Noname manuscript No.
(will be inserted by the editor)

An improved column generation algorithm for minimum sum-of-squares clustering

Daniel Aloise · Pierre Hansen · Leo Liberti

Received: date / Accepted: date

Abstract Given a set of entities associated with points in Euclidean space, minimum sum-of-squares clustering (MSSC) consist in partitioning this set into clusters such that the sum of squared distances from each point to the centroid of its cluster is minimized. A column generation algorithm for MSSC was given in du Merle et al. [15]. The bottleneck of that algorithm is resolution of the auxiliary problem of finding a column with negative reduced cost. We propose a new way to solve this auxiliary problem based on geometric arguments. This greatly improves the efficiency of the whole algorithm and leads to exact solution of instances with over 2300 entities, i.e., more than 10 times as much as previously done.

Keywords clustering, sum-of-squares, column generation, ACCPM

Research of the first author has been supported by CAPES/Brazil grant number 2479-04-4. Research of the second author has been supported by NSERC grant number 105574-07, FQRNT grant 2007-PR-112176 and the Data Mining Chair of HEC Montréal.

Daniel Aloise
École Polytechnique de Montréal, Mathématiques et Génie Industriel, C.P. 6079, Succ. Centre-Ville, Montréal, Québec, Canada, H3C 3A7
E-mail: daniel.aloise@gerad.ca

Pierre Hansen
GERAD and HEC Montréal, 3000, Chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada, H3T 2A7 and
LIX, École Polytechnique, F-91128 Palaiseau, France
E-mail: pierre.hansen@gerad.ca

Leo Liberti
LIX, École Polytechnique, F-91128 Palaiseau, France
E-mail: liberti@lix.polytechnique.fr

1 Introduction

Clustering is a basic chapter in data analysis. It adresses the following problem: given a set of entities find subsets, called clusters, which are homogeneous and/or well separated (e.g. Hartigan [28]; Jain, Murty and Flynn [31]; Mirkin [44]). Many different criteria are used in the literature to express homogeneity and/or separation of the clusters to be found (see [22] for a survey). Among them, a frequently used one is the minimum sum of squared Euclidean distances from each entity to the centroid of the cluster to which it belongs. Partitioning n entities into k clusters with this criterion is known as minimum sum-of-squares clustering (MSSC).

For $k \geq 2$ and one dimensional data, MSSC can be solved in $O(n^3)$ time [57]. The problem is NP-hard in the plane for general values of k [42]. In general dimension, MSSC is NP-hard even for $k = 2$ [1]. If both k and dimension s are fixed, the problem can be solved in $O(n^{sk+1})$ time [30], which may be very time-consuming even for instances in the plane.

MSSC has several properties:

(i) It expresses both homogeneity and separation as explained in Späth's book [57], pages 60–61;
(ii) Given the assignments, the cluster centers are located in their centroids, due to first order optimality conditions. These are determined by a simple closed-form expression;
(iii) Given the centroids, each entity is assigned to its closest centroid, due to local optimality. This just requires a few comparisons;
(iv) Clusters obtained are spheroidal due to minimization of squared Euclidean distances. This may be desirable or not, depending on the problem considered.

Mathematical properties of MSSC are discussed in the books of Späth [57], Mirkin [45] and Kogan [34]. Several hundred papers have been written on heuristics for MSSC and several thousand on their applications in many domains (see, for instance, Steinley's half century synthesis [58]). The best known heuristic for MSSC is k-means [41] (indeed MSSC is sometimes called the k-means problem). This heuristic alternately applies properties (ii) and (iii) above until a local optimum is reached. It has been shown by Hansen and Mladenović [24] that while k-means usually gives good results for small number of clusters its performance deteriorates, sometimes drastically, when this number increases. Modifying k-means by adding a *jump* move of a centroid to an entity location gives a much better heuristic called j-means. Finally, combining j-means with a Variable Neighborhood Search (VNS) heuristic [25],[26],[46] gives a heuristic which often provides optimal solutions or best known ones. This empirical observation will be exploited in the algorithm presented below.

Other recent heuristics for MSSC include the global k-means method of Likas, Vlassis and Verbeek [40], analyzed in [27] and modified by Bagirov [6], Bagirov and Yearwood's nonsmooth optimization algorithm [7], smoothing optimization algorithms due to Teboulle and Kogan [60] and Xavier et al. [64],

Merz's iterated local search [43], Pacheco's scatter search [47], Pacheco and Valencia's hybrids [48], Taillard's decomposition methods [59], Laszlo and Mukherjee's genetic algorithms [36][37], Christou's restricted column generation and partitioning method [11], and the D.C. heuristic of An, Belghiti and Tao [4]. A systematic comparison of twelve heuristics for MSSC was made by Brusco and Steinley in [10].

Exact algorithms for MSSC are much less numerous than heuristics. To the best of our knowledge, there are less than a dozen papers published on that topic. However, the approaches followed are very diverse. Early branch-and-bound algorithms are due to Koontz, Narendra and Fukunaga [35] and Diehr [12]. Bounds depend on distances between entities assigned to the same cluster and a limited look-ahead component.

A column generation method for MSSC was proposed by du Merle et al. in [15]. It solved for the first time medium size benchmark instances (i.e., instances with 100-200 entities), including Fisher's 150 Iris [19]. The master problem is solved by the ACCPM interior point method of Goffin, Haurie, and Vial [20]. The auxiliary of finding a column with negative reduced cost is expressed by as a hyperbolic programming in 0-1 variables. It is solved by a Dinkelbach-like algorithm [13] which relies on a branch-and-bound algorithm for unconstrained quadratic 0-1 optimization. Another branch-and-bound on the master problem leads, if needed, to an integer solution. Finally, VNS heuristics are used both at the ouset to find a good initial solution together with tentative bounds on the dual variables, as well as in the auxiliary problem to accelerate its solution. The bottleneck of the algorithm lies in the resolution of its auxiliary problem, and more precisely, in the unconstrained quadratic 0-1 optimization problem arising there. In this paper, we will propose an alternate geometric-based approach for that step.

More recently, Xia and Peng [65] proved that the objective function of MSSC is concave in the relaxed feasible domain. In their paper, they propose an adaptation of Tuy's [61] cutting plane method to solve it. Approximate results are reported for a version where this algorithm is halted before global convergence. Some experiments of ours showed that small instances with about 25 entities can be solved exactly with that approach.

MSSC can also be solved by non-serial dynamic programming as shown by Jensen [32]. An improved implementation due to van Os and Meulman [62] allows solutions of instances with about 28 entities.

Brusco [9] proposed a repetitive branch-and-bound procedure which, after ordering the entities, solves by branch-and-bound the problem defined by the $k+1$ last ones, then the problem with $k+2$ last ones, and so on, until the problem with all given entities is solved. The bound used at any iteration of one of those iterated branch-and-bound procedures comprises two components, i.e., an usual one corresponding to distances between already assigned entities and a sophisticated look-ahead one which corresponds to distances in an optimal solution for the set of unassigned entities. These much improved bounds led to efficient solution of some well-known benchmark instances, including Fisher's 150 iris [19], particularly when the number of cluster is small. Artificially

generated examples with well-separated clusters and up to $n = 220$ entities could be solved also.

The hardest task when devising exact algorithms for MSSC is to compute good lower bounds in a reasonable amount of time. Sherali and Desai [56] proposed to obtain such bounds by linearizing the model via the reformulation-linearization technique [55]. They claim to solve instances with up to 1,000 entities by means of a branch-and-bound algorithm. However, these results could not be reproduced and computing times in an attempted replication of [3] were already high for real data sets with about 22 entities.

Recently, Peng and Xia [51] proved the equivalence of MSSC and a model called 0-1 semidefinite programming (SDP), in which eigenvalues are binary. On the basis of these results, the present authors developed in [2] a branch-and-cut SDP-based algorithm for MSSC with lower bounds obtained from the LP relaxation of the 0-1 SDP model. This algorithm obtains exact solutions for fairly large data sets, i.e., $n = 202$ and $k \geq 9$, with computing times comparable with those obtained by the column generation method proposed by du Merle et al. [15].

This paper is organized as follows. Section 2 revisits the formulation of the problem and the column generation approach. In Section 3, we show how the auxiliary problem can be solved for MSSC instances in the Euclidean plane by taking advantage of its geometric properties. This is made by a connexion with the Weber problem with limited distances [14]. Section 4 shows how the geometric reasoning can be further exploited to solve auxiliary problems arising from the resolution of MSSC instances in general Euclidean space. Computational experiments for instances commonly used in the literature are reported in Section 5. Finally, conclusions are given in Section 6.

2 Column generation algorithm revisited

A mathematical programming formulation of MSSC is as follows:

$$\begin{aligned} \min_{x,y} \quad & \sum_{i=1}^{n} \sum_{j=1}^{k} x_{ij} \|p_i - y_j\|^2 \\ \text{subject to} \quad & \\ & \sum_{j=1}^{k} x_{ij} = 1, \quad \forall i = 1, \ldots, n \\ & x_{ij} \in \{0, 1\}, \quad \forall i = 1, \ldots, n; \forall j = 1, \ldots, k. \end{aligned} \tag{1}$$

The n entities $\{o_1, o_2, \ldots, o_n\}$ to be clustered are at given points $p_i = (p_i^r, r = 1, \ldots, s)$ of $\mathbb{R}^s$ for $i = 1, \ldots, n$; k cluster centers must be located at unknown points $y_j \in \mathbb{R}^s$ for $j = 1, \ldots, k$; the norm $\|\cdot\|$ denotes the Euclidean distance between the two points in its argument in the s-dimensional space under consideration. The decision variables x_{ij} express the assignment of the entity o_i to the cluster j. We assume that the number of entities n is greater than k, otherwise the problem is trivially solved by locating one cluster center at the position of each entity.

If y is fixed, the condition $x_{ij} \in \{0,1\}$ can be replaced by $x_{ij} \in [0,1]$, since in an optimal solution for the resulting problem each entity belongs to the cluster with the nearest center. Besides, for a fixed x, first order conditions on the gradient of the objective function require that at an optimal solution

$$\sum_{i=1}^{n} x_{ij}(y_j^r - p_i^r) = 0, \ \forall j, r, \quad \text{i.e.,} \quad y_j^r = \frac{\sum\limits_{i=1}^{n} x_{ij} p_i^r}{\sum\limits_{i=1}^{n} x_{ij}}, \ \forall j, r. \tag{2}$$

Hence, the optimal cluster centers are always at the centroids of the clusters.

Partitioning problems in cluster analysis can also be mathematically formulated by considering all possible clusters. Let us consider any cluster C_t for which

$$a_{it} = \begin{cases} 1 & \text{if entity } o_i \text{ belongs to cluster } C_t \\ 0 & \text{otherwise,} \end{cases}$$

and let us denote by y_t the centroid of points p_i such that $a_{it} = 1$. Thus, the cost c_t of cluster C_t can be written as

$$c_t = \sum_{i=1}^{n} \|p_i - y_t\|^2 a_{it}.$$

An alternative formulation for MSSC is then given by

$$\begin{array}{l} \min\limits_{z} \sum\limits_{t \in T} c_t z_t \\ \text{subject to} \\ \sum\limits_{t \in T} a_{it} z_t = 1, \quad \forall i = 1, \ldots, n \\ \sum\limits_{t \in T} z_t = k \\ z_t \in \{0,1\} \quad \forall t \in T, \end{array} \tag{3}$$

where $T = \{1, \ldots, 2^n - 1\}$. The z_t variables are equal to 1 if cluster C_t is in the optimal partition and to 0 otherwise. The first set of constraints state that each entity belongs to one cluster, and the following constraint expresses that the optimal partition contains exactly k clusters. Without loss of generality, they can be replaced by

$$\sum_{t \in T} a_{it} z_t \geq 1, \quad \forall i = 1, \ldots, n, \quad \text{and} \quad \sum_{t \in T} z_t \leq k$$

This is a large linear partitioning problem with a size constraint, for which the number of variables is exponential in the number n of entities. Therefore, it cannot be explicitly written and solved in a straightforward way unless n is small. The column generation method proposed in [15] works with a reasonably small subset $T' \subseteq T$ of the columns in (3), i.e., with a *restricted master problem*. The method is combined with branch-and-bound in order

to solve exactly (3) for medium size (about 100-200 entities) to fairly large instances (1000 entities or more).

Resolution of (3) is iteratively done by augmenting the number of columns in the restricted master problem until optimality is proved with the columns at hand. Entering columns are found by solving an auxiliary problem, i.e., finding the list of entities of a cluster whose associated variable in (3) has negative reduced cost. Since a standard column generation method for solving the linear relaxation of the formulation (3) suffers from very slow convergence due to high degeneracy, two strategies for stabilizing column generation [16] were used and compared in [15]. That one for which the linear relaxation is solved by an interior-point algorithm, i.e., the weighted version of the analytic center cutting plane method (ACCPM) of Goffin, Haurie, and Vial [20], was found to be the best.

Once the linear relaxation of the problem is solved, the integrality of the obtained solution is checked (and often found to hold for small to medium size problems with few clusters). Then, if the solution is not integer, branching is needed. The branching rule used in [15] is the standard one, due to Ryan and Foster [54], i.e., branching by imposing in one hand that two entities belong to the same cluster and on the other hand that at most one of these entities belongs to any given cluster.

2.1 Auxiliary problem

The biggest obstacle for an efficient exact resolution of the MSSC via column generation is the difficulty of the auxiliary problem. The dual of the formulation (3) is expressed by

$$\begin{array}{lll} \max & -k\sigma + \sum_{i=1}^{n} \lambda_i & \\ \text{subject to} & & \\ & -\sigma + \sum_{i=1}^{n} a_{it}\lambda_i \leq c_t & \forall t \in T \\ & \lambda_i \geq 0 & i = 1, \ldots, n \\ & \sigma \geq 0, & \end{array} \tag{4}$$

where the λ_i for $i = 1, \ldots, n$ and σ are dual variables associated with the covering constraints and with the cardinality constraint.

Problem (4) is solved using a cutting plane method, starting with a relaxation and adding constraints as necessary. In classical Kelley's cutting plane method [33], cuts are generated at an extreme point of the relaxed dual formulation. However, Kelley's method is known to slow down considerably in the presence of degeneracy [16]. ACCPM tackles this shortcoming by generating cuts at an analytic center of the current dual feasible region (cf. [18]). In both

cases, given dual values λ, σ, a violated cut is searched to be added to the relaxed dual problem. The violation π_t of a constraint is given by

$$\pi_t = c_t + \sigma - \sum_{i=1}^{n} \lambda_i a_{it}.$$

Since we are interested in finding violated constraints $\pi_t < 0$. The auxiliary problem is then given by $\pi^* = \min_t \pi_t$. Although the enumeration of π_t for all $t \in T$ is too expensive, the value of π^* can be found by solving

$$\pi^* = \sigma + \min_{y_v \in \mathbb{R}^s, v \in \mathbb{B}^n} \sum_{i=1}^{n} (\|p_i - y_v\|^2 - \lambda_i) v_i. \tag{5}$$

with y_v denoting the centroid of points p_i for which $v_i = 1$. If $\pi^* < 0$, then the optimal solution v^* to (5) is added as a cut to the relaxed dual problem (in the primal, this is equivalent to adding a column to the restricted master problem together with its associated primal variable). Otherwise, problem (4) (or equivalently, problem (3)) is solved optimally.

From Huygens' theorem (e.g., Edwards and Cavalli-Sforza [17]), which states that the sum of squared distances from all entities of a given cluster to its centroid is equal to the sum of squared distances between pairs of entities of this cluster divided by its cardinality, problem (5) can be expressed by

$$\begin{aligned} \pi^* &= \sigma + \min_{v \in \mathbb{B}^n} \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \|p_i - p_j\|^2 v_i v_j}{\sum_{i=1}^{n} v_i} - \sum_{i=1}^{n} \lambda_i v_i \\ &= \sigma + \min_{v \in \mathbb{B}^n} \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (\|p_i - p_j\|^2 - \lambda_i - \lambda_j) v_i v_j - \sum_{i=1}^{n} \lambda_i v_i}{\sum_{i=1}^{n} v_i}. \end{aligned} \tag{6}$$

It is a hyperbolic (or fractional) program in 0-1 variables with quadratic numerator and linear denominator. This problem is solved in [15] by an adaptation to binary variables of Dinkelbach's algorithm [13]. This algorithm begins with a temptative value for (6) then reduces the problem to unconstrained quadratic 0-1 optimization by multiplying both sizes by the denominator and regrouping terms. If a positive value is obtained for the optimal solution of this last problem its corresponding value in (6) is computed and the procedure iterated. Its most expensive step is the resolution of a sequence of unconstrained quadratic 0-1 programs, which are solved in [15] by a VNS heuristic until optimality must be checked by a branch-and-bound algorithm.

3 A geometric approach

The auxiliary problem (5) can be viewed as minimizing the sum of functions equal to squared distances from the cluster center y_v to each of the entities, but with a limit on each of the distances, after which the corresponding function does not increase anymore. Clearly, for a given location y_v, v_i is equal to 1 if $\|p_i - y_v\|^2 \leq \lambda_i$, and to 0 otherwise. Geometrically, in the plane, this is equivalent to the condition that $v_i = 1$ if y_v belongs to a disc with radius $\sqrt{\lambda_i}$ centered at p_i, and 0 otherwise.

A branch-and-bound algorithm based on the vector v would consider implicitly all 2^n subproblems generated by branching on binary variables v_i for $i = 1, \ldots, n$, while adding constraints $\|p_i - y_v\|^2 \leq \lambda_i$ and $\|p_i - y_v\|^2 \geq \lambda_i$ to the resulting subproblems. However, the resulting problems pertain to D.C. programming and are difficult to solve. Another possibility is to focus on components v_i of v which are equal to 1. We then consider subproblems of the following type:

$$\begin{array}{ll} \min_y & \sum_{i \in S} \|p_i - y\|^2 \\ \text{subject to} & \\ \|p_i - y\|^2 \leq \lambda_i & \forall i \in S, \end{array} \tag{7}$$

where $S \subseteq \{1, 2, \ldots, n\}$ is a non-empty set. Suproblems of type (7) are convex programming problems. Proposition 1 shows that an optimal solution for (5) is guaranteed to be an optimal solution to a subproblem of type (7).

Proposition 1 *Let (y_v^*, v^*) be the optimal solution to (5). Then, y_v^* is the optimal solution to a subproblem of type (7) with a set S for which $\|p_i - y_v^*\|^2 > \lambda_i$ for all $i \notin S$.*

Proof Define S^* as the index set of all points p_i such that $\|p_i - y_v^*\|^2 \leq \lambda_i$. Thus, for $i \notin S^*$, $\|p_i - y_v^*\|^2 > \lambda_i$. Now let y' be the optimal solution for (7) with S^* and suppose that y_v^* is not the optimal solution for it. Since, $\|p_i - y_v^*\|^2 > \min\{\|p_i - y'\|^2, \lambda_i\}$ for all $i \notin S^*$, the cost of (y', v^*) is smaller than that of (y_v^*, v^*) in (5), which is a contradiction. □

The auxiliary problem (5) still has another very important property which states that at optimal solution (v_v^*, v^*), y_v^* is at the centroid of points p_i for which $v_i^* = 1$. Given a subproblem of type (7) with index set S, this implies that if the centroid of the points p_i such that $i \in S$ is not a feasible solution, then we conclude that the subproblem does not contain the optimal solution to (5). In the plane, it amounts to say that the centroid must belong to the intersection of all discs with index $i \in S$ (which includes the particular case where S is a singleton).

Let us define A as the set of discs whose boundaries intersect at least one other boundary of a disc in two points, and B as the set of discs that do not belong to A. They include isolated discs and nested discs (i.e., discs that

contain another discs in their interior and discs that are entirely contained into other ones). An useful result is shown by the following proposition:

Proposition 2 *The number T of distinct regions which are intersection of discs $\|p_i - y\|^2 \leq \lambda_i$ is bounded by $2n(n-1)$.*

Proof The total number of points of intersection among discs in A is at most $|A|(|A|-1)$. Since each one of them can be associated with at most 4 different regions, and as each of these regions contains at least two of these points, the number of regions r_A which are delimited by discs in A is bounded by $2|A|(|A|-1)$.

Each one of the discs in B can delimit at most one region. Consequently, the number of regions r_B delimited by discs in B is equal to $|B|$.

Thus,

$$\begin{aligned} T = r_A + r_B &\leq 2|A|(|A|-1) + |B| \\ &\leq 2(|A|+|B|)(|A|+|B|-1) \\ &\leq 2n(n-1) \end{aligned}$$

□

Proposition 2 implies that the number of subproblems of type (7) that need to be solved in order to obtain an optimal solution to (5) is polynomially bounded.

An algorithm was proposed in [14] for a similar problem in location theory, i.e., the 1-center Weber problem with limited distances. The only difference between this problem and (5) lies in the fact that Euclidean distances are used instead of squared ones. The algorithm proceeds by considering all intersection points between discs in the plane, and then solves, for each one of these points, the subproblems of type (7) corresponding to the four possible regions which are adjacent to the point. For instance, suppose that p is an intersection point between discs centered at points p_i and p_j, then the four possible non-empty index sets corresponding to regions for which p can be a vertex are formed by: $S_a = \{\ell : \|p_\ell - p\|^2 \leq \lambda_\ell, \ell \neq i, j\}$; $S_b = \{i\} \cup S_a$; $S_c = \{j\} \cup S_a$; and $S_d = \{i, j\} \cup S_a$.

It appears that the algorithm of [14] implicitly assumes that regions delimited by discs in B either do not exist or can be discarded for evaluation. However, this is not true either for the 1-center Weber problem with limited distances or for (5), which makes the algorithm proposed in [14] incomplete.

Figure 1 exhibits an auxiliary problem configuration which appears after 11 iterations of our column generation algorithm while clustering the 10 points described at the top of Figure 1 into 3 clusters. The shaded region (2) in the figure corresponds to the optimal solution of the auxiliary problem while region (1) is the solution provided if the algorithm of [14] is used instead.

Algorithm 1 presents the new algorithm obtained after completing the algorithm of [14] in order to consider sets S corresponding to regions delimited

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
	690	190	823	73	782	338	287	410	769	962
	166	887	695	125	979	894	340	263	768	831
$\sqrt{\lambda}$	382.78	360.47	203.34	379.22	208.24	168.79	211.61	198.70	138.14	332.88

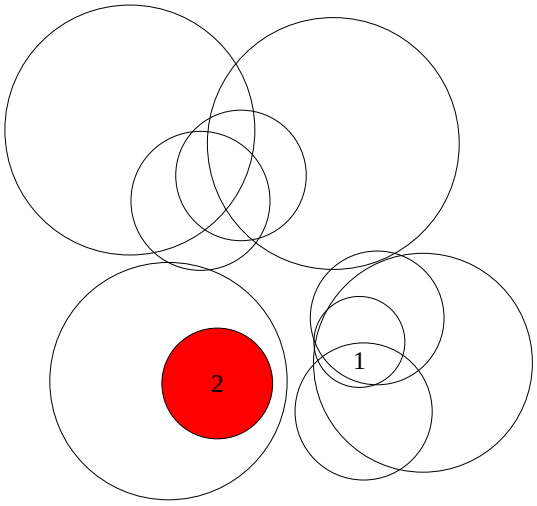

Fig. 1 Configuration of convex regions experimentally obtained

by discs of B. This algorithm requires $O(n^3)$ time since there are $O(n^2)$ possible intersection points and step 4 takes $O(n)$ time per subproblem. Additional operations due to steps 6-8 are performed in $O(n^2)$ time.

Algorithm 1

1. Enumerate all intersection points of pairs of convex regions in the plane as well as all discs whose boundary does not intersect any other one. Let L_1 and L_2 be the corresponding lists.
2. For each intersection point $p \in L_1$ defined by discs centered at points p_i and p_j, find the set S of all k such that $k \neq i, j$ and $\|p_k - p\|^2 \leq \lambda_k$.
3. Consider four sets: $S, S \cup \{i\}, S \cup \{j\}$, and $S \cup \{i, j\}$.
4. Solve subproblems of type (7) defined by each of these sets.
5. Update the best solution if an improving one is found.
6. For each disc in L_2 find the set S' composed of its own index and the indices of all discs containing it.
7. Solve subproblems of type (7) defined by each S'.
8. Update the best solution if an improving one is found.

The simple following condition holds if two discs associated to points p_i and p_j intersect

$$\|p_i - p_j\| \leq \sqrt{\lambda_i} + \sqrt{\lambda_j},$$

one disc being contained in the other if

$$\|p_i - p_j\| \leq |\sqrt{\lambda_i} - \sqrt{\lambda_j}|.$$

Based on these conditions, an acceleration procedure for Algorithm 1 is to build for each point p_i, $i = 1, \ldots, n$ a list of non-decreasing distances to any other point. In step 1 of Algorithm 1, each point p_i is tested in turn with all other points p_j for $j = 1, \ldots, n$, such that $j > i$, in order to know if their respective discs intersect. Indeed these points can be considered in the order given by the sorted list of p_i and the search for intersections halted as soon as

$$\|p_i - p_j\| > \sqrt{\lambda_i} + \sqrt{\lambda_{max}},$$

where $\lambda_{max} = \max\{\lambda_i\}$ for $i = i + 1, \ldots, n$. Note that exactly the same test can be used in order to speed up step 2 of the algorithm.

3.1 Branching

The classical branching rule is applied whenever branching is needed to solve (3). It consists on finding two rows i_1, i_2 such that there are two columns t_1 and t_2 with fractional values at the optimum and such that $a_{i_1 t_1} = a_{i_2 t_1} = 1$ and $a_{i_1 t_2} = 1, a_{i_2 t_2} = 0$. Then, constraints are introduced in the auxiliary problem of both subproblems in the form (i) $v_{i_1} = v_{i_2}$ for one branch, and (ii) $v_{i_1} + v_{i_2} \leq 1$ for the other one. Problem (5) in the presence of branching constraints can be expressed as

$$\begin{array}{lll} \min & \displaystyle\sum_{i=1}^{n} (\|p_i - y_v\|^2 - \lambda_i) v_i & \\ s.t. & v_i + v_j \leq 1 & \text{for} \quad (i,j) \in I_1 \\ & v_i = v_j & \text{for} \quad (i,j) \in I_2 \\ & v_i \in \mathbb{B} & \text{for} \quad i = 1, \ldots, n \end{array} \tag{8}$$

where I_1, I_2 are the index sets of pairs of entities involved in constraints of form (i) and (ii), respectively.

Algorithm 1 is not able to solve problem (8), since optimal solutions may now be associated to index sets which do not correspond directly to a region in the plane. In fact, Proposition 1 is no longer valid in the presence of branching constraints. A very simple example consists of two points p_i, p_j whose discs of radius $\sqrt{\lambda_i}$ and $\sqrt{\lambda_j}$ do not intersect while a constraint states that points p_i and p_j must be together. In this case, none of the index sets S scanned by Algorithm 1 is able to provide a feasible solution to the problem.

Fortunately, Proposition 3 below shows that Algorithm 1 can be slightly modified in order to solve problem (8) exactly. Let us first define three index sets associated with any vector y_v

- $S_1(y_v)$ is the index set of points p_i for which $\|p_i - y_v\|^2 \leq \lambda_i$, and for which $(i,j) \in I_1$ or $(i,j) \in I_2$ with $j \in S_1(y_v) \cup S_2(y_v)$;
- $S_2(y_v)$ is the index set of points p_i for which $\|p_i - y_v\|^2 > \lambda_i$, and for which $(i,j) \in I_1$ with $j \in S_1(y_v)$;
- $S_3(y_v)$ is the index set of points p_i for which $\|p_i - y_v\|^2 \leq \lambda_i$, and such that $i \notin S_1(y_v)$.

Proposition 3 *Let (y_v^*, v^*) be the optimal solution of (8) and let $\bar{v}^* = (v_i \mid i \in S_1(y_v^*) \cup S_2(y_v^*))$. Then, $(y_v^*, \bar{v}^*)$ is the optimal solution of a subproblem given by*

$$
\begin{array}{lll}
\min & \displaystyle\sum_{i \in S_1 \cup S_2} \|p_i - y\|^2 v_i + \sum_{i \in S_3} \|p_i - y\|^2 & \\
s.t. & \|p_i - y\|^2 v_i \leq \lambda_i & \forall i \in S_1 \\
& \|p_i - y\|^2 \leq \lambda_i & \forall i \in S_3 \\
& v_i \in \mathbb{B} & \forall \in S_1 \cup S_2 \\
& v \in X & \\
& y \in \mathbb{R}^s &
\end{array}
\tag{9}
$$

with sets $S_1, S_2, S_3 \subseteq \{1, \ldots, n\}$ and where X is the polyhedron of branching constraints.

Proof From the definition of $S_1(y_v^*)$, $S_2(y_v^*)$ and $S_3(y_v^*)$, $\|p_i - y_v^*\| > \lambda_i$ for all $i \notin S_1(y_v^*) \cup S_2(y_v^*) \cup S_3(y_v^*)$.

Now let $(y_v', \bar{v}')$ be the optimal solution to (9) regarding $S_1 = S_1(y_v^*)$, $S_2 = S_2(y_v^*)$ and $S_3 = S_3(y_v^*)$, and suppose that the optimal solution of (8) $(y_v^*, \bar{v}^*)$ is not optimal for (9). Then, we can construct v' as:

- $v_i' = \bar{v}_i', \forall i \in S_1 \cup S_2$;
- $v_i' = 1, \forall i \in S_3$;
- $v_i' = 0$, otherwise;

such that the cost of (y_v', v') is smaller than that of (y_v^*, v^*) in (8), which is a contradiction. □

The importance of Proposition 3 lies in the fact that, given the optimal y_v^*, the optimal subproblem of type (9) with sets $S_1 = S_1(y_v^*), S_2 = S_2(y_v^*)$ and $S_3 = S_3(y_v^*)$ is by definition associated to the region in the plane originated from the intersection of discs $\|p_i - y_v^*\|^2 \leq \lambda_i$. This fact implies that the number of subproblems of type (9) which need to be considered in order to solve (8)

is polynomially bounded. However, (9) is a problem with binary variables for which an enumeration method of resolution is needed.

Algorithm 1 can be modified to solve subproblems of type (9). For each region in the plane, sets S_1, S_2 and S_3 are determined to form a subproblem of type (9) (remark that any location y in a given region of the plane defines the same sets $S_1(y)$, $S_2(y)$ and $S_3(y)$). Then, the subproblem is solved by a branch-and-bound procedure. Note that whenever $S_1, S_2 = \emptyset$, subproblem (9) turns out to be equivalent to subproblem (7), and therefore, enumeration is not needed.

Decisions in the branch-and-bound algorithm are made by presence-absence dichotomy on variables v_i, for $\forall i \in S_1 \cup S_2$. Lower bounds are calculated in each node as the difference of two values:

1. the cost of the node solution, which is calculated with respect to the centroid of points p_i for which decision $v_i = 1$ is fixed;
2. the sum of the prices λ_i of the free variables v_i.

When (8) contains a few branching constraints, sets S_1 and S_2 have small cardinality by definition. So, the given branch-and-bound method to solve (9) performs very well in practice. In the presence of a larger number of branching rules, solving (9) becomes a more difficult task. To this purpose, we remark that (9) can be reformulated exactly (in the sense of [39]) by introducing parameters:

$$M_i \geq \max_j \|p_i - p_j\|^2 \quad \forall i \in S_1 \cup S_2,$$

decision variables:

$$\omega_i \in [0, M_i] \quad \forall i \in S_1 \cup S_2,$$

and constraints:

$$\|p_i - y\|^2 \leq \omega_i + (1 - v_i)M_i \quad \forall i \in S_1 \cup S_2$$

to (9). We then replace constraints $\|p_i - y\|^2 v_i \leq \lambda_i \ \forall i \in S_1$ by

$$\|p_i - y\|^2 \leq \lambda_i + (1 - v_i)M_i \quad \forall i \in S_1,$$

and the terms $\|p_i - y\|^2 v_i$ for $i \in S_1 \cup S_2$ in the objective function by ω_i. We thus obtain the reformulated problem:

$$\begin{array}{lll}
\min & \displaystyle\sum_{i \in S_1 \cup S_2} \omega_i + \sum_{i \in S_3} \|p_i - y\|^2 & \\
s.t. & \|p_i - y\|^2 \leq \lambda_i + (1 - v_i)M_i & \forall i \in S_1 \\
& \|p_i - y\|^2 \leq \omega_i + (1 - v_i)M_i & \forall i \in S_1 \cup S_2 \\
& \|p_i - y\|^2 \leq \lambda_i & \forall i \in S_3 \\
& v_i \in \mathbb{B} & \forall i \in S_1 \cup S_2 \\
& v \in X & \\
& y \in \mathbb{R}^s & \\
& \omega_i \in [0, M_i] & \forall i \in S_1 \cup S_2
\end{array} \tag{10}$$

which is a convex MINLP, for which there exist practically efficient algorithms (e.g. [8,38]). We also remark that its continuous relaxation is a continuous NLP which can be solved in polynomial time [63].

Finally, note that Algorithm 1 can be used without modifications to provide approximate solutions to (8). This can be done up to the moment that the exact resolution of (8) is required to prove that (3) was in fact optimally solved.

4 Generalization to the Euclidean space

Let us consider a graph $G = (N, E)$ for which there is a node $n_i \in N$ corresponding to each point p_i, for $i = 1, \ldots, n$. Besides, an edge e_{ij} exists in G if and only if

$$\|p_i - p_j\| \leq \sqrt{\lambda_i} + \sqrt{\lambda_j},$$

i.e., $e_{ij} \in E$ if and only if the hyperspheres G centered at p_i and p_j with radius $\sqrt{\lambda_i}$ and $\sqrt{\lambda_j}$ intersect.

The following result allows us to generalize the geometric approach in the plane by considering the intersection graph of hyperspheres centered at the points p_i, for $i = 1, \ldots, n$.

Proposition 4 *If a solution (y_v^*, v^*) is optimal to (5) then the elements of the set $N^* = \{n_i | v_i^* = 1\}$ form a clique in G.*

Proof Let us suppose that (y_v^*, v^*) is the optimal solution of (5) and that the elements of N^* do not form a clique in G. Hence, there are two nodes n_i, n_j in N^* for which $e_{ij} \notin E$, i.e., the hyperspheres centered at p_i and p_j with radius $\sqrt{\lambda_i}$ and $\sqrt{\lambda_j}$ do not intersect. In such a case, y_v^* is certainly located outside at least one of these hyperspheres. Suppose $\|p_i - y_v^*\| > \sqrt{\lambda_i}$, then a reduction in the cost of the solution is obtained by setting $v_i^* = 0$, which contradicts the optimality of (y_v^*, v^*). □

The number of distinct regions resulting from the intersection of hyperspheres is not polynomially bounded in n only. However, Proposition 4 allows to better exploit (6) above. Indeed it can be written as

$$\sigma + \min_{v_i \in \{0,1\}} \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (d_{ij}^2 - \lambda_i - \lambda_j) v_i v_j - \sum_{i=1}^{n} \lambda_i v_i}{\sum_{i=1}^{n} v_i},$$

where d_{ij} represents the Euclidean distance between the entities associated to variables v_i and v_j. Coefficients $d_{ij}^2 - \lambda_i - \lambda_j$ of the product $v_i v_j$ can be made arbitrarily large in (6) if $d_{ij} > \sqrt{\lambda_i} + \sqrt{\lambda_j}$ due to Preposition 4, since $v_i = v_j = 1$ does not occur in the optimal solution.

4.1 Branching

As proposed in [15], branching constraints of type $v_i = v_j$ can be added to the auxiliary problem (6) by reducing by one the number of its variables and updating coefficients accordingly. In the case of branching constraints of type $v_i + v_j \leq 1$, it suffices to set coefficient $d_{ij}^2 - \lambda_i - \lambda_j$ to an arbitrary large value. Thus, the auxiliary problem is expressed by

$$\sigma + \min_{v_{i'} \in \{0,1\}} \frac{\sum_{i'=1}^{n'-1} \sum_{j'=i'+1}^{n'} (d_{i'j'}^2 - w_{j'}\lambda_{i'} - w_{i'}\lambda_{j'}) v_{i'} v_{j'} - \sum_{i'=1}^{n'} (w_{i'}\lambda_{i'} - d_{i'i'}^2) v_{i'}}{\sum_{i'=1}^{n'} w_{i'} v_{i'}}, \tag{11}$$

where $w_{i'}$ is the number of variables merged in variable $v_{i'}$. Note that the form of the auxiliary problem is not changed. It is still a fractional program in 0-1 variables with quadratic numerator and linear denominator.

An observation must be made when setting coefficients based on the intersection graph of hyperspheres in the presence of branching constraints of type $v_i = v_j$. Suppose entities o_i and o_j for which there is a constraint stating that $v_i = v_j$. Consequently, variables v_i and v_j are merged together in a single variable $v_{i'}$ of (11). Let us consider now $v_{k'}$ the variable associated to entity o_k, then coefficient $d_{i'k'}^2 - \lambda_{i'} - 2\lambda_{k'}$ is set to an arbitrary large value in (11) only if

$$d_{ik} > \sqrt{\lambda_i} + \sqrt{\lambda_k} \qquad \text{and} \qquad d_{jk} > \sqrt{\lambda_j} + \sqrt{\lambda_k},$$

i.e., only if

$$d_{i'k} > \sqrt{\lambda_{i'}} + 2\sqrt{\lambda_{k'}}.$$

This can be generalized to any pair of variables $v_{i'}, v_{j'}$ in the follwong manner. If

$$d_{i'j'} > w_{j'}\sqrt{\lambda_{i'}} + w_{i'}\sqrt{\lambda_{j'}}$$

then $d_{i'j'}^2 - w_{j'}\lambda_{i'} - w_{i'}\lambda_{j'}$ can be set to an arbitrary large value in (11).

4.2 Solving by cliques

Moreover, Proposition 4 permits to exactly solve the auxiliary problem by directly searching for cliques in G. Algorithm 2 presents the steps to compute the optimal solution to (11) from the intersection graph of hyperspheres $G = (N, E)$.

Algorithm 2

1. While G is not empty

(a) Find a vertex n_i with smallest degree in G.
(b) Consider $G^i = (N^i, E^i)$ the subgraph composed by n_i and its adjacent vertices.
(c) Solve (11) for variables v_ℓ such that $n_\ell \in G^i$.
(d) Save the clique obtained if it is the best found so far.
(e) Remove n_i and its adjacent edges from G.

2. Return the best clique found.

Clearly, Algorithm 2 is more efficient for sparse graphs G than for dense ones as subproblems (11) solved in (c) tend to have less variables Indeed, the sparsity of G depends on the dual values λ, which tends to decrease with the number of clusters. This is due to the fact that when k is large, entities are likely to be close to their second-closest centroids in the optimal solution. Consequently, a second copy of an entity has little impact on the objective function value which means that the values λ of the dual variables are small.

5 Computational results

Computational experiments were performed on a AMD 64 bits platform with a 2 GHz clock and 10 Gigabytes of RAM memory. The algorithms were implemented in C++ and compiled by gcc 3.4. Unconstrained 0-1 quadratic programs are solved by the algorithm proposed in [23] which was observed to perform better than CPLEX 10.1. Eleven real-world data sets were used in our numerical experiments. They are briefly listed in Table 1 together with references to where more information about them can be found.

Table 1 List of data sets

Data sets	n	s
Ruspini's data [53]	75	2
Grötschel and Holland's 202 cities coordinates [21]	202	2
Grötschel and Holland's 666 cities coordinates [21]	666	2
Reinelt's hole-drilling data [52]	1060	2
Padberg and Rinaldi's hole-drilling data [49]	2392	2
Fisher's Iris [5]	150	4
Glass identification [5]	214	9
Body measurements[1] [29]	507	5
Indian Telugu vowel sounds [50]	871	3
Concrete compressive strength [5,66]	1030	8
Image segmentation [5]	2310	19

[1] the attributes used are: weight, height, chest girth, waist girth and hip girth

For all experiments reported here, initial solutions are obtained by j-means [24]. They are used to add initial cuts to model (4) as well as to estimate initial dual bounds (c.f. [15]) which may be adjusted throughout execution if necessary.

5.1 Results in the plane

In this subsection we compare the column generation of [15], denoted accpm-vns-qp, with two improved ones, i.e., (i) accpm-a1 which uses Algorithm 1 to exactly solve all auxiliary problems, and (ii) which uses one iteration of heuristic VNS (which reachs the largest neighboord once) to provide approximate solutions to auxiliary problems until optimality must be proved by Algorithm 1. Note that it is not worthwhile to use VNS for many iterations since Algorithm 1 is polynomially bounded in $O(n^3)$.

The results are also compared to those of two other methods proposed in the literature, i.e., the repetitive branch-and-bound algorithm (rbba) of Brusco [9] and the best branch-and-cut SDP-based algorithm (bb-sdp) of [2].

Tables 2–7 show results for data sets in the plane. They present in the first column the number k of clusters, and optimal solution values f_{opt} are reported in the second column. The values associated to each algorithm refer to their respective CPU times (in seconds) spent on solving exactly the instance. Finally, a last column is included to present gap values between upper and lower bounds obtained at the root node, denoted UB^0 and LB^0 respectively, which are calculated as $(UB^0 - LB^0)/LB^0$. The letter 'i' indicates that no initial gap exists, i.e., the problem is already solved by the accpm algorithms at the root node, without branching. Otherwise, the number of branch-and-bound nodes is given in parenthesis.

Table 2 shows that all methods perform well or very well for Ruspini's data set with $n = 75$ entities. Algorithm rbba is particularly efficient for small values of k, while its performance quickly deteriorates as k increases. This is due to the fact that the number of branchs in RBBA is $O(k^n)$. For $k \geq 5$, algorithms accpm-a1 and accpm-vns-a1 are always faster than the other methods.

Table 2 Results for Ruspini data set with 75 entities

k	f_{opt}	rbba	bb-sdp	accpm-vns-qp	accpm-vns-a1	accpm-a1	$gap(\%)$
2	0.893378e+05	0.01	3.56	0.55	0.24	0.39	i
3	0.510634e+05	0.28	8.34	0.57	0.20	0.42	i
4	0.128810e+05	0.01	0.48	0.53	0.14	0.07	i
5	0.101267e+05	0.17	0.57	0.59	0.16	0.10	i
6	0.857541e+04	21.97	1.03	0.91	0.27	0.18	i
7	0.712620e+04	181.90	0.98	1.12	0.28	0.18	i
8	0.614964e+04	2921.93	7.27	1.04	0.44	0.23	0.01(3)
9	0.518165e+04	$>$ 1h	2.87	1.20	0.30	0.17	i
10	0.444628e+04	$>$ 1h	2.39	1.17	0.26	0.12	i

Table 3 presents results obtained in less than 12 hours of CPU time for the Grötschel and Holand's data set with $n = 202$. Algorithm rbba is not able to solve even the problem with $k = 2$ clusters in less than 12 hours. So, we do not refer to its results in the subsequent tables. As empirically observed in [2], the performance of algorithm bb-sdp deteriorates as k decreases, in contrast with

algorithm rbba. It is unable to solve problems for $k \leq 8$ in less than 12 hours. It also appears that it is better to approximately solve the auxiliary problems by VNS up to $k = 15$. For $k \geq 20$, the sparsity of the discs in the plane, which is implied by small dual values, makes Algorithm 1 more efficient than VNS to solve the auxiliary problems. So, algorithm accpm-a1 performs better than accpm-vns-a1 for these values of k. The sparsity effect also appears to be advantageous to the unconstrained 0-1 quadratic programming solver since the algorithm is faster for instances with larger number of clusters.

Table 3 Results for Grötschel and Holland's data set with 202 entities

k	f_{opt}	bb-sdp	accpm-vns-qp	accpm-vns-a1	accpm-a1	$gap(\%)$
2	0.234374e+05	> 12h	> 12h	19.85	61.54	i
3	0.153274e+05	> 12h	> 12h	19.64	79.65	i
4	0.114556e+05	> 12h	> 12h	21.87	82.89	i
5	0.889490e+04	> 12h	> 12h	15.62	63.95	i
6	0.676488e+04	> 12h	> 12h	26.33	69.97	i
7	0.581757e+04	> 12h	> 12h	33.79	85.56	i
8	0.500610e+04	> 12h	1526.63	48.80	65.56	i
9	0.437619e+04	48885.38	1334.06	33.79	47.87	i
10	0.379249e+04	23680.84	496.85	16.42	35.84	i
15	0.232008e+04	39756.23	41.49	18.43	30.71	i
20	0.152351e+04	3839.77	59.90	18.87	17.75	i
25	0.108556e+04	1915.05	33.95	18.24	11.05	i
30	0.799311e+03	1060.77	27.03	17.78	5.96	i

Regarding the results for the Grötschel and Holand's data set with $n = 666$ entities presented in Table 4, a CPU time limit of 1 day was established, which proved not to be enough for algorithms bb-sdp and accpm-vns-qp. Therefore, the results of these algorithms will not be reported from now on since they demand too much time to exactly solve instances of the largest data sets. Table 4 shows that algorithm accpm-a1 is faster than accpm-vns-a1 from $k \geq 4$.

Table 4 Results for Grötschel and Holland's data set with 666 entities

k	f_{opt}	accpm-vns-a1	accpm-a1	$gap(\%)$
2	$1.754012e+06$	1179.68	2723.48	i
3	$0.772707e+06$	1525.10	1758.92	i
4	$0.613995e+06$	3585.39	3290.45	i
5	$0.485088e+06$	3277.55	2410.83	i
6	$0.382676e+06$	3162.39	1909.23	i
7	$0.323283e+06$	3082.65	1909.49	i
8	$0.285925e+06$	4314.00	2469.90	i
9	$0.250989e+06$	4134.31	2162.06	i
10	$0.224183e+06$	3131.41	2108.38	i
20	$0.106276e+06$	10504.30	4819.84	0.00(3)
50	$0.351795e+05$	6161.84	447.48	i

The results in Table 5 show that accpm-a1 is faster than accpm-vns-a1 from $k \geq 7$. The algorithms appear to be scalable for larger values of k due to increasing sparsity of discs in the auxiliary problems. It is worthwhile to mention that some of the state-of-art heuristics proposed in [11,24,36,37,47, 59] did not report the optimal solutions found here for the Reinelt's drilling data set with $n = 1060$ entities and $k = 120, 150$. To the best of our knowledge, this is the first time that such solutions are reported in the literature.

Table 5 Results for Reinelt's drilling data set with 1060 entities

k	f_{opt}	accpm-vns-a1	accpm-a1	$gap(\%)$
2	$0.983195e+10$	7417.92	13657.78	i
3	$0.670578e+10$	17897.19	30016.73	i
4	$0.475197e+10$	13429.61	26921.27	i
5	$0.379100e+10$	15966.45	26049.23	i
6	$0.317701e+10$	15128.71	19970.91	i
7	$0.270386e+10$	39966.71	22289.93	i
8	$0.226315e+10$	24863.21	19942.57	i
9	$0.198104e+10$	21810.90	16438.40	i
10	$0.175484e+10$	349793.97	56625.07	0.01(3)
100	$0.963178e+08$	17017.10	496.85	i
110	$0.848396e+08$	14930.74	373.54	i
120	$0.755366e+08$	8165.25	393.21	i
130	$0.675542e+08$	8296.29	301.77	i
140	$0.611196e+08$	13886.32	299.75	i
150	$0.559082e+08$	4998.90	292.37	i
200	$0.361572e+08$	4234.54	229.74	i

Finally, algorithms accpm-vns-a1 and accpm-a1 were tested for Padberg and Rinaldi's data set with $n = 2392$ entities. From the geometric interpretation of the auxiliary problem corroborated by the results presented in the previous tables, we concluded that algorithm accpm-vns-a1 is more efficient for instances with small number of clusters. Therefore, Table 6 presents only the results of accpm-vns-a1 for $2 \leq k \leq 10$. Note that these instances require a lot of computing time to be exactly solved (e.g. more than one week was necessary to solve the instance with $k = 9$).

Table 6 Results for Padberg and Rinaldi's data set with 2392 entities for $2 \leq k \leq 10$

k	f_{opt}	accpm-vns-a1	$gap(\%)$
2	$0.296723e+11$	180581.30	i
3	$0.212012e+11$	393564.16	i
4	$0.141184e+11$	298724.00	i
5	$0.115842e+11$	416314.64	i
6	$0.948900e+10$	218403.68	i
7	$0.818180e+10$	565361.77	i
8	$0.701338e+10$	482525.96	i
9	$0.614600e+10$	663595.15	i
10	$0.532491e+10$	478613.29	i

Table 7 presents the results obtained by algorithm accpm-a1 for the Padberg and Rinaldi's data set with $n = 2392$ entities using large values of k. For these instances, approximately 3-5% of the total computing time is spent solving the auxiliary problems, revealing that at this point (≈ 2000 entities) the resolution of the restricted master problem by ACCPM is the most expensive step of the algorithm. Note that the largest CPU time reported in Table 7 is of approximately 29 hours for $k = 150$.

Table 7 Results for Padberg and Rinaldi's data set with 2392 entities for large values of k

k	f_{opt}	accpm-a1	$gap(\%)$
100	$0.404498e+09$	21528.56	i
150	$0.245685e+09$	106160.43	0.01(7)
200	$0.175431e+09$	18918.16	i
250	$0.132352e+09$	16460.46	i
300	$0.101568e+09$	35939.04	0.00(3)
350	$0.804783e+08$	8131.32	i
400	$0.657989e+08$	9336.05	i

5.2 Results in general Euclidean space

Two other algorithms were implemented in order to check the computational effect of the geometric arguments in general Euclidean space. They are: (i) accpm-vns-qp+, which is similar to accpm-vns-qp proposed in [15] except that some coefficients are modified to arbitrarily large values in the auxiliary problem following the geometrical arguments presented in Section 4, and (ii) accpm-vns-a2, which uses one iteration of VNS to obtain approximate solutions to auxiliary problems until optimality is certified by Algorithm 2.

Table 8 shows CPU times spent by the different algorithms in order to solve exactly instances of the Fisher's Iris data set with $n = 150$ entities in $s = 4$ dimensions. The results shows that again rbba is very efficient for small number of clusters, though its performance deteriorates very fast as k increases. Moreover, except for $k = 2$, algorithm accpm-vns-qp+ performs better than accpm-vns-qp. Finally, since the auxiliary problems are small for this data set ($n = 150$), Algorithm 2 is not very advantageous for solving them. In fact, for the instance with $k = 2$, algorithm accpm-vns-a2 is much less efficient than the others.

The results in Table 9 give CPU times spent on solving exactly instances of the Glass identification data set with $n = 214$ in $s = 9$ dimensions. We notice that instances with $k \leq 10$ cannot be solved in less than 1 day of computation. In particular, algorithm rbba takes more than 1 day to solve even its most favourable case with $k = 2$. Therefore, the next tables will not refer to its results. Likewise, results of algorithm bb-sdp will not be reported in the following tables since it is clearly outperformed by ACCPM algorithms.

Table 8 Results for Fisher's Iris with 150 entities in 4 dimensions

k	f_{opt}	rbba	bb-sdp	accpm-vns-qp	accpm-vns-qp+	accpm-vns-a2	$gap(\%)$
2	0.152348e+03	0.05	169.44	251.04	486.62	1958.06	i
3	0.788514e+02	2.10	283.24	83.09	19.88	19.55	i
4	0.572285e+02	136.29	240.19	138.85	32.71	17.22	i
5	0.464462e+02	1699.75	145.54	42.00	6.52	8.80	i
6	0.390400e+02	$>$ 12h	147.51	15.50	11.70	10.47	i
7	0.342982e+02	$>$ 12h	742.83	10.50	7.83	6.65	i
8	0.299889e+02	$>$ 12h	108.73	7.82	6.41	6.74	i
9	0.277861e+02	$>$ 12h	70.04	6.44	6.11	7.48	i
10	0.25834e+02	$>$ 12h	59.66	8.51	8.38	9.03	i

Table 9 Results for the Glass identification data set with 214 entities in 9 dimensions

k	f_{opt}	bb-sdp	accpm-vns-qp	accpm-vns-qp+	accpm-vns-a2	$gap(\%)$
15	0.155766e+03	$>$ 1 day	$>$ 1 day	37714.82	7983.52	i
20	0.114646e+03	$>$ 1 day	$>$ 1 day	30065.43	13365.79	0.02(3)
25	0.842515e+02	$>$ 1 day	$>$ 1 day	24568.26	19011.65	0.00(3)
30	0.632478e+02	49831.18	269.36	52.80	39.50	i
35	0.492386e+02	25629.86	22.60	16.33	18.87	i
40	0.394983e+02	6272.84	27.87	16.85	18.32	i
45	0.320395e+02	17437.27	43.27	29.37	32.21	0.00(3)
50	0.267675e+02	10032.09	21.69	20.51	21.46	i

From the results om Table 9, algorithm accpm-vns-qp+ outperforms accpm-vns-qp in all tested instances. Since this is also true for the computational experiments on the other data sets, we will not report the results of accpm-vns-qp from now on. This fact confirms the benefits derived from the geometric interpretation of the auxiliary problem. Moreover, algorithm accpm-vns-a2 was more efficient than accpm-vns-qp+ for the instances with the most difficult auxiliary problems (i.e., $15 \leq k \leq 30$), showing that solving (11) by isolating cliques is a good strategy in these cases.

Taking into account the increasing computing times spent by VNS as the value of n increases, one may ask if it would not be better to solve exactly the auxilary problems at each iteration of ACCPM. In order to answer this question, two other algorithms are considered for comparison in Tables 10, 11, 12. They differ only in the way that auxiliary problems are dealt with. While accpm-qp+ always uses Dinkelbach's algorithm to solve the auxiliary problems, accpm-a2 uses Algorithm 2 instead, i.e., using Dinkelbach's algorithm on each clique.

From Table 10, we notice that the algorithms that solve auxiliary problems by cliques (i.e., accpm-vns-a2 and accpm-a2) perform usually better than their counterparts that solve the auxiliary problems by considering the whole intersection graph of hyperspheres (accpm-vns-qp+ and accpm-qp+, respectively). In particular accpm-a2 is the best algorithm from $k \geq 60$. The same conclusions can be extended to Tables 11 and 12, except that for these larger data

Table 10 Results for the Body measurements data set with 507 entities in 5 dimensions

k	f_{opt}	accpm-vns-qp+	accpm-qp+	accpm-vns-a2	accpm-a2	$gap(\%)$
30	0.195299e+05	79819.81	> 2 days	12433.74	> 2 days	0.00(3)
40	0.162318e+05	3981.92	25196.16	3954.62	13396.05	0.00(3)
50	0.139547e+05	26991.10	> 2 days	22945.66	67178.35	0.04(11)
60	0.121826e+05	2847.94	3284.43	2242.53	1860.72	0.00(3)
70	0.107869e+05	2606.16	2421.93	2534.06	1329.71	0.00(3)
80	0.964873e+04	5565.30	5026.03	6191.68	2705.14	0.01(5)

sets accpm-a2 is very often the best algorithm for the instances that can be exaclty solved within a CPU time limit of 2 days.

Table 11 Results for the Indian Telugu vowel sounds data set with 871 entities in 3 dimensions

k	f_{opt}	accpm-vns-qp+	accpm-qp+	accpm-vns-a2	accpm-a2	$gap(\%)$
40	0.636653e+07			10232.80	8209.48	i
50	0.524020e+07		14304.07	4314.11	2450.54	i
60						
70	0.375286e+07	7439.66		6524.48	1726.57	0.00(3)
80	0.324801e+07	2538.37	2320.29	2389.09	323.95	i
90	0.285069e+07	2227.94	1929.68	1980.14	282.73	i
100	0.251058e+07	5717.78	1606.62	5054.39	195.53	0.00(3)

Table 12 Results for the Concrete compressive strength data set with 1030 entities in 9 dimensions

k	f_{opt}	accpm-vns-qp+	accpm-qp+	accpm-vns-a2	accpm-a2	$gap(\%)$
50						
60	0.288107e+07	> 2 days	> 2 days	93018.98	114291.96	i
70	0.247893e+07	32524.80	33373.40	8671.61	2825.70	i
80	0.215791e+07	5622.55	7538.82	5717.15	1405.62	i
90						
100	0.168778e+07	3330.97	3530.60	3773.60	380.75	i

We have still obtained results for a larger data set consisting of 2310 entities in 19 dimensions taken from [5] by means of algorithm accpm-a2. The results presented in Table 13 shows that instances with a ratio of $n/k \approx 10$ can be exactly solved in a reasonable amount of time by the column generation algorithm, which is a new record for benchmark data sets of this magnitude ($n = 2310$) and this dimension ($s = 19$).

Table 13 Results for the Image segmentation data set with 2310 entities in 19 dimensions

k	f_{opt}	accpm-a2	$gap(\%)$
230			
250	0.421018e+06	10864.30	i
300	0.338072e+06	25693.02	0.00(3)
350	0.276957e+06	7036.09	i
400	0.230310e+06	99554.55	0.00(11)
450	0.195101e+06	66655.32	0.00(7)
500	0.157153e+06	36772.86	0.01(5)

5.3 Comparison of approaches in the plane and in general Euclidean space

Finally, we compare our approach in the plane with that tailored for problems in general Euclidean space. Since the superiority of the approach in the plane for a small number of clusters is obvious, we decided to focus this comparison on instances with large values of k. The best algorithm regarding each one of the approaches is then selected for comparison, i.e., accpm-a1 from the class of algorithms which tackles exclusively instances in the plane and accpm-a2 from the class of algorithm dealing with instances in general Euclidean space.

In the graph of Figure 2, we plot the percentage of CPU time spent by algorithm accpm-a2 in excess of the CPU time spent by algorithm accpm-a1 when solving different instances of the Reinelt's planar data set with 1060 entities.

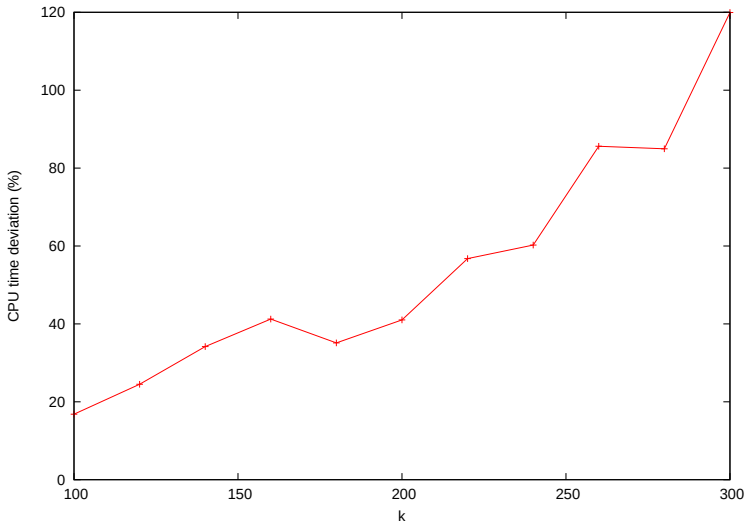

Fig. 2 Percentage of CPU time spent by algorithm accpm-a2 in excess of the CPU time spent by algorithm accpm-a1 for instances of the Reinelt's planar data set with 1060 entities

From the graph, we notice that accpm-a1 tends to be increasingly better than accpm-a2 as k augments, though the computing times are smaller for instances with a large number of clusters.

6 Conclusions

MSSC is a central problem in cluster analysis. Numerous heuristics as well as a variety of exact algorithms have been proposed for its solution. These last ones include the column generation algorithm of du Merle et al. [15] which is the point of departure of this paper. The bottleneck step of that algorithm appeared within the auxiliary problem and was the solution of unconstrained 0-1 quadratic programs. Based on geometric reasoning, a different and more efficient way of solving this auxiliary problem is proposed in this paper. It exploits systematically the property that far apart points will not belong to the same cluster. This property is made precise by proving that it is the case when their mutual distance exceeds the sum of square roots of the corresponding dual variables at the current iteration. Geometrically, solutions in the plane correspond to a quadratic number of regions which are determined by a $O(n^2)$ algorithm. This leads to solution of the auxiliary problem in $O(n^3)$, at least when there is little branching in the master problem which appears to be most often the case. Finding all similar regions in a higher dimensional space would be time consuming. However, the way to solve the auxiliary problem can still be improved by replacing by a large value coefficients in the unconstrained 0-1 quadratic programs corresponding to far apart entities. This has led to substantially increase the size of instances solved exactly. In the plane, instances with n up to 2392 entities and $k \geq 2$ have been solved exactly most of them for the first time. The increase in the size of the instances exactly solved has thus been multiplied by more than 10. In general Euclidean space problems with up to $n = 2310$ and $k = 250$ clusters in 19 dimensions have been solved. However, it appears that the number of entities per cluster should be small, i.e. n/k roughly equal to 10, in order to solve such instances in reasonable time.

References

1. D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning*, 75:245–249, 2009.
2. D. Aloise and P. Hansen. A branch-and-cut SDP-based algorithm algorithm for minimum sum-of-squares clustering. *Les Cahiers du GERAD*, G-2008-40, 2008.
3. D. Aloise and P. Hansen. Evaluating a branch-and-bound RLT-based algorithm for minimum sum-of-squares clustering. *to appear in Journal of Global Optimization*, 2009.
4. L.T. An, M.T. Belghiti, and P.D. Tao. A new efficient algorithm based on DC programming and DCA for clustering. *Journal of Global Optimization*, 37:593–608, 2007.
5. A. Asuncion and D.J. Newman. UCI machine learning repository. 2007. http://www.ics.uci.edu/∼mlearn/MLRepository.html.
6. A.M. Bagirov. Modified global k-means algorithm for minimum sum-of-squares clustering problems. *Pattern Recognition*, 41:3192–3199, 2008.

7. A.M. Bagirov and J. Yearwoord. Hierarchical grouping to optimize an objective function. *European Journal of Operational Research*, 170:578–596, 2006.
8. P. Bonami and J. Lee. `BONMIN` user's manual. Technical report, IBM Corporation, June 2007.
9. M.J. Brusco. A repetitive branch-and-bound procedure for minimum within-cluster sum of squares partitioning. *Psychometrika*, 71:347–363, 2006.
10. M.J. Brusco and D. Steinley. A comparison of heuristics procedures for minimum within-cluster sums of squares partitioning. *Psychometrika*, 72:583–600, 2007.
11. I.T. Christou. Exact method-based coordination of cluster ensembles. *Athens Information Technology Technical Report 2055; Provisionally accepted with major revisions in IEEE Transactions in Pattern Analysis and Machine Intelligence*, 2009.
12. G. Diehr. Evaluation of a branch and bound algorithm for clustering. *SIAM Journal Scientific and Statistical Computing*, 6:268–284, 1985.
13. W. Dinkelbach. On nonlinear fractional programming. *Management Science*, 13:492–498, 1967.
14. Z. Drezner, A. Mehrez, and G.O. Wesolowsky. The facility location problem with limited distances. *Transportation Science*, 25:183–187, 1991.
15. O. du Merle, P. Hansen, B. Jaumard, and N. Mladenović. An interior point algorithm for minimum sum-of-squares clustering. *SIAM Journal Scientific Computing*, 21:1485–1505, 2000.
16. O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.
17. A.W. Edwards and L.L. Cavalli-Sforza. A method for cluster analysis. *Biometrics*, 21:362–375, 1965.
18. S. Elhedhli and J.-L. Goffin. The integration of an interior-point cutting plane method within a branch-and-price algorithm. *Mathematical Programming*, 100:267–294, 2004.
19. R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, VII:179–188, 1936.
20. J.-L. Goffin, A. Haurie, and J.-P.Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, 38:284–302, 1992.
21. M. Grötschel and O. Holland. Solution of large-scale symmetric traveling salesman problems. *Mathematical Programming*, 51:141–202, 1991. Data sets available at [http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp].
22. P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming*, 79:191–215, 1997.
23. P. Hansen, B. Jaumard, and C. Meyer. A simple enumerative algorithm for unconstrained $0-1$ quadratic programming. Cahier du GERAD G-2000-59, GERAD, November 2000.
24. P. Hansen and N. Mladenović. J-means: a new local search heuristic for minimum sum of squares clustering. *Pattern Recognition*, 34:405–413, 2001.
25. P. Hansen and N. Mladenović. Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.
26. P. Hansen, N. Mladenović, and J.A.M. Pérez. Variable neighborhood search: methods and applications. *to appear in 4OR*, 2008.
27. P. Hansen, E. Negai, B.K. Cheung, and N. Mladenović. Analysis of global k-means, an incremental heuristic for minimum sum-of-squares clustering. *Journal of Classification*, 22:287–310, 2005.
28. J.A. Hartigan. *Clustering Algorithms*. Wiley, New York, 1975.
29. G. Heinz, L.J. Peterson, R.W. Johnson, and C.J. Kerk. Exploring relationships in body dimensions. *Journal of Statistics Education*, 11, 2003. Data set available at [www.amstat.org/publications/jse/v11n2/datasets.heinz.html].
30. M. Inaba, N. Katoh, and H. Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the 10th ACM Symposium on Computational Geometry*, pages 332–339, 1994.
31. A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31:264–323, 1999.
32. R.E. Jensen. A dynamic programming algorithm for cluster analysis. *Operations Research*, 17:1034–1057, 1969.

33. J.E. Kelley. The cutting plane method for solving convex programs. *J. SIAM*, 8:703–712, 1960.
34. J. Kogan. *Introduction to Clustering Large and High-Dimensional Data*. Cambridge University Press, New York, 2006.
35. W.L.G. Koontz, P.M. Narendra, and K. Fukunaga. A branch and bound clustering algorithm. *IEEE Trans. Comput.*, C-24:908–915, 1975.
36. M. Laszlo and S. Mukherjee. A genetic algorithm using hyper-quadtrees for low-dimensional k-means clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28:533–543, 2006.
37. M. Laszlo and S. Mukherjee. A genetic algorithm that exchanges neighboring centers for k-means clustering. *Pattern Recognition Letters*, 36:451–461, 2007.
38. S. Leyffer. User manual for MINLP_BB. Technical report, University of Dundee, UK, March 1999.
39. L. Liberti. Reformulations in mathematical programming: Definitions and systematics. *RAIRO-RO*, 43(1):55–86, 2009.
40. A. Likas, N. Vlassis, and J.J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36:451–461, 2003.
41. J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of* 5^{th} *Berkeley Symposium on Mathematical Statistics and Probability*, volume 2, pages 281–297, Berkeley, CA, 1967.
42. M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is NP-hard. *Lecture Notes in Computer Science*, 5431:274–285, 2009.
43. P. Merz. An iterated local search for minimum sum-of-squares clustering. *Lecture Notes in Computer Science*, 2810:286–296, 2003.
44. B. Mirkin. *Mathematical Classification and Clustering*. Kluwer, Dordrecht, The Netherlands, 1996.
45. B. Mirkin. *Clustering for Data Mining: A Data Recovery Approach*. Chapman and Hall/CRC, Boca Raton, 2005.
46. N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.
47. J.A. Pacheco. A scatter search approach for the minimum sum-of-squares clustering problem. *Computers & Operations Research*, 32:1325–1335, 2005.
48. J.A. Pacheco and O. Valencia. Design of hybrids for the minimum sum-of-squares clustering problem. *Computational Statistics & Data Analysis*, 43:235–248, 2003.
49. M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resoution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991. Data set available at [http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp].
50. S.K. Pal and D.D. Majumder. Fuzzy sets and decision making approaches in vowel and speaker recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 7:625–629, 1977. Data set available at [http://www.isical.ac.in/ sushmita/patterns/vowel.dat].
51. J. Peng and Y. Xia. A new theoretical framework for k-means-type clustering. *Studies in Fuzziness and Soft Computing*, 180:79–96, 2005.
52. G. Reinelt. TSPLIB – a traveling salesman library. *ORSA Journal on Computing*, 3:319–350, 1991. [www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95].
53. E.H. Ruspini. Numerical method for fuzzy clustering. *Information Sciences*, 2:319–350, 1970.
54. D.M. Ryan and B.A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269–280. North-Holland, 1981.
55. H.D. Sherali and W.P. Adams. Reformulation-linearization techniques for discrete optimization problems. In D.Z. Du and P.M. Pardalos, editors, *Handbook of combinatorial optimization 1*, pages 479–532. Kluwer, 1999.
56. H.D. Sherali and J. Desai. A global optimization RLT-based approach for solving the hard clustering problem. *Journal of Global Optimization*, 32:281–306, 2005.
57. H. Späth. *Cluster analysis algorithm for data reduction and classification of objects*. John Wiley & sons, New York, 1980.
58. D. Steinley. K-means clustering: A half-century synthesis. *British Journal of Mathematical and Statistical Psychology*, 59:1–34, 2006.

59. É.D. Taillard. Heuristic methods for large centroid clustering problems. *Journal of Heuristics*, 9:51–73, 2003.
60. M. Teboulle. A unified continuous optimization framework for center-based clustering methods. *Journal of Machine Learning Research*, 8:65–102, 2007.
61. H. Tuy. Concave programming under linear constraints. *Soviet Mathematics*, 5:1437–1440, 1964.
62. B.J. van Os and J.J. Meulman. Improving dynamic programming strategies for partitioning. *Journal of Classification*, 21:207–230, 2004.
63. S.A. Vavasis. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, Oxford, 1991.
64. A.E. Xavier, M.J. Negreiros, N. Maculan, and P. Michelon. The use of the hyperbolic smoothing clustering method for planning the tasks of sanitary agents in combating dengue. In *Proceedings of IFORS 2005*, 2005.
65. Y. Xia and J. Peng. A cutting algorithm for the minimum sum-of-squared error clustering. In *Proceedings of the SIAM International Data Mining Conference*, 2005.
66. I-C. Yeh. Modeling of strength of high performance concrete using artificial neural networks. *Cement and Concrete Research*, 28:1797–1808, 1998. Data set available at [http://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength].