Constrained Minimum Sum of Squares Clustering by Constraint Programming

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

Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067, Orléans, France
{thi-bich-hanh.dao, khanh-chuong.duong, christel.vrain}@univ-orleans.fr

Abstract. The Within-Cluster Sum of Squares (WCSS) is the most used criterion in cluster analysis. Optimizing this criterion is proved to be NP-Hard and has been studied by different communities. On the other hand, Constrained Clustering allowing to integrate previous user knowledge in the clustering process has received much attention this last decade. As far as we know, there is a single approach that aims at finding the optimal solution for the WCSS criterion and that integrates different kinds of user constraints. This method is based on integer linear programming and column generation. In this paper, we propose a global optimization constraint for this criterion and develop a filtering algorithm. It is integrated in our Constraint Programming general and declarative framework for Constrained Clustering. Experiments on classic datasets show that our approach outperforms the exact approach based on integer linear programming and column generation.

1 Introduction

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 usually formulated as an optimization problem and different optimization criteria have been defined [18]. One of the most used criteria is minimizing the Within-Cluster Sum of Squares (WCSS) Euclidean distances from each object to the centroid of the cluster to which it belongs. The well-known k-means algorithm as well as numerous heuristic algorithms optimize it and find a local optimum [27]. Finding a global optimum for this criterion is a NP-Hard problem and even finding a good lower bound is difficult [1]. The best exact approach for clustering with this criterion is based on Integer Linear Programming (ILP) and column generation [2].

On the other hand, since this last decade, user-defined constraints have been integrated to clustering task to make it more accurate, leading to Constrained Clustering. User constraints usually make the clustering task harder. The extension to user constraints is done either by adapting classic algorithms to handle constraints or by modifying distances between objects to take into account constraints [30, 4, 9]. Recently an exact approach has been proposed, which aims at finding an optimal solution for the WCSS criterion satisfying all the user constraints [3]. This approach extends the method based on ILP and column generation [2]. It integrates different kinds of user constraints but only handle the WCSS criterion.

Recently general and declarative frameworks for Data Mining have attracted more and more attention from Constraint Programming (CP) and Data Mining communities [11, 16, 28, 3]. In our previous work [7, 8] we have proposed a general framework based on CP for Constrained Clustering. Different from classic algorithms that are designed for one specific criterion or for some kinds of user constraints, the framework offers in a unified setting a choice among different optimization criteria and a combination of different kinds of user constraints. It allows to find a global optimal solution that satisfies all the constraints if one exists. Classic heuristic algorithms can quickly find a solution and can scale on very large datasets, however they do not guarantee the satisfaction of all the constraints or the quality of the solution. A declarative and exact approach allows a better understanding of the data, which is essential for small valuable datasets that may take years to collect. In order to make the CP approach efficient, it is important to invest in dedicated global constraints for the clustering tasks [8].

In this paper we propose a global optimization constraint *wcss* to represent the WCSS criterion. We propose a method based on dynamic programming to compute a lower bound and develop a filtering algorithm. The filtering algorithm filters not only the objective variable, but also the decision variables. The constraint extends our framework to the WCSS criterion. Experiments on classic datasets show that our approach based on CP outperforms the state-of-the-art approach based on ILP and column generation that handles user constraints [3].

Outline. Section 2 gives preliminaries on constrained clustering and reviews related work on existing approaches. Section 3 presents the framework based on CP and introduces the constraint *wcss*. Section 4 presents our contributions: the computation of a lower bound and the filtering algorithm for the constraint $wcss$. Section 5 is devoted to experiments and comparisons of our approach with existing approaches. Section 6 discusses perspectives and concludes.

2 Preliminaries

2.1 Constrained Clustering

A clustering task aims at partitioning a given set of objects into clusters, in such a way that the objects in 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 consists in finding a partition of objects that optimizes it. Let us consider a dataset of n objects $\mathcal{O}$ and a dissimilarity measure $d(o, o')$ between any two objects $o, o' \in \mathcal{O}$ (d is defined by the user). A partition Δ of objects into k classes $C_1, \ldots, C_k$ is such that: (1) $\forall c \in [1, k]$[1], $C_c \neq \emptyset$, (2) $\cup_c C_c = \mathcal{O}$ and (3) $\forall c \neq c'$, $C_c \cap C_{c'} = \emptyset$. The optimization criterion may be, among others:

- Maximizing the minimal split between clusters, where the minimal split of a partition Δ is defined by: $S(\Delta) = \min_{c \neq c' \in [1,k]} \min_{o \in C_c, o' \in C_{c'}} d(o, o')$.
- Minimizing the maximal diameter of clusters, which is defined by: $D(\Delta) = \max_{c \in [1,k]} \max_{o, o' \in C_c} d(o, o')$.

[1] For integer value we use $[1, k]$ to denote the set $\{1, .., k\}$.

– Minimizing the Within-Cluster Sum of Dissimilarities, which is defined by:

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

– Minimizing the Within-Cluster Sum of Squares:

$$WCSS(\Delta) = \sum_{c \in [1,k]} \sum_{o \in C_c} d(o, m_c)$$

where for each $c \in [1,k]$, m_c is the mean (centroid) of the cluster C_c and $d(o, m_c)$ is defined by the squared Euclidean distance $||o - m_c||^2$. When the dissimilarity between objects is defined by the squared Euclidean distance $d(o,o') = ||o - o'||^2$, we have [12, 18]:

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

All these criteria except the split criterion are NP-Hard (see e.g. [14] for the diameter criterion, [1] for WCSS, or concerning WCSD, the weighted max-cut problem, which is NP-Complete, is an instance with $k = 2$). Most of classic algorithms use heuristics and search for a local optimum [17]. For instance the k-means algorithm finds a local optimum for the WCSS criterion or FPF (Furthest Point First) [14] for the diameter criterion. Different optima may exist, some may be closer to the one expected by the user. In order to better model the task, user knowledge is integrated to clustering task. It is usually expressed by constraints, leading to Constrained Clustering. User constraints can be cluster-level, giving requirements on clusters, or instance-level, specifying requirements on pairs of objects. The last kind, introduced in [29], is the most used. An instance-level constraint on two objects can be a must-link or a cannot-link constraint, which states that the objects must be or cannot be in the same cluster, respectively.

Different kinds of cluster-level constraints exist. The minimal (maximal) capacity constraint requires that each cluster must have at least (at most, resp.) a given α (β, resp.) objects: $\forall c \in [1,k],\ |C_c| \geq \alpha$ (resp. $\forall c \in [1,k],\ |C_c| \leq \beta$). A diameter constraint sets an upper bound γ on the cluster diameter: $\forall c \in [1,k], \forall o,o' \in C_c,\ d(o,o') \leq \gamma$. A split constraint, named δ-constraint in [9], sets a lower bound δ on the split: $\forall c \neq c' \in [1,k]$, $\forall o \in C_c, o' \in C_{c'}, d(o,o') \geq \delta$. A density constraint, which extends ϵ-constraints in [9], requires that each object o must have in its neighborhood of radius ϵ at least m objects belonging to the same cluster as itself: $\forall c \in [1,k], \forall o \in C_c, \exists o_1, .., o_m \in C_c, o_i \neq o \wedge d(o, o_i) \leq \epsilon$.

User-constraints can make the clustering task easier (e.g. must-link constraints) but usually they make the task harder, for instance the split criterion that is polynomial becomes NP-Hard with cannot-link constraints [10].

2.2 Related Work

We are interested in constrained clustering with the WCSS criterion. The clustering task with this criterion is NP-Hard [1]. The popular k-means algorithm finds a local optimum for this criterion without user constraints. This algorithm starts with a random partition and repeats two steps until a stable state: (1) compute the centroid of the clusters, (2) reassign the objects, where each one is assigned to the cluster whose centroid is the closest to the object. The found solution thus depends on the initial random partition. Numerous heuristic algorithms have been proposed for this criterion, see e.g. [27] for a survey. Exact methods are much less numerous than heuristic. Some of them are based on branch-and-bound [20, 6] or on dynamic programming [19, 23]. An algorithm based on Integer Linear Programming (ILP) and column generation is proposed in [22]. This algorithm is improved in [2] and to our knowledge, it is the most efficient exact approach. It can handle datasets up to 2300 objects with some heuristics [2], but it does not handle user constraints.

The COP-kmeans algorithm [30] extends the k-means algorithm to handle must-link and cannot-link constraints and tries to satisfy all of them. This algorithm changes k-means in step (2): it tries to assign each object to the cluster whose centroid is the closest, without violating the user constraints. If all the possible assignments violate a user constraint, the algorithm stops. This is a greedy and fast algorithm without back-track, but it can fail to find a solution that satisfies all the user constraints, even when such a solution exists [10]. Another family of approaches tries to satisfy only most of the user constraints. The CVQE (Constrained Vector Quantization Error) algorithm [9] or an improved version LCVQE [24] extend the k-means algorithm to must-link and cannot-link constraints by modifying the objective function.

Recently, a general framework, which minimizes the WCSS criterion and which can handle different kinds of user constraints has been developed [3]. This framework extends the approach [2] based on ILP and column generation. It allows to find a global optimum and which satisfies all the user constraints. Due to the choice of variables in the linear program, the framework is however specified for this very criterion and does not handle other optimization criteria.

In our previous work, we have developed a general framework based on Constraint Programming for Constrained Clustering [7, 8]. This framework offers a choice among different optimization criteria (diameter, split or WCSD) and integrates all popular kinds of user constraints. Moreover, the capacity of handling different optimization criteria and different kinds of user constraints allows the framework to be used in bi-criterion constrained clustering tasks [8]. In this paper we extend this framework to integrate the WCSS criterion.

3 Constraint Programming Model

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). In [8] we have presented a CP model for constrained clustering, which integrates the diameter, split and WCSD criteria. We present an extension of this model to the WCSS criterion.

In this model the number of clusters k is not set, but only bounded by k_{min} and k_{max} ($k_{min} \leq k \leq k_{max}$). The values k_{min} and k_{max} are set by the user. If the user needs a known number k of clusters, all he has to do is to set $k_{min} = k_{max} = k$. The clusters are identified by their index, which is a value from 1 to k for a partition of k clusters. In order to represent the assignment of points into clusters, we use integer value variables $G_1, \ldots, G_n$, each one having as domain the set of integers $[1, k_{max}]$. An assignment $G_i = c$ expresses that point i is assigned to the cluster number c. Let $\mathcal{G}$ be $[G_1, \ldots, G_n]$. To represent the WCSS criterion, we introduce a float value variable V, with $Dom(V) = [0, \infty)$.

3.1 Constraints

Any complete assignment of the variables in $\mathcal{G}$ defines a partition of points into clusters. In order to break symmetries between the partitions, we put the constraint *precede*$(\mathcal{G}, [1, \ldots, k_{max}])$ [21], which states that $G_1 = 1$ (the first point is in cluster 1) and $\forall c \in [2, k_{max}]$, if there exists $G_i = c$ with $i \in [2, n]$, then there must exists $j < i$ such that $G_j = c - 1$. Since the domain of the variables G_i is $[1, k_{max}]$, there are at most k_{max} clusters. The fact that there are at least k_{min} clusters means that all the values from 1 to k_{min} must appear in $\mathcal{G}$. We only need to require that k_{min} must appear in $\mathcal{G}$, since with the use of the constraint *precede*, if k_{min} is taken, then $k_{min} - 1, k_{min} - 2, \ldots, 1$ are also taken. This means $\#\{i \mid G_i = k_{min}\} \geq 1$ and can be expressed by the constraint: *atleast*$(1, \mathcal{G}, k_{min})$.

Instance-level user constraints can be easily integrated within this model. A must-link constraint on two points i, j is expressed by $G_i = G_j$ and a cannot-link constraint by $G_i \neq G_j$. All popular cluster-level constraints are also integrated. For instance, a minimal capacity α constraint is expressed by the fact that each point must be in a cluster with at least α points including itself. Therefore, for each $i \in [1, n]$, the value taken by G_i must appear at least α times in $\mathcal{G}$, *i.e.* $\#\{j \mid G_j = G_i\} \geq \alpha$. For each $i \in [1, n]$, we put: *atleast*$(\alpha, \mathcal{G}, G_i)$. This requirement allows to infer an upper bound on the number of clusters. Indeed, $G_i \leq \lfloor n/\alpha \rfloor$, for all $i \in [1, n]$. For other kinds of user constraints, such as maximal capacity, diameter or density constraints, we refer the reader to [8].

In order to express that V is the within-cluster sum of squares of the partition formed by the assignment of variables in $\mathcal{G}$, we develop a global optimization constraint *wcss*$(\mathcal{G}, V, d)$. The filtering algorithm for this constraint is presented in Section 4. The value of V is to be minimized.

3.2 Search Strategy

The branching is on the variables in $\mathcal{G}$. A mixed strategy is used with a branch-and-bound mechanism. A greedy search is used to find the first solution: at each branching, a variable G_i and a value $c \in Dom(G_i)$ such that the assignment $G_i = c$ increases V the least are chosen. The value of V at the first found solution gives an upper bound of the domain of V. After finding the first solution, the search strategy changes. In the new strategy, at each branching, for each unassigned variable G_i, for each value $c \in Dom(G_i)$ we compute the value a_{ic}, which is the added amount to V if point i is assigned to cluster c. For each unassigned variable G_i, let $a_i = \min_{c \in Dom(G_i)} a_{ic}$. The value a_i thus represents the minimal added amount to V when point i is assigned to a cluster. Since each point must be assigned to a cluster, at each branching, the variable G_i

with the greatest value a_i is chosen, and for this variable, the value c with the smallest value a_{ic} is chosen. Two branches are then created, corresponding to two cases where $G_i = c$ and $G_i \neq c$. This strategy tends to detect failure more quickly or in case of success, to find a solution with the value of V as small as possible.

4 Filtering Algorithm for WCSS

We consider that the objects are in an Euclidean space and the dissimilarity measure is defined by the squared Euclidean distance. The sum of dissimilarities of a cluster C_c is defined by $WCSD(C_c) = \frac{1}{2}\sum_{o,o' \in C_c} d(o,o')$. The sum of squares of C_c is defined by $WCSS(C_c) = \frac{1}{|C_c|}WCSD(C_c)$. The WCSS of a partition $\Delta = \{C_1, \ldots, C_k\}$ is $WCSS(\Delta) = \sum_{c \in [1,k]} WCSS(C_c)$.

We introduce a new constraint *wcss*$(\mathcal{G}, V, d)$ and develop a filtering algorithm. This constraint maintains the relation that the float value variable V is the sum of squares of the clusters formed by the assignment of the decision variables of $\mathcal{G}$, given a dissimilarity measure d. Given a partial assignment of some variables in $\mathcal{G}$, we develop a lower bound computation for V and an algorithm to filter the domains of the variables. Since the variable V represents the objective function, this constraint is a global optimization constraint [13, 26]. The filtering algorithm filters not only the domain of the objective variable V, but also the domain of decision variables in $\mathcal{G}$.

A partial assignment of variables of $\mathcal{G}$ represents a case where some points have been already assigned to a cluster and there are unassigned points. Let $k = \max\{c \mid c \in \bigcup_i Dom(G_i)\}$. The value k is the greatest cluster index among those remaining in all the domains $Dom(G_i)$ and thus it is the greatest possible number of clusters in the partition. Let $\mathcal{C}$ be the set of clusters $C_1, \ldots, C_k$. Some of these clusters can be empty, they correspond to indexes that remain in some non-singleton domains $Dom(G_i)$ but not in a singleton domain. For each cluster C_c, let n_c be the number of points already assigned to C_c ($n_c \geq 0$) and let $S_1(C_c)$ be the sum of dissimilarities of all the assigned points in C_c: $S_1(C_c) = \frac{1}{2}\sum_{i,j \in C_c} d(i,j)$. Let U be the set of unassigned points and let $q = |U|$.

4.1 Lower Bound Computation

We compute a lower bound of V considering all the possibilities for assigning all the points in U into the clusters $C_1, \ldots, C_k$. This is done in two steps:

1. For each $m \in [0,q]$ and $c \in [1,k]$, we compute a lower bound $\underline{V}(C_c, m)$ of $WCSS(C_c)$ considering all possible assignments of m points of U into C_c.
2. For each $m \in [0,q]$ and $c \in [2,k]$, we compute a lower bound $\underline{V}(C_1 \ldots C_c, m)$ of $WCSS(\{C_1, .., C_c\})$ considering all the possibilities for assigning any m points of U into the clusters $C_1, \ldots, C_c$.

Existing branch-and-bound approaches [20, 6] are also based on the computation of a lower bound. However, these algorithms only make use of dissimilarities between the unassigned points. In our lower bound computation, we exploit not only dissimilarities between the unassigned points, but also the dissimilarities between unassigned points and assigned points. The computation is achieved by Algorithm 1. The two steps are detailed below.

Algorithm 1: Lower_bound()

```
input  : a partial assignment of $\mathcal{G}$, a set $U = \{i \mid G_i$ unassigned$\}$, $q = |U|$
output: a lower bound of the sum of squares $V$
1  foreach $x \in U$ do
2      for $c \leftarrow 1$ to $k$ do
3          if $c \notin Dom(G_x)$ then $s_2[x, c] \leftarrow \infty$ ;
4          else $s_2[x, c] \leftarrow 0$;
5      foreach $v \in [1, n]$ such that $|Dom(G_v)| = 1$ do
6          if $val(G_v) \in Dom(G_x)$ then $s_2[x, val(G_v)] = s_2[x, val(G_v)] + d(x, v)$;
7      sort $u \in U$ in the increasing order of $d(x, u)$
8      $s_3[x, 0] \leftarrow 0$
9      for $m \leftarrow 1$ to $q$ do
10         $s_3[x, m] \leftarrow s_3[x, m - 1] + d(x, u_m)/2$
11 for $c \leftarrow 1$ to $k$ do
12     for $m \leftarrow 0$ to $q$ do
13         foreach $x \in U$ do
14             $s[x] = s_2(x, c) + s_3(x, m)$
15         sort the array $s$ increasingly
16         $\underline{S_2}(C_c, m) \leftarrow \sum_{i=1}^{m} s[i]$
17         if $n_c + m = 0$ then $\underline{V}(C_c, m) \leftarrow 0$ ;
18         else
19             $\underline{V}(C_c, m) \leftarrow (S_1(C_c) + \underline{S_2}(C_c, m))/(n_c + m)$
20 for $c \leftarrow 2$ to $k$ do
21     $\underline{V}(C_1..C_c, 0) \leftarrow \underline{V}(C_1..C_{c-1}, 0) + \underline{V}(C_c, 0)$
22     for $m \leftarrow 1$ to $q$ do
23         $\underline{V}(C_1..C_c, m) \leftarrow \min_{i \in [0,m]}(\underline{V}(C_1..C_{c-1}, i) + \underline{V}(C_c, m - i))$
24 return $\underline{V}(C_1..C_k, q)$
```

Assignment of any m points of U into a cluster C_c. If we choose a subset $U' \subseteq U$ with $|U'| = m$ and we assign the points of U' into the cluster C_c, the sum of squares of C_c after the assignment will be:

$$V(C_c, U') = \frac{S_1(C_c) + S_2(C_c, U')}{n_c + m}$$

Here $S_1(C_c)$ is the sum of dissimilarities of the points already assigned in C_c and $S_2(C_c, U')$ is the sum of dissimilarities related to points in U'. The value of $S_1(C_c)$ is known. If the set U' is known the value $S_2(C_c, U')$ can be computed exactly by:

$$S_2(C_c, U') = \sum_{u \in U', v \in C_c} d(u,v) + \frac{1}{2} \sum_{u,v \in U'} d(u,v)$$

But $S_2(C_c, U')$ remains unknown while U' is not defined. However, for any subset U' of size m, we can compute a lower bound $\underline{S_2}(C_c, m)$ as follows. Each point $x \in U$, in case of assignment to the cluster C_c together with other $m-1$ points of U, will contribute an amount $s(x,c,m) = s_2(x,c) + s_3(x,m)$, where:

- $s_2(x,c)$ represents the sum of dissimilarities between x and the points already in the cluster C_c. If $c \notin Dom(G_x)$ then $s_2(x,c) = +\infty$, since x cannot be assigned to C_c. Otherwise $s_2(x,c) = \sum_{v \in C_c} d(x,v)$. This value $s_2(x,c)$ is 0 if the cluster C_c is empty. It is computed by lines 2–6 in Algorithm 1.
- $s_3(x,m)$ represents a half of the sum of dissimilarities $d(x,z)$, for all the $m-1$ other points z. These points z can be any points in U, however, if we order all the points $u \in U$ in an increasing order on $d(x,u)$ and we denote by u_i the i-th point in this order, we have a lower bound for $s_3(x,m)$ (lines 7–10 in Algorithm 1): $s_3(x,m) \ \geq \ \frac{1}{2} \sum_{i=1}^{m-1} d(x, u_i)$.

A lower bound $\underline{S_2}(C_c, m)$ is thus the sum of the m smallest contributions $s(x,c,m)$ (represented by $s[x]$ in Algorithm 1, for fixed values c and m), for all points $x \in U$. The lower bound $\underline{V}(C_c, m)$ is 0 if $n_c + m = 0$ or otherwise is computed by:

$$\underline{V}(C_c, m) \ = \ \frac{S_1(C_c) + \underline{S_2}(C_c, m)}{n_c + m} \qquad (1)$$

This is computed for all $c \in [1,k]$ and $m \in [0,q]$ (lines 11–19 in Algorithm 1).

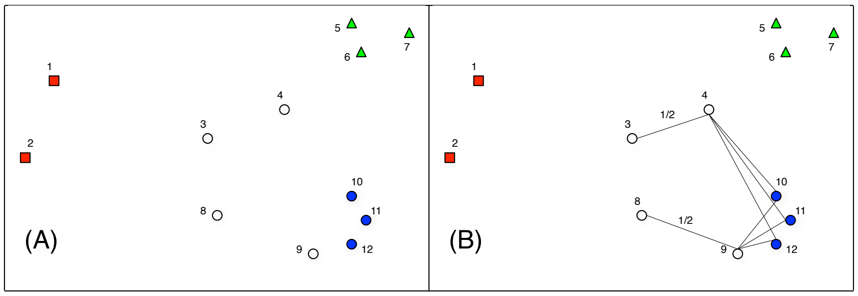

Fig. 1. Example: (A) partial assignment, (B) dissimilarities used in $\underline{S_2}(C_3, 2)$

For an example, let us consider the partial assignment given in Figure 1 (A) where $k = 3$ and some points have been assigned into 3 clusters C_1 (square), C_2 (triangle) and C_3 (circle). The set of unassigned points U is $\{3, 4, 8, 9\}$. A lower bound $\underline{V}(C_3, 2)$ for the sum of squares of C_3 in case of assignment of any 2 points of U into C_3 is computed by Formula (1), with $n_c = 3$ and $m = 2$. In this formula, $S_1(C_3) = d(10, 11) + d(11, 12) + d(10, 12)$ and $\underline{S_2}(C_3, 2)$ is the sum of the 2 smallest contributions to C_3 among those of all the unassigned points. They are the contributions of points 4 and 9. Figure 1 (B) presents the dissimilarities used in the computation of $\underline{S_2}(C_3, 2)$. For the contribution of point 4, we make use of one ($= m-1$) smallest dissimilarity from point 4 to the other unassigned points, which is $d(4, 3)$. Idem for point 9, where $d(9, 8)$ is used. Let us note that the contribution of each point is computed separately, in order to avoid combinatory cases. Therefore $d(4, 9)$ is not used, even though points 4 and 9 are assumed to be assigned to C_3.

Assignment of any m points of U into c clusters $C_1, .., C_c$. Any assignment of m points to c clusters is expressed by an assignment of some i points to the first $c-1$ clusters and the remaining $m-i$ points to the last cluster. Reasoning only on the number of points to be assigned, we always have the following relation:

$$V(C_1..C_c, m) \geq \min_{i \in [0,m]} (V(C_1..C_{c-1}, i) + V(C_c, m-i))$$

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

$$\underline{V}(C_1..C_c, m) = \min_{i \in [0,m]} (\underline{V}(C_1..C_{c-1}, i) + \underline{V}(C_c, m-i)) \tag{2}$$

This is computed by a dynamic program for all $c \in [2, k]$ and $m \in [0, q]$ (lines 20–23 in Algorithm 1). Let us notice that with (2), for all $c \in [1, k]$ and $m \in [0, q]$:

$$\underline{V}(C_1..C_c, m) = \min_{m_1 + .. + m_c = m} (\underline{V}(C_1, m_1) + \cdots + \underline{V}(C_c, m_c)) \tag{3}$$

Let us reconsider the example given in Figure 1. The value $\underline{V}(C_1..C_3, 4)$ computed by (2) corresponds to the case $\underline{V}(C_1..C_2, 1) + \underline{V}(C_3, 3)$, i.e when one point is assigned to the clusters C_1, C_2 and 3 points are assigned to C_3. The value $\underline{V}(C_1..C_2, 1)$ corresponds to the case $\underline{V}(C_1, 0) + \underline{V}(C_2, 1)$. The value $\underline{V}(C_2, 1)$ corresponds to the case where point 4 is assigned to cluster C_2 and $\underline{V}(C_3, 3)$ to the case where points 4, 8 and 9 are assigned to cluster C_3. We note that in this lower bound, point 4 is considered twice and point 3 is not considered.

Concerning the complexity, the complexity of the first loop (lines 1–10) is $O(q(k + n + q \log q + q)) = O(q^2 \log q + qn)$. The complexity of the second loop (lines 11–19) is $O(kq(q + q \log q)) = O(kq^2 \log q)$ and the complexity of the last loop (lines 20–23) is $O(kq^2)$. The complexity of Algorithm 1 is then $O(kq^2 \log q + qn)$. Let us notice that in clustering tasks, the number of clusters k is usually constant or much smaller than n.

4.2 Filtering Algorithm

The filtering algorithm for the constraint *wcss*$(\mathcal{G}, V, d)$ is presented in Algorithm 2, given a partial assignment of variables in $\mathcal{G}$. The value $\underline{V}(C_1..C_k, q)$ in Algorithm 1 represents a lower bound for the sum of squares V, for all possible assignments of all the points in U into the clusters $C_1, \ldots, C_k$. Let $[V.lb, V.ub)$ be the actual domain of V, where $V.lb$ is the lower bound, which can be initially 0, and $V.ub$ is the upper bound, which can be either $+\infty$ or the value of V in the previous solution. The upper bound is strict since in a branch-and-bound search the next solution must be strictly better than the previous solution. The lower bound $V.lb$ is then set to $\max(V.lb, \underline{V}(C_1..C_k, q))$. We present below the filtering of unassigned decision variables in $\mathcal{G}$.

For each value $c \in [1, k]$, for each unassigned variable G_i, if $c \in Dom(G_i)$, with the assumption of assigning point i into the cluster C_c, we compute a new lower bound of V. Let C'_c be the cluster $C_c \cup \{i\}$ and let $\mathcal{C}' = \{C_l \mid l \neq c\} \cup \{C'_c\}$. A new lower bound V' of V is the value $\underline{V}(\mathcal{C}', q-1)$, since there still remain $q-1$ points of $U\backslash\{i\}$ to be assigned to the k clusters. According to (2):

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

For all $m \in [0, q-1]$, we revise the lower bounds $\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m)$ and $\underline{V}(C'_c, m)$ by exploiting informations constructed by Algorithm 1. The revision will be detailed in the remainder of this subsection. The new lower bound V' is computed by line 8 of Algorithm 2. Therefore, since $Dom(V) = [V.lb, V.ub)$, if $V' \geq V.ub$, the variable G_i cannot take the value c. The value c is then removed from $Dom(G_i)$ (lines 9–10). The complexity of Algorithm 2 is the complexity of computing the lower bound $O(kq^2 \log q + qn)$ plus the complexity of the loop (lines 2–10) $O(kq^2)$. The overall complexity therefore is $O(kq^2 \log q + qn)$.

Algorithm 2: Filtering of $wcss(\mathcal{G}, V, d)$

```
   input : a partial assignment of $\mathcal{G}$, a set $U = \{i \mid G_i$ unassigned$\}$, $q = |U|$
1  $V.lb \leftarrow \max(V.lb,$ Lower_bound()$)$
2  for $c \leftarrow 1$ to $k$ do
3      for $m \leftarrow 0$ to $q-1$ do
4          $\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m) \leftarrow \max_{m' \in [0,q-m]}(\underline{V}(\mathcal{C}, m+m') - \underline{V}(C_c, m'))$
5      foreach $i \in U$ such that $c \in Dom(G_i)$ do
6          for $m \leftarrow 0$ to $q-1$ do
7              $\underline{V}(C'_c, m) \leftarrow ((n_c+m)\underline{V}(C_c, m) + s_2(i,c) + s_3(i,m))/(n_c+m+1)$
8          $V' \leftarrow \min_{m \in [0,q-1]}(\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m) + \underline{V}(C'_c, q-1-m))$
9          if $V' \geq V.ub$ then
10             remove $c$ from $Dom(G_i)$
```

Computing $\underline{V}(C'_c, m)$. Let us recall that C'_c is the cluster C_c augmented by point i, and $\underline{V}(C'_c, m)$ is the lower bound of the sum of squares of C'_c after adding any m points of $U\backslash\{i\}$ into C'_c. According to (1):

$$\underline{V}(C'_c, m) = \frac{S_1(C'_c) + \underline{S_2}(C'_c, m)}{n_c + 1 + m}$$

We have $S_1(C'_c) = S_1(C_c) + s_2(i,c)$. The value of $\underline{S_2}(C'_c, m)$ can be revised from $\underline{S_2}(C_c, m)$ by:

$$\underline{S_2}(C'_c, m) = \underline{S_2}(C_c, m) + s_3(i, m)$$

According to (1), we have (line 7 of Algorithm 2):

$$\underline{V}(C'_c, m) = \frac{(n_c + m)\underline{V}(C_c, m) + s_2(i,c) + s_3(i,m)}{n_c + m + 1}$$

Computing $\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m)$. This value represents a lower bound of the sum of squares for any assignment of m points in $U\backslash\{i\}$ into the clusters different from C'_c. According to (3), for all $q' \in [m, q]$:

$$\underline{V}(\mathcal{C}, q') \ = \min_{m+m'=q'} (\underline{V}(\mathcal{C}\backslash\{C_c\}, m) + \underline{V}(C_c, m'))$$

so for all $q' \in [m, q]$ and with $m + m' = q'$, we have:

$$\underline{V}(\mathcal{C}, m + m') \ \leq \ \underline{V}(\mathcal{C}\backslash\{C_c\}, m) + \underline{V}(C_c, m')$$

which corresponds to:

$$\underline{V}(\mathcal{C}\backslash\{C_c\}, m) \geq \underline{V}(\mathcal{C}, m + m') - \underline{V}(C_c, m')$$

Since $m \leq q' \leq q$ and $m + m' = q'$, we have $0 \leq m' \leq q - m$. We then have:

$$\underline{V}(\mathcal{C}\backslash\{C_c\}, m) \geq \max_{m' \in [0, q-m]} (\underline{V}(\mathcal{C}, m + m') - \underline{V}(C_c, m'))$$

We also have:

$$\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m) \geq \underline{V}(\mathcal{C}\backslash\{C_c\}, m)$$

since $\mathcal{C}'\backslash\{C'_c\}$ and $\mathcal{C}\backslash\{C_c\}$ denote the same set of clusters, $\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m)$ is computed for any m points of $U\backslash\{i\}$, while $\underline{V}(\mathcal{C}\backslash\{C_c\}, m)$ is computed for any m points of U. Therefore we can exploit the columns computed by the dynamic program in Algorithm 1 to revise a new lower bound (line 4 in Algorithm 2):

$$\underline{V}(\mathcal{C}'\backslash\{C'_c\}, m) = \max_{m' \in [0, q-m]} (\underline{V}(\mathcal{C}, m + m') - \underline{V}(C_c, m'))$$

5 Experiments

Our model is implemented with Gecode library version 4.2.7[2], which supports float and integer variables. Experiments have been performed on a 3.0 GHz Core i7 Intel computer with 8 Gb memory under Ubuntu. All our programs are available at `http://cp4clustering.com`. We have considered the datasets Iris, Soybean and Wine from the UCI Machine Learning Repository[3]. The number of objects and the number of classes are respectively 150 and 3 for Iris, 47 and 4 for Soybean and 178 and 3

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

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

for Wine dataset. We compare our model with the approach proposed in [3], based on Integer Linear Programming and column generation and optimizing WCSS criterion with user constraints. Our approach is also compared to COP-kmeans [30] that extends k-means algorithm to integrate user constraints and to Repetitive Branch-and-Bound Algorithm (RBBA) [5], without user constraints, since this algorithm is not able to handle them.

In MiningZinc, a modeling language for constraint-based mining [15], it is shown that clustering with the WCSS criterion can be modeled[4]. The model can be translated to different backend solvers including Gecode. However, because of the intrinsic difficulty of the WCSS criterion, this example model cannot handle 14 points randomly selected from Iris dataset within 30 min, whereas our model takes 0.01s to solve them.

5.1 Optimizing WCSS in Presence of User Constraints

The most widespread constraints in clustering are must-link or cannot-link constraints, since they can be derived from partial previous knowledge (e.g. cluster labels known for a subset of objects). Therefore we choose these two kinds of constraints in the experiments. To generate user constraints, pairs of objects are randomly drawn and either a must-link or a cannot-link constraint is created depending on whether the objects belong to the same class or not. The process is repeated until the desired number for each kind of constraints is reached. For each number of constraints, five different constraint sets are generated for the tests. In each test, we compute the WCSS value, the Rand index of the solution compared to the ground truth partition and the total run-time. The Rand index [25] measures the similarity between two partitions, P and P^*. It is defined by $RI = (a+b)/(a+b+c+d)$, where a and b are the number of pairs of points for which P and P^* are in agreement (a is the number of pairs of points that are in the same class in P and in P^*, b is the number of pairs of points that are in different classes in P and in P^*), c and d are the number of pairs of points for which P and P^* disagree (same class in P but different classes in P^* and vice versa). This index varies between 0 and 1 and the better the partitions are in agreement, the closer to 1. Since experiments are performed on 5 sets of constraints, the mean value μ and the standard deviation σ are computed for run-time and RI. A timeout is set to 30 min. A minus sign (-) in the tables means that the timeout has been reached without completing the search. Since the ground truth partition is used to generate user constraints, experiments are done with k equal to the ground truth number of classes for each dataset.

Table 1 gives results for our model (CP) and for the approach based on ILP and column generation (ILP) [3] for the Iris dataset with different numbers $\#c$ of must-link constraints. For both the execution time and the Rand index, the mean value of the five tests and the coefficient of variation (σ/μ) are reported. Since the two approaches are exact, they find partitions having the same WCSS value. It can be noticed that must-link constraints help to improve quality of the solution as well as to reduce the execution time for this dataset. Our approach can find a global optimal without user constraints, whereas ILP approach needs at least 100 must-link constraints to be able to prove the optimality. With more than 100 must-link constraints, our approach always takes less

[4] `http://inductiveconstraints.eu/miningzinc/examples/kmeans.mzn`

time to complete the search. Our approach is also more efficient to handle must-link

$\#c$	CP		ILP		RI	
	μ	σ/μ	μ	σ/μ	μ	σ/μ
0	888.99	0.83 %	-	-	0.879	0 %
50	332.06	78.96 %	-	-	0.940	1.66 %
100	7.09	40.18 %	62	74.24 %	0.978	1.68 %
150	0.31	36.39 %	0.45	44.55 %	0.989	0.66 %
200	0.07	24.83 %	0.11	48.03 %	0.992	0.66 %
250	0.05	10.63 %	0.06	35.56 %	0.996	0.70 %
300	0.04	9.01 %	0.04	19.04 %	0.998	0.35 %

Table 1: Time (in seconds) and RI for Iris dataset with $\#c$ must-link constraints

and cannot-link constraints. Table 2 (left) reports mean execution time in seconds for the five tests and the coefficient of variation σ/μ. In each case, the same number $\#c$ of must-link and cannot-link constraints are added. We can see that when $\#c \leq 75$, our approach can complete the search within the timeout and in the other cases, it performs better than ILP. Concerning the Wine dataset, the two approaches cannot prove the optimality of the solution in less than 30 min, when there are less than 150 must-link constraints, as shown in Table 2 (right).

$\#c$	CP		ILP	
	μ	σ/μ	μ	σ/μ
25	969.33	51.98 %	-	-
50	43.85	46.67 %	-	-
75	4.97	150 %	-	-
100	0.41	49.8 %	107	72.35 %
125	0.09	52.07 %	4.4	95.85 %
150	0.06	22.6 %	0.8	50 %

$\#c$	CP	ILP
150	6.84	12.98
200	0.11	0.32
250	0.08	0.11
300	0.08	0.06

Table 2: Time in seconds for Iris dataset with $\#c$ must-link and $\#c$ cannot-link constraints (left) and Wine dataset with $\#c$ must-link constraints (right)

Our CP approach makes better use of cannot-link constraints, as shown in Table 3. This table reports the mean time in seconds and the percentage of tests for which each system completes the search within the timeout. The execution time varies a lot, depending on the constraints. If we consider the Iris database, Table 3 (left) shows that our model is able to find an optimal solution and to prove it for roughly 60 % cases, wheres ILP can solve no cases. If we consider the Wine dataset, Table 3 (right) shows that when 100 must-link and 100 cannot-link constraints are added, CP can solve all the cases, whereas ILP cannot solve them. When 125 must-link constraints and 125 cannot-link constraints are added, both approaches can solve all the cases, but our approach is less time-consuming.

$\#c$	CP		ILP	
	μ	solved	μ	solved
50	1146.86	20 %	-	0 %
100	719.53	80 %	-	0 %
150	404.77	60 %	-	0 %
200	1130.33	40 %	-	0 %
250	172.81	60 %	-	0 %
300	743.64	60 %	-	0 %

$\#c$	CP		ILP	
	μ	solved	μ	solved
100	10.32	100 %	-	0 %
125	0.35	100 %	497.6	100 %
150	0.12	100 %	13.98	100 %

Table 3: Iris dataset with $\#c$ cannot-link constraints (left) and Wine dataset with $\#c$ must-link and $\#c$ cannot-link constraints (right)

Experiments with the Soybean dataset lead to the same observations. With a number of must-link constraints varying from 10 to 80, the mean run-times for both CP and ILP approaches decrease from 0.3 s to 0.01 s. However, with different numbers of cannot-link constraints, CP always outperforms ILP approach. For instance, the mean time is 5.19 s (CP) vs. 278.60 s (ILP) with 20 cannot-link constraints, or 2.5 s (CP) vs. 126 s (ILP) with 80 cannot-link constraints.

5.2 Comparisons with COP-kmeans and RBBA

RBBA [5] is based on a repetitive branch-and-bound strategy and finds an exact solution for WCSS. This algorithm takes 0.53 s to find and prove the optimal solution for the Iris dataset and 35.56 s for the Wine dataset. But it does not handle user constraints. Concerning the quality of the lower bound, when most of the points are unassigned, the lower bound of RBBA is better than ours. However when more points are assigned to the clusters, our lower bound can be better than the lower bound of RBBA.

On the other hand, COP-kmeans algorithm [30] extends k-means algorithm to must-link and cannot-link constraints. This algorithm is based on a greedy strategy to find a solution that satisfies all the constraints. When there are only must-link constraints, COP-kmeans always finds a partition satisfying all the constraints, which is a local optimum of WCSS. Nevertheless, when considering also cannot-link constraints, the algorithm may fail to find a solution satisfying all the constraints, even when such a solution exists.

We perform the same tests, but for each set of constraints, COP-kmeans is run 1000 times and we report the number of times COP-kmeans has been able to find a partition. Figure 2 (left) shows the percentage of successes when cannot-link constraints are added. With the two datasets Iris and Wine, COP-kmeans fails to find a partition when 150 constraints are added. Our CP model always find a solution satisfying all the constraints. For Iris dataset, our model succeeds in proving the optimality for roughly 60 % cases (Table 3 left).

Figure 2 (right) gives the results when $\#c$ must-link and $\#c$ cannot-link constraints are added. For Wine dataset, COP-kmeans always fail to find a partition when $\#c = 75$, 125 or 150. Our CP approach finds a solution satisfying all the constraints in all the cases. It completes the search in all the cases for Iris dataset, as shown in Table 2 (left), and in all the cases where $\#c \geq 100$ for Wine dataset, as shown in Table 3 (right).

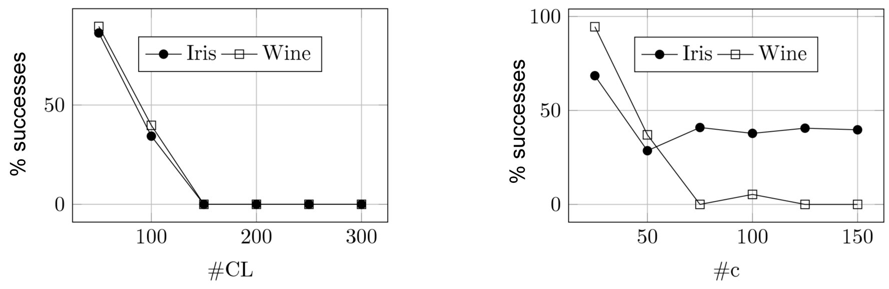

Fig. 2. COP-kmeans with cannot-link (left), with $\#c$ must-link and $\#c$ cannot-link constraints (right)

6 Conclusion

In this paper we address the well-known WCSS criterion in cluster analysis. We develop a new global optimization constraint *wcss* and present a filtering algorithm, which filters not only the domain of the objective variable but also those of decision variables. This constraint integrated to our CP framework [7, 8] extends it to model constrained minimum sum of squares clustering tasks. Experiments on classic datasets show that our framework outperforms the state-of-the-art best exact approach, which is based on Integer Linear Programming and column generation [3].

Working on search strategies and on constraint propagation enables to improve substantially the efficiency of our CP model. We continue studying these aspects to make the framework able to deal with larger datasets. We are working on exploiting user constraints inside the filtering algorithm, either by using connected components or by modifying the dissimilarities according to the user constraints. We exploit the flexibility of the CP framework to offer a choice between exact or approximate solutions, by studying the use of approximate search strategies, such as local search methods.

References

1. Aloise, D., Deshpande, A., Hansen, P., Popat, P.: NP-hardness of Euclidean Sum-of-squares Clustering. Machine Learning 75(2), 245–248 (2009)
2. Aloise, D., Hansen, P., Liberti, L.: An improved column generation algorithm for minimum sum-of-squares clustering. Mathematical Programming 131(1-2), 195–220 (2012)

3. Babaki, B., Guns, T., Nijssen, S.: 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. pp. 438–454 (2014)
4. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of the 21st International Conference on Machine Learning. pp. 11–18 (2004)
5. Brusco, M., Stahl, S.: Branch-and-Bound Applications in Combinatorial Data Analysis (Statistics and Computing). Springer, 1 edn. (2005)
6. Brusco, M.J.: An enhanced branch-and-bound algorithm for a partitioning problem. British Journal of Mathematical and Statistical Psychology 56(1), 83–92 (2003)
7. Dao, T.B.H., Duong, K.C., Vrain, C.: A Declarative Framework for Constrained Clustering. In: Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. pp. 419–434 (2013)
8. Dao, T.B.H., Duong, K.C., Vrain, C.: Constrained clustering by constraint programming. Artificial Intelligence p. DOI: 10.1016/j.artint.2015.05.006 (2015)
9. Davidson, I., Ravi, S.S.: Clustering with Constraints: Feasibility Issues and the k-Means Algorithm. In: Proceedings of the 5th SIAM International Conference on Data Mining. pp. 138–149 (2005)
10. Davidson, I., Ravi, S.S.: The Complexity of Non-hierarchical Clustering with Instance and Cluster Level Constraints. Data Mining Knowledge Discovery 14(1), 25–61 (2007)
11. De Raedt, L., Guns, T., Nijssen, S.: Constraint Programming for Data Mining and Machine Learning. In: Proc. of the 24th AAAI Conference on Artificial Intelligence (2010)
12. Edwards, A.W.F., Cavalli-Sforza, L.L.: A method for cluster analysis. Biometrics 21(2), 362–375 (1965)
13. Focacci, F., Lodi, A., Milano, M.: Cost-based domain filtering. In: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming. pp. 189–203 (1999)
14. Gonzalez, T.: Clustering to minimize the maximum intercluster distance. Theoretical Computer Science 38, 293–306 (1985)
15. Guns, T., Dries, A., Tack, G., Nijssen, S., De Raedt, L.: Miningzinc: A modeling language for constraint-based mining. In: IJCAI (2013)
16. Guns, T., Nijssen, S., De Raedt, L.: k-Pattern set mining under constraints. IEEE Transactions on Knowledge and Data Engineering 25(2), 402–418 (2013)
17. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn. (2011)
18. Hansen, P., Jaumard, B.: Cluster analysis and mathematical programming. Mathematical Programming 79(1-3), 191–215 (1997)
19. Jensen, R.E.: A dynamic programming algorithm for cluster analysis. Journal of the Operations Research Society of America 7, 1034–1057 (1969)
20. Koontz, W.L.G., Narendra, P.M., Fukunaga, K.: A branch and bound clustering algorithm. IEEE Trans. Comput. 24(9), 908–915 (1975)
21. Law, Y.C., Lee, J.H.M.: Global constraints for integer and set value precedence. In: Wallace, M. (ed.) Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming. pp. 362–376 (2004)
22. du Merle, O., Hansen, P., Jaumard, B., Mladenovic, N.: An interior point algorithm for minimum sum-of-squares clustering. SIAM Journal on Scientific Computing 21(4), 1485–1505 (1999)
23. B.J. van Os, J.M.: Improving Dynamic Programming Strategies for Partitioning. Journal of Classification (2004)

24. Pelleg, D., Baras, D.: K-means with large and noisy constraint sets. In: Machine Learning: ECML 2007. Lecture Notes in Computer Science, vol. 4701, pp. 674–682. Springer Berlin Heidelberg (2007)
25. Rand, W.M.: Objective Criteria for the Evaluation of Clustering Methods. Journal of the American Statistical Association 66(336), 846–850 (1971)
26. Régin, J.C.: Arc consistency for global cardinality constraints with costs. In: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming. pp. 390–404 (1999)
27. Steinley, D.: k-means clustering: A half-century synthesis. British Journal of Mathematical and Statistical Psychology 59(1), 1–34 (2006)
28. Ugarte, W.R., Boizumault, P., Loudni, S., Crémilleux, B., Lepailleur, A.: Mining (soft-) skypatterns using dynamic CSP. In: Proceedings of the 11th International Conference on Integration of AI and OR Techniques in Constraint Programming. pp. 71–87 (2014)
29. Wagstaff, K., Cardie, C.: Clustering with instance-level constraints. In: Proceedings of the 17th International Conference on Machine Learning. pp. 1103–1110 (2000)
30. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S.: Constrained K-means Clustering with Background Knowledge. In: Proceedings of the 18th International Conference on Machine Learning. pp. 577–584 (2001)