Pattern Recognition 47 (2014) 3376–3386

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

K-means*: Clustering by gradual data transformation

Mikko I. Malinen*, Radu Mariescu-Istodor, Pasi Fränti

Speech and Image Processing Unit, School of Computing, University of Eastern Finland, Box 111, FIN-80101 Joensuu, Finland

ARTICLE INFO

Article history:
Received 30 September 2013
Received in revised form
27 March 2014
Accepted 29 March 2014
Available online 18 April 2014

Keywords:
Clustering
K-means
Data transformation

ABSTRACT

Traditional approach to clustering is to fit a model (partition or prototypes) for the given data. We propose a completely opposite approach by fitting the data into a given clustering model that is optimal for similar pathological data of equal size and dimensions. We then perform inverse transform from this pathological data back to the original data while refining the optimal clustering structure during the process. The key idea is that we do not need to find optimal global allocation of the prototypes. Instead, we only need to perform local fine-tuning of the clustering prototypes during the transformation in order to preserve the already optimal clustering structure.

1. Introduction

Euclidean sum-of-squares clustering is an NP-hard problem [1], where one assigns n data points to k clusters. The aim is to minimize the mean squared error (MSE), which is the mean distance of the data points from the nearest centroids. When the number of clusters k is constant, Euclidean sum-of-squares clustering can be done in polynomial $O(n^{kd+1})$ time [2], where d is the number of dimensions. This is slow in practice, since the power $kd+1$ is high, and thus, suboptimal algorithms are used. The *K*-means algorithm [3] is fast and simple, although its worst-case running time is high, since the upper bound for the number of iterations is $O(n^{kd})$ [4].

In k-means, given a set of data points $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, one tries to assign the data points into k sets ($k < n$), $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$, so that MSE is minimized:

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of S_i. An initial set of the k means $\mathbf{m}_1^{(1)}, \ldots, \mathbf{m}_k^{(1)}$ may be given randomly or by some heuristic [5–7]. The k-means algorithm alternates between the two steps [8]:

Assignment step: Assign the data points to clusters specified by the nearest centroid:

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \mathbf{m}_i^{(t)}\| \le \|\mathbf{x}_j - \mathbf{m}_{i^*}^{(t)}\|, \ \forall\ i^* = 1, \ldots, k \right\}$$

Update step: Calculate the mean of each cluster:

$$\mathbf{m}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

The k-means algorithm converges when the assignments no longer change. In practice, the k-means algorithm stops when the criterion of inertia does not vary significantly: it is useful to avoid non-convergence when the clusters are symmetrical, and in the other cluster configurations, to avoid too long time of convergence.

The main advantage of k-means is that it always finds a local optimum for any given initial centroid locations. The main drawback of k-means is that it cannot solve global problems in the clustering structure (see Fig. 1). By solved global clustering structure we mean such initial centroid locations from which the optimum can be reached by k-means. This is why slower agglomerative clustering [9–11], or more complex k-means variants [7,12–14] are sometimes used. *K*-means++ [7] is like k-means, but there is a more complex initialization of centroids. Gaussian mixture models can also be used (Expectation-Maximization algorithm) [15,16] and cut-based methods have been found to give competitive results [17]. To get a view of the recent research in clustering, see [18–20], which deal with analytic clustering, particle swarm optimization and minimum spanning tree based split-and-merge algorithm.

In this paper, we attack the clustering problem by a completely different approach than the traditional methods. Instead of trying to solve the correct global allocation of the clusters by fitting the clustering model to the data X, we do the opposite and fit the data to an optimal clustering structure. We first generate an artificial data X^* of the same size (n) and dimension (d) as the input data, so that the data vectors are divided into k perfectly separated clusters

* Corresponding author.
E-mail addresses: mmali@cs.uef.fi (M.I. Malinen), radum@cs.uef.fi (R. Mariescu-Istodor), franti@cs.uef.fi (P. Fränti).
URLS: http://cs.uef.fi/~mmali (M.I. Malinen), http://cs.uef.fi/~radum (R. Mariescu-Istodor), http://cs.uef.fi/pages/franti (P. Fränti).

http://dx.doi.org/10.1016/j.patcog.2014.03.034
0031-3203/

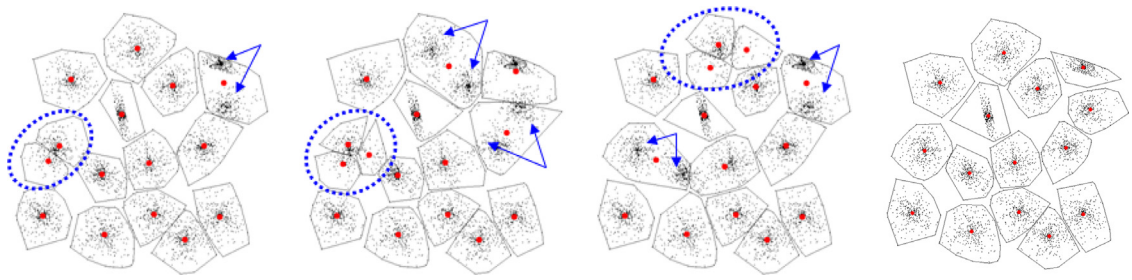

Fig. 1. Results of k-means for three random initializations (left) showing that k-means cannot solve global problems in the clustering structure. Circles show clusters that have too many centroids. Arrows show clusters that have too few centroids. Clustering result obtained by the proposed method (right).

without any variation. We then perform one-to-one bijective mapping of the input data to the artificial data ($X \rightarrow X^*$).

The key point is that we already have a clustering that is optimal for the artificial data, but not for the real data. In the next step, we then perform inverse transform of the artificial data back to the original data by a sequence of gradual changes. While doing this, the clustering model is updated after each change by k-means. If the changes are small, the data vectors will gradually move to their original position without breaking the clustering structure. The details of the algorithm including the pseudocode are given in Section 2. An online animator demonstrating the progress of the algorithm is available at http://cs.uef.fi/sipu/clustering/animator/. The animation starts when "Gradual k-means" is chosen from the menu.

The main design problems of this approach are to find a suitable artificial data structure, how to perform the mapping, and how to control the inverse transformation. We will demonstrate next that the proposed approach works with simple design choices, and overcomes the locality problem of k-means. It cannot be proven to provide optimal result every time, as there are pathological counter-examples where it fails to find the optimal solution. Nevertheless, we show by experiments that the method is significantly better than k-means, significantly better than k-means++ and competes equally with repeated k-means. It also rarely ends up to a bad solution that is typical to k-means. Experiments will show that only a few transformation steps are needed to obtain a good quality clustering.

2. *K*-means* algorithm

In the following subsections, we will go through the phases of the algorithm. For pseudocode, see Algorithm 1. We call this algorithm k-means*, because of the repeated use of k-means. However, instead of applying k-means to the original data points, we create another artificial dataset which is prearranged into k clearly separated zero-variance clusters.

2.1. Data initialization

The algorithm starts by choosing the artificial clustering structure and then dividing the data points among these equally. We do this by creating a new dataset X_2 and by assigning each data point in the original dataset X_1 to a corresponding data point in X_2, see Fig. 2. We consider seven different structures for the initialization:

- line
- diagonal
- random
- random with optimal partition
- initialization used in k-means++
- line with uneven clusters
- point.

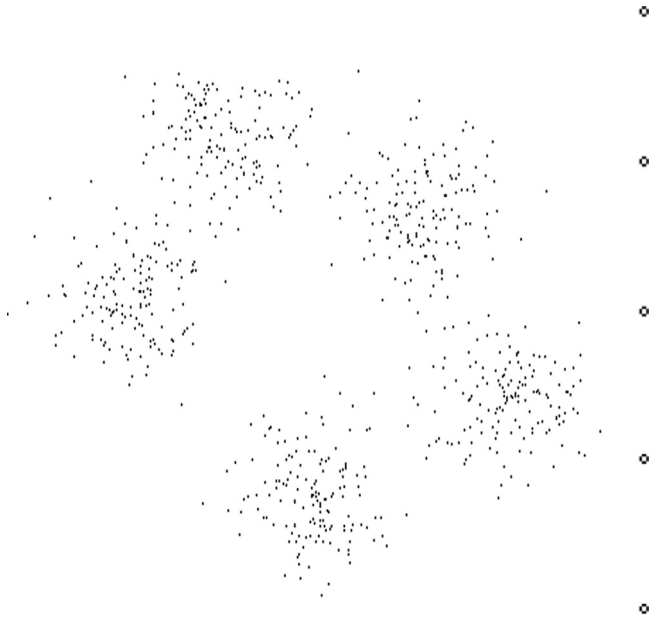

Fig. 2. Original dataset (left), and the corresponding artificial dataset using line init (right).

In the line structure, the clusters are arranged along a line. The k locations are set as the middle value of the range in each dimension, except the last dimension where the k clusters are distributed uniformly along the line, see Fig. 3 (left) and the animator http://cs.uef.fi/sipu/clustering/animator/. The range of 10% nearest to the borders is left without clusters. In the diagonal structure, the k locations are set uniformly to the diagonal of the range of the dataset. In the random structure, the initial clusters are selected randomly among the data point locations in the original dataset, see Fig. 3 (right). In these structuring strategies, data point locations are initialized randomly to these cluster locations. Even distribution among the clusters is a natural choice. To justify it further, lower cardinality clusters can more easily become empty later, which is an undesirable situation.

The fourth structure is random locations but using optimal partitions for the mapping. This means assigning the data points to the nearest clusters. The fifth structure corresponds to the initialization strategy used in k-means++ [7]. This initialization is done as follows: at any given time, let $D(X_i)$ denote the shortest distance from a data point X_i to its closest centroid we have already chosen.

Choose first centroid C_1 uniformly at random from X.

Repeat: Choose the next centroid as a point X_i, using a weighted probability distribution where a point is chosen with probability proportional to $D(X_i)^2$.

Until we have chosen a total of k centroids.

As a result, new centers are added more likely to the areas lacking centroids. The sixth structure is the line with uneven clusters, in which

we place twice more points to most centrally located half of the cluster locations. The seventh structure is the point. It is like line structure but we put the clusters in a very short line, which looks like a single point in larger scale. In this way the dataset "explodes" from a single point during the inverse transform. This structure is useful mainly for the visualization purpose in the web-animator. The *k*-means++-style structure with evenly distributed data points is the recommended structure because it works best in practice, and therefore we use it in further experiments. In choosing the structure, good results are achieved when there is a notable separation between clusters and evenly distributed data points in clusters.

Once the initial structure is chosen, each data point in the original dataset is assigned to a corresponding data point in the initial structure. The data points in this manually-created dataset are randomly but evenly located in this initial structure.

2.2. Inverse transformation steps

The algorithm proceeds by executing a given number of *steps*, which is a user-set integer parameter (steps > 1). Default value for steps is 20. At each step, all data points are transformed towards their original location by amount

$$\frac{1}{steps}\cdot(X_{1,i}-X_{2,i}), \tag{1}$$

where $X_{1,i}$ is the location of the i:th datapoint in the original data and $X_{2,i}$ is its location in the artificial structure. After every transform, *k*-means is executed given the previous codebook along with the modified dataset as input. After all the steps have been completed, the resulting codebook C is output.

It is possible, that two points that belong to the same cluster in the original dataset will be put to different clusters in the manually-created dataset. Then they smoothly move to final locations during the inverse transform.

Algorithm 1. *K*-means*.

Input: dataset X_1, number of clusters k, *steps*,
Output: Codebook C.

$n \leftarrow size(X_1)$
$[X_2, C] \leftarrow Initialize()$
for repeats = 1 to steps **do**
  for i = 1 to n **do**
    $X_{3,i} \leftarrow X_{2,i} + (repeats/steps) * (X_{1,i} - X_{2,i})$
  end for
  $C \leftarrow kmeans(X_3, k, C)$
end for
output C

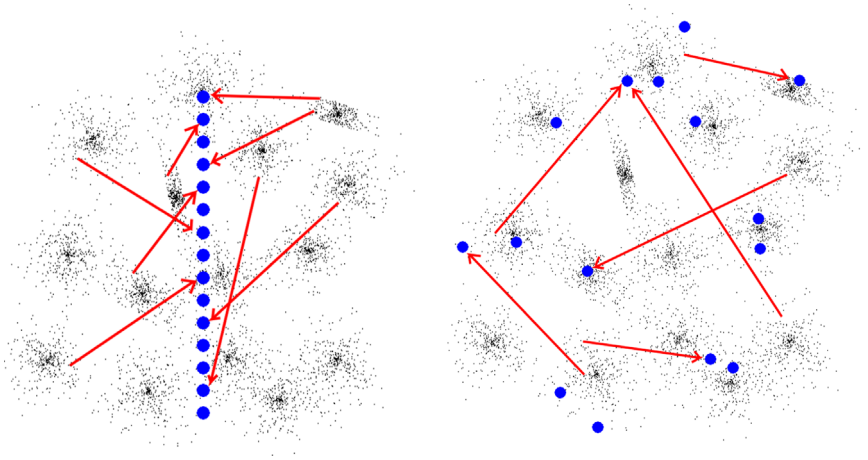

Fig. 3. Original dataset and line init (left) or random init (right) with sample mappings shown by arrows.

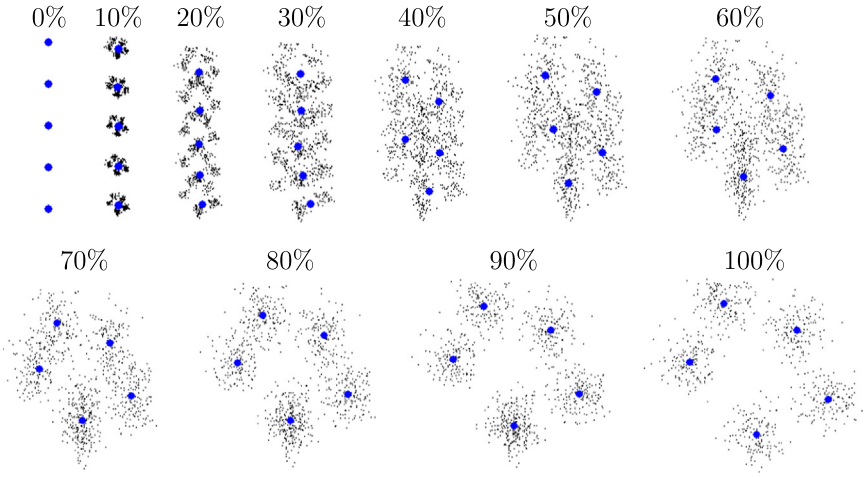

Fig. 4. Progress of the algorithm for a subset of 5 clusters of dataset a3. Data spreads towards the original dataset, and centroids follow in optimal locations. The subfigures correspond to phases 0%, 10%, 20%,…,100% completed.

2.3. Optimality considerations

The basic idea is that if the codebook was all the time optimal for all intermediate datasets, the generated final clustering would also be optimal for the original data. In fact, many times this optimality is reached; see Fig. 4 for an example how the algorithm proceeds. However, the optimality cannot be always guaranteed.

There are a couple of counter-examples, which may happen during the execution of the algorithm. The first is non-optimality of global allocation, which in some form is present in all practical clustering algorithms. Consider the setting in Fig. 5. The data points $x_{1..6}$ are traversing away from their centroid C_1. Two centroids would be needed there, one for the data points $x_{1..3}$ and another one for the data points $x_{4..6}$. On the other hand, the data points $x_{13..15}$ and $x_{16..18}$ are approaching each other and only one of the centroids C_3 or C_4 would be needed. This counter-example shows that this algorithm cannot guarantee optimal result, in general.

2.4. Empty cluster generation

Another undesired situation that may happen during the clustering is generation of an empty cluster, see Fig. 6. Here the data points $x_{1..6}$ are traversing away from their centroid C_2 and eventually leave the cluster empty. This is undesirable, because one cannot execute k-means with an empty cluster. However, this problem is easy to detect and can be fixed in most cases by a random swap strategy [12]. Here the problematic centroid is swapped to a new location randomly chosen from the data points. We move the centroids of empty clusters in the same manner.

2.5. Time complexity

The worst case complexities of the phases are listed in Table 1. The overall time complexity is not more than for the k-means, see Table 1. The proposed algorithm is asymptotically faster than global k-means and even faster than the fast variant of global k-means, see Table 2.

The derivation of the complexities in Table 1 is straightforward, and we therefore discuss here only the empty cluster detection and removal phases. There are n data points, which will be assigned to k centroids. To detect empty clusters we have to go

Table 1
Time complexity of the proposed algorithm.

Algorithm	k free	$k=O(n)$	$k=O(\sqrt{n})$	$k=O(1)$
Theoretical				
Initialization	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Dataset transform	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Empty clusters removal	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
k-means	$O(kn^{kd+1})$	$O(n^{O(n)\cdot d+2})$	$O(n^{O(\sqrt{n}d+3/2)})$	$O(n^{kd+1})$
Algorithm total	$O(kn^{kd+1})$	$O(n^{O(n)\cdot d+2})$	$O(n^{O(\sqrt{n}d+3/2)})$	$O(n^{kd+1})$
Fixed k-means				
Initialization	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Dataset transform	$O(n)$	$O(n)$	$O(n)$	$O(n)$
Empty clusters removal	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
k-means	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$
Algorithm total	$O(kn)$	$O(n^2)$	$O(n^{1.5})$	$O(n)$

Table 2
Time complexity comparison for k-means* and global k-means.

Algorithm	Time complexity for fixed k-means
Global k-means	$O(n\cdot k\cdot$ complexity of $k-$means$)=O(k^2\cdot n^2)$
Fast global k-means	$O(k\cdot$ complexity of $k-$means$)=O(k^2\cdot n)$
K-means*	$O($steps$\cdot$ complexity of $k-$means$)=O($steps$\cdot k\cdot n)$

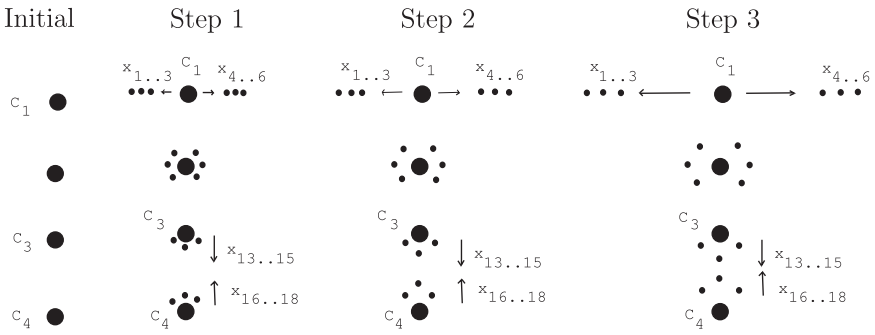

Fig. 5. Clustering that leads to non-optimal solution.

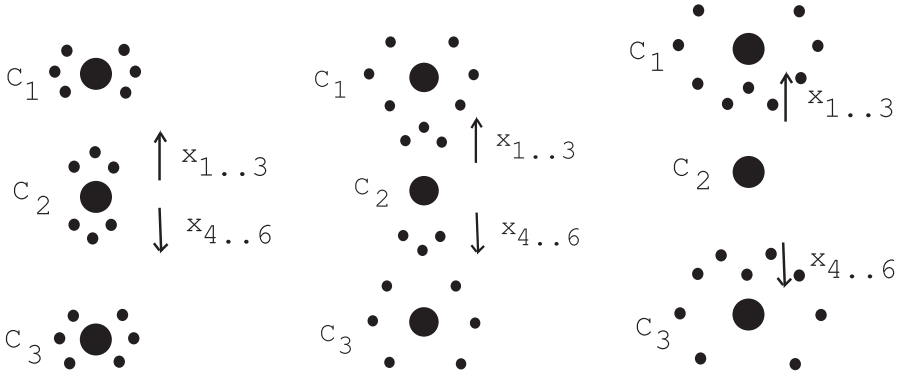

Fig. 6. A progress, which leads to an empty cluster.

through all the n points and find for them the nearest of the k centroids. So detecting empty clusters takes $O(kn)$ time.

For the empty clusters removal phase, we introduce two variants. The first is a one, which is more accurate, but slower, $O(k^2n)$ in time complexity. The second is a faster variant with $O(kn)$ time complexity. We present now first the accurate and then the fast variant.

Accurate removal: For the removal phase, there are k centroids, and therefore, at most $k-1$ empty clusters. Each empty cluster is replaced by a new location from one of the n datapoints. The new location is chosen so that it belongs to a cluster with more than one point. To find such a location takes $O(k)$ time in the worst case. The number of points in a cluster is calculated in the detection phase. Also, the new location is chosen so that there is not another centroid in that location. To check this it takes $O(k)$ time per location. After changing centroid location we have to detect again empty clusters. This loop together with the detection we repeat until all the at most $k-1$ empty clusters are filled. So the total time complexity for empty cluster removals is $O(k^2n)$.

Fast removal: In the detection phase, also the number of points per cluster and the nearest data points from the centroids of the non-empty clusters are calculated. The subphases of the removal are as follows:

- Move the centroids of the non-empty clusters to the calculated nearest data points, $T_1 = O(k)$.
- For all the $< k$ centroids, that form the empty clusters:
  - choose the biggest cluster, that has more than one data point, $T_2 = O(k)$.
  - choose the first free data point from this cluster, and put the centroid there, $T_3 = O(n)$.
  - re-partition this cluster, $T_4 = O(n)$.

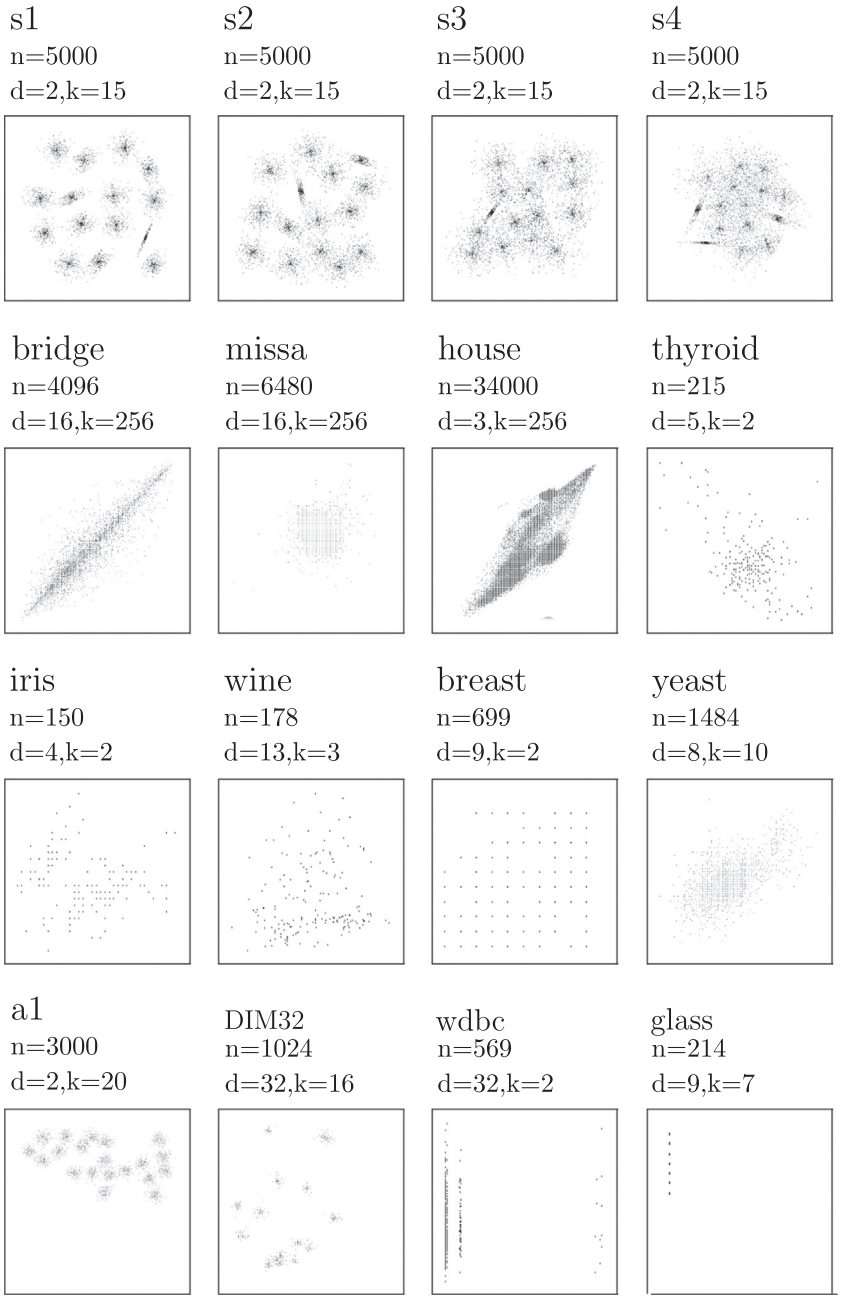

Fig. 7. Datasets s1–s4, and first two dimensions of the other datasets.

The total time complexity of removals is $T_1+k\cdot(T_2+T_3+T_4)=O(kn)$. This variant suffers somewhat from the fact that the centroids are moved to their nearest datapoints to ensure non-empty clusters.

Theoretically, k-means is the bottleneck of the algorithm. In the worst case, it takes $O(kn^{kd+1})$ time, which results in total time complexity of $O(n^{kd+1})$ when k is constant. This over-estimates the expected time complexity, which in practice, can be significantly lower. By limiting the number of k-means iterations to a constant, the time complexity reduces to linear time $O(n)$, when k is constant. When k equals to $\sqrt{n}$, the time complexity is $O(n^{1.5})$.

3. Experimental results

We ran the algorithm with a different number of steps and for several datasets. For MSE calculation we use the formula

$$\mathrm{MSE}=\frac{\sum_{j=1}^{k}\sum_{X_i\in C_j}\|X_i-C_j\|^2}{n\cdot d},$$

where MSE is normalized per feature. Some of the datasets used in the experiments are plotted in Fig. 7. All the datasets can be found in the SIPU web page http://cs.uef.fi/sipu/datasets. Some intermediate datasets and codebooks for a subset of a3 were plotted already in Fig. 4. The sets s1, s2, s3 and s4 are artificial datasets consisting of Gaussian clusters with the same variance but increasing overlap. Given 15 seeds, data points are randomly generated around them. In a1 and DIM sets the clusters are clearly separated whereas in s1–s4 they are more overlapping. These sets are chosen because they are still easy enough for a good algorithm to find the clusters correctly but hard enough for a bad algorithm to fail. We performed several runs by varying the number of steps between 1..20, 1000, 100,000, and 500,000. Most relevant results are collected in Table 3, and the results for the number of steps 2..20 are plotted in Fig. 8.

From the experience we observe that 20 steps are enough for this algorithm (Fig. 8 and Table 3). Many clustering results of these datasets stabilize at around 6 steps. More steps give only a marginal additional benefit, but at the cost of longer execution time. For some of the datasets, even just 1 step gives the best result. In these cases, initial positions for centroids just happen to be good. Phases of clustering show that 1 step gives as good result as 2 steps for a particular run for a particular dataset (Fig. 9). When the number of steps is large, the results sometimes get worse, because the codebook stays too tightly in a local optimum and the change of dataset is too marginal.

We tested the algorithm against k-means, k-means++ [7], global k-means [14] and repeated k-means. As a comparison, we made also runs with alternative structures. The results indicate that, on average, the best structures are the initial structure used in k-means++ and the random, see Table 4. The proposed algorithm with the k-means++-style initialization structure is better than k-means++ itself in the case of 15 out of 19 datasets. For one dataset the results are equal and for three datasets it is worse. These results show that the proposed algorithm is favorable to k-means++. The individual cases when it fails are due to statistical reasons. A clustering algorithm cannot be guaranteed to be better than other in every case. In real-world applications, k-means is often applied by repeating it several times starting from different random initializations and the best solution is kept finally. The intrinsic difference between our approach and the above trick is that we use educated calculation to obtain the centroids to current step, where the previous steps contribute to the current step, whereas repeated k-means initializes randomly at every repeat. From Table 5, we can see that the proposed algorithm is significantly better than k-means and k-means++. In most cases, it competes equally with repeated k-means, but in the case of high dimensionality datasets it works significantly better.

For high-dimensional clustered data, k-means++-style initial structure works best. We therefore recommend this initialization for high-dimensional unknown distributions. In most other cases, the random structure is equally good and can be used as an alternative, see Table 4.

Overall, different initial artificial structures lead to different clustering results. Our experiments did not reveal any unsuccessful cases in this. The worst results were obtained by random structure with optimal partition, but even for it, the results were at the same level as that of k-means. We did not observe any systematic dependency between the result and the size, dimensionality or type of data.

The method can also be considered as a post-processing algorithm similarly as k-means. We tested the method with the initial structure given by (complete) k-means, (complete) k-means++ and by Random Swap [12] (one of the best methods available). Results for these have been added in Table 6. We can see that the results for the proposed method using Random Swap as preprocessing are significantly better than running Repeated k-means.

We calculated also Adjusted Rand index [21], Van Dongen index [22] and Normalized mutual information index [23], to validate the clustering quality. The results in Table 7 indicate that the proposed method has a clear advantage over k-means.

Finding optimal codebook with high probability is another important goal of clustering. We used dataset s2 to compare the results of

Table 3
MSE for dataset s2 as a function of number of steps. K-means++-style structure. Mean of 200 runs except when steps ≥ 1000. (*) estimated from the best known result in [11].

Number of steps (k-means*) or repeats (repeated k-means)	K-means*		Repeated k-means	
	MSE ($\times 10^9$)	Time	MSE ($\times 10^9$)	Time
2	1.55	1 s	1.72	0.1 s
3	1.54	1 s	1.65	0.1 s
4	1.57	1 s	1.56	0.1 s
20	1.47	2 s	1.35	1 s
100	1.46	5 s	1.33	3 s
1000	1.45	24 s	1.33	9 s
100,000	1.33	26 min	1.33	58 min
500,000	1.33	128 min	1.33	290 min
K-means	1.94	0.2 s		
Global k-means	1.33	6 min		
Fast global k-means	1.33	3 s		
Optimal*	1.33	N/A		

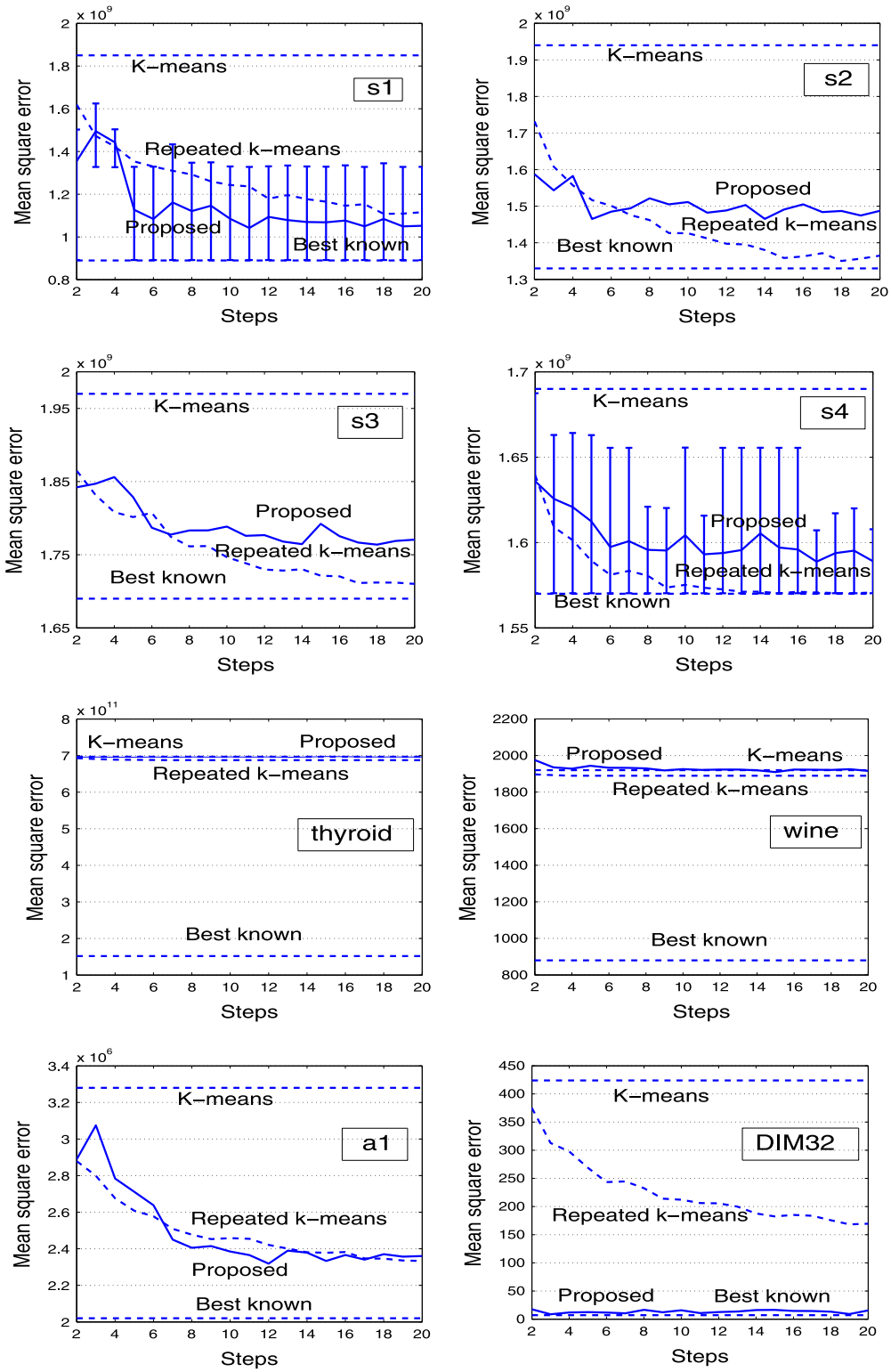

Fig. 8. Results of the algorithm (average over 200 runs) for datasets s1, s2, s3, s4, thyroid, wine, a1 and DIM32 with a different number of steps. For repeated *k*-means there are equal number of repeats than there are steps in the proposed algorithm. For s1 and s4 sets also 75% error bounds are shown. Step size 20 will be selected.

the proposed algorithm (using 20 steps), and results of the *k*-means and *k*-means++ algorithms to the known ground truth codebook of s2. We calculated how many clusters are mis-located, i.e., how many swaps of centroids would be needed to correct the global allocation of a codebook to that of the ground truth. Of the 50 runs, 18 ended up to the optimal allocation, whereas *k*-means succeeded only with 7 runs, see Table 8. Among these 50 test runs the proposed algorithm had never more than 1 incorrect cluster allocation, whereas *k*-means had up to 4 and *k*-means++ had up to 2 in the worst case. Fig. 10 demonstrates typical results.

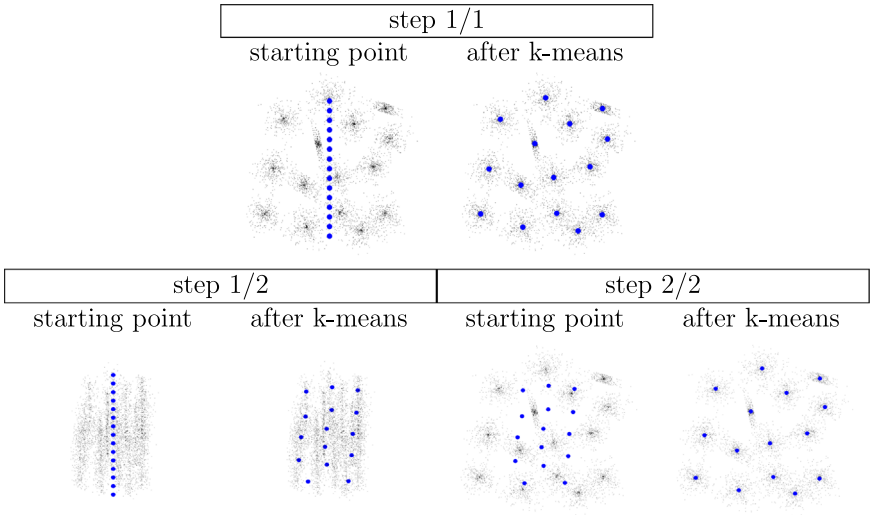

Fig. 9. Phases of clustering for 1 step and 2 steps for dataset s2.

Table 4
MSE for different datasets, averages over several (≥ 10) runs, 10 or 20 steps are used. Most significant digits are shown.

Dataset	*K*-means*					
	Diagonal	Line	Random	*k*-means++ style	Random + optimal partition	Line with uneven clusters
s1	1.21	**1.01**	1.22	1.05	1.93	1.04
s2	1.65	1.52	1.41	**1.40**	2.04	1.46
s3	1.75	**1.71**	1.74	1.78	1.95	1.73
s4	1.67	1.63	1.60	**1.59**	1.70	1.64
a1	2.40	2.40	2.35	2.38	3.07	**2.25**
DIM32	151	136	64	**7.10**	517	113
DIM64	98	168	65	**3.31**	466	157
DIM128	153	92	101	**2.10**	573	132
DIM256	135	159	60	**0.92**	674	125
Bridge	**165**	**165**	**165**	**165**	167	168
Missa	**5.11**	5.15	5.24	5.19	5.32	5.16
House	9.67	**9.48**	9.55	9.49	9.80	9.88
Thyroid	6.93	**6.92**	6.96	6.96	6.98	**6.92**
Iris	**2.33**	**2.33**	**2.33**	2.42	2.42	**2.33**
Wine	**1.89**	1.90	**1.89**	1.93	1.92	**1.89**
Breast	**3.13**	**3.13**	**3.13**	**3.13**	**3.13**	**3.13**
Yeast	0.044	0.051	**0.037**	0.041	0.039	0.050
wdbc	**2.62**	**2.62**	**2.62**	**2.62**	**2.62**	**2.62**
Glass	**0.22**	0.23	**0.22**	**0.22**	0.23	0.24
Best	7	8	7	10	2	6

Table 5
MSE for different datasets, averages over several (≥ 10) runs. Most significant digits are shown. (*) The best known results are obtained from among all the methods or by 2 h run of random swap algorithm [12].

Dataset	Dimensionality	*K*-means	Repeated *k*-means	*K*-means++	*K*-means* (proposed)	Fast GKM	Best known*
s1	2	1.85	1.07	1.28	1.05	**0.89**	**0.89**
s2	2	1.94	1.38	1.55	1.40	**1.33**	**1.33**
s3	2	1.97	1.71	1.95	1.78	**1.69**	**1.69**
s4	2	1.69	**1.57**	1.70	1.59	**1.57**	**1.57**
a1	2	3.28	2.32	2.66	2.38	**2.02**	**2.02**
DIM32	32	424	159	7.18	**7.10**	**7.10**	**7.10**
DIM64	64	498	181	3.39	**3.31**	3.39	**3.31**
DIM128	128	615	276	2.17	**2.10**	2.17	**2.10**
DIM256	256	671	296	0.99	**0.92**	0.99	**0.92**
Bridge	16	168	166	177	165	164	**161**
Missa	16	5.33	5.28	5.62	5.19	5.34	**5.11**
House	3	9.88	9.63	6.38	9.49	5.94	**5.86**
Thyroid	5	6.97	6.88	6.96	6.96	**1.52**	**1.52**
Iris	4	3.70	2.33	2.60	2.42	**2.02**	**2.02**
Wine	13	1.92	1.89	0.89	1.93	**0.88**	**0.88**

Table 5 (*continued*)

Dataset	Dimensionality	*K*-means	Repeated *k*-means	*K*-means++	*K*-means* (proposed)	Fast GKM	Best known*
Breast	9	**3.13**	**3.13**	3.20	**3.13**	**3.13**	**3.13**
Yeast	8	0.0041	**0.0038**	0.061	0.041	0.0039	**0.0038**
wdbc	31	2.62	2.61	**1.28**	2.62	2.62	**1.28**
Glass	9	0.16	**0.15**	0.28	0.22	0.16	**0.15**
Best		1	4	1	5	10	19

Table 6
MSE for *k*-means* as postprocessing, having different clustering algorithms as preprocessing. Averages over 20 runs, 20 steps are used. Most significant digits are shown.

Dataset	Repeated *k*-means	*K*-means*			
		K-means	*K*-means++	Random swap, 20 swap trials	Random swap, 100 swap trials
s1	1.07	0.99	1.08	0.99	**0.89**
s2	1.38	1.53	1.51	1.46	**1.33**
s3	1.71	1.80	1.76	1.77	**1.69**
s4	**1.57**	1.58	1.59	1.59	**1.57**
a1	2.32	2.54	2.37	2.31	**2.02**
DIM32	159	79.4	11.68	44.8	**7.10**
DIM64	181	59.4	**3.31**	48.5	9.35
DIM128	276	44.7	**2.10**	67.9	**2.10**
DIM256	296	107.1	**0.92**	16.2	16.5
Bridge	166	**164**	**164**	**164**	**164**
Missa	5.28	5.20	5.19	5.19	**5.18**
House	9.63	9.43	9.42	9.42	**9.30**
Thyroid	**6.88**	6.95	6.93	6.89	**6.88**
Iris	**2.33**	**2.33**	**2.33**	2.38	**2.33**
Wine	**1.89**	1.93	1.93	1.90	**1.89**
Breast	**3.13**	**3.13**	**3.13**	**3.13**	**3.13**
Yeast	**0.0038**	0.042	0.040	0.039	**0.0038**
wdbc	**2.61**	2.62	2.62	2.62	2.62
Glass	**0.15**	0.21	0.21	0.21	**0.15**
Best	8	3	6	2	16

Table 7
Adjusted Rand, Normalized Van Dongen and NMI indices for s-sets. Line structure (Rand), *K*-means++ initialization structure (NVD and NMI), 10 steps, mean of 30 runs (Rand) and mean of 10 runs (NVD and NMI). Best value for Rand is 1, for NVD it is 0 and for NMI it is 1.

Adjusted rand

Dataset	*k*-means	Proposed	GKM
s1	0.85	0.98	1.00
s2	0.86	0.93	0.99
s3	0.83	0.95	0.96
s4	0.83	0.87	0.94

NMI

Dataset	*k*-means	Proposed	GKM
s1	0.94	0.98	1.00
s2	0.96	0.97	0.99
s3	0.91	0.93	0.97
s4	0.91	0.93	0.95

Normalized Van Dongen

Dataset	*k*-means	GKM	Proposed
s1	0.08	0.03	0.001
s2	0.04	0.04	0.004
s3	0.09	0.06	0.02
s4	0.09	0.04	0.03

Table 8
Occurrences of wrong clusters obtained by the *k*-means, *k*-means++ and proposed algorithms in 50 runs for s2.

Incorrect clusters	*K*-means (%)	*k*-means++ (%)	Proposed (line structure) (%)
0	14	28	36
1	38	60	64
2	34	12	0
3	10	0	0
4	2	0	0
Total	100	100	100

The reason why the algorithm works well is that starting from an artificial structure, we have an optimal clustering. Then, when making the gradual inverse transform, we do not have to optimize the structure of clustering (it is already optimal). It is enough that the data points move one by one from clusters to others by *k*-means operations. The operation is the same as in *k*-means, but the clustering of the starting point is already optimal. If the structure remains optimal during the transformation, an optimal result will be obtained. Bare *k*-means cannot do this except only in special cases, that is usually is tried to compensate by using Repeated *k*-means or *k*-means++.

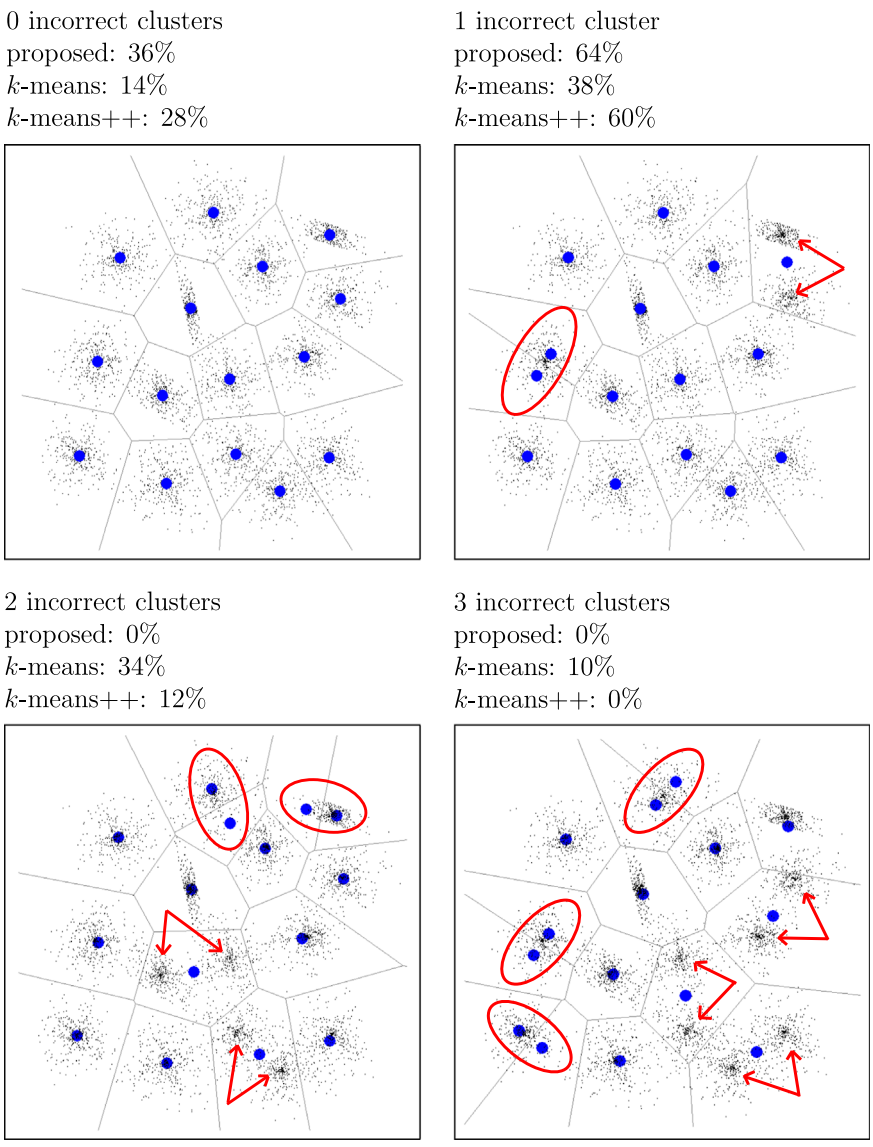

Fig. 10. Sample runs of the *k*-means and the proposed algorithm and frequencies of 0–3 incorrect clusters for dataset s2 out of 50 test runs.

4. Conclusions

We have proposed an alternative approach for clustering by fitting the data to the clustering model and not vice versa. Instead of solving the clustering problem as such, the problem is to find a proper inverse transform from the artificial data with optimal cluster allocation, to the original data. Although it cannot solve all pathological cases, we have demonstrated that the algorithm, with a relatively simple design, can solve the problem in many cases.

The method is designed as a clustering algorithm where the initial structure is not important. We only considered simple structures, of which the initialization of *k*-means++ is most complicated (note that entire *k*-means++ is not applied). However, it could also be considered as a post-processing algorithm similarly as *k*-means. But then it is not limited to be post-processing to *k*-means++ but for any other algorithm.

Future work is how to optimize the number of steps in order to avoid extensive computation but still retain the quality. Adding randomness to the process could also be used to avoid the pathological cases. The optimality of these variants and their efficiency in comparison to other algorithms have also theoretical interest.

Conflict of interest statement

None declared.

References

[1] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, Mach. Learn. 75 (2009) 245–248.
[2] M. Inaba, N. Katoh, H. Imai, Applications of weighted voronoi diagrams and randomization to variance-based *k*-clustering, in: Proceedings of the 10th Annual ACM Symposium on Computational Geometry (SCG 1994), 1994, pp. 332–339.
[3] J. MacQueen, Some methods of classification and analysis of multivariate observations, in: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability, vol. 1, 1967, pp. 281–296.
[4] D. Arthur, S. Vassilvitskii, How slow is the *k*-means method?, in: Proceedings of the 2006 Symposium on Computational Geometry (SoCG), 2006, pp. 144–153.
[5] J.M. Peña, J.A. Lozano, P. Larrañaga, An empirical comparison of four initialization methods for the K-means algorithm, Pattern Recognit. Lett. 20 (10) (1999) 1027–1040.
[6] D. Steinley, M.J. Brusco, Initializing K-means batch clustering: a critical evaluation of several techniques, J. Class. 24 (1) (2007) 99–121.
[7] D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, in: SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2007, pp. 1027–1035.

[8] D. MacKay, "Chapter 20. An Example Inference task: Clustering", in: Information Theory, Inference and Learning Algorithms, Cambridge University Press, Cambridge, 2003, pp. 284–292.
[9] W.H. Equitz, A new vector quantization clustering algorithm, IEEE Trans. Acoust. Speech Signal Process. 37 (1989) 1568–1575.
[10] P. Fränti, O. Virmajoki, V. Hautamäki, Fast agglomerative clustering using a k-nearest neighbor graph, IEEE Trans. Pattern Anal. Mach. Intell. 28 (11) (2006) 1875–1881.
[11] P. Fränti, O. Virmajoki, Iterative shrinking method for clustering problems, Pattern Recognit. 39 (5) (2006) 761–765.
[12] P. Fränti, J. Kivijärvi, Randomized local search algorithm for the clustering problem, Pattern Anal. Appl. 3 (4) (2000) 358–369.
[13] D. Pelleg, A. Moore, X-means: extending k-means with efficient estimation of the number of clusters, in: Proceedings of the Seventeenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 2000, pp. 727–734.
[14] A. Likas, N. Vlassis, J. Verbeek, The global k-means clustering algorithm, Pattern Recognit. 36 (2003) 451–461.
[15] A.P. Dempster, N.M. Laird, D.B. Rubin, Maximun likelihood from incomplete data via the EM algorithm, J. R. Stat. Soc. B 39 (1977) 1–38.
[16] Q. Zhao, V. Hautamäki, I. Kärkkäinen, P. Fränti, Random swap EM algorithm for finite mixture models in image segmentation, in: 16th IEEE International Conference on Image Processing (ICIP), 2009, pp. 2397–2400.
[17] C.H.Q. Ding, X. He, H. Zha, M. Gu, H.D. Simon, A min-max cut algorithm for graph partitioning and data clustering, in: Proceedings of IEEE International Conference on Data Mining (ICDM), 2001, pp. 107–114.
[18] M.I. Malinen, P. Fränti, Clustering by analytic functions, Inf. Sci. 217 (0) (2012) 31–38.
[19] A. Ahmadi, F. Karray, M.S. Kamel, Model order selection for multiple cooperative swarms clustering using stability analysis, Inf. Sci. 182 (1) (2012) 169–183.
[20] C. Zhong, D. Miao, P. Fränti, Minimum spanning tree based split-and-merge: a hierarchical clustering method, Inf. Sci. 181 (16) (2011) 3397–3410.
[21] L. Hubert, P. Arabie, Comparing partitions, J. Class. 2 (1) (1985) 193–218.
[22] S. van Dongen, Performance criteria for graph clustering and markov cluster experiments, Technical Report INSR0012, Centrum voor Wiskunde en Informatica, 2000.
[23] N. Vinh, J. Epps, J. Bailey, Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for change, J. Mach. Learn. Res. 11 (2010) 2837–2854.

Mikko Malinen received the B.Sc. and M.Sc. degrees in communications engineering from Helsinki University of Technology, Espoo, Finland, in 2006 and 2009, respectively. Currently he is a doctoral student at the University of Eastern Finland. His research interests include data clustering and data compression.

Radu Mariescu-Istodor received the B.Sc. degree in information technology from West University of Timisoara, Romania, in 2011 and M.Sc. degree in computer science from University of Eastern Finland, in 2013. Currently he is a doctoral student at the University of Eastern Finland. His research includes data clustering and GPS trajectory analysis.

Pasi Fränti received his MSc and PhD degrees in computer science from the University of Turku, Finland, in 1991 and 1994, respectively. From 1996 to 1999 he was a postdoctoral researcher funded by the Academy of Finland. Since 2000, he has been a professor in the University of Eastern Finland (Joensuu) where he is leading the speech & image processing unit (SIPU).

Prof. Fränti has published over 50 refereed journal and over 130 conference papers. His primary research interests are in clustering, image compression and mobile location-based applications.