



LIFO - Bâtiment 3IA
Rue Léonard de Vinci
BP 6759
45067 Orléans Cedex 2, France

Rapport de Recherche

A Constraint Programming Approach for Constrained Clustering

Thi-Bich-Hanh Dao, Khanh-Chuong Duong, Christel Vrain
LIFO, Université d'Orléans

Rapport n° **RR-2013-03**

A Constraint Programming Approach for Constrained Clustering

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

Abstract

Clustering is an important task in Data Mining and many algorithms have been designed for it. It has been extended to semi-supervised clustering, so as to integrate some previous knowledge on objects that must be or cannot be in the same cluster, and many classical algorithms have been extended to handle such constraints. Other kinds of constraints could be specified by the user, as for instance the sizes of the clusters or their diameters, but adding such constraints generally requires to develop new algorithms. We propose a declarative and generic framework, based on Constraint Programming, which enables to design a clustering task by specifying an optimization criterion and some constraints either on the clusters or on pairs of objects. In our framework, several classical optimization criteria are considered and they can be coupled with different kinds of constraints. Relying on Constraint Programming has two main advantages: the declarativity, which enables to easily add new constraints and the ability to find an optimal solution satisfying all the constraints (when there exists one). On the other hand, computation time depends on the constraints and on their ability to reduce the domain of variables, thus avoiding an exhaustive search.

1 Introduction and Motivation

Clustering is an important task in Data Mining and many algorithms have been designed for it. It has been extended to semi-supervised clustering, so as to integrate previous knowledge on objects that must be or cannot be in the same cluster, and most algorithms have been adapted to handle such information. Other kinds of constraints could be specified by the user, as for instance the sizes of the clusters or their diameters, but classical frameworks are not designed to integrate different types of knowledge. Yet, in the context of an exploratory process, it would be important to be able to express constraints on the task at hand, tuning the model for getting finer solutions. Constrained clustering aims at integrating constraints in the clustering process, but the algorithms are usually developed for handling one kind of constraints. Developing general solvers with the ability of handling different kinds of constraints is therefore of high importance for Data Mining. We propose a declarative and generic framework, based on Constraint Programming, which enables to design a clustering task by specifying an optimization criterion and some constraints either on the clusters or on pairs of objects.

Relying on Constraint Programming (CP) has two main advantages: the declarativity, which enables to easily add new constraints and the ability to find an optimal solution satisfying all the constraints (when there exists one). Recent progress in CP have made this paradigm more powerful and several work [11, 10, 5] have already shown its interest for Data Mining.

In this paper, we propose a CP model for constrained clustering, aiming at finding a partition of data that optimizes a criterion. We generalize the model with different optimization criteria, namely minimizing the maximal diameter of clusters, maximizing the margin between clusters and minimizing the Within-Cluster Sums of Dissimilarities (WCSD). Clustering with WCSD criterion is NP-Hard, since one of the instance of this problem is the weighted max-cut problem, which is NP-Complete [13]. Recent work [6] has addressed the problem of finding an exact optimum but the size of the database must be quite small, all the more when k is high. One of the key constraints of our model is a combination of a sum constraint with reified equality constraints

$V = \sum_{1 \leq i < j \leq n} (G[i] == G[j])a_{ij}$, with V and the $G[i]$ variables and a_{ij} constants. We have developed a filtering algorithm for this constraint for the WCSD problem, and experiments show that we are able of finding the optimal solution for small to medium databases. Moreover, adding constraints allows to reduce the computation time.

There are two main contributions in our paper: a general framework for Constrained Clustering which is integrated with different kinds of optimization criteria and a filtering algorithm for the sum constraint $V = \sum_{1 \leq i < j \leq n} (G[i] == G[j])a_{ij}$.

In our model, the number k of classes is fixed (in theory no bound is given on k but the larger k , the higher the complexity). The model takes as input a matrix of distances between objects, and thus can deal with qualitative and quantitative databases. We show that our framework can be directly extended to instance-level and cluster-level constraints.

The paper is organized as follows. We present preliminary notions on constraint-based clustering in Section 2 and those on CP in Section 3. Related works are discussed in Section 4. Section 5 is devoted to the presentation of our model and Section 6 to experiments. Conclusion and discussion on future works are given in Section 7.

2 Preliminaries

2.1 Definition of Clustering

Clustering is the process of grouping data into classes or clusters, so that objects within a cluster have high similarity but are very dissimilar to objects in other clusters. More formally, we consider a database of n objects $\mathcal{O} = \{o_1, \dots, o_n\}$ and a dissimilarity measure $d(o_i, o_j)$ between two objects o_i and o_j of \mathcal{O} . The objects are usually described by attributes, which may be qualitative or quantitative attributes. In case of quantitative attributes, the Euclidean measure is classically used. It is defined by

$$d(o_i, o_j) = \sqrt{\sum_{t=1}^m (x_{it} - x_{jt})^2} \quad (1)$$

where $o_i = (x_{i1}, x_{i2}, \dots, x_{im})$ and $o_j = (x_{j1}, x_{j2}, \dots, x_{jm})$ are two m -dimensional objects. In some cases, the system is only provided with a matrix given the dissimilarity between pairs of objects.

Clustering is often seen as an optimization problem, i.e., finding a partition of the objects, optimizing a given criterion. There are many variations of the clustering problem. Several points of view can be used to classify clustering methods, such as the form of the learned structure (partition, hierarchy, overlapping or not, ...), or the criterion to optimize. In this paper we are interested in partitioning methods that learn a partition of the set of objects. It means finding a set of k classes C_1, \dots, C_k such that: (1) for all $c \in [1, k]$, $C_c \neq \emptyset$, (2) $\cup_c C_c = \mathcal{O}$, (3) for all c, c' such that $c \neq c'$, $C_c \cap C_{c'} = \emptyset$, and (4) a criterion E is optimized. Optimized criteria may be, among others:

- Within-Cluster Sum of Squares (WCSS) criterion:

$$E = \sum_{c=1}^k \sum_{o_i \in C_c} d(m_c, o_i)^2$$

where m_c is the center of cluster C_c .

- Within-Cluster Sums of Dissimilarities (WCS D) criterion

$$E = \sum_{c=1}^k \sum_{o_i, o_j \in C_c} d(o_i, o_j)^2$$

where o_i, o_j are objects in the cluster C_c . When Euclidean distance is used, this criterion once standardized via the division by the size of each group, $\sum_{c=1}^k (\sum_{i,j \in C_c} d(i,j)^2) / |C_c|$, is equivalent to the Within-Cluster Sum of Square criterion.

- Absolute-error criterion:

$$E = \sum_{c=1}^k \sum_{o_i \in C_c} d(o_i, r_c)$$

where r_c is a representative object of the cluster C_c .

- Diameter-based criterion:

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

E is the maximum diameter of the clusters, where the diameter of a cluster is the maximum distance between any two of its objects.

- Margin-based criterion

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

E is the minimal margin between clusters, where the minimal margin between 2 clusters $C_c, C_{c'}$ is the minimum value of the distances $d(o_i, o_j)$, with $o_i \in C_c$ and $o_j \in C_{c'}$.

Clustering task with these criteria is NP-Hard, well-known algorithms such as k-means, k-medoids use heuristics and usually find a local optimum.

2.2 Partitioning methods

In this section, we present most common partitioning methods. Most of partitioning methods are heuristic, they are iterative relocation clustering algorithms. The quality of the solutions is measured by the clustering criterion. At each iteration, the iterative relocation algorithms reduce the value of the criterion function until convergence. The convergence is local and the globally optimal solution cannot be guaranteed. The problem of local minimal could be avoided by using exhaustive search methods. However, finding the globally optimal partition is known to be a NP-hard problem.

2.2.1 k-means

The k-means algorithm is a simple iterative method to partition a dataset into k clusters. The algorithm is initialized by picking k objects from \mathcal{O} as the k initial cluster representatives or centroids. Then each object is assigned to the closest centroid, forming a partition of the dataset. Next, each representative is relocated by the mean value of all objects assigned to it. Using the new cluster representatives, objects are assigned again. This process iterates until there is no change of assignments of objects, or equivalently, until there is no change of centroids. The k-means algorithm always converges to a solution where the centroids are stable.

The algorithm aims at minimizing the WCSS. In each iteration of the algorithm, the WCSS reduces and converges to a local optimum. This method is efficient if clusters are separated from each other. Now we will show how the update of cluster centroids minimizes the WCSS, specialized

for one-dimensional data. It is shown in [25] that: the k^{th} centroid of the cluster m_k minimizes the WCSS, by setting the differentiating of the WCSS to 0 and solving:

$$\frac{\partial}{\partial m_k} WCSS = \frac{\partial}{\partial m_k} \sum_{c=1}^k \sum_{o_i \in C_c} d(m_c, o_i)^2 \quad (2)$$

$$= \frac{\partial}{\partial m_k} \sum_{c=1}^k \sum_{o_i \in C_c} (m_c - o_i)^2 \quad (3)$$

$$= \sum_{c=1}^k \sum_{o_i \in C_c} \frac{\partial}{\partial m_k} (m_c - o_i)^2 \quad (4)$$

$$= \sum_{o_k \in C_k} 2 * (m_k - o_k) = 0 \quad (5)$$

$$\sum_{o_k \in C_k} 2 * (m_k - o_k) = 0 \Rightarrow |c_k| m_k = \sum_{o_k \in C_k} o_k \Rightarrow m_k = \frac{1}{|c_k|} \sum_{o_k \in C_k} \quad (6)$$

So, the best centroid for minimizing the WCSS of a cluster is the mean of the points in the cluster.

Algorithm 1: k-Means algorithm

- 1 Generate randomly k objects as initial cluster centers;
 - 2 **repeat**
 - 3 Assign each object to the cluster with the closest center;
 - 4 Re-calculate the new center of each cluster as the mean value of the objects in each cluster;
 - 5 **until** *no change of cluster center*;
-

The k-means is efficient if clusters are separated from each other. It can handle large dataset because the complexity of the algorithm is $O(nkt)$ where t is the number of iterations. However, this method can be applied only when the mean value of clusters can be computed. Moreover, this method is sensible to noises and outliers because those values affect the mean value of clusters.

2.2.2 k-medoids

In this method, an actual object is chosen in each cluster as a representative object. This method uses the absolute-error criterion and it differs from k-means where the representative is computed as the mean value of the objects. It starts by choosing k objects as the initial cluster representative. Each object is assigned to its nearest representative object. It then chooses new representative object for each cluster allowing to minimizing the absolute error criterion. With the new representative, the method reassigned the objects. The algorithm iterates until each representative object is actually the medoid, or most centrally located object of its cluster. It is more robust to noise and outliers as compared to k-means. However, both k-medoids and k-means search for an optimal local, it means that the quality of the partitioning depends on the initial state.

Algorithm 2: k-medoids algorithm for partitioning

- 1 Randomly generate k objects as initial cluster representatives;
 - 2 **repeat**
 - 3 Assign each object to the cluster with the closest representative;
 - 4 Randomly select a non representative object o_i ;
 - 5 Compute the cost S when swap the object o_i with a representative o_j ;
 - 6 **if** $S < 0$ **then**
 - 7 | swap o_i with o_j , o_i is now a new representative ;
 - 8 **until** *no change of cluster representatives*;
-

2.2.3 FPF

The method FPF (Furthest Point First) introduced in [14] uses the maximum diameter criterion. In this method, each cluster has an object as the head and every objects in a cluster are closer to their head than other heads. The method starts by choosing an object as the head of the first cluster and assigns all objects to it. In the next iteration, the furthest object from the first head is chosen as the head of the second cluster. Any object which is closer to the second head than the first one will be moved to the second cluster. The algorithm iterates: choosing the object that is furthest to its head as the head of the new cluster and reassigning objects. It stops after k iterations since k clusters are needed. The time complexity is $O(kn)$. Let d_{opt} be the global optimal, then this heuristic is guaranteed to find a clustering with a maximum diameter d such that $d_{opt} \leq d \leq 2d_{opt}$ if the distance satisfies the triangle inequality. Moreover, Gonzalez showed that this approximation ratio is tight: finding d such that $d \leq (2 - \epsilon)d_{opt}$ is NP-hard for all $\epsilon > 0$ when objects are in a three dimensional space.

Algorithm 3: FPF algorithm for partitioning

- 1 Randomly select an object, mark it as $head_1$, the head of cluster 1;
 - 2 **for** $i \leftarrow 2$ **to** k **do**
 - 3 Select the object that is furthest to its head, mark it as $head_i$;
 - 4 Assign every object o to cluster i if o is closer to $head_i$ than to its head;
-

2.2.4 DBSCAN

Most partitioning methods find spherical-shaped clusters. The method DBSCAN [12] is based on the notion of density, which allows to discovery arbitrary shaped clusters. In this method, the number of k is not given, but additional parameters defining the density must be specified. DBSCAN requires two input parameters $MinPts$ and Eps and its complexity is $O(n^2)$. It discovers clusters with arbitrary shape but with the same density. This approach is based on the following notions.

- The Eps neighborhood of a point p is defined by:

$$N_{Eps}(p) = \{q \in \mathcal{O} \mid dist(p, q) \leq Eps\}$$

- Core point: It has at least $MinPts$ points within a radius Eps . A point p is a core point iff $|N_{Eps}(p)| \geq MinPts$.
- Border point: It doesn't have at least $MinPts$ points within a radius Eps but it is the neighborhood of one or more core points. A point p is a border point iff $|N_{Eps}(p)| < MinPts$ and $\exists q : p \in N_{Eps}(q), |N_{Eps}(q)| \geq MinPts$.
- Noises: Point which is neither a core point nor border point.
- Directly Density Reachable: A point p is directly density reachable from a point q if q is a core point and p is the neighborhood of q :

$$p \in N_{Eps}(q), |N_{Eps}(q)| \geq MinPts$$

- Density Reachable: A point p is density reachable from a point q if there is a chain of point p_1, \dots, p_n , such that $p_1 = q, p_n = p$ and p_{i+1} is directly density reachable from p_i .
- Density connected: A point p is density connected to a point q if there is a point o such that both p and q are density reachable from o .

The method DBSCAN starts with an arbitrary point p and it retrieves the neighborhood of p to verify if p is a core point or not. If p do not have at least $MinPts$ within a radius Eps , it is

marked as a noise (it can be remarked as a border point later). Otherwise, p is a core point and a new cluster is formed from p and its neighborhoods. DBSCAN iteratively expand the new cluster by retrieving all points which are density-reachable from p . After that, the algorithm visits the next unvisited point. DBSCAN stops when all the points are visited.

Algorithm 4: DBSCAN algorithm

```

1  $C \leftarrow 0$ ;
2 for each unvisited point  $p$  in dataset  $\mathcal{O}$  do
3   Mark  $p$  as visited;
4    $NeighborPts \leftarrow regionQuery(p, Eps)$ ;
5   if  $sizeof(NeighborPts) < MinPts$  then
6     Mark  $p$  as NOISE;
7   else
8      $C \leftarrow C + 1$ ;
9      $expandCluster(p, NeighborPts, C, Eps, MinPts)$ ;

```

Function $expandCluster(p, NeighborPts, cluster, Eps, MinPts)$

```

1 Add  $p$  to cluster  $C$ ;
2 for each point  $p'$  in  $NeighborPts$  do
3   if  $p'$  is not yet member of any cluster then
4     add  $p'$  to cluster  $C$ ;
5    $NeighborPts' = regionQuery(p', eps)$ ;
6   if  $sizeof(NeighborPts') \geq MinPts$  then
7     add  $NeighborPts'$  to  $NeighborPts$ 

```

Function $regionQuery$

```

1 return all points within the radius  $Eps$  from  $p$  ;

```

2.3 Constraint-based Clustering

Most clustering methods rely on an optimization criterion, and because of the inherent complexity search for a local optimum. Several optima may exist, some may be closer to the one expected by the user. In order to better model the task, but also in the hope of reducing the complexity, constraints can be added, leading to Constraint-based Clustering that aims at finding clusters that satisfy user-specified constraints. They can be classified into cluster-level constraints, specifying requirements on the clusters as for instance, the number of elements, or instance-level constraints, specifying that two elements must be or cannot be in the same cluster.

Most of the attention has been put on instance-level constraints, first introduced in [26]. Commonly, two kinds of constraints are used: must-link and cannot-link.

- A must-link constraint specifies that two objects o_i and o_j have to appear in the same cluster:

$$\forall c \in [1, k], o_i \in C_c \Leftrightarrow o_j \in C_c.$$

- A cannot-link constraint specifies that two objects must not be put 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 at least a minimum number α of objects:

$$\forall c \in [1, k], |C_c| \geq \alpha$$

whereas the maximum capacity constraint requires each cluster to have at most a maximum number β of objects:

$$\forall c \in [1, k], |C_c| \leq \beta$$

- The maximum diameter constraint specifies the maximum diameter γ of clusters:

$$\forall c \in [1, k], \forall o_i, o_j \in C_c, d(o_i, o_j) \leq \gamma$$

- The minimum margin 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 :

$$\forall c \in [1, k], \forall c' \neq c, \forall o_i \in C_c, o_j \in C_{c'}, d(o_i, o_j) \geq \delta$$

- The ϵ -constraint introduced in [8] requires for each point o_i to have in its neighborhood of radius ϵ at least another point of the same cluster:

$$\forall c \in [1, k], \forall o_i \in C_c, \exists o_j \in C_c, o_j \neq o_i \wedge d(o_i, o_j) \leq \epsilon$$

Such a constraint tries to capture the notion of density, introduced in DBSCAN. We propose a new density-based constraint, stronger than the ϵ -constraint: it requires that for each point o_i , its neighborhood with radius ϵ contains at least *MinPts* 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, as for instance an extension of COBWEB [26], of k-means [27, 4], hierarchical non supervised clustering [7] or spectral clustering [17, 28], etc. This is achieved either by modifying the dissimilarity measure, or the objective function or the search strategy. However, to the best of our knowledge there is no general solution to extend traditional algorithms to different types of constraints. Our framework relying on Constraint Programming allows to add directly user-specified constraints.

3 Constraint Programming

Constraint Programming (CP) is a powerful paradigm for solving combinatorial search problems that relies on a wide range of techniques from Artificial Intelligence, Computer Science and Operational Research. The main principles in CP are: (1) users specify the problem declaratively in a Constraint Satisfaction Problem; (2) solvers search for solutions by constraint propagation and search. A Constraint Satisfaction Problem (CSP) is a triple $\langle X, D, C \rangle$ where:

- $X = \{x_1, x_2, \dots, x_n\}$ is a set of variables,
- $D = \{D_1, D_2, \dots, D_n\}$ is a set of domains, with $x_i \in D_i$,
- $C = \{C_1, C_2, \dots, C_t\}$ is a set of constraints, each C_i is a condition on a subset of X .

A solution of the CSP is a complete assignment of a value $a_i \in D_i$ to each variable x_i , such that all constraints in C are satisfied. A Constraint Optimization Problem (COP) is a CSP associated to an objective function. In this case, the optimal solution is a solution of the CSP that optimizes the objective function.

In general, CSPs are NP-hard. However, techniques used in CP solvers allow to solve many practical applications efficiently. Most popular techniques are constraint propagation, branching and search.

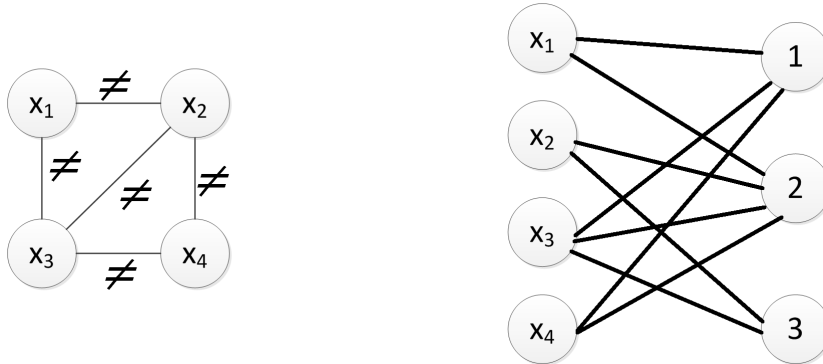


Figure 1: constraint graph and value graph of exemple 2

3.1 Constraint Propagation

Constraint propagation consists of removing values from the domain of some variables when it can be determined that the values cannot be in any solution of a constraint. Propagation is done by propagators: each constraint is realized by a set of propagators, depending on which kind of consistency the constraint is considered. If propagators implement the arc-consistency for a constraint, they can remove any inconsistent values of the domain with respect to the constraint. If they implement the bound-consistency for a constraint, they only modify the upper or the lower bound of the variable domains.

The objective of the consistency techniques is to detect values of variables that can not be in the solution with the help of constraints. The consistency techniques try to detect inconsistent assignments as soon as possible to prune the search space.

Example 1 Assume we have 2 variables x_1, x_2 taking integer values in $[1, 4]$ with the constraint: $x_1 < x_2 - 1$. Consistency techniques can reduce the domain of x_1 and x_2 without loss of any solution, $D_1 = \{1, 2\}$ and $D_2 = \{3, 4\}$.

We present in this section basic consistency algorithms for unary and binary constraints. Normally, a CSP can be represented by a constraint graph where variables are nodes and constraints corresponds to hyper-arcs. To express domain of variables, a value graph can be used. It is a bipartite graph whose vertices are divided into two disjoint sets: a set of vertices from variables and a set of vertices from values. Every edge in the value graph connect a vertice of variable to a vertice of value.

Example 2 Assume we have 4 variables x_1, x_2, x_3 and x_4 and their domains are: $x_1 \in [1, 2], x_2 \in [2, 3], x_3 \in [1, 3], x_4 \in [1, 2]$ with the constraints: $x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3, x_2 \neq x_4, x_3 \neq x_4$. Constraint graph and value graph of this CSP are expressed in figure 1. The consistency techniques try to detect inconsistent assignments, or they try to remove as many edges as possible in the value graph. The value graph of this CSP after constraint propagation is expressed in figure 2

Node Consistency Node Consistency is the simplest technique. A variable x_i is node consistency iff for every value of the current domain D_i , each unary constraint on x_i is satisfied. A CSP is node consistent iff all variables x_i are node consistent.

Example 3 Assume the variable x_1 taking integer values in $[1, 10]$ with the unary constraint: $x_1 > 3$. By applying node consistency algorithm, we can delete values $[1, 3]$ from D_1 . D_1 is now $[4, 10]$.

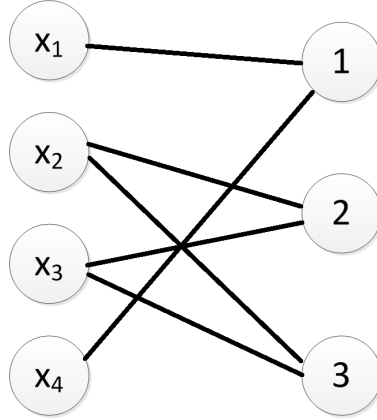


Figure 2: value graph after constraint propagation

Algorithm 5: Node consistency algorithm

```

1 forall the variable  $x_i$  do
2   forall the value  $a_i$  in  $D_i$  do
3     if unary constraint on  $X$  is inconsistent with  $a_i$  then
4       delete  $a_i$  from  $D_i$ ;
  
```

Arc Consistency Arc Consistency is used to guarantee the consistency of each binary constraint. An arc (x_i, x_j) is arc consistent iff for every value a_i in the current domain of D_i there are some values a_j of D_j so that (a_i, a_j) satisfy the binary constraint between x_i and x_j . A CSP is arc consistent iff every arc (x_i, x_j) is arc consistent.

Algorithm 6: Arc Consistency-1 algorithm

```

1 repeat
2   CHANGED  $\leftarrow$  FALSE ;
3   forall the arc  $(x_i, x_j)$  do
4     CHANGED  $\leftarrow$  Revise( $x_i, x_j$ )  $\vee$  CHANGED;
5 until not CHANGED;
  
```

Function Revise(x_i, x_j)

```

1 DELETED  $\leftarrow$  false forall the value  $a_i$  in  $D_i$  do
2   if there is no  $a_j$  in  $D_j$  such that  $(a_i, a_j)$  satisfies all binary constraints on  $(x_i, x_j)$  then
3     delete  $a_i$  in  $D_i$ ;
4     DELETED  $\leftarrow$  true;
5 return DELETED
  
```

In this basic arc consistency algorithm, all the arcs will be revised again if any inconsistency is detected and removed. Many other arc consistency algorithms (AC-2, ..., AC-7) therefore proposed to reduce the number of revises.

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

- we first select the arc (x_1, x_2) and remove the value 4 from D_1 because if $x_1 = 4$, we cannot find a value for x_2 such that $x_1 < x_2$

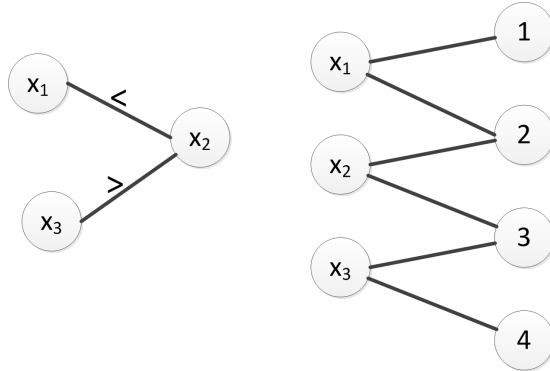


Figure 3: constraint graph and value graph of exemple 4

- *arc* (x_2, x_1) is revised and the value 1 is removed from D_2 for the same reason.
- *arc* (x_2, x_3) is revised and the value 4 is removed from D_2 . Now $D_1 = [1, 3]$, $D_2 = [2, 3]$, $D_3 = [1, 4]$
- *arc* (x_3, x_2) is revised and the value 1 and 2 is removed from D_3 . Now $D_1 = [1, 3]$, $D_2 = [2, 3]$, $D_3 = [3, 4]$
- all the arcs are revised again: when revising (x_1, x_2) , the value 3 is removed from D_1
- Finally $D_1 = [1, 2]$, $D_2 = [2, 3]$, $D_3 = [3, 4]$

Constraint graph and value graph of this CSP after constraint propagation are expressed in figure 3.

The Arc Consistency techniques reduce significant by the search space in many CSPs but there still exists many other possible inconsistencies.

Example 5 Assume the variables x_1 , x_2 and x_3 taking integer values in $[1, 2]$ with constraints: $x_1 \neq x_2$, $x_2 \neq x_3$ and $x_3 \neq x_1$. The Arc Consistency techniques can not detect any inconsistencies in this case, where there is no solution.

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

Global Constraint A global Constraint is a constraint stated on a set of variables. It is equivalent to a conjunction of elementary constraints. It is used to simplify the modeling of a problem and more importantly, to exploit the semantics of constraint for efficient propagation. Its consistency is normally stronger than the combination of Arc Consistency on elementary constraints and it is often performed by an filtering algorithm based on graph theory or operations research techniques. From the literature in constraint programming, there are more than 350 global constraints issued¹. Some common global constraints are :

- The *alldifferent* constraint constrains values of variables to be pairwise different. A solution of the *alldifferent* constraint is equivalent to a maximum matching in a bipartite graph. Filtering algorithm for the *alldifferent* constraint can use matching theory to prune any values that are not in any maximum matching.

¹A catalog of global constraints can be found at <http://www.emn.fr/z-info/sdemasse/gccat/>

- The global cardinality constraint (*gcc*) restricts the number of times a values can be appeared in the solution. A solution to *gcc* constraint is equivalent to a particular network flow and filtering algorithm for this constraint uses the residual network.
- The knapsack constraint corresponds to the classical knapsack problem. This constraint is defined on 0-1 variables where x_i represents the belonging of item i to the knapsack. This problem is NP-Hard but a weak filtering algorithm for this constraint can be obtained with a low complexity.

3.2 Search and Strategies

Solvers search for solutions by recursively combining two steps: constraint propagation and branching. The solver propagates all the constraints until reaching a stable state, where either the domain of a variable is reduced to the empty set or no variable domain can be reduced. In the first case, there is no solution and the solver backtracks. In the other case, if all the variable domains are singleton, a solution is reached, otherwise the solver chooses a variable whose domain is non-singleton and splits the domain into two or more parts, which creates two or more branches in the search tree. The solver explores then each branch, where constraint propagation becomes reactive again, because of the modifications of a variable domain.

The search tree exploration strategy can be defined by the user. In a depth-first search strategy, the solver orders the branches in the order specified by the user and explores in depth each branch to find solutions. In an optimization problem, a branch-and-bound strategy is obtained from a depth-first search: each time the solver reaches a solution of the CSP, the value of the objective function on the solution is calculated and a new constraint is added that forbids all solutions which are not better than this one.

Strategies of choosing variables and of choosing values are extremely important, since they can reduce drastically the search tree. For more details, we refer to the text book on Constraint Programming by [24].

3.2.1 Variable Ordering

The ordering in which variables are instantiated can affect the search space of backtrack search. The underlying idea is to detect the failure early to avoid useless branchings. Since all variables have to be instantiated, variable is chosen in order to maximally reduce the search space or add constraints. Therefore different heuristics may be used during the search to determine the next variable to be instantiated.

- The variable with the smallest domain is selected for instantiation.
- The variable that participated in a most number of constraints is selected.
- The variable that has the largest number of constraints with instantiated variables.

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

Example 6 Find distinct digits for the letters S, E, N, D, M, O, R, Y such that

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

In this problem, we can define a variable with the domain $[0, 9]$ for each letter with following constraints:

- $AllDifferent(S, E, N, D, M, O, R, Y)$
- $S \neq 0, M \neq 0$

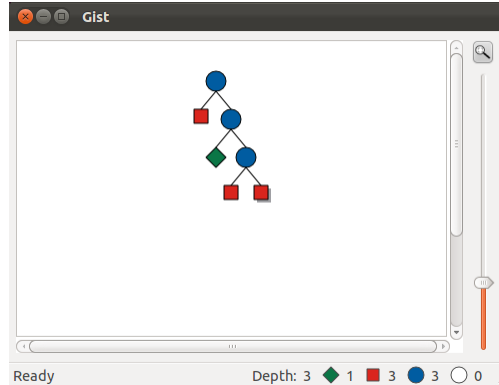


Figure 4: Search space with the strategy: The variable with the smallest domain is selected for instantiation

- $1000S + 100E + 10N + D + 1000M + 100O + 10R + E = 10000M + 1000O + 100N + 10E + Y$

By using consistency techniques, we can reduce the domain of variables to: $D_S = \{9\}$, $D_E = [4, 7]$, $D_N = [5, 8]$, $D_D = [2, 8]$, $D_M = \{1\}$, $D_O = \{0\}$, $D_R = [2, 8]$, $D_Y = [2, 8]$. Figures 4 and 5 show different search trees with different strategies for the next variable to be instantiated. These search trees are generated by Gist environment of the Gecode solver where 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 intermediate solution and the orange diamond is the optimal solution.

3.2.2 Branching strategy

In the backtracking algorithm, a node $p = \{x_1 = a_1, \dots, x_j = a_j\}$ in the search tree is a set of assignments of instantiated variables. A branching extends p by selecting a variable x_i that is not instantiated and adding the branches for assignments of x_i . Three popular branching strategies are often used:

- Enumeration: A branch is generated for each value in the domain of the variable.
- Binary choice points: The variable x_i is instantiated by a 2-way branching. Assuming the value a_i is chosen for branch, two branches are generated with $x_i = a_i$ and $x_i \neq a_i$.
- Domain splitting: Here the domain of the variable is split to create the branching. For example, assume the value a_i is chosen for the split, two branches are generated with $x_i \leq a_i$ and $x_i > a_i$.

3.2.3 Branch-and-bound technique

To solve optimization CSPs, the common approach is to use branch-and-bound techniques. Initially, a backtracking search is used to find any solution p which satisfies all the constraints. An additional constraint is added to the CSP when a solution is found allowing to forbid solutions that are not better than this solution. The solvers continue to search for solutions of the new CSP until the CSP is unsatisfiable. The last solution found has been proven optimal. Moreover, heuristic can be used to prune more the search space. The heuristic here is a function h that calculate the bound from values of instantiated variables. The efficiency of this technique is affected how quickly we can improve the bound:

- Whether a good solution which is close to the solution optimal is found early: When a solution is found, the lower bound (in case the CSP consists of minimizing a function) is updated and a good bound can be obtained from a good solution.

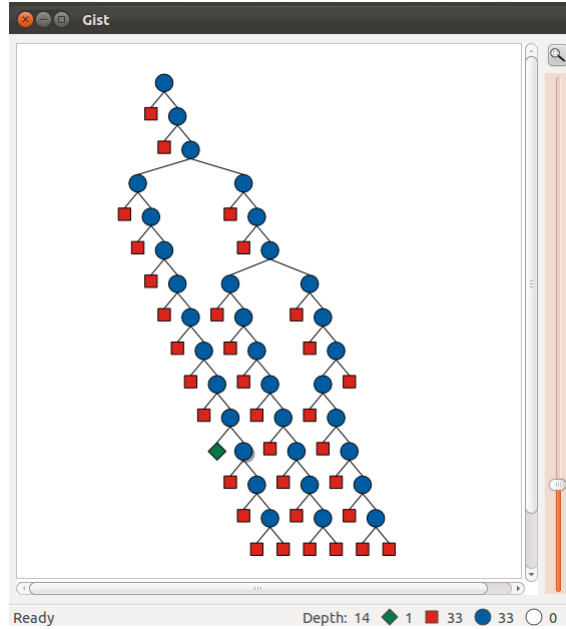


Figure 5: Search space with the strategy: The variable with the biggest domain is selected for instantiation

- Quality of the heuristic function: A better heuristic can calculate better lower/upper bounds from less number of instantiated variables.

Let us consider the following example for an illustration of Constraint Optimization Problem and search strategies.

Example 7 We have to find an assignment of digits to letters such that

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

and where the value of MONEY is maximized. We can model this problem as a Constraint Optimization Problem, where we use eight variables S, E, N, D, M, O, T, Y , whose domain is the set of digits $[0, 9]$. Constraints which specify the problem are:

- the letters S and M must not be 0: $S \neq 0, M \neq 0$
- all the letters are pairwise different: $\text{alldifferent}(S, E, N, D, M, O, T, Y)$
(note that instead of using the constraint \neq on each pair of variables, we use this very one constraint, which is stated on all variables, this constraint is called a "global constraint" in CP, the same kind as the following linear constraint)
- $(1000S + 100E + 10N + D) + (1000M + 100O + 10S + T) = 10000M + 1000O + 100N + 10E + Y$
- $\text{maximize}(1000M + 1000O + 100N + 10E + Y)$

The optimal solution of this problem is the assignment $S = 9, E = 7, N = 8, D = 2, M = 1, O = 0, T = 4, Y = 6$, with $\text{MONEY} = 10876$. Initial propagation from these constraints 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 $D_D = D_T = D_Y = \{2, 3, 4, 5, 6, 7, 8\}$. Strategies consist of the way to chose variables and for each chosen variable, the way to chose values. If variables are chosen in the

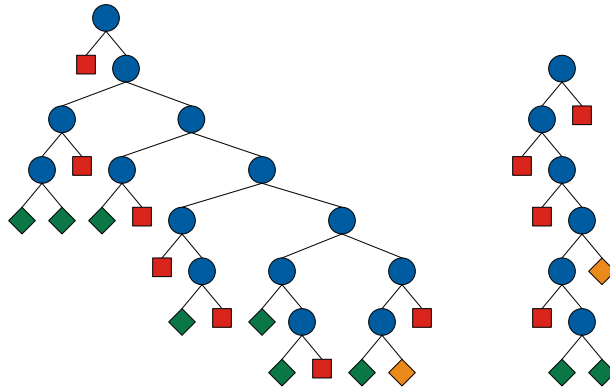


Figure 6: Search trees

order S, E, N, D, M, O, T, Y and for each variable, the remaining values in its domain are chosen in an increasing order, the search tree has 29 states with 7 intermediate solutions (solution which is better than the precedent). However, if variables are chosen in the order S, T, Y, N, D, E, M, O , the search tree has only 13 states with 2 intermediate solutions. These two search trees are showed in Figure 6. For each stable state, the left branch is the case where the chosen variable is assigned to the chosen value, the right branch is the other case where the chosen value is removed from the domain of the chosen variable. It is worth to notice that it is an exhaustive search, the returned solution is guaranteed to be optimal.

4 Related works

Recent advances in CP make the framework powerful to solve data mining problems. Several works aims at modeling data mining problems, and particularly clustering problems in a CP framework. L. De Raedt, T. Guns and S. Nijssen present in [11] a CP framework for modeling the itemset mining problem. They present in [16] a framework for k -patterns set mining and show how it can be applied to the conceptual clustering. The problem consists of finding k -patterns that do not overlap and cover all the transactions: each transaction can be viewed as an object and each pattern can be viewed as a cluster, two objects are similar if they share the same itemset. Conceptual clustering generates a concept description for each cluster and here, the concept is the itemset. For modeling the problem, boolean variables are used to model the presence of each item and each transaction in each pattern. Constraints and reified constraints express the relationship of covering, non-overlapping, etc. Additionnal constraints can be added to express the criterion function: maximize the minimum size of cluster or minimize the difference between cluster sizes.

P. Boizumault, B. Crémilleux *et al.* present in [5] and [18] a constraint-based language expressing queries to discover patterns in data mining. Queries are expressed by constraints over terms that are built from constraints, variables, operators and function symbols. The constraint conceptual clustering problem can be expressed in this language. The language is translated into a set of clauses and a SAT solver is used to solve the clustering task.

These works consider the conceptual clustering problem, where clusters are non-overlapping partitions on qualitative databases. Our approach considers the clustering problem in its original form and is capable to proceed on qualitative or quantitative databases.

A SAT-based framework for constraint-based clustering has been proposed by I. Davidson, S. Ravi and L. Shamis in [9]. The problem is however limited to only two clusters and is then transformed to a 2-SAT problem, which is polynomial. In this work, both instance-level constraints and cluster-level constraints are modeled. The model also allows users to add criterion function based on the diameter or the separation of clusters. The optimization algorithm is based on the

binary search for the criterion value, at each step the upper-bound or the lower-bound of this value is modified, and a 2-SAT solver is used to determine if all the constraints are satisfied with these bounds. Our approach based on constraint programming includes all the search and the constraint satisfaction, and is more general since the number of clusters is not limited to 2.

Mueller et al. proposed in [19] an approach to constrained clustering based on integer linear programming. This approach takes a set of candidate clusters as input and construct a clustering by selecting a suitable subset. It allows constraints on the degree of completeness of a clustering, on the overlap of clusters and it supports set-level constraints which restrict the combinations of admissible clusters. This approach is different as it takes into account constraints on candidate clusters, but not constraints on individual objects. It has different function objectives: optimizing the minimum, the mean and the median of the individual clusters' qualities in a clustering. Their framework is flexible and guarantees a global optimum but requires a set of candidate clusters. This condition makes the framework to be less convenient for clustering in general as finding a good set of candidate clusters is a difficult task and the number of candidate clusters is exponential compared to the number of points. However, this approach is more suitable for the itemset classified clustering or conceptual clustering where objects are itemsets and candidate clusters might be created from frequent itemsets.

Because of the difficulty of the problems, most algorithms are heuristics. For instance, k-means is a heuristic algorithm finding a local optimum for the WCSS criterion. There are few exact algorithms for the WCSD and WCSS criteria, they rely on lower bounds, which must be computed in a reasonable time and this is a difficult subtask. The best known exact method for both WCSD and the maximum diameter criterion is a repetitive branch-and-bound algorithm (denoted by RBBA in the experiment section) proposed in [6]. This algorithm is efficient when the number k of groups is small; it solves the problem first with $k + 1$ objects, then with $k + 2$ objects and so on, until all n objects are considered. When solving large problems, smaller problems are solved for quickly calculating good lower bounds. The authors give the size n of the databases that can be handled: $n = 250$ for the minimum diameter criterion, $n = 220$ for the WCSD criterion, and only $n = 50$ with k up to 5 or 6 for the WCSS criterion, since in this case, the lower bound is not very sharp. For this WCSD criterion, the best known exact method is a recent column generation algorithm proposed in [1]. The authors solve problems with up to 2300 objects; however, the number of objects per group (n/k) should be small, roughly equal to 10, in order to solve the problem in reasonable time. To the best of our knowledge, there exists no exact algorithms for WCSD or WCSS criterion that integrate user-constraints.

5 A CP framework for constraint-based clustering

We present a CP model for constrained clustering. As input, we have a dataset of n points (objects) and a distance measure between pairs of points, denoted by $d(o_i, o_j)$. Without loss of generality, we suppose that points are indexed and named by their index. The number of clusters is fixed by the user and we aim at finding a partition of data into k clusters, satisfying a set of constraints specified by the user and optimizing a given criterion.

5.1 Model

Variables For each cluster $c \in [1, k]$, the point with the smallest index is considered as the representative point of the cluster. An integer variable $I[c]$ is introduced, its value is the index of the representative point of c ; the domain of $I[c]$ is therefore the interval $[1, n]$. This representative point must not be confused with the notion of representative in the medoid approach. Here it only allows to have a single representation of a cluster.

Assigning a point to a cluster is equivalent to 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 of i .

Let us for instance suppose that we have 7 points o_1, \dots, o_7 and that we have 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 integer (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).

A variable is also introduced for representing the optimization criterion. It is denoted by D for the maximal diameter, S for the minimal margin and V for the Within-Cluster Sums of Dissimilarities. They are real variables, since distance are real numbers. The domains of the variables D and S are the intervals the lower bound is the minimal distance between two points and the upper bound is the maximal distance between any pair of points. The domain of V is upper bounded by the sum of the square distances between all pairs of points. Our model allows to find an optimal solution satisfying the following constraints.

Constraints on the representation. The relations between points and their clusters are expressed by these constraints:

- A representative belongs to the cluster it represents:

$$\forall c \in [1, k], G[I[c]] = I[c].$$

- Each point must be assigned to a representative: for $i \in [1, n]$,

$$\bigvee_{c \in [1, k]} (G[i] = I[c]).$$

This can be expressed by a cardinality constraint:

$$\forall i \in [1, n], \#\{c \mid I[c] = G[i]\} = 1.$$

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

$$\forall i \in [1, n], G[i] \leq i.$$

A set of clusters could be differently represented, depending on the order of clusters. For instance, in the previous example, 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:

- Representatives are sorted in increasing order:

$$\forall c < c' \in [1, k], I[c] < I[c'].$$

- The representative of the first cluster is the first point:

$$I[1] = 1.$$

Modeling different objective criteria. Different constraints are added, depending on the criterion to optimize.

- When minimizing the maximal diameter:
 - Two points at a distance greater than the maximal diameter must be in different clusters: for all $i, j \in [1, n]$ such that $i < j$

$$d(i, j) > D \rightarrow (G[j] \neq G[i] \wedge G[j] \neq i) \quad (7)$$

- The maximal diameter is minimized: minimize D .
- When maximizing the minimal margin between clusters:
 - Two points at a distance less than the minimal margin must be in the same cluster. For all $i, j \in [1, n]$ such that $i < j$

$$d(i, j) < S \rightarrow G[j] = G[i] \quad (8)$$

- The minimal margin is maximized: maximize S .
- When minimizing the Within-Cluster Sums of Dissimilarities (WCSD):
 - Let V the variable measuring WCSD,

$$V = \sum_{i, j \in [1, n]} (G[i] \neq G[j]) d(i, j)^2 \quad (9)$$

- minimize V

Modeling user-defined constraints The model allows to extend the framework by adding new constraints. The following user-defined constraints may be straightforwardly put on clusters:

- A constraint on the minimal size α of clusters:

$$\forall c \in [1, k], \#\{i \mid G[i] = I[c]\} \geq \alpha.$$

- A constraint on the maximal size β of clusters:

$$\forall c \in [1, k], \#\{i \mid G[i] = I[c]\} \leq \beta.$$

- The δ -constraint expresses that the margin between two clusters must be at least δ . Therefore, for each $i < j \in [1, n]$ satisfying $d(i, j) < \delta$, we put the constraint:

$$G[i] = G[j]$$

- The diameter constraint expresses that the diameter of each cluster must be at most γ , therefore for each $i < j \in [1, n]$ such that $d(i, j) > \gamma$, we put the constraints:

$$G[j] \neq G[i]$$

- The density constraint that we have introduced expresses that each point must have in its neighborhood with radius ϵ , at least *MinPts* points belonging to the same cluster as itself. So, for each $i \in [1, n]$, the set of points in its ϵ -neighborhood is computed and a constraint is put on its cardinality:

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

- A must-link constraint on two points i and j is represented by:

$$G[i] = G[j]$$

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

$$G[i] \neq G[j]$$

Adding such constraints involves other constraints on D or S , as for instance $G[i] = G[j]$ implies $D \geq d(i, j)$ and $G[i] \neq G[j]$ implies $S \leq d(i, j)$.

5.2 Model improvement

Since constraint propagation allows to reduce the search space by deleting values in the domain of variables that cannot lead to solutions, CP solvers perform an exhaustive search, allowing to find an optimal solution. Different aspects are considered to improve the efficiency of the model.

Improvement by ordering the points Variables of I are chosen in the order $I[1], \dots, I[k]$ and for each variable, the index for representatives are chosen in an increasing order. The way points are indexed is therefore very important. Points are ordered and indexed, so that points that are possible representatives have the smallest index. In order to achieve this, we rely on FPF algorithm, introduced in Section 2. We apply it with $k = n$ (as many classes as points): a first point is chosen, the second point is the furthest object from this point, the third one is the furthest object from the two first and so on until all points have been chosen.

Improvement when minimizing the maximal diameter Let us consider first the case where no user-defined constraints are put in the model. In [14], it is proven that if d_{FPF} represents the maximal diameter of the partition computed by FPF, then it satisfies $d_{opt} \leq d_{FPF} \leq 2d_{opt}$, with d_{opt} the maximal diameter of the optimal solution. This knowledge gives us bounds on D : $D \in [d_{FPF}/2, d_{FPF}]$. Moreover, for each pair of points i, j :

- if $d(i, j) < d_{FPF}/2$, the reified constraint (7) on i, j is no longer put,
- if $d(i, j) > d_{FPF}$, the constraint (7) is replaced by a cannot-link constraint : $G[j] \neq G[i]$.

Such a result allows to remove several reified constraints, without modifying the semantics of the model, and thus allows to improve the efficiency of the model, since handling reified constraints requires to introduce new variables.

In the case where user constraints are added, this result is no longer true, since the optimal diameter is in general greater than the optimal diameter d_{opt} obtained with no user constraints. The upper bound is no longer satisfied but we still have the lower bound, namely $d_{FPF}/2$. Therefore for each pair of points i, j , if $d(i, j) < d_{FPF}/2$, the constraint (7) on i, j is not put.

Improvement when minimizing WCSD Computing WCSD (9) requires to use a linear constraint on boolean variables ($G[i] == G[j]$). However, a partial assignment of points to clusters does not allow to filter the domain of the remaining values, thus leading to an inefficient constraint propagation. We have proposed a new method in the sub section below for propagating this constraint and filtering the domain of remaining variables. A new branching strategy for minimizing WCSD is also proposed for better computing.

5.3 Filtering algorithm for the criterion sum constraint

The constraint (9) for the criterion WCSD can be implemented by a linear sum constraint $V = \sum_{i < j} S_{ij} d(i, j)^2$ on boolean variables S_{ij} , with reified constraints $S_{ij} = (G[i] == G[j])$. However, these constraints, while considered independently, do not offer enough propagation. For example, with $k = 2$, given 4 points 1 to 4 and a partial assignment where $G[1] = 1$ and $G[2] = G[3] = 2$. We have three boolean variables S_{14}, S_{24}, S_{34} with $S_{i4} = (G[i] == G[4])$. Let us assume that during a branch-and-bound search, the upper bound of V is set to 4, and with the minimization of V the constraint $S_{14} + 2S_{24} + 3S_{34} < 4$ is added. We can see that S_{24} and S_{34} must be equal since $G[2] = G[3]$ and then must not be equal to 1, otherwise the constraint is violated. We should then infer that point 4 cannot be in the same cluster as points 2 and 3, so value 2 should be removed from the domain of $G[4]$. This filtering however is not done, since the constraints are considered independently.

Several recent works have proposed more efficient filtering for the sum constraint, when it is considered with other constraints. For a sum constraint $y = \sum x_i$ with inequality constraints $x_j - x_i \leq c$, a domain filtering algorithm reduces the domain of x_i when new bounds for y are known [22]. A bound-consistency algorithm is proposed for a sum constraint with increasing order constraints $x_i \leq x_{i+1}$ [20] or with a constraint *alldifferent*(x_1, \dots, x_n) [3]. Our constraint however does not fit these cases. A generic bound-consistency algorithm for a sum constraint with a set of constraints is proposed in [21]. In our case, the domain of $G[i]$ is a set of representative indices, which is not an interval in general, and where we wish to remove any inconsistent value. We have therefore developed a filtering algorithm for a new global constraint on a variable V , an array of variables G of size n and an array of constants a , which is of the form:

$$V = \sum_{1 \leq i < j \leq n} (G[i] == G[j]) a_{ij}. \quad (10)$$

Taking into account the partitioning problem, the domain of each variable $G[i]$ is a set of the representative indices of all clusters, into which point i can be assigned. Let us assume that $V \in [V.lb, V.ub]$ and we have a partial assignment of variables in G , where there is at least one point assigned for each group (*e.g* the representative of the group, cf. sub-section 5.4). Let K be the set of points i which have been already assigned to a group ($G[i]$ is instantiated) and U the set of the unassigned points. The sum in (10) is split into three parts:

$$\sum_{i, j \in K, i < j, G[i] = G[j]} a_{ij} + \sum_{i \in U, j \in K} (G[i] == G[j]) a_{ij} + \sum_{i < j, i, j \in U} (G[i] == G[j]) a_{ij}$$

An exact value v_1 is calculated for the first part. For the second part, a lower bound v_2 can be defined by $\sum_{i \in U} v_{2i}$, where v_{2i} is the minimal added amount when point $i \in U$ will be assigned to a group, with respect to the points in K , $v_{2i} = \min_c (\sum_{j \in K \cap C_c} a_{ij})$. Let us calculate now a lower bound v_3 for the third part. Let $p = |U|$, what is the minimal number of pairwise connections between any two points in the same group, while considering all possibilities of assignment of p points into k groups? For example with $p = 10$, $k = 3$ this minimal number is 12, corresponding to a partition with 2 groups of 3 points and 1 group of 4 points. Let m be the quotient of the division of p by k and m' the remainder. Let the number of points in each group c be $m + \alpha_c$, with $\alpha_c < 0$ when the group c has less than m points, $\alpha_c \geq 0$ otherwise. We have then $\sum_{1 \leq c \leq k} (m + \alpha_c) = p = km + m'$, so $m' = \sum_{1 \leq c \leq k} \alpha_c$. The total number of connections between any two points in the same group is:

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

The equality is reached when α_c is 1 for m' groups and is 0 for $k - m'$ groups. We denote by $f(p)$ the formula $(km^2 + 2mm' - km)/2$. With a set U of unassigned points, with the constants

a_{ij} ($i, j \in U$) ordered increasingly, a lower bound v_3 is then calculated by the sum of the $f(|U|)$ first constants in this order. It is worth to notice that when any point in U is assigned to a group, the new lower bound v'_3 for the remaining points in U is greater or equal to the sum of $f(|U| - 1)$ first constants in this order (we name it by v_4 in Algorithm 7), *i.e.* $v'_3 \geq v_4$.

The filtering algorithm is presented in Algorithm 7. This algorithm uses arrays add and min , where $add[i, c]$ is the added amount if i is assigned to group c ($add[i, c] = \sum_{j \in K \cap C_c} a_{ij}$) and $m[i]$ is the minimal added amount while considering all possible assignments for i ($m[i] = \min_c add[i, c]$). Since the constants a_{ij} must be ordered increasingly, they are ordered once in the array ord , so $ord[pos]$ gives the constant a_{ij} in the order at position pos , and $px[pos]$ ($py[pos]$) gives the index i (j , resp.) of the constant. At the moment, the filtering algorithm is developed for the clustering task, where values in the domain of $G[i]$ are the representative of all clusters for which point i can be assigned. Given an index a , the function $gr(a)$ gives the number of the cluster corresponding to a .

Algorithm 7: Filtering algorithm

```

1  $v_1 \leftarrow 0; v_2 \leftarrow 0; v_3 \leftarrow 0; v_4 \leftarrow 0;$ 
2 for  $i \leftarrow 1$  to  $n$  where  $G[i]$  is instanciated do
3   for  $j \leftarrow 1$  to  $n$  do
4     if  $G[j]$  is instanciated and  $G[j] == G[i]$  and  $i < j$  then  $v_1 \leftarrow v_1 + a_{ij};$ 
5     if  $G[j]$  is not instanciated then  $add[j, gr(G[i])] \leftarrow add[j, gr(G[i])] + a_{ij};$ 
6 for  $i \leftarrow 1$  to  $n$  where  $G[i]$  is not instanciated do
7    $m[i] \leftarrow add[i, 1];$ 
8   for  $c \leftarrow 2$  to  $k$  do
9     if  $add[i, c] > 0$  and  $m[i] > add[i, c]$  then  $m[i] \leftarrow add[i, c];$ 
10   $v_2 \leftarrow v_2 + m[i];$ 
11  $p \leftarrow$  number of uninstanciated variables in  $G$ ;
12  $cpt \leftarrow 0; pos \leftarrow 1;$ 
13 while  $cpt < f(p)$  do
14    $i \leftarrow px[pos]; j \leftarrow py[pos];$ 
15   if  $G[i]$  is not instanciated and  $G[j]$  is not instanciated then
16      $cpt \leftarrow cpt + 1;$ 
17      $v_3 \leftarrow v_3 + ord[pos];$ 
18     if  $cpt \leq f(p - 1)$  then  $v_4 \leftarrow v_4 + ord[pos];$ 
19    $pos \leftarrow pos + 1;$ 
20  $V.lb \leftarrow \max(V.lb, v_1 + v_2 + v_3);$ 
21 for  $i \leftarrow 1$  to  $n$  where  $G[i]$  is not instanciated do
22   foreach  $value a \in Dom(G[i])$  do
23     if  $V.lb + add[i, gr(a)] - m[i] - v_3 + v_4 > V.ub$  then
24     delete  $a$  from  $Dom(G[i]);$ 

```

The lower bound of V is revised in line 20. Lines 21 to 24 filter the domain of $G[i]$ ($i \in U$): for each value a in the domain, in case of assignment of i into group $gr(a)$, a new lower bound for V is $(v_1 + add[i, gr(a)]) + (v_2 - m[i]) + v'_3$, which is greater or equal to $V.lb + add[i, gr(a)] - m[i] - v_3 + v_4$, so if this value is greater than the actual upper bound of V , a is inconsistent.

The complexity of this algorithm is $O(n^2 + nk)$, since the domain of each $G[i]$ is of size at most k . Since $k \leq n$, the complexity is then $O(n^2)$.

5.4 Search strategies

Let us recall that a solver iterates two steps: constraint propagation and branching when needed. Variables are chosen first those in I , then those in G , which means the cluster representatives are first identified, points are then assigned to clusters. Since a cluster representative $I[c]$ must have the smallest index among those in the cluster, values for $I[c]$ are chosen in increasing order. Point index are then really important, by consequent, points are ordered previously in such a way that those which are more likely to be representative have small index. We use FPF (Furthest Point First) heuristic [14] to reorder points.

The branching on uninstantiated variables in G finds a variable $G[i]$ and a value c in the domain of $G[i]$ and makes two alternatives: $G[i] = c$ and $G[i] \neq c$. To improve the efficiency of the model, different branching strategies for variables in G are used:

- When minimizing the maximal diameter or maximizing the minimal margin between clusters: Variables $G[i]$ are chosen so that the ones with the smallest remaining domain are chosen first. Among those which have the smallest domain size, the variable $G[i]$ with smallest index i is chosen. For instantiating $G[i]$, the index of the closest representative is chosen first.
- When minimizing the Within-Cluster Sums of Dissimilarities: A mixed strategy is used. The variable $G[i]$ is always selected among those which have the smallest domain size. The value c is always the representative of the closest group to point i . Because an upper bound is necessary for the constraint (10), a greedy search is used first to find quickly a solution. In this step, $G[i]$ and c are selected to make sure that the value of V will increase as little as possible. The solution found in general is quite good. After first solution, the search strategy is changed to a “first-fail” search, which tends to cause the failure early. In this strategy, the branching will try to make alternatives on frontier points, *i.e.* those that make most changes on V .

6 Experimentations

6.1 Datasets and methodology

Eleven data sets are used in our experiments. They vary significantly in their size, number of attributes and number of clusters. Nine data sets are from the UCI repository ([2]): Iris, Wine, Glass, Ionosphere, WDBC, Letter Recognition, Synthetic Control, Vehicle, Yeast. For the data set Letter Recognition, only 600 objects of 3 classes are considered from the 20.000 objects in the original data set, they are composed of the first 200 objects of each class. The data sets GR431 and GR666 are obtained from the library TSPLIB ([23]); they contain coordinates of 431 and 666 cities in European [15]. These two data sets do not contain the information about the number of clusters k and we choose $k = 3$ for the tests. Table 1 summarizes informations about these data sets.

There are few works dealing with finding the best optimum, and as far as we know no work in the framework of user constraint. In the case with no user-constraints, our model is compared to the Repetitive Branch-and-Bound Algorithm (RBBA) defined in [6]², which, to the best of our knowledge, is the best exact algorithm for minimum diameter partitioning and minimum within-cluster sums of dissimilarities partitioning. Our model is implemented with the Gecode 4.0.0 library³. In this newest version of Gecode released in 2013, float variable is supported. This property is important for our model to obtain exact optimal value. The distance between objects are the Euclidean distance and the dissimilarity is measured as the squared Euclidean distance. Since there is no exact algorithms that handle user-constraints, we aim at showing the ability of our model to handle different kinds of user constraints.

²The program can be found in <http://mailer.fsu.edu/~mbrusco/>

³<http://www.gecode.org>

Table 1: Properties of data sets used in the experimentation

Data set	# Objects	# Attributes	# Clusters
Iris	150	4	3
Wine	178	13	3
Glass	214	9	7
Ionosphere	351	34	2
GR431	431	2	not available
GR666	666	2	not available
WDBC	569	30	2
Letter Recognition	600	16	3
Synthetic Control	600	60	6
Vehicle	846	18	4
Yeast	1484	8	10

Table 2: Comparison of performance with the minimum diameter criterion

Data set	Optimal Diameter	Our Model	RBBA
Iris	2.58	0.03s	1.4s
Wine	458.13	0.3s	2.0s
Glass	4.97	0.9s	42.0s
Ionosphere	8.60	8.6s	> 2 hours
GR431	141.15	0.6s	> 2 hours
GR666	180.00	31.7s	> 2 hours
WDBC	2377.96	0.7s	> 2 hours
Letter Recognition	18.84	111.6s	> 2 hours
Synthetic Control	109.36	56.1s	> 2 hours
Vehicle	264.83	14.9s	> 2 hours
Yeast	0.67	2389.9s	> 2 hours

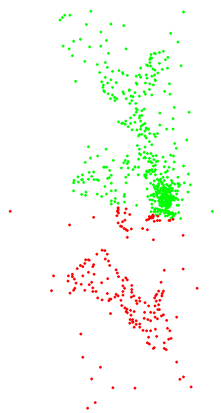
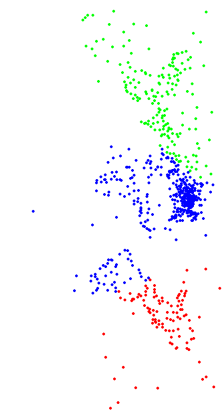
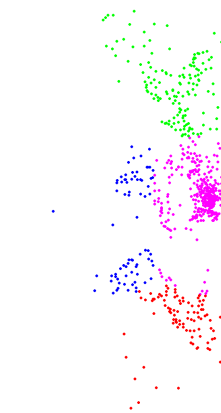
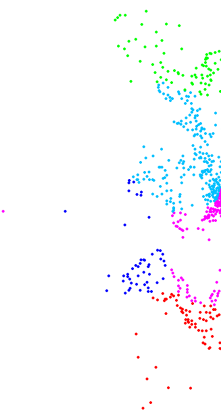
Experiments with our model and with RBBA programs are all performed on a PC Intel core i5 with 3.8 GHz clock and 8 Gigabytes of Ram memory. The time limit for each test is 2 hours.

6.2 Minimizing maximum diameter without user-constraint

Table 2 shows the results for the minimum diameter criterion. The first column gives the data sets, whereas the second column reports the optimal values of the diameter. They are the same for both our model and the RBBA approach, since both approaches find the global optimal. The third and fourth columns give the total CPU times (in seconds) for each approach.

The results show that RBBA found the optimal diameter only for the first three data sets. In [6], the authors also mention that their algorithm is effective for databases with less than 250 objects. Table 2 shows that our model is able to find the optimal diameter for partitioning a data set with up to $n = 1484$ objects and $k = 10$ clusters. The performance does not only depend on the number of objects n and on the number of groups k , but also on the separation of objects and on the database features. The behavior of our model differs from classical models: for instance, when the number of clusters k increases, the search space is larger and in many approaches, solving such a problem takes more computational time. Indeed, as shown in Table 3, since there are more clusters, the maximum diameter is smaller, and propagation of the diameter constraint is more effective, thus explaining that in some cases, it takes less computational time. As already mentioned, they may exist several partitions with the same optimal diameter; because of the search strategy of Constraint Optimization Problem in CP, our model finds only one partition with the optimal diameter.

Table 3: Data set GR666 with the minimum diameter criterion

			
k=2	k=3	k=4	k=5
Total time = 1.21s	Total time = 30.69s	Total time = 5.03s	Total time = 33.51s
Diameter = 224.63	Diameter = 180.00	Diameter = 141.15	Diameter = 115.13

6.3 Minimizing maximum diameter with user-constraints

Let us consider now the behavior of our system with user-constraints with the Letter Recognition dataset. Figure 7 presents the results when must-link constraints (generated from the real class of objects) and a separation constraint δ (the margin between two clusters must be at least δ) is used. The number of must-link constraints varies from 0.01% to 1% the total number of pairs of objects where δ ranges from 3% to 12% the maximum distance between two objects. Regarding to the results, both must-link and separation constraints boost the performance for this data set as they reduce the search space (or the number of feasible solutions).

Results in Table 4 show different partitions with user-constraint of separation on the data set GR666 with the number of clusters $k = 2$. The minimum separation constraint varies from 1%, 4% and 7% the maximum distance between two objects. The results show that with appropriate user-constraint, a user can get more meaningful partition.

6.4 Minimizing Within-Cluster Sums of Dissimilarities

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

WCSD and separation constraint Let us add a separation constraint δ (the margin between two clusters must be at least δ), where δ ranges from 0% (no constraint) to 12% of the maximum distance between two objects. Table 5 reports the WCSD value of an optimal solution with the total time for computation. It shows that when the separation constraint is weak, the optimal WCSD value does not change. But the computation time decreases significantly when this additional constraint becomes stronger. The reason is that the total number of feasible solutions decreases and the search space is reduced. When the separation constraint is weak, propagating this constraint is more time-consuming than its benefits. With appropriate user-constraints, user can find meaningful partition while the optimal solution is always guaranteed.

Figure 7: Must-link and separation constraints for WCSD with data set Iris

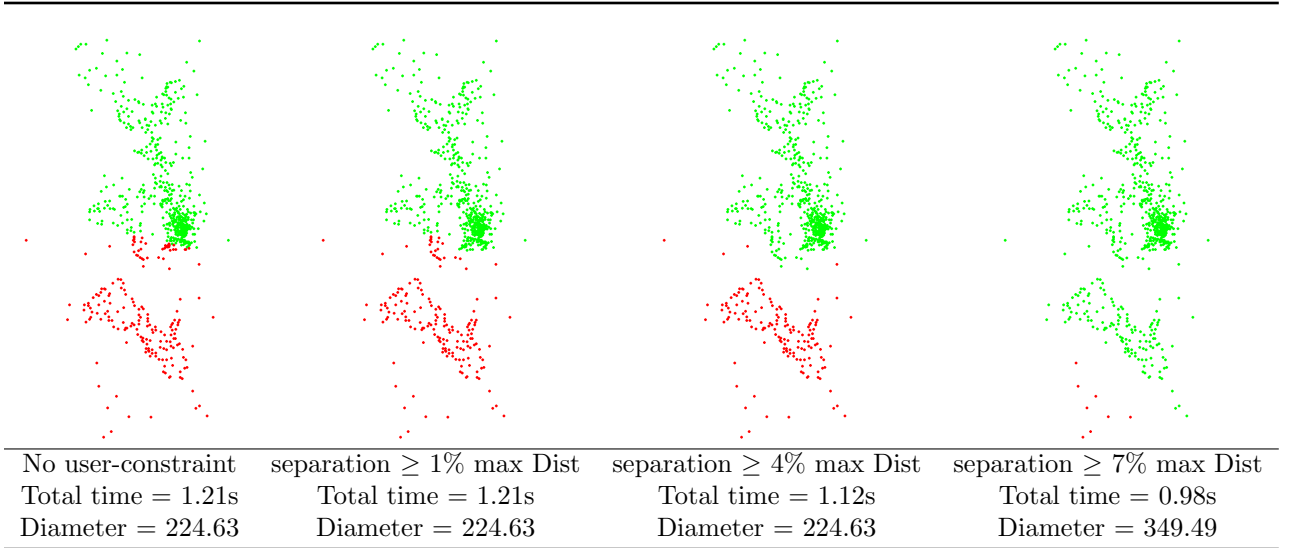
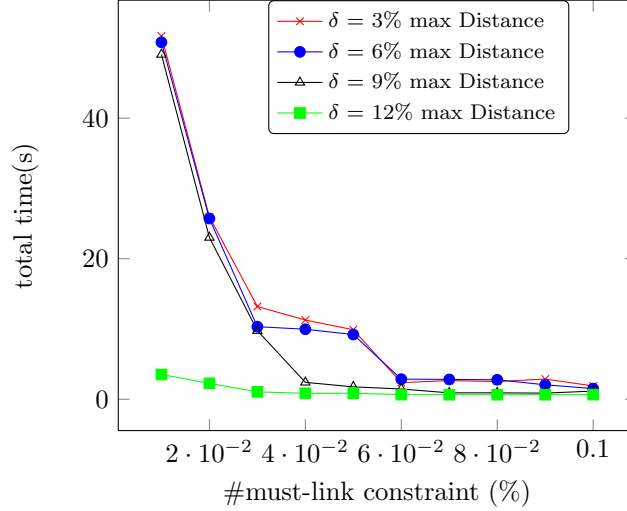


Table 4: database GR666 with separation constraint

Table 5: Separation constraint with data set Iris

Separation Constraint	WCS	Total time
no constraint	573.552	4174s
$\delta = 2\%$ max Distance	573.552	1452s
$\delta = 4\%$ max Distance	573.552	84.4s
$\delta = 6\%$ max Distance	573.552	0.3s
$\delta = 8\%$ max Distance	2169.21	0.1s
$\delta = 10\%$ max Distance	2412.43	0.04s
$\delta = 12\%$ max Distance	2451.32	0.04s
$\delta = 14\%$ max Distance	No Solution	0.04s

Table 6: Must-link constraints with data set Iris

# must-link Constraints	WCSD	Total time
no constraint	573.552	4174s
0.2%	602.551	1275.1s
0.4%	602.551	35.6s
0.6%	617.012	16.1s
0.8%	622.5	3.5s
1%	622.5	1.6s
100%	646.045	0.04s

Table 7: Example of appropriate combinations of user-constraints

Data set	User constraints	WCSD	Total time
Wine	separation: $\delta = 1.5\%$ max Distance minimum capacity: $\beta = 30$	1.40×10^6	11.2s
GR666	separation: $\delta = 1.5\%$ max Distance diameter: $\gamma = 50\%$ max Distance	1.79×10^8	12.4s
Letter Recognition	# must-link constraints = 0.1% total pairs # cannot-link constraints = 0.1% total pairs separation: $\delta = 10\%$ max Distance	5.84×10^6	11.5s
Vehicle	separation: $\delta = 3\%$ max Distance diameter: $\gamma = 40\%$ max Distance	1.93×10^9	1.6s

WCSD and must-link constraint Let us now add must-link constraints, where the number of must-link constraints, generated from the real class of objects, varies from 0.2 to 1% of the total number of pairs. Results are expressed in Table 6, giving the WCSD value and the total computation time. In fact, the optimal value of WCSD, with no information on classes, does not correspond to the WCSD found when considering the partition of this dataset into the 3 defined classes. The more must-link constraints, the less computation time is needed for finding the optimal value, and the closer to the value of WCSD, when considering the 3 initial classes. The reduction of computation time can be easily explained, since when an object is instantiated, objects that must be linked to it are immediately instantiated too. Furthermore, with any kind of additional constraint, the total number of feasible solutions is always equal or less than the case with no constraint.

WCSD and appropriate user-constraints Finding an exact solution minimizing the WCSD is difficult. However, with appropriate combination of user-constraints, the performance can be boosted. Table 7 presents some examples where our model can get an optimal solution with different user-constraints, which reduce significantly the search space.

6.5 Quality of the solution

The possibility of combining different constraints allows to find the desired solution and generally improves the quality of the solution found. We consider three data sets 2D which are similar to those used in [12]. The data are shown in Figure 8.

In the first data set, there are four groups of different diameters. The second data set is more difficult because the groups have different shapes. With the criterion of minimizing maximum diameter, the solver finds groups with homogeneous diameters, so this criterion alone is not very suitable, as shown in Figure 9. The addition of a separation constraint, with the parameter δ is equal to 5% of the maximum distance between pairs of points, significantly improves the quality of the solution, as presented Figure 10. Note that the criterion of maximizing the minimum separation also allows you to find the solution.

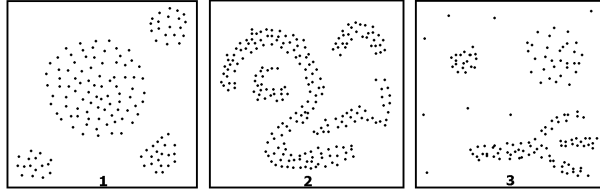


Figure 8: Data sets

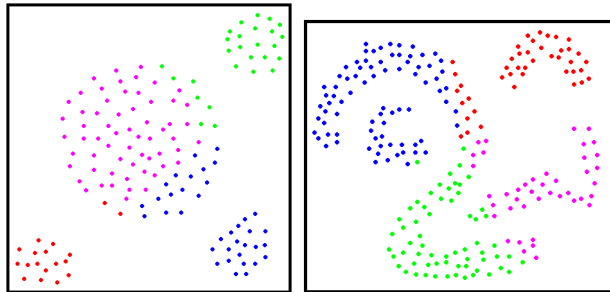


Figure 9: Criterion of diameter

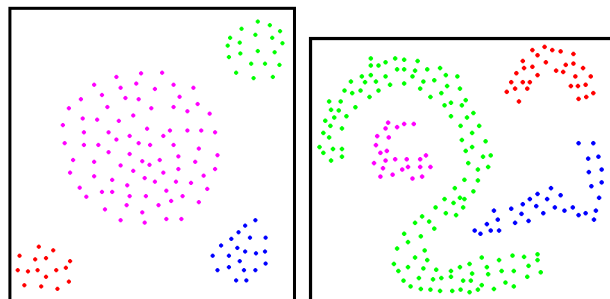


Figure 10: best solution

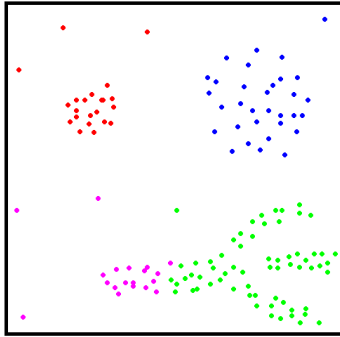


Figure 11: Minimize the maximal diameter

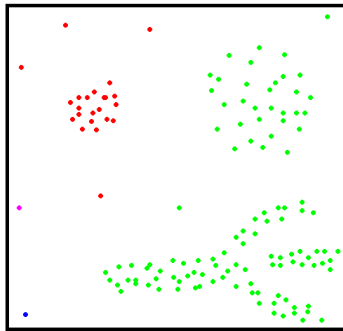


Figure 12: Maximize the minimal separation

The third database contains separate points (outliers). Our model does not detect outliers, so these points are also clustered. The criterion of minimizing maximum diameter (Figure 11) or maximizing minimum separation (Figure 12) does not find a good solution. However, the quality of the solution is much improved when a density constraint is added, with $MintPts = 4$ and ϵ is equal to 25% maximum distance between pairs of points (Figure 13).

7 Conclusion

We have proposed a declarative framework for Constrained Clustering based on Constraint Programming. It allows to choose among different optimization criteria and to integrate various kinds of constraints. One of the advantage of Constraint Programming, besides its declarative, is that

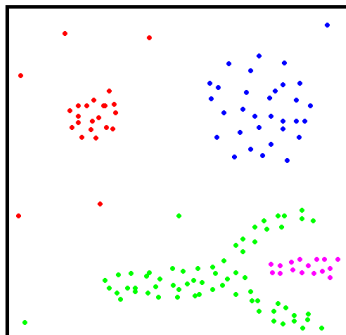


Figure 13: Criterion of separation with density constraints

it allows to find an optimal solution, whereas most approaches find only local optima. On the other hand, complexity makes it difficult to handle very large databases. Nevertheless, integrating constraints enables to reduce the search space, depending on their ability to filter the domain of variables. Moreover, working on search strategies and on constraint propagation enables to increase the efficiency and to deal with larger problems.

In the future, we plan to work more on the search strategies and on the constraint propagators, thus being able to address bigger databases. We do believe that global constraints adapted to the clustering tasks must be developed. From the Data Mining point of view, more optimization criteria should be added. In the other side, the filtering algorithm for the WCSD criterion could be generalized to be used in other constraint satisfaction problems.

References

- [1] Daniel Aloise, Pierre Hansen, and Leo Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Math. Program.*, 131(1-2):195–220, 2012.
- [2] K. Bache and M. Lichman. UCI machine learning repository, 2013.
- [3] Nicolas Beldiceanu, Mats Carlsson, Thierry Petit, and Jean-Charles Régin. An $o(n \log n)$ bound consistency algorithm for the conjunction of an alldifferent and an inequality between a sum of variables and a constant, and its generalization. In *ECAI 2012 - 20th European Conference on Artificial Intelligence*, pages 145–150, 2012.
- [4] M. Bilenko, S. Basu, and R. J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 11–18, 2004.
- [5] Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, Samir Loudni, and Jean-Philippe Métivier. Discovering Knowledge using a Constraint-based Language. *CoRR*, abs/1107.3407, 2011.
- [6] Michael Brusco and Stephanie Stahl. *Branch-and-Bound Applications in Combinatorial Data Analysis (Statistics and Computing)*. Springer, 1 edition, July 2005.
- [7] I. Davidson and S. S. Ravi. Agglomerative hierarchical clustering with constraints: Theoretical and empirical results. *Proceedings of the 9th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, pages 59–70, 2005.
- [8] Ian Davidson and S. S. Ravi. Clustering with Constraints: Feasibility Issues and the k-Means Algorithm. In *Proc. 5th SIAM Data Mining Conference*, 2005.
- [9] Ian Davidson, S. S. Ravi, and Leonid Shamis. A SAT-based Framework for Efficient Constrained Clustering. In *SDM*, pages 94–105, 2010.
- [10] L. De Raedt, T. Guns, and S. Nijssen. Constraint Programming for Data Mining and Machine Learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.
- [11] Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint programming for itemset mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 204–212, 2008.
- [12] Martin Ester, Hans P. Kriegel, Jorg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, 1996. AAAI Press.

- [13] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [14] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
- [15] Martin Grötschel and Olaf Holland. Solution of large-scale symmetric travelling salesman problems. *Math. Program.*, 51:141–202, 1991.
- [16] Tias Guns, Siegfried Nijssen, and Luc De Raedt. k-Pattern set mining under constraints. *IEEE Transactions on Knowledge and Data Engineering*, 2011. Accepted.
- [17] Zhengdong Lu and Miguel A. Carreira-Perpinan. Constrained spectral clustering through affinity propagation. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, June 2008.
- [18] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. Constrained Clustering Using SAT. In *IDA 2012, LNCS 7619*, pages 207–218, 2012.
- [19] Marianne Mueller and Stefan Kramer. Integer linear programming models for constrained clustering. In *Discovery Science*, pages 159–173, 2010.
- [20] Thierry Petit, Jean-Charles Régin, and Nicolas Beldiceanu. A $\theta(n)$ bound-consistency algorithm for the increasing sum constraint. In *Principles and Practice of Constraint Programming CP 2011*, pages 721–728, 2011.
- [21] Jean-Charles Régin and Thierry Petit. The objective sum constraint. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CPAIOR 2011*, pages 190–195, 2011.
- [22] Jean-Charles Régin and Michel Rueher. Inequality-sum: a global constraint capturing the objective function. *RAIRO - Operations Research*, 39(2):123–139, 2005.
- [23] G. Reinelt. TSPLIB - A t.s.p. library. Technical Report 250, Universität Augsburg, Institut für Mathematik, Augsburg, 1990.
- [24] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*. Foundations of Artificial Intelligence. Elsevier B.V., Amsterdam, Netherlands, August 2006.
- [25] Michael Steinbach, Pang-Ning Tan, and Vipin Kumar. *Introduction to data mining*. Addison-Wesley, us ed edition, May 2005.
- [26] K. Wagstaff and C. Cardie. Clustering with instance-level constraints. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 1103–1110, 2000.
- [27] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the 8th Int. Conf. on Machine Learning*, pages 577–584, 2001.
- [28] Xiang Wang and Ian Davidson. Flexible constrained spectral clustering. In *KDD '10: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 563–572, 2010.