An Exact CP Approach for the Cardinality-Constrained Euclidean Minimum Sum-of-Squares Clustering Problem

Mohammed Najib Haouas, Daniel Aloise, and Gilles Pesant

Polytechnique Montréal, Canada
{mohammed-najib.haouas,daniel.aloise,gilles.pesant}@polymtl.ca

Abstract. Clustering consists in finding hidden groups from unlabeled data which are as homogeneous and well-separated as possible. Some contexts impose constraints on the clustering solutions such as restrictions on the size of each cluster, known as cardinality-constrained clustering. In this work we present an exact approach to solve the Cardinality-Constrained Euclidean Minimum Sum-of-Squares Clustering Problem. We take advantage of the structure of the problem to improve several aspects of previous constraint programming approaches: lower bounds, domain filtering, and branching. Computational experiments on benchmark instances taken from the literature confirm that our approach improves our solving capability over previously-proposed exact methods for this problem.

1 Introduction

Data analysis has become an important field of study in an age dominated by substantial and indiscriminate data collection. One of the most direct ways to extract information from a set of data observations takes the form of a clustering procedure wherein data is grouped in homogeneous and/or well separated bundles based on some measure of similarity/dissimilarity. The partitioned data offers a more tractable presentation of the unlabeled observations. Depending on the criterion on which the partitioning is based, different clusters may be achieved.

Definition 1. *Let $O = \{o_1, o_2, ..., o_n\}$ be a set of n data observations in some space and $d : O^2 \mapsto \mathbb{R}^+$ a dissimilarity measure (not necessarily a distance). A* k-partition *($k < n$) $\Delta \in \mathcal{A}$ of O into a set of classes $\mathcal{C} = \{C_c\}_{1 \leq c \leq k}$ (with $\mathcal{A}$ the set of all possible k-partitions) is such that :*

$$C_c \neq \varnothing \ \ \forall\, 1 \leq c \leq k, \quad \cup_{1 \leq c \leq k} C_c = O, \quad C_c \cap C_{c'} = \varnothing \ \ \forall\, 1 \leq c < c' \leq k$$

Let $\gamma_d : \mathcal{A} \mapsto \mathbb{R}^+$ be a partitioning criterion based on d. Δ^* is an optimal partition *if $\Delta^* = \operatorname{argmin}_{\Delta \in \mathcal{A}} \gamma_d(\Delta)$.*

One popular partitioning criterion is the Euclidean Minimum Sum-of-Squares Clustering (MSSC) which is widely used to produce high quality, homogeneous, and well-separated clusters [2]. At its core, it minimizes intra-cluster variance.

Definition 2. *Consider observations in $\mathbb{R}^s$. MSSC aims to find the cluster centers $c_j \in \mathbb{R}^s$, as well as cluster assignments w_{ij} that solve the following program [2]*

$$\begin{aligned} \textit{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{k} w_{ij}\|o_i - c_j\|^2 & \\ \textit{s.t.} \quad & \sum_{j=1}^{k} w_{ij} = 1 & \forall 1 \leq i \leq n \\ & w_{ij} \in \{0,1\} & \forall 1 \leq i \leq n,\ \forall 1 \leq j \leq k, \end{aligned}$$

where $w_{ij} = 1$ represents the assignment of observation o_i to cluster C_j.

MSSC is NP-hard in general dimension [1].

Often, prior information is known about the data and can be introduced to the clustering process in order to increase performance as well as solution quality. This is possible through expression of custom constraints on the observations or the resulting clusters [26, 6]. In this paper we propose an exact approach to solve a specific variant of constrained MSSC: one that involves cardinality constraints on the resulting clusters (ccMSSC). Strict cardinality constraints in clustering can be encountered in various fields such as image segmentation [17], distributed clustering [4], category management in business [5], document clustering [5], and workgroup composition [15]. Cardinality constraints can also be used to reinforce the clustering procedure against the presence of outliers as well as groups that are either too large or too small [23, 25]. The ccMSSC is already NP-hard in one dimension for $k \geq 2$ [9]. In principle, existing Constraint Programming (CP) approaches for MSSC [14, 13, 16] may be extended in order to handle such a variant by adding a global cardinality constraint. Our contribution shows that we can achieve better performance with specialized global constraints with targeted filtering algorithms as well as an adapted search heuristic, both designed to take advantage of the special structure of the problem in order to quickly reduce the search space. Furthermore, in using CP we ensure easy extension of this work to include independent user-defined constraints.

In the rest of the paper, Section 2 defines the CP model used to solve MSSC, to which constraints can be added for the special case of ccMSSC, among others. Section 3 is devoted to a review of the literature surrounding MSSC as well as constrained MSSC. Sections 4 and 5 present our contributions: two filtering algorithms dedicated to ccMSSC resolution as well as an updated and more robust version of an existing search heuristic for MSSC. Section 6 summarizes experimental results as well as comparisons to existing methods. Finally, Section 7 provides a brief summary of our work and discusses future research avenues.

2 Basic CP Model

The problem stated in Definition 2 was modeled in CP by Dao et al [14]:

Variables. Each observation o_i is represented as an integer variable $x_i, D(x_i) = \{1, \ldots, k\}$ representing the index of the class to which the corresponding observation belongs. A variable $n_c, D(n_c) = \{0, 1, ..., n\}$ is introduced for each cluster to represent its cardinality.

Objective. Recall MSSC involves finding an optimal set of cluster centers that minimize the intra-cluster variance, per Definition 2. However there is an equivalent formulation [13] of the objective which circumvents these centers, enabling us to solve the problem without making them explicit:

$$\text{minimize} \quad \sum_{c=1}^{k} \frac{1}{2} \frac{1}{|C_c|} \sum_{o,o' \in C_c} \|o - o'\|^2. \tag{1}$$

Using reified constraints, the objective can be further simplified and rewritten as follows:

$$\text{minimize} \quad \sum_{c=1}^{k} \frac{1}{n_c} \sum_{\mathtt{i}=1}^{n-1} \sum_{j=i+1}^{n} (x_i = c \wedge x_j = c) \cdot \|o_i - o_j\|^2 \tag{2}$$

The objective expression in Equation 2 can be constrained to be equal to a real variable $Z, D(Z) = [0, \infty[$ known as the Within Cluster Sum of Squares (WCSS), from which the new objective is:

$$\text{minimize} \quad Z \tag{3}$$

Constraints. A Global Cardinality Constraint (GCC) [21] constrains variables n_c to take on the cardinality of their corresponding cluster:

$$\texttt{GCC}\left(\{n_c\}_{1 \leq c \leq k}, \{1, 2, \ldots, k\}, \{x_i\}_{1 \leq i \leq n}\right) \tag{4}$$

This model contains a value symmetry which can hinder performance (cluster indices are interchangeable). One way to overcome this is to maintain pairwise integer value precedence on the branching variables as follows [27]:

$$\texttt{intValPrecedence}\left(\{x_i\}_{1 \leq i \leq n}, c-1, c\right) \qquad \forall\, 1 < c \leq k \tag{5}$$

In essence, each instance of the above constraint ensures that if $x_i = c$ then $\exists\, j < i$ such that $x_j = c - 1$. A higher level of propagation can theoretically be achieved by considering each possible pair (as opposed to only adjacent pairs) of values. However, this comes at a price for virtually no benefit to domain reductions in practice [20].

3 Related Work

MSSC is a very well-studied problem and one that is often tackled through heuristics due to its extremely hard nature. K-means is perhaps the most important and widely-used algorithm to solve the unconstrained MSSC problem [28]. It

performs a local search to find a partition with minimal within-cluster variance, iteratively relocating cluster centers and stopping at a local optimum. Among exact methods, *CP Clustering* (CPC) presented in [13] is a first successful attempt at using CP for MSSC. Improving on the model presented in Section 2, the authors suggest a simple search heuristic as well as a global constraint to efficiently navigate the search space looking for a globally optimal solution to the problem. The authors leverage calculation of lower bounds to filter the objective variable as well as perform cost-based filtering on the branching variables. *CP Repetitive Branch and Bound* (CP RBBA) presented in [16] is a second attempt at leveraging CP to solve MSSC. Its operation is inspired from Repetitive Branch and Bound (RBBA) in [10] where MSSC is divided into sub-problems, each treated as an independent CP model in CP RBBA. This enables the use of a range of user constraints (which RBBA doesn't support) as well as the computation of tighter bounds, leading to substantially better performance for many instances.

Turning now to constrained variants of MSSC, the K-means heuristic approach has been extended to support various constraints [26, 8]. A special case of ccMSSC, the balanced MSSC, is approached in [12] using a simple Variable Neighborhood Search. Through constant-time reevaluations of the objective after each reassignment as well as carefully selected local search neighborhoods, the authors are able to find the best known values of several large instances. More relevant to us, the authors of [23] suggest a method for solving the ccMSSC using convex relaxations of the problem, whose solutions can be "rounded" to a valid one for the main problem. Their approach distinguishes itself from the others by providing *a posteriori* guarantees on the sub-optimality of the solutions obtained. In fact, based on these guarantees, the authors are able to declare several of the solutions they found as being globally optimal. An exact Column Generation framework for solving constrained MSSC was proposed in [3], supporting anti-monotone constraints which can be used to restrict the maximum cardinality of the clusters. Of course the CP methods previously described for MSSC, CPC and CP RBBA, can solve the ccMSSC by simply adding a GCC but a contribution of our work is to show that, for such a constraint, a more integrated approach is much more productive.

4 Filtering Based on Cardinality-Constrained Clustering

In this section we present two filtering algorithms for a global constraint [13] aimed at accelerating resolution of the model in Section 2 for the case of ccMSSC.

4.1 Basic filtering derived from CPC

This first filtering algorithm represents a specialization of CPC for the ccMSSC. We both accelerate and tighten its bound computation by exploiting the fact that cluster sizes $\{n_c\}_{1 \leq c \leq k}$ are fixed.

The global constraint in CPC evaluates, at each search tree node, the minimum contribution $\underline{Z}(C_c, m)$ to the objective Z for each cluster C_c whenever any

m free observations are assigned to it:

$$\underline{Z}(C_c, m) = \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m} R_i(c)}{|C_c| + m} \tag{6}$$

where $Z(C_c)$ represents the WCSS of the partially filled C_c and $(R_i(c))_{i=1}^{q}$ is a non-decreasing sequence where each term represents the lowest individual contribution of the i-th free observation to C_c (among q which are unassigned at the current node) such that:

$$R_i(c) = r_2(i, c) + \sum_{j=1}^{m} r_{3,j}(i) \tag{7}$$

where $r_2(i, c)$ is the contribution of the i-th free observation due to the observations already in C_c and $(r_{3,j}(i))_{j=1}^{q}$ is a non-decreasing sequence where each term represents half the distance between that same i-th free observation and a nearby element in U, the set of free observations (itself included, i.e. $r_{3,1}(i) = 0\ \forall i$). Refer to Fig. 1 for an illustration.

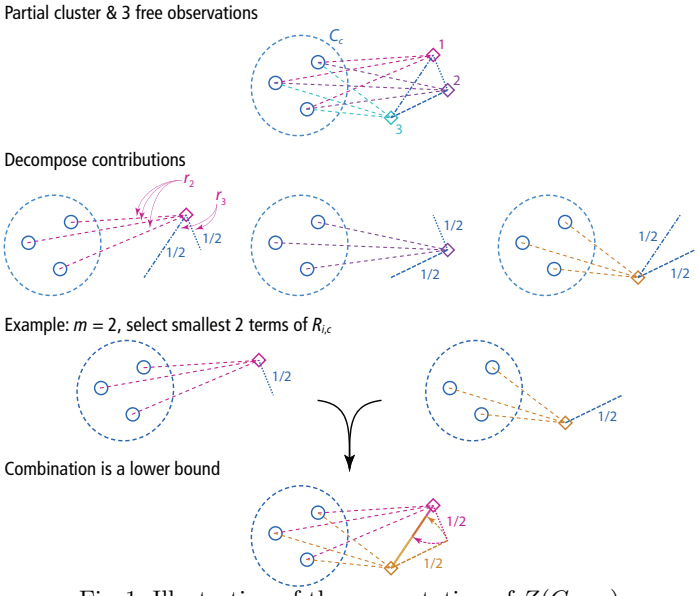

Fig. 1: Illustration of the computation of $\underline{Z}(C_c, m)$

The authors of CPC make use of dynamic programming in conjunction with Equation 6 to compute lower bounds for the general MSSC problem as well as to perform the necessary filtering on variables [13].

Global lower bound for ccMSSC. We observe that at each node of the ccMSSC resolution, one knows exactly how many observations are to be assigned to each

cluster C_c to complete it to its target cardinality n_c. As such, given a partial assignment, a lower bound on the cost of a full solution can be more simply computed as follows without resorting to dynamic programming to compute terms for different values of m:

$$\underline{Z}(\mathcal{C}) = \sum_{c=1}^{k} \underline{Z}(C_c, n_c - |C_c|) = \sum_{c=1}^{k} \underline{Z}(C_c, m_c) = \sum_{c=1}^{k} \underline{Z}_0(C_c) \tag{8}$$

where we denote as $\underline{Z}_0(C_c)$ the minimum individual contribution of C_c when it is completed to its target cardinality n_c, using m_c observations ($m_c := n_c - |C_c|$). Equation 8 filters the objective Z by tightening its lower bound. It also prunes branches that cannot result in a solution better than the incumbent.

Cost-based filtering on cluster assignment variables. It is possible to recycle computations in order to reevaluate a global lower bound to the problem for each value-variable assignment in order to enable effective cost-based filtering.

Consider assigning the ℓ-th free observation o' (w.r.t. the order of the sequence $(R_i(c))_{i=1}^{q}$) to cluster C_c. Let $\mathcal{C}' = \{C_1, \ldots, C'_c, \ldots, C_k\}$ denote the set of partially filled clusters identical to $\mathcal{C}$ except for C'_c which also contains o' ($C'_c = C_c \cup \{o'\}$). It is then possible to write the following:

$$\underline{Z}(\mathcal{C}') = \underline{Z}(\mathcal{C}) - \underline{Z}_0(C_c) + \underline{Z}_0(C'_c) \tag{9}$$

All the terms in Equation 9 are available except the last one. Therefore we devise a simple way to get a lower bound on it:

$$\begin{aligned}
\underline{Z}_0(C'_c) &= \frac{\underline{Z}(C_c, m_c - 1) \cdot (|C_c| + m_c - 1) + \ell\text{-th observation's contribution}}{|C_c| + m_c} \\
&\geq \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} R_i(c) + \sum_{i=1}^{m_c-1} r_{3,m_c}(i) + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c} \\
&= \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} \left[r_2(i,c) + \sum_{j=1}^{m_c} r_{3,j}(i) \right] + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c} \\
&= \frac{(|C_c| + m_c - 1) \cdot \underline{Z}_1(C_c) + r_{2,c}(\ell) + \sum_{j=1}^{m_c} r_{3,j}(\ell)}{|C_c| + m_c}
\end{aligned} \tag{10}$$

The ℓ-th observation's contribution represents the sum of the following quantities:

- the sum of dissimilarities between it and C_c's components: $r_{2,c}(\ell)$;
- half dissimilarities between it and $m_c - 1$ other free observations, which is greater than or equal to $\sum_{j=1}^{m_c} r_{3,j}(\ell)$;
- the other half dissimilarities between it and $m_c - 1$ other free observations, which is greater than or equal to $\sum_{i=1}^{m_c-1} r_{3,m_c}(i)$.

with

$$\underline{Z}_1(C_c) = \frac{Z(C_c) \cdot |C_c| + \sum_{i=1}^{m_c-1} R_i(c)}{|C_c| + m_c - 1} \qquad \text{where } R_i(c) = r_2(i,c) + \sum_{j=1}^{m_c} r_{3,j}(i) \tag{11}$$

which is similar to $\underline{Z}_0(C_c)$ with the difference being we only select $m_c - 1$ terms of $(R_i(c))_{i=1}^{q}$ in $\underline{Z}_1(C_c)$ instead of m_c. This enables the sequential computation of both values with the same complexity. Comparing Equation 9 against the upper bound of Z for each assignment considered enables filtering of values that cannot result in a solution better than the incumbent.

Summary. Propagation algorithms are called whenever the domain of some variable x_i changes or bounds on Z are tightened. Algorithm 4.1 summarizes the results of this section.

Algorithm 4.1 `propagate` method: basic filtering

(Computation of r_2 and r_3 not shown for brevity and are identical to [13])
for $c \leftarrow 1 \,..\, k$ **do**
  for $v \leftarrow 0 \,..\, 1$ **do**
    for $i \leftarrow 1 \,..\, n$ where $|D(x_i)| > 1$ **do** ▷ there are q unassigned observations
      if $m_c - v > 0$ **then**
        $R[i] \leftarrow r_2[c, i] + r_3[i, m_c - 1]$ ▷ r_3 represents the sum in Eq. 7 directly
    `sort` $(R[i \in 1..n : |D(x_i)| > 1])$ ▷ sort the contribs of the q free observations
    $\underline{Z}_v(C_c) \leftarrow \frac{Z(C_c)\cdot|C_c| + \sum_{i=1}^{m_c - v} R[i]}{m_c + |C_c| - v}$ ▷ $\underline{Z}_0(C_c)$ and $\underline{Z}_1(C_c)$
$\mathrm{LB}(Z) \leftarrow \sum_{c=1}^{k} \underline{Z}_0(C_c)$ ▷ filter objective, Eq. 8
for $c \leftarrow 1 \,..\, k$ **do** ▷ cost-based filtering
  $\mathrm{LB}_E \leftarrow \mathrm{LB}(Z) - \underline{Z}_0(C_c)$
  for $\ell \leftarrow 1 \,..\, n$ where $|D(x_\ell)| > 1$ **do**
    if $c \in D(x_\ell)$ **then**
      $\mathrm{LB}_P \leftarrow \frac{(|C_c| + m_c - 1)\cdot \underline{Z}_1(C_c) + r_2[c,\ell] + r_3[\ell, m_c - 1]}{|C_c| + m_c}$ ▷ Eq. 10
      if $(\mathrm{LB}_E + \mathrm{LB}_P \geq \mathrm{UB}(Z))$ **then**
        $D(x_\ell) \leftarrow D(x_\ell) \setminus \{c\}$ ▷ filter if incumbent cost exceeded

The modified CPC filtering algorithm specialized for ccMSSC has a time complexity in $\mathcal{O}(qn + q^2 \log q + kq \log q + k + kq) = \mathcal{O}(qn + q^2 \log q)$ (down from $\mathcal{O}(qn + kq^2 \log q)$ [13]) and a space complexity in $\mathcal{O}(n^2)$. The reduction in asymptotic complexity is less important than the tighter bounds produced which enable more aggressive domain reduction for the case of ccMSSC compared to the original version of the constraint. Computing individual r_2 and r_3 contributions incrementally has a detrimental effect in practice due to the overhead involved in pinpointing the changes that have occurred since the last node.

However, this filtering algorithm is limited by each cluster's individual minimum contribution being computed at a local level, regardless of that of other clusters. This means that it is possible for a given observation to be considered for the minimum contribution of two distinct clusters, hindering lower bound quality. We propose a way to correct this in the next section.

4.2 Improved filtering

A tighter global lower bound for ccMSSC. In computing the smallest cost of the solution extended from a partial assignment (i.e., the global lower bound at a certain node of the search tree), it helps to consider all clusters as a whole rather than each of them separately while distributing free observations between them. This eliminates the issue identified with the basic filtering discussed above and can be achieved by solving a minimum-cost flow (MCF) problem. At each node of the search tree, where $|U| = q$ observations are unassigned, a network can be built as follows:

1. start from a bipartite assignment graph where the first set of vertices represents the q free observations and the second set of vertices represents the k' ($k' \leq k$) incomplete clusters;
2. supply each of the vertices representing the observations with one unit of flow using a common source;
3. connect each of the vertices representing the partially filled clusters with arcs of capacity m_c to a common sink;
4. all other arcs have a capacity equal to 1;
5. only arcs connecting observations to clusters bear a cost, equal to $R_i(c)/n_c$. Such arcs only exist if the assignment is possible.

The MCF solution is integral because the constraints matrix for the corresponding linear program is Totally Unimodular and all other coefficients are integers. Arcs selected by the MCF represent the optimal division of the free observations between the incomplete clusters. The corresponding cost incurred by this completion, based on the minimum individual contributions of each free observation, necessarily leads to a lower bound on the cost of the solution derived from the current partial assignment:

$$\underline{Z}(\mathcal{C}) = \sum_{c=1}^{k} \frac{Z(C_c) \cdot |C_c|}{n_c} + \mathrm{MCF}^*_{\mathrm{cost}} \tag{12}$$

This lower bound is greater or equal to the one given by Equation 8.

On the surface, the method being discussed here resembles a GCC with costs [22]. However it is inapplicable here due to changing costs at each node of the search tree. Indeed, the cost incurred by the assignment of an individual observation is not known *a priori* as it changes every time a cluster is modified (which happens repeatedly in the search tree). Moreover, this continuously varying nature of the problem prevents us from taking advantage of most incremental computations involved in maintaining arc consistency in GCC with costs.

A new MCF instance must be solved each time an impactful change occurs in the search tree. We define such a change as one where an assignment variable has been fixed or one where a value has been filtered from the domain of an assignment variable such that it eliminates a flow-carrying arc in the current MCF solution. Otherwise the latter solution is still valid. If a new MCF solution must be computed, we use Network Simplex due to its speed and the fact that implementations of it are readily available.

A more thorough cost-based filtering. The same way adopting a global view of the problem facilitates generation of tighter bounds on Z, it is possible to leverage the flow formulation discussed above to perform a more powerful filtering of the decision variables. This is done through forcing flow on an arc using augmenting constraints in the current MCF problem to mandate a particular assignment. If a bound calculated using an augmenting constraint is higher than the cost of the incumbent solution, the value corresponding to the assumption made is filtered.

For the sake of efficiency, instead of recomputing a solution to the MCF problem for every possible augmenting constraint, we start by modifying the one that has been computed for the global lower bound. Such a modification will result in an infeasible solution (Fig. 2, left) because one cluster will be overfilled by one unit (red arc in violation) while another will be missing one unit (transparent bold arc). The task shifts to reestablishing a feasible and optimal solution from the situation depicted.

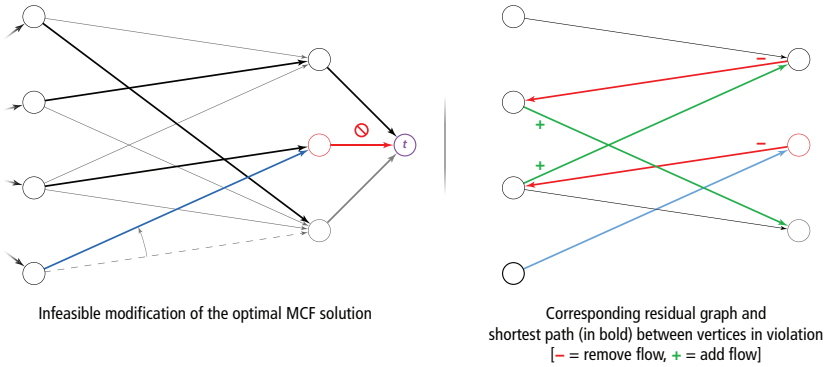

Fig. 2: MCF solution update for cost-based filtering

One way to fix this is to proceed, in an alternating fashion, to the removal and the addition of flow in arcs of the network designed to send the excess unit of flow from the vertex corresponding to the overfilled cluster to the one corresponding to the underfilled cluster. For the solution to be optimal, this alternating sequence of removals and additions should result in the lowest possible added cost.

A more straightforward way to look at this operation is through a residual graph derived from the optimal MCF solution (Fig. 2, right). A flow-carrying arc in the current solution is flipped in the residual graph and given the opposite cost. Restoring the optimal solution becomes a shortest path problem between the vertices in violation. This again is reminiscent of GCC with costs [22]. However our case is a more targeted one where the resulting residual graph is a simple bipartite digraph. We use the Bellman-Ford algorithm [7] (due to presence of negative-cost arcs) to solve the shortest path problem for each assumption. If a path cannot be found, then the augmented MCF problem is inconsistent and the

value corresponding to the assumption made must be filtered. The cost increase of the MCF solution after introduction of the augmenting constraint is equal to the variation due to relocating the free observation between clusters plus the weight of the shortest path.

Summary. Below is an algorithmic summary of the advanced filtering propagation. Changes with respect to the basic filtering algorithm are shown in green.

Algorithm 4.2 `propagate` method: advanced filtering

(Computation of R, r_2, and r_3 not shown for brevity; identical to [13] and Alg. 4.1)
1: **if** `impactfulChangeHasOccurred()` **then**
2: `makeMCFModel()`
3: `solution` $\leftarrow$ `solveMCFModel()` ▷ Network Simplex
4: LB(Z) $\leftarrow$ partial WCSS + `solution.cost()` ▷ filter objective, Eq. 12
5: **for** $c \leftarrow 1\,..\,k$ **do** ▷ cost-based filtering
6: **for** $\ell \leftarrow 1\,..\,n$ where $|D(x_\ell)| > 1$ **do**
7: ▷ Only consider non-redundant assumptions, hence the test below
8: **if** $c \in D(x_\ell) \wedge \neg$`solution.hasFlow(`$x_\ell$`,` c`)` **then**
9: $\delta \leftarrow$ `shortestDistUpdate(solution,` x_ℓ`,` c`)`
10: **if** $\delta = \infty \vee \mathrm{LB}(Z) + \delta \geq \mathrm{UB}(Z)$ **then**
11: $D(x_\ell) \leftarrow D(x_\ell) \setminus \{c\}$ ▷ filtrer if bound exceeded or pb inconsistent

The time and space complexities of Algorithm 4.2 are dominated by Network Simplex when called. Depending on the implementation [19], these vary and can be linked to arc costs. In practice, complexity analysis around Network Simplex rarely represents a faithful depiction of real world performance. The function `impactfulChangeHasOccurred()` runs in $\mathcal{O}(n)$ time. Solving the shortest path problem using the Bellman-Ford algorithm is done in time $\mathcal{O}(qk(k+q))$ due to the graph comprising $\mathcal{O}(qk)$ arcs and $\mathcal{O}(q+k)$ vertices [7]. Overall time complexity for the cost-based filtering of assignment variables is thus $\mathcal{O}(q^2k^2(k+q))$.

5 Search Strategy

The search heuristic discussed in this section is inspired from the one proposed for CPC [13] with two key improvements.

5.1 Bootstrapping from a heuristic solution

To solve MSSC, CPC starts from a feasible, heuristically generated solution whose cost is used for the first domain reductions (recall that CPC makes use of a cost-based filtering mechanism). A superior left branch in our search tree, leading to an initial solution, helps to reduce its size by acting on two separate aspects of the problem: it provides a tighter initial upper bound and it moves potentially unsuccessful alternate, future branches near the top of the search tree to avoid revisiting them repeatedly.

Initial solution generation. Instead of starting from a greedy assignment whose results depend on the order of the observations in O [13] and which produces poor results when user constraints are present, suppose a feasible good-enough initial solution τ_0 is known in advance (which may or may not be globally optimal). We can use τ_0 as a guide for the first n branching assignments of the CP search to ensure the first solution found is equal to τ_0, thus also ensuring the initial upper bound is equal to the cost of τ_0. Generating this solution can be done using any number of existing heuristic methods (some of which have been discussed in Section 3) to solve a constrained MSSC.

The cost of τ_0 is only part of what helps the CP resolution. The order in which individual assignments appear has its importance.

Order of initial assignments. Authors of [11] demonstrate the substantial impact of initial data sequencing on branch and bound searches. They show that solution times can spread over several orders of magnitude for randomly sampled sequences. It is possible to prune off large sub-trees by ordering the data of a heuristically generated solution in an careful manner.

Based on this, given an initial solution τ_0 with initial cluster centers, we suggest two variable orderings in the left branch of the search tree which showed competitive results in our empirical testing:

- *Decreasing distances to own cluster's center*: this method orders assignments in τ_0 from the one whose corresponding observation is farthest from the center of its cluster to the one that is closest. It tries to place potentially disruptive, hard to assign observations near the root of the search tree, where we have greater flexibility to recover from a poor choice.
- *Decreasing minimal distances to other clusters' centers*: this method is similar to the previous one with the difference being that the ordering is based on the minimum distance between each observation and centers of clusters which are not its own. Therefore, it maximizes the likelihood alternate branches will fail the closer they are to the top, eliminating bigger sub-trees.

5.2 Dynamic tie-breaking

Once the left branch has been generated, another branching strategy takes over. The heuristic in CPC is adequate but displays a major weakness. For each unassigned x_i and $c \in D(x_i)$, CPC computes $t_{i,c}$, the WCSS increase on Z if $x_i = c$. It then branches on the variable given by $\operatorname{argmax}_{i\,:\,|D(x_i)|>1} \min_{c \in D(x_i)} t_{i,c}$ [13]. However, ties may occur whenever a cluster becomes empty upon backtracking: if every unassigned variable has this cluster in its domain, the minimum will be zero for all. In that case, the heuristic essentially falls back to a lexicographical one. To correct this, we design a dynamic tie-breaking strategy.

When presented with a tie as a result of a cluster becoming empty, we branch in a way that assigns to the empty cluster the observation whose sum-of-squares between it and other unassigned observations is the highest:

$$o^\star = \operatorname*{argmax}_{o \in U} \sum_{o' \in U} \|o - o'\|^2 \tag{13}$$

Since bound computations in CPC directly involve the sum of squared distances between observations of each cluster, the choice depicted in Equation 13 is akin to a fail-first strategy: we initiate a cluster with the observation which is most likely to produce worse solutions through elevation of its cluster's contribution.

6 Experiments

We compare our CP-centered approach to solving ccMSSC to the works discussed in Section 3. In it, we cited CPC in [13] and CP RBBA in [16] as CP frameworks for solving MSSC. These two approaches can easily be extended to solve ccMSSC through the introduction of adequate cardinality constraints to their CP models. We also cited a numerical method for solving ccMSSC with guarantees on the sub-optimality of the solutions [23] as well as a column generation framework to solve constrained MSSC [3]. However the latter's current implementation does not support solving ccMSSC and would require a significant amount of work to add the necessary constraints.[1]

To carry out our experiments we select 19 instances, summarized in Table 1. All of them are available in the UCI Machine Learning Repository[2] except for `HA` [18] and `RU` [24]. Instances from exact methods presented in Section 3 all appear in Table 1 and have been completed with randomly sampled datasets from the UCI repository with 200 data points or less and with numerical attributes.

Table 1: Description of selected instances

Code	Name	n	s	k	Targeted cluster cardinalities	Notes
AI3	Acute Inflammations	120	6	3	40 40 40	1, 2
AI4	Acute Inflammations	120	6	4	30 30 30 30	1, 2
BC	Breast Cancer Coimbra	116	9	3	38 39 39	1, 2
BT	Breast Tissue	106	9	6	18 17 17 18 18 18	1, 2
CB	Connectionist Bench	208	60	2	111 97	
CS	Concrete Slump Test	103	7	3	34 34 35	1, 2
GI	Glass Identification	214	9	6	70 17 9 29 76 13	
HA	Hatco	100	14	3	34 33 33	1, 2
FI2	Fisher's Iris	150	4	2	60 90	2
FI3	Fisher's Iris	150	4	3	50 50 50	1
PA	Parkinson's	195	22	2	147 48	
PR	Planning Relax	182	12	2	130 52	
RU	Ruspini	75	2	4	18 19 19 19	1, 2
SE	Seeds	210	7	3	70 70 70	1
SY	Soybean	47	35	4	11 12 12 12	1, 2, 3
TH	Thyroid Disease	215	5	3	72 71 72	1, 2, 4
UL	Urban Land Cover	168	147	9	14 16 14 25 15 23 17 15 29	
WNO	Wine	178	13	3	60 40 78	2
WNB	Wine	178	13	3	59 60 59	1, 2

Table 1 legend: (1) Instance with balanced classes; (2) Number of classes and/or target cardinalities decided randomly by us; (3) Multiple versions available, version *Small* used here; (4) Multiple versions available, version with 215 observations used here.

[1] personal communication from one of the authors

[2] https://archive.ics.uci.edu

New algorithms[3] were implemented in C++ using IBM ILOG CPLEX Optimization Studio 12.9.0. CPC was reimplemented using this same software framework. CP RBBA's C++ GECODE implementation is publicly available[4] and was used with slight alterations (to introduce the necessary constraints for ccMSSC).

All algorithms were compiled using Intel C++ Compiler 19.0.3.199 and run on stock Intel Xeon Gold 6148 processors. Each process was allocated 1 GB of memory and one core. Maximum runtime was set to 86400 seconds (i.e., 1 day).

6.1 Impact of dynamic tie-breaking

We present in Table 2 the impact of our dynamic tie-breaking strategy on the performance of CPC. To isolate the effect of the tie-breaking, we do not introduce any cardinality constraints to the model for this specific test. Dashes represent a run that has timed out. Due to space constraints, we show results for 6 instances that faithfully convey the general trend.

Table 2: Impact of dynamic tie-breaking on the performance of CPC

	No tie-breaking			Dynamic sum-of-squares tie-breaking		
Inst.	Time [s]	Fails	Branches	Time [s]	Fails	Branches
BC	886.2	79.08k	158.21k	**56.6**	**4.03k**	**8.06k**
BT	—	—	—	**184**	**7.73k**	**15.45k**
HA	338.4	40.63k	81.27k	**289.8**	**33.56k**	**67.14k**
FI3	1337.2	56.59k	113.21k	**911.5**	**41.08k**	**82.19k**
RU	0.4	83	170	**0.4**	**82**	**168**
SY	3.2	1.69k	3.38k	**2.2**	**1.15k**	**2.31k**

Table 2 shows a clear and generalized improvement brought on by the tie-breaking strategy, both with respect to search space size as well as run time. Notably, *Breast Tissue* can only be solved by applying the dynamic tie-breaking to the search process.

6.2 Resolution of ccMSSC

We suggest in Section 5.1 starting the CP search from known good solutions to the problem and arranging them in the search tree to improve performance. To this end, we make use of two heuristic approaches to start ccMSSC resolution: LIMA-VNS [12] (discussed in Section 3) for the balanced instances and Constrained K-Means Clustering [8] for the others. We use a third-party public implementation[5] for the latter and an executable supplied by the authors for the former.

Both heuristic algorithms are seeded using `/dev/random` and run 10 times for each instance. The median of the 10 runs is picked as a starting point for the CP search.

[3] Source-code can be retrieved from: https://github.com/mnhaouas/card-const-MSSC
[4] https://cp4clustering.github.io/
[5] https://github.com/Behrouz-Babaki/MinSizeKmeans

Each instance is solved 8 times: twice through CP RBBA using two distinct observation orderings recommended by its authors (`FF` for Farthest First and `NN` for Nearest Neighbor) and twice through each of CPC, the basic approach in Section 4.1, and the advanced approach in Section 4.2. For each, we make use of the two observation orderings suggested in Section 5.1 (`OC` for *Decreasing distances to own cluster's center* and `RC` for *Decreasing minimal distances to remote clusters' centers*). We also make use of our tie-breaking strategy to accelerate all methods except CP RBBA due to its fundamentally different nature.

Table 3: ccMSSC resolution statistics for all algorithms

	CP RBBA & GCC			CPC & GCC			Basic filtering			Advanced filtering			
Inst.	Ord.	Time [s]	Fails	Ord.	Time [s]	Fails	Gap [%]	Time [s]	Fails	Gap [%]	Time [s]	Fails	Gap [%]
AI3	NN	–	–	OC	–	–	53.2	–	–	53.2	–	–	38.0
	FF	–	–	RC	–	–	55.4	–	–	55.4	–	–	**37.7**
AI4	NN	–	–	OC	–	–	57.9	–	–	57.8	–	–	30.0
	FF	–	–	RC	–	–	55.7	–	–	55.7	–	–	**29.6**
BC	NN	–	–	OC	–	–	81.6	42.6	29.22k	–	34.7	6.82k	–
	FF	–	–	RC	–	–	81.6	12.2	5.52k	–	**5.4**	**787**	–
BT	NN	–	–	OC	–	–	99.9	23.0	12.98k	–	**9.1**	**886**	–
	FF	–	–	RC	–	–	99.9	–	–	4.0	–	–	3.7
CB	NN	–	–	OC	–	–	36.2	–	–	35.9	–	–	17.4
	FF	–	–	RC	–	–	35.9	–	–	35.6	–	–	**17.3**
CS	NN	–	–	OC	–	–	45.7	–	–	45.7	–	–	**28.0**
	FF	–	–	RC	–	–	45.5	–	–	45.5	–	–	28.1
GI	NN	–	–	OC	–	–	96.5	–	–	86.4	–	–	**55.2**
	FF	–	–	RC	–	–	96.5	–	–	90.3	–	–	58.9
HA	NN	877.1	195.78k	OC	60.1	14.46k	–	6.0	2.58k	–	4.0	559	–
	FF	2.0	161.82k	RC	46.0	11.31k	–	3.7	1.72k	–	**1.4**	**127**	–
FI2	NN	24.7	19.11k	OC	380.3	43.51k	–	10.8	2.53k	–	9.4	1.02k	–
	FF	2.0	91.56k	RC	23.9	6.17k	–	3.2	706	–	**1.5**	**142**	–
FI3	NN	18.9	1.30M	OC	2498.7	207.59k	–	321.6	58.07k	–	147.6	9.94k	–
	FF	–	–	RC	350.6	33.49k	–	54.3	11.93k	–	**7.8**	**530**	–
PA	NN	–	–	OC	–	–	79.6	–	–	68.0	27022.1	1.74M	–
	FF	–	–	RC	–	–	79.6	–	–	68.0	**24867.4**	**1.66M**	–
PR	NN	–	–	OC	–	–	49.3	–	–	40.4	–	–	**25.2**
	FF	–	–	RC	–	–	49.3	–	–	40.4	–	–	**25.2**
RU	NN	1056.9	103.27M	OC	5290.6	3.52M	–	157.5	167.28k	–	107.1	16.58k	–
	FF	5192.1	586.64M	RC	3341.4	2.21M	–	38.8	50.73k	–	**9.3**	**1.74k**	–
SE	NN	–	–	OC	–	–	47.7	–	–	46.8	–	–	24.2
	FF	–	–	RC	–	–	47.7	–	–	46.8	**37197.0**	**1.69M**	–
SY	NN	13543.4	2.51G	OC	442.3	766.06k	–	6.0	16.43k	–	13.1	3.54k	–
	FF	58074.8	11.67G	RC	99.7	283.89k	–	**0.9**	**2.84k**	–	1.8	643	–
TH	NN	–	–	OC	–	–	87.4	–	–	83.3	–	–	**37.8**
	FF	–	–	RC	–	–	87.4	–	–	83.3	–	–	**37.8**
UL	NN	–	–	OC	–	–	92.2	–	–	60.8	–	–	**30.3**
	FF	–	–	RC	–	–	92.2	–	–	60.8	–	–	**30.3**
WN0	NN	–	–	OC	–	–	70.6	32239.3	6.60M	–	**6420.8**	**567.35k**	–
	FF	–	–	RC	–	–	70.6	58373.0	9.28M	–	11335.3	755.86k	–
WNB	NN	–	–	OC	–	–	69.4	497.7	154.16k	–	240.5	31.71k	–
	FF	–	–	RC	–	–	69.4	95.4	15.69k	–	**13.7**	**853**	–

Table 3 shows a clear advantage for both variants of the filtering algorithms proposed and particularly for the advanced, flow-based approach. Eight of the 19 instances could not be solved to optimality. However, on the flip side, the advanced approach is capable of solving two instances none of the other methods could solve in the allotted time. The basic approach, while fast, produces lower quality bounds and loses its advantage to substantially bigger search trees.

Overall, ordering `RC` yields the best results. However, it is not superior for all instances shown. For example, `OC` is best for the non balanced version of *Wine* as well as *Breast Tissue*.

CPC is also able to take advantage of our improved search strategy to show competitive results compared to CP RBBA, tighter bounds computed by the latter notwithstanding. Without our improved search, CPC with GCC is only able to solve `HA`, `RU` and `SY` in 12398, 15462 and 6006 seconds respectively. This confirms the important role of a reinforced search strategy for ccMSSC.

6.3 Comparison with IBM CP Optimizer default search

In order to frame the performance of our search strategy in a recognizable reference, we compared it to the default strategy shipped with IBM CP Optimizer. Solving the 6 instances in Table 2 as ccMSSC problems using the advanced filtering method yielded, on average, search trees 27 times bigger for CP Optimizer search and run times were increased by a factor of 20.

6.4 Comparison with the conic optimization approach

The semidefinite programming lower bound and the rounding heuristic of [23] were able to prove the optimality of `FI3`, `SE`, `PR`, and `PA` in 584, 3823, 2637, and 2000 seconds, respectively, thus surpassing our best advanced filtering approach except for `FI3` where we show vastly improved results. They are also able to guarantee a solution to `CB` and `UL` with gaps of 0.001% and 3% respectively while our approach's best gaps are equal to 17% and 30% respectively for these instances.

However this numerical method does not allow easy expression of user constraints (ours can leverage the flexibility of CP to quickly and easily introduce any extra constraints). Besides, our CP method is designed as a global optimization method which ends its execution only when all possibilities in the search space have been exhausted. The method of [23] is not conceived towards obtaining the global optimum of the problem shall the upper bound produced by the rounding method not coincide with the lower bound obtained via the semidefinite programming relaxation.

7 Conclusion

We presented in this paper a CP approach for exact resolution of the cardinality-constrained MSSC problem. We suggest both a bolstered search strategy as well as a global constraint with two distinct filtering schemes: a basic one and a more advanced one. Experiments on widely used data sets confirm our approach outperforms previously available exact methods for solving ccMSSC.

Our work can be improved upon by identifying ways that can extend our global constraint developed for ccMSSC to support soft cardinality constraints where deviations from target cardinalities could be allowed if it meant obtaining a lower cost solution. Moreover, as seen previously, performance is heavily dependent on bound quality. Therefore, looking for more innovative ways to fully exploit the structure of ccMSSC for even tighter bounds could be another avenue for future research.

Acknowledgements

Financial support from a Natural Sciences and Engineering Research Council of Canada (NSERC) graduate scholarship is gratefully acknowledged.

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, May 2009.
2. Daniel Aloise and Pierre Hansen. Evaluating a branch-and-bound rlt-based algorithm for minimum sum-of-squares clustering. *Journal of Global Optimization*, 49(3):449–465, Mar 2011.
3. Behrouz Babaki, Tias Guns, and Siegfried Nijssen. Constrained clustering using column generation. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 438–454, Cham, 2014. Springer International Publishing.
4. Maria Fiorina Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general topologies. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 1995–2003, USA, 2013. Curran Associates Inc.
5. Arindam Banerjee and Joydeep Ghosh. Scalable clustering algorithms with balancing constraints. *Data Mining and Knowledge Discovery*, 13(3):365–395, Nov 2006.
6. Sugato Basu, Ian Davidson, and Kiri Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall/CRC, 1 edition, 2008.
7. Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.
8. Kristin P. Bennett, Paul S. Bradley, and Ayhan Demiriz. Constrained k-means clustering. Technical Report MSR-TR-2000-65, Microsoft Research, May 2000.
9. Alberto Bertoni, Massimiliano Goldwurm, Jianyi Lin, and Francesco Saccà. Size constrained distance clustering: Separation properties and some complexity results. *Fundamenta Informaticae*, 115:125–139, 01 2012.
10. Michael J. Brusco. A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning. *Psychometrika*, 71(2):347–363, Jun 2006.
11. Réal A. Carbonneau, Gilles Caporossi, and Pierre Hansen. Extensions to the repetitive branch and bound algorithm for globally optimal clusterwise regression. *Computers Operations Research*, 39(11):2748 – 2762, 2012.
12. Leandro R. Costa, Daniel Aloise, and Nenad Mladenović. Less is more: basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering. *Information Sciences*, 415-416:247 – 253, 2017.
13. Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained minimum sum of squares clustering by constraint programming. In Gilles Pesant, editor, *Principles and Practice of Constraint Programming*, pages 557–573, Cham, 2015. Springer International Publishing.
14. Thi-Bich-Hanh Dao, Khanh-Chuong Duong, and Christel Vrain. Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70 – 94, 2017. Combining Constraint Solving with Mining and Learning.
15. Jacques Desrosiers, Nenad Mladenović, and Daniel Villeneuve. Design of balanced mba student teams. *Journal of the Operational Research Society*, 56(1):60–66, 2005.

16. Tias Guns, Thi-Bich-Hanh Dao, Christel Vrain, and Khanh-Chuong Duong. Repetitive branch-and-bound using constraint programming for constrained minimum sum-of-squares clustering. In *Proceedings of the Twenty-second European Conference on Artificial Intelligence*, ECAI'16, pages 462–470, Amsterdam, The Netherlands, The Netherlands, 2016. IOS Press.
17. Lars Hagen and Andrew B. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(9):1074–1085, Sep. 1992.
18. Joseph F. Hair, Ronald L. Tatham, Rolph E. Anderson, and William Black. *Multivariate Data Analysis*. Pearson, New York, NY, fifth edition, 1998.
19. Dieter Jungnickel. *The Network Simplex Algorithm*, pages 321–339. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.
20. Yat Chiu Law and Jimmy H. M. Lee. Global constraints for integer and set value precedence. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 362–376, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
21. Claude-Guy Quimper, Alejandro López-Ortiz, Peter van Beek, and Alexander Golynski. Improved algorithms for the global cardinality constraint. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 542–556, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
22. Jean-Charles Régin. Arc consistency for global cardinality constraints with costs. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming – CP'99*, pages 390–404, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
23. Napat Rujeerapaiboon, Kilian Schindler, Daniel Kuhn, and Wolfram Wiesemann. Size matters: Cardinality-constrained clustering and outlier detection via conic optimization. *SIAM Journal on Optimization*, 29(2):1211–1239, 2019.
24. Enrique H. Ruspini. Numerical methods for fuzzy clustering. *Information Sciences*, 2(3):319 – 350, 1970.
25. W. Tang, Y. Yang, L. Zeng, and Y. Zhan. Size constrained clustering with milp formulation. *IEEE Access*, 8:1587–1599, 2020.
26. Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 577–584, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
27. Toby Walsh. Symmetry breaking constraints: Recent results. In *AAAI Conference on Artificial Intelligence*, 2012.
28. Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, Jan 2008.