Global Optimality in k-means Clustering

Cristina Tîrnăucă[a,*], Domingo Gómez-Pérez[a], José L Balcázar[b], José L Montaña[a]

[a] *Departamento de Matemáticas, Estadística y Computación, Universidad de Cantabria, Santander 39005, Spain*
[b] *Department of Computer Science, Universitat Politècnica de Catalunya, Barcelona 08034, Spain*

Abstract

We study the problem of finding an optimum clustering, a problem known to be NP-hard. Existing literature contains algorithms running in time proportional to the number of points raised to a power that depends on the dimensionality and on the number of clusters. Published validations of some of these algorithms are unfortunately incomplete; besides, the constant factors (with respect to the number of points) in their running time bounds have seen several published important improvements but are still huge, exponential on the dimension and on the number of clusters, making the corresponding algorithms fully impractical. We provide a new algorithm, with its corresponding complexity-theoretic analysis. It reduces both the exponent and the constant factor, to the extent that it becomes feasible for relevant particular cases. Additionally, it parallelizes extremely well, so that its implementation on current high-performance hardware is quite straightforward. Our proposal opens the door to potential improvements along a research line that had no practical significance so far; besides, a long but single-shot run of our algorithm allows one to identify absolutely optimum solutions for benchmark problems, whereby alternative heuristic proposals can evaluate the goodness of their solutions and the precise price paid for their faster running times.

Keywords: clustering, Voronoi diagrams, cell arrangement

1. Introduction

Assume we are given a finite set $S = \{\vec{p}_1, \ldots, \vec{p}_n\} \subseteq \mathbb{R}^d$, containing n observations, for a fixed dimensionality d. Assume also that an integer value k is given. We consider the problem that we will call "k-means globally optimum clustering": find k points, called "centroids", $\vec{q}_1, \ldots, \vec{q}_k$ that minimize the *within-cluster sum of squares* (WCSS) obtained by adding together the square of the Euclidian distance between each point $\vec{p}_i$ and its closest centroid.

An equivalent way of stating the problem is the following: S is to be partitioned into k disjoint subsets S_j called *clusters*, in such a way that the following expression is minimized:

$$cost(S_1, \ldots, S_k) = \sum_{j=1}^{k} \sum_{\vec{p} \in S_j} \|\vec{p} - \vec{q}_j\|^2, \text{ where } \vec{q}_j = \frac{1}{|S_j|} \sum_{\vec{p} \in S_j} \vec{p}.$$

One popular approach to this problem is the algorithm usually known as *k-means*, also called sometimes *Lloyd's heuristic* [1]. To our knowledge, it was first used in [2], and has become a standard algorithm in practice; implementations abound, and it is available in most major Data Mining software suites.

Lloyd's ubiquitous heuristic consists of selecting k initial centroids $\vec{q}_1, \ldots, \vec{q}_k$ according to some criterion (most often, randomly among the data points), constructing each S_j as the set of points $\vec{p}$ that are closer to $\vec{q}_j$ than to any other centroid (ties can be broken arbitrarily), recomputing the centroids as mass centers of the current S_j, and iterating until stability of the clusters. This heuristic is based upon the following known (and easy to prove) facts:

*Corresponding author
Email addresses: cristina.tirnauca@unican.es (Cristina Tîrnăucă), domingo.gomez@unican.es (Domingo Gómez-Pérez), jose.luis.balcazar@upc.edu (José L Balcázar), montanjl@unican.es (José L Montaña)

Fact 1. *For a fixed tuple of centroids $\vec{q}_1, \ldots, \vec{q}_k$, the clustering that minimizes the cost is obtained by including in cluster S_j the points $\vec{p}$ that are closer to $\vec{q}_j$ than to any other centroid; points that are at equal minimal distances from several centroids can be assigned to any of the corresponding clusters.*

Fact 2. *For a fixed clustering $S_1, \ldots, S_k$, among all possible tuples of centroids, the one that minimizes the WCSS is defined by:*

$$\vec{q}_j = \frac{1}{|S_j|} \sum_{\vec{p} \in S_j} \vec{p}, \quad \forall j \in \{1, \ldots, k\}$$

The first fact is clear; to argue the second, it suffices to make the partial derivatives equal to zero and solve the equations so obtained; the sign of the second derivative proves minimality.

Interesting studies of the goodness of the solutions obtained by this heuristic version of k-means, mostly in terms of the initialization criterion, are [3, 4, 5]. This is a very popular algorithm: often, we have far more observations than a human user can look at and make sense of. "Abstracting" them out into a handful of "representative" centroids, maybe with an indication of the cardinalities of the clusterings, is a way of understanding a bit of the inherent structure of the dataset. Note that, for this process to actually make sense, we wish a smallish k, and there is a fair number of available options in order to choose the right one (see [6] for one successful approach).

Most of the practical implementations of k-means offer no indication of the quality of the locally optimal solution found. How difficult is it to come up with the actual global solution which minimizes to optimality the expression above? A simple approach is based on working with partial clusterings of the data points, where single points are added sequentially to each of the clusters. This approach defines different uninformed search strategies depending on the order to explore the different partial clusterings. The book [7] contains a detailed exposition of several of these strategies. In Section 5 we compare the best of them (in terms of speed and memory requirements) with our proposal. Needless to say, these schemes are all exponential in the number of points since the problem is known to be NP-hard, see [8, 9]; it remains so for $k = 2$ (see [10]) and also for $d = 2$ (see [11]).

An interesting alternative was proposed in [12]: by plainly enumerating possible clusterings, evaluating them, and keeping the best seen so far, it is possible to find the global optimum in time $\mathcal{O}(n^{dk+1})$. This enumeration is made via a reduction to one particular geometric problem for which the literature contains, actually, several algorithms. Here we add one more step to this line of research, by identifying a new algorithm for which the running times, on mildly realistic problems, start to be feasible on current equipment. We report on related literature at the end of the next section, once the notation is established.

2. Preliminaries and Related Work

Let us describe, in a more formalized way, the approach in [12]; we start by formalizing Fact 1 in an algebraic form that will lead to the opportunity of employing ideas from algebraic geometry. One of the goals of the formalization is to clarify how to handle the ambiguous case of points that are equidistant from several closest centroids.

Given $S = \{\vec{p}_1, \ldots, \vec{p}_n\}$, each from $\mathbb{R}^d$, consider k candidates to centroids, $\{\vec{q}_1, \ldots, \vec{q}_k\}$, each from $\mathbb{R}^d$ as well. Taken together, they can be seen as a tuple of dk real numbers, so that we can refer to the whole tuple $\vec{q} = (\vec{q}_1, \ldots, \vec{q}_k) \in \mathbb{R}^{dk}$, with the understanding that vectors of this dimension are seen, whenever convenient, as tuples of k vectors of dimension d.

Definition 1. The *Voronoi partition* of S associated to $\vec{q} = (\vec{q}_1, \ldots, \vec{q}_k) \in \mathbb{R}^{dk}$ is $(S_1, \ldots, S_k)$ where:

- S_1 contains the data points that are at least as close to q_1 as to any other centroid: $\forall j \in \{2, \ldots, k\}$,

$$\|\vec{p} - \vec{q}_1\|^2 \leq \|\vec{p} - \vec{q}_j\|^2.$$

Thus, points that are at the same distance of q_1 and other closest centroids are won for the S_1 cluster.

- Likewise, S_2 wins equidistant points to all other clusters except S_1: this cluster is made of points such that, $\forall j \in \{3, \ldots, k\}$,

$$\|\vec{p} - \vec{q}_2\|^2 \leq \|\vec{p} - \vec{q}_j\|^2, \quad \|\vec{p} - \vec{q}_2\|^2 < \|\vec{p} - \vec{q}_1\|^2.$$

- In general,

$$\begin{aligned} S_i = \{\vec{p} \in S \mid \|\vec{p} - \vec{q}_i\|^2 \leq \|\vec{p} - \vec{q}_j\|^2, \quad \forall j \in \{i+1, \ldots, k\}, \\ \|\vec{p} - \vec{q}_i\|^2 < \|\vec{p} - \vec{q}_j\|^2, \quad \forall j \in \{1, \ldots, i-1\}\} \end{aligned}$$

That is, in case of coincident minimal distances, the point is assigned to the centroid with smallest index. Voronoi partitions are relevant due to the following known fact:

Fact 3. *There exists $\vec{q}$ in $\mathbb{R}^{dk}$ such that the Voronoi partition associated to $\vec{q}$ is optimal.*

Proof. Let $S = \{S_1, \ldots, S_k\}$ be an optimal partition. Let $\vec{q} = (\vec{q}_1, \ldots, \vec{q}_k)$ be the corresponding tuple of centroids as per Fact 2. Now let us show that the Voronoi partition $\{S'_1, \ldots, S'_k\}$ of S associated to $\vec{q}$, although it may be different from $\{S_1, \ldots, S_k\}$, leads to the same WCSS (thus also being an optimal partition).

Assume by contrary that it does not. Take $i \in \{1, \ldots, k\}$ to be the smallest index such that $S_i \neq S'_i$. Let $\vec{p}$ be a point in their symmetric difference $(S_i \backslash S'_i) \cup (S'_i \backslash S_i)$. We distinguish two cases:

- $\vec{p} \in S_i \backslash S'_i$. Thus, $\exists j > i$ such that $\vec{p} \in S'_j$, and therefore, $\|\vec{p} - \vec{q}_j\|^2 < \|\vec{p} - \vec{q}_i\|^2$. Consider now the partition constructed from $\{S_1, \ldots, S_k\}$ by moving $\vec{p}$ from S_i to S_j. It would have a WCSS with respect to $\vec{q} = (\vec{q}_1, \ldots, \vec{q}_k)$ strictly smaller than the optimal one (because the contribution of $\vec{p}$ to the sum is smaller), a contradiction.

- $\vec{p} \in S'_i \backslash S_i$. Thus, $\exists j > i$ such that $\vec{p} \in S_j$. Since $\vec{p} \in S'_i$ and $j > i$ we get that $\|\vec{p} - \vec{q}_i\|^2 \leq \|\vec{p} - \vec{q}_j\|^2$. Now, if $\|\vec{p} - \vec{q}_i\|^2 < \|\vec{p} - \vec{q}_j\|^2$ we can construct a new partition from $\{S_1, \ldots, S_k\}$ by moving $\vec{p}$ from S_j to S_i. This new partition would have a strictly smaller WCSS with respect to $\vec{q} = (\vec{q}_1, \ldots, \vec{q}_k)$ than the optimal one, a contradiction. We are left with the case in which $\|\vec{p} - \vec{q}_i\|^2 = \|\vec{p} - \vec{q}_j\|^2$. If $\vec{p}$ was the only point that distinguished the two partitions, then both partitions would have the same minimal WCSS with respect to $\vec{q} = (\vec{q}_1, \ldots, \vec{q}_k)$, contradicting our assumption. So there must be another point, say $\vec{p}'$, such that $\vec{p}'$ is in $(S_j \backslash S'_j) \cup (S'_j \backslash S_j)$ for some $j \geq i$.

  Following the same type of reasoning, we eventually get to a contradiction.

□

The key point of the whole approach is to reformulate each of the expressions $\|\vec{p} - \vec{q}_i\|^2 \leq \|\vec{p} - \vec{q}_j\|^2$, with $i < j$, in the equivalent form

$$\|\vec{q}_i\|^2 - \|\vec{q}_j\|^2 - 2\vec{p} \cdot (\vec{q}_i - \vec{q}_j) \leq 0,$$

(and likewise for strict inequalities) where $\cdot$ denotes the standard dot product, and expressing this condition as a polynomial $P_{\vec{p},i,j}(X_1, \ldots, X_{dk})$ corresponding to that $\vec{p}$, that i, and that j, where $1 \leq i < j \leq k$:

$$P_{\vec{p},i,j} = \sum_{r=1}^{d} \left(X^2_{r+(i-1)d} - X^2_{r+(j-1)d} \right) - 2 \sum_{r=1}^{d} p_r (X_{r+(i-1)d} - X_{r+(j-1)d}), \tag{1}$$

where $\vec{p} = (p_1, \ldots, p_d)$. Indeed, now it can be checked that for each $\vec{p} \in S$, $\vec{p} \in S_i$ if and only if: for all $j < i$, $P_{\vec{p},j,i}(\vec{q}) > 0$ and for all $j > i$, $P_{\vec{p},i,j}(\vec{q}) \leq 0$.

One can thus redefine the Voronoi partition associated to a point $\vec{q}$ in terms of these newly defined polynomials:

Fact 4. *The Voronoi partition of S associated to $\vec{q}$ is $(S_1, \ldots, S_k)$ where:*

$$S_i = \{\vec{p} \in S \mid P_{\vec{p},i,j}(\vec{q}) \leq 0, \quad \forall j \in \{i+1, \ldots, k\}, \\ P_{\vec{p},j,i}(\vec{q}) > 0, \quad \forall j \in \{1, \ldots, i-1\}\}$$

Note that the family of polynomials is on dk variables (the dimensionality of the tuple of centroids) but each individual polynomial $P_{\vec{p},i,j}$, with $i < j$, only uses $2d$ of them, corresponding to the coordinates of the two centroid candidates $\vec{q}_i$ and $\vec{q}_j$. Also, note that the total number of polynomials at work is $l = nk(k-1)/2$.

The relevant information, hence, is the sign of each of the l polynomials, evaluated on the tuple of centroids. For technical reasons, we will distinguish three signs, namely, -1, 0, and 1, and structure them into "sign vectors":

Definition 2. Let $\mathcal{P} = \{P_s \mid s \in I\}$ be a family of polynomials on m variables, indexed by some arbitrary finite index set I. The *sign vector* of a point $\vec{q} \in \mathbb{R}^m$ with respect to $\mathcal{P}$ is $sv_{\mathcal{P}}(\vec{q}) \in \{-1, 0, +1\}^{|I|}$ where, for $s \in I$, the s-th component of $sv_{\mathcal{P}}(\vec{q})$ is

$$sv_{\mathcal{P}}(\vec{q})_s = \begin{cases} +1 & \text{if } P_s(\vec{q}) > 0, \\ -1 & \text{if } P_s(\vec{q}) < 0, \\ 0 & \text{if } P_s(\vec{q}) = 0. \end{cases}$$

Whenever $\mathcal{P}$ is clear from the context, the subscript is omitted. For this section, the family $\mathcal{P}$ of polynomials will be, of course, the one in equation (1), indexed by the corresponding triples $(\vec{p}, i, j)$. We give an example later on.

Sign vectors will play two related but separate roles. The first role is simply to replace the conditions on the polynomials upon describing each cluster among those defined by $\vec{q}$: simply using $sv(\vec{q})_{\vec{p},i,j}$ appropriately in Fact 4, we see that:

Fact 5. *The Voronoi partition of S associated to $\vec{q}$ is $(S_1, \ldots, S_k)$ where:*

$$S_i = \{\vec{p} \in S \mid sv(\vec{q})_{\vec{p},i,j} \leq 0, \quad \forall j \in \{i+1, \ldots, k\}, \\ sv(\vec{q})_{\vec{p},j,i} = 1, \quad \forall j \in \{1, \ldots, i-1\}\}$$

So far we have not yet seen anything to gain from this approach. A brute force algorithm can just check out all possible sign vectors, that is, all possible clusterings, each obtained from each sign vector according to Fact 5, evaluate their cost, and keep the best seen along. The fact that there are more possible sign vectors than $sv(\mathbb{R}^{dk})$ is harmless for the correctness: in the worst case scenario, the set of optimal partitions encountered this way may strictly include the set of optimal Voronoi partitions associated to points in $\mathbb{R}^{dk}$; nevertheless, due to Fact 3, we know we cannot do any better in terms of the optimal cost. In fact, in this brute force approach, the value 0 is easily seen to be unnecessary for the sign vectors, and the algorithm would test 2^l sign vectors. For low enough number of points n, this can be better than the alternative that we deploy in the rest of this paper.

The second role of the sign vectors, which leads to all the known algorithms within the approach (and also to our own contribution), is as a way of classifying candidate centroid tuples. It is immediate from Fact 5 that the Voronoi partition associated to $\vec{q}$, actually, only depends on $sv(\vec{q})$. This defines an equivalence relation on $\mathbb{R}^{dk}$, and it suffices to find one representative from each equivalence class, and to check all the Voronoi partitions of these representatives, to make sure that the optimum clustering will be identified.

Note that the same clustering may correspond to non-equivalent points. Indeed, since the order in which we list the sets of each partition does not matter, we may end up having different sign vectors defining the same clustering simply because $\{S_1, S_2\}$ and $\{S_2, S_1\}$ represent the same clustering. But, it may also happen that two different vector signs define the same partition, in the same order, as we see next.

Example 1. *Let $S = \{-11, 0, 10\}$ be a set of three points in $\mathbb{R}$, and assume we want to separate them into three clusters. As we shall see, there are vectors in $\mathbb{R}^3$ that lead to the same clustering although they have*

different sign vectors. Intuitively, the reason will be that the differing sign targets whether point 0 *goes to the leftmost cluster or to the rightmost cluster, a decision that will be indeed different for the two sets of centroids, but that is irrelevant as that point goes into its own middle cluster.*
First, we need to define $l = nk(k-1)/2 = 9$ *polynomials, 3 for each point:*

$P_{-11,1,2} = X_1^2 - X_2^2 - 2 * (-11) * (X_1 - X_2)$
$P_{-11,1,3} = X_1^2 - X_3^2 - 2 * (-11) * (X_1 - X_3)$
$P_{-11,2,3} = X_2^2 - X_3^2 - 2 * (-11) * (X_2 - X_3)$
$P_{0,1,2} = X_1^2 - X_2^2 - 2 * 0 * (X_1 - X_2)$
$P_{0,1,3} = X_1^2 - X_3^2 - 2 * 0 * (X_1 - X_3)$
$P_{0,2,3} = X_2^2 - X_3^2 - 2 * 0 * (X_2 - X_3)$
$P_{10,1,2} = X_1^2 - X_2^2 - 2 * 10 * (X_1 - X_2)$
$P_{10,1,3} = X_1^2 - X_3^2 - 2 * 10 * (X_1 - X_3)$
$P_{10,2,3} = X_2^2 - X_3^2 - 2 * 10 * (X_2 - X_3)$

Now take $\vec{q} = (-10, 0, 11)$ *and* $\vec{q}' = (-11, 0, 10)$*. It can be checked that*

$sv(\vec{q}) = (-1, -1, -1, +1, -1, -1, +1, +1, +1)$,
$sv(\vec{q}') = (-1, -1, -1, +1, +1, -1, +1, +1, +1)$.

Nevertheless, the partition defined by both sign vectors is $\{S_1, S_2, S_3\}$*, where* $S_1 = \{-11\}$*,* $S_2 = \{0\}$ *and* $S_3 = \{10\}$*.*

Now, it turns out that existing techniques of algebraic geometry apply, so as to compute these sign vectors, which results in the already published algorithms—in fact, the equivalence classes just defined are an example of the configurations called "cell arrangements" in that context. As we show below, one can construct a sequence of polynomial equations in such a way that by solving all of them in turn, sufficiently many sign vectors are obtained to ensure that one of them corresponds to the optimum clustering.

That approach reduces optimum clustering to that general "cell enumeration" problem. Then, our main point in this paper is that we can show how to make these techniques more efficient, giving up a now unnecessary generality, and focusing on "cell enumeration" for the particular case of our polynomials defining the sign vectors. This is how we obtain our new nontrivial algorithmic improvements.

The idea that cell arrangements can be used to enumerate all Voronoi partitions appeared already in [13], where the authors claim (without proof) that "all the Voronoi partitions can be enumerated in $\mathcal{O}(n^{dk(k+1)/2})$ time by using the hyperplane arrangement". In a paper that appeared one year later (see [12, pp. 335]), the authors argue that "the number of Voronoi partitions is bounded by the combinatorial complexity of $nk(k-1)/2$ constant-degree algebraic surfaces", and therefore, "all the Voronoi partitions can be enumerated in $\mathcal{O}(n^{dk+1})$ time", with no further reference about which is the cell enumeration algorithm they had in mind. We believe that the authors referred to the result in [14], which states that given $\mathcal{H}$, an arrangement of l algebraic varieties in $\mathbb{R}^m$, there exists an algorithm for the cell enumeration problem with time complexity $l^{m+1}2^{\mathcal{O}(m^2)}$ (for the particular case in which the maximal degree of polynomials in $\mathcal{H}$ is 2).

An improvement of the algorithm in [14] was presented later in [15]. Their algorithm is based on a result along the lines of our Proposition 1 below, and its time complexity (for polynomials of degree at most 2, which is our case) is $l^{m+1}(1/m)^m 2^{O(m)}$ where, here, $m = dk$ and $l = nk(k-1)/2$. Proposition 1 basically says that instead of solving a system of l inequalities, one may solve $\binom{l}{i}$ systems of i equations of a certain type, for all $i \in \{1, \ldots, l\}$. Proposition 1 in [15] is similar in spirit, but it uses infinitesimals, which we would like to avoid. On the other hand, exploiting the algebraic side of the problem using the Euclidean distance also appears in [16], where the authors study the minimum value δ with which the centroids can be perturbed such that the corresponding clustering does not change. This is also implicitly calculated in our algorithms.

If we were to use our Proposition 1 below *ad litteram*, we would need much more computational power than the naive brute force algorithm presented above: even if we could solve each system of equations in constant time, we would still need $4^l - 1 = \sum_{i=1}^{l} 3^i \binom{l}{i}$ operations only for enumerating them. In [15], building on top of the results from [17], the authors show that it is enough to explore systems of at most m equations (where m is the dimensionality of the polynomials) if the polynomials are in general position (something that does not hold in our case). Also, they show how infinitesimals can be used to achieve

this property if this is not the case. The library called *RAGlib* of Maple (which can be freely downloaded from [18]) implements a function *PointsPerComponents* that contains the key ideas in [17] for finding at least one point in each of the connected components defined by a set of polynomial inequalities. The most recent version available is the result of several years of development and it includes improvements to speed up CPU time, see [19]. However, we could check experimentally that, for our concrete application of finding globally optimal k-means centroids, it is very slow compared to our approach (see Section 5). Instead, in the following section we show how we can prune the search space for our particular cases of interest by taking advantage of the properties of our specific family of polynomials.

3. Finding the Optimal Clustering

Recall that the optimal clustering can be obtained by enumerating all Voronoi partitions associated to points in $\mathbb{R}^{dk}$, and that this is equivalent to finding all elements of $sv_{\mathcal{P}}(\mathbb{R}^{dk})$ (see Fact 5), where $\mathcal{P}$ is a family of polynomials as in (1).

3.1. Solving a System of Inequalities

The process of finding one point in each equivalence class is, actually, that of solving a system that contains both equations and inequalities: for each sign vector, we wish a point that makes our polynomials attain their corresponding signs, either equality to 0, or being positive or negative.

The first step, namely Proposition 1, is to transform the system into one containing only equations. This approach is also employed in [15]. Nevertheless, we choose to include an alternative proof for this particular formulation, because the one in that reference resorts to extending the real line with infinitesimals through the usage of Puiseux series, and we would not be able to argue convincingly that the result does apply to our case, which lies fully within the real numbers. Moreover, avoiding infinitesimals will prove handy to attain much better running times.

Proposition 1. *Let $(e_1, \ldots, e_l)$ be a sign vector in $\{-1, 0, 1\}^l$ and P_1, ..., P_l a family of polynomials in $\mathbb{R}[X_1, \ldots, X_m]$. Consider the system of equations defined as follows*

$$sign(P_i) = e_i, \qquad \forall i \in \{1, \ldots, l\}, \tag{2}$$

and assume it has solutions in $\mathbb{R}^m$. Then, there exist a strictly positive $\varepsilon \in \mathbb{R}$ and a subset $I \subseteq \{1, \ldots, l\}$ such that (3) *has solutions, and at least one connected component of solutions of* (3) *is contained in the set of solutions of* (2)*:*

$$P_i = e_i\varepsilon, \qquad \forall i \in I. \tag{3}$$

Proof. First, without loss of generality, assume that the equations are permuted in such a way that $e_1 = e_2 = \ldots = e_s = 0$ and the rest of the e_i are nonzero. By additional multiplication by e_i for the nonzero cases, system (2) can therefore be written equivalently:

$$\begin{cases} P_1 = 0, \\ \cdots \\ P_s = 0, \\ e_{s+1}P_{s+1} > 0, \\ \cdots \\ e_lP_l > 0, \end{cases} \tag{4}$$

We will prove inductively that, for every r, with $s \leq r \leq l$, there is $\varepsilon_0 \in \mathbb{R}$, and there are subsets I, J such that $|J| = r$, $\{1, \ldots, s\} \subseteq I \subseteq J \subseteq \{1, \ldots, l\}$ and for all $\varepsilon \in \mathbb{R}$ with $0 < \varepsilon < \varepsilon_0$, the system

$$P_i = e_i\varepsilon, \quad \forall i \in I \tag{5}$$

has solutions, and at least one connected component of solutions of (5) is contained in the set of solutions of the system

$$sign(P_i) = e_i, \quad \forall i \in J; \tag{6}$$

furthermore, for all $\varepsilon \in \mathbb{R}$ with $0 < \varepsilon < \varepsilon_0$, there is $\vec{x}_0^{(\varepsilon)} \in \mathbb{R}^m$ that is both a solution of (4) and (5).

Observe that our claim follows from $r = l$ as, in this case, J covers the whole system (4) (any value less than the corresponding ε_0 can be used as ε).

The inductive basis is $r = s$ and there is little to argue, as we can take $I = J = \{1, \ldots, s\}$ that correspond to null e_i's; let us fix $\varepsilon_0 = 1$ but actually any positive value will do. As we have assumed that (4) has solutions, we can take any of them as $\vec{x}_0^{(\varepsilon)}$.

For the inductive step, we need to increase by 1 the size of the set J (unless $|J| = l$ already). We consider two cases. The easy one is when for all $\varepsilon \in (0, \varepsilon_0)$ and for all $\vec{x}$ that belong to the same connected component of solutions of (5) as $\vec{x}_0^{(\varepsilon)}$, $\vec{x}$ is also a solution of (2), which includes (6) for all extensions of J. Then we simply add j to J for some $j \notin J$, and leave $\vec{x}_0^{(\varepsilon)}$ unchanged.

Otherwise, there exists $\varepsilon_1 \in (0, \varepsilon_0)$ and $\vec{x}_1^{(\varepsilon_1)}$ that belongs to the same connected component of solutions of (5) as $\vec{x}_0^{(\varepsilon)}$, such that $\vec{x}_1^{(\varepsilon_1)}$ is not a solution of (2). Let f be a continuous function from $[0, 1]$ to the connected component of solutions of (5) that contains $\vec{x}_0^{(\varepsilon)}$, such that $f(0) = \vec{x}_0^{(\varepsilon_1)}$ and $f(1) = \vec{x}_1^{(\varepsilon_1)}$.

Since $\vec{x}_0^{(\varepsilon_1)}$ is solution of both (4) and (5), and $\vec{x}_1^{(\varepsilon_1)}$ is solution of (5) but not of (4), there must exist $j \in \{s + 1, \ldots, l\}$ such that the set $\{t \mid e_j P_j(f(t)) \leq 0\}$ is not empty (we know that $j \notin J$ because the whole image of f consists of solutions of (5)). Because the previous set equals $(e_j P_j \circ f)^{-1}((-\infty, 0])$, which is closed, we can take $t_0 \in (0, 1]$ minimal such that $f(t_0)$ is solution of (5) but not of (4) (in particular, $e_j P_j(f(t_0)) \leq 0$). Therefore, for all $t \in [0, t_0)$, $f(t)$ is a solution of (4). Since $f(t_0)$ is solution of (5), by the induction hypothesis, $f(t_0)$ satisfies (6). Now, if $e_j P_j(f(t_0)) < 0$, as $e_j P_j(f(0)) > 0$ because $\vec{x}_0^{(\varepsilon_1)}$ satisfies (4) and f, P_j are continuous functions, the value 0 would be attained midway through, contradicting the minimality of t_0; hence, $e_j P_j(f(t_0)) = 0$.

Now we show that we can safely add j to both I and J, where the role of ε_0 for the new I and J will be played by $\min\{\varepsilon_0, e_j P_j(f(0))\}$.

Fixing ε with $0 < \varepsilon < \min\{\varepsilon_0, e_j P_j(f(0))\}$,

- the system (5) for $I \cup \{j\}$ has solutions: since f, P_j are continuous functions and $e_j P_j(f(t_0))) = 0$, there exists $t \in (0, t_0)$ such that $e_j P_j(f(t))) = \varepsilon$, that is, $f(t)$ is a solution of $P_j(f(t))) = e_j \varepsilon$. We define the new $\vec{x}_0^{(\varepsilon)}$ as $f(t)$ for this t. Recall that $f(t)$ is also solution of (5), and therefore, $f(t)$ is a solution of the system (5) for $I \cup \{j\}$.

- the connected component of solutions of (5) for $I \cup \{j\}$ that contains $\vec{x}_0^{(\varepsilon)}$ is included in the set of solutions of (6) for $J \cup \{j\}$: by the induction hypothesis, the connected component of solutions of (5) for I that contains $\vec{x}_0^{(\varepsilon)}$ is included in the set of solutions of (6) for J. Besides, all solutions of $P_j = e_j \varepsilon$ must be solutions of $sign(P_j) = e_j$.

- for the newly defined $\vec{x}_0^{(\varepsilon)}$ we have: $\vec{x}_0^{(\varepsilon)}$ is solution of (5) for $I \cup \{j\}$ and solution of (4) since $t \in (0, t_0)$.

This concludes the proof. □

Therefore, given an arbitrary vector $\vec{e} = (e_1, \ldots, e_l)$ in $\{-1, 0, 1\}^l$ and an arbitrary family of polynomials $\mathcal{P} = \{P_1, \ldots, P_l\}$ in $\mathbb{R}^m$, if $\vec{e} \in sv_{\mathcal{P}}(\mathbb{R}^m)$ then there exists $\varepsilon > 0$ and a subfamily $\mathcal{P}_I = \{P_s \mid s \in I\}$ (for some $I \subseteq \{1, \ldots, l\}$) such that the system $P_i = e_i \varepsilon, \forall i \in I$ has solutions. Moreover, at least one connected component of solutions of the later system is contained in $sv_{\mathcal{P}}^{-1}(\vec{e})$.

This way, instead of dealing with systems that, most often, include inequalities, we reduced the problem to various systems of equations only. But, on one hand, we drastically increased the computational time (since the number of systems to be solved is exponential in the number of polynomials), and on the other hand, instead of having to find just one $\vec{x}$ in $sv_{\mathcal{P}}^{-1}(\vec{e})$, we must now go through all connected components of a big number of systems.

In the sequel, we show how to express our problem with systems of just linear equations. This will have a triple positive effect:

- basic linear algebra tells us that a compatible linear system that has more equations than variables can be reduced to an equivalent one that has at most as many equations as variables by eliminating all redundant information, and this will save us considerable time;
- linear systems are faster to solve;
- the space of solutions of a linear system has at most one connected component, so that the corresponding issue for applying Proposition 1 becomes irrelevant.

3.2. Obtaining a Linear System of Equations

We apply a change of variables such that, on one hand, the systems of equations to be solved become linear, and on the other hand, the number of variables is kept relatively small. As we shall see, the transformation we employ is not reversible.

Consider the family $\mathcal{P} = (P_{\vec{p},i,j})_{\vec{p}\in S, 1\leq i<j\leq k}$ of polynomials with kd variables $X_1, \ldots, X_{kd}$ defined as in (1). We observe that $P_{\vec{p},i,j} = P_{\vec{p},1,j} - P_{\vec{p},1,i}$ for all $2 \leq i < j \leq k$. We perform the following change of variables:

$Y_i = \sum_{r=1}^{d} X^2_{r+(i-1)d}$, for all $i \in \{1, \ldots, k\}$, and

$Z_{i+(j-2)d} = X_i - X_{i+(j-1)d}$, for all $i \in \{1, \ldots, d\}$ and $j \in \{2, \ldots, k\}$,

We get $\mathcal{P}' = (P'_{\vec{p},i,j})_{\vec{p}\in S, 1\leq i<j\leq k}$ polynomials in $\mathbb{R}[Y_1, \ldots, Y_d, Z_1, \ldots, Z_{(k-1)d}]$. It is easy to check that

$$\begin{aligned} P'_{\vec{p},1,i} &= Y_1 - Y_i - 2\sum_{r=1}^{d} p_r Z_{r+(i-2)d}, \forall i \in \{2, \ldots, k\} \\ P'_{\vec{p},i,j} &= P'_{\vec{p},1,j} - P'_{\vec{p},1,i}, \forall 1 < i < j \leq k \end{aligned} \tag{7}$$

Note that the new family of polynomials has dimensionality $k + (k-1)d$, whereas the old system has dimensionality kd.

Let $\mathcal{I}$ be the index set $\mathcal{I} = \{(\vec{p}, i, j) \mid \vec{p} \in S, 1 \leq i < j \leq k\}$; fix an order on its elements, say $o : \mathcal{I} \to \{1, \ldots, l\}$ with $l = nk(k-1)/2$.

Fact 6. *Let $\mathcal{P} = \{P_s \mid s \in \mathcal{I}\}$ be a family of polynomials in $\mathbb{R}[X_1, \ldots, X_{dk}]$ as in equation (1) and $(e_1, \ldots, e_l)$ a sign vector in $\{-1, 0, 1\}^l$. Consider the system of equations defined by*

$$sign(P_s) = e_{o(s)}, \qquad \forall s \in \mathcal{I}, \tag{8}$$

If (8) has solutions, then the following system

$$sign(P'_s) = e_{o(s)}, \qquad \forall s \in \mathcal{I}, \tag{9}$$

also has.

The proof of this fact is a straightforward application of the transformation defined above. Note that the converse does not hold in general.

Our objective will be now to enumerate all vectors in $sv_{\mathcal{P}'}(\mathbb{R}^{k+(k-1)d})$. Indeed, we want to traverse $sv_{\mathcal{P}}(\mathbb{R}^{dk})$. For each such sign vector, there is a solution of (8). The corresponding system (9) has solutions by Fact 6, which would give us the same sign vector, the object we actually need. Thus, by enumerating $sv_{\mathcal{P}'}(\mathbb{R}^{k+(k-1)d})$, we include all of $sv_{\mathcal{P}}(\mathbb{R}^{dk})$. Again, we might end up obtaining more sign vectors than we actually need, but as we argued before, this does not affect the correctness of the result.

Corollary 1. *Let $(e_1, \ldots, e_l)$ be a sign vector in $\{-1, 0, 1\}^l$ and the family $\mathcal{P}' = \{P'_s \mid s \in \mathcal{I}\}$ of polynomials in $\mathbb{R}[X_1, \ldots, X_m]$ with $m = k + (k-1)d$ as in equation (7). Assume that the system of equations defined as*

in (9) *has solutions in* $\mathbb{R}^m$*. Then there exist a strictly positive* $\varepsilon \in \mathbb{R}$ *and a subset* $I \subseteq \mathcal{I}$ *such that* (10) *has solutions, and all solutions of* (10) *are solutions of* (9)*:*

$$P'_s = e_{o(s)}\varepsilon, \qquad \forall s \in I. \tag{10}$$

Furthermore, consider

$$P'_s = e_{o(s)}, \qquad \forall s \in I. \tag{11}$$

This system has solutions if and only if (10) *has solutions.*

Proof. The first claim about (10) is a direct consequence of applying Proposition 1 to a system of linear equations, in which the space of solutions can have at most one connected component (see [20, Theorem 16.1.1]). The second claim just corresponds to the fact that, as the equations are linear and $\varepsilon > 0$, we can freely multiply or divide the solution values by ε and apply distributivity as needed.

□

3.3. Reducing the Number of Equations

According to Corollary 1, in order to enumerate all the sign vectors in $sv_{\mathcal{P}'}(\mathbb{R}^{k+(k-1)d})$, one can check, for all subfamilies $\mathcal{P}_I$ of the original family $\mathcal{P}_\mathcal{I}$, for positive $\varepsilon \in \mathbb{R}$, and for all sign vectors $\vec{e}$ in $\{-1, 0, 1\}^l$, whether the system (10) (or, equivalently, (11), where we do not need ε anymore) has solutions. If it does, any solution $\vec{x} \in \mathbb{R}^{k+(k-1)d}$ satisfies $sv(\vec{x}) = \vec{e}$.

As we mentioned before, this implies checking a huge number of systems. But, any compatible system of type (11) that has more equations than variables is necessarily equivalent to a smaller one, in which at most $k + (k-1)d$ equations (that is, the number of variables) of the original one are present. Thus, we may simply ignore bigger systems, since their solutions will eventually appear in our enumeration anyway. This leads to the following fact.

Fact 7. *In order to enumerate the elements of* $sv_{\mathcal{P}'}(\mathbb{R}^{k+(k-1)d})$*, one can check, for all* $I \subseteq \mathcal{I}$ *with* $|I| \leq k + (k-1)d$ *and for all* $\vec{e} = (e_1, \ldots, e_l)$ *in* $\{-1, 0, 1\}^l$*, whether the system* $P'_s = e_{o(s)}, \forall s \in I$ *has solutions. If it does, any solution* $\vec{x} \in \mathbb{R}^{k+(k-1)d}$ *satisfies* $sv(\vec{x}) = \vec{e}$*.*

3.4. Global Optimum Clustering Algorithm

All the ideas presented above are encapsulated in Algorithm 1. Note that each (sub)system $P'_s = e_{o(s)}, \forall s \in I$ as in Fact 7 depends only on $l' = |I|$ of the l components of the sign vector $\vec{e}$. Therefore, it is enough for the algorithm to go through vectors in $\{-1, 0, 1\}^{l'}$ instead of all vectors $\{-1, 0, 1\}^l$. With this in mind, we need a bit of additional notation to finally express our algorithm. Let $o(I) = \{i_1, \ldots, i_{l'}\}$ and $h_I : \mathbb{R}^{l'} \to \mathbb{R}^l$ be a function such that if $\vec{b} = (b_1, \ldots, b_{l'})$ then

$$h_I(\vec{b})_i = \begin{cases} 0, & \text{if } i \notin o(I) \\ b_j, & \text{if } i = i_j \in o(I) \end{cases}$$

The correctness of this algorithm is a direct consequence of Fact 7. In the sequel, we discuss its time complexity. For this, we define $g : \mathbb{N}^3 \to \mathbb{N}$ by

$$g(x, y, z) = \sum_{i=0}^{y} z^i \binom{x}{i}.$$

The following result holds.

Theorem 2. *Given* $S \subset \mathbb{R}^d$ *a set of cardinality* n*, the number of Voronoi partitions into* k *clusters is, at most,* $g(l, m, 3)$*, and the number of operations necessary to generate each of the partitions is at most* $m^3 + m^2 l$*, where* $m = k + (k-1)d$ *and* $l = nk(k-1)/2$*.*

Algorithm 1 Global optimum clustering

1: input: $S = \{\vec{p}_1, \ldots, \vec{p}_n\}$ a set of n points in $\mathbb{R}^d$, k = the number of clusters
2: $m = k + (k-1)d$ //the number of variables
3: $l = nk(k-1)/2$ //the number of polynomials
4: let $\mathcal{I} = \{(\vec{p}, i, j) \mid \vec{p} \in S, 1 \leq i < j \leq k\}$, and $o : \mathcal{I} \to \{1, \ldots, l\}$ a bijection
5: let $\mathcal{P}' = (P'_s)_{s \in \mathcal{I}}$ be the family defined as in (7)
6: min $= \infty$
7: **for all** l' in $\{1, \ldots, m\}$ **do**
8: **for all** $I \subseteq \mathcal{I}$ with $|I| = l'$ **do**
9: **for all** $\vec{b}$ in $\{-1, 0, 1\}^{l'}$ **do**
10: $\vec{e} := h_I(\vec{b})$
11: **if** $\exists\, \vec{x} \in \mathbb{R}^m$ solution of (11) **then**
12: let $\{S_1, \ldots, S_k\}$ be the partition defined by $sv_{\mathcal{P}'}(\vec{x})$
13: **if** $cost(S_1, \ldots, S_k) <$ min **then**
14: min $= cost(S_1, \ldots, S_k)$
15: save $\{S_1, \ldots, S_k\}$ as the optimal partition
16: **end if**
17: **end if**
18: **end for**
19: **end for**
20: **end for**
21: Output: min and the optimal partition

Proof. The upper bound for the number of Voronoi partitions can be obtained by simply counting the number of times we reach line 10 of the algorithm, that is, $3\binom{l}{1} + 3^2\binom{l}{2} + \cdots + 3^m\binom{l}{m} = g(l, m, 3) - 1$.

In order to find a solution, the algorithm needs to solve a system of linear equations (this can be done in m^3 operations, where $m = k + (k-1)d$ is the number of variables), and then, for each solution found, it has to evaluate its sign vector. Since there are $l = nk(k-1)/2$ polynomials in $\mathbb{R}^m$, the total running time for this operation is $m^2 l$.

□

Corollary 2. *Given $S \subset \mathbb{R}^d$ a set of cardinality n, Algorithm 1 outputs a global optimum clustering in time $\mathcal{O}(n^{k+(k-1)d+1})$.*

Proof. The result follows from Theorem 2, by multiplying the total number of Voronoi partitions visited with the time spent on getting and evaluating the actual partition, and taking into account that $g(x, y, 3)$ is bounded by $\mathcal{O}(x^y)$. Indeed, by replacing the values of l and m into $\mathcal{O}(l^m)(m^3 + m^2 l)$, we get $\mathcal{O}((nk(k-1)/2)^{k+(k-1)d}((k + (k-1)d)^3 + (k + (k-1)d)^2 nk(k-1)/2)$, which is exactly $\mathcal{O}(n^{k+(k-1)d+1})$ if k and d are treated as constants. □

A complexity-theoretic comparison with the previous work is provided in Table 1 in the end of Section 4, together with the corresponding comparison with our subsequent contributions in the forthcoming sections. Beyond the differences in running time predicted by that analysis, one important property of this whole approach, hence shared as well by Algorithm 1, is the following: the only interaction between all the different clusterings tested is at the time of finding which one is best. Finding each candidate clustering and evaluating its cost is a task fully independent of the analogous task for other candidate clusterings. This means that all these algorithms are highly parallelizable. Indeed we take advantage of this possibility and we report in Section 5 execution times of our implementation on massively parallel hardware.

4. Particular Cases of Special Interest

Note that our approach is better than the one in [12] only when $k \leq d$. Indeed, in most cases of interest, the dimensionality of the data tends to be high, whereas the user would wish not to be overwhelmed with a large number of clusters. So, for a given dimensionality $d \geq 2$, it makes sense to use our algorithm for any $k \in \{2, \ldots, d\}$. The case of $d = 1$ can be efficiently solved by dynamic programming (see [21, 22]) in $\mathcal{O}(n^2k)$. The use of dynamic programming has also been employed to cope with the general case ($d > 2$) in [23, 24] leading to improved k-means like algorithms.

An interesting particular problem is that of "binary splitting", that is, separating the data points into exactly two clusters; in our notation, this corresponds to the special case of $k = 2$. For this case, we present an algorithm that returns the global optimum clustering in $\mathcal{O}(n^{d+2})$, which may not seem much of an improvement over Algorithm 1 in theory, but it makes a lot of difference in practice as we shall see in Section 5.

4.1. Binary Splitting ($k = 2$)

We know from Proposition 1 that in order to go through all feasible sign vectors corresponding to the polynomials $\{P_1, \ldots, P_l\}$ defined by our initial data points S, we can solve instead systems like (3) for subsets $I \subseteq \{1, \ldots, l\}$ and some strictly positive $\varepsilon \in \mathbb{R}$. Taken together, these solutions cover all equivalence classes of tuples of centroids. Therefore, that solution yielding best cost of the corresponding clustering tells us the global optimum. As we did in the general case, we show that for $k = 2$ it is not necessary to check all the subsets of $I \subseteq \{1, \ldots, l\}$ (a quite large number, although note that $l = n$ for $k = 2$), but a much smaller family.

Theorem 3. *Consider the family of polynomials $\mathcal{P} = \{P_{\vec{p},1,2} \mid \vec{p} \in S\}$ in $\mathbb{R}^{2d}$, defined as in (1), but restricted to $k = 2$. Let $\varepsilon \in \mathbb{R}$ be positive. Then, there is a second family of n polynomials $\mathcal{P}' = \{P'_i \mid 1 \leq i \leq n\}$ in $\mathbb{R}^{2d}$ and an invertible $2d \times 2d$ matrix M such that the following holds:*

1. *For each $\vec{x} \in \mathbb{R}^{2d}$, $sv_{\mathcal{P}}(\vec{x}) = sv_{\mathcal{P}'}(M\vec{x}) \in \mathbb{R}^n$.*
2. *For every $I \subseteq \{1, \ldots, n\}$, and every $e_i \in \{-1, 0, +1\}$ for $i \in I$, there is $J \subseteq I$, with $|J| \leq d + 1$, such that $\vec{x}$ is a solution of $P_{\vec{p}_i,1,2} = e_i\varepsilon, \forall i \in I$ if and only if $M\vec{x}$ is a solution of*

$$P'_i = e_i \qquad \forall i \in J. \tag{12}$$

3. *The solution of this system $\{P'_i = e_i \mid i \in J\}$, if it exists, can be obtained by solving successively two linear systems of dimension d.*

Proof. The matrix M and the new family of polynomials are defined via the following change of variables:
$Y_r = (X_r - X_{r+d})/\varepsilon$, for all $r \in \{1, \ldots, d\}$,
$Z_r = X_r + X_{r+d}$, for all $r \in \{1, \ldots, d\}$.
It is easy to check that the reverse transformation is:
$X_r = (Z_r + \varepsilon Y_r)/2$, for all $r \in \{1, \ldots, d\}$,
$X_{r+d} = (Z_r - \varepsilon Y_r)/2$, for all $r \in \{1, \ldots, d\}$.
In this way, the new family of polynomials and the matrix associated to the transformation become:

- $\mathcal{P}' = \{P'_i \mid 1 \leq i \leq n\}$, with

$$P'_i = \sum_{r=1}^{d} Y_r(Z_r - 2p_r^{(i)}), \forall i \in \{1, \ldots, n\} \tag{13}$$

where, in order to simplify subindexes, we write as $p_r^{(i)} = (\vec{p}_i)_r$, the r-th component of $\vec{p}_i \in S$ (we keep this notation for the current proof).

- $M = (m_{ij})_{1 \le i,j \le 2d}$ is a matrix defined by

$$m_{ij} = \begin{cases} 1/\varepsilon & \text{if } i = j \le d \\ -1/\varepsilon & \text{if } i = j - d \\ 1 & \text{if } i = j > d \text{ or } i = j + d \\ 0 & \text{otherwise} \end{cases}$$

To prove the first claim, consider a component of $sv_{\mathcal{P}}(\vec{x})$, that is, the sign of some $P_{\vec{p}_i,1,2}(X_1, \ldots, X_{2d}) = \sum_{r=1}^{d} (X_r - X_{r+d})(X_r + X_{r+d} - 2p_r^{(i)})$ evaluated on $\vec{x}$. We have

$$P_{\vec{p}_i,1,2}(X_1, \ldots, X_{2d}) = \sum_{r=1}^{d} \varepsilon Y_r (Z_r - 2p_r^{(i)}) = \varepsilon P'_i(Y_1, \ldots, Y_d, Z_1, \ldots, Z_d),$$

i.e., its sign coincides with that of the corresponding component of $sv_{\mathcal{P}'}(M\vec{x})$, because $\varepsilon > 0$.

The second and third claims will follow from the following additional transformation. Fix $I = \{i_1, \ldots, i_{l'}\}$ and the corresponding e_i, so that we are working with the transformed system $\{P'_i = e_i \mid i \in I\}$. By deducting the first equation from all the others, this system of equations becomes:

$$\begin{cases} \sum_{r=1}^{d} Y_r (Z_r - 2p_r^{(i_1)}) = e_{i_1}, \\ -2 \sum_{r=1}^{d} (p_r^{(i_2)} - p_r^{(i_1)}) Y_r = (e_{i_2} - e_{i_1}), \\ \ldots = \ldots, \\ -2 \sum_{r=1}^{d} (p_r^{(i_{l'})} - p_r^{(i_1)}) Y_r = (e_{i_{l'}} - e_{i_1}), \end{cases}$$

In matrix notation,

$$\begin{cases} \sum_{r=1}^{d} Y_r (Z_r - 2p_r^{(i_1)}) = e_{i_1}, \\ A \cdot Y = b \end{cases} \tag{14}$$

where $Y = (Y_1, \ldots, Y_d)^T$, $b = (e_{i_2} - e_{i_1}, \ldots, e_{i_{l'}} - e_{i_1})^T$ and $A = (a_{jr})_{j,r}$ with $a_{jr} = -2(p_r^{(i_{j+1})} - p_r^{(i_1)})$ for all $j \in \{1, \ldots, l' - 1\}$ and $r \in \{1, \ldots, d\}$.

Since the rank of A is at most d, the system of equations $A \cdot Y = b$ either has no solutions, being inconsistent, or, if it does have one or more solutions and $l' - 1 > d$, it must be the case that one of the equations can be written as a linear combination of other equations, and can be eliminated while maintaining an equivalent system. This can be repeated as long as we have more than d equations. Then, J is simply the set of indexes of the equations left over. Note that the equations in $A \cdot Y = b$ correspond bijectively to the equations in both the original system $\{P_{\vec{p}_i,1,2} = e_i\varepsilon \mid i \in I\}$ and the transformed one, $\{P'_i = e_i \mid i \in I\}$. Hence, the solution can be found by solving $\{P'_i = e_i \mid i \in J\}$.

The same transformation explains how to solve it with a standard linear equation system solver. Solving $A \cdot Y = b$ (restricted to J, of course) provides us with values for the Y variables. Then, substituting them into $\sum_{r=1}^{d} Y_r (Z_r - 2p_r^{(i_1)}) = e_{i_1}$, we obtain one linear equation on $Z_1, \ldots, Z_d$, and we simply solve it as well. □

The algorithm that outputs the global minimal clustering for $k = 2$ is described below (Algorithm 2). Its correctness is a direct consequence of Theorem 3, and its time complexity is given by the following theorem.

Theorem 4. *Given $S \subset \mathbb{R}^d$ a set of cardinality n, the number of Voronoi partitions into two clusters is, at most, $g(n, d+1, 3)$, and the number of operations necessary to generate each of the partitions is at most $(d+1)^3 + (d+1)^2 n$.*

The proof is very similar to the one of Theorem 2, and we omit it.

Corollary 3. *Given $S \subset \mathbb{R}^d$ a set of cardinality n, Algorithm 2 outputs a global optimum binary clustering in $\mathcal{O}(n^{d+2})$.*

Algorithm 2 Global optimum clustering for $k = 2$

```
input: S = {p_1, ..., p_n} a set of n points in R^d
let P' = {P'_i | 1 ≤ i ≤ n} be the family defined as in (13)
min = ∞, E = ∅
for all l' in {1, ..., d, d+1} do
  for all J ⊆ {1, ..., n} with |J| = l' do
    for all b ∈ {−1, 0, +1}^{l'} do
      e := h_J(b)
      if ∃ y ∈ R^{2d} solution of (12) then
        let {S_1, S_2} be the partition defined by sv_P(M^{-1} y)
        if cost(S_1, S_2) < min then
          min = cost(S_1, S_2)
          save {S_1, S_2} as the optimal partition
        end if
        add sv_P(M^{-1} y) to E
      end if
    end for
  end for
end for
Output: min and the optimal partition
```

Maintaining the set E of feasible sign vectors is a step that can be skipped for $k = 2$. As we shall shortly see, we only make use of it as a baseline for the general clustering problem ($k > 2$).

4.2. Reducing k-clustering to 2-clustering

In order to explain how the k-clustering problem can be reduced to the 2-clustering problem we need to introduce some notation. As before, let $S = \{\vec{p}_1, \ldots, \vec{p}_n\}$ in $\mathbb{R}^d$ and let us fix $k > 2$. We define a family of projection functions $pr^{ij} : \mathbb{R}^{dk} \to \mathbb{R}^{2d}$, with $1 \leq i < j \leq k$, by

$$pr^{ij}(x_1, \ldots, x_{dk}) = (x_{(i-1)d+1}, \ldots, x_{id}, x_{(j-1)d+1}, \ldots, x_{jd}).$$

Let $\mathcal{P} = (P_{\vec{p},i,j})_{\vec{p} \in S, 1 \leq i < j \leq k}$ be a family of $nk(k-1)/2$ polynomials in $\mathbb{R}[X_1, \ldots, X_{dk}]$ defined as in (1) and $\mathcal{P}' = (P'_{\vec{p},1,2})_{\vec{p} \in S}$ be the family of n polynomials in $\mathbb{R}[X_1, \ldots, X_{2d}]$ defined as in (1) for the particular case of the 2-clustering problem.

Recall that for the index set $\mathcal{I} = \{(\vec{p}, i, j) \mid \vec{p} \in S, 1 \leq i < j \leq k\}$ we have arbitrarily fixed an order $o : \mathcal{I} \to \{1, \ldots, l\}$ on its elements ($l = nk(k-1)/2$). Now we make this order explicit by setting

$$o(\vec{p}_t, i, j) = t + n * (j - i - 1 + (i-1) * k - i * (i-1)/2), \tag{15}$$

for all $t \in \{1, \ldots, n\}$ and for all $i, j \in \{1, \ldots, k\}$ with $i < j$. Note that o is a bijection.

An important observation is that given $\vec{p}$ in S, $P_{\vec{p},i,j}$ only uses $2d$ of its variables, being transformable to $P'_{\vec{p},1,2}$ for all $1 \leq i < j \leq k$. More formally, $P_{\vec{p},i,j}(\vec{x}) = P'_{\vec{p},1,2}(pr^{ij}(\vec{x}))$.

With this notation, we have the following lemma.

Lemma 5. *If $(e_1, \ldots, e_l)$ is a feasible sign vector with respect to $\mathcal{P}$, then $(e_1, \ldots, e_n), (e_{n+1}, \ldots, e_{2n}), \ldots, (e_{n(k(k-1)/2-1)+1}, \ldots$ are all feasible sign vectors with respect to $\mathcal{P}'$ if the order of polynomials in $\mathcal{P}$ is given by the bijection o defined as in (15).*

Proof. Let $\vec{e} = (e_1, \ldots, e_l)$ be a feasible sign vector with respect to $\mathcal{P}$ and let $\vec{x} = (x_1, \ldots, x_{kd})$ in $\mathbb{R}^{dk}$ be such that $sv(\vec{x}) = \vec{e}$. Since the order of polynomials in $\mathcal{P}$ is given by o, we have that $sign(P_{\vec{p}_t,i,j}(\vec{x})) = e_s$, where $s = o(\vec{p}_t, i, j)$ is a number in $\{1, \ldots, l\}$ for all $t \in \{1, \ldots, n\}$ and for all $i, j \in \{1, \ldots, k\}$ with $i < j$. More precisely, $s = t + n * (j - i - 1 + (i-1) * k - i * (i-1)/2)$, and thus, $s = t + n\sigma(i,j)$ where $\sigma(i,j) := j - i - 1 + (i-1) * k - i * (i-1)/2$. Note that $\sigma : \{(i,j) \mid 1 \leq i < j \leq k\} \to \{0, \ldots, k(k-1)/2 - 1\}$ is a bijection. Therefore,

$$\begin{aligned}(e_{1+n\sigma(i,j)}, \ldots, e_{n+n\sigma(i,j)}) &= (sign(P_{\vec{p}_1,i,j}(\vec{x})), \ldots, sign(P_{\vec{p}_n,i,j}(\vec{x}))) \\ &= (sign(P'_{\vec{p}_1,1,2}(pr^{ij}(\vec{x}))), \ldots, sign(P'_{\vec{p}_n,1,2})(pr^{ij}(\vec{x}))) \\ &= (sign(P'_{\vec{p}_1,1,2}(\vec{y})), \ldots, sign(P'_{\vec{p}_n,1,2})(\vec{y})),\end{aligned}$$

where $\vec{y} = pr^{ij}(\vec{x})$.

Hence, each $(e_{1+ns'}, \ldots, e_{n+ns'})$ for s' in $\{0, \ldots, k(k-1)/2 - 1\}$ is a feasible sign vector with respect to $\mathcal{P}'$, which concludes our proof. □

Lemma 5 ensures that the Algorithm 3 is correctly outputting at least all feasible sign vectors with respect to $\mathcal{P}$ (it may also output non feasible sign vectors, but those will not affect the solution to the k-clustering problem).

Algorithm 3 Global optimum clustering for $k > 2$

```
input: S = {p_1, ..., p_n} a set of n points in R^d
Let E be the set obtained while running Algorithm 2 on S
min = ∞
for all e in E^{k(k-1)/2} do
    let {S_1, ..., S_k} be the partition defined by e
    if cost(S_1, ..., S_k) < min then
        min = cost(S_1, ..., S_k)
        save {S_1, ..., S_k} as the optimal partition
    end if
end for
Output: min and the optimal partition
```

In order to evaluate the time complexity of Algorithm 3, we need some further technical results.

Lemma 6. *Let $\mathcal{H} = (H_{\vec{p}})_{\vec{p} \in S}$ with $H_{\vec{p}} = \{\vec{x} \in \mathbb{R}^{2d} \mid P_{\vec{p},1,2}(\vec{x}) = 0\}$, where each $P_{\vec{p},1,2}$ is defined as in (1) for the particular case of $k = 2$. Then the number of cells in the cell arrangement defined by $\mathcal{H}$ is bounded by $2g(n, d, 2)$, and thus, it is of order $\mathcal{O}(n^d)$.*

Proof. Notice that $P_{\vec{p},1,2} = \sum_{r=1}^{d} (X_r^2 - X_{r+d}^2) - \sum_{r=1}^{d} 2p_r(X_r - X_{r+d})$ can be written as $P_{\vec{p},1,2} = Z - \sum_{r=1}^{d} 2p_r Y_r$, where $Z = \sum_{r=1}^{d} (X_r^2 - X_{r+d}^2)$ and $Y_r = X_r - X_{r+d}$.

With this change of variables, we would have a cell arrangement in which all the algebraic varieties are hyperplanes in $\mathbb{R}^{d+1}$. Then, one could use the result by H. Edelsbrunner (see [25] Lemma 1.2 on page 7) to bound the number of cells by

$$\sum_{i=0}^{d+1} \sum_{j=0}^{i} \binom{d+1-j}{i-j} \binom{n}{d+1-j}.$$

Next, we show that in fact, because all hyperplanes pass through the origin, the dimension can be further decreased to d, while paying a small cost for it.

After applying the above mentioned change of variables, $\mathcal{H}$ is a cell arrangement in $\mathbb{R}^{d+1}$ over the following set of variables: $\{Y_r \mid \forall r \in \{1, \ldots, d\}\} \cup \{Z\}$. We shall consider the cell arrangement $\mathcal{H}' = \mathcal{H} \cup H$

where $H = \{\vec{x} \in \mathbb{R}^{d+1} \mid P(\vec{x}) = 0\}$ and $P = Z$. Clearly, the number of cells in $\mathcal{H}$ is smaller than the number of cells in $\mathcal{H}'$. In order to evaluate the number of cells in $\mathcal{H}'$, note that H divides each of the cells in $\mathcal{H}'$ in at most two cells. We shall consider then the following two cell arrangements $\mathcal{H}^0$ and $\mathcal{H}^1$, where $\mathcal{H}^a = (H^a_{\vec{p}})_{\vec{p}\in S}$, $H^a_{\vec{p}} = \{\vec{x} \in \mathbb{R}^d \mid P^a_{\vec{p}}(\vec{x}) = 0\}$ and $P^a_{\vec{p}} = a - \sum_{r=1}^{d} 2p_r Y_r$ ($a \in \{0, 1\}$), corresponding to points in $\mathbb{R}^{d+1}$ being either in H or outside H. According to Lemma 1.2 on page 7 in [25], the number of cells in $\mathcal{H}^0$ and $\mathcal{H}^1$ is bounded by

$$\sum_{i=0}^{d}\sum_{j=0}^{i}\binom{d-j}{i-j}\binom{n}{d-j}.$$

By some straightforward calculation, this sum can be shown to be equal to $\sum_{i=0}^{d} 2^i\binom{n}{i}$, which concludes the proof.

□

The complexity of Algorithm 3 is therefore given by the following result.

Theorem 7. *Given $S \subset \mathbb{R}^d$ a set of cardinality n, the number of operations necessary to generate each of the partitions is of order $\mathcal{O}(n^{dk(k-1)/2})$.*

Proof. According to Theorem 4, the total number of operations needed to output the set E is $(d+1)^2(d+1+n)g(n, d+1, 3)$. The second part of the algorithm needs $|E|^{k(k-1)/2}$ operations. By Lemma 6, $|E|$ is bounded by $2g(n, d, 2)$. The total running time of the algorithm is therefore bounded by
$(d+1)^2(d+1+n)g(n, d+1, 3) + (2g(n, d, 2))^{k(k-1)/2}$.
Since $g(x, y, z) \leq z^y x^y (y+1)$, we can conclude that the total number of operations is of order $\mathcal{O}(n^{dk(k-1)/2})$.
□

A comparison of the complexity-theoretic analysis of the algorithms considered appears in Table 1.

	Algorithm 1	The algorithm in [12]	Algorithm 2	Algorithm 3
general	$\mathcal{O}(n^{k+(k-1)d+1})$	$\mathcal{O}(n^{kd+1})$		$\mathcal{O}(n^{dk(k-1)/2})$
$k = 2$	$\mathcal{O}(n^{d+3})$	$\mathcal{O}(n^{2d+1})$	$\mathcal{O}(n^{d+2})$	
$k = 3$	$\mathcal{O}(n^{2d+4})$	$\mathcal{O}(n^{3d+1})$		$\mathcal{O}(n^{3d})$

Table 1: Time complexity comparison

5. Experimental results

Implementations of the algorithms introduced in this paper can be found at `https://github.com/domingoUnican/optkmeans`.

As far as we know, this is the first implementation that finds the optimum clustering of multidimensional input data, so there was nothing to compare it with in a direct manner. However, the essence of both our algorithms and the one in [12] is finding all elements of $sv_{\mathcal{P}}(\mathbb{R}^m)$ given a certain family $\mathcal{P}$ of polynomials on m variables; the RAGlib library of Maple allows for an easy implementation of these tasks, including the algorithm described in [15]. Thus, we ran a set of preliminary experiments on a Windows laptop equipped with Maple and found that the algorithm in [12], already with the very modest values of $k = 2$, $d = 2$ and $n = 5$, was more than 10000 times slower than Algorithm 1, even if its theoretical time complexity was the same, $\mathcal{O}(n^5)$. This was to be expected as this comparison is unfair, since the RAGlib-based algorithm is solving a much more general problem. We consider that it does not make sense to proceed further with this comparison at larger values of the parameters. Therefore, we implemented two existing exponential algorithms: one based on backtracking (see *Uniform cost search* on page 75 in [7]) and the other one based

Criterion	Uniform-Cost	Depth-First
Time	b^d	b^m
Space	b^d	b^m
Optimal?	Yes	No
Complete?	Yes	No

Table 2: Evaluation of search strategies. b is the branching factor; d is the depth of solution; m is the maximum depth of the search tree

on a depth first search (see *Depth-first search* on page 77 in [7]). Their time and space complexities are reported below.

We tested our algorithms on the User Knowledge Modeling Data Set, which can be found online on the UCI Machine Learning Repository (see [26] for more details). It is a smallish real dataset with students' knowledge status about the subject of Electrical DC Machines, and it contains 258 instances with 5 attributes, each of these attributes being a real number in the $[0, 1]$ interval.

In order to evaluate the performance for specific k and d, we ran the algorithms with the first n instances over the first d attributes for increasing n and d, and the time elapsed (expressed in seconds on a natural logarithmic scale) was plotted against the number of points. We show first a comparison between the running times of Algorithm 1 and Algorithm 2 for $k = 2$ with $d = 2, 3, 4, 5$ and the other two existing algorithms: UniformCost and Depth-First (Figure 1).

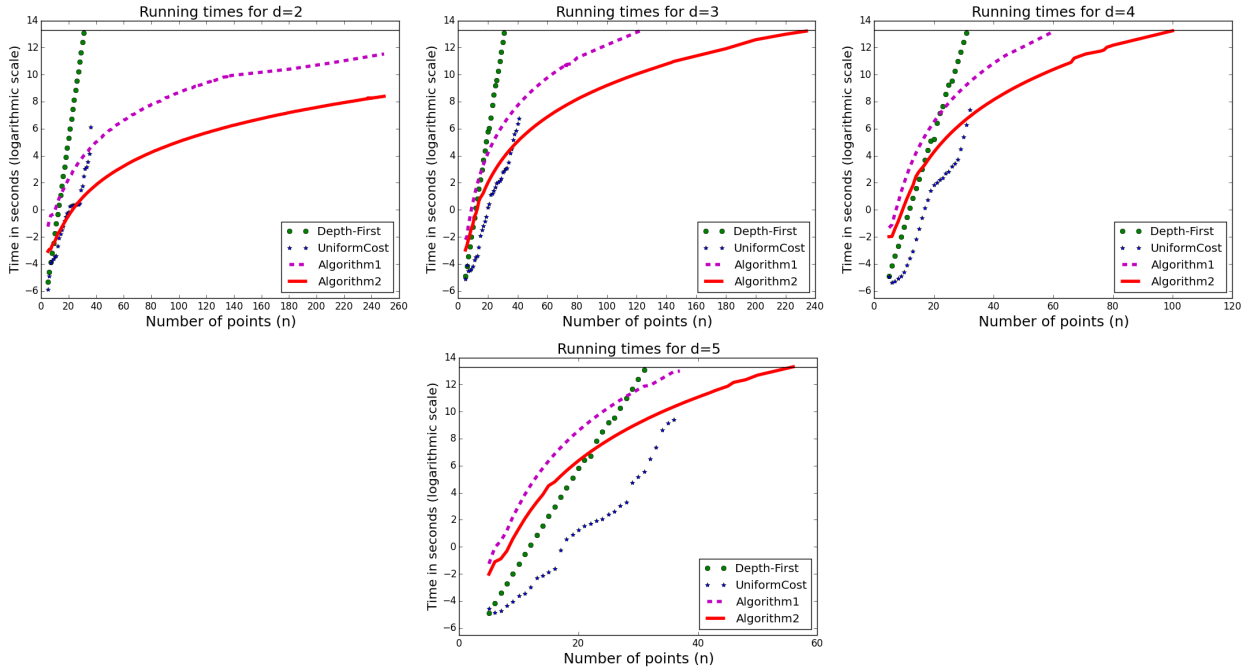

Figure 1: Running times for $k = 2$

The top horizontal line represents a seven days time limit (imposed by the supercomputing center we used). Note that although the UniformCost algorithm outperforms all other algorithms for small values of n, it runs out of memory very quickly (it cannot handle more than 40 points).

Then, we show a comparison between Algorithm 3 and the two exponential algorithms (UniformCost and Depth-First Search) for $k = 3$ and $d = 2, 3, 4, 5$. In this case, Algorithm 1 turns out to be very slow (we

were able to plot its running time only for very small values of n).

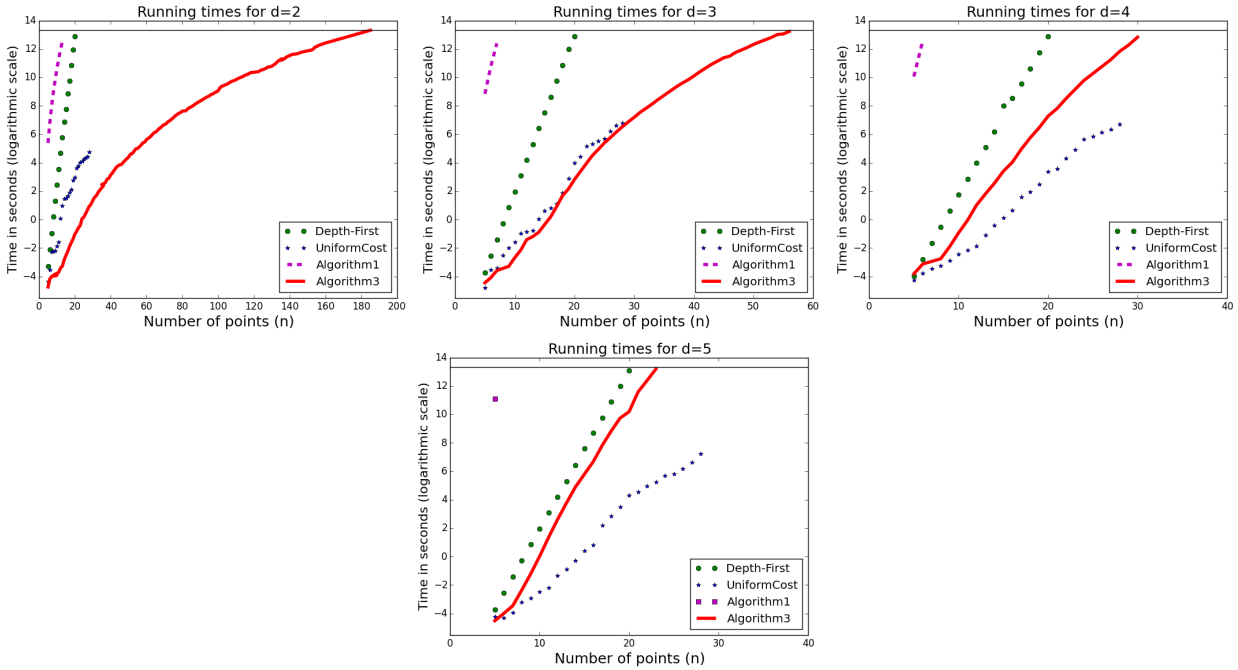

Figure 2: Running times for $k = 3$

Algorithm 3 was implemented only for the specific case of $k = 3$ so we could count on specific further optimizations. For example, we took advantage of some trivial observations in order to avoid checking all $\vec{e} \in E^3$. The first remark is that one of the three clusters is fixed when given two sign vectors in E (knowing the sign of $P_{\vec{p},1,2}$ and $P_{\vec{p},1,3}$ for all points $\vec{p}$ in S, we can already construct S_1). This allows to calculate one of the centroids and also to partially evaluate the WCSS. Experiments indicate that a great proportion of the clusterings can be shown to be not optimal just by using this partial evaluation of the WCSS. A second improvement is using the fact that not all vectors $\vec{e}$ in E^3 define a feasible sign vector for the set of polynomials defined in (1) (some combinations of signs for $P_{\vec{p},1,2}$, $P_{\vec{p},1,3}$ and $P_{\vec{p},2,3}$ are incompatible; for example, if $P_{\vec{p},1,2}(\vec{x}) < 0$ and $P_{\vec{p},1,3}(\vec{x}) > 0$ than $P_{\vec{p},2,3}(\vec{x}) > 0$ must also hold, so any sign that says otherwise may be discarded). In other words, the converse of Lemma 5 is clearly false. Indeed, we could experimentally check that implementing these kind of rules significantly improved the performance of the algorithm.

All the experiments were done on ALTAMIRA, the supercomputing node at the University of Cantabria within the Spanish Supercomputing Network. More precisely, we used an HPC cluster, integrating IBM-idataplex dx360m4 servers, each one with 2xE5-2670 Intel Sandybridge Xeon processors, 64GB RAM, 500GB HD and all of them interconnected with InfiniBand FDR10 cards and switches from Mellanox in fat tree topology, completing a system with more than 2500 cores and 10 Terabytes of memory. The peak capacity exceeds 50 Tflops.

A list of optimal WCSS values for $k = 2$ and $k = 3$ with different dimensions and number of points is available at `https://github.com/domingoUnican/optkmeans`. This information can be used as a benchmark repository and could be extended to include other datasets usually employed in testing partitional clustering methods based on the Euclidean distance (for example, those used in [27] or [28]).

6. Conclusion

Obtaining the optimum global k-clustering of a set of multidimensional points is a problem known to be NP-hard. Existing faster, practical algorithms employ different heuristics to get a local optimum. The issue is that the solution obtained by these algorithms could be quite far from the optimal one. It is easy to find examples in which k-means is unable to return the right answer, no matter how the centroids are instantiated. For instance, given $S = \{(0,0),(0,1),(0,2),(0,3),(1,5),(3,2)\}$ and $k = 2$, if the two centroids are initialized with values from the original set of observations S, k-means never encounters the optimum clustering (given by the following partition: $S_1 = \{(0,0),(0,1),(0,2)\}, S_2 = \{(0,3),(1,5),(3,2)\}$). Note that this particular example can be generalized to demonstrate the sub-optimality of heuristic k-means solutions. More precisely, by multiplying the six points in S by ϵ, we can make the k-means algorithm to perform arbitrarily poorly with respect to the global optimal solution (for any real number δ, if $\epsilon = \sqrt{6\delta}$, the difference between any solution returned by k-means and the global optimal one is at least δ). The initial choice of centroids is actually essential for this class of algorithms: if they are randomly chosen among the data points, the ratio between the obtained solution and the optimal one can be made arbitrarily bad (see [29], page 4); on the other hand, choosing a good set of seed points could lead to $\log k$ or even constant factor approximations for the k-means objective (see k-means++ [30] and [31]).

In certain applications, reaching the guaranteed global optimum may be desirable. Also, in order to empirically estimate the "goodness" of any (new or existing) clustering algorithm, one needs to know which is the actual optimum value. For example, we could experimentally check how the k-means algorithm's output is affected by the number n_{iter} of initial random initializations of the centroids. We obtained a 51.43 error rate for $n_{iter} = 1$, 9.16 error rate for $n_{iter} = 10$, 0.66 error rate for $n_{iter} = 100$ and no errors for $n_{iter} = 1000$.

Other applications may require an online version of the algorithms presented, in which new points are added or some points are changed while all possible Voronoi partitions are precomputed for the initial set of points. If changes affect only a small number of points, of order $\mathcal{O}(\log n)$, one solution is to consider all precomputed clusterings and to try all different possibilities for including the new points in each of the clusterings. Notice that each of the algorithms presented in this work enumerate all possible clusterings and require $\mathcal{O}(n)$ operations per sign vector, so the running time is better comparing with running from scratch. Other advantages of this method are: it only needs to keep the sign vectors and it does not require to solve any new system of equations. Unfortunately, it is only useful when there are only a few changes. A little more complicated is when the number of different points grows. Notice that the algorithms do not need the points to be in any order, so new points can be processed at the end. However, this only speeds up the algorithm by a small constant, because it is necessary to place the new points in the previously computed clusters and to find new clusterings solving new systems of equations. Another improvement is by using Gauss triangulation. Fact 6 gives a relation between the sign vectors and solving linear systems. Assuming that there is only one new point in the dataset, it is possible to save all partial Gauss triangulations and use them to solve new systems. This improves the running time by a factor of $O(k)$.

Our contribution is two fold. On one hand, we propose algorithms that improve, in several particular cases of interest, the theoretical computation time of the state-of-the art globally optimum clustering algorithm, and on the other hand, we provide actual implementations that can be freely used by researchers in order to contrast their clustering algorithms against the optimum.

Acknowledgments

We acknowledge Santander Supercomputación support group at the University of Cantabria who provided access to the Altamira Supercomputer at the Institute of Physics of Cantabria (IFCA-CSIC), member of the Spanish Supercomputing Network, for performing simulations. This work was supported by Ministerio de Economia y Competividad (MINECO), Spain [grants number MTM2014-55262-P and MTM2014-55421-P].

References

[1] S. P. Lloyd, Least squares quantization in PCM, IEEE Transactions on Information Theory 28 (2) (1982) 129–137.

[2] J. B. MacQueen, Some methods for classification and analysis of multivariate observations, in: Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1, University of California Press, 1967, pp. 281–297.

[3] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '07, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.

[4] C. Zhang, S. Xia, K-means clustering algorithm with improved initial center, in: Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on, 2009, pp. 790 –792. doi:10.1109/WKDD.2009.210.

[5] R. Ostrovsky, Y. Rabani, L. J. Schulman, C. Swamy, The effectiveness of Lloyd-type methods for the k-means problem, Journal of the ACM (JACM) 59 (6) (2012) 28.

[6] D. Pelleg, A. W. Moore, X-means: Extending k-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, pp. 727–734.

[7] S. J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 2nd Edition, Pearson Education, 2003.

[8] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, Machine Learning 75 (2009) 245–248. doi:http://dx.doi.org/10.1007/s10994-009-5103-0.

[9] P. Drineas, A. Frieze, R. Kannan, S. Vempala, V. Vinay, Clustering large graphs via the singular value decomposition, Machine Learning 56 (2004) 9–33. doi:http://dx.doi.org/10.1023/B:MACH.0000033113.59016.96.

[10] S. Dasgupta, The hardness of k-means clustering, (Technical Report CS2008-0916). University of California. (2008).

[11] M. Mahajan, P. Nimbhorkar, K. Varadarajan, The planar k-means problem is NP-hard, in: Proceedings of the 3rd International Workshop on Algorithms and Computation, WALCOM '09, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 274–285. doi:http://dx.doi.org/10.1007/978-3-642-00202-1_24.

[12] M. Inaba, N. Katoh, H. Imai, Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering (extended abstract), in: Symposium on Computational Geometry, 1994, pp. 332–339. doi:http://doi.acm.org/10.1145/177424.178042.

[13] S. Hasegawa, H. Imai, M. Inaba, N. Katoh, Efficient algorithms for variance-based k-clustering, in: Proceedings of the First Pacific Conference on Computer Graphics and Applications, 1993, pp. 75–89.

[14] J. Canny, Improved algorithms for sign determination and existential quantifier elimination, Comput. J. 36 (5) (1993) 409–418. doi:http://dx.doi.org/10.1093/comjnl/36.5.409.

[15] S. Basu, R. Pollack, M.-F. Roy, A new algorithm to find a point in every cell defined by a family of polynomials, in: Quantifier elimination and cylindrical algebraic decomposition (Linz, 1993), Texts Monogr. Symbol. Comput., Springer, Vienna, 1998, pp. 341–350.

[16] K. Sabo, R. Scitovski, An approach to cluster separability in a partition, Information Sciences 305 (2015) 208 – 218. doi:http://dx.doi.org/10.1016/j.ins.2015.02.011.

[17] F. Rouillier, M.-F. Roy, M. Safey El Din, Finding at least one point in each connected component of a real algebraic set defined by a single equation, J. Complexity 16 (4) (2000) 716–750. doi:http://dx.doi.org/10.1006/jcom.2000.0563.

[18] M. Safey El Din, RAGlib A library for real solving polynomial systems of equations and inequalities, INRIA (2013).
URL http://www-polsys.lip6.fr/~safey/RAGLib/distrib.html

[19] C. Le Guernic, F. Rouillier, M. Safey El Din, On the practical computation of one point in each connected component of a semi-algebraic set defined by a polynomial system of equations and non-strict inequalities., Tech. rep., INRIA (2004).

[20] J. E. Goodman, J. O'Rourke, Handbook of discrete and computational geometry, CRC press, 2010.

[21] R. E. Jensen, A dynamic programming algorithm for cluster analysis, Operations Research 17 (6) (1969) 1034–1057.

[22] H. Wang, M. Song, Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming, The R Journal 3 (2) (2011) 29–33.

[23] L. Bai, J. Liang, C. Sui, C. Dang, Fast global k-means clustering based on local geometrical information, Information Sciences 245 (2013) 168 – 180. doi:http://dx.doi.org/10.1016/j.ins.2013.05.023.

[24] A. Likas, N. Vlassis, J. J. Verbeek, The global k-means clustering algorithm, Pattern Recognition 36 (2) (2003) 451 – 461. doi:http://dx.doi.org/10.1016/S0031-3203(02)00060-2.

[25] H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[26] H. T. Kahraman, S. Sagiroglu, I. Colak, The development of intuitive knowledge classifier and the modeling of domain dependent data, Know.-Based Syst. 37 (2013) 283–295. doi:http://dx.doi.org/10.1016/j.knosys.2012.08.009.

[27] J.-Y. Chen, H.-H. He, A fast density-based data stream clustering algorithm with cluster centers self-determined for mixed data, Information Sciences 345 (2016) 271 – 293. doi:http://dx.doi.org/10.1016/j.ins.2016.01.071.

[28] A. Pakrashi, B. B. Chaudhuri, A Kalman filtering induced heuristic optimization based partitional data clustering, Inf. Sci. 369 (C) (2016) 704–717. doi:10.1016/j.ins.2016.07.057.

[29] P. Awasthi, M. florina Balcan, Center based clustering: A foundational perspective (2013).

[30] D. Arthur, S. Vassilvitskii, K-means++: The advantages of careful seeding, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.
URL http://dl.acm.org/citation.cfm?id=1283383.1283494

[31] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, A. Y. Wu, A local search approximation algorithm for k-means clustering, Computational Geometry 28 (2) (2004) 89 – 112, special Issue on the 18th Annual

Symposium on Computational Geometry - SoCG2002. doi:http://dx.doi.org/10.1016/j.comgeo.2004.03.003.
URL http://www.sciencedirect.com/science/article/pii/S0925772104000215