

# SOS-SDP: an Exact Solver for Minimum Sum-of-Squares Clustering

Veronica Piccialli, Antonio M. Sudoso, Angelika Wiegele

April 23, 2021

The minimum sum-of-squares clustering problem (MSSC) consists in partitioning  $n$  observations into  $k$  clusters in order to minimize the sum of squared distances from the points to the centroid of their cluster. In this paper, we propose an exact algorithm for the MSSC problem based on the branch-and-bound technique. The lower bound is computed by using a cutting-plane procedure where valid inequalities are iteratively added to the Peng-Wei SDP relaxation. The upper bound is computed with the constrained version of  $k$ -means where the initial centroids are extracted from the solution of the SDP relaxation. In the branch-and-bound procedure, we incorporate instance-level must-link and cannot-link constraints to express knowledge about which instances should or should not be grouped together. We manage to reduce the size of the problem at each level preserving the structure of the SDP problem itself. The obtained results show that the approach allows to successfully solve for the first time real-world instances up to 4000 data points.

## 1 Introduction

Clustering is the task of partitioning a set of objects into homogeneous and/or well-separated groups, called clusters. Cluster analysis is the discipline that studies methods and algorithms for clustering objects according to a suitable similarity measure. It belongs to unsupervised learning since it does not use class labels. Two main clustering approaches exist: hierarchical clustering, which assumes a tree structure in the data and builds nested clusters, and partitional clustering. Partitional clustering generates all the clusters at the same time without assuming a nested structure. Among partitional clustering, the minimum sum-of-squares clustering problem (MSSC) or sum-of-squares (SOS) clustering, is one of the most popular and well studied. MSSC asks to partition  $n$  given data points into  $k$  clusters so that the sum of the Euclidean distances from each data point to the cluster centroid is minimized.

The MSSC commonly arises in a wide range of disciplines and applications, as for example image segmentation [18, 56], credit risk evaluation [11], biology [31], customer segmentation [57], document clustering [43], and as a technique for the missing values imputation [63].

The MSSC can be stated as follows.

$$\min \sum_{i=1}^n \sum_{j=1}^k x_{ij} \|p_i - c_j\|^2 \quad (1a)$$

$$\text{s.t. } \sum_{j=1}^k x_{ij} = 1, \quad \forall i \in \{1, \dots, n\} \quad (1b)$$

$$\sum_{i=1}^n x_{ij} \geq 1, \quad \forall j \in \{1, \dots, k\} \quad (1c)$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\} \forall j \in \{1, \dots, k\} \quad (1d)$$

$$c_j \in \mathbb{R}^d, \quad \forall j \in \{1, \dots, k\}. \quad (1e)$$

Here,  $p_i \in \mathbb{R}^d$ ,  $i \in \{1, \dots, n\}$ , are the data points, and the centers of the  $k$  clusters are at the (unknown) points  $c_j$ ,  $j \in \{1, \dots, k\}$ . For convenience, we sometimes collect all the data points  $p_i$  as rows in a matrix  $W_p$ . The binary decision variable  $x_{ij}$  expresses whether data point  $i$  is assigned to cluster  $j$  or not. Constraints (1b) make sure that each point is assigned to a cluster, and constraints (1c) guarantee that none of the  $k$  clusters is empty.

Setting the gradient of the objective function with respect to  $c$  to zero yields

$$\sum_{i=1}^n x_{ij} (c_j^r - p_i^r) = 0, \quad \forall j \in \{1, \dots, k\} \forall r \in \{1, \dots, d\}$$

and we obtain the formula for the point in the center of each cluster

$$c_j^r = \frac{\sum_{i=1}^n x_{ij} p_i^r}{\sum_{i=1}^n x_{ij}}, \quad \forall j \in \{1, \dots, k\} \forall r \in \{1, \dots, d\}.$$

Replacing the formula for  $c$  in (1), we get

$$\min \sum_{i=1}^n \sum_{j=1}^k x_{ij} \left\| p_i - \frac{\sum_{l=1}^n x_{lj} p_l}{\sum_{l=1}^n x_{lj}} \right\|^2 \quad (2a)$$

$$\text{s.t. } \sum_{j=1}^k x_{ij} = 1, \quad \forall i \in \{1, \dots, n\} \quad (2b)$$

$$\sum_{i=1}^n x_{ij} \geq 1, \quad \forall j \in \{1, \dots, k\} \quad (2c)$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, \dots, n\} \forall j \in \{1, \dots, k\}. \quad (2d)$$

## 1.1 Literature Review

The MSSC is known to be NP-hard in  $\mathbb{R}^2$  for general values of  $k$  [42], and in higher dimension even for  $k = 2$  [4]. The one-dimensional case is proven to be solvable in polynomial time. In particular, [60] proposed an  $O(kn^2)$  time and  $O(kn)$  space dynamic programming algorithm for solving this special case. Because of MSSC's computational complexity, heuristic approaches and approximate algorithms are usually preferred over exact methods.

The most popular heuristic for solving MSSC is  $k$ -means [41, 40], that alternates the centroid initialization with the assignments of points until centroids do not move anymore. The main disadvantage of  $k$ -means is that it produces locally optimal solutions that can be far from the global minimum, and it is extremely sensitive to the initial assignment of centroids. For this reason, a lot of research has been dedicated to finding efficient initialization for  $k$ -means (see for example [6, 62, 25] and references therein). However, an efficient initialization may not be enough in some instances, so that different strategies have been implemented in order to improve the exploration capability of the algorithm. A variety of heuristics and metaheuristics have been proposed, following the standard metaheuristic framework, e.g., simulated annealing [37], tabu search [1], variable neighborhood search [28, 47], iterated local search [39], evolutionary algorithms [45, 33]). In the work of [58, 8, 34, 35], DC (Difference of Convex functions) programming is used to define efficient heuristic algorithms for clustering large datasets. The algorithm  $k$ -means has also been used as a local search subroutine in different algorithms, as in the population-based metaheuristic developed in [27] and in the differential evolution scheme proposed in [44].

Recently, thanks to the enhancements in computers' computational power and to the progress in mathematical programming, the exact resolution of MSSC has become way more achievable. In this direction, mathematical programming algorithms based on branch-and-bound and column generation have produced guaranteed globally optimal solutions for small and medium scale instances. Due to the NP-hardness of the MSSC, the computational time of globally optimal algorithms quickly increases with the size of the problem. However, besides the importance of finding optimal solutions for some clustering applications, certified optimal solutions remain extremely valuable as a benchmark tool since they can be used for evaluating, improving, and developing heuristics and approximate methods. Compared to the huge number of papers proposing heuristics and approximate methods for the MSSC problem, the number of articles proposing exact algorithms is much smaller.

One of the earliest attempts was the integer programming formulation proposed by Rao [51], which requires the cluster sizes to be fixed in advance and is limited to small instances. A first branch-and-bound algorithm was proposed by Koontz et al. [36] and extended by Diehr [20]. The idea is to use partial clustering solutions on a subset  $S$  of the main dataset  $D$  to determine improved bounds and clusters on the entire sample by a branch-and-bound search. The key observation is that the optimal objective function value of the MSSC on  $D$  is greater or equal than the optimal objective function value of the MSSC on  $S$  plus the optimal objective function value of the MSSC on  $D - S$ .

This approach was later improved by Brusco [9], who developed a repetitive-branch-and-bound algorithm (RBBA). After a proper reordering of the entities in  $D$ , RBBA solves a sequence of subproblems of increasing size with the branch-and-bound technique. While performing a branch-and-bound for a certain subproblem, Brusco’s algorithm exploits the optimal solutions found for the previous subproblems which provide tighter bounds compared to the ones used by [36] and [20]. RBBA provided optimal solutions for well separated synthetic datasets with up to 240 objects. Poorly separated problems with no inherent cluster structure were optimally solved for up to 60 objects. Sherali and Desai [55] proposed a different branch-and-bound algorithm where tight lower bounds are determined by using the reformulation-linearization-technique (RLT), see Sherali and Adams [54]. The authors claim that this algorithm allows for the exact resolution of problems of size up to 1000 entities, but those results seem to be hard to reproduce. The computing times in an attempted replication by Aloise and Hansen [3] were already high for real datasets with about 20 objects.

A column generation algorithm for MSSC was proposed by Du Merle et al. [21]. The master problem is solved by an interior point method, whereas the auxiliary problem of finding a column with negative reduced cost is expressed as a hyperbolic program with binary variables. Variable-neighborhood-search heuristics are used to find a good initial solution and to accelerate the resolution of the auxiliary problem. This approach has been considered a successful one, since it solved for the first time medium size benchmark instances (i.e., instances with 100–200 entities), including the popular Iris dataset, which encounters 150 entities. However, the bottleneck of the algorithm lies in the resolution of the auxiliary problem, and more precisely, in the unconstrained quadratic 0-1 optimization problem. Later this algorithm was further improved by Aloise et al. [5] who define a different geometric-based approach for solving the auxiliary problem. In particular, the solution of the auxiliary problem is achieved by solving a certain number of convex quadratic problems. If the points to be clustered are in the plane, the maximum number of convex problems to solve is polynomially bounded. When the points are not in the plane, in order to solve the auxiliary problems the cliques in a certain graph (induced by the current solution of the master problem) have to be found. The algorithm is more efficient when the graph is sparse, and the graph becomes sparser when the number of clusters  $k$  increases. Therefore, the algorithm proposed in Aloise et al. [5] is particularly efficient in the plane and when  $k$  is large. Their method was able to provide exact solutions for large scale problems, including one instance of 2300 entities when the ratio between  $n$  and  $k$  is small.

Recently, Peng and Wei [48] by using matrix arguments proved the equivalence between the MSSC formulation and a model called 0-1 semidefinite programming (SDP), in which the eigenvalues of the matrix variable are binary. Using this result, Aloise and Hansen [2] proposed a branch-and-cut algorithm for MSSC where lower bounds are obtained from the linear programming relaxation of the 0-1 SDP model. This algorithm manages to obtain exact solutions for datasets up to 200 entities with computing times comparable with those obtained by the column generation method proposed by Du Merle et al. [21].

Constant-factor approximation algorithms have also been developed in the literature,

both for fixed number of clusters  $k$  and for fixed dimension  $d$  [32]. Among these methods, Peng and Wei [48] proposed a rounding procedure to extract a feasible solution of the original MSSC from the approximate solution of the relaxed SDP problem. More in detail, they use the Principal Component Analysis (PCA) to reduce the dimension of the dataset and then perform clustering on the projected PCA space. They showed that this algorithm can provide a 2-approximate solution to the MSSC. More recently, Prasad and Hanasusanto [50] proposed a new approximation algorithm that utilizes an improved copositive conic reformulation of the MSSC. Starting from this reformulation, the authors derived a hierarchy of accurate SDP relaxations obtained by replacing the completely positive cone with progressively tighter semidefinite outer approximations. Their SDP relaxations provide better lower bounds than the Peng-Wei one but do not scale well when the size of the problem increases.

## Main results and outline

The main contributions of this paper are the following:

- (i) we define the first SDP based branch-and-bound algorithm for MSSC, and we use a cutting-plane procedure for strengthening the bound, following a recent strand of research [17];
- (ii) we define a shrinking procedure that allows reducing the size of the problem when introducing must link constraints;
- (iii) we exploit the SDP solution for a smart initialization of the constrained version of  $k$ -means that yields high quality upper bounds;
- (iv) for the first time, we manage to find the exact solution for instances of size up to  $n = 4000$ .

This paper is structured as follows. In Section 2 we introduce equivalent formulations for the MSSC and derive relaxations based on semidefinite programming (SDP). In Section 3 we analyze the SDP problems that arise at each node within the branch-and-bound tree and discuss the selection of the branching variable. In Section 4 the details about the bound computation are discussed, including a post-processing procedure that produces a “safe” bound from an SDP that is solved to medium precision only. Section 5 gives all the details on the heuristic used to generate feasible clusterings. The details of our implementation and exhaustive numerical results are presented in Section 6. Finally, Section 7 concludes the paper.

## Notation

Let  $\mathcal{S}^n$  denote the set of all  $n \times n$  real symmetric matrices. We denote by  $M \succeq 0$  that matrix  $M$  is positive semidefinite and let  $\mathcal{S}_+^n$  be the set of all positive semidefinite matrices of order  $n \times n$ . We denote by  $\langle \cdot, \cdot \rangle$  the trace inner product. That is, for any  $M, N \in \mathbf{R}^{n \times n}$ , we define  $\langle M, N \rangle := \text{trace}(M^\top N)$ . Its associated norm is the Frobenius

norm, denoted by  $\|M\|_F := \sqrt{\text{trace}(M^\top M)}$ . We define the linear map  $\mathcal{A} : \mathcal{S}^n \rightarrow \mathbb{R}^{m_1}$  as  $(\mathcal{A}(X))_i = \langle A_i, X \rangle$ , where  $A_i \in \mathcal{S}^n$ ,  $i = 1, \dots, m_1$ , and the linear map  $\mathcal{B} : \mathcal{S}^n \rightarrow \mathbb{R}^{m_2}$  as  $(\mathcal{B}(X))_i = \langle B_i, X \rangle$ , where  $B_i \in \mathcal{S}^n$ ,  $i = 1, \dots, m_2$ .

We define by  $e_n$  the vector of all ones of length  $n$ . We omit the subscript in case the dimension is clear from the context. We denote by  $E_i$  the symmetric matrix such that  $\langle E_i, Z \rangle$  is the sum of row  $i$  of  $Z$ .

## 2 A Lower Bound based on Semidefinite Programming

We briefly remind the Peng-Wei SDP relaxation to Problem (2) that will be the basis of the bounding procedures within our exact algorithm.

Consider matrix  $W$  where the entries are the inner products of the data points, i.e.,  $W_{ij} = p_i^\top p_j$  for  $i, j \in \{1, \dots, n\}$ . Furthermore, collect the binary decision variables  $x_{ij}$  from (2) in the  $n \times k$  matrix  $X$  and define matrix  $Z$  as

$$Z = X(X^\top X)^{-1}X^\top.$$

Peng and Wei [48] introduced a different but equivalent formulation for the MSSC, yielding the following optimization problem:

$$\min \langle -W, Z \rangle \tag{3a}$$

$$\text{s.t. } Ze = e \tag{3b}$$

$$\text{tr}(Z) = k \tag{3c}$$

$$Z \geq 0, Z^2 = Z, Z = Z^\top. \tag{3d}$$

We can convert Problem (3) into a rank constrained optimization problem. In fact we can replace the constraints  $Z^2 = Z$  and  $Z = Z^\top$  with a rank constraint and a positive semidefiniteness constraint on  $Z$ , yielding the following problem:

$$\min \langle -W, Z \rangle \tag{4a}$$

$$\text{s.t. } Ze = e \tag{4b}$$

$$\text{tr}(Z) = k \tag{4c}$$

$$Z \geq 0, Z \in \mathcal{S}_+^n \tag{4d}$$

$$\text{rank}(Z) = k. \tag{4e}$$

In order to prove the equivalence of Problems (3) and (4), we need the definition of an idempotent matrix and its characterization in terms of eigenvalues given by Lemma 2.2.

**Definition 2.1.** *A symmetric matrix  $Z$  is idempotent if  $Z^2 = ZZ = Z$ .*

**Lemma 2.2.** *A symmetric matrix  $Z$  is idempotent if and only if all its eigenvalues are either 0 or 1.*

*Proof.* Let  $Z$  be idempotent,  $\lambda$  be an eigenvalue and  $v$  a corresponding eigenvector then  $\lambda v = Zv = ZZv = \lambda Zv = \lambda^2 v$ . Since  $v \neq 0$  we find  $\lambda - \lambda^2 = \lambda(1 - \lambda) = 0$  so either  $\lambda = 0$  or  $\lambda = 1$ .

To prove the other direction, consider the eigenvalue decomposition of  $Z$ ,  $Z = P\Lambda P^\top$ , where  $\Lambda$  is a diagonal matrix having the eigenvalues 0 and 1 on the diagonal, and  $P$  is orthogonal. Then, since  $\Lambda^2 = \Lambda$ , we get

$$Z^2 = P\Lambda P^\top P\Lambda P^\top = P\Lambda^2 P^\top = P\Lambda P^\top = Z.$$

□

**Theorem 2.3.** *Problems (3) and (4) are equivalent.*

*Proof.* Let  $Z$  be a feasible solution of Problem (3). We first show that  $Z^2 = Z$  and  $Z = Z^\top$  imply  $Z \in \mathcal{S}_+^n$ . In fact, for all  $v$  we have:

$$v^\top Zv = v^\top Z^2v = v^\top ZZv = v^\top Z(v^\top Z^\top)^\top = (v^\top Z)(v^\top Z)^\top = \|v^\top Z\|_2^2 \geq 0.$$

Since  $Z$  is symmetric idempotent, the number of eigenvalues equal to 1 is  $\text{tr}(Z) = \text{rank}(Z) = k$ .

To prove the other direction, let  $Z$  be a feasible solution of Problem (4). If  $\text{rank}(Z) = k$ , then  $Z$  has  $n - k$  eigenvalues equal to 0. Furthermore, let  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$  be the eigenvalues of  $Z$ , then

$$\text{tr}(Z) = \sum_{i=1}^n \lambda_i = \sum_{i=1}^k \lambda_i + \sum_{i=k+1}^n \lambda_i = \sum_{i=1}^k \lambda_i = k.$$

Constraints  $Z \succeq 0$ ,  $Z \geq 0$  and  $Ze = e$  imply that the eigenvalues of  $Z$  are bounded by one (see, e.g., Lemma 4.1). Hence, the trace constraint is satisfied if and only if the positive eigenvalues are all equal to 1. This shows that  $\lambda(Z) \in \{0, 1\}$  and therefore  $Z$  is symmetric idempotent. □

By dropping the non-convex rank constraint from Problem (4), we obtain the SDP relaxation which is the convex optimization problem

$$\min \langle -W, Z \rangle \tag{5a}$$

$$\text{s.t. } Ze = e \tag{5b}$$

$$\text{tr}(Z) = k \tag{5c}$$

$$Z \geq 0, Z \in \mathcal{S}_+^n \tag{5d}$$

## 2.1 Strengthening the Bound through Inequalities

The SDP relaxation (5) can be tightened by adding valid inequalities and solving the resulting SDP in a cutting-plane fashion. In this section, we present the class of inequalities we use for strengthening the bound. For each class, we describe the separation routine used.

We consider three different sets of inequalities:

**Pair inequalities.** In any feasible solution of (4), it holds that

$$Z_{ij} \leq Z_{ii}, \quad Z_{ij} \leq Z_{jj} \quad \forall i, j \in \{1, \dots, n\}, i \neq j. \quad (6)$$

This set of  $n(n-1)$  inequalities were used by Peng and Xia [49] and in the branch-and-cut proposed by Aloise and Hansen [2].

**Triangle Inequalities.** The triangle inequalities are based on the observation that if points  $i$  and  $j$  are in the same cluster and points  $j$  and  $h$  are in the same cluster, then points  $i$  and  $h$  necessarily must be in the same cluster. The resulting  $3\binom{n}{3}$  inequalities are:

$$Z_{ij} + Z_{ih} \leq Z_{ii} + Z_{jh} \quad \forall i, j, h \in \{1, \dots, n\}, i, j, h \text{ distinct}. \quad (7)$$

These inequalities were already introduced by Peng and Xia [49], and used also by Aloise and Hansen [2].

**Clique Inequalities.** If the number of clusters is  $k$ , for any subset  $Q$  of  $k+1$  points at least two points have to be in the same cluster (meaning that at least one  $Z_{ij}$  needs to be positive and equal to  $Z_{ii}$  for all  $(i, j) \in Q$ ). This can be enforced by the following inequalities:

$$\sum_{(i,j) \in Q, i < j} Z_{ij} \geq \frac{1}{n-k+1} \quad \forall Q \subset \{1, \dots, n\}, |Q| = k+1. \quad (8)$$

These  $\binom{n}{k+1}$  inequalities are similar to the clique inequalities for the  $k$ -partitioning problem [13], the difference lies in the right hand side, that in that case is equal to 1, whereas here we use the smallest possible value that an element on the diagonal of  $Z$  can hold.

Pair and triangle inequalities are known to be valid for Problem (3), see [49]. It remains to show that also the clique inequalities are valid.

**Lemma 2.4.** *The clique inequalities (8) are valid for Problem (3).*

*Proof.* The left hand side of (8) has  $\binom{k+1}{2}$  terms, and we know that  $Z_{ii} \geq \frac{1}{n-k+1}$ , since the cardinality of a cluster can be at most  $n-k+1$ . Given that the number of clusters is  $k$ , for any set of  $k+1$  points at least two points have to be in the same cluster, say points  $i$  and  $j$ . Then, for any feasible clustering  $Z$ , at least the element  $Z_{ij}$  in the left hand side of (8) needs to be different from zero, therefore equal to  $Z_{ii}$ , and hence (8) must hold.  $\square$

### 3 Branching: Subproblems within a Branch-and-Bound Algorithm and Variable Selection

Our final goal is to develop a branch-and-bound scheme to solve the MSSC to optimality using relaxation (5) strengthened by some of the inequalities (6)–(8). In this section



we examine the problems that arise after branching. To keep the presentation simple and since everything carries over in a straightforward way, we omit in this section the inclusion of inequalities (6)–(8).

The branching decisions are as follows. Given a pair  $(i, j)$ ,

- points  $p_i$  and  $p_j$  should be in different clusters, i.e., they *cannot link* or
- points  $p_i$  and  $p_j$  should be in the same cluster, i.e., they *must link*.

By adding constraints due to the branching decisions, the problem changes. However, the structure of the SDP remains similar. In this section we describe the subproblems to be solved at each node in the branch-and-bound tree. Each such SDP is of the form

$$\min \langle -\mathcal{T}^\ell W(\mathcal{T}^\ell)^\top, Z^\ell \rangle \quad (9a)$$

$$\text{s.t. } Z^\ell e^\ell = e \quad (9b)$$

$$\langle \text{Diag}(e^\ell), Z^\ell \rangle = k \quad (9c)$$

$$Z_{ij}^\ell = 0 \quad (i, j) \in \text{CL} \quad (9d)$$

$$Z^\ell \geq 0, \quad Z^\ell \in \mathcal{S}_{n-\ell}^+ \quad (9e)$$

where CL (cannot link) is the set of pairs that must be in different clusters and  $\mathcal{T}^\ell$  and  $e^\ell$  encode the branching decisions that ask data points to be in the same cluster (i.e., they must link). We describe this in more detail in the subsequent sections.

### 3.1 Decison $i$ and $j$ are in different clusters

In this case, we add the constraint  $Z_{ij} = 0$  to the SDP, i.e., we add the pair  $(i, j)$  to the set CL.

### 3.2 Decison $i$ and $j$ are in the same cluster

Assume at the current node we have  $n$  points and we decide that on this branch the two points  $p_i$  and  $p_j$  have to be in the same cluster. We can reduce the size of  $W_p$  (the matrix having data points  $p_i$  as rows) by substituting row  $i$  by  $p_i + p_j$  and omitting row  $j$ . To formalize this procedure, we introduce the following notation.

Let  $b(r) = (i, j)$ ,  $i < j$ , be the branching pair in branching decision at level  $r$  and  $b(1), \dots, b(\ell)$  a sequence of consecutive branching decisions. Furthermore, let  $g(r) = (\underline{i}, \underline{j})$  be the corresponding global indices.

Define  $\mathcal{T}^\ell \in \{0, 1\}^{(n-\ell) \times n}$  as

$$\mathcal{T}^\ell = T^{b(\ell)} T^{b(\ell-1)} \dots T^{b(1)}$$

where the  $(n-r) \times (n-r+1)$  matrix  $T^{b(r)}$  for branching decision  $b(r) = (i, j)$  is defined by

$$T_{s,\cdot}^{b(r)} = \begin{cases} u_s & \text{if } 1 \leq s < i \text{ and } i+1 \leq s \leq j \\ u_i + u_j & \text{if } s = i \\ u_{s+1} & \text{if } j < s \leq n-r \end{cases}$$

with  $u_s$  being the unit vector of size  $(n - r + 1)$ . Furthermore, we define  $T^{b(0)} = I_n$ .

Note that  $T^{b(r)} \cdot M$  builds a matrix of size  $(n - r) \times (n - r + 1)$  by adding rows  $i$  and  $j$  of  $M$  and putting the result into row  $i$  while row  $j$  is removed and all other rows remain the same.

We also define the vector  $e^\ell \in \mathbb{R}^{n-\ell}$  as

$$e^\ell = \mathcal{T}^\ell e$$

where  $e$  is the vector of all ones of length  $n$ .

*Remark 3.1.* Note that in  $(e^\ell)$  the number of points that have been fixed to belong to the same cluster along the branching decisions  $b(1), \dots, b(\ell)$  are given. Furthermore,  $\mathcal{T}^\ell (\mathcal{T}^\ell)^\top = \text{Diag}(e^\ell)$ .  $\triangle$

We now show that this shrinking operation corresponds to the must-link branching decisions. Consider the following two semidefinite programs.

$$\min - \langle \mathcal{T}^\ell W (\mathcal{T}^\ell)^\top, Z^\ell \rangle \tag{10a}$$

$$\text{s.t. } Z^\ell e^\ell = e_{n-\ell} \tag{10b}$$

$$\langle \mathcal{T}^\ell (\mathcal{T}^\ell)^\top, Z^\ell \rangle = k \tag{10c}$$

$$Z^\ell \geq 0, Z^\ell \in \mathcal{S}_{n-\ell}^+ \tag{10d}$$

and

$$\min - \langle W, Z \rangle \tag{11a}$$

$$\text{s.t. } Ze = e \tag{11b}$$

$$\langle I, Z \rangle = k \tag{11c}$$

$$Z_{i\cdot} = Z_{j\cdot} \quad \forall \{i, j\} \in g(l), \quad l \in \{1, \dots, \ell\} \tag{11d}$$

$$Z \geq 0, Z \in \mathcal{S}_+^n \tag{11e}$$

**Theorem 3.2.** *Problems (10) and (11) are equivalent.*

*Proof.* Let  $Z^\ell$  be a feasible solution of Problem (10). Define  $Z = (\mathcal{T}^\ell)^\top Z^\ell \mathcal{T}^\ell$ . This is equivalent to expanding the matrix by replicating the rows according to branching decisions. Therefore, (11d) holds by construction. Clearly,  $Z \geq 0$  and  $Z \in \mathcal{S}_+^n$  hold as well. Moreover, we have that

$$\langle I, Z \rangle = \langle I, (\mathcal{T}^\ell)^\top Z^\ell \mathcal{T}^\ell \rangle = \langle \mathcal{T}^\ell (\mathcal{T}^\ell)^\top, Z^\ell \rangle = k$$

and

$$Ze = (\mathcal{T}^\ell)^\top Z^\ell \mathcal{T}^\ell e = (\mathcal{T}^\ell)^\top Z^\ell e^\ell = (\mathcal{T}^\ell)^\top e_{n-\ell} = e_n.$$

Furthermore,

$$\langle W, Z \rangle = \langle W, (\mathcal{T}^\ell)^\top Z^\ell \mathcal{T}^\ell \rangle = \langle (\mathcal{T}^\ell) W (\mathcal{T}^\ell)^\top, Z^\ell \rangle$$

and thus  $Z$  is a feasible solution of Problem (11) and the values of the objective functions coincide.

We next prove that any feasible solution of Problem (11) can be transformed into a feasible solution of Problem (10) with the same objective function value. In order to do so, we define the matrix

$$\mathcal{D}^\ell = \text{Diag}(1/e^\ell)$$

where  $1/e^\ell$  denotes the vector that takes the inverse elementwise. It is straightforward to check that

$$\mathcal{D}^\ell \mathcal{T}^\ell (\mathcal{T}^\ell)^\top \mathcal{D}^\ell = \mathcal{D}^\ell.$$

Assume that  $Z$  is a feasible solution of Problem (11) and set  $Z^\ell = \mathcal{D}^\ell \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top \mathcal{D}^\ell$ . If  $Z$  is nonnegative and positive semidefinite, then so is  $Z^\ell$ . Furthermore, we can derive

$$\begin{aligned} \langle \mathcal{T}^\ell (\mathcal{T}^\ell)^\top, Z^\ell \rangle &= \langle \mathcal{T}^\ell (\mathcal{T}^\ell)^\top, \mathcal{D}^\ell \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top \mathcal{D}^\ell \rangle \\ &= \langle \mathcal{D}^\ell \mathcal{T}^\ell (\mathcal{T}^\ell)^\top \mathcal{D}^\ell, \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top \rangle \\ &= \langle \mathcal{D}^\ell, \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top \rangle = \sum_{l=1}^{n-\ell} \frac{1}{e_l^\ell} \sum_{j \in g(l)} \sum_{i \in g(l)} Z_{ij} \\ &\stackrel{(*)}{=} \sum_{l=1}^{n-\ell} \frac{1}{e_l^\ell} \sum_{j \in g(l)} \sum_{i \in g(l)} Z_{ii} = \sum_{l=1}^{n-\ell} \frac{1}{e_l^\ell} e_l^\ell \sum_{i \in g(l)} Z_{ii} \\ &= \sum_{l=1}^{n-\ell} \sum_{i \in g(l)} Z_{ii} = \sum_{i=1}^n Z_{ii} = k. \end{aligned}$$

Note that the equality  $(*)$  holds since  $Z_{i,j} = Z_{r,s}$  for any  $i, j, r, s \in g(l)$ . This ensures that constraint (10c) holds for  $Z^\ell$ .

To prove (10b) consider the equations

$$\begin{aligned} Z^\ell e^\ell &= \mathcal{D}^\ell \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top \mathcal{D}^\ell e^\ell = \mathcal{D}^\ell \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top e_{n-\ell} \\ &= \mathcal{D}^\ell \mathcal{T}^\ell Z e = \mathcal{D}^\ell \mathcal{T}^\ell e = \mathcal{D}^\ell e^\ell = e_{n-\ell}. \end{aligned}$$

It remains to show that the objective function values coincide.

$$\begin{aligned} \langle \mathcal{T}^\ell W (\mathcal{T}^\ell)^\top, Z^\ell \rangle &= \langle \mathcal{T}^\ell W (\mathcal{T}^\ell)^\top, \mathcal{D}^\ell \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top \mathcal{D}^\ell \rangle \\ &= \langle W, (\mathcal{T}^\ell)^\top \mathcal{D}^\ell \mathcal{T}^\ell Z (\mathcal{T}^\ell)^\top \mathcal{D}^\ell \mathcal{T}^\ell \rangle \\ &= \langle W, Z \rangle. \end{aligned}$$

As for the last equation, note that pre- and postmultiplying  $Z$  by  $(\mathcal{T}^\ell)^\top \mathcal{D}^\ell \mathcal{T}^\ell$  “averages” over the respective rows of matrix  $Z$ . Since these respective rows are identical due to (11d), the last equation holds.  $\square$

*Remark 3.3.* The addition of constraints  $Z_{ij} = 0$  for datapoints  $i, j$  that should not belong to the same cluster also goes through in the above equivalence. However, to keep the presentation simple we did not include it in the statement of the theorem above.  $\triangle$

*Remark 3.4.* It is straightforward to include the additional constraints (6), (7), and (8) in the subproblems, i.e., in case of shrinking the problem, the constraints are still valid. Again, to keep notation simple, we omitted these constraints in the presentation above. Further discussions on including these inequalities are in Section 4.1.  $\triangle$

### 3.3 Variable Selection for Branching

In a matrix  $Z$  corresponding to a clustering, for each pair  $(i, j)$  either  $Z_{ij} = 0$  or  $Z_{ii} = Z_{ij}$ . Peng and Xia [49] propose a simple branching scheme. Suppose that for the optimal solution of the SDP relaxation there are indices  $i$  and  $j$  such that  $Z_{ij}(Z_{ii} - Z_{ij}) \neq 0$  then one can produce a cannot-link branch with  $Z_{ij} = 0$  and a must-link branch with  $Z_{ii} = Z_{ij}$ . Regarding the variable selection the idea is to choose indices  $i$  and  $j$  such that in both branches we expect a significant improvement of the lower bound. In [49] the branching pair is chosen as the

$$\arg \max_{i,j} \{\min\{Z_{ij}, Z_{ii} - Z_{ij}\}\}.$$

Here we propose a variable selection strategy that is coherent with the way we generate the cannot-link and the must-link subproblems. In fact, we observe that in a matrix  $Z$  corresponding to a clustering, for each pair  $(i, j)$  either  $Z_{ij} = 0$  or  $Z_{ii} = Z_{jj}$ . This motivates the following strategy to select a pair of data points to branch on

$$\arg \max_{i,j} \{\min\{Z_{ij}, \|Z_{i\cdot} - Z_{j\cdot}\|_2^2\}\}.$$

In case this maximizer gives a value close to zero, say  $10^{-5}$ , the SDP solution corresponds to a feasible clustering.

#### Variable selection on the shrunk problem

The strategy for the variable selection still carries over on the shrunk problems. Since  $Z$  is obtained from  $Z^\ell$  only by repeating rows and columns, every pair  $(Z_{ij}^\ell, Z_{ii}^\ell)$  appears also in  $Z$  and vice versa. Moreover, within the already merged points, by construction  $Z_{ii}^\ell = Z_{ij}^\ell$  and hence this can never be a branching candidate again.

## 4 Branch-and-Bound Algorithm

We now put the bound computation (see Sections 2 and 3) together with our way of branching (see Section 3.3) to form our algorithm **SOS-SDP**. The final ingredient, a heuristic for providing upper bounds, is described in Section 5.

### 4.1 The Bound Computation

In order to obtain a strong lower bound, we solve the SDP relaxation (5) strengthened by the inequalities given in Section 2.

The enumeration of all pair and triangle inequalities is computationally intractable even for medium size instances. Therefore we use a similar separation routine for both types of inequalities:

1. Generate randomly up to  $t$  inequalities violated by at least  $\varepsilon_{\text{viol}}$
2. Sort the  $t$  inequalities by decreasing violation
3. Add to the current bounding problem the  $p \ll t$  most violated ones.

As for the clique inequalities, we use the heuristic separation routine described in [26] for the minimum  $k$ -partition problem, that returns at most  $n$  valid clique inequalities. More in detail, at each cutting-plane iteration, these cuts are determined by finding  $n$  subsets  $Q$  with a greedy principle. For each point  $i \in S = \{1, \dots, n\}$ ,  $Q$  is initialized as  $Q = \{i\}$ . Then, until the cardinality of  $Q$  does not reach the size  $k + 1$ ,  $Q$  is updated as  $Q = Q \cup \{\arg \min_{j \in S \setminus Q} \sum_{q \in Q} Z_{qj}\}$ .

We denote by  $\mathcal{A}(Z^\ell) = b$  the equations from the must-link and cannot-link constraints and by  $l \leq \mathcal{B}(Z^\ell) \leq u$  the inequalities representing the cutting planes. The cutting-plane procedure performed at each node is outlined in Algorithm 1.

We stop the procedure when we reach the maximum number of iterations  $cp_{\text{max}}$ . Another stopping criterion is based on the relative variation of the bound between two consecutive iterations. If the variation is lower than a tolerance  $\varepsilon_{\text{cp}}$ , the cutting-plane method terminates, and we branch.

At each node, we use a cuts inheritance procedure to quickly retrieve several effective inequalities from the parent node and save a significant number of cutting-plane iterations during the bound computation of the children. More in detail, the inequalities that were included in the parent node during the last cutting-plane iteration are passed to its children and included in their problem from the beginning. While inheriting inequalities in the  $(i, j)$  must link child, the shrinking procedure must be taken into account, updating the indices in the inequalities involved and deleting inequalities involving both points  $i$  and  $j$ .

In addition to the cuts inheritance, we use a cuts management procedure. A standard cutting-plane algorithm expects the valid inequalities not to be touched after having been included. The efficiency of state-of-the-art SDP solvers considerably deteriorates as we add these cuts, especially when solving large scale instances in terms of  $n$ . For this reason, after solving the current SDP, we remove the constraints that are not active at the optimum. Of course, inactive constraints may become active again in the subsequent cutting-plane iteration, and this operation could prevent the lower bounds from increasing monotonically; however, empirical results show that this situation happens rarely, and in this case, we decide to stop the cutting-plane procedure and we branch. From the practical standpoint, we notice that removing inactive constraints makes a huge difference since it keeps the SDP problem to a computationally tractable size. The result is that each cutting-plane iteration is more lightweight in comparison to the standard version, and this significantly impacts the overall efficiency of our branch-and-bound algorithm.

---

**Algorithm 1:** The node processing loop in the branch-and-cut algorithm

---

**Data:** A subproblem defined through the current set of equalities  $\mathcal{A}(Z^\ell) = b$ , and inequalities  $l \leq \mathcal{B}(Z^\ell) \leq u$ , the current global upper bound  $\varphi$ , the maximum number of cutting-plane iterations  $cp_{\max}$ , the cutting-plane tolerance  $\varepsilon_{\text{cp}}$ , the cuts violation tolerance  $\varepsilon_{\text{viol}}$ , and the cuts removal tolerance  $\varepsilon_{\text{act}}$ .

**Result:** A lower bound  $\hat{\delta}^\ell$  on the optimal value of the subproblem

**Initialization:**  $i \leftarrow 1$ ,  $\delta_0 \leftarrow -\infty$

**repeat**

    solve the current SDP relaxation:

$$\hat{\delta}_i^\ell = \min \{ \langle -\mathcal{T}^\ell W(\mathcal{T}^\ell)^\top, Z^\ell \rangle : \mathcal{A}(Z^\ell) = b, l \leq \mathcal{B}(Z^\ell) \leq u, Z^\ell \geq 0, Z^\ell \in \mathcal{S}_+^{n-\ell} \}$$

    and let  $\hat{Z}_i^\ell$  be the optimizer;

**if**  $\hat{\delta}_i^\ell \geq \varphi$  **then**

**stop;** the node can be pruned;

**if**  $i \geq cp_{\max}$  **or**  $\frac{|\hat{\delta}_i^\ell - \hat{\delta}_{i-1}^\ell|}{\hat{\delta}_{i-1}^\ell} \leq \varepsilon_{\text{cp}}$  **then**

**stop;** return the lower bound  $\hat{\delta}_i^\ell$  and branch;

    remove inactive inequalities with tolerance  $\varepsilon_{\text{act}}$  by updating  $(\cdot), l, u$ ;

    apply the separation routines for pair, triangle and clique inequalities with tolerance  $\varepsilon_{\text{viol}}$  and add them to  $(\mathcal{B}(\cdot), l, u)$ ;

**if no violated inequalities found then**

**stop;** return the lower bound  $\hat{\delta}_i^\ell$  and branch;

**else**

        add the inequalities by updating  $(\mathcal{B}(\cdot), l, u)$ ;

        set  $i \leftarrow i + 1$ ;

**until no violated inequalities found;**

---

## 4.2 Post-processing Using Error Bounds

Using the optimal solution of the SDP relaxation within a branch-and-bound framework requires the computation of “safe” bounds. Such safe bounds are obtained by solving the SDP to high precision, which, however, is out of reach when using first-order methods. In order to obtain a safe bound, we run a post-processing procedure where we use a method to obtain rigorous lower bounds on the optimal value of our SDP relaxation introduced by Jansson et al. [30]. Before describing our post-processing, we state a result bounding the eigenvalues of any feasible solution of (5).

**Lemma 4.1.** *Let  $Z \succeq 0$  and  $Z \geq 0$ . Furthermore, let  $Ze = e$ . Then the eigenvalues of  $Z$  are bounded by one.*

*Proof.* Let  $\lambda$  be an eigenvalue of  $Z$  with eigenvector  $v$ , i.e.,  $Zv = \lambda v$ . This implies

$$\lambda|v_i| = \left| \sum_{j=1}^n z_{ij}v_j \right| \leq \max_{1 \leq j \leq n} |v_j| \sum_{i=1}^n z_{ij} = \max_{1 \leq j \leq n} |v_j| \text{ for all } i \in \{1, \dots, n\}$$

by nonnegativity of  $Z$  and since the row sums of  $Z$  are one. Therefore, the inequality

$$\lambda \leq \frac{\max_{1 \leq j \leq n} |v_j|}{|v_i|}$$

holds for all  $i \in \{1, \dots, n\}$ , and in particular for  $i \in \arg \max_{1 \leq j \leq n} |v_j|$  which proves  $\lambda \leq 1$ .  $\square$

We now restate Lemma 3.1 from [30] in our context.

**Lemma 4.2.** *Let  $S, Z$  be symmetric matrices that satisfy  $0 \leq \lambda_{\min}(Z)$  and  $\lambda_{\max}(Z) \leq \bar{z}$  for some  $\bar{z} \in \mathbb{R}$ . Then the inequality*

$$\langle S, Z \rangle \geq \bar{z} \sum_{i: \lambda_i(S) < 0} \lambda_i(S)$$

*holds.*

*Proof.* Let  $S$  have the eigenvalue decomposition  $S = Q\Lambda Q^\top$  where  $QQ^\top = I$  and  $\Lambda = \text{Diag}(\lambda(S))$ . Then

$$\langle S, Z \rangle = \langle Q\Lambda Q^\top, Z \rangle = \langle \Lambda, Q^\top Z Q \rangle = \sum_{i=1}^n \lambda_i(S) Q_{\cdot, i}^\top Z Q_{\cdot, i}$$

where  $Q_{\cdot, i}$  is column  $i$  of matrix  $Q$ . Because of the bounds on the eigenvalues of  $Z$ , we have  $0 \leq Q_{\cdot, i}^\top Z Q_{\cdot, i} \leq \bar{z}$ . Therefore  $\langle S, Z \rangle \geq \bar{z} \sum_{i: \lambda_i(S) < 0} \lambda_i(S)$ .  $\square$

**Theorem 4.3.** *Consider the SDP (5) together with equations  $\mathcal{A}(Z) = b$  (e.g., from cannot-link constraints) and inequalities  $l \leq \mathcal{B}(Z) \leq u$  (representing cutting planes) with optimal objective function value  $p^*$ . Denote the dual variables by  $(\tilde{y}, \tilde{u}, \tilde{v}, \tilde{w}, \tilde{P})$ ,*

with  $\tilde{y} \in \mathbb{R}^{n+1}$ ,  $\tilde{u}$ ,  $\tilde{v}$ ,  $\tilde{w}$  being vectors of appropriate size,  $\tilde{P} \in \mathcal{S}^n$ ,  $\tilde{P} \geq 0$  and set  $\tilde{S} = -W - \sum_{i=1}^n \tilde{y}_i E_i - \tilde{y}_{n+1} I - \mathcal{A}^\top(\tilde{u}) + \mathcal{B}^\top(\tilde{v}) - \mathcal{B}^\top(\tilde{w}) - \tilde{P}$ . Then

$$p^* \geq \sum_{i=1}^n \tilde{y}_i + k\tilde{y}_{n+1} + b^\top \tilde{u} - l^\top \tilde{v} + u^\top \tilde{w} + \bar{z} \sum_{i: \lambda_i(\tilde{S}) < 0} \lambda_i(\tilde{S}).$$

*Proof.* Let  $Z^*$  be an optimal solution of (5) with the additional constraints  $\mathcal{A}(Z) = b$  and  $l \leq \mathcal{B}(Z) \leq u$  and  $(\tilde{y}, \tilde{z}, \tilde{u}, \tilde{v}, \tilde{w}, \tilde{P})$  dual feasible. Then

$$\begin{aligned} \langle -W, Z^* \rangle &= \left( \sum_{i=1}^n \tilde{y}_i + k\tilde{y}_{n+1} + 1 + b^\top \tilde{u} - l^\top \tilde{v} + u^\top \tilde{w} \right) \\ &= \langle -W, Z^* \rangle - \sum_{i=1}^n \tilde{y}_i \langle E_i, Z^* \rangle - \tilde{z} \langle I, Z^* \rangle - \langle \mathcal{A}(Z^*), \tilde{u} \rangle + \langle \mathcal{B}(Z^*), \tilde{v} \rangle - \langle \mathcal{B}(Z^*), \tilde{w} \rangle \\ &= \langle -W - \sum_{i=1}^n \tilde{y}_i E_i - \tilde{y}_{n+1} I - \mathcal{A}^\top(\tilde{u}) + \mathcal{B}^\top(\tilde{v}) - \mathcal{B}^\top(\tilde{w}), Z^* \rangle \\ &= \langle \tilde{P} + \tilde{S}, Z^* \rangle = \langle \tilde{P}, Z^* \rangle + \langle \tilde{S}, Z^* \rangle. \end{aligned}$$

We have  $\tilde{P} \geq 0$ ,  $Z^* \geq 0$ . Furthermore, the eigenvalues of  $Z^*$  are nonnegative and bounded by one (Lemma 4.1). Using this and Lemma 4.2, we obtain

$$\begin{aligned} p^* = \langle -W, Z^* \rangle &\geq \sum_{i=1}^n \tilde{y}_i + k\tilde{y}_{n+1} + b^\top \tilde{u} - l^\top \tilde{v} + u^\top \tilde{w} + \langle \tilde{S}, Z^* \rangle \\ &\geq \sum_{i=1}^n \tilde{y}_i + k\tilde{y}_{n+1} + b^\top \tilde{u} - l^\top \tilde{v} + u^\top \tilde{w} + \sum_{i: \lambda_i(\tilde{S}) < 0} \lambda_i(\tilde{S}). \end{aligned}$$

□

Before stating the result used in the branch-and-bound tree after merging data points, we introduce the following notation. Let  $E_i^\ell$  be the symmetric matrix such that  $\langle E_i^\ell, Z^\ell \rangle = (Z^\ell e^\ell)_i$ .

**Corollary 4.4.** *Consider the SDP (10) together with equations  $\mathcal{A}(Z^\ell) = b$  (e.g., from cannot-link constraints) and inequalities  $l \leq \mathcal{B}(Z^\ell) \leq u$  (representing cutting planes) with optimal objective function value  $p^*$ . Let  $\tilde{y} \in \mathbb{R}^{n-\ell+1}$ ,  $\tilde{u}$ ,  $\tilde{v}$ ,  $\tilde{w}$  being vectors of appropriate size,  $\tilde{P} \in \mathcal{S}^{n-\ell}$ ,  $\tilde{P} \geq 0$  and set  $\tilde{S} = -W^\ell - \sum_{i=1}^{n-\ell} \tilde{y}_i E_i^\ell + \tilde{y}_{n-\ell+1} \text{Diag}(e^\ell) - \mathcal{A}^\top(\tilde{u}) + \mathcal{B}^\top(\tilde{v}) - \mathcal{B}^\top(\tilde{w}) - \tilde{P}$ . Then*

$$p^* \geq \sum_{i=1}^{n-\ell} \tilde{y}_i + k\tilde{y}_{n-\ell+1} + b^\top \tilde{u} - l^\top \tilde{v} + u^\top \tilde{w} + \sum_{i: \lambda_i(\tilde{S}) < 0} \lambda_i(\tilde{S}).$$

*Proof.* Constraint (10b) implies that the row-sum of any row in  $Z^\ell$  is bounded by one since

$$\sum_{j=1}^{n-\ell} z_{ij}^\ell \leq \sum_{j=1}^{n-\ell} z_{ij}^\ell e_j^\ell = 1 \quad \text{for all } i \in \{1, \dots, n-\ell\}.$$



Hence using the same arguments as in Lemma 4.1 we can bound the eigenvalues by one and apply Theorem 4.3.  $\square$

## 5 Heuristic

The most popular heuristic for solving MSSC is  $k$ -means [41, 40]. It can be viewed as a greedy algorithm. During each update step, all the data points are assigned to their nearest centers. Afterwards, the cluster centers are repositioned by calculating the mean of the assigned observations to the respective centroids. The update process is performed until the centroids are no longer updated and therefore all observations remain at the assigned clusters. In this paper, we use COP  $k$ -means [59], a constrained version of  $k$ -means that aims at finding high quality clusters using prior knowledge. COP  $k$ -means is a constrained clustering algorithm that belongs to a class of semi-supervised machine learning algorithms. Constrained clustering incorporates a set of must-link and cannot-link constraints that define a relationship between two data instances: a must-link constraint (ML) is used to specify that the two points in the must-link relation should be in the same cluster, whereas a cannot-link constraint (CL) is used to specify that the two points in the cannot-link relation should not be in the same cluster. These sets of constraints, which are naturally available as branching decisions while visiting the branch-and-bound tree, represent the prior knowledge on the problem for which  $k$ -means will attempt to find clusters that satisfy the specified ML and CL constraints. The algorithm returns an empty partition if no such clustering exists which satisfies the constraints. COP  $k$ -means is described in Algorithm 2.

Like other local solvers for non-convex optimization problems,  $k$ -means (both in the unconstrained and constrained version) is very sensitive to the choice of the initial centroids, therefore, it often converges to a local minimum rather than the global minimum of the MSSC objective. To overcome this drawback, the algorithm is initialized with several different starting points, choosing then the clustering with the lowest objective function [25].

In the literature, several initialization algorithms have been proposed to prevent  $k$ -means to get stuck in a low quality local minimum. The most popular strategy for initializing  $k$ -means is  $k$ -means++ [6]. The basic idea behind this approach is to spread out the  $k$  initial cluster centers to avoid the poor clustering that can be found by the standard  $k$ -means algorithm with random initialization. More in detail, in  $k$ -means++, the first cluster center is randomly chosen from the data points. Then, each subsequent cluster center is chosen from the remaining data points with probability proportional to its squared distance from the already chosen cluster centers.

We aim to exploit the information available in the solution of the SDP relaxation in order to extract a centroid initialization for COP  $k$ -means.

In the literature, theoretical properties of the Peng-Wei relaxation have been studied under specific stochastic models. A feasible clustering can be derived by the solution of the SDP relaxation (5) by a rounding step. Sometimes, the rounding step is unnecessary because the SDP relaxation finds a solution that is feasible for the original MSSC. This

---

**Algorithm 2:** COP  $k$ -means

---

K-MEANS(*dataset*  $\mathcal{D}$ , *initial cluster centers*  $m_1, \dots, m_k$ , *must-link constraints*  $ML \subseteq \mathcal{D} \times \mathcal{D}$ , *cannot-link constraints*  $CL \subseteq \mathcal{D} \times \mathcal{D}$ )

**repeat**

**foreach** *data point*  $s_i \in \mathcal{D}$  **do**

$j \leftarrow \arg \min \{ \|s_i - m_j\|^2 : j \in \{1, \dots, k\} \text{ \& } \text{VIOLATE\_CONSTRAINT}(s_i, C_j, ML, CL) \text{ is false} \};$

**if**  $j < \infty$  **then**

            assign  $s_i$  to  $C_j$ ;

**else**

**return** empty partition;

**foreach** *cluster*  $C_j$  **do**

$m_j \leftarrow$  mean of the data points  $s_i$  assigned to  $C_j$ ;

**until** *convergence*;

**return**  $C_1, \dots, C_k$

VIOLATE\_CONSTRAINTS(*data point*  $s_i$ , *cluster*  $C_j$ , *must-link constraints*  $ML \subseteq \mathcal{D} \times \mathcal{D}$ , *cannot-link constraints*  $CL \subseteq \mathcal{D} \times \mathcal{D}$ )

**foreach**  $(s_i, s_h) \in ML$  **do**

**if**  $s_h \notin C_j$  **then** **return** true;

**foreach**  $(s_i, s_h) \in CL$  **do**

**if**  $s_h \in C_j$  **then** **return** true;

**return** false;

---

phenomenon is known in the literature as exact recovery or tightness of the relaxation. Recovery guarantees have been established under a model called the subgaussian mixtures model, whose special cases include the stochastic ball model and Gaussian mixture model [7, 29, 46, 38]. Under this distributional setting, cluster recovery is guaranteed with high probability whenever the distances between the clusters are sufficiently large. However, the generative assumption may not be satisfied by real data, and this implies that in general a rounding procedure is needed, and if possible also a bound improvement. Instead of building a rounding procedure, we decide to derive a “smart” initialization for the constrained  $k$ -means based on the solution of our bounding problem. Here, we build the initialization exploiting the matrix  $Z_{SDP}$  solution of the current bounding problem. The idea is that if the relaxation were tight, then  $Z_{SDP}$  would be a clustering feasible for the rank constrained SDP (4), and hence would allow to easily recover the centroids. If the relaxation is not tight, the closest rank- $k$  approximation is built and it is used to recover the centroids. More in detail, let  $Z$  be a feasible solution of the rank constrained SDP (4). It is straightforward [46] to see that  $Z$  can be written as the sum of  $k$  rank-one matrices:

$$Z = \sum_{j=1}^k \frac{1}{|C_j|} \mathbb{1}_{C_j} \mathbb{1}_{C_j}^\top, \quad (12)$$

where  $\mathbb{1}_{C_j} \in \{0, 1\}^n$  is the indicator vector of the  $j$ -th cluster, i.e., the  $i$ -th component of  $\mathbb{1}_{C_j}$  is 1 if the data point  $p_i \in C_j$  and 0 otherwise. If we post-multiply  $Z$  by the data matrix  $W_p \in \mathbf{R}^{n \times d}$  whose  $i$ -th row is the data point  $p_i$ , we obtain a matrix  $M = ZW_p$  with a well defined structure. In fact, from equation (12) it follows that, for each  $j \in \{1, \dots, k\}$ ,  $M$  contains  $|C_j|$  rows equal to the centroid of the data points assigned to  $C_j$ . If the SDP relaxation is tight, the different rows of  $M$  are equal to the optimal centroids. In this case, it is natural to use the convex relaxation directly to obtain the underlying ground truth solution without the need for a rounding step. In practice, the optimizer of the SDP relaxation may not always be a clustering matrix, i.e., a low-rank solution as described by equation (12).

The idea now is to build the rank- $k$  approximation  $\hat{Z}$  which is obtained exploiting the following result.

**Proposition 5.1.** [24] *Let  $X$  be a positive semidefinite matrix with the eigenvalues  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n \geq 0$  and the corresponding eigenvectors  $v_1, v_2, \dots, v_n$ . If  $X$  has rank  $r$ , for any  $k < r$ , the best rank  $k$  approximation of  $X$ , for both the Frobenius and the spectral norms is given by*

$$\hat{X} = \sum_{i=1}^k \lambda_i v_i v_i^\top, \quad (13)$$

*which is the truncated eigenvalue decomposition of  $X$ .*

Then, we compute the approximate centroid matrix  $M = \hat{Z}W_p$ . In order to derive the  $k$  centroids, the unconstrained  $k$ -means is applied to the rows of matrix  $M$ . Finally, the obtained centroids are used in order to initialize the algorithm COP  $k$ -means, which is run just once. The procedure is summarized in Algorithm 3.

---

**Algorithm 3:** SDP-based initialization of  $k$ -means

---

SDP-INIT(*dataset*  $\mathcal{D}$ , *number of clusters*  $k$ , *must-link constraints*  $ML \subseteq \mathcal{D} \times \mathcal{D}$ ,  
*cannot-link constraints*  $CL \subseteq \mathcal{D} \times \mathcal{D}$ )  
solve the SDP relaxation and obtain the optimizer  $Z_{SDP}$ ;  
find the best rank  $k$  approximation of  $Z_{SDP}$  and obtain  $\hat{Z}$  by (13);  
compute  $M = \hat{Z}W_p$ ;  
cluster the rows of  $M$  with unconstrained  $k$ -means to get the centroids  
 $m_1, \dots, m_k$ ;  
use  $m_1, \dots, m_k$  as the starting point of constrained  $k$ -means;

---

The intuition is that the better the SDP solution, the better the initialization, and hence the produced clustering. In order to confirm this intuition, we show the behavior of the heuristic on a synthetic example with 150 points in 2 dimensions. We denote by circles the points in  $W_p$ , by crosses the rows of matrix  $M$  produced at Step 3, by diamonds the centroids obtained by clustering the rows of  $M$  at Step 4 of the Algorithm 3. In Figure 1 we assume  $k = 3$  and apply our heuristic on different solutions of the SDPs generated during our bounding procedure: in Figure 1 (a) we use as  $Z_{SDP}$  the solution obtained by solving problem (5), and we can see that there is some gap (the upper and lower bounds are displayed on top of each figure) and that matrix  $M$  has many different rows. In Figures 1 (b), (c), and (d) we consider as  $Z_{SDP}$  the solution of the SDP obtained by performing respectively 1, 2 and 3 iterations of adding cutting-planes, i.e., solving problem (5) with some additional constraints (6)–(8). It is clear how the rows of  $M$  converge to three different centroids that, in this case, correspond to the optimal solution (the gap here is zero).

The use of SDP-INIT as a standalone initialization procedure could be expensive since it needs to solve a certain number of SDP problems and to perform an eigenvalue decomposition on the solution that gives the best lower bound. However, when embedded in our branch-and-bound, the extra cost of running SDP-INIT is only the computation of the spectral decomposition of the SDP solution providing the lower bound at the node, which is negligible with respect to the bound computation.

The effectiveness of the proposed heuristic algorithm is confirmed by the numerical results presented in Section 6.5.

## 6 Numerical Results

In this section we describe the implementation details and we show the numerical results of SOS-SDP on synthetic and real-world datasets.

### 6.1 Details on the Implementation

SOS-SDP is implemented in C++ and we use as internal subroutine for computing the bound SDPNAL+ [61, 64], which is implemented in MATLAB. SDPNAL+ is called using

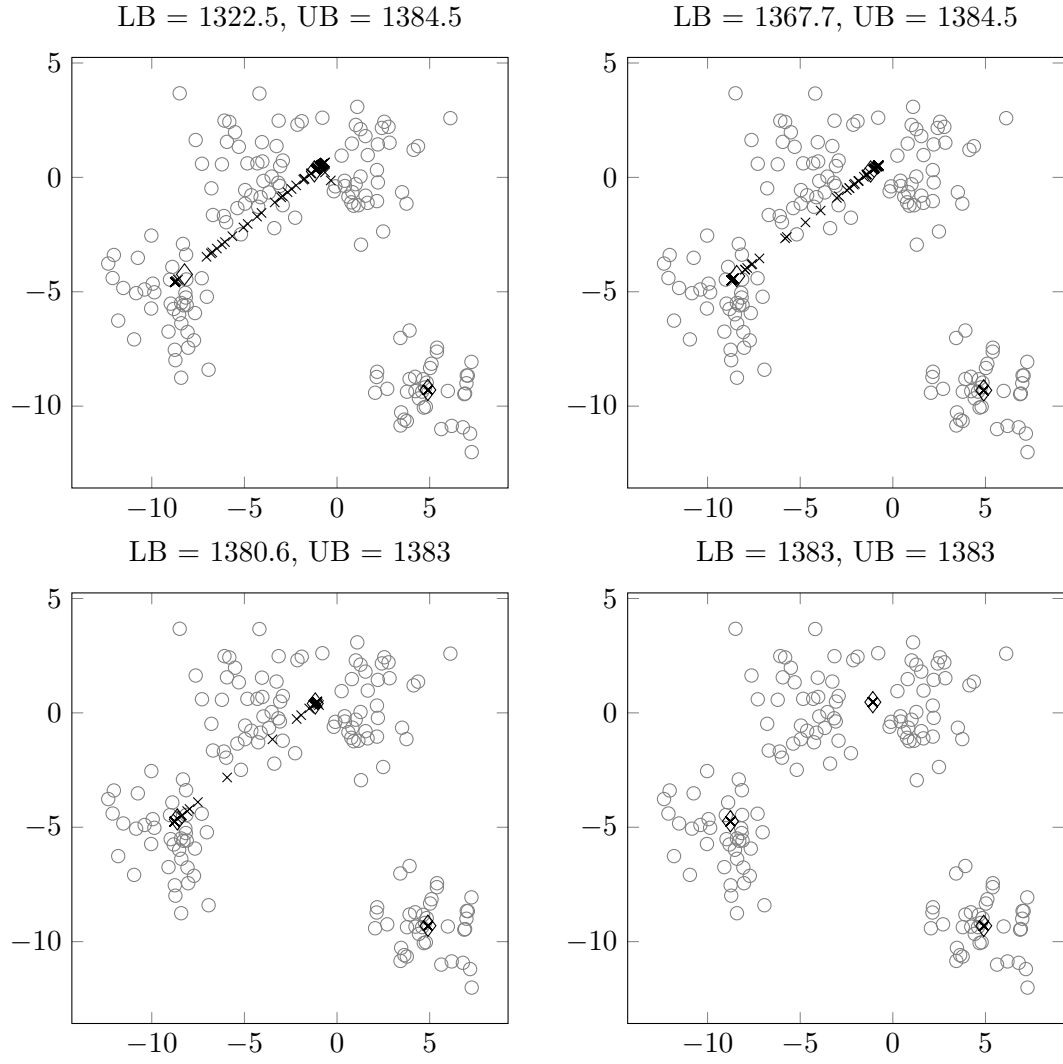


Figure 1: An instance with 150 points and  $k = 3$ .

the MATLAB Engine API that enables running MATLAB code from C++ programs. We run our experiments on a machine with Intel(R) Xeon(R) 8124M CPU @ 3.00GHz with 16 cores, 64 GB of RAM, and Ubuntu Server 20.04. The C++ Armadillo library [53] is extensively used to handle matrices and linear algebra operations efficiently. **SOS-SDP** can be efficiently executed in a multi-thread environment. In order to guarantee an easy and highly configurable parallelization, we use the thread pool pattern. This pattern allows controlling the number of threads the branch-and-bound is creating and saving resources by reusing threads for processing different nodes of the tree. We adopt the same branch-and-bound configuration for each instance. In particular, we visit the tree with the best-first search strategy. When the problem at a given level is divided into the *must-link* and the *cannot-link* sub-problems, each node is submitted to the thread pool and run in parallel with the other threads of the pool. Each thread of the branch-and-bound algorithm runs in a separate MATLAB session. Furthermore, since numerical algebra and linear functions are multi-threaded in MATLAB, these functions automatically execute on multiple computational threads in a single MATLAB session. To balance resource allocations for multiple MATLAB sessions and use all the available cores of the machine, we set a maximum number of computational threads allowed in each session.

**Branch-and-bound setting** On all the numerical tests, we adopt the following parameters setting. As for the pair and triangle inequalities, we randomly separate at most 100000 valid cuts, we sort them in decreasing order with respect to the violation, and we select the first 5% of violated ones, yielding at most 5000 pairs and at most 5000 triangles added in each cutting-plane iteration. Since effective inequalities are inherited from the parent to its children, at the root node the maximum number of cutting-plane iterations is set to  $cp_{\max} = 50$ , whereas for the children this number is set to 30. The tolerance for checking the violation of the cuts is set to  $\varepsilon_{\text{viol}} = 10^{-4}$ , whereas the tolerance for identifying the active inequalities is set to  $\varepsilon_{\text{act}} = 10^{-6}$ . Finally, we set the accuracy tolerance of SDPNAL+ to  $10^{-5}$ .

As for the parallel setting, we use different configurations depending on the size of the instances since the solver requires a higher number of threads to efficiently solve large size problems. For small instances ( $n < 500$ ) we create a pool of 16 threads, each of them running on a session with a single component thread. For medium instances ( $500 \leq n < 1000$ ) we use a pool of 8 threads, each of them running on a session with 2 component threads. For ( $1000 \leq n < 1500$ ) we use a pool of 4 threads, each of them runs on a session with 4 component threads. Finally, for large scale instances ( $n \geq 1500$ ) we use a pool of 2 threads, each of them running on a session with 8 component threads. In all cases, the MATLAB session for the computation at the root node uses all the available cores. The source code and the pre-processed datasets are available at <https://github.com/antoniosudoso/sos-sdp>.

## 6.2 Benchmark Instances

In order to test extensively the efficiency of **SOS-SDP** we use both artificial datasets that are built in such a way to be compliant with the MSSC assumptions and real-world datasets.

**Artificial Instances** Due to the minimization of the sum of squared Euclidean distances, an algorithm that solves the MSSC finds spherically distributed clusters around the centers. In order to show the effectiveness of our algorithm on instances compliant with the MSSC assumptions, we generate very large scale Gaussian datasets in the plane ( $d = 2$ ) with varying number of data points  $n \in \{2000, 2500, 3000\}$ , number of clusters  $k \in \{10, 15\}$  and degree of overlap. More in detail, we sample  $n$  points from a mixture of  $k$  Gaussian distributions  $\mathcal{N}(\mu_j, \Sigma_j)$  with equal mixing proportions, mean  $\mu_j$  and shared spherical covariance matrix  $\Sigma_j = \sigma I$ , where  $\sigma \in \{0.5, 1.0\}$  is the standard deviation. The cluster centers  $\mu_j$  are sampled from a uniform distribution in the interval  $[-\frac{n}{1000} - k, \frac{n}{1000} + k]$ . We use the following notation to name the instances:  $\{n\}_{-}\{k\}_{-}\{\sigma\}$ . Note that in this case, we know in advance the correct number of clusters, so we only solve the instances for that value of  $k$ .

**Real-world Datasets** We use a set of 34 real-world datasets coming from different domains, with a number of entities  $n$  ranging between 75 and 4177, and with a number of features  $d$  ranging between 2 and 20531. The datasets' characteristics are reported in Table 1.

## 6.3 Branch-and-Bound Results on Artificial instances

In Table 2 we report the dataset name according to the notation  $\{n\}_{-}\{k\}_{-}\{\sigma\}$ , the optimal objective function  $f_{\text{opt}}$ , the number of cutting-plane iterations at the root (cp), the number of cuts added in the last cutting-plane iteration at the root ( $\text{cuts}_{\text{cp}}$ ), the gap at the root ( $\text{gap}_0$ ) when problem (9) is solved without adding valid inequalities, in brackets the gap at the end of the cutting-plane procedure at the root node ( $\text{gap}_{\text{cp}}$ ), the number of nodes of the branch-and-bound tree (N), and the wall clock time in seconds (time).

Dataset	$f_{\text{opt}}$	cp	$\text{cuts}_{\text{cp}}$	$\text{gap}_0$ ( $\text{gap}_{\text{cp}}$ )	N	time
2000_10_0.5	955.668	0	0	0.000039 (0.000039)	1	848.88
2000_10_1.0	3601.310	3	10999	0.006171 (0.003578)	3	8794.17
2000_15_0.5	955.800	1	6177	0.001556 (0.000009)	1	1155.06
2000_15_1.0	3658.730	3	11035	0.006192 (0.002059)	3	8351.91
2500_10_0.5	1199.080	1	5249	0.000184 (0.000083)	1	2859.30
2500_10_1.0	4522.350	12	11539	0.008008 (0.000553)	1	20495.43
2500_15_0.5	1194.550	0	0	0.000699 (0.000699)	1	1049.76
2500_15_1.0	4574.360	6	10146	0.005311 (0.000971)	1	10245.69
3000_10_0.5	1446.480	0	0	0.000067 (0.000067)	1	2220.21

3000_10_1.0	5512.370	9	10769	0.004601 (0.000606)	1	27781.38
3000_15_0.5	1439.940	0	0	0.000433 (0.000433)	1	2003.94
3000_15_1.0	5537.200	10	15608	0.006245 (0.001205)	3	38330.01

Table 2: Results for the artificial datasets.

As we increase  $\sigma$ , the cluster separation decreases, and the degree of overlap increases (see Figure 2). In this scenario, the SDP relaxation is not tight anymore and the global minimum is certified by our specialized branch-and-bound algorithm. For  $\sigma = 0.5$  each problem is solved at the root with zero (i.e., the SDP relaxation is tight) or with at most one cutting-plane iteration. As we decrease the cluster separation by increasing  $\sigma$  the problem becomes harder since some clusters overlap and the cluster boundaries are less clear. In this case, more cutting-plane iterations are needed (up to a maximum of 12 iterations). In any case, we need at most 3 nodes for solving these instances, and this confirms that, if the generative assumption is met, the cutting-plane procedure at the root node is the main ingredient for success. In the next section, we show how the behavior changes in real world instances, where we do not have information on the data distribution and on the correct value of  $k$ . In this case, the overall branch-and-bound algorithm becomes fundamental in order to solve the problems.

#### 6.4 Branch-and-Bound Results on Real World Datasets

The MSSC requires the user to specify the number of clusters  $k$  to generate. Determining the right  $k$  for a data set is a different issue from the process of solving the clustering problem. This is still an open problem since, depending on the chosen distance measure, one value of  $k$  may be better than another one. Hence, choosing  $k$  is often based on assumptions on the application, prior knowledge of the properties of the dataset, and practical experience. In the literature, clustering validity indices in conjunction with the  $k$ -means algorithm are commonly used to determine the “right” number of clusters. Most of these methods minimize or maximize an external validity index by running a clustering algorithm (for example  $k$ -means) several times for different values of  $k$ . We recall that the basic idea behind the MSSC is to define clusters such that the total within-cluster sum of squares is minimized. This objective function measures the compactness of the clustering and we want it to be as small as possible. The “elbow method” is probably the most popular method for determining the number of clusters. It requires running the  $k$ -means algorithm with an increasing number of clusters. The suggested  $k$  can be determined by looking at the MSSC objective as a function of  $k$  and by finding the inflection point. The location of the inflection point (knee) in the plot is generally considered as an indicator of the appropriate number of clusters. The drawback of this method is that the identification of the knee could not be obvious. Hence, different validity indices have been proposed in the literature to identify the suitable number of clusters or to check whether a given dataset exhibits some kind of a structure that can be captured by a clustering algorithm for a given  $k$ . All these indices are computed a posteriori given the clustering produced for different values of



Dataset	$n$	$d$	Dataset	$n$	$d$
Ruspini	75	2	Strawberry	613	235
Voice	126	310	Energy	768	16
Iris	150	4	Gene	801	20531
Wine	178	13	SalesWeekly	810	106
Gr202	202	2	Vehicle	846	18
Seeds	210	7	Arcene	900	10000
Glass	214	9	Wafer	1000	152
CatsDogs	328	14773	Power	1096	24
Accent	329	12	Phishing	1353	9
Ecoli	336	7	Aspirin	1500	63
RealEstate	414	5	Car	1727	11
Wholesale	440	11	Wifi	2000	7
ECG5000	500	140	Ethanol	2000	27
Hungarian	522	20	Mallat	2400	1024
Wdbc	569	30	Advertising	3279	1558
Control	600	60	Rice	3810	7
Heartbeat	606	3053	Abalone	4177	10

Table 1: Characteristics of the real world datasets. They all can be downloaded at the UCI [22], UCR [14] and sGDML [12] websites.

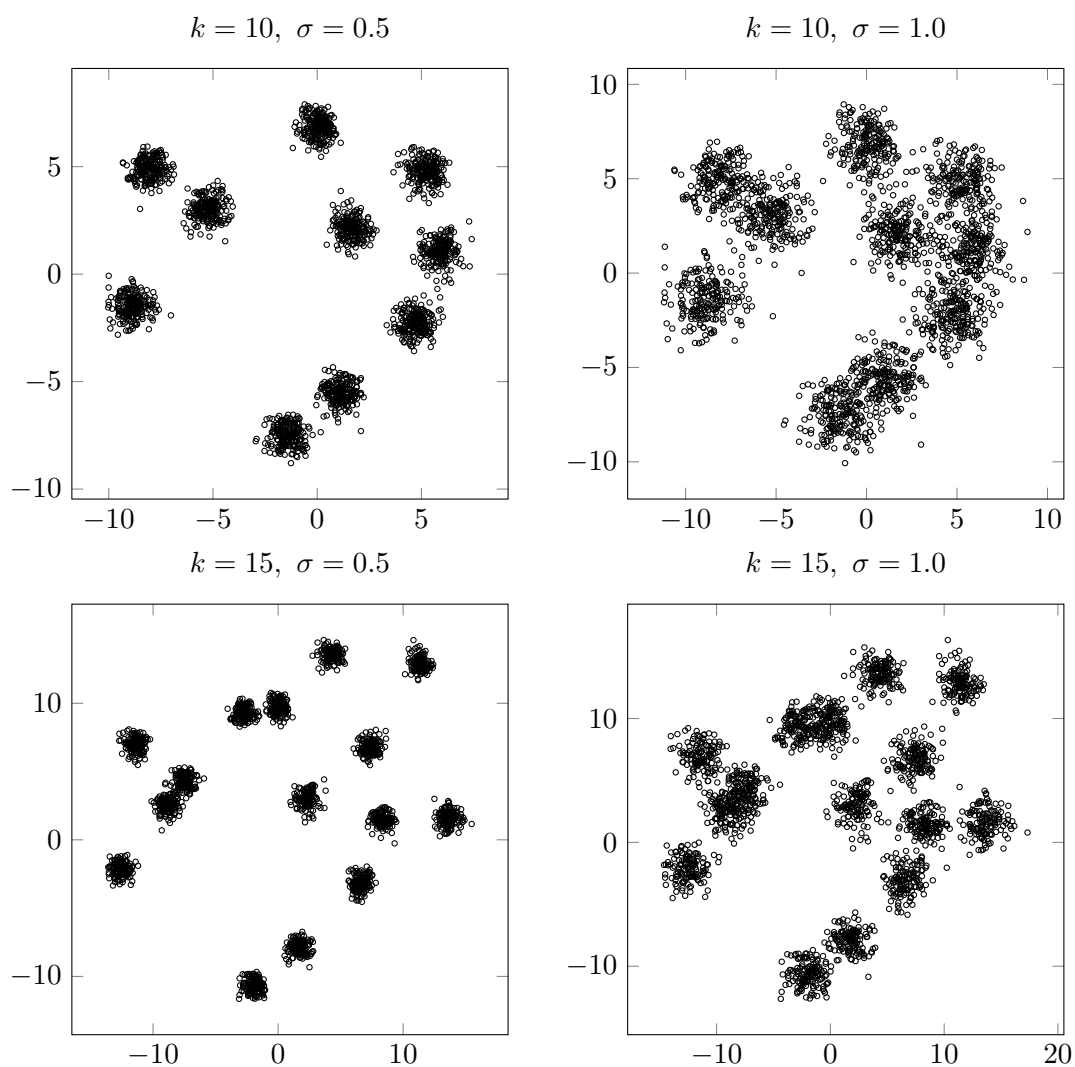


Figure 2: Artificial instances for  $n = 2000$  and  $d = 2$ .

$k$ . In addition to the elbow method, we use three cluster validity measures that are compliant with the assumptions of the MSSC: namely the Silhouette index [52], the Calinski–Harabasz (CH) index [10] and Davies–Bouldin (DB) index [16]. The Silhouette index determines how well each object lies within its cluster and is given by the average Silhouette coefficient over all the data points. The Silhouette coefficient is defined for each data point and is composed of two scores: the mean distance between a sample and all other points in the same class and the mean distance between a sample and all other points in the next nearest cluster. The CH index is the ratio of the sum of between-clusters dispersion and within-cluster dispersion for all clusters. The DB index is defined as the average similarity between each cluster and its most similar one.

The Silhouette index and the CH index are higher when the clusters are dense and well separated, which relates to the standard concept of clustering, whereas for the DB index lower values indicate a better partition.

Since the exact resolution of the MSCC problem could be expensive and time consuming from the computational point of view, one may be interested in finding the global solution for a specified or restricted number of clusters. In practice, one can run the  $k$ -means algorithm for different values of  $k$  and then use the exact algorithm to find and certify the global optimum for the  $k$  suitable for the application of interest. Hence, we choose to run our algorithm on a large number of datasets, and for each dataset, we run it only for the suggested number of clusters obtained with the help of the criteria mentioned above. Whenever there is some ambiguity, i.e., the different criteria suggest different values of  $k$ , we run our algorithm for all the suggested values. With this criterion, we end up solving 54 clustering instances with different size  $n$ , different dimension  $d$ , and different values of  $k$ .

In Table 3 we report:

- the dataset name
- the number of clusters ( $k$ )
- the optimal objective function ( $f_{\text{opt}}$ ). We add a (\*) whenever the optimum we certify is not found by  $k$ -means at the root node
- the number of cutting-plane iterations at root (cp)
- the number of inequalities of the last SDP problem solved at the root in the cutting-plane procedure ( $\text{cuts}_{\text{cp}}$ )
- the gap at the root ( $\text{gap}_0$ ) when problem (9) is solved without adding valid inequalities, and in brackets the gap at the end of the cutting-plane procedure at the root node ( $\text{gap}_{\text{cp}}$ )
- the number of nodes (N) of the branch-and-bound tree
- the wall clock time in seconds (time).

Small and medium scale instances ( $n < 1000$ ) are considered solved when the relative gap tolerance is less or equal than  $10^{-4}$ , whereas for large scale instances ( $n \geq 1000$ ) the branch-and-bound algorithm is stopped when the tolerance is less or equal than  $10^{-3}$ , which we feel is an adequate tolerance for large scale real-world applications. The gap measures the difference between the best upper and lower bounds and it is calculated as  $(UB - LB)/UB$ .

The numerical results show that our method is able to solve successfully all the instances up to a size of  $n = 4177$  entities.

Dataset	$k$	$f_{\text{opt}}$	cp	cuts <sub>cp</sub>	$gap_0$ ( $gap_{cp}$ )	N	time
Ruspini	4	1.28811e+04	0	0	2.23e-04 (2.23e-04)	1	2.55
Voice	2	1.13277e+22	2	7593	5.40e-02 (1.66e-06)	1	14.45
Voice	9	5.74324e+20*	4	6115	1.07e-01 (6.45e-04)	3	128.35
Iris	2	1.52348e+02	2	7701	1.10e-02 (2.19e-06)	1	17
Iris	3	7.88514e+01	4	7136	4.23e-02 (1.18e-04)	5	83.3
Iris	4	5.72285e+01	4	7262	4.28e-02 (4.20e-04)	3	104.55
Wine	2	4.54375e+06	3	8162	3.45e-02 (2.69e-07)	1	53.55
Wine	7	4.12138e+05*	4	5759	5.81e-02 (1.03e-04)	3	87.55
Gr202	6	6.76488e+03	6	6607	6.72e-02 (8.53e-04)	17	298.35
Seeds	2	1.01161e+03*	9	10186	4.31e-02 (4.77e-04)	29	957.1
Seeds	3	5.87319e+02	4	6620	2.67e-02 (1.26e-05)	1	68.85
Glass	3	1.14341e+02	5	6799	4.68e-02 (1.64e-04)	3	193.8
Glass	6	7.29647e+01*	7	3014	5.45e-02 (4.36e-04)	5	198.9
CatsDogs	2	1.14099e+05	1	5368	1.83e-03 (2.23e-09)	1	108.8
Accent	2	3.28685e+04	0	0	6.55e-06 (6.55e-06)	1	11.05
Accent	6	1.84360e+04*	8	4523	2.94e-02 (2.08e-05)	1	244.8
Ecoli	3	2.32610e+01	4	10101	7.71e-03 (1.89e-04)	3	181.9
RealEstate	3	5.50785e+07	3	6236	1.59e-02 (3.51e-05)	1	104.55
RealEstate	5	2.18711e+07	5	8006	6.82e-02 (2.64e-05)	1	258.4
Wholesale	5	2.04735e+03	6	7668	6.43e-02 (2.06e-05)	1	421.6
Wholesale	6	1.73496e+03*	10	11161	6.32e-02 (7.06e-04)	3	1782.45
ECG5000	2	1.61359e+04	3	9312	1.02e-03 (7.49e-05)	1	119
ECG5000	5	1.15458e+04	25	6289	4.93e-02 (1.01e-04)	3	2524.5
Hungarian	2	8.80283e+06	7	11265	1.03e-02 (1.31e-05)	1	551.65
Wdbc	2	7.79431e+07	5	8645	3.21e-02 (2.10e-05)	1	436.05
Wdbc	5	2.05352e+07*	23	10662	7.45e-02 (5.27e-04)	15	2436.95
Control	3	1.23438e+06	6	12381	2.80e-03 (1.26e-04)	9	895.9
Heartbeat	2	2.79391e+04	0	0	8.15e-06 (8.15e-06)	1	66.3
Strawberry	2	2.79363e+03	15	23776	5.44e-02 (4.02e-04)	37	5250.45
Energy	2	9.64123e+03	0	0	9.86e-09 (9.86e-09)	1	18.7
Energy	12	4.87456e+03	0	0	4.03e-07 (4.03e-07)	1	29.75
Gene	5	1.78019e+07*	2	15589	1.83e-03 (1.30e-04)	3	3851.35
Gene	6	1.70738e+07	5	14620	3.82e-03 (2.08e-04)	11	9896.55

SalesWeekly	2	1.44942e+06*	6	8508	2.50e-02 (1.33e-03)	9	2341.75
SalesWeekly	3	7.09183e+05*	4	9096	1.03e-03 (9.44e-05)	1	262.65
SalesWeekly	5	5.20938e+05*	4	11811	1.67e-03 (1.12e-04)	5	1045.5
Vehicle	2	7.29088e+06	5	10395	7.68e-03 (3.72e-04)	11	1842.8
Arcene	2	3.48490e+10	3	36100	2.59e-03 (1.26e-04)	3	1369.35
Arcene	3	2.02369e+10	0	0	3.50e-06 (3.50e-06)	1	758.2
Arcene	5	1.69096e+10*	7	8327	7.57e-03 (1.55e-04)	27	6885
Wafer	2	6.19539e+04	3	7254	7.82e-04 (1.00e-04)	1	379.1
Wafer	4	4.42751e+04	22	16957	1.97e-02 (8.76e-04)	1	6756.65
Power	2	3.22063e+03	3	11350	1.05e-02 (2.89e-03)	3	3381.3
Phishing	9	3.15888e+03*	46	12459	2.48e-02 (7.00e-04)	1	18866.6
Aspirin	3	1.27669e+04	2	10000	4.39e-03 (3.02e-03)	9	3779.1
Car	4	5.61600e+03	23	38582	1.61e-03 (1.02e-05)	1	5989.95
Ethanol	2	7.26854e+03	0	0	5.33e-08 (5.33e-08)	1	310.25
Wifi	5	2.04311e+05	7	20886	1.13e-02 (2.18e-03)	7	22754.5
Mallat	3	9.08648e+04	5	17092	3.61e-03 (9.59e-04)	1	5970.4
Mallat	4	7.45227e+04	6	15305	6.80e-03 (4.49e-03)	5	26344.9
Advertising	2	5.00383e+06*	1	12533	1.53e-03 (2.16e-05)	1	6465.1
Advertising	8	4.54497e+06*	4	19948	2.98e-03 (1.08e-04)	1	25114.1
Rice	2	1.39251e+04	24	7258	1.43e-02 (7.14e-03)	5	103710.2
Abalone	3	1.00507e+03	0	0	3.14e-04 (3.14e-04)	1	9428.2

Table 3: Results for the real world datasets

To the best of our knowledge, the exact algorithm proposed in [5] represents the actual state-of-the-art. Indeed it is the only algorithm able to exactly solve instances of size larger than 1000, satisfying one of the following strong assumptions (due to the geometrical approach involved): either the instance is on the plane ( $d = 2$ ) or the required number of clusters is large with respect to the number of points. Indeed they were able to solve a TSP instance with  $d = 2$  of size  $n = 2392$  for numbers of clusters ranging from  $k = 2$  to  $k = 10$ , and for large number of clusters ( $k$  between 100 and 400), and an instance of size  $n = 2310$  with  $d = 19$  but only for large number of clusters ( $k$  between 230 and 500). Our algorithm has orthogonal capabilities in some sense to the one proposed in [5], since is not influenced by the number of features (we solve problems with thousands of features, which would be completely out of reach for the algorithm in [5]). Indeed, in the SDP formulation, the number of features is hidden in the matrix  $W$ , which is computed only once, so that it does not influence the computational cost of the algorithm. On the other hand, it is well known that the difficulty (and the gap) of the SDP relaxation (9) increases when the boundary of the clusters are confused, and this phenomenon becomes more frequent when the number of clusters is high with respect to the number of points, and far away from the correct  $k$  for the MSSC objective function. The strength of our bounding procedure is confirmed by 28 problems out of 54 solved at the root. Among these 28 problems, only 8 are tight, in the sense that problem (9) without inequalities produces the optimal solution. The efficiency of

SOS-SDP comes from the combination of the cutting-plane procedure that allows us to close a significant amount of the gap even when the bound without inequalities is not tight, and the heuristic that when the SDP solution is good allows us to find the optimal solution. Note that in 15 out of 34 instances, our algorithm certifies the optimality of a solution that  $k$ -means at the root could not find.

Overall, the number of nodes of the branch-and-bound tree is always smaller than 40, but the computational cost of the single node may be high due to the high number of cutting-plane iterations. The values of  $\text{cuts}_{\text{cp}}$  confirm that the removal of inactive inequalities is effective, and allows to keep the number of inequalities moderate so that the SDP at each cutting-plane iteration is computationally tractable.

## 6.5 Numerical Results of SDP-INIT

In order to test the efficiency of our initialization of constrained  $k$ -means, we report the behaviour at the root node on a subset of real-world datasets. We selected the most popular on the UCI website with size in the range of 150–569. To have more difficult instances, we run the heuristic for all the values of  $k$  in the range from 2 to 10. Note that for  $k$  far from the values suggested by the validation indices, the optimal solution may be constituted by overlapped and confused clusters that are more difficult to find for any heuristic.

In Table 4, we report the results obtained by our heuristic, compared with 50 runs of  $k$ -means initialized with  $k$ -means++ and with random initialization.

In each table, we report:

- the lower bound obtained by solving the basic SDP relaxation ( $LB_0$ ), and the corresponding heuristic solution ( $UB_0$ )
- the lower bound obtained after performing  $CP$  cutting-plane iterations  $LB_{CP}$  and the corresponding heuristic solution ( $UB_{CP}$ )
- the solution produced by  $k$ -means after 50 runs initialized with  $k$ -means++ ( $UB_{++}$ )
- the solution produced by  $k$ -means after 50 runs randomly initialized ( $UB_{RAND}$ )

We highlight the best solution in boldface. The results show that the solution  $UB_{CP}$  is always the best, apart from 1 case. Note that in many cases, the solution  $UB_0$  is fairly competitive both in terms of bound quality and computational effort since it requires the solution of exactly one SDP.

## 7 Conclusions

We developed an exact solution algorithm for the minimum sum-of-squares clustering problem (MSSC) using tools from semidefinite programming. We use a semidefinite relaxation that exploits three types of valid inequalities in a cutting plane fashion to generate tight lower bounds for the MSSC.

$K$	$CP$	$LB_0$	$LB_{CP}$	$UB_0$	$UB_{CP}$	$UB_{++}$	$UB_{RAND}$
Iris dataset							
2	2	1.50679e+02	1.52348e+02	<b>1.52348e+02</b>	<b>1.52348e+02</b>	<b>1.52348e+02</b>	<b>1.52348e+02</b>
3	4	7.55144e+01	7.88421e+01	7.88557e+01	<b>7.88514e+01</b>	7.88518e+01	7.88527e+01
4	6	5.47766e+01	5.72281e+01	<b>5.72285e+01</b>	<b>5.72285e+01</b>	5.72560e+01	5.72560e+01
5	3	4.38467e+01	4.64369e+01	4.64612e+01	<b>4.64462e+01</b>	<b>4.64462e+01</b>	4.64612e+01
6	4	3.67110e+01	3.90175e+01	3.90660e+01	<b>3.90400e+01</b>	3.90660e+01	<b>3.90400e+00</b>
7	6	3.18467e+01	3.42788e+01	3.43058e+01	<b>3.42982e+01</b>	3.44090e+01	3.43859e+01
8	3	2.88697e+01	2.99660e+01	<b>2.99904e+01</b>	<b>2.99904e+01</b>	<b>2.99904e+01</b>	3.04762e+01
9	3	2.64849e+01	2.77836e+01	2.79408e+01	<b>2.77861e+01</b>	2.78921e+01	2.83071e+01
10	4	2.44186e+01	2.58329e+01	2.62712e+01	<b>2.58341e+01</b>	2.59644e+01	2.65776e+01
Glass dataset							
2	6	1.35499e+02	1.36525e+02	1.36537e+02	<b>1.36528e+02</b>	<b>1.36528e+02</b>	1.36537e+02
3	5	1.08991e+02	1.14320e+02	<b>1.14341e+02</b>	<b>1.14341e+02</b>	<b>1.14341e+02</b>	<b>1.14341e+02</b>
4	5	9.14749e+01	9.47742e+01	9.48402e+01	<b>9.47899e+01</b>	9.48402e+01	9.48402e+01
5	6	7.87104e+01	8.34045e+01	8.40062e+01	<b>8.35054e+01</b>	8.42973e+01	8.40502e+01
6	8	6.89918e+01	7.29430e+01	<b>7.29647e+01</b>	<b>7.29647e+01</b>	7.37947e+01	7.43696e+01
7	8	6.19552e+01	6.47908e+01	6.53398e+01	<b>6.47973e+01</b>	7.08087e+01	6.66828e+01
8	6	5.61534e+01	5.85654e+01	5.87606e+01	<b>5.85699e+01</b>	5.90119e+01	6.08941e+01
9	10	5.12932e+01	5.37277e+01	5.41810e+01	<b>5.37580e+01</b>	5.55979e+01	5.61847e+01
10	4	4.70718e+01	4.93411e+01	4.97866e+01	<b>4.97382e+01</b>	5.15837e+01	5.25047e+01
Wholesale dataset							
2	2	3.48221e+03	3.48656e+03	<b>3.48657e+03</b>	<b>3.48657e+03</b>	<b>3.48657e+03</b>	<b>3.48657e+03</b>
3	5	2.85705e+03	2.91234e+03	<b>2.91252e+03</b>	<b>2.91252e+03</b>	<b>2.91252e+03</b>	<b>2.91254e+03</b>
4	9	2.33207e+03	2.46555e+03	<b>2.46558e+03</b>	<b>2.46558e+03</b>	<b>2.46558e+03</b>	<b>2.46558e+03</b>
5	7	1.91575e+03	2.04735e+03	2.04741e+03	<b>2.04735e+03</b>	2.04891e+03	2.04891e+03
6	10	1.63098e+03	1.73382e+03	1.74322e+03	<b>1.73496e+03</b>	1.74096e+03	1.75359e+03
7	12	1.44236e+03	1.52350e+03	1.52551e+03	<b>1.52383e+03</b>	1.52693e+03	1.53677e+03
8	11	1.28695e+03	1.36289e+03	1.36949e+03	<b>1.36290e+03</b>	1.36621e+03	1.39735e+03
9	10	1.14692e+03	1.21928e+03	1.22008e+03	<b>1.21978e+03</b>	<b>1.21978e+03</b>	1.26105e+03
10	6	1.03078e+03	1.07843e+03	1.08010e+03	<b>1.07843e+03</b>	1.13670e+03	1.21282e+03
Wdbc dataset							
2	6	7.54429e+07	7.79415e+07	<b>7.79431e+07</b>	<b>7.79431e+07</b>	<b>7.79431e+07</b>	<b>7.79431e+07</b>
3	27	4.14673e+07	4.72612e+07	4.74219e+07	<b>4.72648e+07</b>	<b>4.72648e+07</b>	4.74999e+07
4	22	2.62662e+07	2.91013e+07	2.92269e+07	<b>2.92265e+07</b>	<b>2.92265e+07</b>	<b>2.92265e+07</b>
5	20	1.90062e+07	2.05248e+07	2.05806e+07	<b>2.05352e+07</b>	<b>2.05352e+07</b>	2.06727e+07
6	6	1.47880e+07	1.55897e+07	1.69771e+07	1.69343e+07	<b>1.66461e+07</b>	1.71215e+07
7	22	1.20747e+07	1.31868e+07	1.32742e+07	<b>1.32470e+07</b>	1.32655e+07	1.33533e+07
8	8	1.02027e+07	1.07390e+07	1.12114e+07	<b>1.12064e+07</b>	1.12441e+07	1.15090e+07
9	3	8.83658e+06	9.09983e+06	<b>9.43290e+06</b>	<b>9.43290e+06</b>	9.47386e+06	1.05951e+07
10	1	7.72013e+06	7.72013e+06	<b>8.37902e+06</b>	<b>8.37902e+06</b>	8.54589e+06	9.83225e+06

Table 4: Heuristic performance for selected datasets. Best upper bounds found are type-set in boldface.

Besides these lower bounds, the semidefinite relaxation also provides a primal solution that can be used for generating data to initialize constrained  $k$ -means, which is known to be sensitive concerning the starting point. Numerical experiments undoubtedly demonstrate the advantage of using this initialization procedure.

We implemented a branch-and-bound algorithm using the ingredients described above. Our way of branching allows us to decrease the size of the problem while going down the branch-and-bound tree. Notably, the shrinking procedure preserves the structure of the problem which is beneficial for our routine computing the bounds in each node of the branch-and-bound tree. Our code is parallelized in two ways: the nodes in the branch-and-bound tree are evaluated in parallel and the bound computation within a node is executed in a multi-threaded MATLAB environment.

The numerical results impressively exhibit the efficiency of our algorithm: we can solve real-world instances up to 4000 data points. To the best of our knowledge, no other exact solution methods can handle generic instances of that size. Moreover, the dimension of the data points does not influence the performance of our algorithm, we solve instances with more than 20 000 features.

Our algorithm can be extended to deal with certain constrained versions of sum-of-squares clustering like those with diameter constraints, split constraints, density constraints, or capacity constraints [15, 23]. This is left for future work. Also, kernel-based clustering is a promising extension that we plan to consider [19]. Finally, we have ideas in mind on how to use our algorithm in a heuristic fashion for obtaining high quality solutions for huge graphs.

## Acknowledgements

Parts of this project were carried out during a research stay of the third author at the University Tor Vergata, funded by the University of Rome Tor Vergata Visiting Professor grant 2018. Furthermore, this project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement MINOA No 764759.

We thank Kim-Chuan Toh for bringing our attention to [30] and for providing an implementation of the method therein.

## References

- [1] Khaled S. Al-Sultan. A Tabu search approach to the clustering problem. *Pattern Recognition*, 28(9):1443–1451, 1995. ISSN 0031-3203. doi: [https://doi.org/10.1016/0031-3203\(95\)00022-R](https://doi.org/10.1016/0031-3203(95)00022-R).
- [2] Daniel Aloise and Pierre Hansen. A branch-and-cut SDP-based algorithm for minimum sum-of-squares clustering. *Pesquisa Operacional*, 29(3):503–516, 2009.
- [3] Daniel Aloise and Pierre Hansen. Evaluating a branch-and-bound RLT-based algo-



- rithm for minimum sum-of-squares clustering. *Journal of Global Optimization*, 49(3):449–465, 2011.
- [4] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248, 2009.
  - [5] Daniel Aloise, Pierre Hansen, and Leo Liberti. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, 131(1):195–220, 2012.
  - [6] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.
  - [7] Pranjal Awasthi, Afonso S Bandeira, Moses Charikar, Ravishankar Krishnaswamy, Soledad Villar, and Rachel Ward. Relax, no need to round: Integrality of clustering formulations. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, pages 191–200, 2015.
  - [8] Adil M. Bagirov, Sona Taheri, and Julien Ugon. Nonsmooth DC programming approach to the minimum sum-of-squares clustering problems. *Pattern Recognition*, 53:12–24, 2016. ISSN 0031-3203. doi: <https://doi.org/10.1016/j.patcog.2015.11.011>.
  - [9] Michael J Brusco. A repetitive branch-and-bound procedure for minimum within-cluster sums of squares partitioning. *Psychometrika*, 71(2):347–363, 2006.
  - [10] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.
  - [11] G. Caruso, S.A. Gattone, F. Fortuna, and T. Di Battista. Cluster analysis for mixed data: An application to credit risk evaluation. *Socio-Economic Planning Sciences*, 73:100850, 2021. ISSN 0038-0121. doi: <https://doi.org/10.1016/j.seps.2020.100850>. Statistical methods & models for the evaluation systems of the public sector.
  - [12] Stefan Chmiela, Huziel E Sauceda, Igor Poltavsky, Klaus-Robert Müller, and Alexandre Tkatchenko. sGDML: Constructing accurate and data efficient molecular force fields using machine learning. *Computer Physics Communications*, 240: 38–45, 2019.
  - [13] Sunil Chopra and Mendu R Rao. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993.
  - [14] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. The UCR Time Series Classification Archive, October 2018. [https://www.cs.ucr.edu/~eamonn/time\\_series\\_data\\_2018/](https://www.cs.ucr.edu/~eamonn/time_series_data_2018/).

- [15] Ian Davidson and SS Ravi. Clustering with constraints: Feasibility issues and the k-means algorithm. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 138–149. SIAM, 2005.
- [16] David L Davies and Donald W Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, 1(2):224–227, 1979.
- [17] Frank de Meijer and Renata Sotirov. SDP-based bounds for the Quadratic Cycle Cover Problem via cutting plane augmented Lagrangian methods and reinforcement learning, 2021. to appear in *INFORMS Journal on Computing*.
- [18] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015.
- [19] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 551–556, 2004.
- [20] George Diehr. Evaluation of a branch and bound algorithm for clustering. *SIAM Journal on Scientific and Statistical Computing*, 6(2):268–284, 1985.
- [21] Olivier Du Merle, Pierre Hansen, Brigitte Jaumard, and Nenad Mladenovic. An interior point algorithm for minimum sum-of-squares clustering. *SIAM Journal on Scientific Computing*, 21(4):1485–1505, 1999.
- [22] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL <http://archive.ics.uci.edu/ml>.
- [23] Khanh-Chuong Duong, Christel Vrain, et al. Constrained clustering by constraint programming. *Artificial Intelligence*, 244:70–94, 2017.
- [24] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.
- [25] Pasi Fränti and Sami Sieranoja. How much can k-means be improved by using better initialization and repeats? *Pattern Recognition*, 93:95–112, 2019.
- [26] Bissan Ghaddar, Miguel F Anjos, and Frauke Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. *Annals of Operations Research*, 188(1):155–174, 2011.
- [27] Daniel Gribel and Thibaut Vidal. HG-means: A scalable hybrid genetic algorithm for minimum sum-of-squares clustering. *Pattern Recognition*, 88:569–583, 2019.
- [28] Pierre Hansen and Nenad Mladenović. J-Means: a new local search heuristic for minimum sum of squares clustering. *Pattern Recognition*, 34(2):405–413, 2001. ISSN 0031-3203. doi: [https://doi.org/10.1016/S0031-3203\(99\)00216-2](https://doi.org/10.1016/S0031-3203(99)00216-2).

- [29] Takayuki Iguchi, Dustin G Mixon, Jesse Peterson, and Soledad Villar. Probably certifiably correct k-means clustering. *Mathematical Programming*, 165(2):605–642, 2017.
- [30] Christian Jansson, Denis Chaykin, and Christian Keil. Rigorous error bounds for the optimal value in semidefinite programming. *SIAM Journal on Numerical Analysis*, 46(1):180–200, 2007/08. doi: 10.1137/050622870. URL <https://doi.org/10.1137/050622870>.
- [31] Daxin Jiang, Chun Tang, and Aidong Zhang. Cluster analysis for gene expression data: A survey. *IEEE Transactions on knowledge and data engineering*, 16(11):1370–1386, 2004.
- [32] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry*, 28(2-3):89–112, 2004.
- [33] Napsu Karmitsa, Adil M. Bagirov, and Sona Taheri. A clustering algorithm using an evolutionary programming-based approach. *Pattern Recognition Letters*, 18(10):975–986, 1997. ISSN 0167-8655. doi: [https://doi.org/10.1016/S0167-8655\(97\)00122-0](https://doi.org/10.1016/S0167-8655(97)00122-0).
- [34] Napsu Karmitsa, Adil M. Bagirov, and Sona Taheri. New diagonal bundle method for clustering problems in large data sets. *European Journal of Operational Research*, 263(2):367–379, 2017. ISSN 0377-2217. doi: <https://doi.org/10.1016/j.ejor.2017.06.010>.
- [35] Napsu Karmitsa, Adil M. Bagirov, and Sona Taheri. Clustering in large data sets with the limited memory bundle method. *Pattern Recognition*, 83:245–259, 2018. ISSN 0031-3203. doi: <https://doi.org/10.1016/j.patcog.2018.05.028>.
- [36] Warren L. G. Koontz, Patrenahalli M. Narendra, and Keinosuke Fukunaga. A branch and bound clustering algorithm. *IEEE Transactions on Computers*, 100(9):908–915, 1975.
- [37] Julian Lee and David Perkins. A simulated annealing algorithm with a dual perturbation method for clustering. *Pattern Recognition*, 112:107713, 2021.
- [38] Xiaodong Li, Yang Li, Shuyang Ling, Thomas Strohmer, and Ke Wei. When do birds of a feather flock together?  $k$ -means, proximity, and conic programming. *Mathematical Programming*, 179(1-2, Ser. A):295–341, 2020.
- [39] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.
- [40] Stuart Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

- [41] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Sympos. Math. Statist. and Probability (Berkeley, Calif., 1965/66)*, pages Vol. I: Statistics, pp. 281–297. Univ. California Press, Berkeley, Calif., 1967.
- [42] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The planar k-means problem is NP-hard. *Theoretical Computer Science*, 442:13–21, 2012.
- [43] Mehrdad Mahdavi and Hassan Abolhassani. Harmony k-means algorithm for document clustering. *Data Mining and Knowledge Discovery*, 18(3):370–391, 2009.
- [44] Pierluigi Mansueto and Fabio Schoen. Memetic differential evolution methods for clustering problems. *Pattern Recognition*, 114:107849, 2021. ISSN 0031-3203. doi: <https://doi.org/10.1016/j.patcog.2021.107849>.
- [45] Ujjwal Maulik and Sanghamitra Bandyopadhyay. Genetic algorithm-based clustering technique. *Pattern Recognition*, 33(9):1455–1465, 2000. ISSN 0031-3203. doi: [https://doi.org/10.1016/S0031-3203\(99\)00137-5](https://doi.org/10.1016/S0031-3203(99)00137-5).
- [46] Dustin G Mixon, Soledad Villar, and Rachel Ward. Clustering subgaussian mixtures by semidefinite programming. *Information and Inference: A Journal of the IMA*, 6(4):389–415, 2017.
- [47] V I Orlov, L A Kazakovtsev, I P Rozhnov, N A Popov, and V V Fedosov. Variable neighborhood search algorithm for k-means clustering. *IOP Conference Series: Materials Science and Engineering*, 450:022035, dec 2018. doi: 10.1088/1757-899x/450/2/022035. URL <https://doi.org/10.1088/1757-899x/450/2/022035>.
- [48] Jiming Peng and Yu Wei. Approximating k-means-type clustering via semidefinite programming. *SIAM journal on optimization*, 18(1):186–205, 2007.
- [49] Jiming Peng and Yu Xia. A new theoretical framework for k-means-type clustering. In *Foundations and advances in data mining*, pages 79–96. Springer, 2005.
- [50] Madhushini Narayana Prasad and Grani A Hanasusanto. Improved conic reformulations for k-means clustering. *SIAM Journal on Optimization*, 28(4):3105–3126, 2018.
- [51] MR Rao. Cluster analysis and mathematical programming. *Journal of the American statistical association*, 66(335):622–626, 1971.
- [52] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.
- [53] Conrad Sanderson and Ryan Curtin. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2):26, 2016.

- [54] Hanif D Sherali and Warren P Adams. Reformulation-linearization techniques for discrete optimization problems. In *Handbook of combinatorial optimization*, pages 479–532. Springer, 1998.
- [55] Hanif D Sherali and Jitamitra Desai. A global optimization RLT-based approach for solving the hard clustering problem. *Journal of Global Optimization*, 32(2):281–306, 2005.
- [56] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.
- [57] MA Syakur, BK Khotimah, EMS Rochman, and BD Satoto. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP Conference Series: Materials Science and Engineering*, volume 336, page 012017. IOP Publishing, 2018.
- [58] Pham Dinh Tao et al. New and efficient DCA based algorithms for minimum sum-of-squares clustering. *Pattern Recognition*, 47(1):388–401, 2014.
- [59] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *Icml*, volume 1, pages 577–584, 2001.
- [60] Haizhou Wang and Mingzhou Song. Ckmeans. 1d. dp: optimal k-means clustering in one dimension by dynamic programming. *The R journal*, 3(2):29, 2011.
- [61] Liuqin Yang, Defeng Sun, and Kim-Chuan Toh. SDPNAL+: a majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints. *Math. Program. Comput.*, 7(3):331–366, 2015. ISSN 1867-2949. doi: 10.1007/s12532-015-0082-6. URL <https://doi.org/10.1007/s12532-015-0082-6>.
- [62] Shyr-Shen Yu, Shao-Wei Chu, Chuin-Mu Wang, Yung-Kuan Chan, and Ting-Cheng Chang. Two improved k-means algorithms. *Applied Soft Computing*, 68:747–755, 2018. ISSN 1568-4946. doi: <https://doi.org/10.1016/j.asoc.2017.08.032>.
- [63] Chengqi Zhang, Yongsong Qin, Xiaofeng Zhu, Jilian Zhang, and Shichao Zhang. Clustering-based missing value imputation for data preprocessing. In *2006 4th IEEE International Conference on Industrial Informatics*, pages 1081–1086. IEEE, 2006.
- [64] Xin-Yuan Zhao, Defeng Sun, and Kim-Chuan Toh. A Newton-CG augmented Lagrangian method for semidefinite programming. *SIAM J. Optim.*, 20(4):1737–1765, 2010. ISSN 1052-6234. doi: 10.1137/080718206. URL <https://doi.org/10.1137/080718206>.

Veronica Piccialli, [veronica.piccialli@uniroma2.it](mailto:veronica.piccialli@uniroma2.it), University of Rome Tor Vergata, via del Politecnico, 00133 Rome, Italy ORCID: 0000-0002-3357-9608

Antonio M. Sudoso, [antonio.maria.sudoso@uniroma2.it](mailto:antonio.maria.sudoso@uniroma2.it), University of Rome Tor Vergata, via del Politecnico, 1, 00133 Rome, Italy ORCID: 0000-0002-2936-9931

Angelika Wiegele, [angelika.wiegele@aau.at](mailto:angelika.wiegele@aau.at), Alpen-Adria-Universität Klagenfurt, Universitätsstraße 65–67, 9020 Klagenfurt, Austria, ORCID: 0000-0003-1670-7951