Constrained Clustering by Constraint Programming

Thi-Bich-Hanh Dao*, Khanh-Chuong Duong, Christel Vrain

Univ. Orléans, INSA Centre Val de Loire, LIFO, EA 4022, F-45067, Orléans, France

Abstract

Constrained Clustering allows to make the clustering task either easier or more accurate by integrating user-constraints, which can be instance-level or cluster-level constraints. Few works consider the integration of different kinds of constraints, they are usually based on declarative frameworks and they are often exact methods, which either enumerate all the solutions satisfying the user-constraints, or find a global optimum when an optimization criterion is specified. In a previous work, we have proposed a model for Constrained Clustering based on a Constraint Programming framework. It is declarative, allowing a user to integrate user-constraints and to choose an optimization criterion among several ones. In this article we present a new and substantially improved model for Constrained Clustering, still based on a Constraint Programming framework. It differs from our earlier model in the way partitions are represented by means of variables and constraints. It is also more flexible since the number of clusters does not need to be fixed beforehand; only a lower and an upper bound on the number of clusters have to be provided. In order to make the model-based approach more efficient, we propose new global optimization constraints with dedicated filtering algorithms. 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 optimal Pareto front of a bi-criterion constrained clustering task. We compare our approach with existing exact approaches, based either on a branch-and-bound approach or on graph coloring on twelve datasets. Experiments show that the model outperforms exact approaches in most cases.

Keywords:
constrained clustering, bi-criterion clustering, constraint programming, modeling, filtering algorithm

1. Introduction

Constrained Clustering has received much attention this last decade. It allows to make the clustering task either easier or more accurate by integrating user-constraints. Several kinds of constraints can be considered. First, constraints may be used to limit the size or the diameter of clusters; second, they can enforce expert knowledge instances must be or cannot be in the same cluster (must-link or cannot-link constraints). Much work has focused on instance-based constraints and has adapted classical clustering methods to handle must-link or cannot-link constraints. A small number of earlier studies have considered the integration of different kinds of constraints. These studies are based on declarative frameworks and offer exact methods that either enumerate all the solutions satisfying the user constraints, or find a global optimum when an optimization criterion is given. For instance, in [1] a SAT based framework for constrained clustering has been proposed, integrating many kinds of user-constraints but limited to clustering tasks into two clusters. A framework for conceptual clustering based on Integer Linear Programming has also been proposed in [2]. In [3], we have presented a model based on Constraint Programming for constrained clustering. This model allows to choose one among different optimization criteria and to integrate various kinds of user constraints. As far as we know, the approach we propose is the only one able to handle different optimization criteria and all popular constraints, for any number of clusters. It is based on Constraint Programming (CP): in such a paradigm, a constraint

*Corresponding author

Email addresses: thi-bich-hanh.dao@univ-orleans.fr (Thi-Bich-Hanh Dao), khanh-chuong.duong@univ-orleans.fr (Khanh-Chuong Duong), christel.vrain@univ-orleans.fr (Christel Vrain)

optimization problem or a constraint satisfaction problem is modeled by defining variables with their domains and by expressing constraints on these variables. Solving a CP problem relies on two operations: constraint propagation that reduces the domain of the variables by removing inconsistent values and branching that divides the problem in subproblems, by taking an unassigned variable and by splitting its domain into several parts. It is important to notice that modeling a task in Constraint Programming implies several choices, which have a high impact on the efficiency of the approach: the choice of the variables and the choice of the constraints for the model, the development of filtering algorithms dedicated to the task and the use of adapted search strategies for solving the model. A point in favor of CP is that the requirement of getting an exact solution can be relaxed by using metaheuristics or local search methods. For the time being, we have fully investigated exact methods, to push the efficiency of the framework as far as possible. Approximate search strategies could be integrated in the future.

In this paper, we propose a new model for Constrained Clustering, still based on Constraint Programming, but significantly improved compared to the previous model [3]. In the previous model, 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 expressing its assignment 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 index of the cluster the point belongs to. As a result, the constraints enforcing the solution to be a partition and breaking symmetries are entirely different. The new model is 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. Moreover, in order to make this model efficient, we have developed dedicated global constraints for three optimization criteria: minimizing the maximal diameter, maximizing the split between clusters, and minimizing the within-cluster sum of dissimilarities.

The approach we propose may be easily embedded in a general process for the task of Constrained Clustering. Considering Data Mining as an iterative and interactive process composed of the classical steps of task formulation, data preparation, application of a tool, thus requiring to set parameters, and validation of the results, a user can specify the task at hand including or not some constraints and decide to change the settings according to the results. He/she may decide to change the constraints, removing or relaxing some constraints, adding or hardening other constraints. The modularity and declarativity of our model allow this easily. In this paper, we illustrate the integration of our model in a more complex process by considering a bi-criterion clustering problem, namely finding the Pareto front when minimizing the maximal diameter and maximizing the minimal split. To achieve this, our framework is integrated in an algorithm, which alternatively calls our model to minimize the maximal diameter and then to maximize the split between clusters with adapted constraints.

Our contributions are as follows.

- We propose a new model based on Constraint Programming, allowing to find an optimal solution for clustering under constraints, given an optimization criterion. This new model improves substantially the previous one, it is more modular (each criterion is implemented by a global constraint) and it is much more efficient.
- 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 optimal Pareto front of a bi-criterion constrained clustering task. As far as we know, this is the first approach to handle bi-criterion clustering in presence of user-constraints.
- We propose new global optimization constraints with dedicated filtering algorithms, thus allowing to make the model more efficient.
- We compare this model with existing exact approaches, based either on a branch-and-bound approach [4] or on graph coloring [5] on twelve datasets. Experiments show that the model we propose is generally more efficient. Moreover we compare the two models based on CP that we have developed and we show that the different changes (search strategy and development of global constraints) allow to improve the model.

The paper is organized as follows. Section 2 is dedicated to preliminaries on Constrained Clustering and Constraint Programming. Related work is presented in Section 3. Section 4 is devoted to the presentation of both CP models, the first one presented in [3] and the new one. The filtering algorithms for the optimization criteria are presented in Section 5. We show in Section 6 how our framework can be easily integrated for solving a bi-criterion constrained clustering task. Experiments are presented in Section 7, showing the performance and the flexibility of our model.

2. Preliminaries

2.1. Constrained Clustering

Cluster analysis is a Data Mining task that aims at partitioning a given set of objects into homogeneous and/or well-separated subsets, called classes or clusters. It is often formulated as the search for a partition such that the objects inside the same cluster are similar, while being different from the objects belonging to other clusters. These requirements are usually expressed by an optimization criterion and the clustering task is usually defined as finding a partition of objects that optimizes the given criterion. In the remainder of the paper, we consider a dataset of n objects $O = \{o_1, \ldots, o_n\}$ and a dissimilarity measure $d(o_i, o_j)$ between any two objects $o_i, o_j \in O$. A partition Δ of objects into k classes $C_1, \ldots, C_k$ is such that: (1) for all $c \in [1, k]$[1], $C_c \neq \emptyset$, (2) $\cup_c C_c = O$ and (3) for all $c \neq c'$, $C_c \cap C_{c'} = \emptyset$. The optimization criterion can be among others:

- Minimizing the maximal diameter of clusters: the *maximal diameter* of a partition Δ is the largest dissimilarity between two objects in the same cluster,

$$D(\Delta) = \max_{c \in [1,k], o_i, o_j \in C_c,} (d(o_i, o_j)).$$

  A clustering task that minimizes this criterion is also called nonhierarchical complete-link clustering.

- Maximizing the minimal split between clusters: the *minimal split* between clusters of a partition Δ is the smallest dissimilarity between two objects of different clusters,

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

  A clustering task that maximizes this criterion is also called single-link clustering.

- Minimizing the within-cluster sum of dissimilarities (WCSD): this sum for a partition Δ is defined as

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

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

- Minimizing the within-cluster sum of squares (WCSS): in a Euclidean space the within-cluster sum of squares is the sum of the squared Euclidean distances between each object o_i and the centroid m_c of the cluster containing o_i

$$WCSS(\Delta) = \sum_{c \in [1,k]} \sum_{o_i \in C_c} \|o_i - m_c\|^2.$$

  Let us notice that, when the squared Euclidean distance is used for measuring the dissimilarities, the WCSS criterion is mathematically equivalent to the WCSD criterion standardized by the division by the size of each cluster:

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

Most of the clustering algorithms rely on an optimization criterion. All of these criteria are NP-Hard, except the split criterion. In consequence, most of the algorithms search for a local optimum. For instance, the k-means algorithm finds a local optimum for the WCSS criterion as well as FPF (Furthest Point First) [6] for the diameter criterion. Several optima may exist, some may be closer to the one expected by the user. In order to model the task better, 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 the user constraints. User constraints can be classified into cluster-level constraints,

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

specifying requirements on clusters, or instance-level constraints, specifying requirements on pairs of objects. Most of the attention has been put on instance-level constraints, first introduced in [7]. Commonly, two kinds of constraints are used: must-link and cannot-link. A must-link constraint on two objects o_i and o_j expresses that they must be in the same cluster: $\forall c \in [1,k],\ o_i \in C_c \Leftrightarrow o_j \in C_c$. A cannot-link constraint on two objects o_i and o_j expresses that they must not be in the same cluster: $\forall c \in [1,k],\ \neg(o_i \in C_c \wedge o_j \in C_c)$.

Cluster-level constraints impose requirements on the clusters. The minimum capacity constraint requires that each cluster has a number of objects greater than a given threshold α: $\forall c \in [1,k],\ |C_c| \geq \alpha$, whereas 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 maximum diameter constraint specifies an upper bound γ on the diameter of the clusters: $\forall c \in [1,k], \forall o_i, o_j \in C_c,\ d(o_i,o_j) \leq \gamma$. The minimum split constraint, also called the δ-constraint in [8], requires the distance between any two points of different clusters to be superior to a given threshold δ: $\forall c \in [1,k], \forall c' \neq c, \forall o_i \in C_c, \forall o_j \in C_{c'}, d(o_i,o_j) \geq \delta$. As observed in [8], the maximum diameter constraint can be represented by a conjunction of cannot-link constraints and the minimum split constraint can be represented by a conjunction of must-link constraints.

The ϵ-constraint introduced in [8] 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 $d(o_i,o_j) \leq \epsilon$. This constraint tries to capture the notion of density, introduced in DBSCAN [9]. 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.

In the last ten years, many works have been done to extend classical algorithms for handling must-link and cannot-link constraints. This is achieved either by modifying the dissimilarity measure, or the objective function or the search strategy. Recently, several works have investigated declarative approaches of constrained clustering, which aim at extending traditional algorithms to different types of user-constraints. A presentation of these works is given in Section 3.

2.2. 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 [10], *i.e.* quite similar objects may be classified in different clusters, in order to keep the diameters small. On the other hand, clustering with the criterion of maximizing the minimal split, which aims at finding well separated clusters, often suffers from the chain effect [11], *i.e.* a chain of closed objects may lead to group very different objects in the same cluster. The popular WCSS criterion, which minimizes the sum of the squared distances between points and the center of their cluster also suffers from undesirable effects. Considering this criterion, objects that should be in a large group may be classified in different clusters in order to keep this sum small. Figure 1 gives an illustration of these effects. Image A shows three groups that can be easily identified. Image B shows the obtained solution with the diameter criterion when the number of clusters is set to 3. In this partition, some points are very close but they are classified in two different groups. The partition obtained when considering the split criterion is shown in Image C. Because of the chain effect, the largest group contains points that are very far each from other. The optimal solution with the WCSS criterion is shown in Image D. In this partition, some points that are very close are grouped in different clusters.

A good partition with homogeneous and well-separated clusters should have a minimal diameter and a maximal split. Unfortunately, such a partition in general does not exist, since the two criteria are often conflicting. This problem can be modeled by considering the bi-criterion of maximizing the minimal split between clusters and minimizing the maximal diameter, as introduced in [5]. Considering these two criteria together is natural and allows to capture both the homogeneity and the separation requirements for a good clustering. A general approach for handling two optimization criteria is to find the Pareto optimal solutions. A Pareto optimal solution is a solution such that it is not possible to improve the value of one criterion without degrading the value of the other one. A partition Δ' *dominates* a partition Δ if and only if:

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

or

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

A partition Δ is *Pareto optimal* if and only if there is no partition Δ' that dominates Δ. Two Pareto optimal solutions Δ_1 and Δ_2 are equivalent if $D(\Delta_1) = D(\Delta_2)$ and $S(\Delta_1) = S(\Delta_2)$. A set $\mathcal{P}$ of Pareto optimal solutions is complete if any

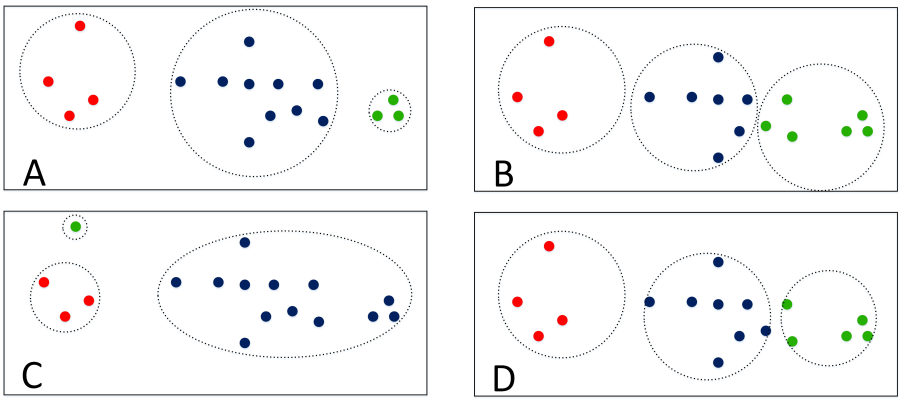

Figure 1: Effect with different criteria. A: intuitive groups; B: complete link; C: single link; D: WCSS

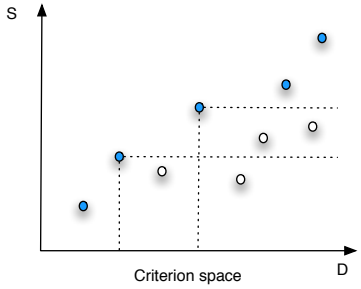

Figure 2: Pareto front

Pareto optimal solution is either in $\mathcal{P}$ or equivalent to an element of $\mathcal{P}$. The set $\mathcal{P}$ is minimal if no two partitions of $\mathcal{P}$ are equivalent. The *Pareto front* is the projection of all the Pareto optimal solutions in the criterion space, *i.e.*, the set of pairs $(D(\Delta), S(\Delta))$ where Δ is a Pareto optimal solution. If $\mathcal{P}$ is a complete and minimal set of Pareto optimal solutions, the set $\{(D(\Delta), S(\Delta)) \mid \Delta \in \mathcal{P}\}$ is equal to the Pareto front. Figure 2 gives an illustration of the Pareto front. A point in the Pareto front can correspond to several partitions.

If the user specifies a function on the criteria to optimize, for example $\max(S/D)$ or $\min[\alpha D - (1-\alpha)S]$ with $0 \leq \alpha \leq 1$, the optimal solution will be among the Pareto optima.

Let us consider, for instance, the example given in Figure 1. When the number of classes is set to 3, a complete and minimal set of Pareto solutions is given in Figure 3. If the ratio S/D is minimized, the optimal solution is solution 5, which is the one that fits the best the intuitive groups. The user can specify conditions on the desired solutions. If for instance it is specified that points 5 and 14 must be in the same cluster, then only solutions 5 and 6 are found. If another condition is added, requiring the size of each group to be at least 2, only solution 5 is found.

A bi-criterion clustering algorithm finding a complete and minimal set of Pareto solutions for different values of the number k of clusters is proposed in [5]. When $k = 2$, an exact polynomial algorithm is proposed in [12, 13]. However, to the best of our knowledge, there is no algorithm dealing with this bi-criterion, while supporting various kinds of user constraints.

2.3. Constraint Programming

Constraint Programming (CP) is a powerful paradigm to solve combinatorial problems, based on Artificial Intelligence or Operational Research methods. A *Constraint Satisfaction Problem (CSP)* is a triple $\langle X, Dom, C\rangle$ where:

- $X = \langle x_1, x_2, \ldots, x_n\rangle$ is a n-tuple of variables,
- $Dom = \langle Dom(x_1), Dom(x_2), \ldots, Dom(x_n)\rangle$ is a corresponding n-tuple of domains such that $x_i \in Dom(x_i)$,

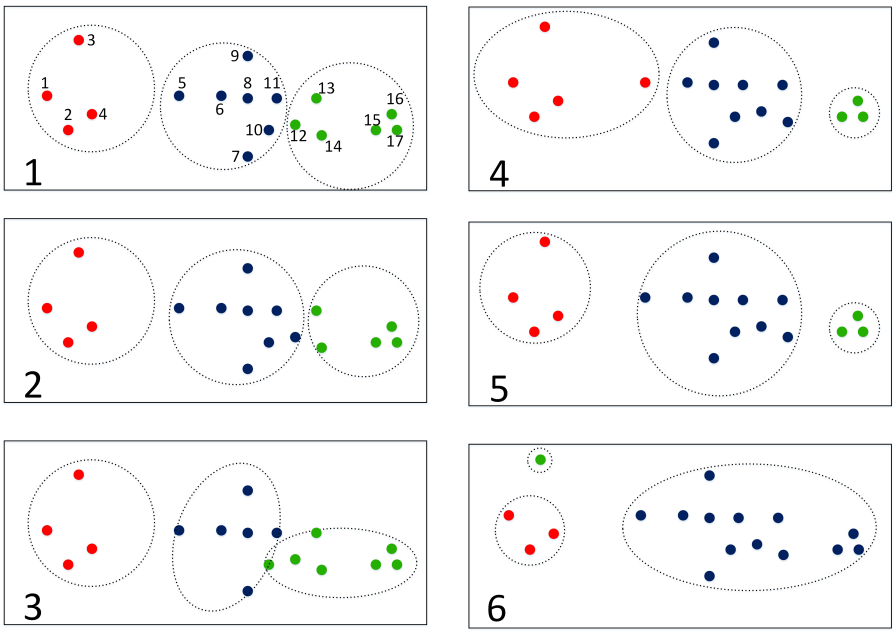

Figure 3: Pareto optimal solutions

- $C = \langle C_1, C_2, ..., C_t \rangle$ is a t-tuple of constraints where each constraint C_i expresses a condition on a subset of X.

A solution of a CSP is a complete assignment of values from $Dom(x_i)$ to each variable x_i that satisfies all the constraints of C. A *Constraint Optimization Problem (COP)* is a CSP with an objective function to be optimized. An optimal solution of a COP is a solution of the CSP that optimizes the objective function. In general, solving a CSP is NP-Hard. Nevertheless, the methods used by the solvers enable to efficiently solve a large number of real applications. They rely on constraint propagation and search strategies.

Constraint propagation of a constraint c reduces the domain of the variables of c, by removing some or all inconsistent values, *i.e.*, values that cannot be part of a solution of c. A set of propagators is associated to each constraint, it depends on the kind of consistency required for this constraint. If arc consistency is required, the propagators remove all the inconsistent values in each domain. If bound consistency is required, the propagators modify only the bounds of the domains. The type of consistency is chosen by the programmer when the constraint is defined. Different kinds of constraints are available for the programmer; they can be elementary constraints expressing arithmetic or logic relations, or global constraints expressing meaningful n-ary relations. One of the best known global constraints is the constraint *alldifferent*$(x_1, \ldots, x_n)$, which imposes the variables x_i to be pairwise different. Global constraints benefit from efficient propagation, performed by a filtering algorithm exploiting results from other domain as for instance graph theory. From a logical point of view, a global constraint is equivalent to a conjunction of elementary constraints, e.g. the constraint *alldifferent*(x_1, x_2, x_3) is equivalent to the conjunction of binary constraints $x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3$. The interesting point is that a global constraint with its filtering algorithm has much more powerful propagation than the set of propagators of the elementary constraints. Different global constraints are developed, each one aims at exploiting more efficiently an n-ary relation. Filtering algorithms for global constraints use operational research techniques or graph theory to achieve generalized arc consistency or bound consistency with low complexity. A catalog of global constraints with more than 400 inventoried global constraints is maintained in [14].

Example 2.1. Let $X = \{x_1, x_2, x_3\}$ with $Dom(x_i) = \{1, 2\}$. Let P_1 be a CSP defined on X by the constraints:

$$x_1 \neq x_2,\ x_1 \neq x_3,\ x_2 \neq x_3.$$

The arc consistency for each individual constraint $x_i \neq x_j$ cannot remove any value from the domains $Dom(x_i)$ and $Dom(x_j)$, since each value is part of a solution ($x_i = 1, x_j = 2$ and $x_i = 2, x_j = 1$). The CSP P_1 is however inconsistent,

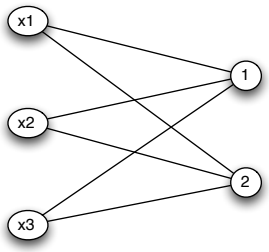

Figure 4: Value graph for *alldifferent*(x_1, x_2, x_3) with $Dom(x_i) = \{1, 2\}$

there is no solution that satisfies all the constraints, but the propagation of individual constraints cannot detect it. Let P_2 be the CSP defined on X by the single constraint *alldifferent*(x_1, x_2, x_3). The filtering algorithm for this constraint [15] maintains the bipartite graph $G = (V, E)$, with $V = \{x_1, x_2, x_3\} \cup \{1, 2\}$ and $E = \{(x_i, v) \mid v \in Dom(x_i)\}$. This bipartite graph, which is also called the value graph of X, is given in Figure 4. A matching $M \subseteq E$ is a set of disjoint edges, *i.e.* two edges in M cannot share a vertex. Two important observations on the relationship between the constraint *alldifferent*$(x_1, \ldots, x_n)$ and matching are introduced in [15]:

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

From these observations, the filtering algorithm can detect inconsistencies and can remove all the inconsistent values. In this example, the bipartite graph G has no matching of cardinality 3. The constraint *alldifferent*(x_1, x_2, x_3) is then inconsistent.

In a CP solver, two steps, constraint propagation and branching, are repeated until a solution is found. Constraints are propagated until a stable state, in which the domains of the variables are reduced as much as possible. If the domains of all the variables are reduced to singletons then a solution is found. If the domain of a variable becomes empty, then there exists no solution with the current partial assignment and the solver backtracks. In the other cases, the solver chooses a variable whose domain is not reduced to a singleton and splits its domain into different parts, thus leading to new branches in the search tree. The solver then explores each branch, activating constraint propagation since the domain of a variable has been modified.

The search strategy can be determined by the programmer. If a depth-first strategy is used, the solver orders branches following the order given by the programmer and explores in depth each branch. For a constraint optimization problem, a branch-and-bound strategy can be integrated to a depth-first search: each time a solution, *i.e.* a complete assignment of variables satisfying the constraints, is found, the value of the objective function for this solution is computed and a new constraint is added, expressing that a new solution must be better than this one. Assume that the objective function is represented by a variable y, which is to be minimized. When a solution to the problem is found, its corresponding objective value f is computed and the constraint $y < f$ is added. This implies that only the first best solution found is returned by the solver. The solver performs a complete search, pruning only branches that cannot lead to a solution and therefore finds an optimal solution. The choice of variables and of values at each branching is very important, since it may drastically reduce the search space and therefore computation time.

In the context of constraint optimization problems, an optimization constraint is a global constraint that is linked to the objective function. Each solution induces a "cost" and the global constraint exploits this cost to filter not only the variable which represents the objective function, but also other decision variables inside the constraint. The first filtering algorithm for this kind of global constraints is proposed in [16]. A well-known example of extension of global constraints to optimization constraints is the constraint *cost_gcc* [17], which extends the Global Cardinality Constraint with cost. For more details on global constraints, search and more generally on CP, we refer the reader to [18].

Example 2.2. Let us illustrate by a simple COP: find an assignment of letters to digits such that $SEND + MOST = MONEY$, and maximizing $MONEY$. This problem can be modeled by a COP with eight variables S, E, N, D, M, O, T, Y,

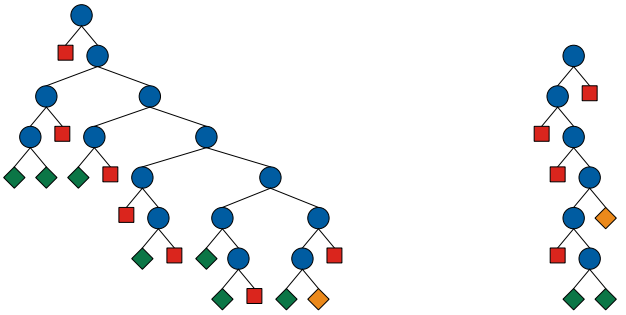

Figure 5: Search trees with variable choice S, E, N, D, M, O, T, Y (left) and S, T, Y, N, D, E, M, O (right)

having as domain the set of digits $\{0, \ldots, 9\}$ and a variable V of domain integer, which represents the objective function, which is to be maximized. Constraints for this problem are:

- the first digits must be different from 0: $S \neq 0,\ M \neq 0$,
- the values of the letters are pairwise different: $alldifferent(S, E, N, D, M, O, T, Y)$,
- $(1000S + 100E + 10N + D) + (1000M + 100O + 10S + T) = 10000M + 1000O + 100N + 10E + Y$,
- $V = 10000M + 1000O + 100N + 10E + Y$.

The initial constraint propagation leads to a stable state, with the domains: $D_S = \{9\}$, $D_E = \{2, 3, 4, 5, 6, 7\}$, $D_M = \{1\}$, $D_O = \{0\}$, $D_N = \{3, 4, 5, 6, 7, 8\}$ and $D_D = D_T = D_Y = \{2, 3, 4, 5, 6, 7, 8\}$. Since some domains are not reduced to singletons, branching is then performed. At the end of the search, we get the optimal solution with the assignment $S = 9, E = 7, N = 8, D = 2, M = 1, O = 0, T = 4, Y = 6$, leading to $MONEY = 10876$.

Strategies specifying the way branching is performed are very important. When variables are chosen in the order S, E, N, D, M, O, T, Y and when values are chosen following an increasing order, the search tree is composed of 29 nodes and 7 intermediary solutions (solutions satisfying all the constraints, better than the previous ones found but not optimal). When variables are chosen in the order S, T, Y, N, D, E, M, O, the search tree has only 13 nodes and 2 intermediary solutions. Figure 5 presents the two corresponding search trees, which are generated by Gist environment of the Gecode solver [19]. 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 (the last diamond) is the optimal solution.

3. Related Work

Due to the hardness of the clustering problem, there are few exact algorithms in the literature, and the algorithms used are often heuristic, metaheuristic or approximation algorithms. Finding a partition maximizing the split between clusters is a polynomial problem [5] but it becomes NP-Hard with user-constraints such as cannot-link constraints [20]. Concerning the minimization of the maximal diameter, the problem is polynomial for $k = 2$, but it becomes NP-Hard when $k \geq 3$ [21].

An exact algorithm based on graph coloring is proposed in [21]. Graph coloring is used to check if a distance between two objects can be the maximal diameter. Another exact approach uses a *branch-and-bound* search [4]. The algorithm uses a hierarchical algorithm to find a good bound and a reordering point strategy to reduce the search space. For the criterion of minimizing the within-cluster sum of dissimilarities, a repetitive branch-and-bound algorithm is presented in [4]. To our knowledge, there is no exact algorithm that supports user-constraints with any of those criteria with $k > 2$.

For the split-diameter bi-criterion optimization without user-constraints, an algorithm finding a complete and minimal set of Pareto optimal solutions, which are partitions with at most k_{max} clusters, is proposed in [5]. It is

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

In the case of bi-partition ($k = 2$), an exact polynomial algorithm to find Pareto optimal solutions is proposed in [12, 13]. For $k > 2$, [13] also offers a 2-approximation algorithm. These two algorithms [12, 13] are both based on the principle of [5]: a spanning tree is built to find the possible values for split and graph coloring tests are used to verify if a dissimilarity can be the maximal diameter. However none of these bi-criterion cluster analysis approaches does support any kind of user-constraints.

Most of the attention in constrained clustering has been put on instance-level constraints, *i.e.* must-link and cannot-link constraints [22]. They were first introduced by Wagstaff [7]. Subsequently, many works have been done to extend classical algorithms for handling must-link and cannot-link constraints, as for instance an extension of COBWEB [7], of k-means [23, 24], hierarchical non supervised clustering [25] or spectral clustering [26, 27], etc. When the constraints are tight, most of those algorithms may not find a solution that satisfies all the constraints even if such a solution exists.

In recent years, it has been realized that many problems in Data Mining, including constrained clustering, can be solved by generic optimization tools. Recent works investigate generic frameworks such as Constraint Programming, SAT or Integer Linear Programming.

In [28], L. De Raedt *et al.* present a framework in Constraint Programming for k-pattern set mining and show how it can be applied to conceptual clustering. In conceptual clustering, an intentional definition represented by a pattern is associated to each class. The objective is to find pairs composed of classes and patterns, such that the elements in a class satisfy the pattern. Constraints are imposed on patterns and groups. In order to find interesting solutions, some optimization criteria can be introduced. J.-P. Métivier *et al.* present in [29] a constraint-based language expressing queries to discover patterns in Data Mining. Conceptual clustering tasks can be expressed by queries as well as some kinds of user constraints. The language elements are translated into SAT clauses which are solved by a SAT solver.

Davidson *et al.* propose a SAT framework [1] for constrained clustering, but only for problems with $k = 2$. Several kinds of constraints are considered: must-link, cannot-link, diameter and split constraints. The algorithm allows to obtain a global optimum with the criterion of diameter or split.

Mueller *et al.* propose in [2] an approach to constrained clustering based on Integer Linear Programming. This approach takes a set of candidate clusters as input and builds 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. It integrates different objective functions based on the quality of the clusters composing the clustering. The framework guarantees to find a global optimum but requires a set of candidate clusters. This condition makes the framework less applicable for clustering in general, since finding a good set of candidate clusters is a difficult task as the number of candidate clusters is exponential compared to the number of objects. This approach is experimented for conceptual clustering where candidate clusters might be generated from frequent patterns.

Recently, Babaki *et al.* present in [30] an exact approach for constrained clustering with the criterion of minimizing the within-cluster sum of squares, based on Integer Linear Programming. This approach extends an exact algorithm which uses column generation [31]. It allows must-link, cannot-link and all constraints that are anti-monotone. User-constraints are handled within the branch-and-bound search, used for generating new columns. This approach is experimented in [30] with small datasets containing less than 200 objects. For clustering task which maximizes the inter-cluster distances, Kotthoff *et al.* present in their talk [32] a Constraint Programming approach and also assert the flexibility and opportunities provided by a CP formulation.

Other tasks of clustering are based on a similarity graph between objects. Spectral clustering is a clustering task which aims at minimizing the ratio cut criterion[2] [33]. Wang *et al.* present in [27, 34] a flexible framework

[2]Based on a similarity measure $s(o_i, o_j)$ between the objects, the ratio cut criterion is defined by $(1/2)\sum_{c\in[1,k]}(1/|C_c|)\sum_{o_i\in C_c, o_j\notin C_c} s(o_i, o_j)$.

for spectral clustering. The framework integrates different kinds of constraints and allows also to specify a threshold setting a lower bound on how well the given constraints are satisfied. Zhi *et al.* present in [35] a framework for spectral clustering which integrates logical combinations of constraints. Logical combinations of constraints are expressed as linear equalities and inequalities so that they can be incorporated into various mathematical programming formulations for clustering.

Multi-view spectral clustering is an extension of spectral clustering to multi-view datasets. Instead of combining different views into a single objective function, Davidson *et al.* propose in [36] a natural formulation that treats the problem as a multi-objective problem and solve it using Pareto optimization.

The clustering tasks we are interested in aim at finding a partition of objects. Another clustering approach is hierarchical clustering, which aims at finding a hierarchy of partitions, that is a sequence of nested partitions. The result is a tree diagram, called a dendrogram. A framework formalizing hierarchical clustering as an Integer Linear Programming problem has recently been proposed by Gilpin *et al.* [37]. Gilpin *et al.* also propose in [38] a framework based on SAT for hierarchical constrained clustering with different types of user-constraints.

Another clustering setting is correlation clustering, which is based on a similarity graph between objects, and which aims at finding a partition that agrees as much as possible with the similarities. Berg *et al.* present in [39] a MaxSAT framework for constrained correlation clustering. In this framework, hard clauses are used to ensure a well-defined clustering and soft clauses are used to encode the cost function.

In this paper, we investigate the use of Constraint Programming for constrained clustering. Constraint Programming has already been shown to be a promising approach for Data Mining through various tasks, such as itemset mining [40, 41, 42, 43, 44], skypattern mining [45] or decision tree construction [46].

4. New CP Model for Constrained Clustering

We are given a collection of n points and a dissimilarity measure between pairs of points i, j, denoted by $d(i, j)$. Without loss of generality, let us suppose that points are indexed and named by their index (1 represents the first point). The model aims at finding a partition into k classes satisfying a set of user constraints and optimizing a given criterion.

The model we propose is composed of a set of CP constraints. They are used to model partition requirements, the optimization criterion and different kinds of user constraints. Thus, they can be separated into three groups:

- the CP constraints expressing that the result must be a partition,
- the CP constraints expressing the user constraints,
- the CP constraints expressing the criterion to be optimized. Please note that when no optimization criterion is given, the CP solver searches for all the partitions satisfying all the constraints.

In a previous work [3], we have presented a CP model for this task. This model was based on a two-level representation: a set of variables for the assignment of a representative to each class and a set of variables for the assignment of a representative to each point. Choosing such a representation requires the number of classes k to be fixed beforehand, since each representative is modeled by a CP variable. In this paper, we introduce a new CP model, which is based only on a set of variables for the assignment of a number (or index) of class to each point. As a result, the number of classes k can be only bounded by k_{min} and k_{max}, where k_{min} and k_{max} are given by the user. In the following, we present the two models to ease the comparison. In both models, the CP constraints expressing the user constraints are similar but those that express partition requirements and optimization criteria are different. All the optimization criteria in the new model are expressed by new global constraints with a filtering algorithm. The differences between the two models are significant, since the new model has much less variables and constraints, while being more efficient than the previous model. Table 1 summarizes the differences between the two models.

4.1. Variables

In the first model, for each cluster $c \in [1, k]$, the point with the smallest index is considered as the representative point of the cluster[3]. An integer variable I_c is introduced, its value is the index of the representative point of cluster

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

c; the domain of I_c is therefore the set of integers $[1, n]$. Let $\mathcal{I}$ be the array $[I_1, \ldots, I_k]$. Assigning a point to a cluster becomes assigning the point to the representative of the cluster. Therefore, for each point $i \in [1, n]$, an integer variable $G_i \in [1, n]$ is introduced: G_i is the representative point of the cluster which contains the point i.

Example 4.1. Assume that we have 7 points $o_1, \ldots, o_7$ and 2 clusters, the first one composed of o_1, o_2, o_4 and the second one composed of the remaining points. The points are denoted by their index (o_1 is denoted by 1, o_2 by 2 and so on). Then $I_1 = 1$ and $I_2 = 3$ (since 1 is the smallest index among $\{1, 2, 4\}$ and 3 is the smallest index among $\{3, 5, 6, 7\}$), $G_1 = G_2 = G_4 = 1$ (since 1 is the representative of the first cluster) and $G_3 = G_5 = G_6 = G_7 = 3$ (since 3 is the representative of the second cluster).

In the new model, 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 integer variables $G_1, \ldots, G_n$ whose domain is the set of integers $[1, k_{max}]$. An assignment $G_i = c$ means that the point i is put into the cluster number c.

The domains of the variables G_i in the two models are different, but the meaning of these variables is identical: they represent the assignment of points to clusters. Let $\mathcal{G}$ denote the array $[G_1, \ldots, G_n]$.

To represent each optimization criterion, in both models, a float value variable is introduced. It is named D for the diameter criterion, S for the split criterion and V for the WCSD criterion. Their domains are $Dom(D) = Dom(S) = [\min_{i,j}(d(i, j)), \max_{i,j}(d(i, j))]$ and $Dom(V) = [0, \sum_{i<j} d(i, j)]$.

4.2. Partition Constraints

4.2.1. First model

To express that the result must be a partition, we put the following constraints:

- Each representative belongs to its cluster: for each $c \in [1, k]$, we put $G_{I_c} = I_c$. This constraint is represented by the CP constraint *element*$(\mathcal{G}, I_c, I_c)$. The constraint *element*(A, I, V) with A an array of variables and I, V variables, sets the relation $A[I] = V$.
- Each point is assigned to a representative: for each $i \in [1, n]$, we need $\bigvee_{c \in [1,k]}(G_i = I_c)$. This relation can be expressed by $\#\{c \mid I_c = G_i\} = 1$, which is represented by a CP constraint *exactly*$(1, \mathcal{I}, G_i)$. This constraint sets the relation requiring that the value of G_i must appear exactly once in the array $\mathcal{I}$.
- The representative of a cluster is the point which has the minimal index in the cluster; in other words, the index i of a point is greater or equal to the index of its representative given by G_i: for each $i \in [1, n]$, we put $G_i \leq i$.

A set of clusters could be differently represented, depending on the order of clusters. For instance, in Example 4.1, we could have chosen $I_1 = 3$ and $I_2 = 1$, thus leading to another representation of the same set of clusters. To avoid this symmetry, the following constraints are added:

- The representatives are sorted in an increasing order: $\forall c < c' \in [1, k],\ I_c < I_{c'}$.
- The representative of the first cluster is the first point: $I_1 = 1$.

4.2.2. Second model

In this model, clusters are identified by their number (index), and each variable G_i gives the index of the cluster that contains point i. A complete assignment of the variables in $\mathcal{G}$ represents a partition. However, a partition can be represented by different complete assignments of $\mathcal{G}$. For instance, given a complete assignment of $\mathcal{G}$, if we make a permutation where all the variables G_i that have the value c_1 take the value c_2 and at the same time, all the variables G_j having the value c_2 take the value c_1, we get a new assignment for $\mathcal{G}$, which still represents the same partition in terms of classes. As a second example, when all the variables G_i with value c_1 receive a value c_3 that is not yet used in an assignment of the other variables of $\mathcal{G}$, this leads to a new assignment representing a symmetric solution. Such a situation appears when building the clusters, a new created cluster can receive any value among the remaining cluster numbers.

To break this kind of symmetries, the clusters are numbered such that the number 1 is the index of the first created cluster and a new number c, with $c > 1$, is used if and only if the number $c-1$ has been already used. A straightforward

way to express this condition is by using a constraint $G_1 = 1$ and the constraints $G_i \leq \max_{j\in[1,i-1]}(G_j)+1$, for $i \in [2, n]$. However, in order to have better interactions and propagations between these relations, a better way is to sum up these relations into one global constraint with a good filtering algorithm. The constraint *precede* [47] helps to achieve this:

$$precede(\mathcal{G}, [1, \ldots, k_{max}]).$$

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 $j < i$ such that $G_j = c - 1$.

The requirement to have 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 the relation $\#\{i \mid G_i = k_{min}\} \geq 1$, which can be represented by the CP constraint:

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

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 $k_{min} = k_{max} = k$.

4.3. User constraints

All popular user-defined constraints may be straightforwardly integrated. They are expressed the same way in both models, since they rely on the use of the variables in $\mathcal{G}$, which represent the assignment of points to clusters.

- Minimal size α of clusters: this means that each point must be in a cluster with at least α points (including itself). For each $i \in [1, n]$, the assigned value of the variable G_i must then appear at least α times in the array $\mathcal{G}$, *i.e.* $\#\{j \mid G_j = G_i\} \geq \alpha$. Therefore, for each $i \in [1, n]$, we put the constraint: $atleast(\alpha, \mathcal{G}, G_i)$.

  This constraint helps also to set a bound on the number of possible clusters. Indeed, the number of clusters cannot exceed $\lfloor n/\alpha \rfloor$. In the second model, this can be expressed by $G_i \leq \lfloor n/\alpha \rfloor$, for all $i \in [1, n]$.

- Maximal size β of clusters: each number $c \in [1, k_{max}]$ must appear in the array $\mathcal{G}$ at most β times (this is still true for an unused value $c \in [k_{min} + 1, k_{max}]$, since it appears 0 time), *i.e.* $\#\{i \mid G_i = c\} \leq \beta$. Therefore, for each $c \in [1, k_{max}]$, we put the constraint: $atmost(\beta, \mathcal{G}, c)$.

  In the second model, this relation also entails: $G_i \geq \lceil n/\beta \rceil$, for all $i \in [1, n]$.

- Minimal split δ: a δ-constraint requires that the split between two clusters must be at least δ. Therefore for each couple $i < j \in [1, n]$ such that $d(i, j) < \delta$, the constraint $G_i = G_j$ is put. The constraint $S \geq \delta$ is also put.

- Maximal diameter γ: a diameter constraint requires that the diameter of each cluster must be at most γ. The constraint $D \leq \gamma$ is put and for each couple $i < j \in [1, n]$ such that $d(i, j) > \gamma$, we put: $G_i \neq G_j$.

- Density constraint: a density constraint expresses that each point must have in its neighborhood of radius ϵ, at least m points belonging to the same cluster as itself. So for each $i \in [1, n]$, the set of variables corresponding to points in its ϵ-neighborhood is computed $\mathcal{N}_{i\epsilon} = \{G_j \mid d(i, j) \leq \epsilon\}$ and this constraint is put: $atleast(m, \mathcal{N}_{i\epsilon}, G_i)$.

- Must-link constraint: a must-link constraint on two points i, j is expressed by: $G_i = G_j$ and $D \geq d(i, j)$.

- Cannot-link constraint: a cannot-link constraint on i, j is expressed by: $G_i \neq G_j$ and $S \leq d(i, j)$.

4.4. Optimization criteria

In the first model, we have proposed to model the optimization criterion by reified CP constraints.

- When minimizing the maximal diameter, since D represents the maximal diameter, any two points at a distance greater than D must be in different clusters:

  $$\forall i < j \in [1, n], \quad (d(i, j) > D) \rightarrow (G_i \neq G_j).$$

  Since D is a variable and its value is still unknown, these relations are expressed using reified constraints[4].

[4] A reified constraint is a logical constraint of the form $A \rightarrow B$ or $A \leftrightarrow B$, where A and B are also constraints. The reified constraint $A \rightarrow B$ means the constraint B must be satisfied if A is satisfied, and $\neg A$ must be satisfied if $\neg B$ is satisfied.

- When maximizing the minimal split between clusters, any two points at a distance less than the minimal split S must be in the same cluster:
$$\forall i < j \in [1, n], \quad (d(i, j) < S) \rightarrow (G_i = G_j).$$
Since S is a variable, these relations are also expressed by reified constraints.

- When minimizing the Within-Cluster Sum of Dissimilarities (WCSD):
$$V = \sum_{i,j \in [1,n]} (G_i = G_j) d(i, j).$$
For this relation, we developed a global constraint $wcsd(\mathcal{G}, V, d)$ with a filtering algorithm.

When minimizing the diameter, in [3] we use heuristics provided by the algorithm FPF [6] to get a lower and an upper bound on the diameter without user constraints, and only an upper bound in the presence of user-constraints. Such bounds allow to reduce the number of reified constraints that are put in the model.

In the new model, for the diameter and split criteria, instead of using reified constraints, we develop two global constraints $diameter(\mathcal{G}, D, d)$ and $split(\mathcal{G}, S, d)$, which exploit the measure d and which operate on the array $\mathcal{G}$ and on the variable D or S. The filtering algorithms for the three constraints $diameter(\mathcal{G}, D, d)$, $split(\mathcal{G}, S, d)$ and $wcsd(\mathcal{G}, V, d)$ are presented in Section 5.

If the user specifies an optimization criterion, an objective function is put in the model, which is:

- *minimize* D in case of minimizing the maximal diameter,
- *maximize* S in case of maximizing the minimal split,
- *minimize* V in case of minimizing the WCSD.

When an optimization criterion is specified, if there exist solutions that satisfy all the constraints, the solver finds an optimal solution, which is a global optimum. If the user does not specify any optimization criterion, the solver finds all the solutions satisfying all the constraints, if some exists.

4.5. Search strategy

Symmetry breaking for partition constraints in the two models is based on the indices of points, such as the constraints $I_c < I_{c'}$ for all $c < c'$ in the first model or the constraint $precede(\mathcal{G}, [1, .., k_{max}])$ in the second model. The way points are indexed is therefore really important. Points are then reordered and reindexed, so that points that are far from the others have a small index. In order to achieve this, we rely on the Furthest Point First algorithm [6]. This algorithm starts by choosing a point, marks it as the first head, links all the points to it and iterates until all points are marked. At each iteration, it chooses the point i that is the furthest to its head, marks it as a new head and links to i all the points that are closer to i than to their head.

The search strategy in the first model is based on instantiating the variables in $\mathcal{I}$ before the variables in $\mathcal{G}$. This means that cluster representatives are identified before assigning points to clusters. Variables in $\mathcal{I}$ are chosen from I_1 to I_k. Since the representative is the point with the minimal index in the cluster, values for instantiating each I_c are chosen in an increasing order. The choice of variables and values in $\mathcal{G}$ depends on the criterion. For the diameter and split criteria, a variable G_i with the smallest remaining domain is chosen. Recall that each value in $j \in Dom(G_i)$ is the index of a cluster representative. All values in $Dom(G_i)$ are examined and the value j which corresponds to the smallest value $d(i, j)$ is chosen and two alternatives are created $G_i = j$ and $G_i \neq j$.

In the new model, the search strategy is based on the choice of variables and values in $\mathcal{G}$, and depends also on the criterion. For the diameter and split criteria, at each branching point, a variable G_i with the smallest remaining domain is chosen. Recall that each value $c \in Dom(G_i)$ is the number of a cluster. All values in $Dom(G_i)$ are examined and the number of the closest cluster to i is chosen. The distance between a point i and a cluster number c is defined as the maximal distance $d(i, j)$ where G_j is already instantiated to c. If the cluster number c is empty (there is no point j such that $G_j = c$), the distance between i and the cluster c is set to zero. This means that the assignment of a point to a new cluster is favored if there are unused cluster numbers. Moreover, the smallest remaining number is chosen. The closest cluster c to the point i is chosen and two alternatives are created $G_i = c$ and $G_i \neq c$.

	First model	Second model
Var.	$\mathcal{I} = [I_1, \ldots, I_k]$, $Dom(I_c) = [1, n]$ $\mathcal{G} = [G_1, \ldots, G_n]$, $Dom(G_i) = [1, n]$	$\mathcal{G} = [G_1, \ldots, G_n]$, $Dom(G_i) = [1, k_{max}]$
	D (diameter), S (split), V (WCSD)	
Partition	$\forall c \in [1, k]$, $element(\mathcal{G}, I_c, I_c)$ $\forall i \in [1, n]$, $exactly(1, \mathcal{I}, G_i)$ $\forall i \in [1, n]$, $G_i \leq i$ $\forall c < c' \in [1, k]$, $I_c \leq I_{c'}$ $I_1 = 1$	$precede(\mathcal{G}, [1, \ldots, k_{max}])$ $atleast(1, \mathcal{G}, k_{min})$
User-constraints	Minimal size α of clusters: $\forall i \in [1, n]$, $atleast(\alpha, \mathcal{G}, G_i)$ Maximal size β of clusters: $\forall c \in [1, k_{max}]$, $atmost(\beta, \mathcal{G}, c)$ Minimal split δ: $S \geq \delta$, $G_i = G_j$, for all $i < j$ st. $d(i, j) < \delta$ Maximal diameter γ: $D \leq \gamma$, $G_i \neq G_j$ for all $i < j$ st. $d(i, j) > \gamma$ Density constraint: $\forall i \in [1, n]$, $atleast(m, N_{i\epsilon}, G_i)$ Must-link constraint: $G_i = G_j$, $D \geq d(i, j)$ Cannot-link constraint: $G_i \neq G_j$, $S \leq d(i, j)$	
Opt. Crit.	WCSD criterion: $wcsd(\mathcal{G}, V, d)$	
	Diameter criterion:	
	$\forall i < j \in [1, n]$, $d(i, j) > D \rightarrow (G_i \neq G_j)$	$diameter(\mathcal{G}, D, d)$
	Split criterion:	
	$\forall i < j \in [1, n]$, $d(i, j) < S \rightarrow (G_i = G_j)$	$split(\mathcal{G}, S, d)$

Table 1: Comparison between the two models.

Concerning the WCSD criterion, a mixed strategy is used in both models. In order to have a good upper bound for the variable V, a greedy search is used to quickly find a solution. At this step, the chosen variable G_i and value c are those such that the assignment $G_i = c$ increases V as little as possible. The first solution found is in general quite good. After finding the first solution, the strategy changes to a "first-fail", which tends to detect failures quickly. In this strategy, a value s_{ic} for each point i and each cluster c is defined as the sum of dissimilarities between i and all points j already assigned to the cluster c. At each branching point, for all points i with G_i uninstantiated, 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.

5. Filtering Algorithms for Optimization Criteria

For each optimization criterion, we have developed a filtering algorithm for the global constraint, which links the variables $\mathcal{G}$ representing a partition to the variable representing the objective function (D, S or V). This kind of global constraints is also called optimization constraint [18]. When a solution is found, its corresponding objective value is computed and a constraint expressing that new solutions must have a better value than this one is added. This constraint sets a new upper bound for D or V, which have to be minimized, or a new lower bound for S, which has to be maximized. By reasoning globally on the objective variable and on the variables representing a partition, more interactions between the domains of these variables can be captured and the search subspaces can be pruned before instantiating all the variables.

5.1. Diameter and Split Criteria

To represent the relations between points and the diameter or the split, we develop filtering algorithms for two global constraints $diameter(\mathcal{G}, D, d)$ and $split(\mathcal{G}, S, d)$, which exploit the dissimilarity measure d between any two

points and which operate on the array $\mathcal{G}$ and the variable D or S. The constraint $diameter(\mathcal{G}, D, d)$ ensures that D is the maximal diameter of the clusters formed by the variables $G_1, \ldots, G_n$. This constraint ensures:

$$\forall i < j \in [1, n], \quad D < d(i, j) \;\rightarrow\; G_i \neq G_j. \tag{1}$$

This kind of relation can be realized by reified constraints, which were indeed used in our previous model [3]. However, a reified constraint is needed for each couple $i < j$, which implies that the number of reified constraints would be quadratic with respect to the number of points. By developing the constraint $diameter(\mathcal{G}, D, d)$, we maintain all these relations in one constraint. The filtering algorithm is presented in Algorithm 1. In this algorithm, $Dom(D)$ is represented by $[D.lb, D.ub)$, where $D.lb$ is the lower bound, which initially can be the minimal dissimilarity between two points, and $D.ub$ is the upper bound, which can be either the maximal dissimilarity between two points, or the value of D in the previous solution found. The bound $D.ub$ is strict since in branch-and-bound search the next solution must have D value strictly smaller than the previous one. The relation (1) is useful when the following cases happen.

- The upper bound $D.ub$ has been changed (e.g. by a new found solution or by a diameter constraint). In this case, for each couple i, j, if $D.ub \leq d(i, j)$, we can conclude $D < d(i, j)$ and by (1) we can infer $G_i \neq G_j$. However, the relation $G_i \neq G_j$ is useful to filter the domain of G_i (or the domain of G_j) only if the variable G_j (or G_i, resp.) has been instantiated. Therefore, Algorithm 1 uses a stack to remember the variables G_i that are instantiated (lines 2–4) and exploits them to filter (line 9). The lower bound $D.lb$ can possibly be revised. (line 10).
- Some variables G_i have been instantiated. In this case, for each couple i, j such that G_i and G_j are instantiated and have the same value, we infer $D \geq d(i, j)$ and can revise $D.lb$. The stack remembers then the variables G_i which have just been instantiated (lines 5–6). This can lead to the revision of the lower bound $D.lb$ (line 10).

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 $O(n^2)$. This algorithm is awaken when the upper bound of D is modified or a variable G_i is instantiated. However, because of its complexity, it is scheduled to be effective after other constraints whose propagators are of lower complexity, as for instance constraints representing must-link or cannot-link constraints.

The constraint $split(\mathcal{G}, S, d)$, on the other hand, maintains that S is the minimal split between the clusters formed by the variables $G_1, \ldots, G_n$. It ensures that:

$$\forall i < j \in [1, n], \quad S > d(i, j) \;\rightarrow\; G_i = G_j. \tag{2}$$

The filtering algorithm is presented in Algorithm 2, where $Dom(S) = (S.lb, S.ub]$. The lower bound $S.lb$ is either the minimal dissimilarity between two objects or the value of S in the previous solution found, since S is to be maximized. In the same manner as for the constraint $diameter(\mathcal{G}, D, d)$, this algorithm is invoked if the lower bound $S.lb$ has been changed or some variables in $\mathcal{G}$ have been instantiated. In this algorithm, if $S.lb$ has been changed, for each couple i, j, if $S.lb \geq d(i, j)$, by (2) we can infer $G_i = G_j$, which is propagated by enforcing $Dom(G_i) = Dom(G_j)$. Otherwise, if some variables in $\mathcal{G}$ have been instantiated, if $G_i \neq G_j$ by (2) we infer $S \leq d(i, j)$, so the upper bound of S can be changed. The worst case complexity is $O(n^2)$.

5.2. Within-Cluster Sum of Dissimilarities Criterion

We have developed a filtering algorithm for a new global optimization constraint $wcsd(\mathcal{G}, V, d)$, which links the variable V, the array of variables $\mathcal{G}$ and which exploits the dissimilarity measure d. This constraint ensures the relation:

$$V = \sum_{1 \leq i < j \leq n} (G_i = G_j) d(i, j). \tag{3}$$

where $G_i = G_j$ is 1 if G_i and G_j have the same value and 0 otherwise. The filtering algorithm used in the first model as well as motivations and proofs are presented in [48]. However, in [48] the algorithm is designed for the clustering task with exactly k clusters, as the case considered in the first model. We present below a generalization of the algorithm for clustering tasks where the number of clusters is only bounded, as considered in the second model.

Let us assume that we have a partial assignment of variables in $\mathcal{G}$. Let $K = \{i \in [1, n] \mid G_i \text{ is assigned}\}$ and $U = \{i \in [1, n] \mid G_i \text{ is unassigned}\}$. We use the computation of a lower bound proposed in [49], which takes into account the unassigned variables. The sum defining V can be split into three parts $V = V_1 + V_2 + V_3$, where:

Algorithm 1: Filtering for constraint *diameter*($\mathcal{G}, D, d$)

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

Algorithm 2: Filtering for constraint *split*($\mathcal{G}, S, d$)

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

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

$$V_1 = \sum_{i,j \in K, i<j} (G_i = G_j) d(i,j)$$

- V_2 is the sum of dissimilarities between the unassigned points and the assigned points:

$$V_2 = \sum_{i \in U, j \in K} (G_i = G_j) d(i,j)$$

- V_3 is the sum of dissimilarities between the unassigned points:

$$V_3 = \sum_{i,j \in U, i<j} (G_i = G_j) d(i,j)$$

Since the set K is already known, the exact value of V_1 can be computed. Since the points of U have not been assigned to a cluster, the value of V_2 is unknown. However, a lower bound of V_2, denoted by $V_2.lb$, can be computed by the sum of the minimal contribution of all unassigned points. For each unassigned point $i \in U$, each value $c \in Dom(G_i)$ represents an index of cluster to which point i can be assigned to. If point i is assigned to the cluster number c, it will contribute to that cluster the sum of dissimilarities between point i and all the assigned points which are in cluster c, i.e. the sum of dissimilarities $d(i,j)$ for all $j \in K$ such that $G_j = c$. The minimal contribution v_{2i} of the point i is the minimal added amount when considering all values in $Dom(G_i)$, with respect to the assigned points:

$$v_{2i} = \min_{c \in Dom(G_i)} \Big(\sum_{j \in K, G_j = c} d(i,j) \Big).$$

A lower bound $V_2.lb$ of V_2 can be computed by the sum of v_{2i}, for all $i \in U$:

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

The exact value of V_3 is unknown too and we use a heuristic to compute a lower bound of V_3. We recall that V_3 is the sum of all $d(i,j)$ such that $i, j \in U$ and i and j are in the same cluster. Let p be the cardinality of U and let k be the cardinality of the union $\cup_{i \in U} Dom(G_i)$. Each value of $\cup_{i \in U} Dom(G_i)$ is the index of a possible cluster to which the points in U can be assigned. The number k is then the maximal number of clusters to which the points in U can be assigned. We can see that the minimal number of terms $d(i,j)$ in the sum V_3 is the minimal number of within-cluster pairwise connections[5], while considering all partitions of p points into at most k clusters.

Let m be the quotient of the division of p by k and m' the remainder. Let $f(p,k) = (km^2 + 2mm' - km)/2$. It is proved in [48] that the total number of within-cluster pairwise connections for all clusters is greater or equal to $f(p,k)$. The equality is reached when m' clusters have $m+1$ elements and $k - m'$ clusters have m elements. Therefore, for the set U of unassigned points, if we order increasingly the constants $d(i,j)$ for all $i < j \in U$, a lower bound V_3, denoted by $V_3.lb$, is then computed by the sum of the $f(p,k)$ first constants in this order.

Example 5.1. Let us consider the case given in Figure 6 with 14 points, which have to be grouped into 2 clusters. Assume that 7 points are grouped and 7 are not. The exact value of V_1 is computed by the sum of solid black lines. The lower bound $V_2.lb$ is the sum of dash lines for each unassigned points. With 7 unassigned points, we have $p = 7, k = 2, m = 3$ and $m' = 1$, the minimal number of connections is $f(7,2) = 9$. The lower bound $V_3.lb$ the sum of dotted lines. Theses lines are the 9 smallest lines that connect two unassigned points. The lower bound $V_3.lb$ is heuristic since these lines do not correspond to any case where the 7 unassigned points are grouped into 2 groups.

[5] A cluster can be seen as a clique and the number of pairwise connections is the number of edges in the clique.

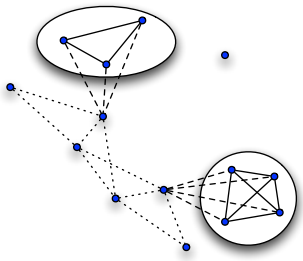

Figure 6: Example of $V.lb = V_1 + V_2.lb + V_3.lb$

Assume that the domain of variable V is $[V.lb, V.ub)$ where $V.lb$ is the lower bound, which can initially be 0, and $V.ub$ is the upper bound, which can be either $+\infty$ or the value of V in the previous solution found. The upper bound $V.ub$ is strict since in a branch-and-bound search the next solution must be better than the previous solution found. Given a partial assignment of variables in $\mathcal{G}$, a new lower bound of variable V is computed by:

$$V.lb = \max(V.lb,\ V_1 + V_2.lb + V_3.lb).$$

We use this lower bound in the filtering algorithm for $wcsd(\mathcal{G}, V, d)$. The algorithm is presented in Algorithm 3. The lower bound $V.lb$ is used for two purposes:

- Detecting failure during the branch-and-bound search. A failure happens when $V.lb \geq V.ub$, which means that the domain of V becomes empty.
- Filtering inconsistent values of unassigned variables. For each unassigned variable G_i, for each value $c \in Dom(G_i)$, under the assumption that $G_i = c$, we propose to revise the lower bound in a constant time. If the revised value is greater or equal than the upper bound $V.ub$ then c is inconsistent and is removed from $Dom(G_i)$.

Since the constants $d(i, j)$ $(i, j \in U,\ i < j)$ must be ordered increasingly for the computation of $V_3.lb$, they are ordered once in the array ord, and at the same time the arrays px and py are constructed. For each value pos, $ord[pos]$ gives the value $d(i, j)$ in this order at position pos, and $px[pos]$ (or $py[pos]$) gives the index i (or j, respectively) of the constant. The arrays ord, px and py are given in input of Algorithm 3. This algorithm computes arrays add and m, where $add[i, c]$ is the added amount if i is assigned to cluster number c ($add[i, c] = \sum_{j \in K, G_j = c} d(i, j)$) and $m[i]$ is the minimal added amount while considering all possible assignments for i ($m[i] = \min_{c \in Dom(G_i)} add[i, c]$).

Lines 1 to 25 computes the lower bound for V, based on a partial assignment of variables in $\mathcal{G}$. Lines 26 to 28 filter the domain of the uninstantiated variables G_i as follows. For each uninstantiated variable G_i, for each value $c \in Dom(G_i)$, in case of assignment of i into cluster number c, a new lower bound for $V.lb$ is revised into $V'.lb = V'_1 + V'_2.lb + V'_3.lb$, where:

- $V'_1 = V_1 + add[i, c]$ because point i is supposed to be assigned to cluster c, the sum of dissimilarities between instantiated points is increased by $add[i, c]$.
- $V'_2 = V_2.lb - m[i]$ because point i is no more unassigned, the contribution of point i in the computation of $V_2.lb$ must be removed.
- $V'_3.lb$ is the sum of the first $f(|U \setminus \{i\}|, k)$ elements that are related to $U \setminus \{i\}$ in the increasingly ordered array ord. In order to revise this bound in a constant time, we actually use V_4 instead of $V'_3.lb$. Here, V_4 is the sum of the first $f(|U| - 1, k) = f(p - 1, k)$ elements of ord. These elements are related to U, it this therefore possible that some of them are related to i. It is evidence that $V'_3.lb \geq V_4$. The value V_4 can be computed once and independently from i and c (line 23).

The revised lower bound, in case of assignment of point i to cluster c is:

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

Algorithm 3: Filtering algorithm for $wcsd(\mathcal{G}, V, d)$

input : U: the set of unassigned variables in $\mathcal{G}$;
ord: array in which all the dissimilarities $d(i, j)$ ($i, j \in U$, $i < j$) are ordered increasingly
px, py: arrays giving the index i, j wrt. ord
output: compute a new lower bound of V and filter unassigned variables in $\mathcal{G}$

$V_1 \leftarrow 0$; $V_2.lb \leftarrow 0$; $V_3.lb \leftarrow 0$; $V_4 \leftarrow 0$;
for $i \leftarrow 1$ **to** n *where* G_i *is unassigned* **do**
  for $c \leftarrow 1$ **to** k_{max} **do**
    if $c \in Dom(G_i)$ **then** $add[i, c] \leftarrow 0$ **else** $add[i, c] \leftarrow +\infty$
for $i \leftarrow 1$ **to** n *where* G_i *is assigned* **do**
  $c \leftarrow val(G_i)$
  for $j \leftarrow 1$ **to** n **do**
    if G_j *is assigned and* $G_j = c$ *and* $i < j$ **then** $V_1 \leftarrow V_1 + d(i, j)$
    if G_j *is unassigned and* $c \in Dom(G_j)$ **then** $add[j, c] \leftarrow add[j, c] + d(i, j)$
for $i \leftarrow 1$ **to** n *where* G_i *is unassigned* **do**
  $m[i] \leftarrow +\infty$;
  foreach *value* $c \in Dom(G_i)$ **do**
    if $m[i] > add[i, c]$ **then** $m[i] \leftarrow add[i, c]$
  $V_2.lb \leftarrow V_2.lb + m[i]$;
$p \leftarrow \text{card}(U)$;
$k \leftarrow \text{card}(\cup_{i \in U} Dom(G_i))$;
$cpt \leftarrow 0$; $pos \leftarrow 1$;
while $cpt < f(p, k)$ **do**
  $i \leftarrow px[pos]$; $j \leftarrow py[pos]$;
  if G_i *is unassigned and* G_j *is unassigned* **then**
    $cpt \leftarrow cpt + 1$;
    $V_3.lb \leftarrow V_3.lb + ord[pos]$;
    if $cpt \leq f(p - 1, k)$ **then** $V_4 \leftarrow V_4 + ord[pos]$
  $pos \leftarrow pos + 1$;
$V.lb \leftarrow \max(V.lb, V_1 + V_2.lb + V_3.lb)$;
for $i \leftarrow 1$ **to** n *where* G_i *is unassigned* **do**
  foreach *value* $c \in Dom(G_i)$ **do**
    if $V.lb + add[i, c] - m[i] - V_3.lb + V_4 \geq V.ub$ **then** delete c from $Dom(G_i)$;

which is greater or equal to:

$$V.lb + add[i, c] - m[i] - V3.lb + V_4.$$

So if this last value is greater than the actual upper bound of V, point i cannot be assigned to cluster c. The value c is therefore inconsistent and is removed from $Dom(G_i)$. The complexity of this algorithm is $O(n^2 + nk) = O(n^2 + nk_{max})$, since the domain of each G_i is of size at most k_{max}. Since $k_{max} \leq n$, the complexity is then $O(n^2)$.

6. Bicriterion Split-Diameter Constrained Clustering

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 ways of using our framework. We show in this section how it can be applied to handle bi-criterion constrained clustering tasks.

Let us consider a constrained clustering task with a set C of user constraints, which is possibly empty. We aim at computing the Pareto front for this constrained clustering task with the bi-criterion $(\min D, \max S)$. One approach to achieve this is described in Algorithm 4; it is comparable to the *ϵ-constraint approach* presented in [50]. In this algorithm, optimization steps with a single criterion are iterated, each time with a condition on the value of the other criterion. The function *Maximize_Split*(C) or *Minimize_Diameter*(C) means the use of our model with the optimization criterion of maximizing the split or minimizing the diameter, respectively, and with the set of constraints C. It returns an optimal solution which satisfies all the constraints in C, if there exists one, or $NULL$ otherwise. We prove that this algorithm computes a complete and minimal set of Pareto optimal solutions.

Algorithm 4: Algorithm computing a complete and minimal set $\mathcal{P} = \{\Delta_1^S, \ldots, \Delta_m^S\}$ of Pareto optimal solutions

```
1  Input: C, a set of user constraints
2  P ← ∅;
3  i ← 1;
4  Δ_i^D ← Minimize_Diameter(C);
5  while Δ_i^D ≠ NULL do
6  |  Δ_i^S ← Maximize_Split(C ∪ {D ≤ D(Δ_i^D)}) ;
7  |  P ← P ∪ {Δ_i^S};
8  |  i ← i + 1;
9  |  Δ_i^D ← Minimize_Diameter(C ∪ {S > S(Δ_{i-1}^S)}) ;
10 Return P
```

Proposition 6.1. *Let $\Delta_1^D, \Delta_1^S, \ldots, \Delta_m^D, \Delta_m^S$ be the partitions visited by Algorithm 4. We have:*

1. *there is no partition Δ satisfying C such that $D(\Delta) < D(\Delta_1^D)$,*
2. *if $\Delta_i^D \neq NULL$ then $\Delta_i^S \neq NULL$,*
3. *for all $2 \leq i \leq m$, $S(\Delta_i^S) > S(\Delta_{i-1}^S)$,*
4. *for all $1 \leq i \leq m$, $D(\Delta_i^S) = D(\Delta_i^D)$,*
5. *for all $2 \leq i \leq m$, $D(\Delta_i^D) > D(\Delta_{i-1}^D)$,*
6. *for all $1 \leq i < m$, there is no partition Δ satisfying C such that $S(\Delta) \geq S(\Delta_i^S)$ and $D(\Delta) < D(\Delta_{i+1}^D)$.*
7. *for all $1 \leq i \leq m$, there is no partition Δ satisfying C such that $S(\Delta) > S(\Delta_i^S)$ and $D(\Delta) = D(\Delta_i^S)$.*

Proof.

1. Since Δ_1^D is a partition that minimizes the diameter among all the partitions satisfying the user constraints C (line 4), there exists no partition Δ satisfying C with $D(\Delta) < D(\Delta_1^D)$.
2. If $\Delta_i^D \neq NULL$, since Δ_i^D satisfies the set C and the condition $D \leq D(\Delta_i^D)$, the set of partitions satisfying $C \cup \{D \leq D(\Delta_i^D)\}$ is not empty. Moreover, this set is finite since the set of all partitions is finite. There exists at least one partition satisfying these constraints and maximizing the split.

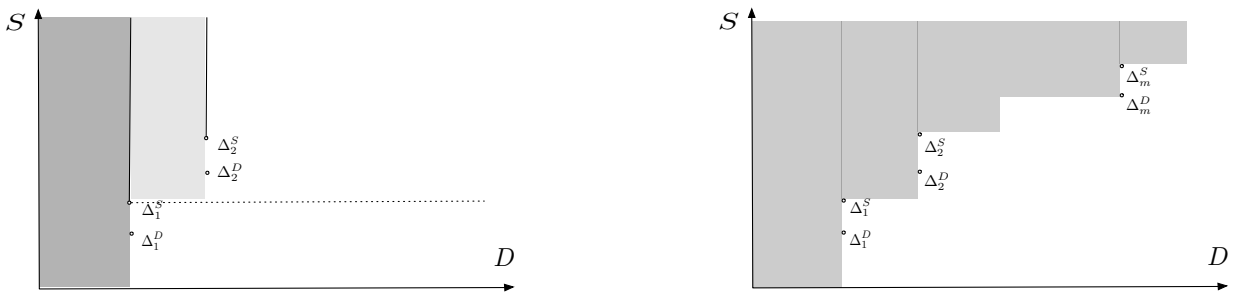

Figure 7: The solutions found by Algorithm 4: (left) by two first steps, (right) by whole algorithm.

3. Since Δ_i^D is among the partitions satisfying $C \cup \{S > S(\Delta_{i-1}^S)\}$ (line 9), we have $S(\Delta_i^D) > S(\Delta_{i-1}^S)$. Since Δ_i^D satisfies $C \cup \{D \leq D(\Delta_i^D)\}$ and since among all the partitions satisfying $C \cup \{D \leq D(\Delta_i^D)\}$, Δ_i^S is the one that maximizes the split (line 6), we have $S(\Delta_i^S) \geq S(\Delta_i^D)$. Therefore $S(\Delta_i^S) > S(\Delta_{i-1}^S)$.
4. To prove this we distinguish two cases.
   Case $i = 1$: Δ_1^S satisfies C and $D(\Delta_1^D)$ is the minimal diameter of the partitions satisfying the user-constraints C, therefore $D(\Delta_1^S) \geq D(\Delta_1^D)$. Δ_1^S belongs to the partitions satisfying $D \leq D(\Delta_1^D)$. So $D(\Delta_1^S) = D(\Delta_1^D)$.
   Case $i \geq 2$: Since Δ_i^S is a partition satisfying the set $C \cup \{D \leq D(\Delta_i^D)\}$ (line 6), we have $D(\Delta_i^S) \leq D(\Delta_i^D)$. As proven in the precedent item, $S(\Delta_i^S) > S(\Delta_{i-1}^S)$, so Δ_i^S satisfies $C \cup \{S > S(\Delta_{i-1}^S)\}$. Since Δ_i^D is a partition which minimizes the diameter among all the ones satisfying $C \cup \{S > S(\Delta_{i-1}^S)\}$ (line 9), we have $D(\Delta_i^S) \geq D(\Delta_i^D)$. Therefore $D(\Delta_i^S) = D(\Delta_i^D)$.
5. For $i = 2$, the set of partitions satisfying $C \cup \{S > S(\Delta_1^S)\}$ is a subset of the set of partitions satisfying C. Therefore $D(\Delta_2^D) \geq D(\Delta_1^D)$. For $i \geq 3$, the set of partitions satisfying $C \cup \{S > S(\Delta_{i-1}^S)\}$ is a subset of the set of partitions satisfying $C \cup \{S > S(\Delta_{i-2}^S)\}$, since $S(\Delta_{i-1}^S) > S(\Delta_{i-2}^S)$. 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 $D(\Delta_i^D) \geq D(\Delta_{i-1}^D)$.
   In all cases ($2 \leq i \leq m$), 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 6: the same constraints entail the same set of partitions and therefore the same maximal split) which contradicts $S(\Delta_i^S) > S(\Delta_{i-1}^S)$. Therefore $D(\Delta_i^D) > D(\Delta_{i-1}^D)$.
6. Assume that there exists a partition Δ that satisfies C and 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 satisfy the condition $S > S(\Delta_i^S)$, we have $D(\Delta) \geq D(\Delta_{i+1}^D)$. This contradicts the fact that $D(\Delta) < D(\Delta_{i+1}^D)$.
7. Assume that there exists a partition Δ satisfying C such that $S(\Delta) > S(\Delta_i^S)$ and $D(\Delta) = D(\Delta_i^S)$. By point 4, $D(\Delta) = D(\Delta_i^D)$, so Δ satisfies $C \cup \{D \leq D(\Delta_i^D)\}$. By line 6, Δ_i^S is a partition which maximizes the split among those satisfying $C \cup \{D \leq D(\Delta_i^D)\}$, so $S(\Delta) \leq S(\Delta_i^S)$. This contradicts the fact that $S(\Delta) > S(\Delta_i^S)$. □

Figure 7 illustrates the positions of the solutions found by Algorithm 4, according to Proposition 6.1. The left image presents the two first steps. By point 1, there is no partition in the dark zone, except on the line $D = D(\Delta_1^D)$. By point 7, there is no partition in the segment $S > S(\Delta_1^S)$ of the line $D = D(\Delta_1^D)$. By point 5 and line 9 of Algorithm 4, the partition Δ_2^D is above the dotted line (partitions in the white zone below the dotted line are dominated by Δ_1^S). By point 6, there is no partition in the grey zone, except on the line $D = D(\Delta_2^D)$. By point 7, there is no partition in the segment $S > S(\Delta_2^S)$ of the line $D = D(\Delta_2^D)$. The right image presents the positions found by the whole algorithm. There exists no partition in the grey zone. A solution in the white zone is dominated by a solution $\Delta_i^S \in \mathcal{P}$.

Proposition 6.2. *The set $\mathcal{P} = \{\Delta_1^S, \ldots, \Delta_m^S\}$ computed by Algorithm 4 is complete and minimal i.e.:*

1. *Δ_i^S ($1 \leq i \leq m$) is a Pareto optimal solution,*
2. *for all Pareto optimal solution Δ satisfying C, there exists $i \in [1, m]$ such that $D(\Delta) = D(\Delta_i^S)$ and $S(\Delta) = S(\Delta_i^S)$.*

The set $\{(D(\Delta), S(\Delta)) \mid \Delta \in \mathcal{P}\}$ is therefore the Pareto front.

Proof. Algorithm 4 terminates since according to Proposition 6.1, $S(\Delta_i^S) > S(\Delta_{i-1}^S)$ and these values are discrete and limited by the maximal dissimilarity of pairs of points.

1. We prove that for all $i \in [1, m]$, there exists no partition Δ satisfying C which dominates Δ_i^S, *i.e.* such that $D(\Delta) \leq D(\Delta_i^S)$ and $S(\Delta) > S(\Delta_i^S)$, or $D(\Delta) < D(\Delta_i^S)$ and $S(\Delta) \geq S(\Delta_i^S)$. Since $D(\Delta_i^S) = D(\Delta_i^D)$, the partition Δ must satisfy $D(\Delta) \leq D(\Delta_i^D)$ and $S(\Delta) > S(\Delta_i^S)$, or $D(\Delta) < D(\Delta_i^D)$ and $S(\Delta) \geq S(\Delta_i^S)$. The first case is impossible since Δ_i^S is a partition which maximizes the split among all those satisfying the condition $D \leq D(\Delta_i^D)$. For the second case, if $i = 1$ then Δ does not exist according to point 1 of Proposition 6.1. 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 point 6 of Proposition 6.1. Therefore each partition Δ_i^S is Pareto optimal.
2. Let Δ be any Pareto optimal solution, *i.e.* Δ is not dominated, satisfying C.
   We cannot have $S(\Delta) > S(\Delta_m^S)$, since Δ_{m+1}^D is null (Algorithm 4 terminates) and therefore there exists no partitions satisfying C and $S(\Delta) > S(\Delta_m^S)$. Therefore $S(\Delta) \leq S(\Delta_m^S)$. Since $\{S(\Delta_i^S)\}$ is strictly increasing, therefore either $S(\Delta) \leq S(\Delta_1^S)$ or there exists $i \in [1, m-1]$ such that $S(\Delta_i^S) < S(\Delta) \leq S(\Delta_{i+1}^S)$.
   Considering the case where $S(\Delta) \leq S(\Delta_1^S)$. By point 1 of Proposition 6.1, $D(\Delta_1^D) \leq D(\Delta)$ and by point 4, $D(\Delta_1^D) = D(\Delta_1^S)$, therefore $D(\Delta_1^S) \leq D(\Delta)$. We have either $(D(\Delta), S(\Delta)) = (D(\Delta_1^S), S(\Delta_1^S))$, or Δ_1^S dominates Δ. Since Δ is not dominated, we must have $D(\Delta) = D(\Delta_1^S)$ and $S(\Delta) = S(\Delta_1^S)$.
   In the other case, where there exists $i \in [1, m-1]$ such that $S(\Delta_i^S) < S(\Delta) \leq S(\Delta_{i+1}^S)$. By point 6 of Proposition 6.1, we have $D(\Delta) \geq D(\Delta_{i+1}^D)$ and by point 4, $D(\Delta) \geq D(\Delta_{i+1}^S)$. We then have $S(\Delta_{i+1}^S) \geq S(\Delta)$ and $D(\Delta_{i+1}^S) \leq D(\Delta)$, therefore either $(D(\Delta), S(\Delta)) = (D(\Delta_{i+1}^S), S(\Delta_{i+1}^S))$, or Δ_i^S dominates Δ. Since Δ is not dominated, we must have $D(\Delta) = D(\Delta_{i+1}^S)$ and $S(\Delta) = S(\Delta_{i+1}^S)$. □

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

Multi-objective optimization in Constraint Programming in only one phase of search is proposed in [51]. 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 as well as an extension with Large Neighborhood Search is proposed in [52]. This constraint *Pareto* can be introduced in our model. Nevertheless this approach for the moment is still much less efficient than Algorithm 4 and therefore deeper studies, as for instance a study of the search strategy, are needed in order to improve the efficiency.

7. Experiments

Our model is implemented in Gecode version 4.2.1[6]. Gecode [19] is an open source Constraint Programming library in C++ and is one of the current state-of-the-art CP solvers. Twelve databases taken from the repository UCI [53] are used in our experiments. They vary on their size and their number of classes. Table 2 summarizes information on these datasets, which are presented in increasing order of the number of objects. Since the problem we address is finding an exact solution for distance-based clustering, the important factors are the number n of objects and the number k of clusters. For the experiments, we have chosen a wide range of datasets with different values of n and k. The experiments are performed on a 3.4GHz Intel Core i5 processor with 8G Ram running Ubuntu. All our programs are available at `http://cp4clustering.com`.

[6] http://www.gecode.org

Dataset	# Objects	# Classes
Iris	150	3
Wine	178	3
Glass	214	7
Ionosphere	351	2
User Knowledge	403	4
Breast Cancer	569	2
Synthetic Control	600	6
Vehicle	846	4
Yeast	1484	10
Multiple Features	2000	10
Image Segmentation	2000	7
Waveform	5000	3

Table 2: Properties of datasets

Dataset	D_{opt}	BaB	GC	CP1	CP2
Iris	2.58	1.4	1.8	< 0.1	< 0.1
Wine	458.13	2	2.3	0.3	< 0.1
Glass	4.97	8.1	42	0.9	0.2
IonoSphere	8.6	–	0.6	0.4[8]	0.3
User Knowledge	1.17	–	3.7	75	0.2
Breast Cancer	2377.96	–	1.8	0.7	0.5
Synthetic Control	109.36	–	–	56.1	1.6
Vehicle	264.83	–	–	14.3	0.9
Yeast	0.67	–	–	2389.9	5.2
Multi Features	12505.5	–	–	$*$	10.4
Image Segmentation	436.4	–	–	589.2	5.7
Waveform	15.6	–	–	$*$	50.1

Table 3: Performance (measured in seconds) with the criterion of minimizing the maximal diameter

7.1. Constrained clustering with a single criterion

7.1.1. Minimizing the maximal diameter of clusters

Performance test. We compare the performance of our previous model (denoted by CP1) and our new model (denoted by CP2), both relying on Gecode solver, with the branch-and-bound approach [4] (denoted by BaB) and the algorithm based on graph coloring [5] (denoted by GC). The program BaB has been obtained from the author's website [7]. Since no implementation of the algorithm GC was available, we coded it ourselves in C++ using a well-known available graph coloring program [54]. We consider clustering without user-constraints since the other algorithms cannot handle them and to our knowledge, there is no exact algorithm handling user-constraints with this criterion. In the experiments, the timeout is set to 1 hour and the Euclidean distance is used to compute the dissimilarity between objects. The number of classes k is set to the number of real classes given in Table 2 (in the new model $k_{min} = k_{max} = k$).

Table 3 shows the results of experiments. For each dataset we present the value D_{opt} (the optimal diameter) in the second column and the run-time in seconds of each system. The symbol – is used when the system cannot complete the search after 1 hour and the symbol $*$ is used to mark that the computer runs out of memory and cannot finish the search. All the algorithms are exact and they find the same value for the optimum diameter.

It is clear that with these datasets, our new model (CP2) is the most efficient in all cases. All twelve datasets can be solved by CP2 within one minute. Among the programs, BaB algorithm is the least efficient. It is not able to solve

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

[8] There was an error in [3] that expressed 8.6s for solving the dataset IonoSphere

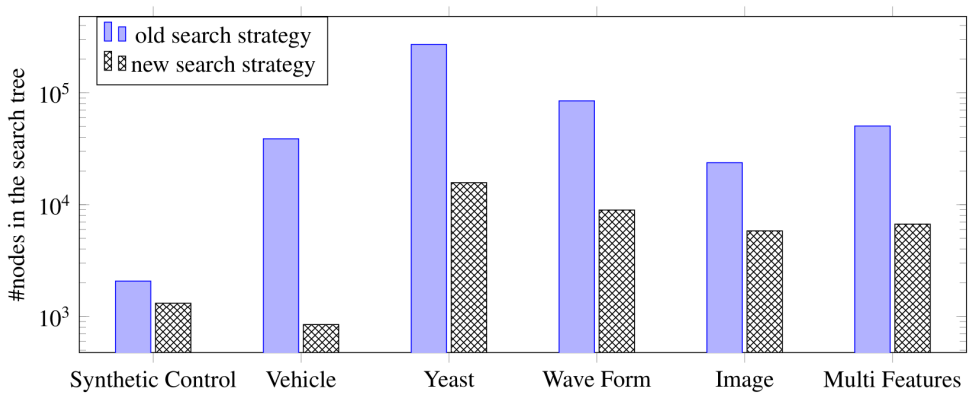

Figure 8: Comparison of #nodes in the search tree with different search strategies

datasets with more than 300 objects. The performance of GC is better than that of BaB but it decreases rapidly when the number n of objects is over 500. BaB algorithm is based on the bounds of the maximal diameter to detect failures during search, while GC algorithm considers all available distances in decreasing order to find the optimum diameter. Our models, which exploits the benefits of Constraint Programming such as constraint propagation and appropriate search strategies, are more efficient.

Analysis of search strategy. Although our two models are based on the Constraint Programming framework, there are two main reasons that explain the significant differences in performance of the two models: the search strategy and the dedicated filtering algorithm. For analyzing the influence of the search strategy, we use our new model (CP2) with different search strategies: the strategy used in the previous model (CP1) and the one used in the new model (CP2). Figure 8 presents the number of nodes in the search trees with these two strategies for the last six datasets given in Table 2. It is clear that the new strategy is better, since the search trees are always smaller. The new search strategy always finds quickly the first solution, the maximal diameter of which is close to the optimal diameter. As a result, the solver can remove more unnecessary branches and there are less number of nodes in the search tree.

Analysis of the dedicated filtering algorithm. In CP1, for modeling the diameter criterion, the number of reified constraints is a square function of the number n of points. Although our dedicated global constraint 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 is checked at least one time. In order to study the efficiency of the filtering algorithm, we test our new model (CP2) but using reified constraints (as in CP1) to express the diameter criterion. Table 4 presents the performances obtained when using reified constraints and when using the dedicated filtering algorithm. We can see that when using reified constraints, the solver cannot find optimal solution with the datasets Wave Form and Multi Features. The reason is that there are too many reified constraints and the computer runs out of memory. Table 4 shows that the filtering algorithm boosts the performance and this becomes more and more significant with larger datasets.

Analysis of bounds on the number of clusters of the new model. We evaluate the influence of the bounds on the number of clusters ($k \in [k_{min}, k_{max}]$) on the performance of CP2. In the first experiment, k_{max} is set to 10 and k_{min} varies from 2 to 10. The diameter criterion favors a high number of clusters, since the higher the number of clusters, the smaller the value of the optimal diameter. Without user-constraints, the optimal solution with the diameter criterion has always a number of clusters equal to k_{max}. For all the datasets, the total time in the search tree is constant when k_{min} changes. It shows that the propagation of the constraint modeling the diameter criterion is effective. After finding and proving the optimal solution with k_{max} clusters, the solver can conclude that it does not exist a better solution with less than k_{max} clusters.

In the second experiment, k_{min} is set to 2 and k_{max} varies from 2 to 10. Figure 9 presents the results obtained with the datasets Vehicle, Yeast, Multi Features and Image Segmentation. For each dataset, we report the number of nodes in the search tree, when the bound k_{max} varies. In general, when k_{max} increases, more partitions have to be considered.

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

Table 4: Performance (measured in seconds) of CP2 with different modeling of diameter criterion

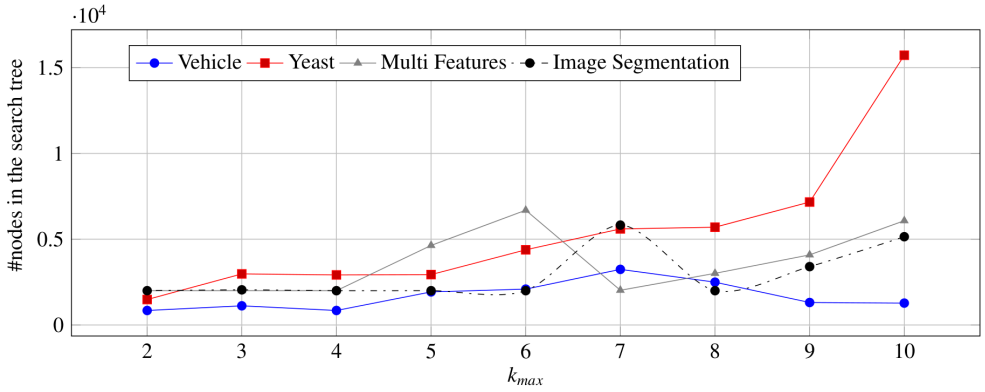

Figure 9: Analysis of bounds: $k_{min} = 2$, $k_{max} = 2, 3, \ldots, 10$

However, we see an interesting trend in Figure 9: the number of nodes in the search tree does not always increase as k_{max} increases. Indeed, since k_{max} is higher, the optimal maximum diameter is smaller, and the propagation of the diameter constraint is more effective. It explains why in some cases, the computation time decreases when k_{max} increases.

7.1.2. Maximizing the minimal split between clusters

Finding a partition maximizing the split between clusters is a polynomial problem. However, with user constraints the problem becomes NP-Hard. To our knowledge, there is no exact algorithm for this criterion that supports any kind of user constraints for a general value $k \geq 3$. When optimizing the split without user constraints, when the number k of classes is not fixed ($k_{min} \leq k \leq k_{max}$), the optimal solution has always a number of classes equal to k_{min}. However, this is no longer true with user constraints, as for instance with a diameter constraint. We have experimented this point with the new model by adding a diameter constraint. In order to set it, we have used the results given in Table 3 and set an upper bound on the diameter of each cluster to $1.5D_{opt}$. The number of classes k is not fixed, it is bounded between $k_{min} = 2$ and k_{max} equal to the actual number of classes given in Table 2. The results are given in Table 5. For each dataset, we present the optimal split S_{opt}, the number of classes of the solution k_{sol}, the actual number of classes k_{real} and the total execution time in seconds. Our model is able to solve almost all data sets, with the exception for the dataset Waveform, which is the largest dataset.

Dataset	S_{opt}	k_{sol}	k_{real}	Total time
Iris	0.53	3	3	< 0.1
Wine	53.33	3	3	< 0.1
Glass	1.78	7	7	1.7
Ionosphere	5.29	2	2	2.4
User Knowledge Modeling	0.32	4	4	11.5
Breast Cancer	421.99	2	2	0.7
Synthetic Control	45.16	5	6	17.4
Vehicle	27.06	4	4	29.5
Yeast	0.15	10	10	639.8
Multi Features	1107.07	10	10	267.9
Image Segmentation	228.70	7	7	51.3
Waveform	–	–	–	–

Table 5: Maximizing the minimal split between clusters with a diameter constraint

7.1.3. Minimizing the within-cluster sum of dissimilarities (WCSD)

Finding an exact solution for minimizing WCSD is difficult and we compare the performance of our new model in Gecode solver with the Repetitive Branch-and-Bound Algorithm (RBBA) [4]. The program RBBA has been obtained from the author's website (`http://mailer.fsu.edu/~mbrusco/`). To our knowledge, it is the best exact algorithm for the WCSD criterion. The dissimilarity between objects is measured by the squared Euclidean distance. Without user constraints, both our model and the RBBA approach can only find the optimal solution with the Iris dataset. Our model needs 4125s to complete the search whereas RBBA takes 3249s. RBBA solves the problem by repetitively solving sub-problems: finding the optimal solution with $k+1, k+2, \ldots, n$ objects. By using the optimal value of WCSD computed in the sub-problems, a better lower bound of WCSD can be computed, enabling RBBA to have better performance. However, extending this algorithm to integrate user-constraints is difficult.

Our model can handle different kinds of user constraints and appropriate combinations of user constraints can boost the performance. A set of 120 instance-level constraints has been generated from the dataset Iris. The constraints were generated following the method described in [7]: two points are chosen randomly from the dataset, if they belong to the same cluster in the real partition, a must-link constraint is generated, otherwise a cannot-link constraint is generated. The first test is without user-constraints, the second one considers the first 30 constraints, the third one takes into account the first 60 constraints and so on. Figure 10 (left) reports the total time needed to solve the dataset with these user-constraints. When there are 30 constraints, the solver takes more computation time. The reason is that, with user-constraints, the optimal value of WCSD is higher and the propagation of the WCSD constraint is weaker. However, when more user-constraints are integrated, the propagation of must-link and cannot-link constraints is stronger and enables to quickly instantiate variables in $\mathcal{G}$. As a result, the solver takes only 94s for solving the problem with 60 constraints, and less than 10s when there are 90 or more constraints.

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

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

where a is the number of pairs of points that are in the same cluster in P and in P', b is the number of pairs of points that are in different clusters in P and in P', c is the number of pairs of points that are in the same cluster in P, but in different clusters in P' and d is the number of pairs of points that are in different clusters in P, but in the same cluster in P'. The results of this experiment are reported in Figure 10 (right). Since the constraints are generated from the true partition of the dataset, the ARI value of the optimal partition improves when more are more constraints are considered.

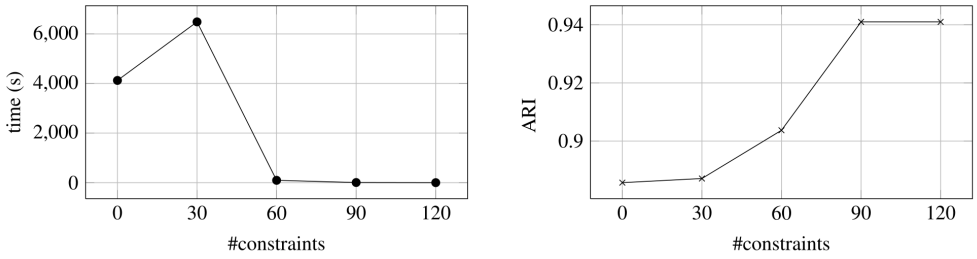

Figure 10: WCSD with user-constraints on Iris: (left) computation time, (right) Adjusted Rand Index

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

Table 6: Comparison of performance (measured in seconds) with bi-criterion Split-Diameter

7.2. Clustering with bi-criterion Split-Diameter

7.2.1. Performance test

For the split-diameter bi-criterion, we compare our new model (CP2) with the bi-criterion clustering algorithm based on graph coloring [5] (denoted bGC). Since no implementation of the program was available, we have coded it in C++. To our knowledge, this is the only exact algorithm for $k \in [k_{min}, k_{max}]$. In the experiments, the datasets in Table 2 were used and the timeout was set to 1 hour. The number of classes k varies between 2 and the real number of classes. Table 6 gives the results of our experiments. The second column (#Sol) gives the number of Pareto optimal solutions found, or equivalently, the number of elements in the complete Pareto front. The following columns give the run time of each approach in seconds. The two programs are exact and they find the same Pareto front. It is clear that our model is the most efficient in most cases. It takes advantage of the efficient constraint propagation mechanism to reduce the search space. As in the case of GC, the algorithm bGC is limited to datasets with less than 500 points.

7.2.2. Bi-criterion clustering with user-constraints

We have generated a set of 80 instance-level constraints from the dataset Iris, as described in Subsection 7.1.3 and we have applied Algorithm 4 with CP2 for solving the task of bi-criterion constrained clustering. The first test is without user-constraints, the second one is with the first 20 user-constraints, the third one with the first 40 constraints and so on. Figure 11 presents the Pareto front for the five cases, using from 0 to 80 user-constraints. As more and more user-constraints are added, the number of feasible solutions decreases and as a result, the criterion space changes significantly. Since we have generated user-constraints from the real partition, it is obvious that the point (D_r, S_r) corresponding to the real partition must be in the region delimited by each Pareto front. Therefore it must be in the region delimited by the Pareto front with 80 constraints. We can see that without user-constraints, there are many points in the Pareto front but all of them are very far from (D_r, S_r). For that reason, it is useful to enable user-constraints for the task of bi-criterion clustering. Moreover, given an element (D_i, S_i) in the Pareto front, our model can be used to enumerate all Pareto optimal solutions that have the maximum diameter D_i and the minimum

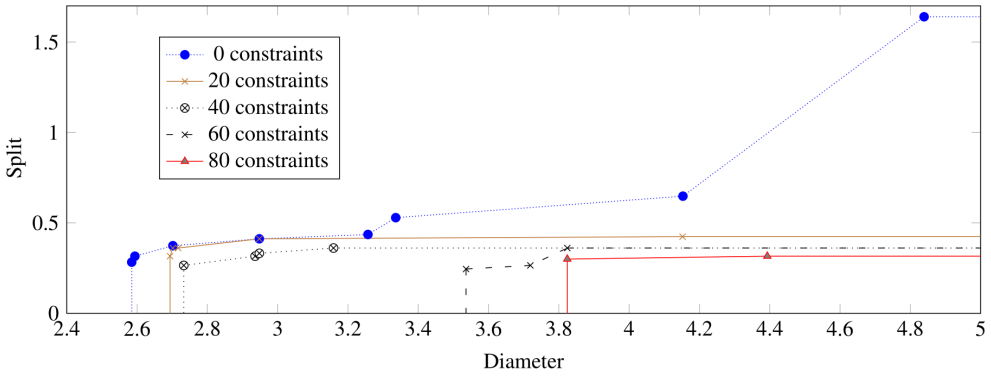

Figure 11: Bi-criterion constrained clustering with dataset Iris

split S_i. For example, considering the Pareto front in the case of 80 instance-level constraints, it is composed of two points, which correspond respectively to 8704 and 4352 partitions.

8. Conclusion

We have presented a new Constraint Programming model for Constrained Clustering. This model is a significantly improved version of our previous model within the Constraint Programming framework [3]. It is based on a different choice of variables and constraints. This model is modular in the sense that dedicated global constraints are developed for different optimization criteria. It is more flexible since no fixed number of clusters needs to be specified; it is sufficient to provide upper and lower bounds on the number of clusters. It is declarative and general since it allows to choose among different optimization criteria and to integrate various kinds of user-constraints. We show that thanks to its properties, it can be directly integrated in more general processes, as for instance for handling bi-criterion constrained clustering. Experiments on classical datasets show that our model outperforms existing exact approaches in most cases.

We do believe that working on search strategies and on constraint propagation enables to improve substantially the efficiency of the model. We continue studying these aspects to make the model able to deal with larger datasets. We also investigate the use of approximate search strategies, such as local search methods.

The use of our model for bi-criterion split-diameter constrained clustering can be generalized to other bi-criterion problems, the only requirement is that each single criterion must be integrated inside the model.

From the Data Mining point of view, we consider integrating other optimization criteria, as for instance the within-cluster sum of squares.

Acknowledgement

We gratefully thank the anonymous referees for their helpful comments to improve this article. This work is supported by a Doctoral grant from the French Ministry of National Education, Higher Education and Research.

References

[1] I. Davidson, S. S. Ravi, L. Shamis, A SAT-based Framework for Efficient Constrained Clustering, in: Proceedings of the 10th SIAM International Conference on Data Mining, 2010, pp. 94–105.

[2] M. Mueller, S. Kramer, Integer Linear Programming Models for Constrained Clustering, in: Proceedings of the 13th International Conference on Discovery Science, 2010, pp. 159–173.

[3] T.-B.-H. Dao, K.-C. Duong, C. 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, 2013, pp. 419–434.

[4] M. Brusco, S. Stahl, Branch-and-Bound Applications in Combinatorial Data Analysis (Statistics and Computing), 1st Edition, Springer, 2005.

[5] M. Delattre, P. Hansen, Bicriterion Cluster Analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence (4) (1980) 277–291.

[6] T. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoretical Computer Science 38 (1985) 293–306.

[7] K. Wagstaff, C. Cardie, Clustering with instance-level constraints, in: Proceedings of the 17th International Conference on Machine Learning, 2000, pp. 1103–1110.
[8] I. Davidson, S. S. Ravi, Clustering with Constraints: Feasibility Issues and the k-Means Algorithm, in: Proceedings of the 5th SIAM International Conference on Data Mining, 2005, pp. 138–149.
[9] M. Ester, H. P. Kriegel, J. Sander, X. 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, 1996, pp. 226–231.
[10] R. Cormack, A review of classification, Journal of the Royal Statistical Society. Series A (General) 134 (3) (1971) 321–367.
[11] S. Johnson, Hierarchical clustering schemes, Psychometrika 32 (3) (1967) 241–254.
[12] Y. Wang, H. Yan, C. Sriskandarajah, The weighted sum of split and diameter clustering, Journal of Classification 13 (2) (1996) 231–248.
[13] J. Wang, J. Chen, Clustering to maximize the ratio of split to diameter, in: Proceedings of the 29th International Conference on Machine Learning, 2012.
[14] N. Beldiceanu, M. Carlsson, J.-X. Rampon, Global constraint catalog, SICS and EMN Technical Report, `http://sofdem.github.io/gccat/`.
[15] J.-C. Régin, A Filtering Algorithm for Constraints of Difference in CSPs, in: Proceedings of the 12th National Conference on Artificial Intelligence (Vol. 1), 1994, pp. 362–367.
[16] F. Focacci, A. Lodi, M. Milano, Cost-based domain filtering, in: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming, 1999, pp. 189–203.
[17] J.-C. Régin, Arc consistency for global cardinality constraints with costs, in: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming, 1999, pp. 390–404.
[18] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Foundations of Artificial Intelligence, Elsevier B.V., Amsterdam, Netherlands, 2006.
[19] Gecode Team, `http://www.gecode.org/`.
[20] I. Davidson, S. S. Ravi, The Complexity of Non-hierarchical Clustering with Instance and Cluster Level Constraints, Data Mining Knowledge Discovery 14 (1) (2007) 25–61.
[21] P. Hansen, M. Delattre, Complete-link cluster analysis by graph coloring, Journal of the American Statistical Association 73 (362) (1978) 397–403.
[22] S. Basu, I. Davidson, K. Wagstaff, Constrained Clustering: Advances in Algorithms, Theory, and Applications, 1st Edition, Chapman & Hall/CRC, 2008.
[23] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, Constrained K-means Clustering with Background Knowledge, in: Proceedings of the 18th International Conference on Machine Learning, 2001, pp. 577–584.
[24] M. Bilenko, S. Basu, R. J. Mooney, Integrating constraints and metric learning in semi-supervised clustering, in: Proceedings of the 21st International Conference on Machine Learning, 2004, pp. 11–18.
[25] I. Davidson, 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, 2005, pp. 59–70.
[26] Z. Lu, M. A. Carreira-Perpinan, Constrained spectral clustering through affinity propagation, in: Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition, 2008, pp. 1–8.
[27] X. Wang, I. Davidson, Flexible constrained spectral clustering, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010, pp. 563–572.
[28] T. Guns, S. Nijssen, L. De Raedt, k-Pattern set mining under constraints, IEEE Transactions on Knowledge and Data Engineering 25 (2) (2013) 402–418.
[29] J.-P. Métivier, P. Boizumault, B. Crémilleux, M. Khiari, S. Loudni, Constrained Clustering Using SAT, in: Proceedings of the 11th International Symposium on Advances in Intelligent Data Analysis, 2012, pp. 207–218.
[30] B. Babaki, T. Guns, S. 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, 2014, pp. 438–454.
[31] D. Aloise, P. Hansen, L. Liberti, An improved column generation algorithm for minimum sum-of-squares clustering, Mathematical Programming 131 (1-2) (2012) 195–220.
[32] L. Kotthoff, B. O'Sullivan, Constraint-based clustering, in: Proceedings of the 10th International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) techniques in Constraint Programming, 2013, presentation-only abstract.
[33] U. Luxburg, A tutorial on spectral clustering, Statistics and Computing 17 (4) (2007) 395–416.
[34] X. Wang, B. Qian, I. Davidson, On constrained spectral clustering and its applications, Data Mining and Knowledge Discovery 28 (1) (2014) 1–30.
[35] W. Zhi, X. Wang, B. Qian, P. Butler, N. Ramakrishnan, I. Davidson, Clustering with complex constraints - algorithms and applications, in: Proceedings of the 27th AAAI Conference on Artificial Intelligence, 2013.
[36] I. Davidson, B. Qian, X. Wang, J. Ye, Multi-objective multi-view spectral clustering via pareto optimization, in: Proceedings of the 13th SIAM International Conference on Data Mining, 2013, pp. 234–242.
[37] S. Gilpin, S. Nijssen, I. N. Davidson, Formalizing hierarchical clustering as integer linear programming, in: Proceedings of the 27th AAAI Conference on Artificial Intelligence, 2013, pp. 372–378.
[38] S. Gilpin, I. N. Davidson, Incorporating SAT solvers into hierarchical clustering algorithms: an efficient and flexible approach, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2011, pp. 1136–1144.
[39] J. Berg, M. Jarvisalo, Optimal Correlation Clustering via MaxSAT, in: Proceedings of the 13th IEEE International Conference on Data Mining Workshops, 2013, pp. 750–757.
[40] L. De Raedt, T. Guns, S. Nijssen, Constraint programming for itemset mining, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 204–212.
[41] L. De Raedt, T. Guns, S. Nijssen, Constraint Programming for Data Mining and Machine Learning, in: Proc. of the 24th AAAI Conference on Artificial Intelligence, 2010.

[42] H. Cambazard, T. Hadzic, B. O'Sullivan, Knowledge compilation for itemset mining, in: Proceedings of the 19th European Conference on Artificial Intelligence, 2010, pp. 1109–1110.
[43] T. Guns, S. Nijssen, L. De Raedt, Itemset mining: A constraint programming perspective, Artificial Intelligence 175 (2011) 1951–1983.
[44] S. Jabbour, L. Sais, Y. Salhi, The top-k frequent closed itemset mining using top-k SAT problem, in: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases, 2013, pp. 403–418.
[45] W. U. Rojas, P. Boizumault, S. Loudni, B. Crémilleux, A. Lepailleur, Mining (soft-) skypatterns using dynamic CSP, in: Proceedings of the 11th International Conference on Integration of AI and OR Techniques in Constraint Programming, 2014, pp. 71–87.
[46] C. Bessiere, E. Hebrard, B. O'Sullivan, Minimising decision tree size as combinatorial optimisation, in: Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming, 2009, pp. 173–187.
[47] Y. C. Law, J. H.-M. Lee, Global constraints for integer and set value precedence., in: M. Wallace (Ed.), Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming, 2004, pp. 362–376.
[48] T.-B.-H. Dao, K.-C. Duong, C. 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, 2013, pp. 1060–1067.
[49] G. Klein, J. E. Aronson, Optimal clustering: A model and method, Naval Research Logistics 38 (3) (1991) 447–461.
[50] V. T'kindt, J.-C. Billaut, Multicriteria Scheduling, Theory, Models and Algorithms, 2nd Edition, Springer, 2006.
[51] M. Gavanelli, An Algorithm for Multi-Criteria Optimization in CSPs, in: F. van Harmelen (Ed.), Proceedings of the 15th European Conference on Artificial Intelligence, 2002, pp. 136–140.
[52] P. Schaus, R. Hartert, Multi-Objective Large Neighborhood Search, in: Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming, 2013, pp. 611–627.
[53] K. Bache, M. Lichman, UCI Machine Learning Repository, `http://archive.ics.uci.edu/ml`.
[54] A. Mehrotra, M. A. Trick, A column generation approach for graph coloring, INFORMS Journal on Computing 8 (1995) 344–354.