The model integrates different objective functions: maximizing the minimum cluster size or minimizing the difference between cluster sizes. Although the proposed CP model, when applied to cluster analysis, is limited to transaction database and conceptual clustering, the model is flexible in modeling constraints and objective functions.

4.2 SAT based framework for clustering

In propositional logic, a propositional variable x_i is a boolean variable, which takes a truth value 0 (false) or 1 (true). A literal l_i is a variable x_i or its negation $\overline{x_i}$, a clause C_i is a disjunction of literals l_i, for example: $C_1 = x_1 \vee x_2 \vee \overline{x_3}$. A conjunctive normal form (CNF) formula is a conjunction of clauses C_i, for example: $C_1 \wedge C_2 \wedge C_3$. A literal x_i (resp. $\overline{x_i}$) is satisfied if x_i gets value 1 (resp. 0). A clause is satisfied if at least one of its literals is satisfied, and a CNF is satisfied if all the clauses are satisfied. The SAT problem consists of finding an assignment of truth values to variables that satisfies a CNF formula. A 2-SAT problem is a SAT problem in which each clause involves no more than 2 variables. In MaxSAT problem, the objective is to satisfy the maximum number of clauses. A partial MaxSAT instance is a CNF formula which contains soft (relaxable) and hard (non-relaxable) clauses. Solving a partial MaxSAT problem consists of finding an assignment that satisfies all hard clauses and the maximum number of soft clauses. The SAT problem was the first known example of an NP-complete problem (except for the 2-SAT problem, which has a polynomial time algorithm). However, modern SAT solvers can efficiently solve problems containing a large number of clauses.

4.2.1 A Constraint Language based on SAT for Declarative Pattern Mining

[Metivier *et al.* 2011] presents a constraint-based language that can be used for modelling different pattern mining problems. The general idea is similar to the framework of [Guns *et al.* 2013], but the resolution is completely different. A pattern mining problem can be defined as queries of the language. A query is a conjunction of constraints

and all primitive constraints of the language are modelled using the SAT framework. The problem then can be solved by a SAT solver. In [Métivier *et al.* 2012], the authors illustrate how to use the language for the problem of clustering on transaction datasets. Each cluster is a pattern that consists of a set of transactions. The problem of finding k patterns/clusters $C_1, \ldots, C_k$ can be expressed by the conjunction of the following constraints in the language:

- $\wedge_{i \in [1,k]}$ isNotEmpty(C_i): each pattern must contain at least one transaction.
- coverTransaction($C_1, \ldots, C_k$): each transaction is covered by at least one pattern.
- noOverlapTransactions($C_1, \ldots, C_k$): there is no transaction, which is covered by more than one pattern.
- canonical($C_1, \ldots, C_k$): the pattern is in lexicographic order. This constraint is for symmetry breaking.

The language also supports must-link, cannot-link and size constraints, which allow users to handle different constrained clustering tasks, by changing or adding constraints to the modelling. A SAT solver then can find some or all partitions that satisfy all the queries. The SAT encoding of the language integrates features of SAT solvers to improve the performance, for example: binary clauses, unit propagation, sorting networks. The experiments show that with this encoding, SAT solvers can solve efficiently the clustering task with medium size datasets. For example, with the MiniSat solver, the first ten solutions are found quickly for the dataset Mushroom with 8124 transactions. In conclusion, this constraint-based language is declarative and flexible, but when applied to constrained clustering, it is limited to itemset data and does not integrate optimization criteria.

4.2.2 A SAT framework for constrained clustering with 2 clusters

[Davidson *et al.* 2010] proposed a 2-SAT framework for constrained clustering, allowing finding a partition with 2 clusters that optimizes the diameter criterion or the split criterion. For the modelling, [Davidson *et al.* 2010] uses n boolean variables x_i; x_i is assigned to 1 (0) if the object o_i is affected to cluster 1 (0). The assignment of truth values to these n variables gives us a partition that consists of two clusters.

With those variables, the instance level constraints can be easily expressed by CNF formulas. A must-link constraint between objects o_i and o_j is expressed by:

$$(\overline{x}_i \vee x_j) \wedge (x_i \vee \overline{x}_j)$$

Similarly, a cannot-link constraint between objects o_i and o_j is expressed by :

$$(x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j)$$

The diameter constraint α can be handled by adding cannot-link constraints to every pair of points o_i and o_j such that their distance exceeds α. With this modelling, a diameter constraint is expressed by adding two clauses $(x_i \vee x_j) \wedge (\overline{x}_i \vee \overline{x}_j)$ for every pair o_i and o_j such that $d_{ij} > \alpha$. Similarly, a split constraint β is handled by adding two clauses $(\overline{x_i} \vee x_j) \wedge (x_i \vee \overline{x_j})$ to every pair o_i and o_j such that $d_{ij} \leq \beta$. With these CNF formula,

a SAT solver can be used to find a partition with 2 clusters that satisfies the must-link, cannot-link, diameter and split constraints. To find a partition that optimizes the diameter criterion (resp. the split criterion), [Davidson *et al.* 2010] uses repeated calls to solve the SAT formula. It is clear that the optimal value of diameter (split) criterion must be one of the pairwise dissimilarities. Therefore, the optimal value can be found by a binary search. For example, with the diameter criterion, at each iteration, a dissimilarity x is chosen for a constraint on the diameter ($diameter \leq x$), then a SAT formula is generated. Then a SAT solver is used to find the assignment of truth values to variables. The assignment gives us a partition that satisfies all the constraints, including the diameter constraint. If such a solution exists, the optimal diameter is less or equal to the dissimilarity x, otherwise the optimal diameter is greater than x. Suppose that there are m different dissimilarities, the algorithm takes at most $O(log_m)$ calls to find the optimal diameter.

The approach is somewhat similar to the algorithm of [Hansen & Delattre 1978]: [Davidson *et al.* 2010] uses a SAT formula to test if a dissimilarity can be the optimal diameter while [Hansen & Delattre 1978] uses a graph coloring method to verify that.

This framework is the first modelling of a clustering problem as an instance of SAT. It is limited to 2 clusters and the optimization criterion is not modelled but is expressed by the search strategy, which repeatedly generates the SAT formula and then calls a SAT solver.

4.2.3 MaxSAT framework for Optimal Constrained Correlation Clustering

Correlation clustering [Bansal *et al.* 2004] is a clustering method that finds the partition based on the relationships between the objects, instead of the actual representations of the objects. Correlation clustering uses a boolean similarity measure s_{ij} between two objects o_i and o_j: $s_{ij} = 1$ if o_i and o_j are similar and $s_{ij} = 0$ if they are dissimilar. In correlation clustering, the objective is to minimize the cost function:

$$\sum_{i,j:\lambda_i=\lambda_j} (1 - s_{ij}) + \sum_{i,j:\lambda_i\neq\lambda_j} s_{ij}$$

where λ_i denotes the index of the cluster the object o_i is assigned to.

The goal of correlation clustering is to find the partition that agrees as much as possible with the similarity measure. This criterion does not require to specify the number of clusters k as a parameter, as the optimal number of clusters could be any value between 1 and n [Bansal *et al.* 2004]. Finding a partition that optimizes this criterion is NP-hard. [Berg & Jarvisalo 2013] presents a partial MaxSAT framework for finding a global optimal solution for Correlation Clustering. The Correlation Clustering is encoded as a partial MaxSAT instance, which contains a set of hard clauses and a set of soft clauses. With this encoding, a MaxSAT solver is used to find an optimal solution of the instance. This solution can be mapped to the optimal solution of the Correlation Clustering. Instance-level constraints can also be encoded in the MaxSAT instance.

The encoding uses boolean variables x_{ij}, where $i < j$, $i, j \in \{1, \ldots, n\}$. x_{ij} is equal to 1 if the two objects o_i and o_j are in the same cluster. Hard clauses are used to ensure a well-defined clustering: for each triple of objects (o_i, o_j, o_t), if o_i, o_j are in the same cluster and o_j, o_t are in the same cluster, then objects o_i, o_t must be in the same cluster. The

condition is equivalent with: if $x_{ij} = 1$ and $x_{jt} = 1$ then $x_{it} = 1$. It can be expressed by a hard clause:

$$(\overline{x}_{ij} \vee \overline{x}_{jt} \vee x_{it})$$

The soft clauses are used to encode the cost function. To minimize the cost function, each pair of objects (o_i, o_j) with the similarity $s_{ij} = 1$ should be in the same cluster, hence x_{ij} should be equal to 1, or it can be expressed by a soft clause (x_{ij}). Similarly, with each pair of objects (o_i, o_j) that $s_{ij} = 1$, a soft clause $(\overline{x}_{ij})$ is added to the MaxSAT formula.

With this encoding, the instance-level constraints can be easily expressed by hard clauses. A must-link constraint between objects o_i and o_j is expressed by a clause (x_{ij}) and a cannot-link constraint between objects o_i and o_j is expressed by a clause $(\overline{x}_{ij})$. From a MaxSAT solution, it is easy to construct a clustering as follows:

- Assign object o_1 to the first cluster C_1,
- For every i such that variable $x_{1i} = 1$, assign object o_i to cluster C_1,
- Find the smallest i for which o_i is not assigned to a cluster, then assign object o_i and any object j such that $x_{ij} = 1$ to cluster C_2,
- Iterate this process until every object is assigned.

It is demonstrated in the paper that the partition found by this method is an optimal partition. In this encoding, the number of clusters k is not bounded and the optimal partition may have a large number of clusters. The authors also propose an alternative MaxSAT formulation, which uses an upper bound k on the number of clusters.

4.3 Column generation framework for Constrained Sum of Squares Clustering

Column generation is an efficient method for solving linear programming (LP) problems with a large number of variables. In many LP problems, most of the variables will be zero in the optimal solution and only a subset of variables is needed to solve the problem. In a column generation method, the LP problem is split into the master and the auxiliary problem. The master problem is the initial problem. The column generation method begins by solving a restricted master problem, which is the master problem but with only a small number of variables. The auxiliary problem is a problem created from the solution of the restricted master problem. It is used to find a new variable with negative reduced cost, *i.e.* a variable that can improve the objective function of the master problem. If such a variable exists, it is added to be considered for resolving the restricted master problem. In the column generation method, the restricted master and the auxiliary problems are solved iteratively until solving the auxiliary problem shows that there exists no variable with non-negative reduced cost. It is proven that the solution of the restricted master problem is now the optimal solution of the origin problem.

[Rao 1969] proposes an integer programming formulation for cluster analysis by considering all possible clusters. With n objects, there are a total of 2^n possible clusters: C_1 to C_{2^n}. A matrix $A = (a_{it})$ with size $n \times 2^n$ is used to represent the relation between

objects and clusters: $a_{it} = 1$ if $o_i \in C_t$ and $a_{it} = 0$ otherwise. For the criterion of SSE, the cost c_t for each cluster C_t is defined as:

$$c_t = SSE(C_t) = \sum_{o_i \in C_t} ||o_i - m_t||^2 = \sum_{i=1}^{n} ||o_i - m_t||^2 a_{it}$$

where m_t denotes the centroid of the cluster C_t.

Let T be the set of all possible cluster indices, i.e. $T = \{1, 2, \ldots, 2^n\}$. For all $t \in T$, let x_t be boolean variables, $x_t = 1$ denotes that the cluster C_t is in the optimal partition.

The problem can now be formulated as an integer linear problem as:

$$\min \sum_{t \in T} c_t x_t \tag{4.1}$$

subject to:

$$\begin{cases} \sum_{t \in T} a_{it} x_t = 1 \qquad \forall i \in \{1, \ldots, n\} & \text{(4.2a)} \\ \sum_{t \in T} x_t = k & \text{(4.2b)} \\ x_t \in \{0, 1\} & \text{(4.2c)} \end{cases}$$

In this formulation, there are n constraints in Equation (4.2a) to express that, in the optimal partitions, each point must be in exactly one cluster. The constraint in Equation (4.2b) expresses that there are exactly k clusters. The dual of the master problem in (4.1) is expressed by [Rao 1969]:

$$\max \left(-k\sigma + \sum_{i=1}^{n} \lambda_i\right) \tag{4.3}$$

subject to:

$$\begin{cases} -\sigma + \sum_{i=1}^{n} a_{it} \lambda_i \leq c_t \qquad \forall t \in T & \text{(4.4a)} \\ \lambda_i \geq 0 \qquad \forall i = 1, 2, \ldots, n & \text{(4.4b)} \\ \sigma \geq 0 & \text{(4.4c)} \end{cases}$$

The n dual values λ_i correspond to n constraints in Equation (4.2a) and the dual value σ corresponds to the constraint in Equation (4.2b). Each column t in the master problem (4.1) corresponds to one constraint of the dual in Equation (4.4a).

This ILP formulation has 2^n variables, therefore it cannot be solved straightforwardly. The column generation method restricts the master problem in Equation (4.1) to a smaller set of columns $T' \subseteq T$. The dual of the master problem, a LP relaxation, can be solved by three different strategies [du Merle *et al.* 1999] and the best strategy is the ACCPM interior point method of [Goffin *et al.* 1992]. Suppose that solving the restricted master problem returns values for σ and λ_i. The objective of the auxiliary problem is to find a

column t with negative reduced cost. The column t with the smallest reduced cost is the solution of:

$$\arg\min_{t \in T} \quad \sigma - \sum_{i=1}^{n} a_{it}\lambda_i + c_t \tag{4.5}$$

For solving the auxiliary problem, [du Merle *et al.* 1999] considers the problem as a hyperbolic program in 0-1 variables, and solves it by a branch-and-bound algorithm for unconstrained quadratic 0-1 optimization. The column generation framework iteratively solves the LP relaxation of the master problem and the auxiliary problem to find the optimal solution of the LP relaxation. This solution is not necessarily integer and branching is added to obtain an optimal partition. The column generation framework [du Merle *et al.* 1999] solved for the first time the optimal sum of squares partition on datasets with 100-200 objects, as for example the dataset Iris.

[Aloise *et al.* 2012] improved the performance of the framework by introducing a new method based on geometric arguments to solve the auxiliary problem. It is claimed to solve the optimal partition with datasets over 2300 objects.

[Babaki *et al.* 2014] introduced for the first time user-constraints to the framework. The constraints can be among must-link, cannot-link and anti-monotone constraints. A constraint on a cluster is an anti-monotone constraint iff whenever a cluster C satisfies this constraint, any sub-cluster $C' \subseteq C$ also satisfies it. According to the authors, an important observation to incorporate user-constraints to the column generation framework is: Must-link, cannot-link and anti-monotone constraints can be evaluated on individual clusters in a partition. For example, a must-link constraint $ML(o_i, o_j)$ is satisfied in cluster C_t if objects o_i, o_j are both in or not in this cluster. Similarly, a cannot-link constraint $CL(o_i, o_j)$ is satisfied in cluster C_t if the cluster contains at most one of two objects. A must-link or cannot-link constraint is satisfied in a partition if it is satisfied in every cluster C_t of the partition. With this observation, it is sufficient to process user-constraints in the auxiliary solver: finding a column t with negative reduced cost such that C_t satisfies all user-constraints. [Babaki *et al.* 2014] proposes a BAB algorithm for finding the cluster C_t which satisfies all user-constraints and corresponds to a negative reduced cost. The search process begins with an empty cluster C_t and a set of candidate blocks such that covers all objects. A block is a set of objects that are connected by must-link constraints. In case there is no must-link constraints, each block is equivalent to an object. The BAB algorithm performs a set-enumeration. In each iteration, a candidate block is considered to be affected to the cluster C_t. With the notion of blocks, all must-link constraints are guaranteed to be satisfied in the returned cluster C_t at the end. The cannot-link and anti-monotone constraints are used to remove incompatible candidate blocks.

4.4 Summary

In this chapter, we described recent declarative frameworks for solving data mining problems. We discussed the ideas and techniques involved and investigate advantages and disadvantages of proposed solutions. These researches inspire us that we could construct a general and flexible framework for the problem of constrained clustering.

Chapter 5

An initial CP Model for Constrained Clustering

Dans ce chapitre, nous proposons un cadre déclaratif et générique, fondé sur la Programmation par Contraintes pour modéliser une tâche de clustering sous contraintes. Nous disposons d'une collection de n points et d'une mesure de dissimilarité entre ces points. Nous considérons le cas où le nombre de clusters k est connu à l'avance. Le modèle a pour objectif de trouver une partition des points en k clusters, optimisant un critère et intégrant facilement des contraintes utilisateur. Le modèle proposé est basé sur une représentation à deux niveaux : un ensemble de variables affectant un point représentatif à chaque classe et un ensemble de variables affectant un point représentatif à chaque objet. Trois critères d'optimisation différents sont modélisés: minimiser le diamètre maximal, maximiser la marge minimale, ou minimiser la somme des dissimilarités intra-classes. Nous montrons que notre cadre intègre naturellement des contraintes d'utilisateur connues, par exemple les contraintes sur les instances (must-link, cannot-link) ou des contraintes sur les classes (capacité maximale ou minimale, séparation, diamètre, densité).

Des améliorations du modèle sont étudiées, reposant sur des résultats théoriques, ce qui permet d'alléger le modèle sans changer la sémantique. De plus, parmi les critères d'optimisation, celui de la somme des dissimilarités se présente sous forme d'une somme linéaire. Afin d'avoir une propagation plus efficace pour ce critère, nous avons proposé un algorithme de filtrage qui exploite les informations données par une affectation partielle de variables.

Des expérimentations sur des bases de données classiques montrent la performance de notre modèle. De plus, la possibilité de combiner différents types de contraintes utilisateur et la flexibilité du choix du nombre de clusters permettent d'obtenir des solutions intéressantes.

Contents

5.1 Introduction

We introduce in this chapter our first model for constrained clustering that can be provided to Constraint Programming solvers. We first present the problem on which we focus. The problem consists of finding a partition composed of a given number of clusters that satisfies a set of user-constraints and optimizes a criterion. We define the problem as follows:

Input

- $\mathcal{O} = \{o_1, \ldots, o_n\}$: set of points to be clustered. Points are indexed and named by their indices.
- k: number of clusters in a partition.
- A matrix of dissimilarities between two points. We denote by d_{ij} the dissimilarity between points i and j.

Output A partition of $\mathcal{O}$ in k clusters $C_1, \ldots, C_k$.

User-Constraints

- Minimal size α of clusters: It requires that each cluster has a number of points greater than a given threshold α, i.e. $\forall c \in [1, k],\ |C_c| \geq \alpha$.
- Maximal size β of clusters: It requires each cluster to have a number of objects inferior to a predefined threshold β, i.e. $\forall c \in [1, k],\ |C_c| \leq \beta$.
- Split δ-constraint: It expresses that the split between any two clusters must be at least δ, i.e. $\forall c \in [1, k], \forall c' \neq c, \forall o_i \in C_c, \forall o_j \in C_{c'}, d_{ij} \geq \delta$.
- Diameter constraint γ: It expresses that the diameter of each cluster must be at most γ, i.e. $\forall c \in [1, k], \forall o_i, o_j \in C_c,\ d_{ij} \leq \gamma$.

- Density constraint: The ϵ-constraint introduced in [Davidson & Ravi 2005b] requires for each point o_i to have in its neighborhood of radius ϵ at least another point belonging to the same cluster: $\forall c \in [1,k], \forall o_i \in C_c, \exists o_j \in C_c, o_j \neq o_i$ and $d_{ij} \leq \epsilon$. This constraint tries to capture the notion of density, introduced in DBSCAN [Ester *et al.* 1996]. We have extended it by proposing a density-based constraint, stronger than the ϵ-constraint: it requires that for each point o_i, its neighborhood of radius ϵ contains at least m points belonging to the same cluster as o_i, i.e. $\forall c \in [1,k], \forall o_i \in C_c, |\{o_j : o_j \in C_c, j \neq i, d_{ij} \leq \epsilon\} \cap C_c| \geq MinPts$.

- Must-link constraint: A must-link constraint on two points i and j requires that these two points must be in the same cluster.

- Cannot-link constraint: A cannot-link constraint on two points i and j requires that these two points must be in different clusters.

Criterion

- No criterion: Finding all partitions with n points divided into k clusters that satisfy all given user-constraints.

- Diameter criterion: Finding a partition with n points divided into k clusters that satisfies all given user-constraints, and the partition has the minimal maximum diameter. The maximum diameter D of a partition is measured as:

$$D = \max_{i,j \text{ s.t. } \lambda_i = \lambda_j} d_{ij}$$

  where λ_i denotes the cluster that contains point i.

- Split criterion: Finding a partition with n points divided into k clusters that satisfies all given user-constraints, and the partition has the optimal minimum split. The minimum split S of a partition is measured as:

$$S = \min_{i,j \text{ s.t. } \lambda_i \neq \lambda_j} d_{ij}$$

- Within Cluster Sum of Dissimilarities (WCSD) criterion: Finding a partition with n points divided into k clusters that satisfies all given user-constraints, and the partition has the optimal WCSD. The WCSD is measured as:

$$WCSD = \sum_{i,j \text{ s.t. } \lambda_i = \lambda_j} d_{ij}$$

  Let us notice that, for this criterion the dissimilarities are usually measured by the squared Euclidean distance.

For solving the described problem, we propose a model in Constraint Programming (CP). CP is a powerful paradigm for solving combinatorial optimization problems. The problems are encoded in CP by defining variables and constraints. A problem can be encoded in different ways in CP. CP relies on constraint propagation and search for solving the problem. As a result, the choice of variables and constraints is extremely important

since different constraints have different filtering algorithms with different complexity and different levels of consistency. Finding a correct and efficient encoding in CP for a specific problem can be a difficult task.

CP provides a rich set of constraints that are designed to express relations between the variables. We present the constraints used in our model:

- Unary constraints: These constraints are defined on a single variable. The propagation of such a constraint is simple, as presented in Chapter 2.

- Binary constraints: Those are constraints between two variables x, y. In most CP solvers, binary constraints are guaranteed to be arc consistent with built in filtering algorithms. A basic arc consistency algorithm is presented in section 2.2 of Chapter 2.

- Global Cardinality Constraint: It bounds the number of times a value or a set of values is assigned to the variables in a set X to be at least l and at most u. We consider only a simple form of this constraint, which bounds the number of times a value v is assigned to variables in X. It is written as:

  $$gcc(X, v, l, u)$$

  where X is a set of variables, v is the value, l and u are the lower bound and upper bound of times v is assigned to variables in X. Another version used in this thesis is:

  $$gcc(X, y, l, u)$$

  where y is a variable. This enforces that the value assigned to y must appear at least l and at most u times in the set X.

  Example 17 Consider a set of variables $X = \{x_1, x_2, x_3\}$ with the domains $Dom(x_1) = Dom(x_2) = Dom(x_3) = \{1, \ldots, 10\}$. The constraint $gcc(X, 3, 1, 2)$ requires that at least one and at most two of the variables x_1, x_2, x_3 are assigned to the value 3.

  Generalized arc consistency for this constraint can be achieved by a filtering algorithm, which is based on a network flow [Régin 1996]. We consider in the thesis two simple versions of the cardinality constraint: AtLeast and AtMost constraints, which are defined as:

  $$AtLeast(X, v, m)$$

  $$AtMost(X, v, m)$$

  These constraints require that there are at least (at most) m variables in $\mathcal{X}$ that are assigned to value v.

- Element constraint: It gives access to a specified variable in an array of variables $[x_1, \ldots, x_n]$. It is written as:

  $$element([x_1, \ldots, x_n], y, i)$$

  It constrains the variable y to be equivalent to the $i-th$ variable in the array, or: $y = x_i$. Generalized arc consistency for this constraint can be achieved [Gent *et al.* 2006].

- Reified constraint: It associates a boolean variable b to a constraint c. It is written as:

$$b \leftrightarrow c \tag{5.1}$$

In this constraint, the variable b takes the value 1 if the constraint c is satisfied and 0 otherwise. The propagation of reified constraints depends on the constraint c, and also depends on solvers. Most constraint solvers support reified versions of unary and binary constraints.

Example 18 Consider a set of variables $X = \{x_1, x_2, x_3\}$ with the domains $Dom(x_1) = Dom(x_2) = Dom(x_3) = \{1, \ldots, 10\}$. Suppose that we need to post a constraint:

$$(x_1 = 2) \rightarrow (x_2 = x_3) \tag{5.2}$$

This constraint cannot be posted directly in most solvers. For modeling this constraint, we can use two auxiliary boolean variables b_1 and b_2: $Dom(b_1) = Dom(b_2) = \{0, 1\}$. Then we post the 2 reified constraints as follows:

$$b_1 \leftrightarrow (x_1 = 2) \tag{5.3}$$

and

$$b_2 \leftrightarrow (x_2 = x_3) \tag{5.4}$$

Those are reified versions of binary constraints and are supported in most of constraint solvers. Finally, we post a binary constraint for binding b_1 and b_2 as follows:

$$b_1 \rightarrow b_2 \tag{5.5}$$

For instance, during the search, suppose that the domains of variables have been reduced to:

$$Dom(x_1) = 2, Dom(x_2) = \{1, 3\}, Dom(x_3) = \{1, \ldots, 10\}$$

According to the constraint (5.3), the variable b_1 is instantiated: $b_1 = 1$. The propagation of constraint (5.5) then instantiates the variable b_2: $b_2 = 1$. The propagation constraint (5.4) actives the constraint $x_2 = x_3$, which restricts the domain of variable x_3:

$$Dom(x_3) = \{1, 3\}$$

5.2 Variables and domains

For modeling the problem described in section 5.1, we propose an encoding on two levels:

Representation of clusters For each cluster, a point in the cluster is chosen to be the representative point. The idea comes from the k-medoids algorithm [Kaufman & Rousseeuw 1987] where each cluster is represented by a medoid, i.e. the point whose average dissimilarity to all the objects in the cluster is minimal. However, identifying the medoid of each cluster requires to know the partition. We propose to replace the condition on representative points: a representative point is not necessary in the

center of a cluster, but it is the point that has smallest index in the cluster. With this condition, modeling the representation of clusters in CP is simple, as presented by constraints given in the next section. This condition also helps to avoid symmetric solutions, as given a cluster, there is only one point satisfying the condition.

Representation of the assignment of points to a cluster Each cluster is associated with a representative point. We therefore propose to replace the assignment of points to a cluster to the assignment of points to the representative point. Suppose that the representative points are identified, the dissimilarity between a point and the representative of each cluster is known. Encoding the assignment of points to a representative point is intuitive and simple.

Based on these ideas, for modeling the problem of constrained clustering, we define two sets of variables:

- $\mathcal{I} = \{I_1, \ldots, I_k\}$: For each cluster $c \in \{1, \ldots, k\}$, the point with the smallest index is considered as the representative point of the cluster. The variable I_c gives the representative point of the cluster c. The domain of each variable I_c is all possible indices, hence $Dom(I_c) = \{1, \ldots, n\}$.
- $\mathcal{G} = \{G_1, \ldots, G_n\}$: Assigning a point to a cluster becomes assigning the point to the representative of the cluster. For each point $i \in \{1, \ldots, n\}$, the variable G_i is the representative point of the cluster that contains the point i. At the beginning, any point can be a representative point, hence the domain of variable G_i is also all possible indices, $Dom(G_i) = \{1, \ldots, n\}$.

A complete assignment of those variables corresponds to a partition. We also propose another encoding which relies on only one one level, as presented in the next chapter.

Example 19 Given a dataset of 7 points $\mathcal{O} = \{o_1, \ldots, o_7\}$ and a clustering task dividing the points into 2 clusters. We define $\mathcal{I} = \{I_1, I_2\}$ and $\mathcal{G} = \{G_1, \ldots, G_7\}$ where the initial domains of all variables are $\{1, \ldots, 7\}$. Let us consider a partition of 2 clusters, the first one composed of o_1, o_2, o_5 and o_6 and the second one composed of the remaining points. The points are denoted by their integer indices (o_1 is denoted by 1, o_2 by 2 and so on). This partition corresponds to the assignment of variables:

$$I_1 = 1, I_2 = 3$$

(since 1 is the smallest index among $\{1, 2, 5, 6\}$ and 3 is the smallest index among $\{3, 4, 7\}$)

$$G_1 = G_2 = G_5 = G_6 = 1$$

(since 1 is the representative of the first cluster)

$$G_3 = G_4 = G_7 = 3$$

(since 3 is the representative of the second cluster).
Figure 5.1 presents this example. The left image shows the dataset and the domain of variables at the beginning whereas the right image presents a partition and the corresponding assignment of variables.

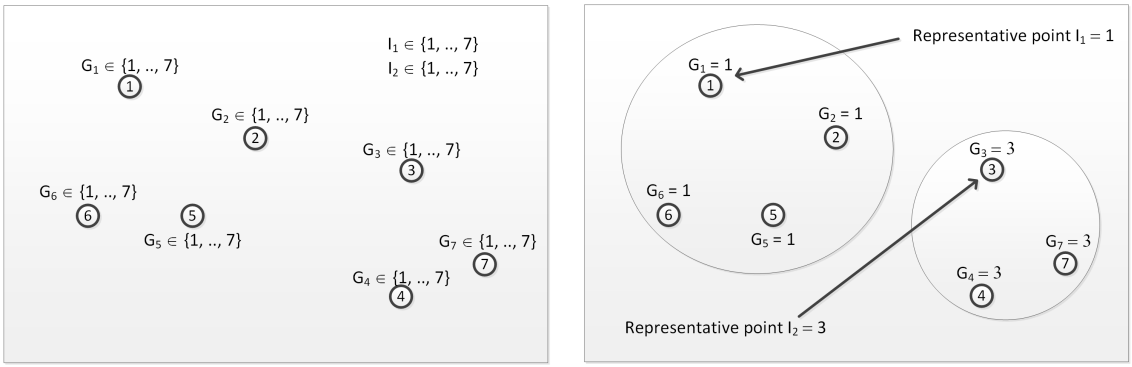

Figure 5.1: Example of assignment values to variables in a solution

A float variable is introduced for representing the optimization criterion. It is denoted by D for the maximal diameter, S for the minimal margin and V for the Within Cluster Sum of Dissimilarities. The domains of D and S are the interval whose lower (upper) bound is the minimal (maximal, resp.) dissimilarity between any two points. The domain of V is upper-bounded by the sum of the dissimilarities between all pairs of points.

5.3 Constraints

5.3.1 Modeling Partition Constraints

The relationships between points and their clusters are expressed by the following constraints:

The representative of a representative point is itself Obviously, if a point i is the representative point of the cluster c, then $I_c = i$. The variable G_i must be assigned to i ($G_i = i$) because the point i is in the cluster c, and the point i is also the representative of this cluster. As a result, the variables G_{I_c}, giving the cluster to which the representative point I_c belongs to, must be equal to I_c. This can be expressed by an element constraint. For each value $c \in \{1, \ldots, k\}$, we post a constraint:

$$element([G_1, \ldots, G_n], I_c, I_c) \tag{5.6}$$

With those constraints, as soon as a representative point i of a cluster c is identified, both variable I_c and G_i are instantiated to the value i: $I_c = G_i = i$.

The representative of a cluster must be the smallest index in the cluster This condition makes the representative be unique for each cluster. For each point i, the value G_i giving the representative point of its cluster, must be smaller or equal than i. To represent this constraint, for each value $i \in \{1, \ldots, n\}$, we post a unary constraint:

$$G_i \leq i. \tag{5.7}$$

Example 20 Given the dataset in Example 19, the constraints (5.7) restrict the domain of variables in $\mathcal{G}$ to:

$$Dom(G_1) = \{1\}, Dom(G_2) = \{1, 2\}, \ldots, Dom(G_7) = \{1, \ldots 7\}.$$

The representative of each point must be one and only one of all the representatives The value of a variable G_i must be among the possible representative points. It means that G_i must be assigned to the value of exactly one of the variables in $\mathcal{I}$. This relation can be expressed by a global cardinality constraint. For each value $i \in \{1, \ldots, n\}$, we post a constraint:

$$gcc(\mathcal{I}, G_i, 1, 1) \tag{5.8}$$

This means that the value of G_i must appear exactly once time in $\mathcal{I}$. With these constraints, when all variables in $\mathcal{I}$ are assigned, the domains of variables in $\mathcal{G}$ contain only the values of representative points.

Example 21 Given the dataset in Example 19, suppose that the two representatives points are identified: $I_1 = 1, I_3 = 3$. With the constraints (5.6), (5.7) and (5.8) the domains of all variables in $\mathcal{G}$ are filtered to:

$$Dom(G_1) = Dom(G_2) = \{1\},$$

$$Dom(G_3) = \{3\},$$

$$Dom(G_4) = Dom(G_5) = Dom(G_6) = Dom(G_7) = \{1, 3\}.$$

Symmetry breaking constraints In our model, given an assignment of variables in $\mathcal{I}$ and $\mathcal{G}$, any permutation of the values of the representative points leads to the same partition. Figure 5.2 presents an example of symmetric solutions, if there is no restriction on the choice of representative points. A simple way to avoid symmetry solutions is to impose a strict ordering on the representatives: the representatives must be in an ascending order. This relation can be expressed by binary constraints. For each value $c \in \{1, \ldots, k-1\}$, we post a constraint:

$$I_c < I_{c+1}.$$

Since a representative must have the smallest index, the representative of the first cluster must be the first point, so we post a constraint:

$$I_1 = 1.$$

5.3.2 Modeling the user-constraints

The advantage of modeling with CP is that many different types of user constraints can be directly added. Both cluster-level constraints and instance-level constraints can be formulated within our model. All popular user-defined constraints may be straightforwardly integrated:

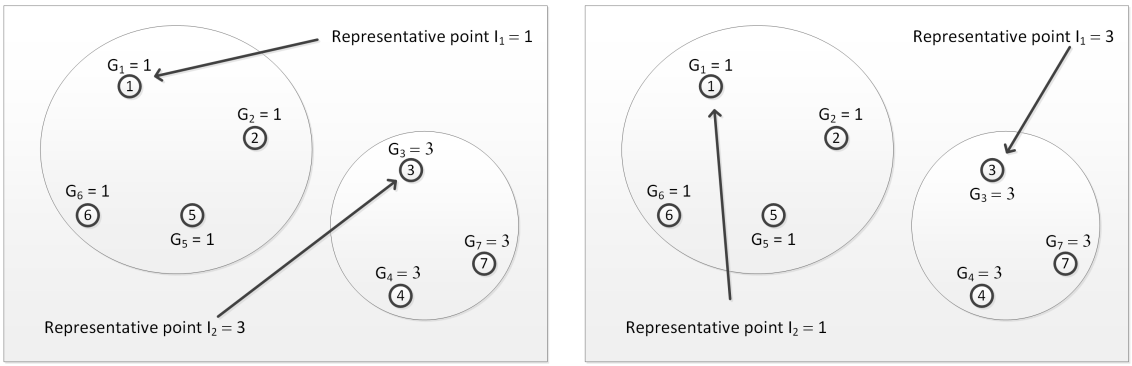

Figure 5.2: Example of symmetric solutions

Minimal size α of clusters It requires that each cluster has a number of points greater than a given threshold α. This constraint can be expressed by an AtLeast constraint. For each value $c \in \{1, \ldots, k\}$, we post a constraint:

$$AtLeast(\mathcal{G}, I_c, \alpha).$$

This constraint expresses that there must be at least α variables in $\mathcal{G}$ taking the value of the variable I_c, which is equivalent to there are at least α points in the cluster c.

Maximal size β of clusters It requires each cluster to have a number of objects inferior to a predefined threshold β. Similarity, for each value $c \in \{1, \ldots, k\}$, we post a AtMost constraint:

$$AtMost(\mathcal{G}, I_c, \beta).$$

This constraint requires that there must be at most β variables in $\mathcal{G}$ taking the value of the variable I_c.

Split δ-constraint It expresses that the split between any two clusters must be at least δ. This constraint is satisfied if for any two points i and j, if the dissimilarity between them is less than δ, we require them to be in the same cluster. Therefore, for each $i < j \in [1, n]$ satisfying the condition $d_{ij} < \delta$, we put a binary constraint:

$$G_i = G_j.$$

Diameter constraint γ It expresses that the diameter of each cluster must be at most γ. This constraint is satisfied if for any two points i, j, if the dissimilarity between them is greater than γ, we require them to be in different clusters. Therefore for each $i < j \in [1, n]$ such that $d_{ij} > \gamma$, we put a binary constraint:

$$G_i \neq G_j$$

Density constraint It requires that for each point i, its neighbourhood of radius ϵ contains at least $MinPts$ points belonging to the same cluster as i. To express this constraint in our model, for each point $i \in [1, n]$, we compute the set of variables $\mathcal{X}_i$

composed of variables G_j ($j \in \{1, \ldots, n\}$) such that $d_{ij} < \epsilon$. We then post an AtLeast constraint:

$$AtLeast(\mathcal{X}_i, G_i, MinPts).$$

It requires that, in the set of variables $\mathcal{X}_i$, there must be at least $MinPts$ variables that have the same value as G_i, which means that there must be at least $MinPts$ points in the same cluster as point i.

Must-link constraint A must-link constraint on two points i and j is expressed by a binary constraint:

$$G_i = G_j.$$

Cannot-link constraint A cannot-link constraint on i and j is expressed by a binary constraint:

$$G_i \neq G_j.$$

5.4 Criterion Modeling

As presented in Chapter 2, a Constraint Optimization Problem (COP) is a CSP together with an objective function $f()$ to be optimized. Typically, a COP is solved in CP by adding a constraint whenever a solution Δ is found during the search. The constraint requires that the value of the optimization function must be better than $f(\Delta)$ in any future solution. For modeling a COP, in general, a variable, i.e. x, is defined and users have to post constraints on x to express the function $f()$. Then an optimization constraint is posted, e.g. in case of minimization:

$$minimize\ x$$

This constraint ensures that, whenever a solution Δ is found, a new constraint is added to the model, e.g. in case of minimization:

$$x < f(\Delta) \tag{5.9}$$

where $f(\Delta)$ is the actual value of the function $f()$ on the solution. $f(\Delta)$ is obtained by the value of the variable x in the solution. In some problems, the variable x is not instantiated and $f(\Delta)$ is obtained by a lower or upper bound of x.

By using the constraint (5.9), the last found solution is guaranteed to be the solution with the minimal value of the function $f()$ among all found solutions. When a complete search is used, the last found solution is guaranteed to be the optimal solution. Note that the constraint (5.9) is strict, providing an increasingly tight bound. As a result, only one optimal solution is found, even if there exists different optimal solutions with the same value of $f()$.

5.4.1 Modeling the diameter criterion

The criterion of diameter is encoded with the float variable D. At the beginning, the domain of D is not a set of values, but an interval:

$$Dom(D) = [d_{min}, d_{max}]$$

where d_{min} and d_{max} are the minimum and the maximum dissimilarity between any two points. To express the criterion, a straightforward way to encode it is to put a requirement on each pair of points: if two points i and j are in the same cluster, the dissimilarity between these two points should be equal or inferior to the value of the maximum diameter:

$$G_i = G_j \rightarrow d_{ij} \leq D \tag{5.10}$$

With these constraints, whenever two points i and j are known to be in the same cluster, the lower bound of the variable D is modified: $D.lb = max(D.lb, d_{ij})$. When all variables in $\mathcal{G}$ are instantiated, the lower bound of D is the maximum diameter.

Note that the constraint (5.10) is a logic constraint in the form $a \rightarrow b$, which is equivalent with $!b \rightarrow !a$, with the meaning: if the dissimilarity between any two points is superior to the diameter, then they must be in different clusters.

$$d_{ij} > D \rightarrow (G_i \neq G_j) \tag{5.11}$$

The condition $d_{ij} > D$ is satisfied if the dissimilarity d_{ij} is superior to the upper bound of the variable D. Whenever $d_{ij} > D.ub$, the constraint $G_i \neq G_j$ has to be activated to restrict the domain of variables G_i and G_j.

For modeling the constraints (5.10, 5.11), we propose to use reified versions of binary constraints. For each pair of points i and j, two auxiliary boolean variable b_{ij1} and b_{ij2} are generated and three constraints are posted. The first one models the condition that the dissimilarity between any two points is superior to the diameter:

$$b_{ij1} \leftrightarrow d_{ij} > D. \tag{5.12}$$

The second one models the condition that the two points are not in the same cluster:

$$b_{ij2} \leftrightarrow (G_i \neq G_j). \tag{5.13}$$

The third one binds the two conditions:

$$b_{ij1} \rightarrow b_{ij2}. \tag{5.14}$$

The propagation of the constraints (5.12, 5.13, 5.14) is as follows:

- If $b_{ij1} = 1$, which happens when $d_{ij} > D.ub$ holds in the constraint (5.12), then the variable b_{ij2} is instantiated to 1 because of the constraint (5.14). The reified constraint (5.13) will then propagate the binary constraint $G_i \neq G_j$.
- If $b_{ij1} = 0$ or b_{ij1} is not instantiated ($b_{ij1} \in [0, 1]$), there is no propagation.
- If $b_{ij2} = 1$ or $(G_i \neq G_j)$ holds, there is no propagation.
- If $b_{ij2} = 0$ or $(G_i = G_j)$ holds in the constraint (5.13), then the variable b_{ij1} is instantiated to 0 because of the constraint (5.14). With the reified constraint (5.12), the constraint $d_{ij} \leq D$ is propagated. This is done by setting the lower bound of the domain of D:

  $$D.lb = max(D.lb, d_{ij})$$

  These propagations ensure that the lower bound of D is always the maximum diameter of clusters.

The optimal diameter is guaranteed with an optimization constraint:

$$minimize\ D. \tag{5.15}$$

The constraint (5.15) is activated each time a solution Δ is found. The diameter maximum of the solution, denoted by D^{Δ}, is obtained by the value $D.lb$, then a new constraint is added to ensure that the next solution (if it exists) must have a better maximum diameter:

$$D < D^{\Delta}. \tag{5.16}$$

5.4.2 Modeling the split criterion

The criterion of split is encoded with the float variable S. At the beginning, the domain of S is not a set of values, but an interval:

$$Dom(S) = [d_{min}, d_{max}]$$

where d_{min} and d_{max} are the minimum and the maximum dissimilarity between any two points.

For the split criterion, a similar requirement is put on each pair of points: if the two points are in different clusters, then the dissimilarity between them must be equal or superior to the split:

$$(G_i \neq G_j) \rightarrow d_{ij} \geq S \tag{5.17}$$

It is equivalent with the constraint: if the dissimilarity between any two points is inferior to the split, then they must be in the same cluster.

$$d_{ij} < S \rightarrow (G_i = G_j) \tag{5.18}$$

Similar to the criterion of diameter, for each pair of points i, j, two auxiliary boolean variables b_{ij1} and b_{ij2} are generated and three constraints are posted as follows:

$$b_{ij1} \leftrightarrow d_{ij} < S. \tag{5.19}$$

$$b_{ij2} \leftrightarrow (G_i = G_j). \tag{5.20}$$

$$b_{ij1} \rightarrow b_{ij2}. \tag{5.21}$$

The propagation of these constraints is as follows:

- If $b_{ij1} = 1$ or $d_{ij} < S$ holds, the variable b_{ij2} is instantiated to 1 because of the constraint (5.21). The reified constraint (5.20) then propagates the binary constraint $(G_i = G_j)$.
- If $b_{ij1} = 0$ or b_{ij1} is not instantiated ($b_{ij1} \in [0, 1]$), there is no propagation.
- If $b_{ij2} = 1$ or $(G_i = G_j)$ holds, there is no propagation.
- If $b_{ij2} = 0$ or $(G_i \neq G_j)$ holds, then the variable b_{ij1} is instantiated to 0, the constraint $d_{ij} \geq S$ is propagated. The lower bound of the domain of S is modified:

  $$S.lb = max(S.lb, d_{ij})$$

  These propagations ensure that the lower bound of S is the minimum split.

The optimal split is guaranteed with an optimization constraint:

$$minimize\ S.$$

With this optimization constraint, each time a solution Δ is found, the minimum split S^{Δ} is obtained by the value of $S.lb$, then a new constraint is added:

$$S > S^{\Delta}.$$

5.4.3 Modeling the criterion of Within Cluster Sum of Dissimilarities

The Within Cluster Sum of Dissimilarities (WCSD) is measured as the sum of dissimilarities between any two points in the same cluster, where the dissimilarities is usually defined as the squared Euclidean distance. To represent the relation between points and WCSD, we need a constraint *wcsd(𝒢,V,d)*, which ensures:

$$V = \sum_{i<j\in[1,n]} (G_i = G_j) d_{ij}. \tag{5.22}$$

where $(G_i = G_j)$ is a truth value in $\{0, 1\}$.

Using predefined constraints, the constraint (5.22) for the WCSD criterion can be implemented by the following constraints:

- A set of reified constraints: for all $1 \leq i < j \leq n$,

  $$b_{ij} \leftrightarrow (G_i = G_j)$$

  b_{ij} is a boolean variable, which is equal to 1 iff $G_i = G_j$.

- A linear sum constraint:

  $$V = \sum_{1\leq i<j\leq n} b_{ij} \times d_{ij}$$

However, these constraints, while considered independently, do not offer enough propagation. For example, with $k = 2$, given 4 points from 1 to 4 and a partial assignment where $G_1 = 1$ and $G_2 = G_3 = 2$ as in Figure 5.3 (the number on each edge $\{i, j\}$ represents the value d_{ij}). We have three instantiated boolean variables:

$$b_{12} = b_{13} = 0, \quad b_{23} = 1.$$

The remaining boolean variables are uninstantiated:

$$b_{14}, b_{24}, b_{34} \in \{0, 1\}.$$

Let us assume that a solution with $V = 5$ was found. With the branch-and-bound search, this solution sets the upper bound of variable V to 5. A new constraint is added:

$$\sum_{i<j} b_{ij} \times d_{ij} < 5.$$

As $b_{12} = b_{13} = 0$, $b_{23} = 1$, this constraint becomes:

$$b_{14} + 2b_{24} + 3b_{34} < 4.$$

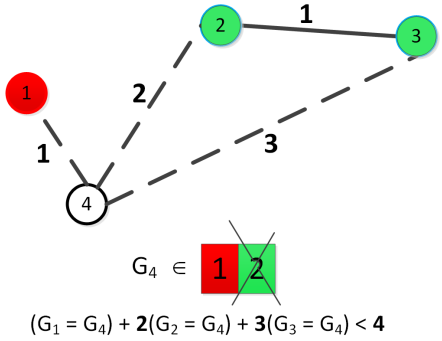

Figure 5.3: Example of filtering

Since $G_2 = G_3$, we can see that b_{24} and b_{34} must be equal and then must not be equal to 1, otherwise the constraint is violated. We should then infer that point 4 cannot be in the cluster number 2, as points 2 and 3, that means value 2 should be removed from the domain of G_4. This filtering however is not done, since the constraints are considered independently.

Several recent works have proposed more efficient filtering for the sum constraint, when it is considered with other constraints. For a sum constraint $y = \sum x_i$ with inequality constraints $x_j - x_i \leq c$, a domain filtering algorithm reduces the domain of x_i when new bounds for y are known [Régin & Rueher 2005]. A bound-consistency algorithm is proposed for a sum constraint with increasing order constraints $x_i \leq x_{i+1}$[Petit *et al.* 2011] or with a constraint *alldifferent*$(x_1, \ldots, x_n)$[Beldiceanu *et al.* 2012]. These cases however do not fit the WCSD criterion constraint (5.22). A generic bound-consistency algorithm for a sum constraint with a set of constraints is proposed in [Régin & Petit 2011]. In our case, the domain of G_i is a set of representative indices, which is not an interval in general, and from which we wish to remove any inconsistent value. We have developed a filtering algorithm for a new global constraint on a variable V, an array of variables G of size n and an array of constants d, which is of the form:

$$V = \sum_{i,j \in [1,n], i<j} (G_i = G_j) d_{ij}. \tag{5.23}$$

Taking into account the partitioning problem, the domain of each variable G_i is the set of the representative indices of the clusters that can contain point i. Let us assume that the domain of the variable V is $[V.lb, V.ub)$ where $V.lb$ is the lower bound, which can be initially 0, and $V.ub$ is the upper bound, which can be the value of V in the last solution with a branch-and-bound search. Suppose that we have a partial assignment of variables in $\mathcal{G}$, where there is at least one point assigned for each group (*e.g* the representative of the group). Let K be the set of points i which have been already assigned to a group (G_i is instantiated) and U the set of the unassigned points. The sum in (5.23) is split into three parts $V = V_1 + V_2 + V_3$, where:

- V_1 is the sum of dissimilarities between the assigned points:

$$V_1 = \sum_{i,j \in K, i<j, G_i = G_j} d_{ij}$$

- V_2 is the sum of dissimilarities between the unassigned points and the assigned points:
$$V_2 = \sum_{i \in U, j \in K} (G_i = G_j) d_{ij}$$
- V_3 is the sum of dissimilarities between the unassigned points:
$$V_3 = \sum_{i<j, i,j \in U} (G_i = G_j) d_{ij}$$

The evaluation of these three parts is based on [Klein & Aronson 1991]. The value of V_1 can be calculated exactly because the set K is already known. For the second part, the value of V_2 is unknown because of the unassigned points. However, a lower bound of V_2, denoted by $V_2.lb$, can be calculated by a sum of the minimum contribution of all the unassigned points. Since each unassigned point i will be assigned to a group, it will contribute to V by a sum of dissimilarities between point i and all the points of K that are in that group. The minimal contribution v_{2i} of the point i is the minimal amount when considering all k groups, with respect to the assigned points:

$$v_{2i} = \min_{c \in [1,k]} (\sum_{j \in K \cap C_c} d_{ij}).$$

A lower bound of V_2 is then the sum of v_{2i}:

$$V_2.lb = \sum_{i \in U} v_{2i}.$$

For the third part, the value of V_3 is unknown too, but a lower bound of V_3 can be calculated by a heuristic. We recall that V_3 is the sum of all d_{ij} with i and j in the same group. Let $p = |U|$, the minimal number of terms d_{ij} in the sum V_3 is the minimal number of within-group pairwise connections[1], while considering all partitions of p points into k groups. This number can be calculated by a function, i.e. $f(p, k)$. With a set U of unassigned points, with the constants d_{ij} $(i, j \in U)$ ordered increasingly, a lower bound V_3, denoted by $V_3.lb$, is then calculated by the sum of the $f(|U|, k)$ first constants in this order.

Example 22 With $p = 10$, $k = 3$ and with a partition into 3 groups of sizes 2, 3 and 5, the number of within-group pairwise connections is 14. The minimal value of this number is 12, corresponding to a partition into 3 groups of sizes 3, 3 and 4.

Theorem 22 *Let m be the quotient of the division of p by k and m' the remainder. The minimal total number of connections for all groups can be calculated as follows:*

$$f(p, k) = (km^2 + 2mm' - km)/2.$$

Proof Let the number of points in each group c be $m + \alpha_c$, with $\alpha_c < 0$ when the group c has less than m points, $\alpha_c \geq 0$ otherwise. We have then

$$\sum_{1 \leq c \leq k} (m + \alpha_c) = p = km + m'.$$

[1] A group is like a clique and the number of pairwise connections is the number of edges in the clique.

It is equivalent with:

$$m' = \sum_{1 \leq c \leq k} \alpha_c.$$

The number of pairwise connections in a group c is:

$$(m + \alpha_c)(m + \alpha_c - 1)/2.$$

The total number for all groups is:

$$\begin{array}{rl} & \sum_{1 \leq c \leq k}(m + \alpha_c)(m + \alpha_c - 1)/2 \\ = & (\sum_{1 \leq c \leq k}(m + \alpha_c)^2 - \sum_{1 \leq c \leq k}(m + \alpha_c))/2 \\ = & (km^2 + 2mm' + \sum_{1 \leq c \leq k}(\alpha_c^2 - km - m'))/2 \end{array}$$

By replacing the value of m' ($m' = \sum_{1 \leq c \leq k} \alpha_c.$), we have:

$$m' \leq \sum_{1 \leq c \leq k} |\alpha_c| \leq \sum_{1 \leq c \leq k} \alpha_c^2$$

where α_c are integers.
Therefore the total number for all groups is greater or equal to:

$$(km^2 + 2mm' - km)/2.$$

The equality is reached when α_c is 1 for m' groups and is 0 for $k - m'$ groups

Example 23 Given a dataset with 8 points as illustrated in Figure 5.4 and a clustering task dividing the points into 2 groups (green and red). Suppose that there are 5 assigned points (3 red points and 2 green points) and 3 unassigned points. The value of V_1 is calculated exactly by the sum of the solid black lines. The lower bound $V_2.lb$ is the sum of the dash red lines and dash green lines. With 3 unassigned points, we have $p = 3, k = 2, m = 1$ and $m' = 1$, the minimum total number of connections is

$$f(p, k) = f(3, 2) = (km^2 + 2mm' - km)/2 = 1.$$

Therefore, the lower bound $V_3.lb$ is the dot blue line.

A lower bound of variable V is given by:

$$V.lb = V_1 + V_2.lb + V_3.lb$$

This lower bound is used for two purposes:

- Detecting the failure during the branch-and-bound search, it happens when $V.lb \geq V.ub$.
- Filtering inconsistent values of unassigned variables. For each value v of an unassigned variable G_i, a new lower bound, denoted by $V'.lb$, is computed with the assumption $G_i = v$ (the computation is explained in the next paragraph). This value is inconsistent if $V'.lb \geq V.ub$. For example in Figure 5.3, value 2 can be removed from the domain of variable G_4 because with the assumption $G_4 = 2$, the new lower bound $V'.lb = 6$, and it is greater than the upper bound $V.ub = 5$.

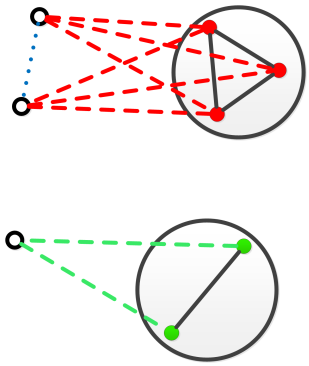

Figure 5.4: Example of $V.lb = V_1 + V_2.lb + V_3.lb$

The filtering algorithm is presented in Algorithm 11. This algorithm uses two arrays *add* and *min*:

- $add[i, c]$ is the added amount if i is assigned to group c:

$$add[i, c] = \sum_{j \in K \cap C_c} d_{ij}.$$

- $m[i]$ is the minimal added amount while considering all possible assignments for i:

$$m[i] = \min_c add[i, c].$$

Since the constants d_{ij} must be ordered increasingly in the computation of $V_3.lb$, they are ordered once in the array *ord*, so $ord[pos]$ gives the constant d_{ij} in the order at position *pos*, and $px[pos]$ ($py[pos]$) gives the index i (j, resp.) of the constant. For the time being, the filtering algorithm is developed for the clustering task, where values in the domain of G_i are the representatives of all clusters for which point i can be assigned. Given a value v in the domain of a variable G_i, the function $gr(v)$ gives the index of the cluster corresponding to v.

The lower bound of V is revised in line 26. Line 30 filters the domain of G_i ($i \in U$): for each value v in the domain, in case of assignment of i into group $gr(v)$, a new lower bound for $V.lb$ is $V'.lb = V'_1 + V'_2.lb + V'_3.lb$ with:

- $V'_1 = V_1 + add[i, gr(v)]$ because point i is supposed to be assigned to group $gr(v)$, the sum of dissimilarities between instantiated points increases of $add[i, gr(v)]$.

- $V'_2 = V_2.lb - m[i]$ because point i is no more unassigned, the contribution of point i in the calculation of $V_2.lb$ must be removed.

- $V'_3.lb$ is the sum of the first $f(|U| - 1, k)$ elements of *ord* that are related to $U \setminus \{i\}$. In order to reduce the complexity of the filtering algorithm, we use V_4 instead of $V'_3.lb$. Here, V_4 is the sum of the first $f(|U| - 1, k)$ elements of *ord* (possibly some related to i). It is obvious that $V'_3.lb \geq V_4$.

The new lower bound is:

$$(V_1 + add[i, gr(v)]) + (V_2.lb - m[i]) + V'_3.lb$$

Algorithm 11: Filtering algorithm

```
1  V_1 ← 0; V_2.lb ← 0; V_3.lb ← 0; V_4 ← 0;
2  for i ← 1 to n where G_i is instantiated do
3  |   for j ← 1 to n do
4  |   |   if G_j is instantiated and G_j == G_i and i < j then V_1 ← V_1 + d_ij;
5  |   |   if G_j is not instantiated then add[j, gr(G_i)] ← add[j, gr(G_i)] + d_ij;
6  |   end
7  end
8  for i ← 1 to n where G_i is not instantiated do
9  |   m[i] ← ∞;
10 |   foreach value v ∈ Dom(G_i) do
11 |   |   if m[i] > add[i, gr(v)] then m[i] ← add[i, gr(v)];
12 |   end
13 |   V_2.lb ← V_2.lb + m[i];
14 end
15 p ← number of uninstantiated variables in G;
16 cpt ← 0; pos ← 1;
17 while cpt < f(p, k) do
18 |   i ← px[pos]; j ← py[pos];
19 |   if G_i is not instantiated and G_j is not instantiated then
20 |   |   cpt ← cpt + 1;
21 |   |   V_3.lb ← V_3.lb + ord[pos];
22 |   |   if cpt ≤ f(p − 1, k) then V_4 ← V_4 + ord[pos];
23 |   end
24 |   pos ← pos + 1;
25 end
26 V.lb ← max(V.lb, V_1 + V_2.lb + V_3.lb);
27 for i ← 1 to n where G_i is not instantiated do
28 |   foreach value v ∈ Dom(G_i) do
29 |   |   if V.lb + add[i, gr(v)] − m[i] − V_3.lb + V_4 ≥ V.ub then
30 |   |   |   delete v from Dom(G_i);
31 |   |   end
32 |   end
33 end
```

which is greater or equal to:

$$V.lb + add[i, gr(v)] - m[i] - V3.lb + V_4$$

So if this last value is greater than the actual upper bound of V, v is inconsistent. The complexity of this algorithm is $O(n^2 + nk)$, since the domain of each G_i is of size at most k. Since $k \leq n$, the complexity is then $O(n^2)$.

5.5 Search Strategy

A search strategy must be given to indicate to the solver the choice of variables and values. Variables will be chosen in the order $\mathcal{I}$ then $\mathcal{G}$. This order represents the fact that the cluster representatives must be identified first, then the solver tries to determine the assignment of points to clusters. Variables in $\mathcal{I}$ are chosen from I_1 to I_k, that means identifying the representative of the first cluster till the last cluster. For the choice of values for each I_c, since the representative is the smallest index in each cluster, the values are enumerated in the ascending order. The next step is branching on variables in $\mathcal{G}$. The branching on uninstantiated variables in $\mathcal{G}$ finds a variable G_i and a value c in the domain of $G[i]$ and makes two alternatives: $G[i] = c$ and $G[i] \neq c$.

For the criterion of maximum diameter and minimum split Variables in $\mathcal{G}$ are chosen in the ascending order of the size of their remaining domain. For the choice of values for each G_i, the closest representative is enumerated first.

Example 24 Given a dataset of 4 points and a clustering task dividing points into 2 clusters. Figure 5.5 expresses the search tree by using the explained strategy. Node 0 presents the initial domain of variables at the beginning of the search. As I_1 is instantiated, the variable I_2 is branched by assigning $I_2 = 2$ first. When $I_2 = 2$, the variable G_2 is instantiated to 2 and the domain of others variables in $\mathcal{G}$ is restricted to $\{1, 2\}$. The variable G_3 is chosen for the next branching. As the representative point I_2 is actually closer to the point 3, G_3 is assigned first to 2, as expressed in Node 2. In Node 19, as soon as the second representative point is instantiated $I_2 = 4$, the variables G_2, G_3 and G_4 are all instantiated, by the propagation of the partition constraints: the representative point is the point with the smallest index in the cluster.

For the criterion of WCSD A mixed strategy is used. Because an upper bound is necessary for the WCSD constraint to be effective, a greedy strategy is used first to quickly find a solution. In this step, G_i and c are selected to make sure that the value of V increases as little as possible. The first solution found in general has a good value of WCSD (which is close to the optimal value). After finding a first solution, the search strategy is changed, following the fail-first principle, which tends to cause the failure early. In this strategy, the branching tries to make alternatives on frontier points. The value c is always the representative of the closest group to point i. The variables in $\mathcal{G}$ with the smallest domain size are all considered and the variable that makes the most changes on V is chosen for branching.

Figure 5.5: Example of search strategy for the criterion of diameter and split

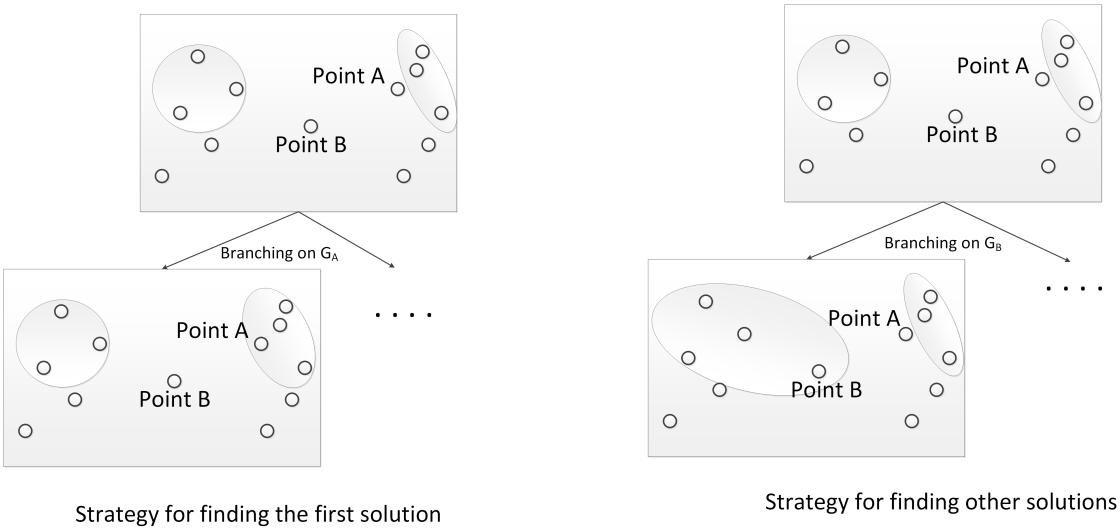

Figure 5.6: Example of search strategy for the criterion of WCSD

Example 25 Given a node in the search tree with 6 points that have been instantiated to two clusters, as illustrated in Figure 5.6. If we have not found the first solution, the variable G_A (that corresponds to point A) is considered for the branching. The reason is, this point is very close to the cluster on the right and by assigning the point A to this cluster, the value of WCSD increases very little. With this strategy, we have a good chance that the WCSD of the first solution found is close to the optimal value. If we already have found one or more solutions, the variable G_B (which corresponds to point B) is considered for branching. This point is in the frontier between two clusters. Although this point is closer to the cluster on the left, there is a high probability that it is not true in the optimal solution, as the difference between the dissimilarities of this point to the cluster on the left and on the right is close to zero. If assigning the point B to the cluster on the left is not correct, we want to detect this failure soon. Branching on point B makes the first part (V_1) of the calculation on bounds on WCSD increase a lot, and the lower bound of $WCSD$ is larger than the bound when we make the branching on point A. As a result, the filtering algorithm for WCSD is more effective, we have a better chance to detect a failure and to filter more values in the domain of variables in $\mathcal{G}$.

5.6 Model Improvements

By using a complete branch and bound search, our model allows to find the optimal solution. In order to improve the efficiency, different aspects have been studied.

5.6.1 Search strategy improvement by point ordering

In order to instantiate cluster representatives, the variables in $\mathcal{I}$ are taken in the order $I_1, \ldots, I_k$ when enumerating with values. Since a representative must have the smallest index in the cluster, for each variable, values (point indexes) are enumerated in the as-

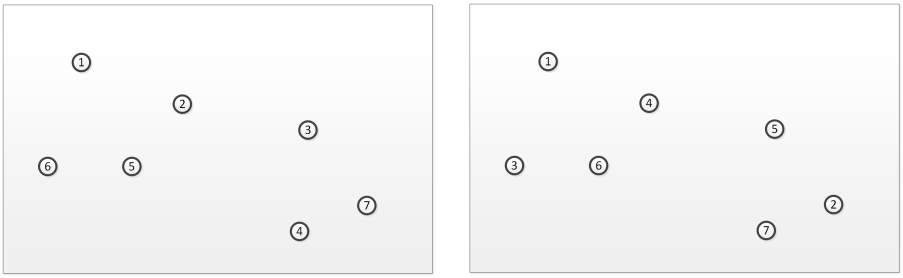

Figure 5.7: Before and After reordering points

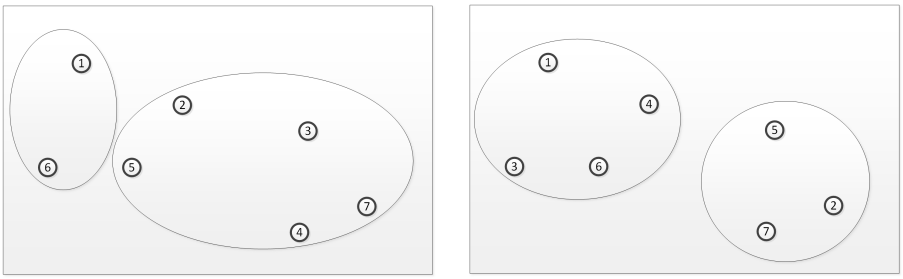

Figure 5.8: The first solution found with initial and new ordering

cending order. The index of points is then really important. Points should be reordered such that those which are more probably a representative should have smaller index.

We present in Algorithm 12 the reordering algorithm. It is based on the FPF heuristic [Gonzalez 1985] to reorder points. The idea is to use the FPF algorithm with $k = n$ (each point is in a cluster), so that each point will be taken in turn. The order in which the algorithm chooses points gives the order of points. Figure 5.7 presents points indices before and after the reordering. Figure 5.8 presents the first solution found with the old and new ordering. With the new ordering, the first solution usually has a diameter which is close to the optimal value. It is very important as this value is used for the upper bound of the variable D.

5.6.2 Constraint improvements

For a given k, without any user-constraints, the diameter d_{FPF} returned by the FPF algorithm has been proved by [Gonzalez 1985] to satisfy:

$$d_{opt} \leq d_{FPF} \leq 2d_{opt} \tag{5.24}$$

where d_{opt} is the optimal diameter, which is aimed to be found.
This entails bounds for the variable D, whose domain is $[d_{FPF}/2, d_{FPF}]$. Moreover, if the dissimilarity between any two points i, j is greater than d_{FPF}, we can directly add constraints to indicate that they must be in different clusters. It replaces the constraints $d_{ij} > D \rightarrow (G_i \neq G_j)$. That is, for all points $i < j \in [1, n]$:

- If $d_{ij} < d_{FPF}/2$: There is no need to post the constraints (5.12, 5.13, 5.14) for the criterion of diameter. The reason is that as $d_{ij} < d_{FPF}/2$ and $d_{FPF}/2 \leq d_{opt}$, the condition $d_{ij} > D$ is always false, there is no propagation on these constraints even if we post them.

Algorithm 12: Reordering algorithm based on the FPF heuristic

Input: a number of points n, a number of clusters k, a matrix of dissimilarity (d_{ij})
Output: a new index of points in array $NewIndex[1..n]$

$i \leftarrow$ the furthest point ($i \leftarrow \arg\max_j \sum_{t\in[1,n]} d_{jt}$)
$point[i] \leftarrow$ visited
$NewIndex[i] \leftarrow 1$
foreach $j \in [1, n]$ **do**
    if *point*$[j]$ *is not visited* **then**
        $DistanceToVisitedPoint[j] \leftarrow d_{ij}$
    end
end
foreach $index \in [2, n]$ **do**
    $i \leftarrow$ the point that is not visited and has the largest value $DistanceToVisitedPoint[i]$
    $point[i] \leftarrow$ visited
    $NewIndex[i] \leftarrow index$
    foreach $j \in [1, n]$ **do**
        if *point*$[j]$ *is not visited* **then**
            $DistanceToVisitedPoint[j] \leftarrow d_{ij}$
        end
    end
end

- If $d_{FPF}/2 \leq d_{ij} \leq d_{FPF}$: There is no change, three constraints (5.12, 5.13, 5.14) are posted for the criterion of diameter.

- If $d_{ij} > d_{FPF}$: a constraint $G_j \neq G_i$ is posted, instead of three constraints (5.12, 5.13, 5.14), because the condition $d_{ij} > D$ is always true.

Lots of reified constraints can be removed from the model, which improves the model efficiency.

In case there are user-constraints, the properties: $d_{opt} \leq d_{FPF}$ is no longer true. The reason is that the optimal diameter in case there are user-constraints is actually equal or superior to the optimal diameter in case without user-constraints. However, we always have: $d_{FPF} \leq 2d_{opt}$ or $d_{FPF}/2 \leq d_{opt}$. As a result, only the lower bound of D is changed to $d_{FPF}/2$. For all points $i < j \in [1, n]$, we make the changes:

- If $d_{ij} < d_{FPF}/2$: There is no need to post the constraints (5.12, 5.13, 5.14) for the criterion of diameter.

- If $d_{FPF}/2 \leq d_{ij}$: Three constraints (5.12, 5.13, 5.14) are posted for the criterion of diameter.

5.7 Experiments

Our model is implemented on Gecode solver version 4.2.1. Twelve databases from the repository UCI [Bache & Lichman 2013] are used in our experiments. They vary by their size, number of attributes and number of classes. Table 5.1 summarizes information on these datasets, presented in the increasing order of the number of objects. The experiments are all performed on a 3.4GHz Intel Core i5 processor running Ubuntu 12.04. We experiment the performance of our model with each criterion by comparing with other exact methods. To our knowledge, there is no exact algorithm for these criteria that supports any kind of user constraints for a general value $k \geq 3$. For the criterion of diameter, our model is compared with the branch-and-bound approach [Brusco & Stahl 2005] and the algorithm based on graph coloring [Delattre & Hansen 1980], in the case without user-constraints. For the split criterion, finding a partition that optimizes the criterion is a polynomial problem. We experiment our model adding user-constraints on the diameter. For the criterion of WCSD, our model is compared with the Repetitive Branch-and-Bound Algorithm (RBBA) [Brusco & Stahl 2005]. These algorithms are detailed in section 3.3.2 of Chapter 3. We also demonstrate the benefit of using user-constraints in our model.

5.7.1 Minimizing the maximal diameter of clusters

We compare our previous model (denoted by CP) with the branch-and-bound approach [Brusco & Stahl 2005] (denoted by BaB) and the algorithm based on graph coloring [Delattre & Hansen 1980] (denoted by GC). Our programs are available at `http://cp4clustering.com`. The program BaB can be found at `http://mailer.fsu.edu/~mbrusco/` whereas the program GC is not available and we have coded it in C++ using a well known available graph coloring program [Mehrotra & Trick 1995]. We consider clustering without user-constraints since the other

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
WDBC	569	30	2
Synthetic Control	600	60	6
Vehicle	846	18	4
Yeast	1484	8	10
Multiple Features	2000	10	6
Image Segmentation	2000	19	7
Waveform	5000	40	3

Table 5.1: Properties of datasets

Dataset	D_{opt}	BaB	GC	CP
Iris	2.58	1.4	1.8	< 0.1
Wine	458.13	2	2.3	0.3
Glass	4.97	8.1	42	0.9
Ionosphere	8.6	–	0.6	0.5
User Knowledge	1.17	–	3.7	84.5
WDBC	2377.96	–	1.8	0.7
Synthetic Control	109.36	–	–	56.1
Vehicle	264.83	–	–	14.9
Yeast	0.67	–	–	2388.1
Multi Features	12505.5	–	–	–
Image Segmentation	436.4	–	–	649.2
Waveform	15.6	–	–	–

Table 5.2: Performance with the criterion of minimizing the maximal diameter

algorithms cannot handle them and in the best of our knowledge, there is no exact algorithm handling user-constraints with this criterion. In the experiments, the time-out is set to 1 hour and the Euclidean distance is used to compute the dissimilarity between objects. The number of clusters k is set to the number of real classes given in Table 5.1. Table 5.2 shows the results of our experiments. For each dataset we present the value of D_{opt} (the optimal diameter) in the second column and the run time in seconds of each system. The – sign is used when the system cannot complete the search after 1 hour. All the algorithms are exact and therefore all find the same value for the optimum diameter.

It is clear that with these datasets, our model is the most efficient in all cases and can find the optimal diameter in 10 datasets. Among the programs, BaB algorithm is the least efficient, it is not able to solve datasets with more than 300 objects. The performance of GC is better than that of BaB but it decreases rapidly if the number of objects n is over 500. The BaB algorithm is based on the bounds of the maximal diameter to detect

Dataset	S_{opt}	Total time
Iris	0.53	0.2
Wine	53.33	0.3
Glass	1.78	1.3
Ionosphere	5.29	52.1
User Knowledge Modeling	0.32	423.1
WDBC	421.99	7.1
Synthetic Control	43.59	31.3
Vehicle	27.06	21.9
Yeast	–	–
Multi Features	–	–
Image Segmentation	–	–
Waveform	–	–

Table 5.3: Maximizing the minimal split between clusters with a diameter constraint

failures during the search, while the GC algorithm considers all available dissimilarities in descending order to find the optimum diameter. Our model, which exploits the benefits of Constraint Programming such as constraint propagation and appropriate search strategies, is more efficient.

5.7.2 Maximizing the minimal split between clusters

Finding a partition maximizing the split between clusters is a polynomial problem. However, this is no longer true with user constraints, as for instance with a diameter constraint. To our knowledge, there is no exact algorithm for this criterion that supports any kind of user constraints for a general value $k \geq 3$. We experimented our model by adding a diameter constraint. In order to set it we use the results given in Table 5.2: the diameter of each cluster must not exceed $1.5D_{opt}$. The number of clusters k is set to the actual number of classes given in Table 5.1. The results are given in Table 5.3. For each dataset, we present the optimal split S_{opt} and the total execution time in seconds. Our model is able to solve datasets with less than 1000 objects.

5.7.3 Minimizing the Within Cluster Sum of Dissimilarities (WCSD)

Finding an exact solution for minimizing the WCSD is difficult and we compare our model with the Repetitive Branch-and-Bound Algorithm (RBBA) [Brusco & Stahl 2005]. The program is available at `http://mailer.fsu.edu/~mbrusco/`, which, to our best knowledge, is the best exact algorithm for the WCSD criterion. In these experiments, the dissimilarity is measured as the squared Euclidean distance.

Minimizing the WCSD is a difficult task since the propagation of the sum constraint is less efficient than the propagation of the diameter constraint. Without user-constraints, both our model and the RBBA approach can find the optimal solutions only with the Iris dataset. Our model needs 4174s to complete the search whereas RBBA takes 3249s. However, with appropriate user-constraints, the performance of our model can be significantly improved.

Table 5.4: split constraint and must-link constraints with dataset Iris

split constraint	WCSD	Total time
no constraint	573.552	4174
$\delta = 2\%\ d_{max}$	573.552	1452
$\delta = 4\%\ d_{max}$	573.552	84.4
$\delta = 6\%\ d_{max}$	573.552	0.3
$\delta = 8\%\ d_{max}$	2169.21	0.1
$\delta = 10\%\ d_{max}$	2412.43	<0.1
$\delta = 12\%\ d_{max}$	2451.32	<0.1

# must-link	WCSD	Total time
no constraint	573.6	4174
0.2%	602.6	1275.1
0.4%	602.6	35.6
0.6%	617	16.1
0.8%	622.5	3.5
1%	622.5	1.6
100%	646	<0.1

WCSD and the split constraint Let us add a split constraint δ, where δ ranges from 0% (no constraint) to 12% of d_{max}, where d_{max} is the maximum dissimilarity between two objects in the dataset. Table 5.4 (left) reports the WCSD value of an optimal solution and the total computation time. It shows that when the split constraint is weak, the optimal WCSD value does not change but the computation time decreases significantly when this additional constraint becomes stronger. The reason is that the total number of feasible solutions decreases and the search space is reduced. When the split constraint is weak, propagating this constraint is more time-consuming than its benefits.

WCSD and must-link constraints Let us now add must-link constraints, where the number of must-link constraints, generated from the true classes of objects, varies from 0.2 to 1% of the total number of pairs. The results are expressed in Table 5.4 (right), giving the WCSD value and the total computation time. In fact, the optimal value of WCSD, with no information on classes, does not correspond to the WCSD found when considering the partition of this dataset into the 3 defined classes. The more must-link constraints, the less computation time is needed for finding the optimal value, and the closer to the value of WCSD, when considering the 3 initial classes. The reduction of computation time can be easily explained, since when an object is instantiated, objects that must be linked to it are immediately instantiated too. Furthermore, with any kind of additional constraint, the total number of feasible solutions is always equal or less than the case without constraint.

WCSD and appropriate user-constraints Finding an exact solution minimizing the WCSD is difficult. However, with appropriate combination of user-constraints, the performance can be boosted. Table 5.5 presents some examples where our model can get an optimal solution with different user-constraints, which reduce significantly the search space.

5.7.4 Flexibility of model

Our system finds an optimal solution when there exists one; otherwise our system proves that there is no solution. Let us show the interest of combining different kinds of constraints. Figure 5.9 presents 3 datasets in 2 dimensions, similar to those used in [Ester *et al.* 1996].

Dataset	User-constraints	WCSD	Total time
Wine	$S \geq 0.015 d_{max}$ $30 \leq$ size of each cluster ≤ 100	1.40×10^8	50.6
WDBC	$D \leq 0.8 d_{max}$ $S \geq 0.01 d_{max}$	1.82×10^{10}	1.7
Vehicle	$S \geq 0.03 d_{max}$, $D \leq 0.65 d_{max}$ $30 \leq$ size of each cluster ≤ 300	1.93×10^9	2.3

Table 5.5: Example of appropriate combinations of user-constraints

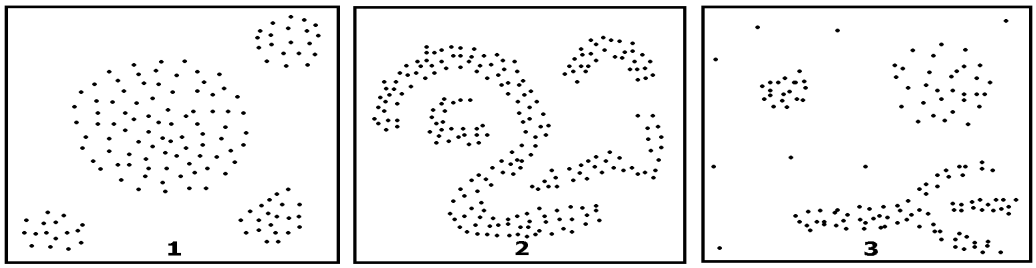

Figure 5.9: Datasets

The first dataset is composed of 4 groups with different diameters. The second one is more difficult, since groups do not have the same shape. The third one contains outliers: outliers are not handled and are therefore integrated in classes.

When optimizing the maximal diameter, the solver tends to find rather homogeneous groups, as shown in Figure 5.10. Adding a split constraint (with $\delta = 5\%$ of the maximum dissimilarity between pairs of points) improves the quality of the solution (see Figure 5.11). Let us notice that maximizing the minimum split allows also to find this solution.

Concerning the third dataset, minimizing the maximum diameter or maximizing the minimum margin do not allow finding a good solution (see Figure 5.12). The quality of the solution is improved when a density constraint is added with $MintPts = 4$ and $\epsilon = 25\%$ from the maximal dissimilarity between pairs of points.

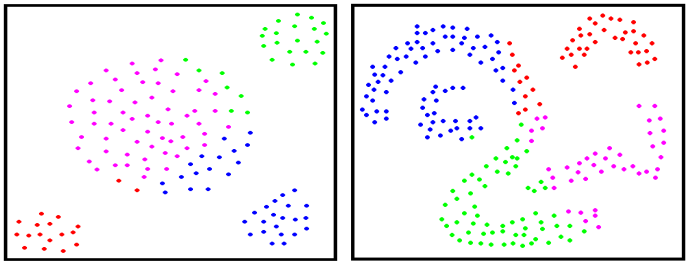

Figure 5.10: Maximal diameter optimization

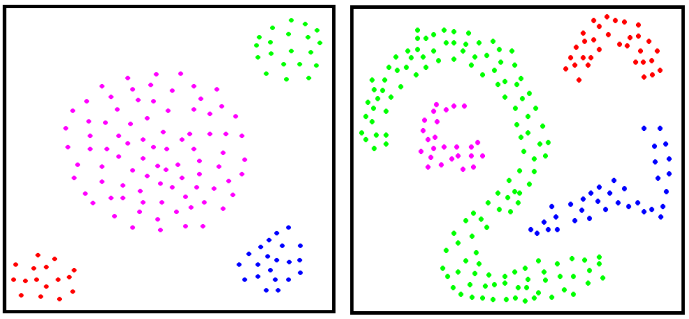

Figure 5.11: Maximal diameter optimization + split constraint

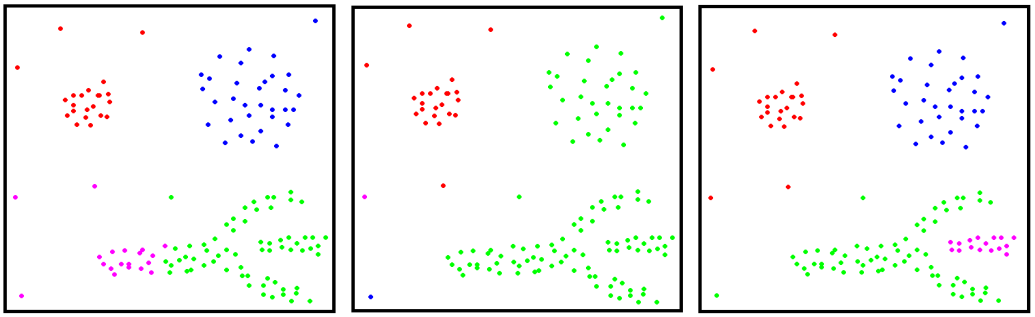

Figure 5.12: Diameter optimization (left) Split optimization (center) Split optimization + density constraint (right)

5.8 Summary

Modeling a problem in Constraint Programming in an efficient and effective way requires a lot of skills and efforts. In practice, in order to find a good model, we may have to try many models before achieving a good formulation. In this chapter we have proposed a general and declarative model in Constraint Programming for solving the problem of Constrained Clustering. Our model allows to choose among different optimization criteria and to integrate various kinds of user-constraints. Experiments on various classical datasets show that our model is competitive with other exact approaches. To increase the efficiency and to deal with larger datasets, we address some analysis that may help us to improve the model:

- The model requires to specify the number of clusters k, which is not always known in the task of clustering. In order to allow more flexible models, where the number of clusters is not fixed, we may need other representations of variables.

- For the criterion of maximum diameter and minimum split, we have to post a large number of reified constraints. Many of these constraints may never help the solver to filter inconsistent values in the domain of variables. For example, for the criterion of minimum split, if the dissimilarity between two points i and j is large, it is obvious that this dissimilarity has nearly zero chance to be the optimal split. As the result, the reified constraints on these points cannot remove values in the domain of variables. However the constraint solver still has to check these constraints every

time it visits a new node in the search tree. Finding other ways for modeling the criteria may speed up the performance.

Chapter 6

A Modular CP Model for Constrained Clustering

Dans ce chapitre, nous proposons un nouveau modèle plus simple, qui est composé d'un seul ensemble de variables affectant à chaque objet l'indice de la classe à laquelle il appartient. Avec ce modèle, l'utilisateur a la flexibilité de ne pas fixer le nombre de clusters à l'avance, mais seulement de donner une borne supérieure k_{max} et une borne inférieure k_{min} sur le nombre de clusters. Le modèle trouve, s'il existe une solution, une partition en k classes, avec $k_{min} \leq k \leq k_{max}$, satisfaisant toutes les contraintes et optimisant globalement le critère spécifié. Afin d'avoir une propagation plus efficace, nous développons deux contraintes globales $diameter(\mathcal{G}, D, d)$ and $split(\mathcal{G}, S, d)$ pour modéliser les critères de diamètre et de marge.

Dans les expérimentations, nous analysons différents aspects de nos deux modèles. Ce nouveau modèle, tout en offrant plus de flexibilité, est plus efficace en temps et en espace que le modèle précédent.

Contents

6.1 Introduction

In this chapter, we propose a second model, still based on Constraint Programming, but significantly different from the one presented in Chapter 5. In the previous model, in order to identify a partition, two sets of variables were introduced, namely a variable for each cluster identifying a cluster by one of its points and a variable for each point assigning it to a cluster. The number of classes had to be fixed beforehand. The new model we present here contains only a variable for each point, giving the number of the cluster the point belongs to. The model is thus lighter in terms of the number of variables, it also enables to remove the restriction on the number of classes, only bounds on the number of classes are required. We develop dedicated filtering algorithms for global constraints for the criteria of minimizing the maximal diameter and maximizing the minimal split between clusters.

6.2 Variables and domains

We present our second CP model, which is based only on a set of variables for the assignment of a number of class to each point. In this model, the number of classes k is only bounded by k_{min} and k_{max}, where k_{min} and k_{max} are given by the user. The difference is significant, the new model offers more flexibility, while it is more efficient than the previous model.

In this model, the clusters are identified by their index, which varies from 1 to k for a partition into k clusters. To represent the assignment of points to clusters, we use a set of integer variables:

$$\mathcal{G} = \{G_1, \ldots, G_n\}$$

The domain of each variable G_i is all possible cluster indices, which ranges from 1 to k_{max}:

$$Dom(G_1) = \cdots = Dom(G_n) = \{1, \ldots, k_{max}\}$$

An assignment $G_i = c$ means the point i is put into the class number c. A complete assignment of those decision variables corresponds to a partition.

For modeling the criteria, we use the same variables as in the previous model: D for the maximal diameter criterion, S for the minimal split criterion and V for the Within-Cluster Sum of Dissimilarities criterion. Figure 6.1 presents a sample data and an assignment of variables in $\mathcal{G}$ which corresponds to a partition. The left image presents the initial domains of the variables in $\mathcal{G}$ in which we want to solve a clustering task into 1 or 2 clusters. The right image presents an assignment that corresponds to a partition of 2 clusters.

6.3 Constraints

6.3.1 Modeling Partition Constraints

Similarly to the previous model, we need constraints to ensure that each possible partition with at least k_{min} clusters and at most k_{max} cluster corresponds to only one complete assignment of variables in $\mathcal{G}$ and vice versa.

Without any constraint, given a partition with k clusters ($k_{min} \leq k \leq k_{max}$), there exists different assignments of variables which correspond to the same partition.

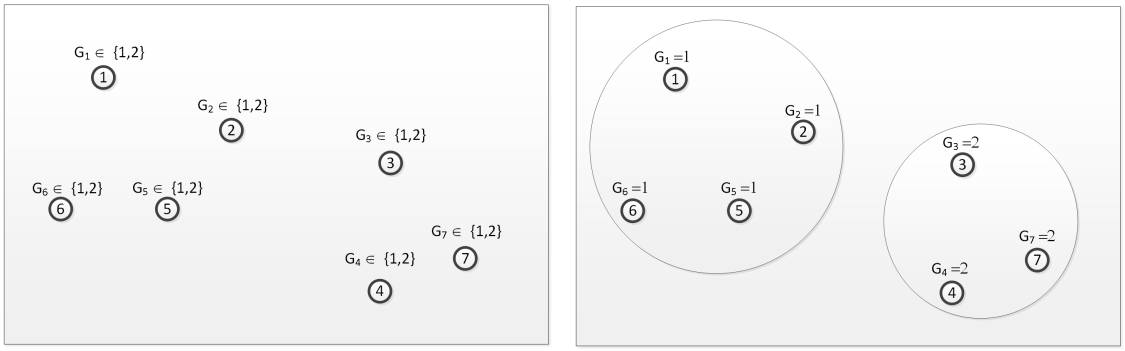

Figure 6.1: Assignment of values to variables

Example 26 Given a dataset of 7 points $\mathcal{O} = \{o_1, \ldots, o_7\}$ and a clustering task dividing the points into 2 or 3 clusters. For solving the clustering task, we define $\mathcal{G} = \{G_1, \ldots, G_7\}$ with the initial domains:

$$Dom(G_1) = \ldots = Dom(G_7) = \{1, 2, 3\}$$

Let us consider a partition of 2 clusters, the first one composed of o_1, o_2, o_5 and o_6 and the second one composed of the remaining points. There are in total 6 assignments of variables, presented in Figure 6.2, which express the same partitions.

From an assignment, we can obtain a symmetric solution by:

- Permutation of the values of variables, for example the solution A and B, C and D, E and F in Figure 6.2.
- Changing the values of variables corresponding to a cluster to another cluster index value that has not yet been assigned, for example the solution A and C, C and E in Figure 6.2.

To avoid the symmetric solutions C, D, E and F, it is simple to put a constraint that: if there are k clusters, then for each value $c \in \{1, \ldots, k\}$, there exists at least one variable G_i that is assigned to c. To avoid the symmetry solution B, a simple way is to post a condition that: For any two cluster indices $c < c'$ that appear in an assignment, if a variable G_i is assigned to c', there must exists a variable G_j that is assigned to c and $j < i$.

The global constraint *precede* [Law & Lee 2004] helps to achieve both requirements above. For avoiding all possible symmetry solutions, all we need is to post a constraint:

$$precede([G_1, \ldots, G_n], [1..k_{max}]).$$

This constraint imposes that $G_1 = 1$ and moreover, if $G_i = c$ with $1 < c \leq k_{max}$, there must exist at least an index $j < i$ with $G_j = c - 1$.

The fact that there are at least k_{min} clusters means that all the numbers among 1 and k_{min} must be used in the assignment of the variables G_i. When using the constraint *precede*, one only needs to require that at least one variable G_i is equal to k_{min}. This is expressed by an AtLeast constraint:

$$AtLeast(\mathcal{G}, k_{min}, 1)$$

Since the domain of each variable G_i is $[1, k_{max}]$, there will be at most k_{max} clusters. If the user needs exactly k clusters, all he/she has to do is to set $k_{min} = k_{max} = k$.

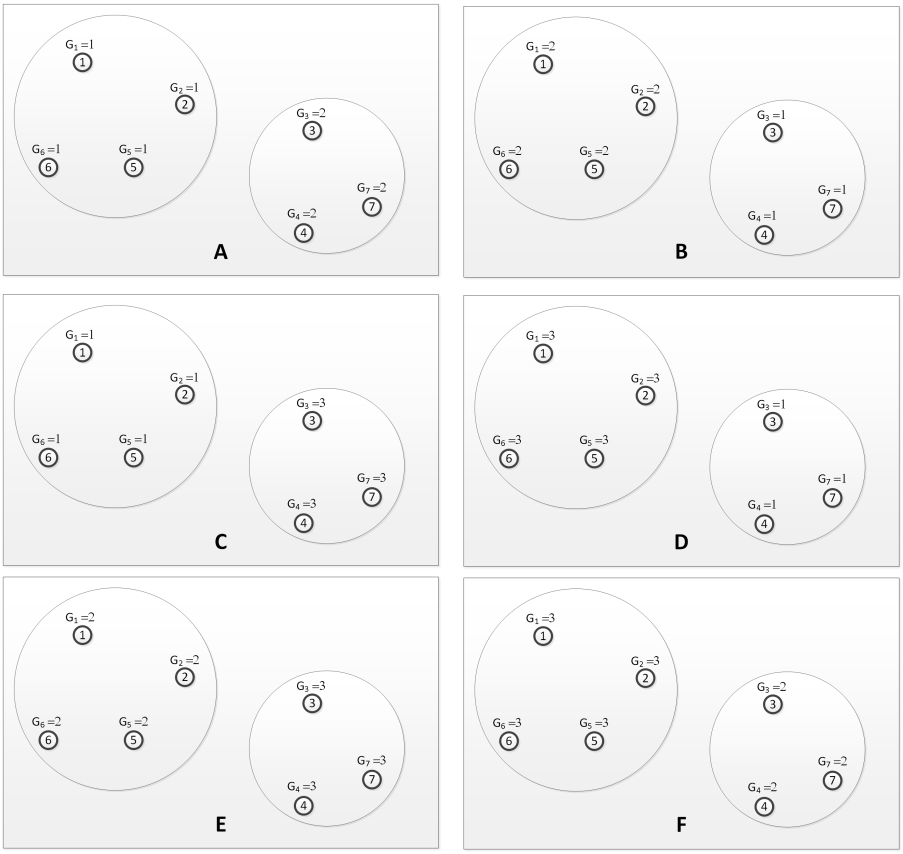

Figure 6.2: Symmetric solutions

6.3.2 Modeling the user-constraints

In our previous model, the user-constraints can be straightforwardly integrated with constraints on variables in $\mathcal{G}$ and $\mathcal{I}$. Although in this new model, the variables in $\mathcal{G}$ express the cluster indices, not the representative, the underlying meaning is the same in the two models: two points i and j are in the same cluster if and only if $G_i = G_j$. There are therefore not many differences in the modeling of user-constraints.

Minimal size α of clusters It requires that each cluster has a number of points greater than a given threshold α. This constraint can be expressed by a constraint AtLeast. If the number of clusters is fixed to k, the constraint can be simply expressed by global cardinality constraints: for each value $c \in \{1, \ldots, k\}$, we post a constraint:

$$AtLeast(\mathcal{G}, c, \alpha).$$

However, in our model, the number of clusters is bounded by two values k_{min} and k_{max}. We, therefore, propose to express the minimal size constraint by n constraints AtLeast: for each value $i \in \{1, \ldots, n\}$, we post a constraint:

$$AtLeast(\mathcal{G}, G_i, \alpha).$$

Each constraint expresses that there must be at least α variables in $\mathcal{G}$ taking the value that is assigned to G_i. This is equivalent with there are at least α points in the cluster that contains point i.

Maximal size β of clusters It requires each cluster to have a number of objects inferior to a predefined threshold β. For each value $c \in \{1, \ldots, k_{max}\}$, we post a constraint AtMost:

$$AtMost(\mathcal{G}, c, \beta).$$

This constraint requires that there must be at most β variables in $\mathcal{G}$ taking the value c.

Split δ-constraint It expresses that the split between any two clusters must be at least δ. Similar to the previous model, for each $i < j \in [1, n]$ satisfying the condition $d_{ij} < \delta$, we put a binary constraint:

$$G_i = G_j.$$

Diameter constraint γ It expresses that the diameter of each cluster must be at most γ. For each $i < j \in [1, n]$ such that $d_{ij} > \gamma$, we put a binary constraint:

$$G_i \neq G_j.$$

Density constraint It requires that for each point i, its neighbourhood of radius ϵ contains at least $MinPts$ points belonging to the same cluster as i. Similar to the previous model, for each point $i \in [1, n]$, we compute the set of variables $\mathcal{X}_i$ which contains variables G_j of points $j \in \{1, \ldots, n\}$ such that $d_{ij} < \epsilon$. We then post a constraint AtLeast:

$$AtLeast(\mathcal{X}_i, G_i, MinPts).$$

It requires that, in the set of variables $\mathcal{X}_i$, there must be at least $MinPts$ variables that is equal to G_i, which means there must be at least $MinPts$ points in the same cluster with point i.

Must-link constraint A must-link constraint on two points i and j is expressed by a binary constraint:

$$G_i = G_j.$$

Cannot-link constraint A cannot-link constraint on i and j is expressed by a binary constraint:

$$G_i \neq G_j.$$

6.4 Criterion Modeling

6.4.1 Modeling the diameter criterion

In our previous model, the criterion of diameter is expressed by implication constraints:

$$d_{ij} > D \rightarrow (G_i \neq G_j) \tag{6.1}$$

For modeling each implication constraint, in most constraint solvers, we have to use several primitive constraints. For instance, we proposed to use two reified constraints and one binary constraint for modeling each implication constraint, as presented in the previous chapter. In order to have better interactions and propagation between these relations, a better way is to sum up these relations into one global constraint with a dedicated filtering algorithm.

We have developed a global constraint for the criterion of maximum diameter, denoted by $\mathit{diameter}(\mathcal{G}, D, d)$, which exploits the dissimilarity measure d between any two points and which operates on the set of variables $\mathcal{G}$ and the variable D. The constraint $\mathit{diameter}(\mathcal{G}, D, d)$ ensures that D is the maximal diameter of the clusters formed by the variables $G_1, \ldots, G_n$. This means, for every couple $1 \leq i < j \leq n$, this constraint ensures the constraints in (6.1). By developing the constraint $\mathit{diameter}(\mathcal{G}, D, d)$, we maintain all these relations in one constraint, which makes the model much less heavy. The filtering algorithm is presented in Algorithm 13, where $Dom(D) = [D.min, D.max)$. In this algorithm, if $D.max$ has been changed and if $D.max < d_{ij}$, the relation (6.1) will infer $G_i \neq G_j$, but we filter the domain of G_i (or G_j) only if the other variable has been instantiated. Otherwise, if some variables in $\mathcal{G}$ have been instantiated, if $G_i = G_j$, with the relation (6.1) we infer $D \geq d_{ij}$ so the lower bound $D.min$ can be changed. Let us notice that as soon as the domain of one variable becomes empty, a failure case is detected by the solver. The worst case complexity is $O(n^2)$.

6.4.2 Modeling the split criterion

For the criterion of split, we have developed a similar global constraint $\mathit{split}(\mathcal{G}, S, d)$ ensuring that for every couple $1 \leq i < j \leq n$:

$$S > d_{ij} \;\rightarrow\; G_i = G_j. \tag{6.2}$$

The filtering algorithm is presented in Algorithm 14, where $Dom(S) = (S.min, S.max]$. In this algorithm, if $S.min$ has been changed and if $S.min \geq d_{ij}$, the relation (6.2) will infer $G_i = G_j$, which is propagated by enforcing $Dom(G_i) = Dom(G_j)$. Otherwise, in line 18, in case some variables G_j have been instantiated, if $G_i \neq G_j$ by (6.2) we infer $S \leq d_{ij}$, so the upper bound of S can be changed. The worst case complexity is $O(n^2)$.

Algorithm 13: Filtering for the constraint $diameter(\mathcal{G}, D, d)$

```
1  stack ← ∅;
2  if D.max has been changed then
3  |  for i ← 1 to n where G_i is instantiated do
4  |  |  stack ← stack ∪ {i};
5  |  end
6  else
7  |  foreach i that G_i has just been instantiated do
8  |  |  stack ← stack ∪ {i};
9  |  end
10 end
11 foreach i ∈ stack do
12 |  for j ← 1 to n do
13 |  |  if d_ij ≥ D.max then
14 |  |  |  delete value of G_i from Dom(G_j);
15 |  |  end
16 |  |  if G_j is instantiated ∧ G_i = G_j then
17 |  |  |  D.min ← max(D.min, d_ij);
18 |  |  end
19 |  end
20 end
```

6.4.3 Modeling the WCSD criterion

For the WCSD criterion, we use the same constraint $wcsd(\mathcal{G}, W, d)$, presented in section 5.4.3 of Chapter 5. This constraint relies only on variables in $\mathcal{G}$, and more specifically, only on the condition $G_i = G_j$. The filtering algorithm for this constraint is therefore mostly unchanged in this model. The only difference is, in this model, the number of clusters k is not fixed but $k_{min} \leq k \leq k_{max}$. Therefore, for calculating the low bound of WCSD, the number of clusters k is set as the maximum number of clusters: $k = max(\cup_{i \in [1,n]} Dom(G_i))$.

6.5 Search strategy

To improve the efficiency, points are beforehand reordered following the strategy presented in section 5.5 in the previous Chapter. For the search strategy, branching is realized on the variables of $\mathcal{G}$.

For the diameter and split criteria At each branching point, a variable G_i with the smallest remaining domain is chosen. With this variable G_i, all the values c in the domain of G_i are examined and the number of the closest cluster to i is chosen. The dissimilarity between a point i and a cluster number c is defined as the maximal dissimilarity d_{ij} where G_j is assigned and $G_j = c$. If the cluster number c is empty (there is no point j such that $G_j = c$), the dissimilarity between i and the cluster c is null. This means that the

Algorithm 14: Filtering for the constraint $split(\mathcal{G}, S, d)$

```
1  stack ← ∅;
2  if S.min has been changed then
3  |  for i ← 1 to n where G_i is instantiated do
4  |  |  stack ← stack ∪ {i};
5  |  end
6  end
7  else
8  |  foreach i that Dom(G_i) has just been changed do
9  |  |  stack ← stack ∪ {i};
10 |  end
11 end
12 foreach i ∈ stack do
13 |  for j ← 1 to n do
14 |  |  if d_ij ≤ S.min then
15 |  |  |  Dom(G_i) ← Dom(G_i) ∩ Dom(G_j);
16 |  |  |  Dom(G_j) ← Dom(G_i);
17 |  |  end
18 |  |  if G_j is instanciated ∧ G_i ≠ G_j then
19 |  |  |  S.max ← min(S.max, d_ij);
20 |  |  end
21 |  end
22 end
```

assignment of a point to a new cluster is favoured if there are unused cluster numbers. In this case, the least remaining number is chosen. The closest cluster c to the point i is chosen and two alternatives are created $G_i = c$ and $G_i \neq c$. This strategy, which takes into account all the points already assigned to decide the branching, is different from the one used in the previous model, where the branching depends on the dissimilarity between the point i and the representative of each cluster.

For the criterion WCSD The same strategy is used as in the previous model. For finding the first partition, the chosen variable G_i and value c are those such that the assignment $G_i = c$ increases WCSD as least as possible. After finding the first solution, the strategy is changed following the first fail principle. A value s_{ic} for each point i and each cluster c is defined as the sum of the squared dissimilarity between i and all points j already assigned to the cluster c. At each branching point, for all points i with G_i unassigned, the minimal value $s_i = \min_{c \in Dom(G_i)} s_{ic}$ is computed. The variable G_i with the smallest value s_i is then chosen and the value $c = \arg\min s_{ic}$ is chosen. Two alternatives where $G_i = c$ and $G_i \neq c$ are created for the branching.

6.6 Experiments

6.6.1 Comparison of two models

We compare our first model (denoted by CP1) with our second model (denoted by CP2) in order to study the following questions:

- **Q1**: Do both models eliminate all symmetry solutions?
- **Q2**: Which one better models variables and partition constraints?
- **Q3**: How do the dedicated global constraints for the criteria of maximum diameter and minimum split help to improve the performance?

To answer the questions **Q1** and **Q2**, we run both models, without giving any criterion or user-constraints, to find all partitions for a clustering task with 2 clusters. Both models CP1 and CP2 are implemented on Gecode solver version 4.2.1. The experiments are performed on a 3.4GHz Intel Core i5 processor running Ubuntu 12.04. They both use the same branching rule: in CP1, variables are chosen in order $I_1, \ldots, I_k$ then $G_1, \ldots, G_n$ whereas in CP2, variables are chosen in order $G_1, \ldots, G_n$. For the choice of values for each variable, the values are enumerated in the ascending order. The choice of the dataset is not important as the dissimilarities between points are not used in partition constraints. The number of partitions depends only on the number of points n and the number of clusters k.

Theorem 23 *Given a number of points n and a clustering task that divides these points into 2 clusters, the number of different partitions, when ignoring the index of each cluster in a partition, is given by a function $f(n)$, which is defined by the recursive relation:*

$$f(n) = \begin{cases} 1 & \text{if } n = 2, \\ 2 \times f(n-1) + 1 & \text{if } n > 2. \end{cases}$$

		CP1		CP2		
n	$f(n)$	#Solutions	Time	#Solutions	Time	Performance gain
18	131071	131071	0.5	131071	0.3	167%
19	262143	262143	1.1	262143	0.7	157%
20	524287	524287	2.3	524287	1.4	164%
21	1048575	1048575	4.8	1048575	3	160%
22	2097151	2097151	10	2097151	6.1	164%
23	4194303	4194303	20.8	4194303	12.8	163%
24	8388607	8388607	42.8	8388607	26.1	164%
25	16777215	16777215	88.8	16777215	53.7	165%

Table 6.1: Comparison of the two models without optimization criterion

Proof It is obvious that in the case there are two points $\{o_1, o_2\}$, there is only one possible partition P with 2 clusters, where each cluster contains one point:

$$P = (C_1 = \{o_1\}, C_2 = \{o_2\})$$

In case $n > 2$, for each partition $P = (C_1, C_2)$ of $n-1$ points $\{o_1, \ldots, o_{n-1}\}$ into 2 clusters, there are two corresponding partitions with n points $\{o_1, \ldots, o_n\}$:

$$P_1 = (C_1 \cup \{o_n\}, C_2), P_2 = (C_1, C_2 \cup \{o_n\}).$$

Moreover, with n points, we have one more partition: $(\{o_1, \ldots, o_{n-1}\}, \{o_n\})$. In total, there are $2 \times f(n-1) + 1$ partitions of n points into 2 clusters.

If there are no symmetric solutions, the total number of solutions given by our models must be equal to the number of partitions given by the function $f()$. Table 6.1 shows the results of our experiments. We present in the first column the number of points n and in the second column the value of $f(n)$. The third and fourth columns present the number of solutions and the total run time taken by CP1. The next two columns express the results corresponding to CP2. The last column expresses the speed up from CP1 to CP2, given in percentage.

It is clear that both our models find the same number of solutions as the value of $f(n)$. It shows that, in practice, there is no symmetry solutions found by both models. The model CP2 is 60% faster than CP1. In all the tests, the number of nodes in the search tree is identical in both models. It shows that the modeling of variables and partition constraints in CP2 is more efficient than CP1.

To respond the question **Q3**, we take the same datasets and the same experimentation as presented in previous chapter in Table 5.1.

For the criterion of maximal diameter We consider the clustering task without user-constraints, and the number of clusters k is fixed (for CP2: $k_{min} = k_{max} = k$). Similarly to the previous chapter, in the experiments, the time-out is set to 1 hour and the Euclidean distance is used to compute the dissimilarity between objects. The number of clusters k is set to the real number of classes given in Table 5.1. Table 6.2 shows the results of our experiments. We present in the first column the datasets. The second and

Dataset	CP1 #Nodes	Time	CP2 #Nodes	Time	Performance gain
Iris	435	< 0.1	307	< 0.1	Unidentified
Wine	1241	0.3	821	< 0.1	Unidentified
Glass	2679	0.9	1264	0.2	450%
Ionosphere	2917	0.5	1535	0.3	167%
User Knowledge	10151	84.5	1413	0.2	422%
WDBC	569	0.7	569	0.4	175%
Synthetic Control	3537	56.1	1537	1.6	3500%
Vehicle	38781	14.9	846	0.9	1656%
Yeast	219622	2388.1	16467	5.2	45925%
Multi Features	-	–	7039	10.4	Unidentified
Image Segmentation	23568	649.2	6047	5.7	11389%
Waveform	-	–	9021	50.1	Unidentified

Table 6.2: Performance of the two models with the criterion of minimizing the maximal diameter

third column present the number of nodes in the search tree and the total run time taken by CP1. The two next columns express the results corresponding to CP2. The last column expresses the performance gain from CP1 to CP2, given in percentage. It is clear that the model CP2 outperforms CP1 on all datasets. All twelve datasets can be solved by CP2 within one minute. Although our two models are based on the Constraint Programming framework, there are three reasons that explain the significant differences in performance of the two models.

- The modeling of variables and partition constraints: the improvement is clear as explained in question Q_2.
- The heuristic of branching: in CP2, for selecting the value for branching, we consider all assigned variables, whereas in CP1, only the representative point is measured. This improvement is significant, in some datasets (for example: User Knowledge, Vehicle, Yeast, Image Segmentation) there are much less number of nodes in the search tree generated by CP2 than CP1. As a result, the total run time of CP2 in those datasets is much faster than CP1. Figure 6.3 compares the diameter of the first solution found by both CP1 and CP2. We compute the ratio of the diameter found in the first solution to the optimal Diameter. With a good heuristic on branching, the ratio should be close to 1. CP2 has a better ratio in most datasets, except for the dataset Synthetic Control. In many tests, the ratio in CP2 is less than 1.1, which means that the diameter of the first solution is very close to the optimal diameter. It explains the boost on performance with the second model.
- The dedicated global constraint for the criterion of diameter: The improvement is clearer with larger datasets. For example with the dataset Yeast, the number of nodes in CP1 is 13 times more than CP2 but CP2 is 459 times faster than CP1. The reason is, in CP1, for modeling the criterion, the number of reified constraints is a square function of the number of points n. Although our dedicated global constraint

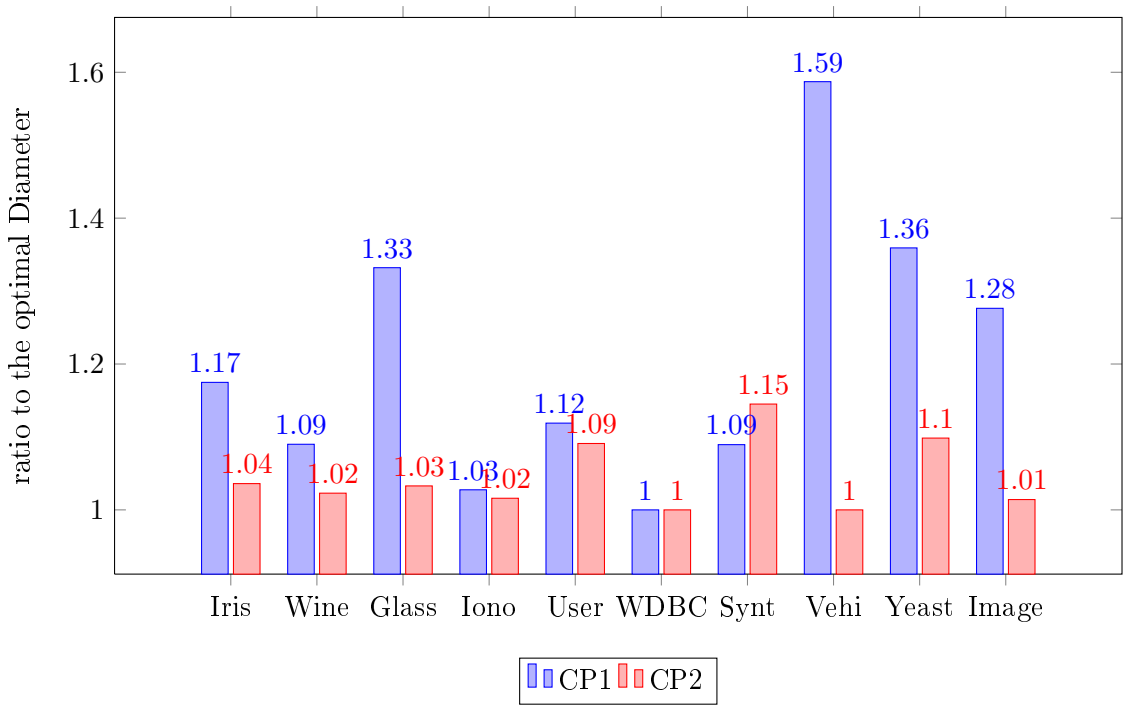

Figure 6.3: Comparison of CP1 and CP2 with the criterion of Diameter on 1st solution

has a complexity in the worst case of $O(n^2)$, it considers only necessary variables whereas in CP1, at each node, every reified constraints are checked at least one time. This improvement is important and allows us to solve larger datasets.

For the criterion of split We redo the experiments of split done in the previous chapter, with the second model (CP2). We consider the clustering task with the criterion of split and with an user-constraint: the diameter of each cluster must not exceed $1.5D_{opt}$, where D_{opt} is the optimal diameter. In addition, we experiment the second model in two cases: the number of classes k is fixed to the actual number of classes, and the number of classes k is bound from 2 to the actual number of classes. Table 6.3 presents the results. Similar to the criterion of diameter, the performance of CP2 is much better than that of CP1. There is a slight increase in time taken in the second case, as the model has to consider more partitions.

6.6.2 Experiments on user-constraints

We evaluate the effectiveness of our constrained clustering model (CP2), in term of the quality of the optimal partition with user-constraints and a given criterion. For measuring a partition, we consider the Adjusted Rand Index (ARI) [Hubert & Arabie 1985]. The ARI measures the similarity between two partitions, in this case, the true partition P of the dataset and the partition P' found by our model. The ARI is defined as:

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

	Case 1			Case 2	
	S_{opt}	CP1	CP2	S_{opt}	CP2
Iris	0.53	0.2	< 0.1	0.53	< 0.1
Wine	53.33	0.3	< 0.1	53.33	< 0.1
Glass	1.78	1.3	0.1	1.78	0.7
Ionosphere	5.29	52.1	1.2	5.29	1.2
User Knowledge	0.32	423.1	2.8	0.32	2.9
WDBC	421.99	7.1	0.6	421.99	0.6
Synthetic Control	43.59	31.3	6.4	45.2	6.5
Vehicle	27.06	21.9	3.7	27.06	3.7
Yeast	0.15	-	106.3	0.15	106.3
Multi Features	1107.1	-	82.2	1107.1	82.9
Image Segmentation	228.7	-	20.4	-	20.6
Waveform	-	-	-	-	

Table 6.3: Comparison with the criterion of split

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'.

We consider the instance-level constraints: must-link and cannot-link constraints. The constraints are generated following the method in [Wagstaff & Cardie 2000]: two points are chosen randomly from the dataset, if they are in the same cluster in P, a must-link constraint is generated, otherwise a cannot-link constraint is generated. For the criterion, we consider the partition given by:

a. the first solution with the criterion of Diameter.

b. the optimal solution with the criterion of Diameter.

c. the first solution with the criterion of WCSD.

d. the optimal solution with the criterion of Split.

The experiments are on the two datasets Iris and Wine. Surprisingly, the criterion of split cannot solve these datasets with pairwise constraints. The underlying reason is that in these datasets, there are two clusters that are very close. As a result, the minimum split in the real partition is close to 0. The propagation of the constraints for the criterion of split is poor and the search space is too large. The solver has to consider many partitions and cannot end the search in one hour. We report in Figure 6.4 and 6.5 the average ARI over 100 constraint sets for 0 (unconstrained), $30, 60, \ldots, 300$ constraints randomly generated from the data. In both datasets, the ARI is improved as the number of constraints increases. For the dataset Wine, the ARI of partitions found without user-constraint is typically worse. However, the ARI improves when there are more than 200 constraints (200 is equivalent to 1.27% of the number of pairs of points in this dataset).

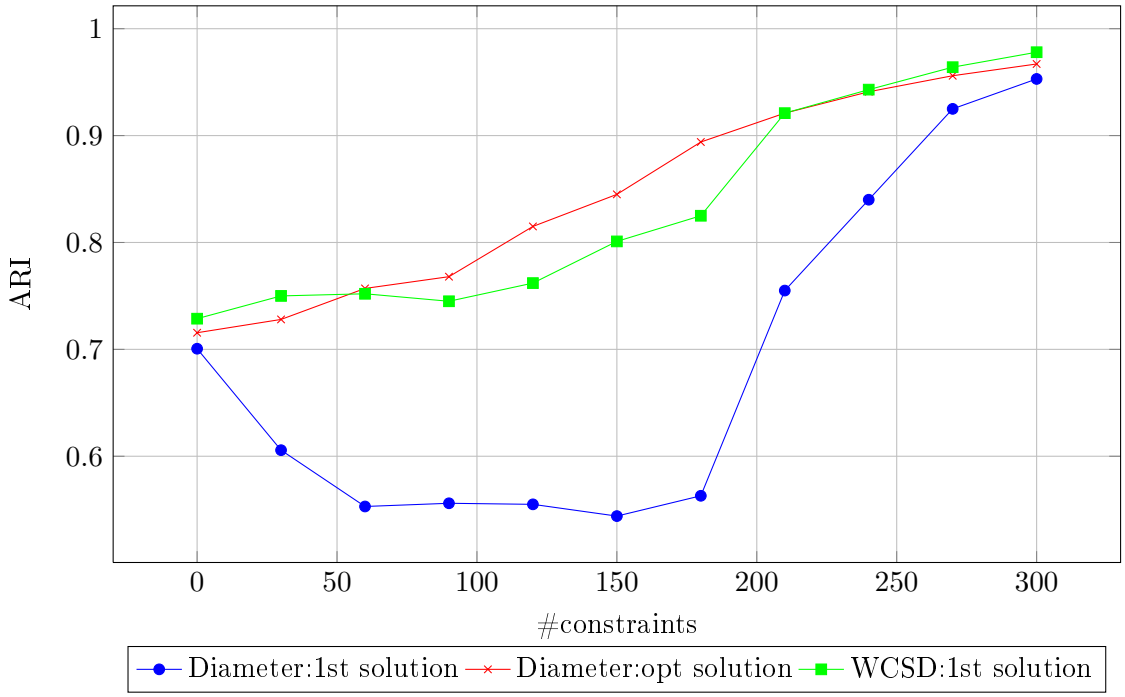

Figure 6.4: Dataset Iris with user-constraints

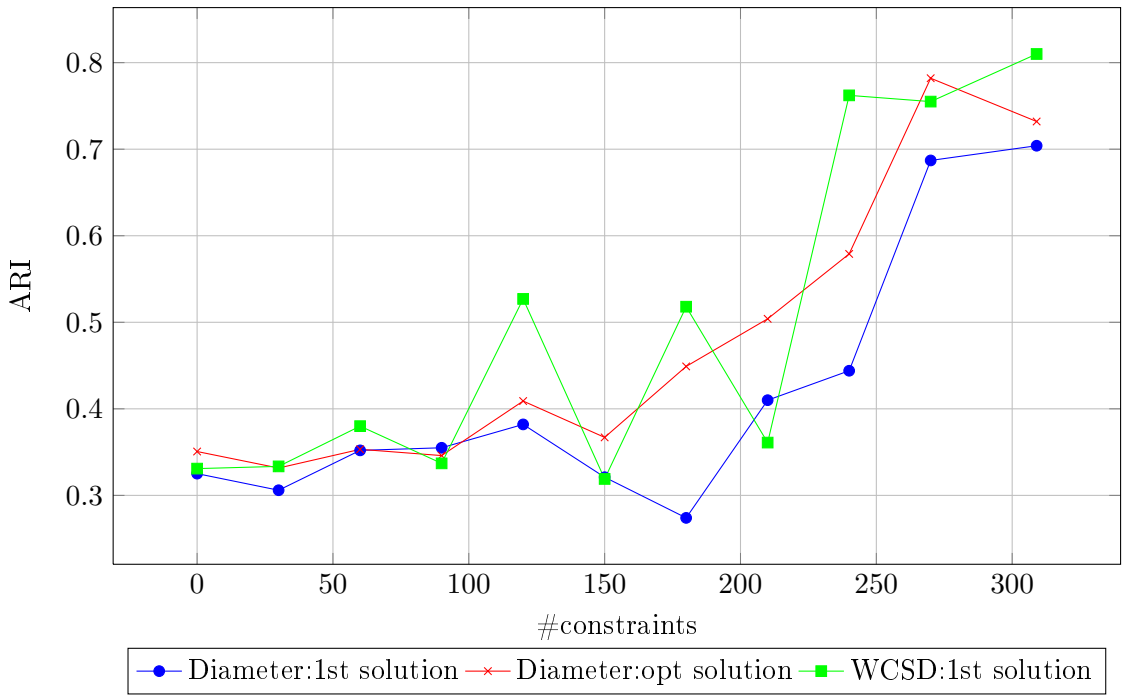

Figure 6.5: Dataset Wine with user-constraints

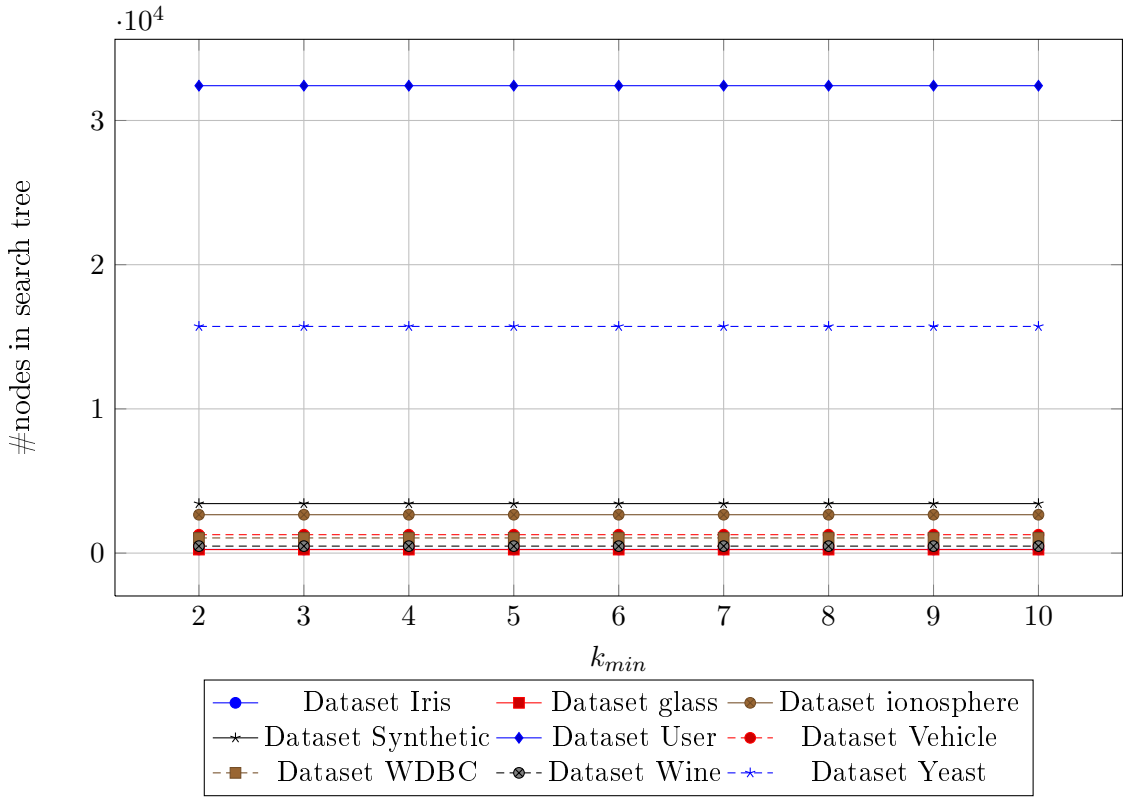

Figure 6.6: Experiments on bound of k: $k_{min} = 2 \dots 10$, $k_{max} = 10$

6.6.3 Analysis of bounds on the number of clusters

We evaluate the influence of the bounds on the number of clusters $k \in [k_{min}, k_{max}]$ on the performance with the diameter criterion. In the first experiment, we set $k_{max} = 10$ and we increment k_{min} from 2 to 10. The diameter criterion favours the high number of clusters such that the higher the number of clusters, the smaller the value of the optimal diameter. Without user-constraints, the optimal solution with the criterion of diameter always has a number of clusters equal to k_{max}. The result is given in Figure 6.6 where we report the number of nodes in the search tree, for each dataset, when we vary the bound k_{min}. For all datasets, the number of nodes in the search tree is constant when k_{min} is changed. By changing the bound, the performance of our model remains the same. It shows that the propagation of the constraint of diameter is effective. After finding and proving the optimal solution with k_{max} clusters, the solver can conclude that there does not exist a better solution, with less than k_{max} clusters.

In the second experiment, we fix $k_{min} = 2$ and we increment k_{max} from 2 to 10. The result is given in Figure 6.7 where we report the number of nodes in the search tree, for each dataset, when we increment the bound k_{max}. In general, when k_{max} increases, there are more partitions to be considered. However, we see an interesting trend in Figure 6.7. In many datasets, the number of nodes in the search tree does not increase as k_{max} increases. Indeed, since k_{max} is higher, the optimal maximum diameter is smaller, and propagation of the diameter constraint is more effective. It explains that in some cases, it takes less computation time when k_{max} increases.

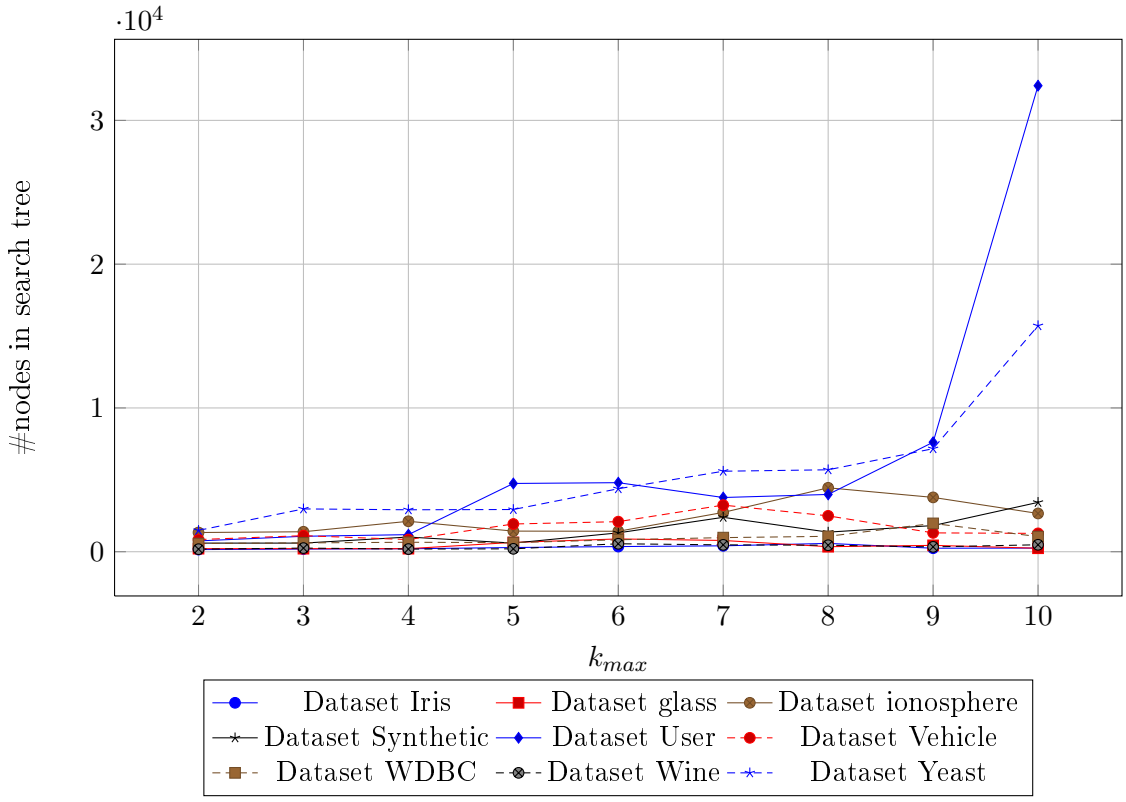

Figure 6.7: Experiments on bound of k: $k_{min} = 2$, $k_{max} = 2, \ldots, 10$

6.7 Summary

In this chapter, we proposed a second model in Constraint Programming for solving the problem of Constrained Clustering. Its goal is to maximally improve the performance as well as to give more flexibility to users . Our first model presented in Chapter 5 is based on a two-level representation: a set of variables for the assignment of a representative for each class and a set of variables for the assignment of a representative for each point. This choice of variables requires that the number of classes k is fixed beforehand, since each representative is modelled by a CP variable. The second model is based only on a set of variables for the assignment of a number of class to each point. In this model, the number of classes k is only bounded by k_{min} and k_{max} given by the user. We designed two new global constraints to replace a large number of reified constraints for the criteria of diameter and split. The difference between two models is significant. We demonstrated that while the second model is more modular and offers more flexibility, it is more efficient than the first model.

Chapter 7

Bi-criteria constrained clustering by CP

Dans ce chapitre, nous montrons que nos modèles peuvent être intégrés dans une procédure plus générale et nous l'illustrons par la recherche de la frontière de Pareto dans un problème de clustering bi-critères sous contraintes utilisateur.

Dans la tache de clustering, le critère de diamètre vise à trouver des clusters compacts mais souffre souvent de l'effet de découpage [Cormack 1971], *i.e.* des objets assez similaires peuvent être classés dans des classes différentes pour réduire le diamètre. Par contre, le critère de la marge peut souffrir de l'effet de chaîne [Johnson 1967], *i.e.* des objets très différents peuvent se trouver dans une même classe. Pour éviter ces problèmes, on peut considérer ces critères en même temps. Nous présentons dans ce chapitre la tâche de clustering bi-critères, à savoir maximiser la marge entre clusters et minimiser le diamètre maximal des clusters. Pour résoudre ce problème, une approche générale est de chercher la frontière de Pareto.

Nos modèles offrent à l'utilisateur le choix du critère d'optimisation et la capacité d'intégrer différents types de contraintes utilisateur. Nous montrons que la généricité et la déclarativité de nos modèles permettent de l'utiliser directement pour traiter le clustering bi-critères de marge-diamètre et sous contraintes utilisateur. Ceci est réalisé par des optimisations mono-critère successives utilisant des contraintes utilisateur. Des expérimentations sur des bases de données classiques montrent que l'utilisation de notre modèle est plus efficace que les algorithmes exacts existants.

Contents

We show in this chapter that our model can easily be embedded in a more general process and we illustrate this on the problem of finding the Pareto front of a bi-criterion optimization process.

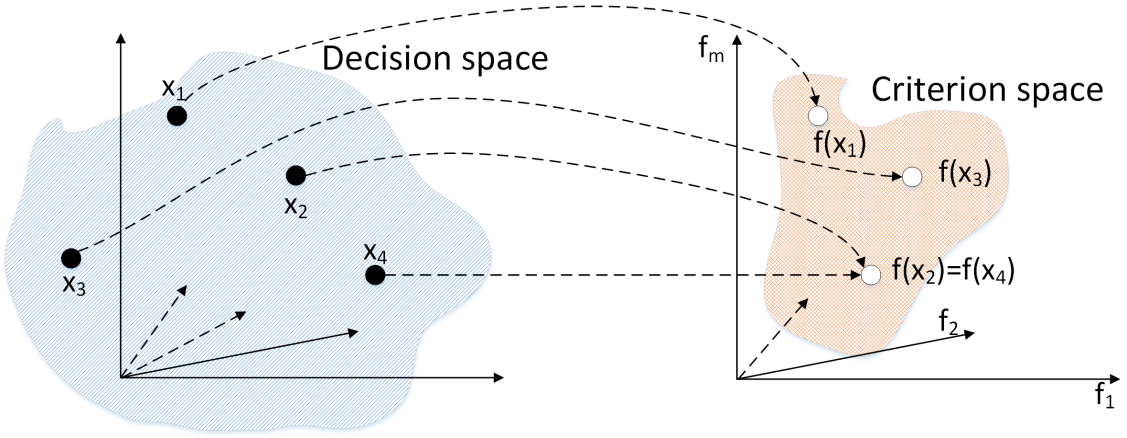

Figure 7.1: Illustration of the decision and the criterion space

7.1 Multi-objective Optimization

In real world, many problems are concerned with different conflicting criteria. For instance, when we consider to buy a new car, we wish to get a car that has high speed, high durability, low petrol consumption, but with low price. The four criteria are clearly in conflict and it is impossible to find such a car that is the best in each criterion whereas finding the best car with any of the four criteria alone is easy. This is an example of a multi-objective optimization problem. In mathematics, multi-objective optimization is the problem of optimization where two or more objective functions are considered simultaneously. A Multi-objective Optimization Problem (MOP) is defined as follows [**?**]:

$$Minimize\ f(x) = [f_1(x), \ldots, f_m(x)]^T$$

$$\text{subject to}\ \ x \in \mathcal{X}$$

where m is the number of criteria. The vector $x \in \mathbb{R}^n$ is composed of n decision variables. Each vector x, also called point x, presents a solution in the multi-objective optimization problem. $\mathcal{X}$ is the decision space, also called the feasible design space, which describes the feasible solutions that can be made. The criterion space $\mathcal{Z}$ is defined as the set $\{f(x)|x \in \mathcal{X}\}$. The vector function $f : \mathbb{R}^n \to \mathbb{R}^m$ is formed by m objective functions $f_i : \mathbb{R}^n \to \mathbb{R}$. In the case where some objective functions are to be maximized, it is equivalent to minimize its negative.

Figure 7.1 expresses an example of the decision space $\mathcal{X}$ and the criterion space $\mathcal{Z}$. Note that a point in the criterion space may correspond to more than one point in the decision space.

Typically, the objective functions conflict each others and there is no single optimal solution that optimizes all objective functions. For evaluating solutions of MOPs, it is necessary to define how to compare different objective vectors $f(x)$ for different solutions $x \in \mathcal{X}$. A common way to define it is using the Pareto dominance relation [Pareto 1896].

Definition 24 *Given two vectors of function values $f(x_1)$ and $f(x_2) \in \mathcal{Z}$, w.r.t two solutions x_1 and x_2. We say the vector $f(x_1)$ Pareto-dominates $f(x_2)$, denoted by $f(x_1) \prec$*

$f(x_2)$, if and only if the solution x_1 is no worse than x_2 in all objectives and the solution x_1 is strictly better than x_2 in at least one objective.

$$f(x_1) \prec f(x_2) \quad \textit{iff} \quad \begin{cases} f_i(x_1) \leq f_i(x_2) & \forall i \in 1, \ldots, m, \\ \exists j \in 1, \ldots, m & f_j(x_1) < f_j(x_2). \end{cases}$$

For simplification, we also say that solution x_1 dominates solution x_2, denoted by $x_1 \prec x_2$, if $f(x_1) \prec f(x_2)$.

From the definition of the Pareto dominance relation, it is obvious that the "best" solutions of a MOP should not be dominated by any other solutions. The solutions that are not dominated by any other solutions are called Pareto optimal solution, defined as follows:

Definition 25 *A feasible solution $x^* \in \mathcal{X}$ is called Pareto optimal, if there is no other $x \in \mathcal{X}$ such that $f(x) \prec f(x^*)$.*

In practice, a common way to solve a MOP is finding the Pareto optimal set, or the Pareto Front set.

Definition 26 *The Pareto optimal set of solutions $\mathcal{P}^*$ is defined as follows:*

$$\mathcal{P}^* = \{x \in \mathcal{X} \mid \nexists y \in \mathcal{X} : f(y) \prec f(x)\}$$

Definition 27 *The Pareto Front set $\mathcal{PF}^*$ of a MOP is the projection of its Pareto optimal set in the criterion space.*

$$\mathcal{PF}^* = \{f(x) \mid x \in \mathcal{P}^*\}$$

Example 27 Let us consider a MOP with two objective functions f_1 and f_2 to be minimized. Figure 7.2 illustrates an example of Pareto optimal set and Pareto front set. In this example, solution x_1 does not dominate any other solutions, as $f_2(x_1)$ is the maximized value. However, the solution x_1 is Pareto optimal, as it is not dominated by any other solutions. The solution x_3 is not Pareto optimal. The reason is that $f_1(x_3) > f_1(x_2)$ and $f_2(x_3) > f_2(x_2)$, so solution x_3 is dominated by solution x_2.

7.2 Bi-criterion Constrained 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 diameter 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 group very different objects in the same cluster. The popular WCSS criterion minimizes the sum of the squared distances between points and the center of their cluster. With this criterion sometimes objects which should be in a large group may be classified in different clusters in order to keep this sum small. Figure 7.3 gives an illustration of these effects. Image A shows three groups that are easily recognized. Image B shows the obtained solution with the diameter criterion when the

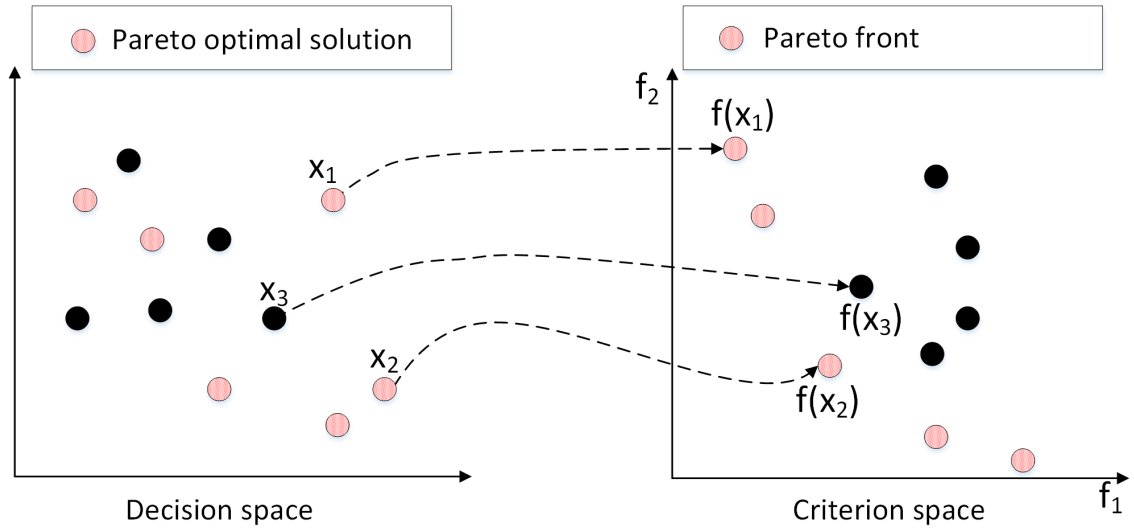

Figure 7.2: Illustration of Pareto optimal set and Pareto front set

number of clusters is set to 3. In this partition, there are points which are very closed but which are classified in two different groups. Considering the split criterion, the obtained partition is shown in Image C. Because of the chain effect, the largest group contains points which are very far each from other. The exact solution with the WCSS criterion is shown in Image D. In this partition, there are also points which are very closed but which are grouped in different clusters.

An ideal partition of homogeneous and well-separated clusters should have a minimum diameter and a maximum split. Unfortunately, such a partition in general does not exist, since the two criteria are often conflicting. [Delattre & Hansen 1980] suggests to consider the bi-criterion optimization problem of maximizing the minimal split between clusters and minimizing the maximal cluster diameter. This bi-criterion is natural and captures both the homogeneity and the separation requirements for a good clustering. [Delattre & Hansen 1980] proposes an algorithm for searching the complete Pareto front set, w.r.t these two criteria. In this thesis, we consider the same bi-criterion problem, but with the user-constraints. As presented in section 7.1, a Pareto optimal solution is a solution for which it is impossible to improve the value of one criterion without degrading the value of the other criterion. Considering the two criteria of diameter and split, given two partitions Δ' and Δ, we say Δ' *dominates* Δ, denoted by $\Delta' \prec \Delta$, if and only if:

$$D(\Delta') \leq D(\Delta) \quad \text{and} \quad S(\Delta') > S(\Delta)$$

or

$$D(\Delta') < D(\Delta) \quad \text{and} \quad S(\Delta') \geq S(\Delta).$$

A partition Δ is *Pareto optimal* if and only if there is no other partition Δ' that dominates Δ. 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 [Wang *et al.* 1996]. For instance, when the number of classes is set to 3, the Pareto optima of the example in Figure 7.3 are given in Figure 7.4. If we minimize the ratio S/D, the optimal solution is the solution 5, which seems the most reasonable. The user can specify

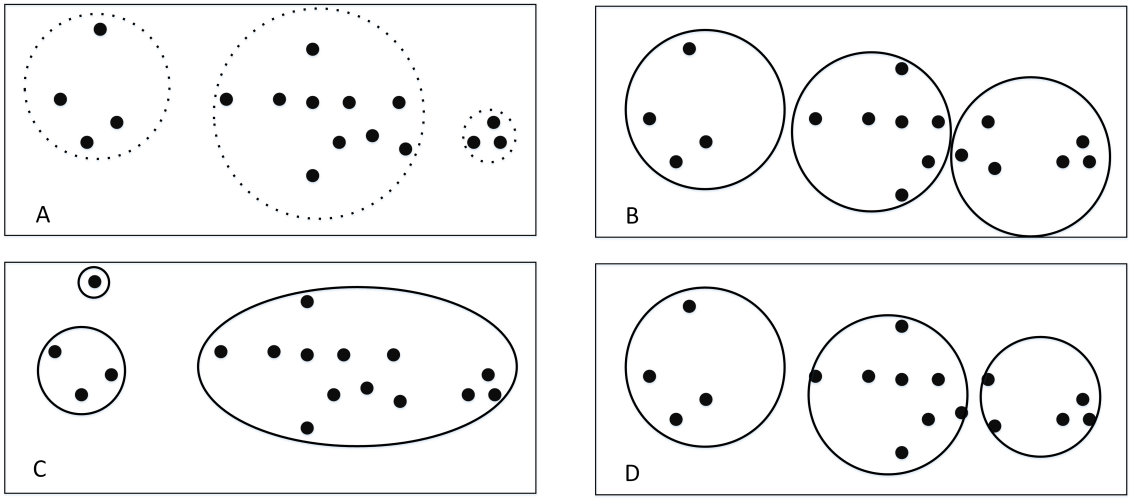

Figure 7.3: Illustration of effects with different criteria

conditions on desired solutions, for instance points 5 and 14 must be in the same cluster. In this case only solutions 5 and 6 are found. If in addition of this condition another condition is added, requiring that the size of each group must be at least 2, only the solution 5 is found. A bi-criterion clustering algorithm finding the complete Pareto front set for different numbers of clusters k is proposed in [Delattre & Hansen 1980]. In case of $k = 2$, 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 any kind of user constraints.

7.3 CP Model for Bi-criterion Clustering with user-constraints

Our CP model represents a general and declarative framework for constrained clustering, where a user can choose one among different optimization criteria and can integrate different kinds of user constraints. This flexibility offers different possibilities of usage. We present in this section the use of our model to deal with bi-criterion constrained clustering tasks. We are given a constrained clustering task with a set of user constraints $\mathcal{U}$, which is possibly empty. We aim at computing the Pareto front set for this constrained clustering task with the bi-criterion $(\max S, \min D)$. One approach to achieve this, is given in Algorithm 15; it is comparable to the *ϵ-constraint approach* as presented in [T'kindt & Billaut 2006]. In this algorithm, optimizations with a single criterion are iterated, each time with a condition on the value of the other criterion. The function *Maximize_ Split*($\mathcal{U}$) or *Minimize_ Diameter*($\mathcal{U}$) means the use of our model with the criterion of maximizing the split or minimizing the diameter, respectively, and with the set of user constraints $\mathcal{U}$. It returns an optimal solution which satisfies all the constraints in $\mathcal{U}$, if there exists one, or $NULL$ otherwise. The idea of the algorithm is presented in Figure 7.5. The first Pareto solution found is the solution with the best diameter whereas the last Pareto solution found is the solution with the best split. We prove that this algorithm finds the complete Pareto front set.

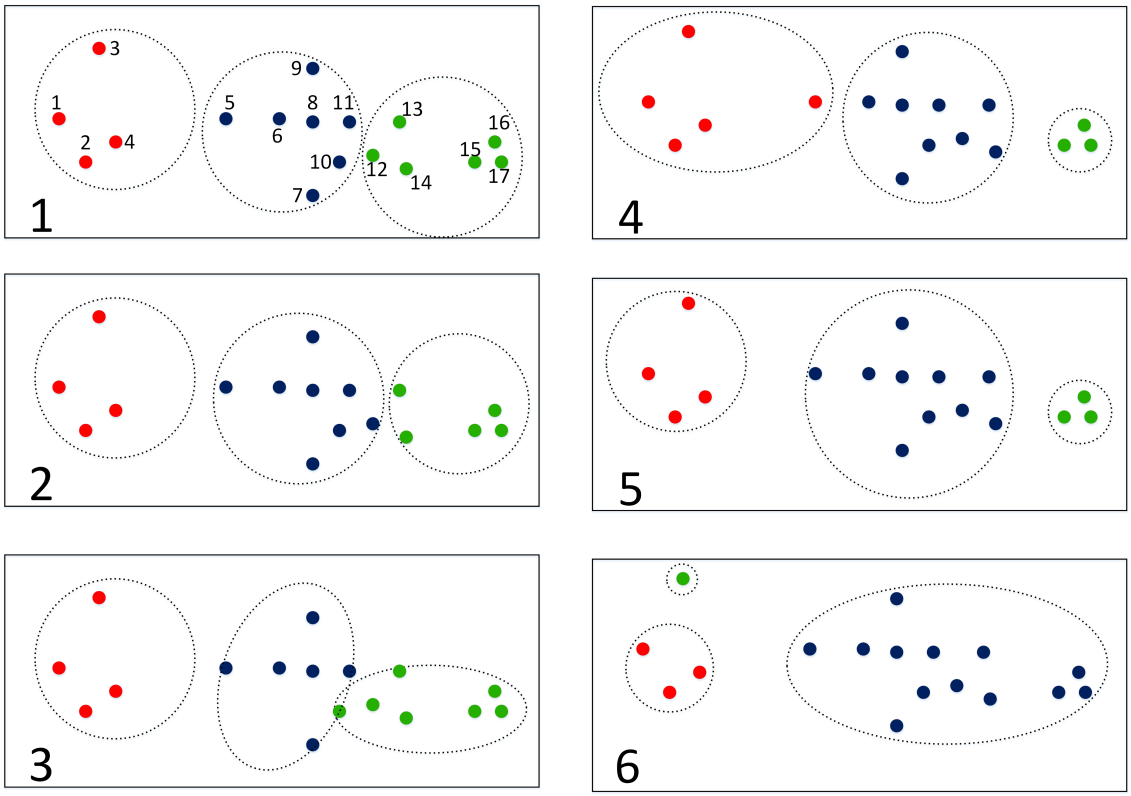

Figure 7.4: Pareto optimal solutions

Algorithm 15: Algorithm computing the set $\mathcal{A}$ of all the Pareto optima

$\mathcal{A} \leftarrow \emptyset$;
$i \leftarrow 1$;
$\Delta_i^D \leftarrow$ Minimize_Diameter($\mathcal{U}$);
while $\Delta_i^D \neq NULL$ **do**
  $\Delta_i^S \leftarrow$ Maximize_Split($\mathcal{U} \cup \{D \leq D(\Delta_i^D)\}$) ;
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{\Delta_i^S\}$;
  $i \leftarrow i + 1$;
  $\Delta_i^D \leftarrow$ Minimize_Diameter($\mathcal{U} \cup \{S > S(\Delta_{i-1}^S)\}$) ;
end

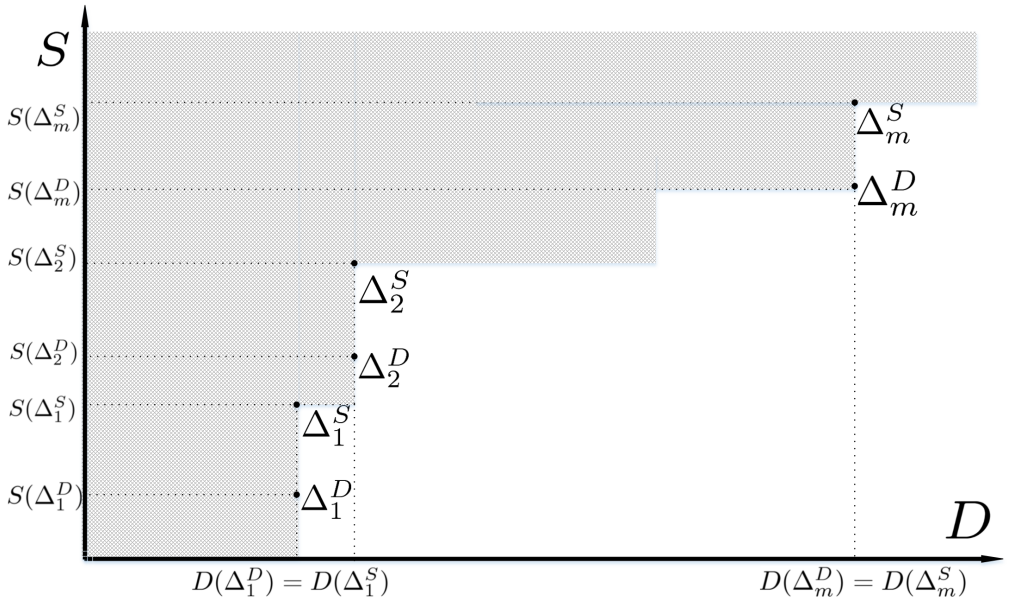

Figure 7.5: The solutions found by Algorithm 15

Proposition 28 *Let $\Delta_1^D, \Delta_1^S, \ldots, \Delta_m^D, \Delta_m^S$ be the solutions visited by Algorithm 15. We have:*

1. *if $\Delta_i^D \neq NULL$ then $\Delta_i^S \neq NULL$,*
2. *for all $1 < i \leq m$, $S(\Delta_i^S) > S(\Delta_{i-1}^S)$,*
3. *for all $1 < i \leq m$, $D(\Delta_i^D) > D(\Delta_{i-1}^D)$,*
4. *for all $1 \leq i \leq m$, $D(\Delta_i^S) = D(\Delta_i^D)$,*
5. *$\nexists \Delta$ such that $D(\Delta) < D(\Delta_1^D)$,*
6. *for all $1 \leq i < m$, $\nexists \Delta$ such that $S(\Delta) \geq S(\Delta_i^S)$ and $D(\Delta) < D(\Delta_{i+1}^D)$.*

Proof

1. If $\Delta_i^D \neq NULL$, since Δ_i^D satisfies the constraint $D \leq D(\Delta_i^D)$, it is a candidate for Δ_i^S.

2. Since among all the partitions satisfying $D \leq D(\Delta_i^D)$, Δ_i^S is the one which maximizes the split (line 5), we have $S(\Delta_i^S) \geq S(\Delta_i^D)$. Since Δ_i^D is among the partitions satisfying $S > S(\Delta_{i-1}^S)$ (line 8), we have $S(\Delta_i^D) > S(\Delta_{i-1}^S)$. Therefore $S(\Delta_i^S) > S(\Delta_{i-1}^S)$.

3. The set of the partitions satisfying $S > S(\Delta_i^S)$ is a strict subset of the set of the partitions satisfying $S > S(\Delta_{i-1}^S)$, since $S(\Delta_i^S) > S(\Delta_{i-1}^S)$. Since Δ_i^D and Δ_{i-1}^D are the partitions which minimize the diameter among all the ones in these two respective sets, we have $D(\Delta_i^D) \geq D(\Delta_{i-1}^D)$. If $D(\Delta_i^D) = D(\Delta_{i-1}^D)$, then $S(\Delta_i^S) = S(\Delta_{i+1}^S)$ (line 5) which contradicts $S(\Delta_i^S) > S(\Delta_{i-1}^S)$. Therefore $D(\Delta_i^D) > D(\Delta_{i-1}^D)$.

4. Since Δ_i^S is a partition satisfying the condition $D \leq D(\Delta_i^D)$, we have $D(\Delta_i^S) \leq D(\Delta_i^D)$. Since $S(\Delta_i^S) > S(\Delta_{i-1}^S)$ and Δ_i^D is a partition which minimizes the diameter among all the ones satisfying the condition $S > S(\Delta_{i-1}^S)$, we have $D(\Delta_i^S) \geq D(\Delta_i^D)$. Therefore $D(\Delta_i^S) = D(\Delta_i^D)$,

5. Since Δ_1^D is a partition which minimizes the diameter under no condition (line 3), there exists no partition Δ with $D(\Delta) < D(\Delta_1^D)$.

6. Assume that there exists a solution Δ such that $S(\Delta) \geq S(\Delta_i^S)$ and $D(\Delta) < D(\Delta_{i+1}^D)$. Since $S(\Delta) \geq S(\Delta_i^S)$ and Δ_{i+1}^D is a partition which minimizes the diameter among all those which satisfy the condition $S > S(\Delta_i^S)$, we have $D(\Delta) \geq D(\Delta_{i+1}^D)$. This contradicts the fact that $D(\Delta) < D(\Delta_{i+1}^D)$. □

By Proposition 28, the position of the solutions found by Algorithm 15 is as presented in Figure 7.5. There exists no solution in the grey zone. A solution in the white zone is dominated by a solution $\Delta_i^S \in \mathcal{A}$.

Proposition 29 *Given the set $\mathcal{A} = \{\Delta_1^S, \ldots, \Delta_m^S\}$ computed by Algorithm 15, the Pareto front set $\mathcal{B} = \{(D(\Delta_1^S), S(\Delta_1^S)), \ldots, (D(\Delta_m^S), S(\Delta_m^S))\}$ is complete, i.e.:*

1. *Δ_i^S $(1 \leq i \leq m)$ is a Pareto optimal solution,*

2. *$\nexists \Delta$ Pareto optimal solution such that $(D(\Delta), S(\Delta)) \notin \mathcal{B}$*

Proof Algorithm 15 terminates since according to Proposition 28, $S(\Delta_i^S) > S(\Delta_{i-1}^S)$ and these values are discrete and limited by the maximal dissimilarity of pairs of points.

1. We prove that for all $\in [1, m]$, there exists no partition Δ which dominates Δ_i^S, *i.e.* $D(\Delta) \leq D(\Delta_i^S)$ et $S(\Delta) > S(\Delta_i^S)$, or $D(\Delta) < D(\Delta_i^S)$ et $S(\Delta) \geq S(\Delta_i^S)$. Since $D(\Delta_i^S) = D(\Delta_i^D)$, the partition Δ must satisfy $D(\Delta) \leq D(\Delta_i^D)$ et $S(\Delta) > S(\Delta_i^S)$, or $D(\Delta) < D(\Delta_i^D)$ et $S(\Delta) \geq S(\Delta_i^S)$. The first case is impossible since Δ_i^S is a partition which maximizes the split among all those which satisfy the condition $D \leq D(\Delta_i^D)$. For the second case, if $i = 1$ then Δ does not exist according to the point 5 of Proposition 28. If $i > 1$, since $S(\Delta_i^S) > S(\Delta_{i-1}^S)$, the partition Δ must satisfy $D(\Delta) < D(\Delta_i^D)$ and $S(\Delta) > S(\Delta_{i-1}^S)$. This is impossible according to the point 6 of Proposition 28.

2. Let us assume that there exists a Pareto optimal solution such that $(D(\Delta), S(\Delta)) \in \mathcal{B}$. According to the points 5 and 6 of Proposition 28, Δ to be not dominated must have $S(\Delta) > S(\Delta_m^S)$. But Algorithme 15 terminates after Δ_m^S, *i.e.* there exists no partition with the split greater than $S(\Delta_m^S)$. Therefore, a Pareto optimal solution $\Delta \notin \mathcal{A}$ does not exist.

Multi-objective optimization in Constraint Programming in only one phase of search is proposed in [Gavanelli 2002]. The idea is to realize a global constraint $Pareto(Obj_1, \ldots, Obj_m, \mathcal{A})$, which keeps a set of non dominated solutions so far computed $\mathcal{A}$ and which operates on the variables representing the objective functions Obj_i. This constraint reduces the domain of a variable Obj_i if the domains of the other variables enter into the dominated zone of a solution in $\mathcal{A}$. A detailed description of this constraint

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
WDBC	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 7.1: Comparison of performance with bi-criteria

as well as an extension with Large Neighborhood Search is proposed in [**?**]. This constraint *Pareto* can be introduced in our model. Nevertheless this approach for the moment is still much less efficient than Algorithm 15 and therefore needs deeper studies, for instance of the search strategy, in order to improve the efficiency.

7.4 Experimentation

7.4.1 Performance test

For the bi-criterion split-diameter, we compare our second model presented in Chapter 6 (denoted by CP2) with the bi-criterion clustering algorithm based on graph coloring [Delattre & Hansen 1980] (denoted bGC). The program is not available, we have coded it in C++. To our knowledge, this is the only exact algorithm for general values of k. In the experiments, datasets presented in Table 5.1 are used and the time-out is set to 1 hour. The number of clusters k varies between 2 and the real number of classes. Table 7.1 shows the result of the experiments. The second column (#Sol) gives the number of Pareto optimal solutions found and the next columns give the run time in seconds of each approach. Note that both approaches find a complete Pareto front set, so the number of Pareto optimal solutions found is the same as it is the cardinality of the complete Pareto front set. However, the partitions found by the two approaches might be different as there exists different partitions with the same values of diameter and split. It can be seen that our model is the most efficient in most cases. It takes advantage of the efficient constraint propagation to reduce the search space. As in the case of GC, the algorithm bGC is limited to datasets with less than 500 points.

7.4.2 Bi-criterion clustering with user-constraints

We consider the instance-level constraints: must-link and cannot-link constraints. The constraints were generated with the same method as in the previous Chapter: two points

are chosen randomly from the dataset, if they are in the same cluster, a must-link constraint is generated, otherwise a cannot-link constraint. We generate a set of 80 instance-level constraints from the dataset Iris and use our second model for the task of bi-criterion constrained clustering. The first test is without user-constraints, the second one is with the first 20 instance-level constraints, the third one is with the first 40 instance-level constraints and so on. Figure 7.6 presents the results of the Pareto front in six cases, where there are from 0 to 80 instance-level constraints. As there are more and more user-constraints, it is clear that there are less feasible solutions and as a result, the criterion space is reduced. Because we generated user-constraints from real labels, it is obvious that the values (diameter,split) of the real partition of the dataset lies in the smallest criterion space (the region defined by red lines in Figure 7.6). There are significantly difference of the location of Pareto front between different cases in Figure 7.6. It shows that without user-constraints, even there are many points in the Pareto front, they may all lie far from the real partition in the criterion space. For that reason, it could be useful to enable user-constraints for the task of bi-criterion clustering.

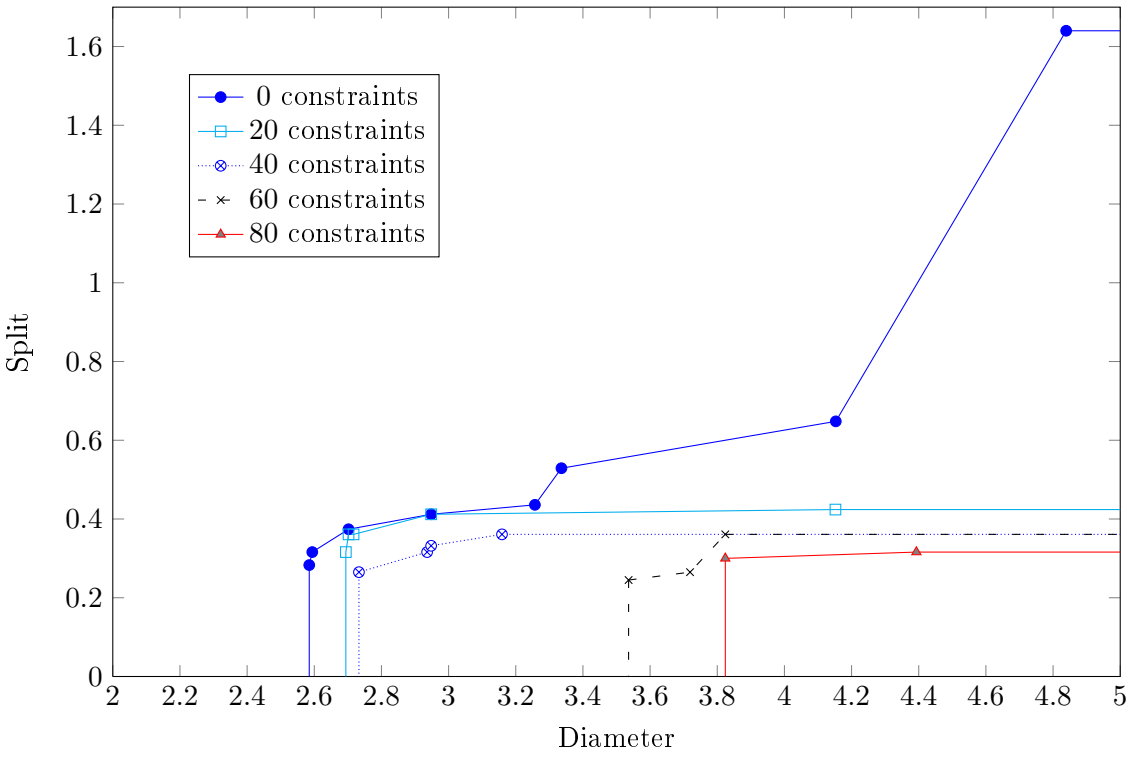

Figure 7.6: Bi-criterion constrained clustering with dataset Iris

7.5 Summary

In this chapter we explore the problem of constrained clustering with the multi-objective optimization. Typically, for solving a multi-objective optimization problem, a common way is to find the set of Pareto optimal solutions. We proposed an efficient algorithm, based on the ϵ-*constraint approach* [T'kindt & Billaut 2006], using our CP model for finding the

complete Pareto front set, w.r.t the two criteria of split and diameter. Experiments on various UCI benchmark datasets show that our model is more effective than the bi-criterion clustering algorithm based on graph coloring [Delattre & Hansen 1980]. More than that, to the best of our knowledge, we are the first to integrate user-constraints to the problem of bi-criterion clustering.

Chapter 8

Conclusion and Future Work

Dans ce chapitre, nous rappelons nos contributions et discutons quelques perspectives. Les deux premières contributions sont deux modèles basés sur la Programmation par Contraintes pour le clustering sous contraintes utilisateur. Les modèles sont généraux et flexibles, ils permettent d'intégrer des contraintes d'instances *must-link* et *cannot-link* et différents types de contraintes sur les clusters. Ils offrent également à l'utilisateur le choix entre différents critères d'optimisation. Afin d'améliorer l'efficacité, divers aspects sont étudiés. Les expérimentations sur des bases de données classiques et variées montrent qu'ils sont compétitifs par rapport aux approches exactes existantes. La dernière contribution de la thèse est de montrer que nos modèles peuvent être intégrés dans une procédure plus générale et nous l'illustrons par la recherche de la frontière de Pareto dans un problème de clustering bi-critères sous contraintes utilisateur.

A notre connaissance, notre approche est la première pour les problèmes de classification non supervisée sous contraintes utilisateur, qui intègre le critère de diamètre ou de marge ou de la somme des dissimilarités intra-cluster et du bi-critère marge-diamètre, en présence de contraintes utilisateur.

Contents

This chapter concludes the dissertation and provides perspectives for future work.

8.1 Contributions of this Dissertation

Cluster analysis is an important task in Data Mining with hundreds of different approaches in the literature. Since the last decade, cluster analysis has been extended to constrained clustering, or semi-supervised clustering, so as to integrate previous knowledge on data to clustering algorithms. In this dissertation we define two models based on Constraint Programming (CP) to address constrained clustering tasks. Relying on CP has two main advantages: the declarativity, which enables to easily add new constraints and the ability to find an optimal solution satisfying all the constraints (when there exists one). However, modelling a problem in CP in an efficient and effective way requires a lot of skills and efforts.

The first model is based on a two-level representation: a set of variables for the assignment of a representative for each class and a set of variables for the assignment of

a representative for each point. In literature, most constrained clustering algorithms are limited to few kinds of user-constraints. Our model, based on CP, is more general in the sense that it supports instance-level constraints, specifying requirements on pairs of objects, as well as different cluster-level constraints, specifying requirements on clusters. It also allows user to choose among different optimization criteria. In order to improve the efficiency, different aspects have been studied in the dissertation. We define a filtering algorithm for the criterion of within cluster sum of dissimilarities in our model. Experiments on various UCI benchmark datasets show that our model is competitive with other exact approaches.

The second model is based only on a set of variables for the assignment of a cluster, represented by its number, to each point. The model is more flexible as it does not require to fix a priori the number of clusters k, but only bounds on k are required. The second model is lighter in terms of the number of variables, as well as the number of constraints. We define two new global constraints for the criteria of split and diameter in order to replace a large number of reified constraints. Experiments show that the second model is more effective and can handle larger datasets.

Another important contribution is to apply the model to the task of constrained bi-criterion clustering. We show that our model can easily be embedded into a more general process and we illustrate this on the problem of finding the Pareto front of a bi-criterion optimization process. To the best of our knowledge, we are the first to:

- Find a global optimum for the problem of Constrained Clustering with the criteria of diameter, split and within cluster sum of dissimilarities.
- Solve the bi-criterion (split-diameter) clustering with user constraints.

8.2 Directions for Future Work

Directions for future work include the following:

- Firstly, we would like to study different heuristics for the search strategy in order to improve the performance of our models. The goal of the search strategy is not only to quickly find an optimal solution, but to quickly prove the optimality. During the search process, the choices on variables and values for branching are important, as the correct choices may help the constraint propagation to reduce efficiently the search space. A good search strategy may not only depend on the criterion, but also depend on the type of datasets. As a consequence, it is worth to take time and effort to study and to experiment with different techniques.
- The goal of our models is to find an optimal solution for a given criterion. In practice, it is not always true that the optimal solution corresponds to a good partition whereas finding and proving an optimal solution requires exhaustive search. We would like to study and apply heuristics search techniques, to balance between the computation time and the quality of the solution. For instance, heuristics based on Large Neighbourhood Search [Shaw 1998] have recently shown tremendous results

in solving various problems in CP. We would like to study and apply Large Neighbourhood Search for solving large-scale datasets, in which it is impossible to find an optimal solution.

- Relating to the multi-objective clustering, our approach presented in Chapter 7 requires to restart the model multiple times in order to get the complete Pareto front. Multi-objective optimization in Constraint Programming in only one phase of search is proposed in [Gavanelli 2002]. The idea is to realize a global *Pareto* constraint, which keeps a set of non dominated solutions and operates on the variables representing the objective functions. A detailed description of this constraint as well as an extension with Large Neighbourhood Search is proposed in [**?**]. This *Pareto* constraint can be introduced in our model. Nevertheless this approach for the moment is still limited and much less efficient than the approach presented in Chapter 7. It therefore needs deeper studies, for instance of the filtering algorithm or of the search strategy, in order to improve the efficiency.

- Our models find an optimal solution that satisfies all given user-constraints. However, in practice, there may exist noise in the set of user-constraints, or the problem is over-constrained, i.e. there is no solution satisfying all constraints. We would like to integrate soft constraints to our model, i.e. constraints that are preferred but not required to be satisfied. Soft constraints allow users to set a threshold on the number of constraints that must be satisfied, while there are some constraints unsatisfied. One solution is to study other modelings for the problem, for instance to design new dedicated filtering algorithms for soft constraints. Another possible technique that we would like to study is applying a schema that allows hard constraints for representing soft constraints [Régin 2011].

- Our models support the criteria of diameter, split and within cluster sum of dissimilarities. We would like to study and integrate other well-known criteria to the model, for instance, the within-cluster sum of squares. Adding a new criterion in our model is simple, by adding new constraints to express the function objective. There are different ways to model a given criterion in CP. The challenge is how to model a criterion with high performance. The propagation of criterion constraints is therefore highly important. We need filtering algorithms that have low computational complexity but are able to compute good bounds on the objective function and to filter as much as possible inconsistent values in the domain of variables. We would like to study and to apply techniques in linear programming, for instance column generation techniques, for designing effective filtering algorithms for other criteria.

- Finally, we would like to extend our research to address other clustering tasks, for instance fuzzy clustering where points can belong to more than one cluster.

Bibliography

[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 2003] Krzysztof Apt. Principles of constraint programming. Cambridge University Press, New York, NY, USA, 2003.

[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 2013] K. Bache and M. Lichman. *UCI Machine Learning Repository*, 2013.

[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.

[Beldiceanu *et al.* 2012] Nicolas Beldiceanu, Mats Carlsson, Thierry Petit and Jean-Charles Régin. *An O(nlog n) Bound Consistency Algorithm for the Conjunction of an alldifferent and an Inequality between a Sum of Variables and a Constant, and its Generalization.* In Proceedings of the 20th European Conference on Artificial Intelligence, pages 145–150, 2012.

[Berg & Jarvisalo 2013] J. Berg and M. Jarvisalo. *Optimal Correlation Clustering via MaxSAT.* In Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on, pages 750–757, Dec 2013.

[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.

[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.

[Carbonneau *et al.* 2012] Réal A. Carbonneau, Gilles Caporossi and Pierre Hansen. *Extensions to the Repetitive Branch and Bound Algorithm for Globally Optimal Clusterwise Regression.* Computer & Operations Research, vol. 39, no. 11, pages 2748–2762, 2012.

[choco Team 2010] choco Team. *choco: an Open Source Java Constraint Programming Library.* Research report 10-02-INFO, École des Mines de Nantes, 2010.

[Chung 1997] F. R. K. Chung. Spectral graph theory. American Mathematical Society, 1997.

[Cohen *et al.* 2005] David Cohen, Peter Jeavons, Christopher Jefferson, KarenE. Petrie and BarbaraM. Smith. *Symmetry Definitions for Constraint Satisfaction Problems.* In Peter van Beek, editeur, Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, volume 3709 of *Lecture Notes in Computer Science*, pages 17–31. 2005.

[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.

[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 PPC pour la classification non supervisée.* In 13e Conférence Francophone sur l'Extraction et la Gestion des Connaissances EGC, 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. *Clustering sous contraintes en PPC.* Revue d'Intelligence Artificielle, 2014. To appear.

[Dao *et al.* 2014c] Thi-Bich-Hanh Dao, Khanh-Chuong Duong and Christel Vrain. *Constrained Clustering by Constraint Programming.* Artificial Intelligence, 2014. To appear with a minor revision.

[Davidson & Basu 2007] Ian Davidson and Sugato Basu. *A survey of clustering with instance level constraints.* ACM Transactions on Knowledge Discovery from Data, pages 1–41, 2007.

[Davidson & Ravi 2005a] I. Davidson and S. S. Ravi. *Agglomerative hierarchical clustering with constraints: Theoretical and empirical results.* In Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, pages 59–70, 2005.

[Davidson & Ravi 2005b] 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.* 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.

[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.

[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.* 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, page 201, 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.

[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.

[Gavanelli 2002] Marco Gavanelli. *An Algorithm for Multi-Criteria Optimization in CSPs.* In Frank van Harmelen, editeur, Proceedings of the 15th European Conference on Artificial Intelligence, pages 136–140, 2002.

[Gecode Team] Gecode Team.

[Gent *et al.* 2006] IanP. Gent, Chris Jefferson and Ian Miguel. *Watched Literals for Constraint Propagation in Minion.* In Frédéric Benhamou, editeur, Principles and Practice of Constraint Programming - CP 2006, volume 4204 of *Lecture Notes in Computer Science*, pages 182–197. Springer Berlin Heidelberg, 2006.

[Goffin *et al.* 1992] J. L. Goffin, A. Haurie and J. P. Vial. *Decomposition and Nondifferentiable Optimization with the Projective Algorithm.* Management Science, vol. 38, no. 2, pages 284–302, 1992.

[Gonzalez 1985] T. Gonzalez. *Clustering to minimize the maximum intercluster distance.* Theoretical Computer Science, vol. 38, pages 293–306, 1985.

[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.

[Hagen & Kahng 1992] L. Hagen and A.B. Kahng. *New spectral methods for ratio cut partitioning and clustering.* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, vol. 11, no. 9, pages 1074–1085, Sep 1992.

[Han *et al.* 2011] Jiawei Han, Micheline Kamber and Jian Pei. Data mining: Concepts and techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd édition, 2011.

[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 1979] Robert M. Haralick and Gordon L. Elliott. *Increasing Tree Search Efficiency for Constraint Satisfaction Problems.* In Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI'79, pages 356–364, San Francisco, CA, USA, 1979. Morgan Kaufmann Publishers Inc.

[Hubert & Arabie 1985] L. Hubert and P. Arabie. *Comparing partitions.* Journal of classification, vol. 2, no. 1, pages 193–218, 1985.

[Johnson 1967] StephenC. Johnson. *Hierarchical clustering schemes.* Psychometrika, vol. 32, no. 3, pages 241–254, 1967.

[Kaufman & Rousseeuw 1987] L. Kaufman and P. Rousseeuw. Clustering by means of medoids. Reports of the Faculty of Mathematics and Informatics. Faculty of Mathematics and Informatics, 1987.

[Khiari *et al.* 2010] Mehdi Khiari, Patrice Boizumault and Bruno Crémilleux. *Constraint Programming for Mining n-ary Patterns.* In David Cohen, editeur, Principles and Practice of Constraint Programming – CP 2010, volume 6308 of *Lecture Notes in Computer Science*, pages 552–567. Springer Berlin Heidelberg, 2010.

[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.

[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.

[Laborie 2009] Philippe Laborie. *IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems.* In Willem-Jan van Hoeve and JohnN. Hooker, editeurs, Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, volume 5547 of *Lecture Notes in Computer Science*, pages 148–162. Springer Berlin Heidelberg, 2009.

[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.

[Lloyd 1994] John W Lloyd. *Practical advantages of declarative programming.* Joint Conference on Declarative Programming, GULP-PRODE, vol. 94, page 94, 1994.

[Lu & Carreira-Perpinan 2008] Zhengdong Lu and Miguel A. Carreira-Perpinan. *Constrained spectral clustering through affinity propagation.* In Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, 2008.

[Luxburg 2007] Ulrike Luxburg. *A Tutorial on Spectral Clustering.* Statistics and Computing, vol. 17, no. 4, pages 395–416, 2007.

[MacQueen 1967] J. B. MacQueen. *Some Methods for Classification and Analysis of MultiVariate Observations.* In L. M. Le Cam and J. Neyman, editeurs, Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297. University of California Press, 1967.

[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.

[Metivier *et al.* 2011] J.-P. Metivier, P. Boizumault, B. Cremilleux, M. Khiari and S. Loudni. *A Constraint-Based Language for Declarative Pattern Discovery.* In Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on, pages 1112–1119, Dec 2011.

[Métivier *et al.* 2012] 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.

[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.

[Pareto 1896] V. Pareto. Cours d'economie politique. F. Rouge, Lausanne, Switzerland, 1896.

[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.

[Petit *et al.* 2011] Thierry Petit, Jean-Charles Régin and Nicolas Beldiceanu. *A $\theta(n)$ Bound-Consistency Algorithm for the Increasing Sum Constraint.* In Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, pages 721–728, 2011.

[Raedt *et al.* 2011] Luc De Raedt, Siegfried Nijssen, Barry O'Sullivan and Pascal Van Hentenryck. *Constraint Programming meets Machine Learning and Data Mining (Dagstuhl Seminar 11201).* Dagstuhl Reports, vol. 1, no. 5, pages 61–83, 2011.

[Rao 1969] R. Rao. Cluster analysis and mathematical programming. Management sciences research report. U.S. Clearinghouse, 1969.

[Régin & Petit 2011] Jean-Charles Régin and Thierry Petit. *The Objective Sum Constraint.* In Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pages 190–195, 2011.

[Régin & Rueher 2005] Jean-Charles Régin and Michel Rueher. *Inequality-sum: a global constraint capturing the objective function.* RAIRO - Operations Research, vol. 39, no. 2, pages 123–139, 2005.

[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 2011] Jean-Charles Régin. *Using Hard Constraints for Representing Soft Constraints.* In Proceedings of the 8th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, pages 176–189, 2011.

[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.

[Shaw 1998] Paul Shaw. *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems.* In Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming, pages 417–431, 1998.

[Shi & Malik 2000] Jianbo Shi and Jitendra Malik. *Normalized Cuts and Image Segmentation.* IEEE Trans. Pattern Anal. Mach. Intell., vol. 22, no. 8, pages 888–905, August 2000.

[Stoer & Wagner 1997] Mechthild Stoer and Frank Wagner. *A Simple Min-cut Algorithm.* J. ACM, vol. 44, no. 4, pages 585–591, July 1997.

[T'kindt & Billaut 2006] Vincent T'kindt and Jean-Charles Billaut. Multicriteria scheduling, theory, models and algorithms. Springer, 2 édition, 2006.

[Van Hentenryck & Michel 2005] Pascal Van Hentenryck and Laurent Michel. Constraint-based local search. The MIT Press, 2005.

[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.

[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.

Khanh-Chuong DUONG

Classification non supervisée sous contrainte utilisateurs par la programmation par contraintes

Résumé : La classification non supervisée, souvent appelée par le terme anglais de *clustering*, est une tâche importante en Fouille de Données. Depuis une dizaine d'années, la classification non supervisée a été étendue pour intégrer des contraintes utilisateur permettant de modéliser des connaissances préalables dans le processus de clustering. Différents types de contraintes utilisateur peuvent être considérés, des contraintes pouvant porter soit sur les clusters, soit sur les instances.

Dans cette thèse, nous étudions le cadre de la Programmation par Contraintes (PPC) pour modéliser les tâches de clustering sous contraintes utilisateur. Utiliser la PPC a deux avantages principaux : la déclarativité, qui permet d'intégrer aisément des contraintes utilisateur et la capacité de trouver une solution optimale qui satisfait toutes les contraintes (s'il en existe).

Nous proposons deux modèles basés sur la PPC pour le clustering sous contraintes utilisateur. Les modèles sont généraux et flexibles, ils permettent d'intégrer des contraintes d'instances *must-link* et *cannot-link* et différents types de contraintes sur les clusters. Ils offrent également à l'utilisateur le choix entre différents critères d'optimisation. Afin d'améliorer l'efficacité, divers aspects sont étudiés. Les expérimentations sur des bases de données classiques et variées montrent qu'ils sont compétitifs par rapport aux approches exactes existantes. Nous montrons que nos modèles peuvent être intégrés dans une procédure plus générale et nous l'illustrons par la recherche de la frontière de Pareto dans un problème de clustering bi-critère sous contraintes utilisateur.

Mots clés : classification non supervisée, contraintes utilisateur, Programmation par Contraintes, clustering bi-critère.

Constrained Clustering by Constraint Programming

Abstract: Cluster analysis is an important task in Data Mining with hundreds of different approaches in the literature. Since the last decade, the cluster analysis has been extended to constrained clustering, also called semi-supervised clustering, so as to integrate previous knowledge on data to clustering algorithms.

In this dissertation, we explore Constraint Programming (CP) for solving the task of constrained clustering. The main principles in CP are: (1) users specify declaratively the problem in a Constraint Satisfaction Problem; (2) solvers search for solutions by constraint propagation and search. Relying on CP has two main advantages: the declarativity, which enables to easily add new constraints and the ability to find an optimal solution satisfying all the constraints (when there exists one).

We propose two models based on CP to address constrained clustering tasks. The models are flexible and general and supports instance-level constraints and different cluster-level constraints. It also allows the users to choose among different optimization criteria. In order to improve the efficiency, different aspects have been studied in the dissertation. Experiments on various classical datasets show that our models are competitive with other exact approaches. We show that our models can easily be embedded in a more general process and we illustrate this on the problem of finding the Pareto front of a bi-criterion optimization process.

Keywords: constrained clustering, bicriterion clustering, constraint programming, modelling, filtering algorithm

Laboratoire d'Informatique Fondamentale d'Orléans
Bat. 3IA, Université d'Orléans
Rue Léonard de Vinci, B.P. 6759
F-45067 ORLEANS Cedex 2

LABORATOIRE D'INFORMATIQUE LIFO FONDAMENTALE D'ORLEANS