Repetitive Branch-and-Bound using Constraint Programming for Constrained Minimum Sum-of-Squares Clustering

Tias Guns[1] and **Thi-Bich-Hanh Dao**[2] and **Christel Vrain**[2] and **Khanh-Chuong Duong**[2]

Abstract. Minimum sum-of-squares clustering (MSSC) is a widely studied task and numerous approximate as well as a number of exact algorithms have been developed for it. Recently the interest of integrating prior knowledge in data mining has been shown, and much attention has gone into incorporating user constraints into clustering algorithms in a generic way.

Exact methods for MSSC using integer linear programming or constraint programming have been shown to be able to incorporate a wide range of constraints. However, a better performing method for unconstrained exact clustering is the Repetitive Branch-and-Bound Algorithm (RBBA) algorithm. In this paper we show that both approaches can be combined. The key idea is to replace the internal branch-and-bound of RBBA by a constraint programming solver, and use it to compute tight lower and upper bounds. To achieve this, we integrate the computed bounds into the solver using a novel constraint. Our method combines the best of both worlds, and is generic as well as performing better than other exact constrained methods. Furthermore, we show that our method can be used for multi-objective MSSC clustering, including constrained multi-objective clustering.

1 INTRODUCTION

Cluster analysis or clustering is an important task in data mining, which has various applications in different domains such as biology, chemistry, medicine or business. Given a set of objects, cluster analysis aims at partitioning the objects into homogeneous subsets, called clusters. The homogeneity is usually formulated by an optimization criterion. One of the most used criterion is minimizing the Within-Cluster Sum of Squares (WCSS), which is defined by the sum of the squared Euclidean distances from each object to the centroid of the cluster to which it belongs. In order to make the clustering task more accurately fit the problem at hand, prior user knowledge has been integrated into the clustering process by means of user-defined constraints.

Minimum sum-of-squares clustering (MSSC) has been proven to be NP-Hard [1] and has been studied in numerous works. The well-known k-means algorithm as well as other dedicated heuristic algorithms find a local optimal for this criteria [21]. They have been also extended to integrate different user constraints but they can fail to find a solution that satisfies all the constraints even when such a solution exists. On the other hand, general and declarative frameworks using generic optimization tools offer the flexibility of handling a wide variety of user constraints, and finding an exact solution of the problem whenever one exists. As a consequence, this precludes the use of these approaches on large datasets, but finding an exact solution may be of high importance on small but valuable datasets. Different frameworks have been proposed, based either on Integer Linear Programming with column generation [4] or on Constraint Programming [9].

On the other hand, Brusco [6] proposed a simple yet effective method for unconstrained MSSC: the Repetitive Branch-and-Bound Algorithm (RBBA). It computes increasingly tight bounds on the MSSC score by repetitively searching for the optimal solution, starting from a small subset of points up to the full set of all points. In this work we show how the idea of clustering with RBBA can be combined with the ideas of clustering with constraint programming [9].

Our contributions are as follows:

- We extend RBBA using Constraint Programming (CP) to support user-defined constraints. The key idea is to use CP in each branch-and-bound step and we show that this eases the modeling of a range of user constraints;
- The use of CP enables the computation of (constrained) lower bounds and upper bounds for the non-linear MSSC, and we develop a novel CP constraint that incorporates these bounds;
- We show that the resulting method is generic yet better performing than other exact constrained clustering methods.
- We experimentally illustrate the interest of our framework by its use in a multi-objective constrained clustering setting that minimizes WCSS and maximizes the split between clusters. To the best of our knowledge, this framework is the first one to support this bi-criterion clustering and different kinds of user-constraints.

Outline. Section 2 gives the preliminaries and Section 3 reviews related work. Section 4 presents RBBA and the extension we propose to integrate user constraints. Section 5 presents a framework using CP to achieve the extension of RBBA. Section 6 is devoted to the experiments and comparisons of our method with other existing approaches. Section 7 discusses perspectives and concludes.

2 PRELIMINARIES

Let us consider a dataset of N objects $\mathcal{O}$ in an Euclidean space. Let d be the Euclidean distance ($d(o, o') = ||o - o'||$). Minimum Sum-of-Squares Clustering (MSSC) aims at finding a partition Δ of the

[1] KU Leuven, Department of Computer Science, Celestijnenlaan 200A, Leuven, Belgium, email: tias.guns@cs.kuleuven.be
[2] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067, Orléans, France, email: thi-bich-hanh.dao@univ-orleans.fr, khanh-chuong.duong@univ-orleans.fr, christel.vrain@univ-orleans.fr

objects into K clusters $C_1, ..., C_K$ such that: (1) $\forall k \in \{1, \ldots, K\}$, $C_k \neq \emptyset$, (2) $\bigcup_k C_k = \mathcal{O}$, (3) $\forall k \neq k'$, $C_k \cap C_{k'} = \emptyset$ and (4) the Within-Cluster Sum of Squares (WCSS) is minimized. The WCSS criterion is defined by:

$$WCSS(\Delta) = \sum_{k \in \{1,\ldots,K\}} \sum_{o \in C_k} d(o, m_k)^2 \qquad (1)$$

where for each $k \in [1, K]$, m_k is the centroid (mean) of the cluster C_k. Equivalently [14, 16]:

$$WCSS(\Delta) = \sum_{k \in \{1,\ldots,K\}} \frac{1}{2|C_k|} \sum_{o,o' \in C_k} d(o, o')^2 \qquad (2)$$

There exists other optimization criteria, such as minimizing the Within-Cluster Sum of Dissimilarities criterion ($WCSD = \sum_{k \in \{1,\ldots,K\}} \sum_{o,o' \in C_k} d(o, o')$), minimizing the maximal diameter D of the clusters ($maxDiam = \max_{k \in \{1,\ldots,K\}} \max_{o,o' \in C_k} d(o, o')$) or maximizing the minimal split S between clusters ($minSplit = \min_{k,k' \in \{1,\ldots,K\}, k \neq k'} \min_{o \in C_k, o' \in C_{k'}} d(o, o')$).

In applications, the user can have prior knowledge or requirements on the objects. For instance, the labels of a subset of objects can be known or an upper bound on the number of objects in each cluster can be required. Prior knowledge is integrated into the clustering process by user-defined constraints that have to be satisfied. User constraints can be *instance-level*, specifying requirements on pairs of objects, or *cluster-level*, giving requirements on the clusters. Instance-level constraints, introduced first in [25], are used most often. They are either must-link (ML) or cannot-link (CL) constraints on pairs of objects, which states that the objects must be or cannot be in the same cluster. Different kinds of cluster-level constraints also exist, the most popular ones being:

- A diameter constraint sets an upper bound γ on the cluster diameter: $\forall k \in \{1, \ldots, K\}, \forall o, o' \in C_k, d(o, o') \leq \gamma$. This constraint can be expressed by cannot-link constraints: each pair of objects o, o' having $d(o, o') > \gamma$ must be in different clusters.
- A split constraint sets a lower bound δ on the separation between clusters: $\forall k \neq k' \in \{1, \ldots, K\}$, $\forall o \in C_k$, $\forall o' \in C_{k'}$, $d(o, o') \geq \delta$. This constraint can be expressed by must-link constraints: each pair of objects o, o' having $d(o, o') < \delta$ must be in the same cluster.
- A density constraint requires that each object has in its neighborhood of radius ϵ at least m objects belonging to the same cluster as itself: $\forall k \in \{1, \ldots, K\}$, $\forall o \in C_k$, $\exists o_1, .., o_m \in C_k \setminus \{o\}$, $d(o, o_i) \leq \epsilon$, or at least $m\%$ objects: $\forall k \in \{1, \ldots, K\}, \forall o \in C_k$, $\frac{|\{o_i \in C_k | d(o,o_i) \leq \epsilon\}|}{|\{o_i \in \mathcal{O} | d(o,o_i) \leq \epsilon\}|} \geq \frac{m}{100}$.
- A minimal (maximal) capacity constraint requires each cluster to have at least (at most, resp.) a given α (β, resp.) number of objects: $\forall k \in \{1, \ldots, K\}$, $|C_k| \geq \alpha$ (or $|C_k| \leq \beta$, resp.).

Constraint Programming (CP) is a constraint-based satisfaction and optimization framework. A constraint optimisation problem is expressed as a quadruple (V, D, C, f) where V is a set of *variables* and each variable $v \in V$ must take a value from its *domain* $D(v)$. The set C is a set of constraints over (a subset of) the variables V. The function f is an objective function defined over V and a solution that maximizes/minimizes f is an optimal solution.

Typical constraint solvers use depth-first branch-and-bound search. Each node in the search tree represents a partial solution consisting of a domain D' where $\forall v \in V : D'(v) \subseteq D(v)$. In each node of the search tree, the constraint solver tries to *propagate* each constraint. Propagation is achieved when a constraint reduces the domains of its variables by removing those values that violate the constraint. For example, a constraint $X > 2$ can remove from $D(X) = \{1, 2, 3, 4, 5\}$ the values 1 and 2. Constraint solvers contain many different constraints, from logical to arithmetic and domain-specific constraints, such as for scheduling, each with its own propagation algorithm. If a propagator detects that the current partial solution cannot be extended to a full solution, namely when the domain of a variable becomes empty, the search backtracks. A solution is reached when the domain of each variable is reduced to a single value: $\forall v \in V : |D(v)| = 1$ and none of the constraints is violated. When a solution is reached, a new bound on the objective function is added stating that the next solution must score better than the currently best solution. Due to this branch-and-bound search, constraint solvers are exact: the search stops when it has proven that no better solution exists.

3 RELATED WORK

Constrained Minimum Sum-of-Squares Clustering has been studied in both heuristic and exact approaches. Among the heuristic approaches, even in the case without user constraints, the k-means algorithm as well as numerous other heuristic algorithms find a local optimal [21]. Considering must-link and cannot-link constraints, the k-means algorithm has been extended to COP-kmeans [26] or LCVQE [20]. However, when the number of constraints increases, such algorithms either fail to find a solution satisfying all the constraints even if one exists, or they find solutions that do not satisfy all the constraints.

Exact approaches for MSSC without user constraints use branch-and-bound search [18, 5, 6], dynamic programming [17, 23], Integer Linear Programming (ILP) and column generation [13, 3], a cutting plane algorithm [29] or a branch-and-cut semi-definite programming [2]. There exists few exact methods for MSSC that can handle user constraints [4, 9]. They are based on a generic optimization tool, so that different kinds of user constraints can be expressed. Extending [3], a framework based on ILP and column generation has been proposed in [4]. Using Constraint Programming (CP), a generic framework has been developed in [9], with a global constraint to compute and prune the search space for the WCSS criterion of MSSC.

Constrained clustering settings using an objective function different from WCSS have also been developed. A framework using ILP is proposed in [19]; it requires a set of clusters to be given in advance and considers different criteria to choose the best clustering from candidate clusters. A SAT based framework has been developed for constrained clustering for the diameter and the split criteria [11]. A well-performing CP based framework is developed in [7, 8] that includes diameter, split and sum of squared distances criteria, as well as user constraints.

Our work extends the Repetitive Branch-and-Bound Algorithm (RBBA) [6]. This algorithm finds a global optimal for MSSC without user constraints. We show that the methodology can be combined with a CP framework to obtain an efficient method that can easily incorporate user constraints.

4 EXTENDING RBBA TO USER-CONSTRAINTS

We first explain the bound used in RBBA and the standard RBBA algorithm. We then show the validity of the bounds under user constraints and how to extend the algorithm to support constraints in a generic way.

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

4.1 Lower Bound Inequalities Without User-Constraints

The bounds used in RBBA rely on the following result [18]. Let S be a subset of $\mathcal{O}$, and let S_1 and S_2 be such that $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$ (non-overlapping). We have:

$$WCSS(\Delta_S) \geq WCSS(\Delta_{S_1}) + WCSS(\Delta_{S_2}) \qquad (3)$$

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

$$WCSS(\Delta_{S_2}) \geq WCSS^*(S_2) \qquad (4)$$

and hence [6]:

$$WCSS(\Delta_S) \geq WCSS(\Delta_{S_1}) + WCSS^*(S_2) \qquad (5)$$

Eq. (5) can be used during the search for an optimal partition of S as follows. Let us suppose that we have previously built a partition of S, thus giving an upper bound for $WCSS^*(S)$, that we have currently built a partial solution Δ_{S_1} and that we know an optimal solution of $WCSS^*(S_2)$. If $WCSS(\Delta_{S_1}) + WCSS^*(S_2)$ is greater than the actual upper bound, then the partial solution Δ_{S_1} can never lead to a better solution than the current upper bound.

4.2 Repetitive Branch-and-Bound Algorithm

The Repetitive Branch-and-Bound Algorithm (RBBA) [6] is presented in Algorithm 1.

Algorithm 1: RBBA *input: objects $\mathcal{O}$, number clusters K*

```
1  OrderPoints($\mathcal{O}$)
2  $\mathcal{O}_K \leftarrow \{o_{N-K+1}, \dots, o_N\}$
3  $\Delta^*_K \leftarrow$ Init($\mathcal{O}_K$)
4  $W_n \leftarrow 0, \forall n \in \{1, \dots, K\}$
5  for $n = K + 1$ to $N$ do
6      $\mathcal{O}_n \leftarrow \mathcal{O}_{n-1} \cup \{o_{N-n+1}\}$
7      $\Delta_n \leftarrow$ Greedy_Extension($\mathcal{O}_n, \Delta^*_{n-1}$)
8      $U_n \leftarrow WCSS(\Delta_n)$
9      $\Delta^*_n \leftarrow$ BaB_Search($\mathcal{O}_n, U_n, W$)
10     $W_n \leftarrow WCSS(\Delta^*_n)$
```

Points in $\mathcal{O}$ are first ordered following an heuristic by *OrderPoints($\mathcal{O}$)*. Different heuristics can be used for ordering points, they will be presented in Subsection 4.5. We assume that according to the ordering, points are named by their index $i \in [1, N]$. $\mathcal{O}_n$ is composed of the *last* n points according to this order.

In this algorithm, Δ_n indicates any partition of $\mathcal{O}_n$ into at most K clusters and Δ^*_n denotes the optimal partition of $\mathcal{O}_n$ into at most K clusters. This algorithm starts with the set $\mathcal{O}_K$ of the last K points and *Init($\mathcal{O}_K$)* creates Δ^*_K by putting each point alone in a cluster. The optimal value $WCSS(\Delta^*_n)$ is stored in W_n for each n, and the first K values $W_1, \dots, W_K$ are 0 (each point in its own cluster).

The algorithm next iterates by adding to the set $\mathcal{O}_n$ one point each time, from the point $N-K$ down to the first point; $\mathcal{O}_n$ represents this set of last n points $o_{N-n+1}, \dots, o_N$; *Greedy_Extension($\mathcal{O}_n, \Delta^*_{n-1}$)* greedily finds a partition Δ_n for $\mathcal{O}_n$, by adding the new point to the previous best partition Δ^*_{n-1} so that the value WCSS is minimally increased. The value $WCSS(\Delta_n)$ constitutes an upper bound U_n for $WCSS(\Delta^*_n)$. *BaB_Search($\mathcal{O}_n, U_n, W$)* is a branch-and-bound algorithm which searches for a global optimal partition Δ^*_n on the set of points $\mathcal{O}_n$, using U_n as an upper bound and exploiting Eq. (5) with the W_i values ($i < n$) as lower bounds. Let $o_m = o_{N-n+1}$ be the new point added at this step. The branch-and-bound search considers the points in $\mathcal{O}_n$ in the order $o_m, o_{m+1}, \dots, o_N$ and tries to assign them to clusters.

Let us consider an arbitrary step when a point number p ($m \leq p < N$) is assigned to a cluster. Let S_1 be the set of points $\{o_m, ..., o_p\}$ and S_2 be $\{o_{p+1}, ..., o_N\}$. All the points in S_1 have already been assigned and hence $WCSS(\Delta_{S_1})$ is known. All the points in S_2 are currently unassigned, however, $WCSS^*(S_2)$ has been computed in a previous step of RBBA and stored in $W_{|S_2|}$; U_n is the current upper bound. Eq. (5) is used and if $WCSS(\Delta_{S_1}) + WCSS^*(S_2) \geq U_n$, we cannot extend Δ_{S_1} to a solution having WCSS better than U_n. Therefore *BaB_Search* will not continue to extend Δ_{S_1} and the branch is pruned. When $p = N$, the partition Δ is complete and U_n is set to $WCSS(\Delta)$. When the entire search space is explored, the last complete partition found is the optimal solution.

This algorithm takes advantage of the optimal solutions previously computed to provide lower bounds in the branch-and-bound search. Also important are the upper bounds found by the greedy extension, they are often tight (meaning that the greedy extension is the optimal partitioning). Because of these tight bounds, even though the algorithm runs the branch-and-bound search N times, it is nevertheless one of the best exact algorithms for minimum sum-of-squares clustering. A similar search method was proposed for valued (soft) CSPs with an additive objective function, called Russian Doll Search [24].

4.3 Lower Bound Inequalities With User-Constraints

We now study the conditions under which Eq. (5) is still valid in the presence of a set of user constraints $\mathcal{C}$ on $\mathcal{O}$. Given a set of points $S \subseteq \mathcal{O}$ and a set of constraints C on S, $\mathcal{S}(S, C)$ denotes the set of all partitions Δ_S of S satisfying C. We denote by $WCSS^*(S, \mathcal{C})$ the optimal WCSS of $S \subseteq \mathcal{O}$ under constraint set $\mathcal{C}$, that is, $WCSS^*(S, \mathcal{C}) = \min(\{WCSS(\Delta_S) | \Delta_S \in \mathcal{S}(S, \mathcal{C})\})$. We denote by $WCSS(\Delta_S, \mathcal{C})$ the WCSS value of a partition Δ_S under the condition that it satisfies the constraint set $\mathcal{C}$.

One can see from this that Eq. (4) still holds when considering a set of constraints $\mathcal{C}$: $WCSS(\Delta_S, \mathcal{C}) \geq WCSS^*(S, \mathcal{C})$. Indeed, any $\Delta_S \in \mathcal{S}(S, \mathcal{C})$ will have a score equal or worse than the optimal one satisfying $\mathcal{C}$.

The main question is then under what conditions Eq. (3), and hence (5), holds in the presence of constraints. Eq. (3) is always true, but the difficulty is that when considering a projection Δ_{S_i} of Δ_S with $S_i \subset S$, some constraints may become ill-defined or even be violated for Δ_{S_i}, even if they are satisfied by Δ_S. For instance, let us consider 5 points $\{a, b, c, d, e\}$, two cannot-link constraints $CL(a, b)$ and $CL(b, c)$ and a minimal size constraint specifying that each class must have at least 2 points. Let $\Delta_S = \{\{a, c\}, \{b, d, e\}\}$, $S_1 = \{a, b\}$ and $S_2 = \{c, d, e\}$. Then $\Delta_{S_1} = \{\{a\}, \{b\}\}$ and

$\Delta_{S_2} = \{\{c\},\{d,e\}\}$. The constraint $CL(a,b)$ is satisfied on S_1 whereas $CL(b,c)$ is undefined on both S_1 and on S_2. Moreover the minimal size constraint is satisfied on Δ_S but it is no longer satisfied on S_1, nor on S_2. The question is hence, given a set of constraints $\mathcal{C}$ on S which Δ_S satisfies, what set of constraints $\mathcal{C}_{S_i}$ can be put on S_1 and S_2 such that Eq. (5) is still valid?

In general, given a set of C of constraints put on objects of S, we can restrict the set C_{S_i} with $S_i \subseteq S$ to those constraints for which all objects in the constraint are in the set S_i. For example, one can add to S_i all instance-level constraints whose two objects are both in S_i. In the previous example, $CL(a,b)$ can be considered on S_1 whereas $CL(b,c)$ cannot. If a partition Δ_S satisfies a set of constraints C, then its projection onto S_i (Δ_{S_i}) will satisfy the subset of constraints C_{S_i}. Therefore

$$WCSS(\Delta, C) \geq WCSS(\Delta_{S_1}, C_{S_1}) + WCSS^*(S_2, C_{S_2}) \quad (6)$$

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

In order to incorporate some cluster-level constraints, we distinguish those that are anti-monotonic from those that are not. A constraint c is said to be anti-monotonic if when satisfied by a partition Δ_S, it is satisfied by all the projections Δ_{S_i}, with $S_i \subseteq S$. In other words, let v_c be the function that tests whether c is satisfied on a partition. Then an anti-monotonic constraint satisfies the following property: if Δ is a partition on S and $S_i \subseteq S$ then $v_c(\Delta_{S_i}) \geq v_c(\Delta)$. As an example, a maximal size constraint is anti-monotonic whereas a minimal size constraint is not.

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

$$WCSS(\Delta, C) \geq WCSS(\Delta_{S_1}) + WCSS(\Delta_{S_2}) \quad (7)$$
$$\geq WCSS(\Delta_{S_1}, C_{S_1} \cup C_a) + WCSS^*(S_2, C_{S_2} \cup C_a) \quad (8)$$

A constraint solver can additionally reason over *partial* solutions, namely over the domain of a set of variables. A constraint solver is guaranteed not to reject a partial solution that can be extended to a full solution, while it can reject partial solutions that provably can not satisfy a constraint (such as an anti-monotonic constraint and more). This will ease searching for a partial solution Δ_{S_1} in branch-and-bound search, without needing to identify $C_{S_1} \cup C_a$ each time S_1 changes.

4.4 RBBA with User Constraints

Let $\mathcal{C}$ be the set of all constraints on $\mathcal{O}$. We assume that the set $\mathcal{C}$ is satisfiable on $\mathcal{O}$, ie. there exists a partition Δ of $\mathcal{O}$ that satisfies $\mathcal{C}$. The extension of RBBA to incorporate user constraints is presented in Algorithm 2.

After ordering points, Algorithm 2 constructs an initial partition Δ_K of *at most* K clusters taking constraints $\mathcal{C}_K = \mathcal{C}_{\mathcal{O}_k}$ into account. It does so by putting each point that can be in its own cluster in a separate cluster (if there is a must-link, the two points must be put in the same cluster). Among all such partitions, the one with smallest $WCSS(\Delta_K)$ is chosen. Since $\mathcal{C}$ is satisfiable on $\mathcal{O}$, the partition Δ^*_K must exist.

At each step n, for the set $\mathcal{O}_n$ of the last n points, Algorithm 2 searches in the solution space $\mathcal{S}(\mathcal{O}_n, \mathcal{C}_n)$. There are different options for the constraint set $\mathcal{C}_n$. As discussed in the previous section, $\mathcal{C}_n$ can be $\mathcal{C}_{\mathcal{O}_n}$ or $\mathcal{C}_{\mathcal{O}_n} \cup \mathcal{C}_a$. We note that the more constraints that are considered at one step, the tighter the lower bound for the next step would be. At the last step, when $\mathcal{O}_N = \mathcal{O}$, the full set of user constraints $\mathcal{C}$, anti-monotonic or not, will be considered.

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

Algorithm 2: Extended RBBA
input: objects $\mathcal{O}$, number clusters K, constraint set $\mathcal{C}$

```
1  OrderPoints($\mathcal{O}$)
2  $\mathcal{O}_K \leftarrow \{o_{N-K+1}, \ldots, o_N\}$
3  $\Delta^*_K \leftarrow$ Init($\mathcal{O}_K, \mathcal{C}_K$)
4  $W_K \leftarrow WCSS(\Delta^*_K)$
5  for $n = K+1$ to $N$ do
6      $\mathcal{O}_n \leftarrow \mathcal{O}_{n-1} \cup \{o_{N-n+1}\}$
7      $\Delta_n \leftarrow$ Feasible_Extension($\mathcal{O}_n, \mathcal{C}_n, \Delta^*_{n-1}$)
8      if $\Delta_n$ exists then
9          $U_n \leftarrow WCSS(\Delta_n)$
10     else
11         $U_n \leftarrow \infty$
12     $\Delta^*_n \leftarrow$ Constrained_BaB($\mathcal{O}_n, \mathcal{C}_n, U_n, W$)
13     $W_n \leftarrow WCSS(\Delta^*_n)$
```

4.5 Ordering of Points

Algorithms 1 and 2 start by ordering points and they do branch-and-bound for an increasing set of points following this order. Different orders can be used. In RBBA [6], the nearest-neighbor separation heuristic is used: at each step of the ordering, the two points that have the smallest distance among all pairs of points are withdrawn from the set of points and are placed at opposite ends in the ordering. This heuristic is aimed at putting *easy-to-cluster* points near the end of the RBBA process, to avoid introducing disruptive points near the end of the process and hence having to do much search there.

The ordering that we will use is based on the furthest-point-first (FPF) algorithm [15]. This algorithm starts by choosing the furthest point from all points and stores it as the first point in the ordering. It then assigns this point as the *head* of all other points. At each iteration, the point i that is the furthest to its head is marked as the next point in the order, and all the unmarked points that are closer to i than to their head change their head to i. This ordering tends to put points that are far from each other early in the ordering, also aiming to consider *disruptive* points earlier in the process.

5 A FRAMEWORK USING CONSTRAINT PROGRAMMING

We present a framework to achieve Algorithm 2. In this framework, CP is used both to do complete branch-and-bound search for each clustering step (*Constrained_Bab*) and to construct a feasible clustering if one exists (*Feasible_Extension*). We also present improvements for enhancing the computation of lower and upper bounds.

5.1 A Basic CP Model for Constrained_BaB

Constrained_BaB($\mathcal{O}_n, \mathcal{C}_n, U_n, W$) in Algorithm 2 aims at finding a clustering Δ_n^* on $\mathcal{O}_n$ that satisfies $\mathcal{C}_n$ and that minimizes the sum-of-squares WCSS.

The CP model for this task is inspired by the model for constrained clustering in [8], the main difference being the objective. In order to define the assignment of points to clusters, integer value variables $G_1, \ldots, G_n$ with $Dom(G_i) = \{1, .., K\}$ are introduced. $G_i = k$ means that point i is assigned to the cluster number k. This formulation ensures that a point can never belong to two clusters. A complete assignment of the variables G_i therefore defines a partitioning. However, different assignment can represent the same partitioning but with a permutation on the cluster indices used. In order to break this kind of symmetry and to enforce that each partition corresponds to one complete assignment, the CP constraint *precede*(G, $[1, ..., K]$) is used [8]. This constraint enforces that point number 1 is in cluster number 1, and point number i can only have cluster number k if there is a point $j < i$ with the same cluster number, or if $k - 1$ is the highest used cluster number so far. For the objective, we introduce a floating point variable V to represent the sum-of-squares of the clustering defined by the variables G. The domain of V is initially $[0, U_n)$. The bounds of V are updated by a novel global constraint $V = sumSquares(G, d, W)$, where d is the (precomputed) distance between each pair of points, and W contains the previous WCSS* values (as per Algorithm 2).

Additional constraints can be expressed over the G variables, including the user constraints defined in Section 2. Instance-level constraints are expressed by $G_i = G_j$ for a must-link constraint and $G_i \neq G_j$ for a cannot-link constraint on i, j. A maximal cluster size constraint, following its formal definition, is expressed by K CP cardinality constraints: $\#\{i \in [1, N] \mid G_i = k\} \leq \beta$ for each $k \in [1, K]$. Each of these constraints enforces that the number of variables G_i that are assigned to k must not exceed β. Other constraints can be modelled following their formal definition as well, see [8] for more examples.

According to the principle of RBBA, the variable order used during search instantiates (branches over) the variables $G_1, \ldots, G_n$ in increasing order of their index.

5.2 A Novel Sum-of-Squares Constraint

The filtering algorithm for constraint $V = sumSquares(G, d, W)$ is detailed in Algorithm 3. Because of the variable order, at any time the propagator is called, there is an index p ($1 \leq p < n$) such that $G_1, .., G_p$ are instantiated and $G_{p+1}, ..., G_n$ are not.

Algorithm 3 enforces bound consistency for V by first computing a lower bound for V. The values $sum[k]$ and $size[k]$ represent respectively the sum of squared distances between any two points in the cluster k and the number of points in that cluster. The value V_1 represents the sum of squares of the partial clustering formed by the first p assigned points, using Equation (2). Since W_{n-p} represents the minimal WCSS value for the last $n - p$ points (the unassigned points $G_{p+1}, ..., G_n$), according to Equation (6) and (8), $V_1 + W_{n-p}$ is a lower bound for V (line 15). Since $V.lb \leq V < V.ub$, a failure will occur if $V_1 + W_{n-p} \geq V.ub$ (line 12) leading the search to backtrack. Otherwise the lower bound $V.lb$ is revised.

Algorithm 3 exploits also W to do a look ahead to filter the domain of G_{p+1}. Each value $s[k]$ represents the contribution of point $p + 1$ in case it is assigned to cluster k. For each $k \in Dom(G_{p+1})$, that is, all clusters k not forbidden for this point because of another constraint, if point $p + 1$ is assigned to the cluster k, V_1' is the revised value of V_1. So V_1' represents the sum of squares of the partial clustering formed by the first $p+1$ points. Since W_{n-p-1} represents the minimal WCSS value for the last $n - p - 1$ points, according to Equation (6), if $V_1' + W_{n-p-1} \geq V.ub$ then a failure would occur. This means point $p + 1$ cannot be assigned to cluster k. The value k is then removed from $Dom(G_{p+1})$.

Algorithm 3: Filtering of: "$V = sumSquares(G, d, W)$"

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

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

5.3 Other Improvements

5.3.1 Must-link Constraints

Must-link constraints agglomerate related points to the same cluster. Therefore to make better use of this kind of constraint, first of all the transitive closure of all the must-link constraints is computed. This defines a set of super-points or ML-blocks [10]. Instead of clustering the set of initial points, we search for a clustering on the set of ML-blocks. Given a set of N initial points, assume that there are M ML-blocks to be considered ($M \leq N$). The distance between two ML-blocks b_i, b_j is defined as $d(b_i, b_j) = \sqrt{\sum_{o \in b_i, o' \in b_j} d(o, o')^2}$. Each block b_i has also its weight $w(i) = \sum_{o,o' \in b_i} d(o, o')^2/2$ and its size $s(i)$ which is the number of initial points in it. A block b_i that contains only one point has $w(i) = 0$ and $s(i) = 1$. Instance-level constraints that remain to be satisfied are only cannot-link constraints. A cannot-link constraint is defined on two blocks b_i, b_j if there exists a cannot-link constraint on two points o, o' such that $o \in b_i$ and $o' \in b_j$.

Using blocks means that in the model of Subsection 5.1, each variable G_i corresponds to a block b_i. All user constraints can be redefined on blocks. For instance, a minimal cardinality constraint states that each cluster should have at least α initial points. To express this constraint, we define an array T, where each variable G_i is repeated $s(i)$ times. The size of T is therefore N and the minimal cardinality constraint has to be expressed by $|\{j \in \{1, \dots, N\} \mid T_j = k\}| \geq \alpha$ for $k \in [1, K]$. Algorithm 3 can also be adapted to take into account size and weight of blocks.

5.3.2 Finding a Feasible Extension

Without user constraints, Greedy_Extension($\mathcal{O}_n, \Delta^*_{n-1}$) is found by adding the new point to the previous best clustering Δ^*_{n-1}. This typically yields a good upper bound, often even being the optimal value. For *Feasible_Extension($\mathcal{O}_n, \mathcal{C}_n, \Delta^*_{n-1}$)* in Algorithm 2, one has to additionally take the user constraints into account, since the clustering Δ_n must satisfy all $\mathcal{C}_n$ constraints.

We aim at finding a good feasible clustering that satisfies all the user constraints quickly. To achieve this, the same model as described in Subsection 5.1 is used with one restriction, namely that the last $n-1$ variables $G_2, \dots, G_n$ are assigned to the value they had in clustering Δ^*_{n-1}; this mimics a greedy strategy as only one variable can be decided, corresponding to adding the point to an existing cluster. If no such extension of the clustering exists, the clustering Δ_n is undefined and its WCSS value is ∞.

5.3.3 Local vs. Full Constraint Sets

Let $\mathcal{C}$ be the set of all user constraints on the whole set of points $\{o_1, \dots, o_N\}$. There may be instance-level constraints (must-link or cannot-link constraints) or cluster-level constraints (cardinality, density constraints etc.). At each step n, *Constrained_BaB* finds a clustering that minimizes the WCSS value and that satisfies the set of constraints $\mathcal{C}_n$. We propose two different ways to define the set $\mathcal{C}_n$ in the constraint solver, following the discussion in Section 4.3.

Local model Let $\mathcal{O}_n$ be the set of points to cluster at step n. The simplest way is to define $\mathcal{C}_n$ by $C_{\mathcal{O}_n}$, the set of user constraints on a (sub)set of the elements of $\mathcal{O}_n$. One can see that for $n = N$, $\mathcal{O}_N = \mathcal{O}$ and hence we will consider the set $C_{\mathcal{O}} = C$ of all constraints.

Full model To obtain tighter bounds, we can take anti-monotonic constraints into account too. However, we can also use CP capabilities to reason over partial solutions, to let it consider *all* constraints at every step. In this case, at each iteration $n \leq N$, *Constrained_BaB* operates on the full set of N variables and all the user constraints in $\mathcal{C}$ are considered in the model. However, since we are interested in finding a best clustering on the last n points of G only, the constraint *sumSquares* is defined only on the last n variables $G_{N-n+1}, \dots, G_N$. The branching is also on these n variables only.

The interest of such a *full* model is that it can allow to prune earlier cases that cannot be extended to a full solution. Let us take an example with 3 points a, b, c ($N = 3$), $K = 2$ and two cannot-link constraints $CL(a, b)$ and $CL(a, c)$. In step $n = 2$, the two last points are considered, $\mathcal{O}_2 = \{b, c\}$. The local model that is defined on G_b, G_c has no constraint ($\mathcal{C}_2 = \emptyset$) and will return a clustering Δ_2 where each point is in one cluster. The clustering Δ_2 cannot be used anymore at the next step, where the constraints cannot-link are taken into account. Meanwhile, the full model at each step has the 3 variables G_a, G_b, G_c and two constraints $G_a \neq G_b$ and $G_a \neq G_c$. At step $n = 2$, even though only two variables G_b, G_c are instantiated, the existence of G_a in the model prevents b and c to be in two different clusters, since otherwise $Dom(G_a) = \emptyset$. The full model can therefore yield better, higher but more realistic, lower bounds for the WCSS attainable in later iterations.

6 EXPERIMENTS

We compare CPRBBA to other state-of-the-art exact clustering approaches: original RBBA[3] [6], CPClustering 2.1[4] [9] using CP with one phase branch-and-bound search and CCCG-0.5.1[5] [4] using Integer Linear Programming and column generation. Both unconstrained and constrained settings are considered. We also show the interest of our generic approach by its use in a multi-objective constrained clustering setting, which minimizes the WCSS and maximizes the separation between clusters.

CPRBBA is developed using the Gecode[6] framework, version 4.3.3. Due to the computational demand of exact clustering we use small but classic datasets from the UCI repository[7] with the true number of class labels, except for the Hatco dataset [6] which has an unknown number of classes, see Table 1. All experiments are performed on Intel Xeon E3-1225 CPUs running Ubuntu 14.04; a time limit of 30 minutes is used and a memory limit of 4 gigabytes (which is never reached). Codes and examples are available on `http://www.cp4clustering.com`.

6.1 Unconstrained Clustering

As noted before, the performance of (CP)RBBA can change depending on the ordering of the variables used. We compare in Table 1 CPRBBA (local model) with 4 different orderings: order in which the points are read from the input file (input), average of 5 random orderings (random), nearest-neighbor separation as used in RBBA (NNS), and the furthest-point first ordering (FPF). We see that the best ordering can differ from dataset to dataset. In the following, we use the FPF strategy as it has the smallest average runtime.

We now compare CPRBBA to RBBA [6], to CPClustering using CP [9] and CCCG using column generation [4]. Other unconstrained exact methods have no publicly available implementation, but the respective experiments point to RBBA as being the fastest for small values of k, as is typical in data mining.

The results are shown in Table 2. We can see that both RBBA and CPRBBA are better than the recent CPClustering and CCCG methods in case no constraints are added, and that the difference in runtime between RBBA and CPRBBA is in accordance to the difference in ordering used as reported in Table 1.

6.2 Clustering with User-Constraints

We compare CPRBBA with CPClustering and CCCG, supporting also user constraints.

[3] `http://www.psiheart.net/QuantPsych/monograph.html`
[4] `http://www.cp4clustering.com/`
[5] `https://dtai.cs.kuleuven.be/CP4IM/cccg/`
[6] `http://www.gecode.org`
[7] `http://archive.ics.uci.edu/ml/`

dataset	N	K	input	random	NNS	FPF
ruspini	75	4	0.06	**0.00**	0.01	0.01
soybean	47	4	773.91	10.01	**0.80**	1.28
hatco	100	2	0.19	**0.02**	0.07	0.05
hatco	100	3	4.68	0.69	0.55	**0.20**
hatco	100	4	980.35	556.33	78.37	**7.52**
hatco	100	5	1800+	1800+	1800+	**1636.41**
iris	150	3	1800+	**0.95**	2.30	1.33
wine	178	3	1800+	1800+	**16.37**	53.57
seeds	210	3	1800+	491.03	1353.26	**170.67**
breast	569	2	**1167.62**	1800+	1800+	1800+
average			*1012.7*	*645.9*	*505.2*	*367.1*

Table 1. Runtimes in seconds of CPRBBA for different point orderings.

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

Table 2. Runtimes in seconds of different exact methods

Instance-level constraints We randomly sampled a number of must-link (ML) and cannot-link (CL) constraints from the true class labels of the datasets. Two points are randomly taken and depending on whether they have the same label or not, a ML or a CL constraint is created. This is repeated until the required ML/CL number is reached.
ML constraints only. We observe in Table 3 that CPRBBA outperforms the other two exact constrained clustering methods, CCCG and CPClustering. For must-link constraints, there is no difference between using -full or -local models because of the use of must-link blocks. In only one case (a 50-constraint set for the wine dataset), CPRBBA is not able to find a solution within the timeout.

	#c	CCCG	CPClustering	CPRBBA-local	CPRBBA-full
iris	10	1800+ (5)	341.59 (0)	**0.81 (0)**	0.86 (0)
iris	50	1800+ (5)	135.32 (0)	**0.23 (0)**	0.25 (0)
iris	100	47.20 (0)	1.20 (0)	**0.01 (0)**	**0.01 (0)**
iris	150	0.20 (0)	0.07 (0)	**0.01 (0)**	**0.01 (0)**
wine	10	1800+ (5)	1800+ (5)	**258.54 (0)**	259.30 (0)
wine	50	1800+ (5)	1800+ (5)	**363.34 (1)**	363.62 (1)
wine	100	1800+ (5)	1800+ (5)	**1.19 (0)**	1.23 (0)
wine	150	10.60 (0)	18.92 (0)	**0.13 (0)**	**0.13 (0)**

Table 3. Runtimes averaged over 5 random samples of #c must-link constraints; between brackets number of runs that timed-out (counted as 1800 seconds in average).

CL constraints only. The results for cannot-link constraints are shown in Table 4. Adding CL constraints can make the problem much harder. Here too CPRBBA outperforms the others, which is in line with the time difference in the unconstrained case. As more constraints are added, an optimal solution can be found in the given timeout for fewer sampled constraint sets (see number between brackets), leading to higher average runtimes.

	#c	CCCG	CPClustering	CPRBBA-local	CPRBBA-full
iris	10	1800+ (5)	727.32 (0)	**1.69 (0)**	1.79 (0)
iris	50	1800+ (5)	1694.03 (4)	**63.94 (0)**	64.07 (0)
iris	100	1800+ (5)	497.90 (0)	368.41 (1)	**15.40 (0)**
iris	150	1800+ (5)	643.72 (1)	721.29 (2)	**361.57 (1)**
iris	250	1800+ (5)	1094.49 (3)	1080.66 (3)	**0.74 (0)**
wine	10	1800+ (5)	1800+ (5)	**622.89 (1)**	625.64 (1)
wine	25	1800+ (5)	1800+ (5)	**1310.99 (2)**	1326.51 (2)
wine	50	1800+ (5)	1800+ (5)	**1697.94 (4)**	1706.36 (4)
wine	100	1800+ (5)	1800+ (5)	1800+ (5)	1800+ (5)

Table 4. Runtimes averaged over 5 random samples of #c cannot-link constraints; between brackets number of runs that timed-out (counted as 1800 seconds in average).

These results extend to the combination of must-link and cannot-link constraints (not shown).

Cluster-level constraints Table 5 shows runtimes for different datasets when adding a minimal or a maximal cluster size constraint. We can see that CPRBBA can handle such constraints well, and better than CPClustering. CPRBBA-full considers more constraints than CPRBBA-local in between iterations, and can hence provide tighter bounds. However, we observe that for some datasets, obtaining tighter bounds requires more search in one iteration to get them, thus loosing the benefits of the tighter bounds in subsequent iterations, and thus leading to overhead. For the iris dataset, the effort of searching for a tighter bound does pay of in the experiments. We observe similar results for a maximum cluster size constraint.

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

	K	max. size	cpclus.	cprbba-local	cprbba-full
ruspini	4	20	0.54	**0.01**	0.05
ruspini	4	19	1800+	**602.82**	794.83
soybean	4	14	**1.28**	1.32	1.83
soybean	4	13	17.52	**13.19**	17.44
iris	3	62	589.92	**1.31**	1.67
iris	3	58	723.63	3.95	**3.04**
iris	3	54	973.09	96.78	**18.31**
iris	3	50	1483.88	1800+	**158.75**

Table 5. Runtime in seconds for clustering with minimum (top) and maximum (bottom) size constraint

6.3 Multi-Objective Constrained Clustering

Constraints offer a way to find solutions that better fit the problem at hand. Changing the objective function is another way. Curiously, whereas the aim of clustering is to find homogeneous as well as well-separated clusters, most measures, including WCSS, express only homogeneity. One solution is to use multi-objective optimization, with one measure for homogeneity and one for well-separatedness. The result is a set of Pareto optimal solutions, where a Pareto optimal solution is one for which it is not possible to improve the value of one criterion without degrading the value of the other one.

We propose an algorithm (Algorithm 4) to compute an exact set of Pareto solutions for bi-objective WCSS/Split optimization, so as to obtain both homogeneous and well-separated clusterings. It is based on the ϵ-*constraint* algorithm [22] and is applicable to any complete method that can optimize WCSS under must-link constraints. In this algorithm, constrained single objective optimization (WCSS) is iterated, each time with a condition on the best value of the other objective (minimal split) found so far. This minimal-split constraint can in turn be translated into must-link constraints.

Algorithm 4: Bi-objective WCSS/Split

```
1 $Pareto\_sols \leftarrow \emptyset$
2 $min\_split \leftarrow 0$
3 repeat
4     $\Delta \leftarrow$ Minimize_WCSS$(\mathcal{O}, \{Split > min\_split\})$
5     $min\_split \leftarrow$ Split$(\Delta)$
6     if $\Delta$ is not dominated in $Pareto\_sols$ then
7         $Pareto\_sols \leftarrow Pareto\_sols \cup \{\Delta\}$
8 until no $\Delta$ was found;
```

In [12, 28, 27] the problem of finding the Pareto optimal solutions for minimizing the maximal diameter of the clusters and maximizing the minimal split between clusters is addressed, but without user-constraints. To our best knowledge the only work that handles user-constraints inside a multi-objective clustering problem is [8]. That work does not consider the WCSS criterion, and the criteria used often lead to thousands of equivalent clusterings corresponding to each Pareto point. Algorithm 4 can be easily modified to incorporate user-constraints, in case the Minimize_WCSS algorithm supports it: another set of user-constraints can simply be added to the split constraint at line 4.

Experiments Table 6 presents runtimes in seconds, number of Pareto solutions and the maximal number of clusterings Δ' corresponding to each Pareto solution Δ (i.e. $WCSS(\Delta') = WCSS(\Delta)$ and $Split(\Delta') = Split(\Delta)$). We can see here (last column) that for each Pareto solution, there is always only one corresponding clustering, which contrasts with the thousands of equivalent solutions found in [8] for the Diameter/Split measure.

	K	time (s)	#sols	#c/s
ruspini	4	0.01	1	1
soybean	4	1.58	4	1
hatco	4	32.52	24	1
hatco	5	1979.38	22	1
iris	3	1.11	10	1
wine	3	100.58	9	1
seeds	3	178.62	17	1

Table 6. Runtime, # Pareto solutions, maximal number of clusterings for each Pareto solution

Our framework can also be used for bi-objective WCSS/Split under user constraints. To the best of our knowledge, it is the first method to support this bi-criterion optimization both for instance- and cluster-level constraints. Table 7 shows the results for different use cases on the Iris dataset. For four of these cases, the exact Pareto fronts are shown in Figure 1 (the two cases for 20 ML/CL constraints with and without the minimal size constraint have the same Pareto front). We can see here the interest of being able to handle user-constraints during the optimization process. Indeed, in this dataset, each ground truth cluster is of size 50, whereas in the unconstrained use case, the Pareto solutions can give clusterings with unbalanced clusters. For instance, the last point in the Pareto front corresponds to a clustering with clusters of size 2, 50 and 98. The constrained cases have the last Pareto solution with WCSS=86.5396 and Split=0.412311. This solution is common to all the 4 cases, and the only corresponding clustering has clusters of size 49, 50, 51.

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

Table 7. Results on Iris for bi-criterion constrained clustering cases

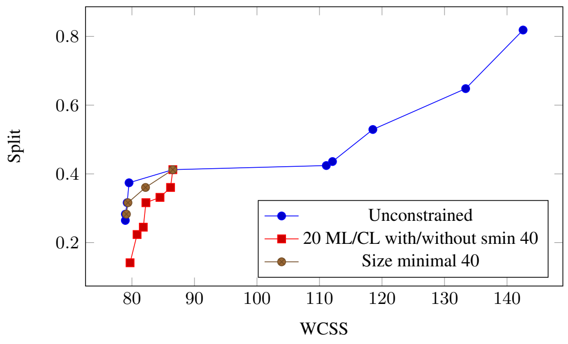

Figure 1. Pareto fronts for different cases on Iris

7 CONCLUSION

In this paper, we address one of the most popular constrained clustering task, the constrained minimum sum-of-squares clustering (MSSC). We extend the Repetitive Branch-and-Bound Algorithm, one of the best method for MSSC without user constraints, to integrate user constraints. The framework we propose is based on Constraint Programming (CP), which is used in each internal branch-and-bound step, as well as in the computation of upper and lower bounds. We propose two different CP models in order to have tight lower bounds and construct a specific propagation mechanism to make better use of the computed bounds. Experiments on classic datasets show that our approach, even though being generic, is competitive compared to a dedicated implementation of RBBA in the unconstrained case. For constrained cases, our approach outperforms the existing state-of-the-art exact approaches. Furthermore, we show how its generality allows it to be used in a bi-objective constrained clustering setting.

To further enhance the efficiency of the framework, one may have to consider other ordering heuristics, including dynamic ones. Moreover, RBBA has been applied to clustering tasks with other optimization criteria such as WCSD, to which our approach can be extended as well. Our bi-objective approach can also be used with non-exact constrained clustering methods, though the resulting Pareto front will be an approximation. Lastly, a mix of Russian Doll Search and our approach may lead to advances for both valued CSPs and clustering.

REFERENCES

[1] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat, 'NP-hardness of Euclidean Sum-of-squares Clustering', *Machine Learning*, **75**(2), 245–248, (2009).

[2] Daniel Aloise and Pierre Hansen, 'An branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering', *Pesquisa Operacional*, **29**(3), 503–516, (2009).

[3] Daniel Aloise, Pierre Hansen, and Leo Liberti, 'An improved column generation algorithm for minimum sum-of-squares clustering', *Mathematical Programming*, **131**(1-2), 195–220, (2012).

[4] Behrouz Babaki, Tias Guns, and Siegfried Nijssen, 'Constrained clustering using column generation', in *Proceedings of the 11th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 438–454, (2014).

[5] Michael J. Brusco, 'An enhanced branch-and-bound algorithm for a partitioning problem', *British Journal of Mathematical and Statistical Psychology*, **56**(1), 83–92, (2003).

[6] Michael J. Brusco, 'A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning', *Psychometrika*, **71**(2), 347–363, (2006).

[7] Thi-Bich-Hanh Dao, Kanh-Chuong Duong, and Christel Vrain, 'A Declarative Framework for Constrained Clustering', in *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pp. 419–434, (2013).

[8] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain, 'Constrained clustering by constraint programming', *Artificial Intelligence*, DOI: 10.1016/j.artint.2015.05.006, (2015).

[9] Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain, 'Constrained minimum sum of squares clustering by constraint programming', in *Principles and Practice of Constraint Programming, CP 2015, Proceedings*, pp. 557–573, (2015).

[10] Ian Davidson and S. S. Ravi, 'Clustering with Constraints: Feasibility Issues and the k-Means Algorithm', in *Proceedings of the 5th SIAM International Conference on Data Mining*, pp. 138–149, (2005).

[11] Ian Davidson, S. S. Ravi, and Leonid Shamis, 'A SAT-based Framework for Efficient Constrained Clustering', in *Proceedings of the 10th SIAM International Conference on Data Mining*, pp. 94–105, (2010).

[12] M. Delattre and P. Hansen, 'Bicriterion cluster analysis', *IEEE Trans. Pattern Anal. Mach. Intell.*, (4), 277–291, (1980).

[13] O. du Merle, P. Hansen, B. Jaumard, and N. Mladenovic, 'An interior point algorithm for minimum sum-of-squares clustering', *SIAM Journal on Scientific Computing*, **21**(4), 1485–1505, (1999).

[14] A. W. F. Edwards and L. L. Cavalli-Sforza, 'A method for cluster analysis', *Biometrics*, **21**(2), 362–375, (1965).

[15] T. Gonzalez, 'Clustering to minimize the maximum intercluster distance', *Theoretical Computer Science*, **38**, 293–306, (1985).

[16] Pierre Hansen and Brigitte Jaumard, 'Cluster analysis and mathematical programming', *Mathematical Programming*, **79**(1-3), 191–215, (1997).

[17] Robert E. Jensen, 'A dynamic programming algorithm for cluster analysis', *Journal of the Operations Research Society of America*, **7**, 1034–1057, (1969).

[18] W. L. G. Koontz, P. M. Narendra, and K. Fukunaga, 'A branch and bound clustering algorithm', *IEEE Trans. Comput.*, **24**(9), 908–915, (1975).

[19] Marianne Mueller and Stefan Kramer, 'Integer Linear Programming Models for Constrained Clustering', in *Proceedings of the 13th International Conference on Discovery Science*, pp. 159–173, (2010).

[20] Dan Pelleg and Dorit Baras, 'K-means with large and noisy constraint sets', in *Machine Learning: ECML 2007*, volume 4701 of *Lecture Notes in Computer Science*, pp. 674–682. Springer Berlin Heidelberg, (2007).

[21] Douglas Steinley, 'k-means clustering: A half-century synthesis', *British Journal of Mathematical and Statistical Psychology*, **59**(1), 1–34, (2006).

[22] Vincent T'kindt and Jean-Charles Billaut, *Multicriteria Scheduling, Theory, Models and Algorithms*, Springer, 2nd edn., 2005.

[23] B.J. van Os and J.J. Meulman, 'Improving Dynamic Programming Strategies for Partitioning', *Journal of Classification*, (2004).

[24] Gérard Verfaillie, Michel Lemaître, and Thomas Schiex, 'Russian doll search for solving constraint optimization problems', in *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96*, pp. 181–187, (1996).

[25] K. Wagstaff and C. Cardie, 'Clustering with instance-level constraints', in *Proceedings of the 17th International Conference on Machine Learning*, pp. 1103–1110, (2000).

[26] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl, 'Constrained K-means Clustering with Background Knowledge', in *Proceedings of the 18th International Conference on Machine Learning*, pp. 577–584, (2001).

[27] J. Wang and J. Chen, 'Clustering to maximize the ratio of split to diameter', in *Proceedings of the 29th International Conference on Machine Learning*, (2012).

[28] Y. Wang, H. Yan, and C. Sriskandarajah, 'The weighted sum of split and diameter clustering', *Journal of Classification*, **2**(12), 231–248, (1996).

[29] Y. Xia and J. Peng, 'A cutting algorithm for the minimum sum-of-squared error clustering', in *SDM*, (2005).