How much k-means can be improved by using better initialization and repeats?

to appear in *Pattern Recognition*
16.4.2019

Pasi Fränti and Sami Sieranoja*

Machine Learning Group
School of Computing, University of Eastern Finland
P.O. Box 111, FIN-80101 Joensuu, FINLAND
pasi.franti@uef.fi, sami.sieranoja@uef.fi

Abstract: In this paper, we study what are the most important factors that deteriorate the performance of the k-means algorithm, and how much this deterioration can be overcome either by using a better initialization technique, or by repeating (restarting) the algorithm. Our main finding is that when the clusters overlap, k-means can be significantly improved using these two tricks. Simple furthest point heuristic (Maxmin) reduces the number of erroneous clusters from 15% to 6%, on average, with our clustering benchmark. Repeating the algorithm 100 times reduces it further down to 1%. This accuracy is more than enough for most pattern recognition applications. However, when the data has well separated clusters, the performance of k-means depends completely on the goodness of the initialization. Therefore, if high clustering accuracy is needed, a better algorithm should be used instead.

Keywords: Clustering algorithms, k-means, initialization, clustering accuracy, prototype selection.

1. Introduction

K-means (KM) algorithm [1, 2, 3] groups N data points into k clusters by minimizing the sum of squared distances between every point and its nearest cluster mean (*centroid*). This objective function is called *sum-of-squared errors* (SSE). Although k-means was originally designed for minimizing SSE of numerical data, it has also been applied for other objective functions (even some non-numeric).

Sometimes the term *k-means* is used to refer to the clustering problem of minimizing SSE [4, 5, 6, 7]. However, we consider here k-means as an *algorithm*. We study how well it performs as a clustering algorithm to minimize the given objective function. This approach follows the recommendation in [8] to establish a clear distinction between the *clustering method* (objective function) and the *clustering algorithm* (how it is optimized).

In real-life applications, the selection of the objective function is much more important. Clustering results depend primarily on the selected objective function, and only secondarily on the selected algorithm. Wrong choice of the function can easily reverse the benefit of a good algorithm so that a proper objective function with a worse algorithm can provide better clustering than good algorithm with wrong objective function. However, it is an open question how much clustering results are biased because of using an inferior algorithm.

There are other algorithms that are known, in many situations, to provide better clustering results than k-means. However, k-means is popular for good reasons. First, it is simple to implement. Second, people often prefer to use an extensively studied algorithm whose limitations are known rather than a potentially better, but less studied, algorithm that might have unknown or hidden limitations. Third, the local fine-tuning

capability of k-means is very effective, and for this reason, it is also used as part of better algorithms such as the genetic algorithm [9, 10], random swap [11,12], particle swarm optimization [13], spectral clustering [14], and density clustering [15]. Therefore, our results can also help better understand those more complex algorithms that rely on the use of k-means.

K-means starts by selecting *k* random data points as the initial set of centroids, which is then improved by two subsequent steps. In the *assignment step*, every point is put into the cluster of the nearest centroid. In the *update step*, the centroid of every cluster is recalculated as the mean of all data points assigned to the cluster. Together, these two steps constitute one *iteration* of k-means. These steps fine-tune both the cluster borders and the centroid locations. The algorithm is iterated a fixed number of times, or until convergence (no further improvement is obtained). MacQueen also presented *sequential* variant of k-means [2], where the centroid is updated immediately after every single assignment.

K-means has excellent fine-tuning capabilities. Given a rough allocation of the initial cluster centroids, it can usually optimize their locations locally. However, the main limitation of k-means is that it rarely succeeds in optimizing the centroid locations globally. The reason is that the centroids cannot move between the clusters if their distance is big, or if there are other stable clusters in between preventing the movements, see Fig. 1. The k-means result therefore depends a lot on the initialization. Poor initialization can cause the iterations to get stuck into an inferior local minimum.

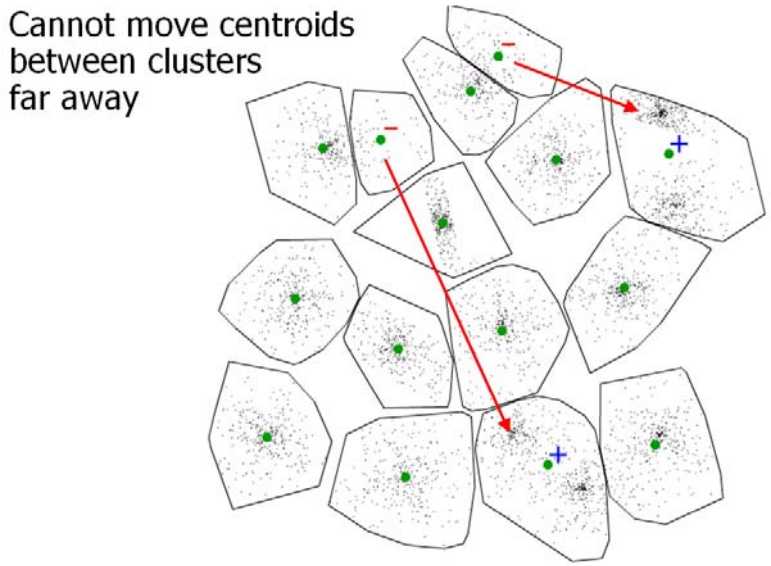

Figure 1: K-means is excellent in fine-tuning cluster borders locally but fails to relocate the centroids globally. Here a minus sign (-) represents a centroid that is not needed, and a plus sign (+) a cluster where more centroids would be needed. K-means cannot do it because there are stable clusters in between.

Fortunately, finding the exact optimum is not always important. In pattern recognition applications, the goal can be merely to model the distribution of the data, and the clustering result is used as a part in a more complex system. In [16], the quality of the clustering was shown not to be critical for the speaker recognition performance when any reasonable clustering algorithm, including repeated k-means, was used.

However, if the quality of clustering is important then k-means algorithm has problems. For example, if we need to solve the number of clusters, the goodness of the algorithm matters much more. Experiments with three different indexes (WB, DBI, Dunn) have shown that k-means rarely achieves the correct number of clusters whereas random swap succeeded in most cases [17]. Similar observations were made with stability-based approach in [18].

To compensate for the mentioned weaknesses of k-means, two main approaches have been considered: (1) using a better initialization, (2) repeating k-means several times by different initial solution. Numerous initialization techniques have been presented in the literature, including the following:

- Random points
- Furthest point heuristic
- Sorting heuristic
- Density-based
- Projection-based
- Splitting technique

Few comparative studies exists [19-22], but there is no consensus of which technique should be used. A clear state-of-the-art is missing. Pena et al. [19] studied four basic variants: *random centroids* [1] and *MacQueen's variant* of it [2], *random partition* and Kaufman's variant of the *Maxmin heuristic* [23]. Their results show that random partition and Maxmin outperform the random centroid variants with the three datasets (Iris, Ruspini, Glass).

He et al. [20] studied random centroids, random perturbation of the mean [24], greedy technique [25], Maxmin [26], and Kaufman's variant of Maxmin [23]. They observed that the Maxmin variants provide slightly better performance. Their argument is that the Maxmin variants are based on distance optimization, which tends to help k-means provide better cluster separation.

Steinley and Brusco [21] studied 12 variants including complete algorithms like *agglomerative clustering* [27] and *global k-means* [28]. They ended up recommending these two algorithms and Steinley's variant [29] without much reservation. The first two are already complete stand-alone algorithms themselves and not true initialization techniques, whereas the last one is a trivial improvement of the random partition.

Steinley and Brusco also concluded that agglomerative clustering should be used only if the size, dimensionality or the number of clusters is big; and that global k-means (GKM) [28] should be used if not enough memory to store the N^2 pairwise distances. However, these recommendations are not sound. First, agglomerative clustering can be implemented without storing the distance matrix [30]. Second, GKM is extremely slow and not practical for bigger datasets. Both these alternatives are also standalone algorithms and they provide better clustering even without k-means.

Celebi et al. [22] performed the most extensive comparison so far with 8 different initialization techniques on 32 real and 12,228 synthetic datasets. They concluded that random centroids and Maxmin often perform poorly and should not be used, and that there are significantly better alternatives with comparable computational requirements. However, their results do not clearly point out a single technique that would be consistently better than others.

The detailed results in [22] showed that a sub-sampling and repeat strategy [31] performs consistently in *the best group* and k-means++ performs *generally well*. For

small datasets Bradley's sub-sampling strategy or greedy variant of k-means++ was recommended. For large data, split-based algorithm was recommended.

The second major improvement, besides the initializations, is to repeat k-means [32]. The idea is simply to restart k-means several times from different initial solution to produce several candidate solutions, and then keeping the best result found as the final solution. This approach requires that the initialization technique produces different starting solutions by involving some randomness in the process. We call this variant *repeated k-means* (RKM). The number of repeats is typically small like R=20 in [33].

Many researchers consider the repeats as an obvious and necessary improvement to the k-means at the cost of increased processing time. Bradley & Fayyad [31] used slightly different variant by combining the repeats and sub-sampling to avoid the increase in the processing time. Besides these papers, it is hard to find any systematic study how the repeats affect on the k-means. For example, none of the review papers investigate the effect of the repeats on the performance.

To sum up, existing literature provides merely relative comparisons between the selected initialization techniques. They lack clear answers of the significance of the results, and present no analysis on which type of data the techniques work and fail. Many of the studies also use classification datasets, which have limited suitability for studying the clustering performance.

We made a brief survey about how recent research papers apply k-means. Random centroids [34, 35, 5] seems to be the most popular initialization method, along with k-means++ [36, 33, 6]. Some papers do not specify how they initialize [37], or it had to be concluded indirectly. For example, Boutsidis [5] used the default method available in MATLAB, which was random centroids in the 2014a version and k-means++ starting from the 2014b version. The method in [38] initializes both the centroids and the partition labels at random. However, as they apply the centroid step first, the random partition is effectively applied.

The number of k-means repeats varies from a relatively small amount of 10-20 [5, 35, 33] to a relatively high value of 100 [36]. The most extreme example is [34] where 20 hours time limit is applied. Although they stop iterating if the running time grows twice as that of their proposed algorithm, it is still quite extensive. Several papers do not repeat k-means at all [37, 6, 7].

The choice of the initialization and the number of repeats might also vary depending on the motivation. The aim of using k-means can be to have a good clustering result, or to provide merely a point of comparison. In the first case, all the good tricks are used, such as more repeats and better initialization. In the second case, some simpler variant is more likely applied. A counter-example is in [34] where serious efforts seem to be made to ensure all algorithms have the best possible performance.

In this paper we study the most popular initialization heuristics. We aim at answering the following questions. First, to what extent k-means can be improved by a better initialization technique? Second, can the fundamental weakness of k-means be eliminated simply by repeating the algorithm several times? Third, can we predict under which conditions k-means works, and which it fails?

In a recent study [39], it was shown that k-means performs poorly when the clusters are well separated. Here we will answer how much a better initialization or repeats can compensate for this weakness. We will also show that dimensionality does not matter

for most variants, and that unbalance of cluster sizes deteriorates the performance of most initializations.

The rest of the paper is organized as follows. In Section 2, we define the methodology and data. We also give brief review of the properties of the standard k-means algorithm. Different initialization techniques are then studied in Section 3. Experimental analysis is performed in Section 4, and conclusions are drawn in Section 5.

2. Performance of k-means

Following the recommendation of Jain [8], we make a clear distinction between the clustering method and algorithm. *Clustering method* refers to the objective function, and *clustering algorithm* to the process optimizing it. Without this distinction, it would be easy to draw wrong conclusions.

For example, k-means has been reported to work poorly with unbalanced cluster sizes [40], and that it can cause large clusters to be wrongly split and smaller clusters wrongly merged [41]. These observations themselves are correct but they miss the root cause, which is the SSE objective function. Even an optimal algorithm minimizing SSE would end up with the same incorrect result. Such observations therefore relate to the objective function, and not to the k-means *algorithm*.

Fig. 2 demonstrates the situation. An algorithm minimizing SSE would find spherical clusters regardless of the data. If the data contain non-spherical clusters, they would be divided into spherical sub-clusters, usually along the direction of the highest variance. Clusters of variable sizes would also cause large clusters to be split, and smaller ones to be merged. In these cases, if natural clusters are wanted, a better clustering result could be achieved by using an objective function based on *Mahalanobis distance* [42] or *Gaussian mixture model* [43] instead of SSE.

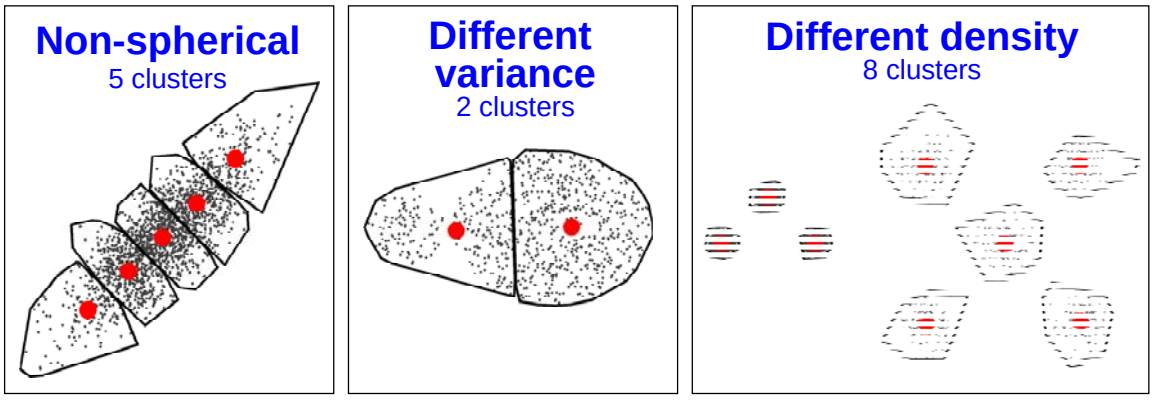

Figure 2: Three examples of clustering result when using SSE cost function. Gaussian cluster is split into several spherical clusters (left); mismatch of the variance causes the larger cluster to be split (middle); mismatch of the cluster sizes does not matter if the clusters are well-separated.

2.1 Datasets

In this paper, we focus on the algorithmic performance of k-means rather than the choice of the objective function. We use the *clustering basic benchmark* [39] as all these datasets can be clustered correctly with SSE. Therefore, any clustering errors made by k-means must originate from the properties of the algorithm, and not from the choice of wrong objective function. The datasets are summarized in Table 1. They are designed to vary the following properties as defined in [39]:

- Cluster overlap
- Number of clusters
- Dimensionality
- Unbalance of cluster sizes

Table 1: Basic clustering benchmark [39].
The data is publicly available here: http://cs.uef.fi/sipu/datasets/

Dataset	Varying	Size	Dimensions	Clusters	Per cluster
A	Number of clusters	3000-7500	2	20-50	150
S	Overlap	5000	2	15	333
Dim	Dimensions	1024	32-1024	16	64
G2	Dimensions + overlap	2048	2-1024	2	1024
Birch	Structure	100,000	2	100	1000
Unbalance	Balance	6500	2	8	100-2000

2.2 Methodology

To measure the success of the algorithm, the value of the objective function itself is the most obvious measure. Existing literature reviews of k-means use either SSE [19, 22], or the deviation of the clusters [20], which is also a variant of SSE. It is calculated as:

$$\text{SSE} = \sum_{i=1}^{N} \left\| x_i - c_j \right\|^2 \tag{1}$$

where x_i is a data point and c_j is its nearest centroid. In [39], SSE is also measured relative to the SSE-value of the ground truth solution (SSE_{opt}):

$$\varepsilon\text{-ratio} = \frac{\left(SSE - SSE_{opt}\right)}{SSE_{opt}} \tag{2}$$

If the ground truth is known, external indexes such as *adjusted Rand index* (ARI), *Van Dongen* (VD), *variation of information* (VI) or *normalized mutual information* (NMI) can also be used [22]. A comparative study of several suitable indexes can be found in [44]. The number of iterations have also been studied in [19, 22], and the time complexities reported in [22].

The problem of SSE, and most of the external indexes, is that the raw value does not tell how significant the result is. We therefore use *Centroid Index* (CI) [45], which indicates how many cluster centroids are wrongly located. Specifically, the value CI=0 implies that the clustering structure is correct with respect to the ground truth.

An example is shown in Fig. 3, where k-means provides $\text{SSE}=3.08\cdot10^{10}$, which is 52% higher than that of the ground truth. But what do these numbers really mean? How significant is the difference? On the other hand, the value CI=4 tells that exactly four real clusters are missing a centroid.

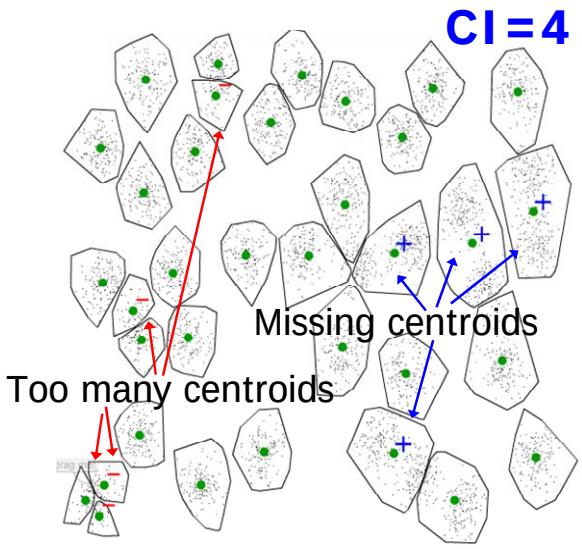

Figure 3: Performance of k-means with the A2 dataset: CI=4, SSE=3.08 ($\cdot 10^{10}$), ε=0.52.

Based on CI, a *success rate* (%) was also defined in [39] to measure the probability of finding the correct clustering. For example, when running k-means 5000 times with dataset A2 (Fig. 3), CI=0 was never reached, and thus, its success rate is 0%. Another example with dataset S2 (Fig. 4) results in success rate of 1/6 = 17%.

The success rate has an important implication. Any value higher than 0% indicates that the correct clustering can be found simply by repeating k-means. For a success rate p, the expected number of repeats is $1/p$. For instance, p=50% indicates that expected number of repeats is 2; and p=1% indicates 100 repeats. Even with as low value as p =0.1% the correct solution is expected to be found in 1000 repeats. This is time consuming, but feasible. However, for some of our datasets the success rate is so low that the number repeats would be unreasonably high. For example, even 200,000 repeats produces 0% success rate in our experiments with some datasets.

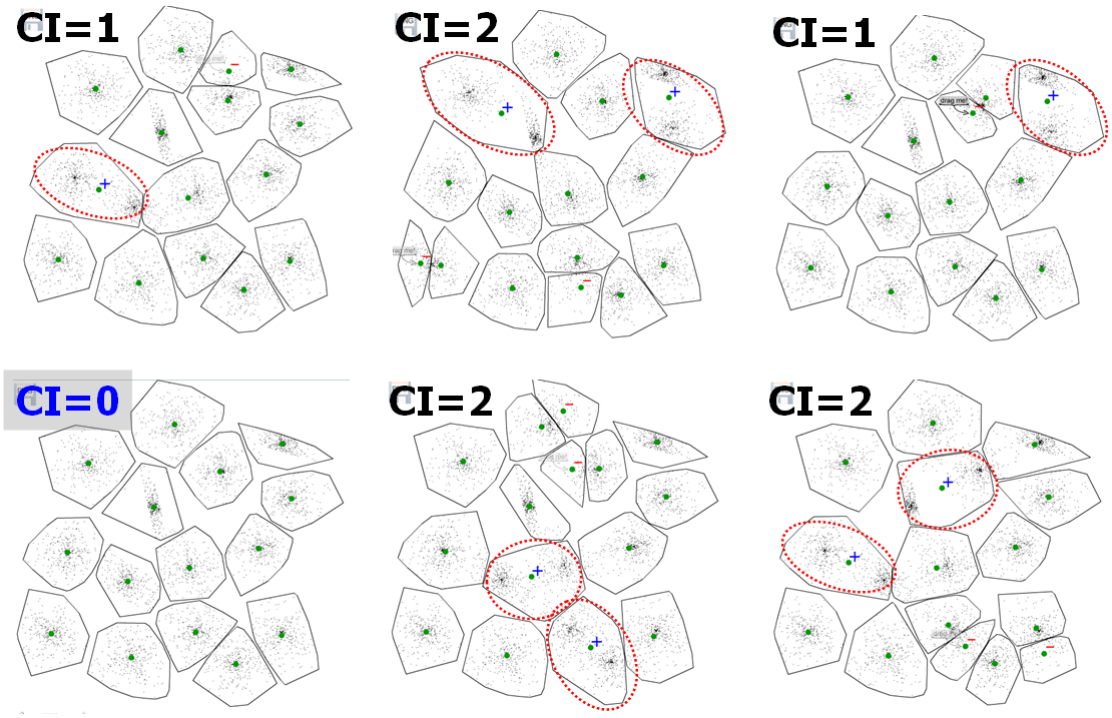

Figure 4: Centroid index measures how many real clusters are missing a centroid (+), or how many centroids are allocated to wrong cluster (-).
Six examples are shown for S2 dataset.

2.3 Properties of k-means

We next briefly summarize the main properties of the k-means algorithm. Generally the clustering problem is the easier the more the clusters are separated. However, in [39] it was found that for k-means it is just the opposite; the less overlap the worse the clustering performance, see Fig. 5. This is a fundamental weakness of the k-means algorithm.

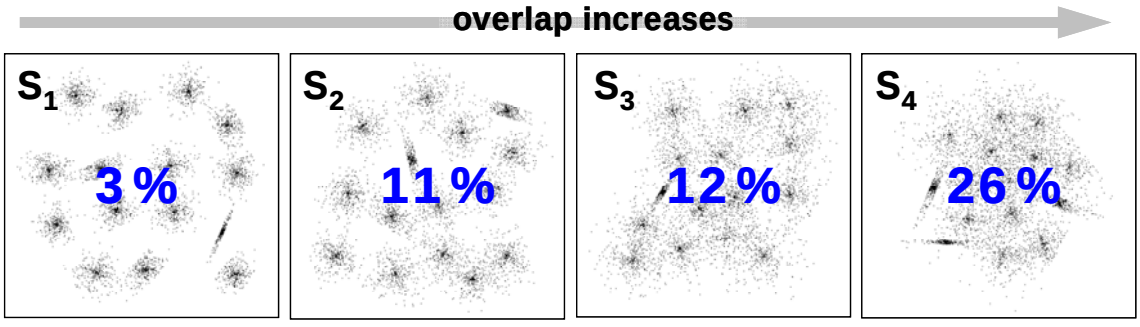

Figure 5: Success rate (%) of k-means, measured as the probability of finding correct clustering, improves when the cluster overlap increases.

In [39], it was also found that the number of errors has linear dependency on the number of clusters (k). For example, the CI-values for the A sets with k=20, 35, 50 clusters were measured as CI=2.5, 4.5, 6.5, respectively. The relative CI-values (CI/k) correspond to a constant of 13% of centroids being wrongly located. Results with the subsets of Birch2 (varying k from 1 to 100) converge to about 16% when k approaches to 100, see Fig. 6.

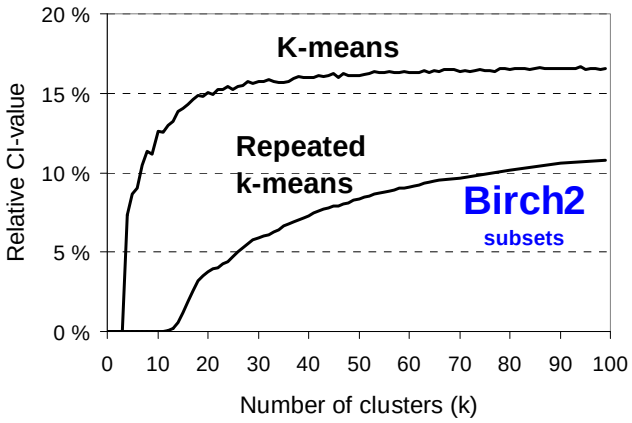

Figure 6: CI-value of k-means increases linearly with *k*, and relative CI converges to 16% with the Birch2 subsets.

Two series of datasets are used to study the dimensionality: DIM and G2. The DIM sets have 16 well separated clusters in high-dimensional space with dimensionality varying from D=32 to 1024. Because of clear cluster separation, these datasets should be easy for any good clustering algorithm to reach CI=0 and 100% success rate. However, k-means again performs poorly; it obtains the values CI=3.6 and 0% success rate regardless of the dimensionality. The reason for the poor performance is again the lack of cluster overlap, and not the dimensionality.

The results with the G2 sets confirmed the dependency between the dimensionality and the success rate. We allocated four centroids with 3:1 unbalance so that the first cluster had three centroids and the second only one. We then ran k-means and checked whether it found the expected 2:2 allocation by moving one of the three centroids to the second group. The results in Fig. 7 show that the overlap is the mediating factor for the success rate: the more overlap, the lower the success rate of k-means.

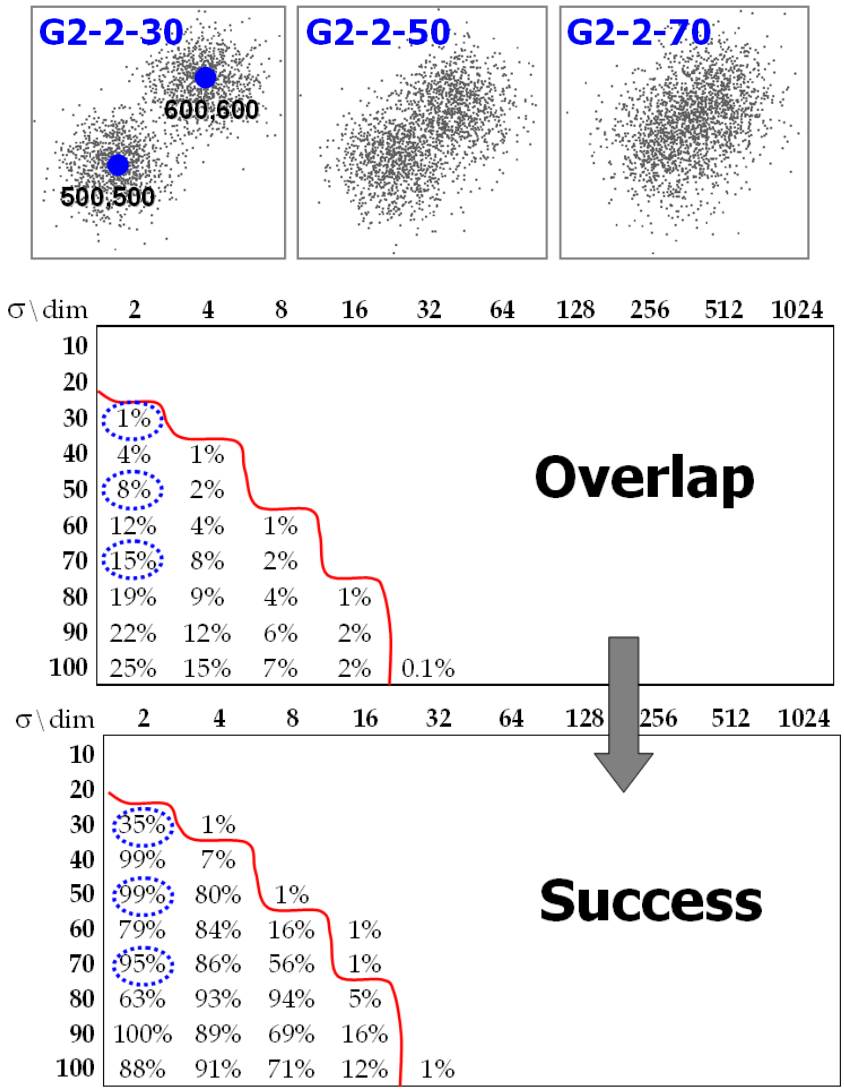

Figure 7: The effect of overlap for the success of k-means with the G2 datasets. The numbers circled with blue are for the three sample datasets shown above. The dataset names are coded as G2-DIM-SD, where DIM refers to the dimensions and SD to the standard deviation; the higher the SD, the more the two clusters overlap.

The cluster size unbalance was also shown in [39] to result in poor performance. The main reason for this was the random initialization, which cannot pick the initial centroids in a balanced way. Another reason was the k-means iterations which fail to improve the initial solution due to lack of cluster overlap.

The effect of the different properties of data on k-means can be summarized as follows:

Property:	**Effect:**
Cluster overlap	Overlap is good
Number of clusters	Linear dependency
Dimension	No direct effect
Unbalance	Bad

3. K-means initialization techniques

Next we study how much these problems of k-means can be solved by the following two improvements:

- Better initialization
- Repeating k-means

K-means is a good algorithm for local fine-tuning but it has serious limitation to relocate the centroids when the clusters do not overlap. It is therefore unrealistic to expect the clustering problem to be solved simply by inventing a better *initialization* for k-means. The question is merely, how much a better initialization can compensate for the weakness of k-means.

Any clustering algorithm could be used as an initialization technique for k-means. However, solving the location of initial centroids is not significantly easier than the original clustering problem itself. Therefore, for an algorithm to be considered as *initialization* technique for k-means, in contrast to being a standalone algorithm, we set the following requirements:

1. Simple to implement
2. Lower (or equal) time complexity than k-means
3. No additional parameters

First, the algorithm should be trivial, or at least very easy to implement. Measuring implementation complexity can be subjective. The number of functions and the lines of code were used in [16]. Repeated k-means was counted to have 5 functions and 162 lines of C-code. In comparison, random swap [11,12], fast agglomerative clustering variant [30], and sophisticated splitting algorithm [46] had 7, 12 and 22 functions, and 226, 317 and 947 lines of codes, respectively. Random initialization had 2 functions and 26 lines of code.

Second, the algorithm should have lower or equal time complexity compared to k-means. Celebi [22] categorizes the algorithms to linear, log-linear and quadratic based on their time complexities. Spending quadratic time cannot be justified as the fastest agglomerative algorithms are already working in close to quadratic time [30]. A faster $O(N \log N)$ time variant also exists [47] but it is significantly more complex to implement and requires to calculate *k-near neighbors* (KNN). K-means requires $O(gkN)$ time, where g is the number of iterations and typically varies from 20 to 50.

The third requirement is that the algorithm should be free of parameters; others than k. For instance, there are algorithms [48, 25] that select the first centroid using some simple rule, and the rest greedily by cluster growing, based on whether the point is within a given distance. Density-connectivity criterion was also used in [49]. Nevertheless, this approach requires one or more threshold parameters.

The most common heuristics are summarized in Table 2. We categorize them roughly into *random*, *furthest point*, *sorting*, and *projection-based* heuristics. Two standalone algorithms are also considered: *Luxburg* [50] and *Split* algorithm. For a good review of several others we refer to [51].

Table 2: Summary of the initialization techniques compared in this paper. *Time* refers to the average processing time with the A3 dataset (N=7500, k=50). *Randomized* refers to whether the technique include randomness naturally. Randomness will be needed for the repeated k-means variant later.

Technique	Ref.	Complexity	Time	Randomized	Parameters
Random partitions	[3]	O(N)	10 ms	Yes	-
Random centroids	[1,2]	O(N)	13 ms	Yes	-
Maxmin	[54]	O(kN)	16 ms	Modified	-
kmeans++	[59]	O(kN)	19 ms	Yes	-
Bradley	[31]	O(kN+Rk^2)	41 ms	Yes	R=10, s=10%
Sorting heuristic	[62]	O(N log$N$)	13 ms	Modified	-
Projection-based	[72]	O(N log$N$)	14 ms	Yes	-
Luxburg	[50]	O(kN log$k$)	29 ms	Yes	-
Split	[68, 46]	O(N log$N$)	67 ms	Yes	k=2

3.1. Random centroids

By far the most common technique is to select k random data objects as the set of initial centroids [1,2]. It guarantees that every cluster includes at least one point. We use *shuffling method* by swapping the position of every data point with another randomly chosen point. This takes O(N) time. After that, we take the first k points from the array. This guarantees that we do not select the same point twice, and that the selection is independent on the order of the data. For the random number generator we use the method in [52]. We refer to this initialization method as *random centroids*.

Slightly different variant in [2] selects simply the first k data points. This is the default option in the Quick Cluster in IBM SPSS Statistics [53]. If the data is in random order the result is effectively the same as random centroids, except that it always provides the same selection.

We note that the randomness is actually a required property for the *repeated* k-means variant. This is because we must be able to produce different solutions at every repeat. Some practitioners might not like the randomness and prefer deterministic algorithms always producing the same result. However, both of these goals can actually be achieved if so wanted. We simply use pseudo-random number generator with the same *seed number*. In this way, single runs of k-means will produce different result but the overall algorithm still produces always the same result for the same input.

3.2. Random partitions

An alternative to random centroids is to generate random partitions. Every point is put into a randomly chosen cluster and their centroids are then calculated. The positive effect is that it avoids selecting outliers from the border areas. The negative effect is that the resulting centroids are concentrated in the central area of the data due to the averaging. According to our observations, the technique works well when the clusters are highly overlapped but performs poorly otherwise, see Fig. 8.

According to [19], the random partition avoids the worst case behavior more often than the random centroids. According to our experiments, this is indeed the case but only when the clusters have high overlap. The behavior of the random partition is also more deterministic than that of random centroids. This is because the centroids are practically always near the center of the data. Unfortunately, this also reduces the benefits of the

repeated k-means because there is very little variation in the initial solutions, and therefore, also the final solutions often become identical.

Steinley [29] repeats the initialization 5000 times and selects the one with the smallest SSE. However, repeating only the initialization does not fix the problem. Instead, it merely slows down the initialization because it takes $5000 \cdot N$ steps, which is typically much more than $O(kN)$.

Thiesson et al. [24] calculate the mean point of the data set and then add random vectors to it. This effectively creates initial centroids like a cloud around the center of the data, with very similar effect as the random partition. The size of this cloud is a parameter. If it is set up high enough, the variant becomes similar to the random centroids technique, with the exception that it can select points also from empty areas.

Fig. 8 shows the effect of the random partition and Steinley's variant. Both variants locate the initial centroids near the center of the data. If the clusters have low overlap, k-means cannot provide enough movement and many of the far away clusters will lack centroids in the final solution.

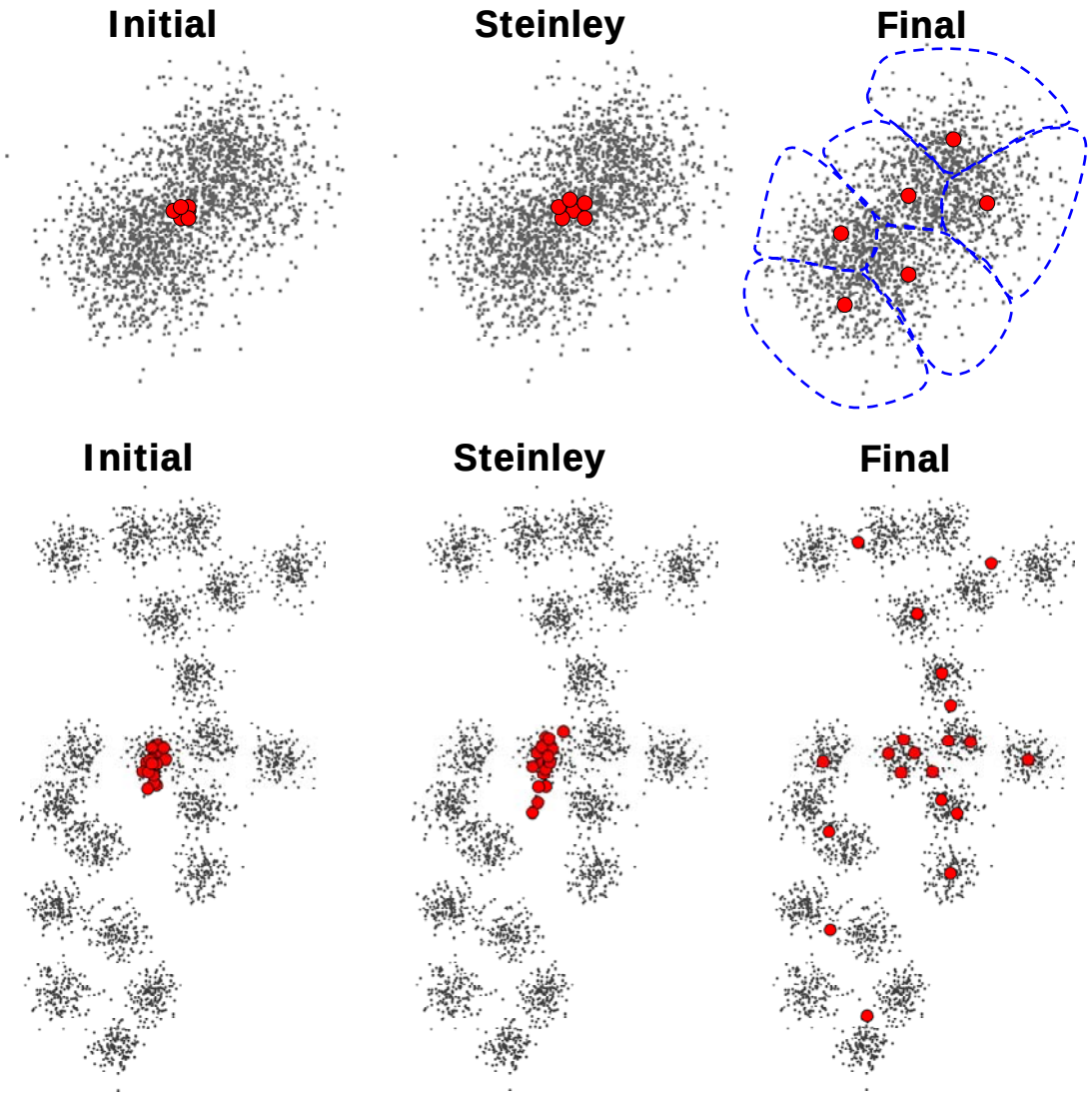

Figure 8: Initial centroids created by random partition (left), by Steinley's variant (middle), and the final result after the k-means iterations (right).

3.3. Furthest point heuristic (Maxmin)

Another popular technique is the furthest point heuristic [54]. It was originally presented as standalone 2-approximate clustering algorithm but has been widely used to initialize k-means. It selects an arbitrary point as the first centroid and then adds new centroids one by one. At each step, the next centroid is the point that is furthest (max) from its nearest (min) existing centroid. This is also known as *Maxmin* [19, 22, 55, 21].

Straightforward implementation requires $O(k^2N)$ time but it can be easily reduced to $O(kN)$ as follows. For each point, we maintain pointer to its nearest centroid. When adding a new centroid, we calculate the distance of every point to this new centroid. If the new distance is smaller than to the previous nearest, then it is updated. This requires

N distance calculations. The process is repeated *k* times, and the time complexity is therefore O(*kN*) in total, which is the same as one iteration of k-means. Further speedup can be achived by searching for the furthest point in just a subset of the data [56].

There are several alternative ways to choose the first centroid. In the original variant the selection is arbitrary [54]. In [55], the furthest pair of points are chosen as the first two centroids. Another variant selects the one with maximum distance to the origin [57] because it is likely to be located far from the center. Maximum density has also been used [51, 58].

K-means++ [59] is a randomized variant of the furthest point heuristic. It chooses the first centroid randomly and the next ones using a weighted probability $p_i = cost_i / \mathrm{SUM}(cost_i)$, where $cost_i$ is the squared distance of the data point x_i to its nearest centroids. This algorithm is an O(log *k*)-approximation to the problem. We also implement k-means++ for our tests because of its popularity.

Chiang and Mirkin [55] recalculate all the centroids after updating the partitions, and the next centroid is selected as the farthest from the recently added centroid. Slightly more complex variant [23] selects the point that decreases the objective function most. It requires calculation of all distances between every pair of points, which takes $O(N^2)$ time. Thus, it does not qualify our criteria for k-means initialization. With the same amount of computation we can already run implement agglomerative clustering algorithm.

Other authors also weight the distances by the density of the point [51, 58]. This reduces the probability that outliers are selected. Erisoglu et al. [60] use cumulative distance to all previous centroids instead of the maxmin criterion. However, this performs worse because it can easily choose two nearby points provided that they have large cumulative distance to all other centroids [61].

We use here a variant that selects the first point randomly [59, 54]. This adds randomness to the process as required by the repeated k-means variant. The next centroids we select using the original maxmin criterion, i.e. choosing the point with biggest distance to its nearest centroid.

Maxmin technique helps to avoid worst case behavior of the random centroids, especially when the cluster sizes have serious unbalance. It also has tendency to pick up outlier points from the border areas, which leads to slightly inferior performance in the case of datasets with high overlap (S3 and S4). However, k-means usually works better with such datasets [39], which compensates for the weakness of Maxmin. Fig. 9 demonstrates the performance of the Maxmin technique.

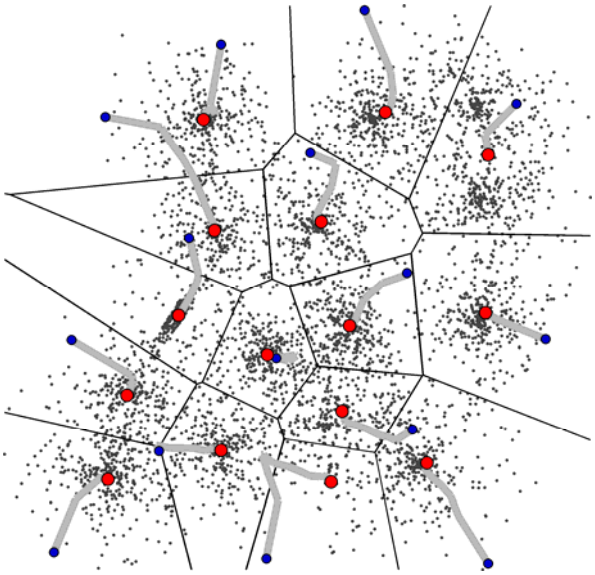

Figure 9: Example of the maxmin heuristic for *S3* dataset. The blue dots are the initial and the red dots the final centroids. The trajectories show their movement during the k-means iterations.

3.4. Sorting heuristics

Another popular technique is to sort the data points according to some criterion. Sorting requires O(N log N) time, which is less than that of one k-means iteration, O(kN), assuming that log$N \leq k$. After sorting, k points are selected from the sorted list using one of the following heuristics:

- First k points.
- First k points while disallowing points closer than ε to already chosen centroids.
- Every $(N/k)^{th}$ point (uniform partition)

For the sorting, at least the following criteria have been considered:

- Distance to center point [62]
- Density [63, 21]
- Centrality [64]
- Attribute with the greatest variance [65]

Hartigan & Wong [62] sort the data points according to their distance to the center of the data. The centroids are then selected as every N/k^{th} point in this order. We include this variant in our tests. To have randomness, we choose a random data point as a reference point instead of the center. This heuristic fulfills our requirements: it is fast, simple, and requires no additional parameters.

Astrahan [63] calculates *density* as the number of other points within a distance d_1. First centroid is the point with the highest density, and the remaining k-1 centroids are chosen at a decreasing order, with the condition that they are not closer than distance d_2 from an already chosen centroid. Steinley [21] recommends using the average pairwise distance (pd) both for d_1 and d_2. This makes the technique free from parameters but it is still slow, O(N^2) time, for calculating the pairwise distances.

It would be possible to simplify this technique further and use random sampling: select N pairs of points, and use this subsample to estimate the value of pd. However, the

calculation of the densities is still the bottleneck, which prevents this approach from meeting the requirements for k-means initialization as such.

Cao et al. [64] proposed a similar approach. They use a primary criterion (*cohesion*) to estimate how central a point is (how far from boundary). Secondary threshold criterion (*coupling*) is used to prevent centroids from being neighbors.

Al-Daoud [65] sorts the data points according to the dimension with the largest variance. The points are then partitioned into k equal size clusters. Median of each cluster is selected instead of the mean. This approach belongs to a more general class of projection-based techniques where the objects are mapped to some linear axis such as diagonal or principal axis.

The sorting heuristic would work if the clusters were well separated, and all have different criterion value (such as the distance from center point). This actually happens with the very high dimensional DIM datasets in our benchmark. However, with most other datasets the clusters tend to be randomly located in respect to the center point, and it is unlikely that all the clusters would have different criterion values. What happens in practice, is that the selected centroids are just random data points in the space, see Fig. 10.

3.5. Projection-based heuristics

Sorting heuristics can also be seen as a projection of the points into a one-dimensional (non-linear) curve in the space. Most criteria would just produce an arbitrary curve connecting the points randomly, and lacking convexity or any sensible shape. However, several linear projection-based techniques have been considered in the literature:

- Diagonal axis [65]
- Single axis [66, 67]
- Principal axis [68, 69, 46, 70, 67, 71]
- Two random points [72]
- Furthest points [72]

After the projection is performed, the points are partitioned into k equal size clusters similarly as with the sorting-based heuristics.

Yedla et al. [66] sort the points according to their distance to origin, and then select every N/k^{th} point. If the origin is the center of data, this is essentially the same technique as in [62]. If the attributes are non-negative, then this is essentially the same as projecting the data to the diagonal axis. Such projection is trivial to implement by calculating the average of the attribute values. It has also been used for speeding-up nearest neighbor searches in clustering in [73].

Al-Daoud [65] sorts the points according to the dimension with the largest variance. The points are then partitioned into k equal size clusters. Median of each cluster is selected instead of the mean. This adapts to the data slightly better than just using the diagonal.

A more common approach is to use *principal axis*, which is the axis of projection that maximizes variance. It has been used effectively in divisive clustering algorithms [68, 69, 46, 70, 67, 71]. Calculation of the principal axis takes $O(DN)$-$O(D^2N)$ depending on the variant [46]. A more complex *principal curve* has also been used for clustering [74].

We consider two simple variants: *random* and *two furthest points* projection as studied in [72]. The first heuristic takes two random data points and projects to the line passing

by these two reference points. The key idea is the randomness; single selection may provide poor initialization but when repeating several times, the chances to find one good initialization increases, see Fig. 11. We include this technique into our experiments and refer to it as *Projection*.

The second heuristic is slightly more deterministic but still random. We start by selecting a random point, and calculate its furthest point. The projection axis is the line passing by these two reference points. We again rely on randomness, but now the choices are expected to be more sensible, potentially providing better results using fewer trials. However, according to [72] this variant does not perform any better than the simpler random heuristic.

Projection works well if the data has one-dimensional structure. In [72], *projective value* is calculated to estimate how well a given projection axis models the data. From our data, Birch2 and G2 have high projective values and suitable for projection-based technique. However, with all other datasets, the projection does not make much more sense than the naïve sorting heuristics, see Fig. 10.

We also note that projection-based techniques also generalize to segmentation-based clustering, where *k*-1 dividing planes are searched simultaneously using dynamic programming [75, 74]. These clustering results usually require fine-tuning by k-means at the final step, but nevertheless, they are standalone algorithms.

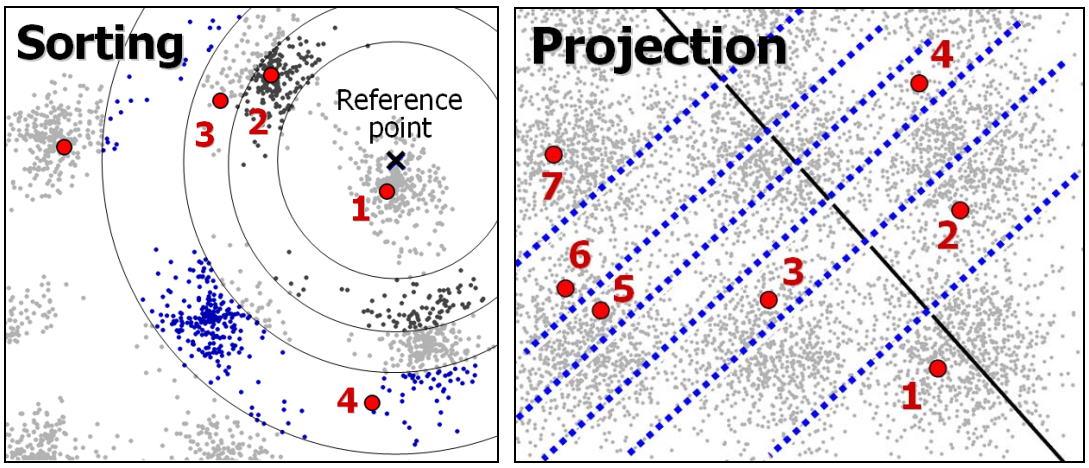

Figure 10: Examples of sorting and projection-based techniques.

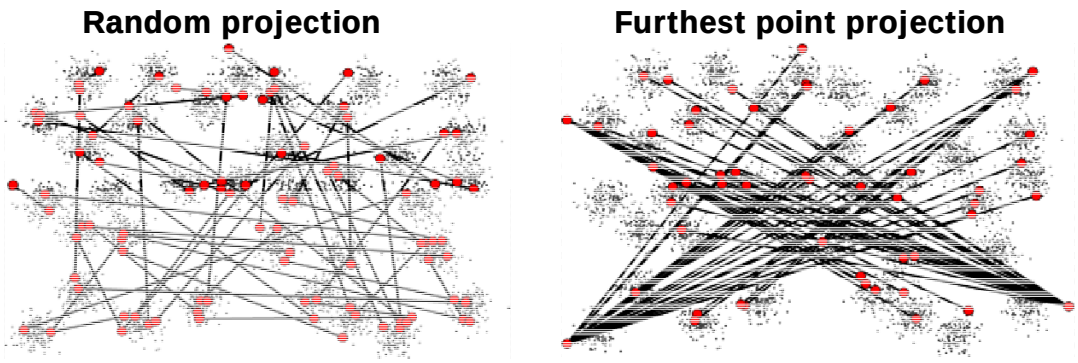

Figure 11: Examples of the two projection-based heuristics for A2 dataset: random points (left), and the furthest point projections (right) [72].

3.6. Density-based heuristics

Density was already used both with the furthest point and the sorting heuristics, but the concept deserves a little bit further discussion. The idea of using density itself is appealing but it is not trivial how to calculate the density, and how to use it in clustering. Especially since the initialization technique should be fast and simple.

The main bottleneck of the algorithms is how to calculate the density is estimated for the points. There are three common approaches for this:

- Buckets
- ε-radius circle
- k-nearest neighbors (KNN)

The first approach divides the space by a regular grid, and counts the frequency of the points in every bucket [76]. The density of a point is then inherited from the bucket it is in. This approach is feasible in low-dimensional space but would become impractical in higher-dimensional spaces. In [61], the problem is addressed by processing the dimensions independently in a heuristic manner. Other authors have used *kd-tree* [51, 57] or space-filling curve [77] to partition the space into buckets containing roughly the same number of points. In [51, 57], the number of buckets is $10 \cdot k$.

The other two approaches calculate the density for every point individually. The traditional one is to define a neighborhood using a cutoff threshold (ε-radius), and then counting the number of other points within this neighborhood [63, 21, 64, 78]. The third approach finds the *k-nearest neighbors* of a point [79], and then calculates the average distance to the points within this neighborhood. Lemke and Keller calculate the density between every pair of points [49].

The bottleneck of the last two approaches is that we need to find the points that are within the neighborhood. This requires $O(N^2)$ distance calculations in both cases. Several speed-up techniques and approximate variants exist [80, 81] but none that is both fast and simple to implement. Calculating density values only for a subset of size SQRT(N) would reduce the complexity to $O(N^{1.5})$ depending whether the distances are calculated to all points or only within the subset. In [82], density is calculated in each dimension separately, and then final approximation is obtained by summing up the individual densities. This allows rapid $O(DN)$ time estimation with more accurate estimation than the sub-sampling approach.

Once calculated, the density can be used jointly with the furthest point heuristic, with the sorting heuristic, or some of their combination. For example, in [51] the furthest point heuristic was modified by weighting the distance by its density so that outliers are less likely chosen. The *density peaks* algorithm in [78] finds for every point its nearest neighbor with higher density. It then applies sorting heuristic based on one of the two features: density and the distance to its neighbor. The method works as a standalone algorithm and does not require k-means at all.

Luxburg [50] first selects k*SQRT(k) preliminary clusters using k-means and then eliminates the smallest ones. After this, the furthest point heuristic is used to select the k clusters from the preliminary set of clusters. When minimizing SSE, the size of the clusters correlates to their density. Thus, Luxburg's technique indirectly implements a density-based approach which favors clusters of high density. We include this technique in our experiments although it does not satisfy our simplicity criterion.

We also note that there are several standalone clustering algorithms based on density [78, 49, 83, 84]. However, they do not fit to our requirements for speed and simplicity. If combined with the faster density estimation in [82], some of these techniques could be made competitive also in speed.

3.7. Splitting algorithm

Split algorithm puts all points into a single cluster, and then iteratively splits one cluster at a time until k clusters are reached. This approach is seemingly simple and tempting to consider for initializing k-means. However, there are two non-trivial design choices to make: which cluster to split, and how to split it. We therefore consider split mainly as a standalone algorithm, but discuss briefly some existing techniques that have been used within k-means.

Linde, Buzo and Gray [85] uses binary split for initialization of their *LBG algorithm* in the vector quantization context. Every cluster is split by replacing the original centroid c by $c+\varepsilon$ and $c-\varepsilon$, where ε refers to a random vector. Splitting every cluster avoids the question of which cluster to split but it does not have any real speed benefit. In [46], ε was calculated as the standard deviation of the points in the cluster, in each dimension separately.

Projection-based approaches are also suitable for the splitting algorithm. The idea is to divide a chosen cluster according to a hyperplane perpendicular to the projection axis. It is possible to find the optimal choice of the cluster to be split, and the optimal location of the hyperplane in $O(N)$ time [68, 46]. This results in a fast, $O(N \cdot \log N \cdot \log k)$ time algorithm, but the implementation is quite complex. It requires 22 functions and 947 lines of codes, compared to 5 functions and 162 lines in repeated k-means [16].

There is also a split-kmeans variant that applies k-means iteration after every split in [46], later popularized under the name *Bisecting k-means* in document clustering [86]. However, this would increase the time complexity to $O(k^2 N)$, which equals to $O(N^2)$ if $k \approx \mathrm{SQRT}(N)$. Tri-level k-means [87] performs the clustering in two stages. It first creates less clusters than k, and then splits the clusters with highest variation before applying the traditional k-means. All these variants are definitely standalone algorithms, and do not qualify as an initialization technique here.

In this paper, we therefore implement a simpler variant. We always select the biggest cluster to be split. The split is done by selecting two random points in the cluster. K-means is then applied but only within the cluster that was split as done in [68]. The main difference to the bisecting k-means [86] and its original split+kmeans variant in [46], is that the time complexity sums up to only $O(N \cdot \log N)$; a proof can be easily derived from the one in [46].

3.8. Repeated k-means

Repeated k-means performs k-means multiple times starting with different initialization, and then keeping the result with lowest SSE-value. This is sometimes referred as *multi-start k-means*. The basic idea of the repeats is to increase the probability of success. Repeated k-means can be formulated as a probabilistic algorithm as follows. If we know that k-means with a certain initialization technique will succeed with a probability of p, the expected number of repeats (R) to find the correct clustering would be:

$$R = 1/p$$

In other words, it is enough that k-means succeeds even sometimes (p>0). It is then merely a question of how many repeats are needed. Only if p≈0 the number of repeats can be unrealistically high. For example, standard k-means with random centroids succeeds 6-26% of the time with the S1-S4 datasets. These corresponds to R=7 to 14 repeats, on average.

If the initialization technique is deterministic (no randomness), then it either succeeds (p=100%) or fails (p=0%) every time. To justify the repeats, a basic requirement is that there is some randomness in the initialization so that the different runs produce different results. Most techniques have the randomness implicitly. The rest of the techniques we modify as follows:

- Rand-P Already included
- Rand-C Already included
- Maxmin First centroid randomly
- Kmeans++ Already included
- Bradley Already included
- Sorting Reference point randomly
- Projection Reference points randomly
- Luxburg Already included
- Split Split centroids randomly

Repeats add one new parameter R. Since p is not known in practice, we cannot derive value for R automatically. In this paper, we use R=100 unless otherwise noted. Fig. 12 shows the overall scheme of the repeated k-means.

Repeating k-means also multiplies the processing time by a factor of R. It is possible to compensate for this by dividing the data into random subsets. For instance, if we divide the data into R subsets of size N/R, the total processing time would be roughly the same as that of a single run.

For example, Bradley&Fayyad [31] apply k-means for a subsample of size N/R, where R=10 was recommended. Each sample is clustered by k-means starting with random centroids. However, instead of taking the best clustering of the repeats, a new dataset is created from the $R{\cdot}k$ centroids. This new dataset is then clustered by repeated k-means (R repeats). The total time complexity is $R{\cdot}k{\cdot}(N/R) + R{\cdot}k^2 = kN+Rk^2$, where the first part comes from clustering the sub-samples, and the second part from clustering the combined set. If k=SQRT(N), then this would be $N^{1.5} + RN$. Overall, the algorithm is fast and satisfies the criteria for initialization technique.

Bahmani et al. [88] have a similar approach. They repeat k-means++ R=O(log$N$) times to obtain $R{\cdot}k$ preliminary centroids, which are then used as a new dataset for clustering by standard k-means. They reported that R=5 would be sufficient for the number of repeats. In our experiments, we consider the Bradley&Fayyad [31] as an initialization, and use R=100 repeats as with all techniques.

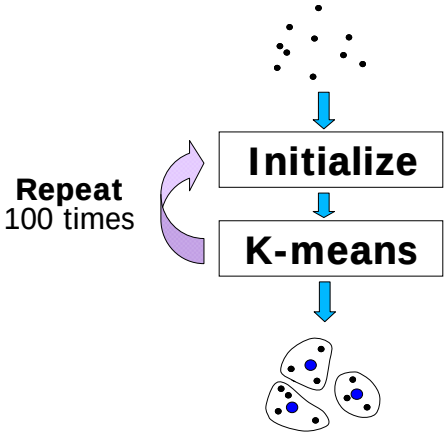

Figure 12: General principle of repeated k-means (RKM). The key idea is that the initialization includes randomness to produce different solutions at every repeat.

4. Experimental results

We study next the overall performance of different initialization techniques, and how the results depend on the following factors:

- Overlap of clusters
- Number of clusters
- Dimensions
- Unbalance of cluster sizes

The overall results (CI-values and success rates) are summarized in Table 3. We also record (as *fails*) how many datasets provide success rate p=0%. This means that the algorithm cannot find the correct clustering even with 5000 repeats. We test the following methods:

- Rand-P
- Rand-C
- Maxmin
- kmeans++
- Bradley
- Sorting
- Projection
- Luxburg
- Split

4.1. Overall results

CI-values: Random partition works clearly worse (CI=12.4) than the random centroids (CI=4.5). Bradley and sorting heuristics are slightly better (CI=3.1 and 3.3), but the maxmin heuristics (Maxmin and kmeans++) are the best among the true initialization techniques (CI=2.2 and 2.3). The standalone algorithms (Luxburg and Split) are better (CI=1.2 and 1.2), but even they provide the correct result (CI=0) only for the easiest dataset: DIM32.

Success rates: The results show that Maxmin is a reasonable heuristic. Its average success rate is 22% compared to 5% of random centroids. It also fails (success rate=0%) only in case of three datasets; the datasets with a high number of clusters (A3, Birch1,

Birch2). Random partition works with S2, S3 and S4 but fails with all the other 8 datasets. The standalone algorithms (Luxburg and Split) provide 40% success rates, on average, and fail only with Birch1 and Unbalance.

Effect of iterations: From the initial results we can see that Luxburg and Bradley are already standalone algorithms for which k-means brings only little improvement. The average CI-value of Luxburg improves only from 1.7 to 1.2 (~30%), and Bradley from 3.4 to 3.1 (~10%). The latter is more understandable as k-means is already involved in the iterations. Split heuristic, although a standalone algorithm, leaves more space for k-means to improve (61%).

Number of iterations: The main observation is that the easier the dataset, and the better the initialization, the fewer the iterations needed. The differences between the initialization vary from 20 (Luxburg) to 36 (Rand-C); with the exception of random partition (Rand-P), which takes 65 iterations.

Table 3: Average CI-values before and after k-means iterations, success rates, and the number of iterations performed. The results are averages of 5000 runs. *Fail* records for how many datasets the correct solution was never found (success rate=0%). From DIM datasets we report only DIM32; the results for the others are practically the same. Note: The values for Impr. and Aver. columns are calculated from precise values and not from the shown rounded values.

CI-values (initial)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.
Rand-P	12.5	14.0	12.8	14.0	19.0	32.9	48.1	7.0	96.0	96.6	13.1	**33.3**
Rand-C	5.3	5.5	5.4	5.4	7.3	12.7	18.2	4.6	36.6	36.6	5.8	**13.0**
Maxmin	1.3	2.9	6.1	6.8	2.1	4.1	5.0	0.9	21.4	9.6	0.0	**5.5**
kmeans++	1.7	2.3	3.2	3.3	3.1	5.6	7.9	0.8	21.3	10.4	0.1	**5.4**
Bradley	1.0	0.7	0.6	0.5	1.5	3.4	5.3	3.3	5.7	13.6	1.7	**3.4**
Sorting	3.3	3.7	4.1	4.4	4.9	10.4	15.6	4.0	34.1	7.2	1.7	**8.5**
Projection	3.0	3.4	3.9	4.2	4.5	9.8	15.2	4.0	33.7	1.0	1.1	**7.6**
Luxburg	0.8	0.8	1.1	1.3	0.9	1.1	1.2	4.2	5.6	1.7	0.0	**1.7**
Split	0.5	0.8	1.4	1.4	1.3	2.4	3.5	4.5	12.0	2.7	0.0	**2.8**

CI-values (final)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.	Impr.
Rand-P	3.3	0.6	1.2	0.4	6.0	10.7	17.9	4.0	11.3	75.6	5.3	**12.4**	**63 %**
Rand-C	1.8	1.4	1.3	0.9	2.5	4.5	6.6	3.9	6.6	16.6	3.6	**4.5**	**65 %**
Maxmin	0.7	1.0	0.7	1.0	1.0	2.6	2.9	0.9	5.5	7.3	0.0	**2.2**	**62 %**
kmeans++	1.0	0.9	1.0	0.8	1.5	2.9	4.2	0.5	4.9	7.2	0.1	**2.3**	**57 %**
Bradley	0.9	0.6	0.5	0.4	1.3	3.0	4.8	3.5	4.6	12.5	1.6	**3.1**	**11 %**
Sorting	1.3	1.1	1.0	0.7	1.5	3.6	5.5	4.0	5.7	4.3	1.4	**2.7**	**69 %**
Projection	1.2	0.9	0.8	0.6	1.2	3.3	5.2	4.0	5.3	0.2	0.9	**2.2**	**71 %**
Luxburg	0.5	0.4	0.6	0.4	0.6	0.9	1.0	4.0	2.7	1.6	0.0	**1.2**	**29 %**
Split	0.2	0.3	0.4	0.4	0.5	1.1	1.8	4.0	2.8	1.6	0.0	**1.2**	**61 %**

Success-%

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.	Fails
Rand-P	0%	47%	5%	63%	0%	0%	0%	0%	0%	0%	0%	**10%**	**8**
Rand-C	3%	11%	12%	26%	1%	0%	0%	0%	0%	0%	0%	**5%**	**6**
Maxmin	37%	16%	36%	9%	15%	1%	0%	22%	0%	0%	100%	**22%**	**3**
kmeans++	21%	24%	18%	30%	7%	0%	0%	51%	0%	0%	88%	**22%**	**4**
Bradley	21%	46%	49%	64%	7%	0%	0%	0%	0%	0%	2%	**17%**	**5**
Sorting	12%	20%	22%	36%	10%	0%	0%	0%	0%	12%	15%	**12%**	**4**
Projection	16%	29%	30%	42%	18%	0%	0%	0%	0%	92%	34%	**24%**	**4**
Luxburg	52%	60%	45%	61%	45%	33%	31%	0%	0%	17%	95%	**40%**	**2**
Split	78%	75%	62%	64%	51%	17%	5%	0%	0%	10%	99%	**42%**	**2**

Number of iterations

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.
Rand-P	32	37	37	39	43	58	76	36	228	130	3	**65**
Rand-C	20	24	27	40	22	26	27	33	117	48	5	**36**
Maxmin	13	19	24	37	20	18	20	4	92	43	2	**26**
kmeans++	14	19	24	35	17	20	22	13	89	43	2	**27**
Bradley	13	12	13	17	12	17	19	24	77	45	2	**23**
Sorting	17	21	25	37	19	24	26	38	104	33	3	**32**
Projection	15	20	25	35	17	24	25	36	99	6	3	**28**

Luxburg	9	12	17	27	11	12	12	33	62	23	2	**20**
Split	7	11	19	27	12	16	18	35	65	27	2	**22**

4.2. Cluster overlap

The results with the S1-S4 datasets (Table 3) demonstrate the effect of the overlap in general: the less overlap, the worse the k-means' performance. Some initialization techniques can compensate for this weakness. For example, the maxmin variants and the standalone algorithms reduce this phenomenon but do not remove it completely. They provide better initial solution with S1 (less overlap) than with S4 (more overlap), but the final result after the k-means iterations is still not much different. An extreme case is DIM32, for which all these better techniques provide correct solution. However, they do it even without k-means iterations!

Further tests with G2 confirm the observation, see Fig. 13. When overlap is less than 2%, the k-means iterations do not help much and the result depends mostly on the initialization. If the correct clustering is found, it is found without k-means. Thus, the clustering is solved by a better algorithm, not by better k-means initialization. In case of high overlap, k-means reaches almost the same result (about 88% success rate) regardless of how it was initialized.

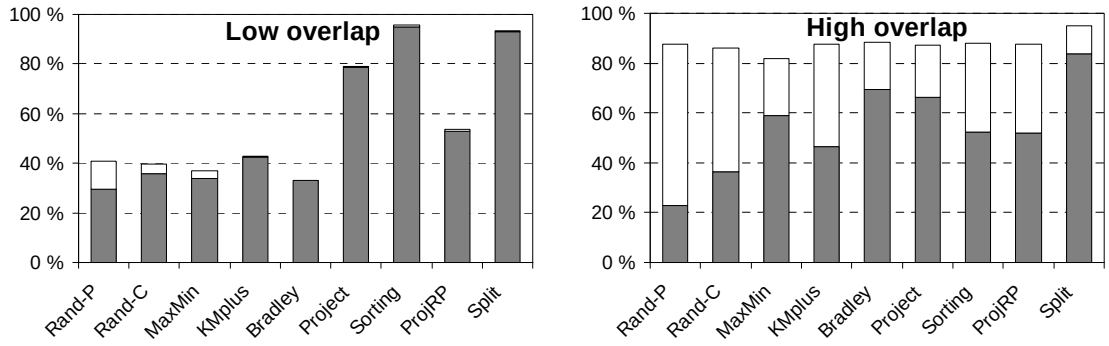

Figure 13: Average success rates for all G2 datasets before (gray) and after k-means (white). The datasets were divided into two categories: those with low overlap <2% (left), and those with high overlap ≥2% (right).

4.3 Number of clusters

The results with the A1-A3 datasets (Table 3) show that the more there are clusters the higher the CI-value and the lower the success rate. This phenomenon holds for all initialization techniques and it is not specific to k-means algorithm only. If an algorithm provides correct clustering with success rate *p* for a dataset of size *k*, then *p* is expected to decrease when *k* increases. Fig. 14 confirms this dependency with the Birch2 subsets. Projection heuristic is the only technique that manages to capture the hidden 1-dimensional structure in this data. The success rate of all other true initialization techniques eventually decreases to 0%.

Fig. 15 shows that the CI-value has a near linear dependency on the number of clusters. In most cases, the *relative* CI-value converges to a constant when *k* approaches its maximum (*k*=100). An exception is Luxburg, which is less sensitive to the increase of *k*; providing values CI=(0.82, 1.25, 1.42, 1.54) for k=(25, 50, 75, 100). Besides this

exception, we conclude that the performance has linear dependency on *k* regardless of the initialization technique.

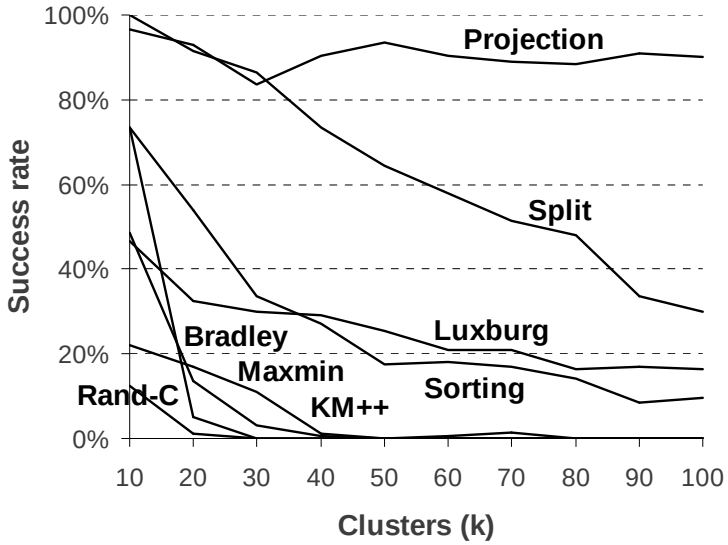

Figure 14: Dependency of the success rate and the number of clusters when using the subsets of Birch2 (*B2-sub*).

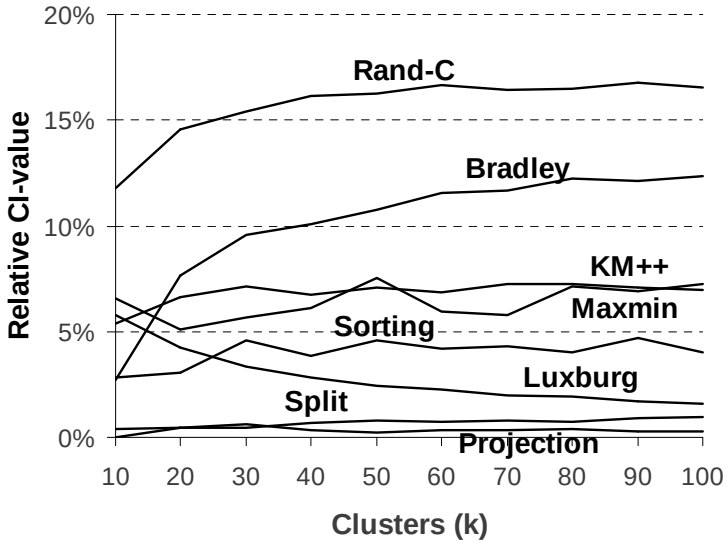

Figure 15: Dependency of the *relative* CI-values (CI/*k*) and the number of clusters when using the subsets of Birch2 (*B2-sub*).

4.4 Dimensions

We tested the effect of dimensions using the DIM and G2 datasets. Two variants (Maxmin, Split) solve the DIM sets almost every time (99-100%), whereas Kmeans++ and Luxburg solve them most of the times (≈95%), see Fig. 16. Interestingly, they find the correct result by the initialization and no k-means iterations are needed. In general, if the initialization technique is able to solve the clustering, it does it regardless of the dimensionality.

The sorting and projection heuristics are exceptions in this sense; their performance actually improves with the highest dimensions. The reason is that when the dimensions increase, the clusters eventually become so clearly separated that even such naïve

heuristics will be able to cluster the data. In general, the reason for success or failure is not the dimensionality but the cluster separation.

The results with G2 confirm the above observation, see Fig. 16. With the lowest dimensions, k-means iterations work because some cluster overlap exists. However, for higher dimensions the overlap eventually disappears and the performance starts to depend mainly on the initialization. We also calculated how much the success rate correlates with the dimensions and the overlap. The results in Table 4 show that the final result correlates much stronger with the overlap than with the dimensionality.

Since there is causality between dimensions and overlap, it is unclear whether the dimensionality has any role at all. To test this further, we generated additional datasets with D=2-16 and compared only those with overlap = 2%, 4%, 8%. The results showed that success of the k-means iterations do not depend on the dimensions even when the clusters overlap.

To sum up, our conclusion is that k-means iterations cannot solve the problem when the clusters are well separated. All techniques that solve these datasets, do it already by the initialization technique without any help of k-means. When there is overlap, k-means works better. But even then, the performance does not depend on the dimensionality.

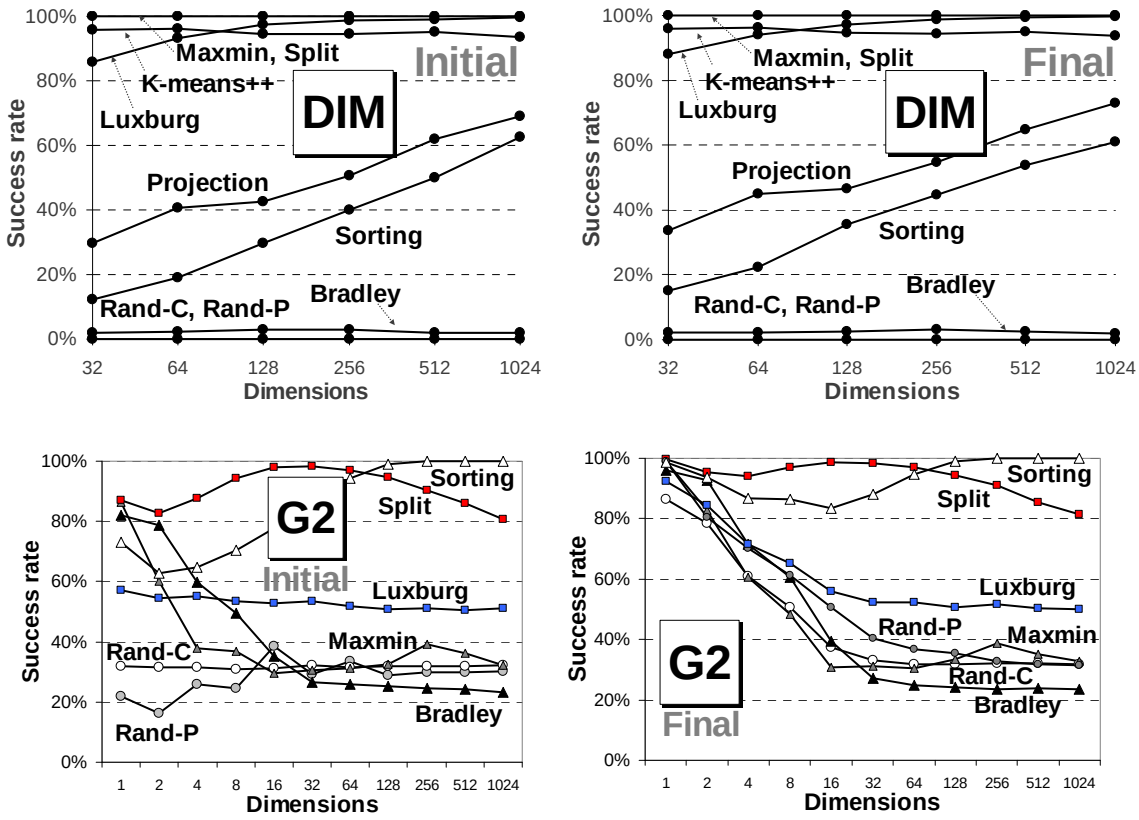

Figure 16: Dependency of success rate on the dimensions when no overlap (DIM sets), and with overlap (G2 datasets). The results of G2 are average success rates for all sd=10-100 (G2-D-sd) with a given dimension D, before and after k-means.

Table 4: Correlation of success rate with increasing overlap (left) and dimensions (right) with the G2 datasets (3:3 centroid allocation test). Red>0.60, Yellow=0.30-0.53.

	Overlap		Dimension	
	Init	Final	Init	Final
Rand-P	-0.34	0.68	0.11	-0.46
Rand-C	0.08	0.82	0.13	-0.35
Maxmin	0.61	0.73	-0.23	-0.32
kmeans++	0.63	0.80	-0.39	-0.41
Bradley	0.67	0.71	-0.48	-0.47
Sorting	0.46	0.69	-0.53	-0.51
Projection	0.02	0.45	0.01	-0.27
Luxburg	0.12	0.80	-0.32	-0.38
Split	-0.61	0.06	-0.39	-0.79

4.5 Unbalance

Unbalance dataset shows one weakness of k-means. The problem is not the different densities as such, but the unbalance of cluster sizes together with the separation of the clusters. If no centroids are selected from the sparse area, k-means iterations manage to move only one centroid into this area, and all other centroids will remain in the dense area, see Fig. 17. The probability that a single random centroid would be selected from the sparse area is p = 500/6500 = 7%. To pick all required five centroids from the sparse area would happen with probability of 0.01%[1], i.e. only once every 8430 runs.

Besides Rand-C and Rand-P, sorting and projection heuristics, Luxburg and Split algorithms all fail with this data by allocating most centroids to the dense area. Bradley works only slightly better and often allocates two centroids to the sparse area. Maxmin heuristics work best because they rely more on distances than on frequencies. K-means++ typically misses one centroid whereas Maxmin does the opposite and allocates one too many centroids in the sparse area. They provide success rates of 22% (Maxmin) and 51% (KM++), in contrast to the other techniques that result in 0% success.

To sum up, success depends mainly on the goodness of the initialization; k-means iterations can do very little with this kind of data. If the correct clustering is found, it is found mainly without k-means.

[1] $\binom{8}{5} p^5 (1-p)^3$

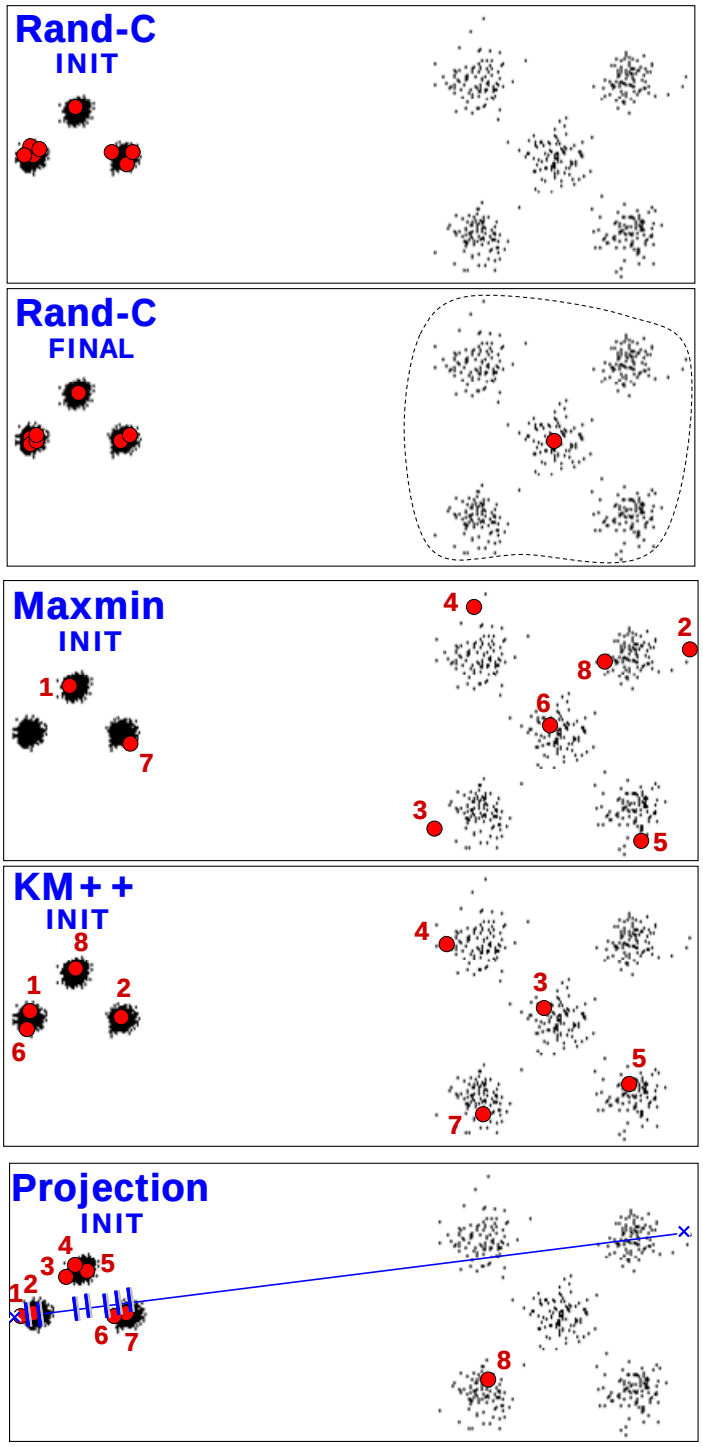

Figure 17: Examples of the initialization technique on the Unbalance dataset. The only techniques that do not badly fail are the maxmin heuristics. The numbers indicate the order in which the centroids are selected.

4.6 Repeats

We next investigate to what extent the k-means performance can be improved by repeating the algorithm several times. Table 5 summarizes the results. We can see that significant improvement is achieved with all initialization techniques. When the success rate of a single run of k-means is 2% or higher, CI=0 can always be reached thanks to

the repeats. However, none of the variants can solve all datasets. Overall performance of the different initialization techniques can be summarized as follows:

- Random partition is almost hopeless and the repeats do not help much. It only works when the clusters have strong overlap. But even then, k-means works relatively well anyway regardless of the initialization.
- Random centroids is improved from CI=4.5 to 2.1, on average, but still it can solve only three datasets (S2, S3, S4). Two other datasets (S1, A1) could be solved with significantly more repeats, but not the rest.
- Maxmin variants are the best among the simple initialization techniques providing CI=0.7, on average, compared to 2.1 of Rand-C. They still fail with four datasets. K-means++ is not significantly better than the simpler Maxmin.
- The standalone algorithms (Luxburg and Split) are the best. They provide average value of CI=1.2 without the repeats, and CI=0.4 with 100 repeats. They fail only with the Unbalance datasets.

The improvement from the repeats is achieved at the cost of increased processing time. We used the fast k-means variant [89] that utilizes the activity of the centroids. For the smaller data sets the results are close to real-time, but with the largest dataset (Birch1, N=100,000), the 100 repeats can take from 10-30 minutes.

Table 5: Performance of the repeated k-means (100 repeats). The last two columns show the average results of all datasets without repeats (KM) and with repeats (RKM).

CI-values

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	KM	RKM
Rand-P	1.4	0.0	0.0	0.0	4.9	8.8	16.7	3.6	8.5	74.0	2.6	12.4	11.0
Rand-C	0.1	0.0	0.0	0.0	0.3	1.8	2.9	2.9	2.8	10.9	1.1	4.5	2.1
Maxmin	0.0	0.0	0.0	0.0	0.0	0.5	0.6	0.0	2.8	3.9	0.0	2.2	0.7
kmeans++	0.0	0.0	0.0	0.0	0.0	0.8	1.6	0.0	1.7	3.4	0.0	2.3	0.7
Bradley	0.0	0.0	0.0	0.0	0.0	0.9	2.1	1.2	2.0	8.5	0.0	3.1	1.3
Sorting	0.0	0.0	0.0	0.0	0.0	0.8	2.2	4.0	2.2	0.0	0.0	2.7	0.8
Projection	0.0	0.0	0.0	0.0	0.0	0.9	2.0	3.9	1.9	0.0	0.0	2.2	0.4
Luxburg	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.7	0.6	0.0	0.0	1.2	0.4
Split	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.6	0.0	0.0	1.2	0.4

Success rate (%)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32	Aver.	Fails
Rand-P	0%	100%	100%	100%	0%	0%	0%	0%	0%	0%	0%	**27%**	**10**
Rand-C	96%	100%	100%	100%	56%	2%	0%	0%	0%	0%	2%	**41%**	**10**
Maxmin	100%	100%	100%	100%	100%	58%	36%	100%	0%	0%	100%	**72%**	**4**
kmeans++	100%	100%	100%	100%	98%	20%	0%	100%	0%	0%	100%	**65%**	**4**
Bradley	100%	100%	100%	100%	100%	4%	4%	4%	0%	0%	84%	**54%**	**6**
Sorting	100%	100%	100%	100%	100%	24%	0%	0%	2%	100%	100%	**66%**	**4**
Projection	100%	100%	100%	100%	100%	18%	0%	0%	0%	100%	100%	**65%**	**4**
Luxburg	100%	100%	100%	100%	100%	100%	100%	0%	46%	100%	100%	**86%**	**2**
Split	100%	100%	100%	100%	100%	100%	100%	0%	36%	100%	100%	**85%**	**2**

Running time (s)

Method	s1	s2	s3	s4	a1	a2	a3	unb	b1	b2	dim32
Rand-P	3.2	4.4	4.7	6.6	3.1	8.9	15	5.0	1657	1037	0.3
Rand-C	2.6	3.2	4.1	8.2	2.0	4.4	7.5	4.8	882	172	0.4
Maxmin	1.9	2.4	3.7	5.6	1.8	3.2	5.6	2.5	596	146	0.3
kmeans++	1.9	2.8	3.7	6.6	1.9	3.7	6.4	4.1	604	143	0.4
Bradley	2.5	2.7	3.5	5.6	1.9	4.5	7.8	4.7	605	195	1.0
Sorting	2.3	3.1	3.8	7.6	1.9	4.4	6.8	4.9	815	148	0.3
Projection	2.1	3.0	3.7	7.1	1.7	4.1	6.6	5.0	768	84	0.3
Luxburg	1.9	2.4	3.2	6.1	1.7	3.5	5.5	4.9	431	126	0.4
Split	3.5	3.9	5.3	6.9	2.3	5.8	10	11	1072	988	1.2

We extended the tests and ran 200,000 repeats for A3 and Unbalance datasets. The results in Table 6 show that Maxmin would need 216 repeats to reach CI=0 with A3, on average, whereas k-means++ would require 8,696 repeats even though it finds CI=1 already after 138 repeats. The results also show that Unbalance dataset is difficult for almost all initialization techniques but the maxmin heuristics are most suitable for this type of data.

Table 6: Number of repeats in RKM to reach certain CI-level. Missing values (-) indicate that this CI-level was never reached during the 200,000 repeats.

A3

Initialization	CI-value						
	6	**5**	**4**	**3**	**2**	**1**	**0**
Rand-P	-	-	-	-	-	-	-
Rand-C	2	4	11	54	428	11111	-
Maxmin				1	3	14	216
Kmeans++		1	2	3	14	138	8696
Bradley		1	2	8	58	1058	33333
Sorting	1	2	4	13	73	1143	-
Projection	1	2	3	9	46	581	18182
Luxburg						1	3
Split					1	2	9

Unbalance

Initialization	CI-value						
	6	**5**	**4**	**3**	**2**	**1**	**0**
Rand-P			1	97	8333	-	-
Rand-C			1	16	69	1695	100k
Maxmin						1	4
Kmeans++						1	2
Bradley			1	3	6	70	1471
Sorting			1	-	-	-	-
Projection			1	935	16667	-	-
Luxburg			1	59	16667	-	-
Split			1	9524	-	-	-

4.7 Summary

We make the following observations:

- Random partition provides an initial solution of similar quality regardless of overlap, but the errors in initial solution can be better fixed by k-means iterations when clusters have high overlap. In this case it can even outperform random centroids. However, repeats do not improve the results much, especially with sets having many clusters (A3, Birch2).
- Cluster overlap is the biggest factor. If there is high overlap, k-means iterations work well regardless of the initialization. If there is no overlap, then the success depends completely on the initialization technique: if it fails, k-means will also fail.
- Practically all initialization techniques perform worse when the number of clusters increases. Success of the k-means depends linearly on the number of clusters. The more clusters, the more errors there are, before and after the iterations.
- Dimensionality does not have a direct effect. It has a slight effect on some initialization techniques but k-means iterations are basically independent on the dimensions.

- Unbalance of cluster sizes can be problematic especially for the random initializations but also for the other techniques. Only the maxmin variants with 100 repeats can overcome this problem.

Table 7 summarizes how the four factors affect the different initialization techniques and the k-means iterations.

Table 7: How the four factors have effect on the performance of the initialization and on the k-means iterations.

Method	Overlap	Clusters	Dimension	Unbalance
Rand-P	No effect	Constant	No effect	Very bad
Rand-C	No effect	Constant	No effect	Very bad
Maxmin	Bad	Constant	No effect	A bit worse
kmeans++	A bit worse	Constant	No effect	A bit worse
Bradley	Good	Constant	No effect	Bad
Sorting	A bit worse	Constant	No effect	Very bad
Projection	A bit worse	Constant	No effect	Very bad
Luxburg	A bit worse	Minor effect	No effect	Very bad
Split	A bit worse	Constant	No effect	Very bad
KM iterations	Good	Constant	No effect	No effect

5. Conclusions

On average, k-means caused errors with about 15% of the clusters (CI=4.5). By repeating k-means 100 times this errors was reduced to 6% (CI=2.0). Using a better initialization technique (Maxmin), the corresponding numbers were 6% (CI=2.1) with k-means as such, and 1% (CI=0.7) with 100 repeats. For most pattern recognition applications this accuracy is more than enough when clustering is just one component within a complex system.

The most important factor is the cluster overlap. In general, well separated clusters make the clustering problem easier but for k-means it is just the opposite. When the clusters overlap, k-means iterations work reasonably well regardless of the initialization. This is the expected situation in most pattern recognition applications.

The number of errors have a linear dependency on the number of clusters (k): the more clusters, the more errors k-means makes, but the percentage remains constant. Unbalance of cluster sizes is more problematic. Most initialization techniques fail, and only the maxmin heuristics worked in this case. The clustering result then depends merely on the goodness of the initialization technique.

Dimensionality itself is not a factor. It merely matters how the dimensions affect the cluster overlap. With our data, the clusters became more separated when the dimensions were increased, which in turn worsened the k-means performance. Besides this indirect effect, the dimensions did not matter much.

With real data the effect might be just the opposite. If the features (attributes) are added in the order of their clustering capability, it is expected that the clusters would become more overlapping when adding more features. As a result, k-means would start to work better but the data itself would become more difficult to cluster, possibly losing the

clustering structure. And vice versa, if good feature selection is applied, the clusters can be more separated, which has the danger that k-means would start to perform worse.

Based on these observations, choosing an initialization technique like Maxmin can compensate for the weaknesses of k-means. With unbalanced cluster sizes it might work best overall. However, it is preferable to repeat the k-means 10-100 times; each time taking a random point as the first centroids and selecting the rest using the Maxmin heuristic. This will keep the number of errors relatively small.

However, the fundamental problem of k-means still remains when the clusters are well separated. From all the tested combinations, none was able to solve all the benchmark datasets despite them being seemingly simple. With 100 repeats, Maxmin and k-means++ solved 7 datasets (out of the 11), thus being the best initialization techniques. The better standalone algorithms (Luxburg and Split) managed to solve 9.

To sum up, if the clusters overlap, the choice of initialization technique does not matter much, and repeated k-means is usually good enough for the application. However, if the data has well-separated clusters, the result of k-means depends merely on the initialization algorithm.

In general, the problem of initialization is not any easier than solving the clustering problem itself. Therefore, if the accuracy of clustering is important, then a better algorithm should be used. Using the same computing time spent for repeating k-means, a simple alternative called random swap (RS) [12] solves all the benchmark datasets. Other standalone algorithms that we have found able to solve all the benchmark sets include genetic algorithm (GA) [10], the split algorithm [46], split k-means [46], and density peaks [78]. Agglomerative clustering [30] solves 10 out of 11.

References

1. E. Forgy, Cluster analysis of multivariate data: efficiency vs. interpretability of classification. *Biometrics*, 21, 768-780, 1965.
2. J. MacQueen, Some methods for classification and analysis of multivariate observations, *Berkeley Symposium on Mathematical Statistics and Probability*, Volume 1: Statistics, pp. 281-297, University of California Press, Berkeley, Calif., 1967.
3. S.P. Lloyd, Least squares quantization in PCM, *IEEE Trans. on Information Theory*, 28 (2), 129–137, 1982.
4. L. Wang and C. Pan, Robust level set image segmentation via a local correntropy-based k-means clustering, *Pattern Recognition*, 47, 1917-1925, 2014.
5. C. Boutsidis, A. Zouzias, M. W. Mahoney and P. Drineas, Randomized dimensionality reduction for k-means clustering, *IEEE Transactions on Information Theory*, 61 (2), 1045-1062, Feb. 2015.
6. M. Capo, A, Perez and J.A. Lozano, An efficient approximation to the k-means clustering for massive data, *Knowledge-Based Systems*, 117, 56-69, 2017.
7. Z. Huang, N. Li, K. Rao, C. Liu, Y. Huang, M. Ma, and Z. Wang, Development of a data-processing method based on Bayesian k-means clustering to discriminate aneugens and clastogens in a high-content micronucleus assay. *Human & Experimental Toxicology*, 37 (3), 285–294, 2018.
8. A.K. Jain, Data clustering: 50 years beyond K-means, *Pattern Recognition Letters*, 31, 651-666, 2010.
9. K. Krishna, and M.N, Murty. Genetic k-means algorithm, *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, 29 (3), 433-439, 1999.

10. P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, *Pattern Recognition Letters*, 21 (1), 61-68, 2000
11. P. Fränti and J. Kivijärvi, Randomized local search algorithm for the clustering problem, *Pattern Analysis and Applications*, 3 (4), 358-369, 2000.
12. P. Fränti, Efficiency of random swap clustering, *Journal of Big Data*, 5:13, 1-29, 2018.
13. S. Kalyani, K.S. Swarup, Particle swarm optimization based K-means clustering approach for security assessment in power systems, *Expert Systems with Applications*, 32 (9), 10839-10846, 2011.
14. D. Yan, L. Huang, and M.I. Jordan, Fast approximate spectral clustering, *ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, 907-916, 2009.
15. L. Bai, X. Cheng, J. Liang, H. Shen, Y. Guo, Fast density clustering strategies based on the k-means algorithm, *Pattern Recognition*, 71, 375-386, 2017.
16. T. Kinnunen, I. Sidoroff, M. Tuononen and P. Fränti, Comparison of clustering methods: a case study of text-independent speaker modeling, *Pattern Recognition Letters*, 32 (13), 1604-1617. October 2011.
17. Q. Zhao and P. Fränti, WB-index: a sum-of-squares based index for cluster validity, *Data & Knowledge Engineering*, 92, 77-89, July 2014.
18. M. Rezaei and P. Fränti Can the number of clusters be solved by external index? manuscript. (submitted)
19. J.M Peña, J.A. Lozano, P. Larrañaga, An empirical comparison of four initialization methods for the *k*-means algorithm, *Pattern Recognition Letters*, 20 (10): 1027-1040, October 1999.
20. J. He, M. Lan, C-L Tan, S-Y Sung, H-B Low, Initialization of Cluster Refinement Algorithms: A review and comparative study, *IEEE Int. Joint Conf. on Neural Networks*, 2004.
21. D. Steinley and M.J. Brusco, Initializing k-means batch clustering: a critical evaluation of several techniques, *Journal of Classification* 24: 99-121, 2007.
22. M.E. Celebi, H.A. Kingravi, P.A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Syst. Appl*. 40, 200-210, 2013.
23. L. Kaufman, and P. Rousseeuw, *Finding groups in data: An introduction to cluster analysis*. Wiley Interscience, 1990.
24. B. Thiesson, C. Meek, D. M. Chickering, and D. Heckerman, Learning mixtures of Bayesian networks, Technical Report MSR-TR-97-30 Cooper & Moral, 1997.
25. J.T. Tou and R.C. Gonzales, *Pattern Recognition Principles*. Addison-Wesley, 1974.
26. T.F. Gonzalez, Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38 (2–3), 293–306, 1985.
27. J.H. Ward, Hierarchical grouping to optimize an objective function, *Journal of American Statistical Association*, 58(301): 236-244, 1963.
28. A. Likas, A., N. Vlassis, and J. Verbeek, The global k-means clustering algorithm, *Pattern Recognition, 36*, 451–461, 2003.
29. D. Steinley, Local optima in k-means clustering: what you don't know may hurt you, *Psychological Methods*, 8, 294–304, 2003.
30. P. Fränti, T. Kaukoranta, D.-F. Shen and K.-S. Chang, Fast and memory efficient implementation of the exact PNN, *IEEE Trans. on Image Processing*, 9 (5), 773-777, May 2000.
31. P. Bradley and U. Fayyad, Refining initial points for k-means clustering, *Int. Conf. on Machine Learning*, 91-99, San Francisco, 1998.
32. Duda, R.O., Hart, P.E., 1973. Pattern Classification and Scene Analysis. John Wiley and Sons, New York.

33. M. Bicego, M.A.T. Figueiredo, Clustering via binary embedding, *Pattern Recognition*, 83, 52-63, 2018.
34. N. Karmitsa, A.M.Bagirov, S. Taheri, Clustering in large data sets with the limited memory bundle method, *Pattern Recognition*, 83, 245–259, 2018.
35. Y. Zhu, K.M. Ting, M.J. Carman, Grouping points by shared subspaces for effective subspace clustering, *Pattern Recognition*, 83 230–244, 2018.
36. P.B. Frandsen, B. Calcott, C. Mayer, R. Lanfear, Automatic selection of partitioning schemes for phylogenetic analyses using iterative k-means clustering of site rates, *BMC Evolutionary Biology*, 15 (13), 2015**.**
37. D.G. Márquez, A. Otero, P. Félix, C.A. García, A novel and simple strategy for evolving prototype based clustering, *Pattern Recognition*, 82, 16–30, 2018.
38. L. Huang, H.-Y. Chao, C.-D. Wang, Multi-view intact space clustering, *Pattern Recognition*, 86, 344–353, 2019.
39. P. Fränti and S. Sieranoja, K-means properties on six clustering benchmark datasets, *Applied Intelligence*, 2018.
40. L. Morissette and S. Chartier, "The k-means clustering technique: general considerations and implementation in Mathematica, *Tutorials in Quantitative Methods for Psychology*, 9 (1), 15-24, 2013.
41. J. Liang, L. Bai, C. Dang F. Cao, The k-means-type algorithms versus imbalanced data distributions, *IEEE Trans. on Fuzzy Systems*, 20 (4), 728-745, August 2012.
42. I. Melnykov, V. Melnykov, On k-means algorithm with the use of Mahalanobis distances, *Statistics & Probability Letters*, 84, 88-95, January 2014.
43. V. Melnykov, S. Michael, and I. Melnykov, Recent developments in model-based clustering with applications. In: Celebi M. (eds) *Partitional Clustering Algorithms*. Springer, Cham, 2015.
44. M. Rezaei and P. Fränti, Set-matching methods for external cluster validity, *IEEE Trans. on Knowledge and Data Engineering*, 28 (8), 2173-2186, August 2016.
45. P. Fränti, M. Rezaei and Q. Zhao, Centroid index: Cluster level similarity measure, *Pattern Recognition*, 47 (9), 3034-3045, 2014.
46. P. Fränti, T. Kaukoranta and O. Nevalainen: On the splitting method for VQ codebook generation, *Optical Engineering*, 36 (11), 3043-3051, November 1997.
47. P. Fränti, O. Virmajoki and V. Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, IEEE Trans. on Pattern Analysis and Machine Intelligence, 28 (11), 1875-1881, November 2006
48. G.H. Ball and D.J. Hall, A clustering technique for summarizing multivariate data, *Systems Research & Behavioral Science*, 12 (2), 153-155, March 1967.
49. O. Lemke and B. Keller, Common nearest neighbor clustering: a benchmark, *Algorithms*, 11 (2), 19, 2018.
50. U. V. Luxburg, Clustering stability: An overview, *Foundations and Trends in Machine Learning*, 2(3), pp. 235-274, 2010.
51. S.J. Redmond and C. Heneghan, A method for initialising the K-means clustering algorithm using kd-trees, *Pattern Recognition Letters*, 28 (8), 965-973, 2007.
52. S. Tezuka and P.L Equyer, Efficient portable combined Tausworthe random number generators, *ACM Transactions on Modelling and Computer Simulation*, 1 99-112, 1991.
53. M.J. Norušis, IBM SPSS statistics 19 guide to data analysis. Upper Saddle River, New Jersey: Prentice Hall, 2011.
54. T. Gonzalez, Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38 (2–3), 293–306, 1985.

55. M.M.-T. Chiang, B. Mirkin, Intelligent choice of the number of clusters in *k*-means clustering: an experimental study with different cluster spreads, *Journal of Classification*, 27, 3-40, 2010.
56. J. Hämäläinen, T. Kärkkäinen, Initialization of big data clustering using distributionally balanced folding, Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning-ESANN, 2016.
57. I. Katsavounidis, C.C.J. Kuo, Z. Zhang, A new initialization technique for generalized Lloyd iteration, *IEEE Signal Processing Letters*, 1 (10), 144-146, 1994.
58. F. Cao, J. Liang, L. Bai, A new initialization method for categorical data clustering, *Expert Systems with Applications*, 36 (7), 10223-10228, 2009.
59. D. Arthur and S. Vassilvitskii, K-means++: the advantages of careful seeding, *ACM-SIAM Symp. on Discrete Algorithms (SODA'07)*, New Orleans, LA, 1027-1035, January, 2007.
60. M. Erisoglu, N. Calis, S. Sakallioglu, A new algorithm for initial cluster centers in *k*-means algorithm, *Pattern Recognition Letters*, 32 (14), 1701-1705, 2011.
61. C. Gingles, M. Celebi, Histogram-based method for effective initialization of the k-means clustering algorithm. *Florida Artificial Intelligence Research Society Conference*, May 2014.
62. J.A. Hartigan and M.A. Wong, Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society C*, 28 (1), 100–108, 1979.
63. M.M. Astrahan, Speech Analysis by Clustering, Or the Hyperphome Method, Stanford Artificial Intelligence Project Memorandum AIM-124, Stanford, CA: Stanford University, 1970.
64. F. Cao, J. Liang, G. Jiang, An initialization method for the k-means algorithm using neighborhood model, *Computers and Mathematics with Applications*, 58, 474–483, 2009.
65. M. Al-Daoud, A new algorithm for cluster initialization, *World Enformatika conf*, 74–76, 2005.
66. M. Yedla, S.R. Pathakota, T.M. Srinivasa, Enhancing k-means clustering algorithm with improved initial center, *Int. Journal of Computer Science and Information Technologies*, 1 (2), 121-125, 2010.
67. T. Su, J.G. Dy, In search of deterministic methods for initializing k-means and gaussian mixture clustering, *Intelligent Data Analysis*, 11 (4), 319-338, 2007.
68. X. Wu and K. Zhang, A better tree-structured vector quantizer, *IEEE Data Compression Conference*, Snowbird, UT, 392-401, 1991.
69. C.-M. Huang and R. W. Harris. A comparison of several vector quantization codebook generation approaches, *IEEE Trans. on Image Processing*, 2 (1), 108-112, 1993.
70. D. Boley, Principal direction divisive partitioning, *Data Mining and Knowledge Discovery*, 2(4):325–344, 1998.
71. M.E. Celebi and H. A. Kingravi. Deterministic initialization of the k-means algorithm using hierarchical clustering. *International Journal of Pattern Recognition and Artificial Intelligence*, 26 (07): 1250018, 2012.
72. S. Sieranoja and P. Fränti, Random projection for k-means clustering, *Int. Conf. Artificial Intelligence and Soft Computing (ICAISC)*, Zakopane, Poland, 680-689, June 2018.
73. S.-W. Ra and J.-K. Kim, A fast mean-distance-ordered partial codebook search algorithm for image vector quantization, *IEEE Trans. Circuits and Systems*, 40, 576–579, Sept. 1993.
74. I. Cleju, P. Fränti, X. Wu, Clustering based on principal curve. *Scandinavian Conf. on Image Analysis*, LNCS, vol. 3540, pp. 872–881. Springer, Heidelberg, 2005.
75. X. Wu, Optimal quantization by matrix searching, *Journal of algorithms*, 12 (4), 663-673, 1991.
76. M.B. Al-Daoud, and S.A. Roberts. New methods for the initialisation of clusters, *Pattern Recognition Letters*, 17 (5), 451-455, 1996.

77. P. Gourgaris, C. Makris, A density based k-means initialization scheme, *EANN workshops*, Rhodes Island, Greece, 2015.
78. A. Rodriquez and A. Laio, Clustering by fast search and find of density peaks, *Science*, 344 (6191), 1492-1496, 2014.
79. P. Mitra, C. Murthy, and S. K. Pal, Density-based multiscale data condensation, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 24 (6), 734-747, 2002.
80. S. Sieranoja and P. Fränti, Constructing a high-dimensional kNN-graph using a Z-order curve, *ACM Journal of Experimental Algorithmics*, 23 (1), 1.9:1-21, October 2018.
81. W. Dong, C. Moses, K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures. In Proceedings of the ACM International Conference on World wide web. ACM, 577–586, 2011.
82. P. Fränti and S. Sieranoja, Dimensionally distributed density estimation, *Int. Conf. Artificial Intelligence and Soft Computing (ICAISC)*, Zakopane, Poland, 343-353, June 2018.
83. H.J. Curti, R.S. Wainschenker, FAUM: Fast Autonomous Unsupervised Multidimensional classification, *Information Sciences*, 462, 182–203, 2018.
84. J. Xie, Z.Y. Xiong, Y.F. Zhang, Y. Feng, J. Ma, Density core-based clustering algorithm with dynamic scanning radius, *Knowledge-Based Systems*, 142, 68-70, 2018.
85. Y. Linde, A. Buzo, and R.M. Gray, An algorithm for vector quantizer design, *IEEE Trans. Commun.*, 28 (1), 84-95, January 1980.
86. M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, in KDD workshop on text mining, vol. 400, pp. 525-526, Boston, 2000.
87. S-S. Yu, S-W. Chu, C-M. Wang, Y-K. Chan, T-C. Chang, Two improved k-means algorithms, *Applied Soft Computing* 68, 747–755, 2018.
88. B. Bahmani, B. Mosley, A. Vattani, R. Kumar, S. Vassilvitski, Scalable k-means++, *Proceedings of the VLDB Endowment* 5 (7), 622-633, 2012.
89. T. Kaukoranta, P. Fränti and O. Nevalainen, A fast exact GLA based on code vector activity detection, *IEEE Trans. on Image Processing*, 9 (8), 1337-1342, August 2000.