Iterative shrinking method for clustering problems

(submitted for publication 20.9.2004)

Pasi Fränti and Olli Virmajoki

Department of Computer Science, University of Joensuu
P.O. Box 111, FIN-80101 Joensuu, FINLAND
franti@cs.joensuu.fi

Abstract: Agglomerative clustering generates the partition hierarchically by a sequence of merge operations. We propose an alternative to the merge-based approach by removing the clusters iteratively one by one until the desired number of clusters is reached. We apply local optimization strategy by always removing the cluster that increases the distortion the least. Data structures and their update strategies are considered. The proposed algorithm is applied as a crossover method in a genetic algorithm, and compared against the best existing clustering algorithms. The proposed method provides best performance in terms of minimizing intra cluster variance.

Keywords: Clustering algorithms, vector quantization, codebook generation, agglomeration, PNN.

Statistics: 23 pages, 16 figures, 6 tables, 7900 words, 39000 characters.

1. Introduction

Clustering is an important problem that must often be solved as a part of more complicated tasks in pattern recognition, image analysis and other fields of science and engineering [1, 2, 3]. Clustering is also needed for designing a *codebook* in vector quantization [4]. The clustering problem is defined here as follows. Given a set of N data vectors $X=\{x_1, x_2, \ldots, x_N\}$, partition the data set into M clusters such that a given distortion function f is minimized.

Agglomerative clustering generates the partition hierarchically by a sequence of merge operations. The clustering starts by initializing each data vector as its own cluster. Two clusters are merged at each step and the process is repeated until the desired number of clusters is obtained. *Ward's method* [5] selects the cluster pair to be merged so that it increases the given objective function value least. In the vector quantization context, this is known as the *pairwise nearest neighbor* (*PNN*) method due to [6]. In the rest of this paper, we denote it as the *PNN method*.

The *PNN* is an attractive approach for clustering because of its conceptual simplicity and relatively good results [7]. It has also been combined with *k-means* clustering as proposed in [8], or used as a component in more sophisticated optimization methods. For example, the *PNN* method has been used in the merge phase in the *split-and-merge* algorithm [9] resulting in a good time-distortion performance, and as the crossover method in *genetic algorithm* [10], which has turned out to be the best clustering method among a wide variety of algorithms in terms of the minimizing the distortion [11].

The main restriction of the *PNN* method is that the clusters are always merged as a whole. Once the vectors have been assigned to the same cluster, it is impossible to separate them later. This restriction is not significant at the early stage of the process when merging smaller clusters but it can deteriorate the clustering performance at the later stages when merging larger clusters.

In this paper, we propose a more general approach called *iterative shrinking* (*IS*), which generates the partition by a sequence of cluster removal operations: clusters are removed one at a time by reassigning the vectors in the removed cluster to the remaining nearby clusters. The *PNN* method can be considered as a special case of the iterative shrinking, in which the vectors of the removed cluster are all forced to move to the same neighbor cluster, see Fig. 1. In the proposed approach, the vectors can be reassigned more freely as shown in Fig. 2. Apart from the difference in the removal operation, we follow the local optimality strategy of the *PNN* method, and always remove the cluster that increases the cost function value least. We also consider briefly the case where the number of clusters must also be determined.

The method is also integrated within a genetic algorithm. The proposed method and its genetic variant are extensively compared against the best existing clustering algorithms. The results show that the iterative shrinking provides competitive result for all test sets, and the variant with the genetic algorithm gives the best result among all tested algorithms in terms of minimizing the intra cluster variance. The running time of the proposed method can be rather large but we show how the genetic variant can also be tuned for better time-distortion performance. The idea of iterative shrinking and its genetic variant have been originally presented in two conference papers [12, 13]

Similar idea has been recently proposed for the opposite (incremental) direction in [14]. The method, known as *Global k-means* (GKM), generates the partition iteratively by adding one new cluster to the partition. At each step, the method considers every data vector as a potential location for the new cluster. It applies *k-means* to all candidate partitions, and keeps the one that decreases the objective function value most. The approach itself is feasible but its time complexity is rather high varying from $O(gNM^3)$ to $O(gN^2M^2)$ depending on the variant, where g is the number of *k-means* iterations applied.

The rest of the paper is organized as follows. In Section 2, we give formal definition of the clustering problem considered here, and then recall the *PNN* method. The new iterative shrinking method is then introduced in Section 3. We first present the definition of the secondary partition in Section 3.1. A straightforward solution for finding the cluster to be removed is given in Section 3.2, and its exact calculation is derived in Section 3.3. Update of the secondary partition is considered in Section 3.4. The relationship between the *PNN* and the *IS* methods is discussed in Section 3.5. The time complexities are summarized in Section 4. In Section 5, we apply the method within a genetic algorithm, and also extend the method to the case of an unknown number of clusters. Experimental results are reported in Section 6, and conclusions drawn in Section 7.

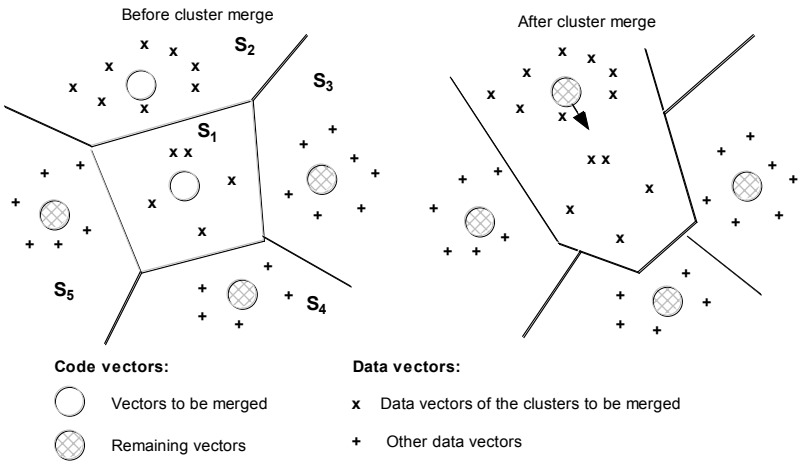


Fig. 1. The merging process of the *PNN* method.

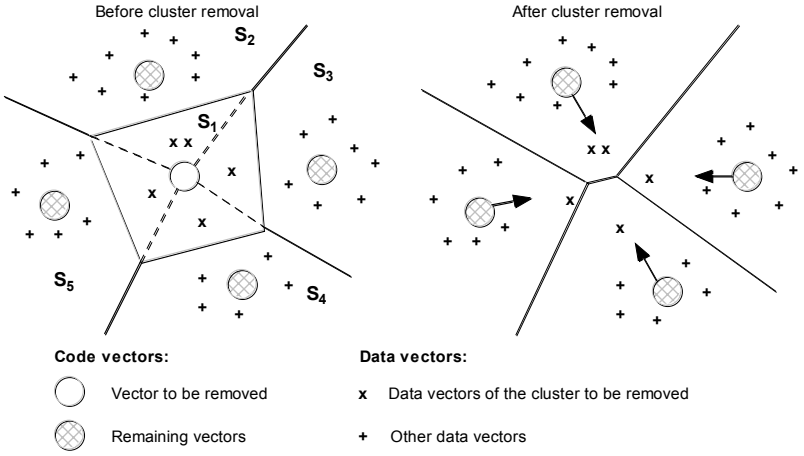


Fig. 2. The cluster removal process of iterative shrinking.

2. Pairwise nearest neighbor

Given a set of N data vectors $X=\{x_1, x_2, \ldots, x_N\}$, clustering aims at solving the partition $P=\{p_1, p_2, \ldots, p_N\}$, which defines for each data vector the index of the cluster where it belongs to. Cluster s_a is defined as the set of data vectors that belong to the same partition a:

$$s_a = \left\{ x_i \middle| p_i = a \right\}. \tag{1}$$

The clustering is then represented as the set $S=\{s_1, s_2, ..., s_M\}$. In vector quantization, the output of the clustering is a codebook $C=\{c_1, c_2, \ldots, c_M\}$, which is usually the set of cluster centroids.

The most important choice in clustering is the cost function f for evaluating the goodness of the clustering. When the data vectors belong to Euclidean space, a commonly used function is the mean square error (*MSE*) between the data vectors and their cluster centroids. Given a partition P and the codebook C, the *MSE* is calculated as:

$$MSE(C,P) = \frac{1}{N} \cdot \sum_{i=1}^{N} \left\| x_i - c_{p_i} \right\|^2 . \quad (2)$$

Ward's method [5], or the *pairwise nearest neighbor* (*PNN*) as it is known in vector quantization [5, 6], generates the clustering hierarchically by a sequence of merge operations as described in Fig. 3. In each step of the algorithm, the number of the clusters is reduced by merging two nearby clusters:

$$s_a \leftarrow s_a \cup s_b . \quad (3)$$

The cost of merging two clusters s_a and s_b is the increase in the *MSE*-value caused by the merge. It can be calculated using the following formula [5, 6]:

$$d_{a,b} = \frac{n_a n_b}{n_a + n_b} \cdot \left\| c_a - c_b \right\|^2 , \quad (4)$$

where n_a and n_b are the corresponding cluster sizes. The *PNN* method applies a local optimization strategy: all possible cluster pairs are considered and the one increasing *MSE* least is chosen:

$$a,b = \arg \min_{\substack{i,j \in [1,m] \\ i \neq j}} d_{i,j} , \quad (5)$$

where m is the current number of clusters. There exist many variants of the *PNN* method. A straightforward implementation recalculates all distances at each step of the algorithm. This takes $O(N^3)$ time because there are $O(N)$ steps in total, and $O(N^2)$ cluster pairs to be checked at each step.

Another approach is to maintain an $N \times N$ matrix of the merge cost values. The merge cost values are needed to be updated only for the newly merged cluster. Nevertheless, the algorithm still requires $O(N^3)$ because the search of the minimum cluster pair takes $O(N^2)$ time [15]. Kurita's method [16] maintains an $N \times N$ matrix but it also utilizes a heap structure for searching the minimum distance. The method thus runs in $O(N^2 \cdot \log N)$ time. The storage of the matrix, however, requires $O(N^2)$ memory, which makes these variants impractical for large data sets.

A fast implementation with linear memory consumption of the *PNN* method is obtained by maintaining a pointer from each cluster to its nearest neighbor, and the corresponding merge cost value [17]. The cluster pair to be merged can be found in $O(N)$ time, and only a small number (denoted by τ) of the nearest neighbor needs to be updated after each merge. The implementation takes $O(\tau N^2)$ time in total. Further speed-up can be achieved by using lazy update of the merge cost values [18], and by reducing the amount of work caused by the distance calculations [7].

All the variants cited above give either asymptotic or relative improvement in the time complexity but they do not provide any improvements in the clustering quality. The clustering result is therefore bounded by the fundamental restriction caused by the merge step of the *PNN* method. The only way to improve the quality of the partition is to replace the merge step by a more general solution.

```
PNN(X, M) → S
    FOR i←1 to N DO
        s_i ← {x_i};
    REPEAT
        (s_a, s_b) ← SearchNearestClusters(S);
        Merge(s_a, s_b);
    UNTIL |S|=M;
```

Fig. 3. Structure of the *PNN* method.

3. Iterative shrinking

Iterative Shrinking (*IS*) starts by assigning each data vector to its own cluster. It then removes one cluster at a time until the desired number of clusters has been reached. The data vectors of the removed cluster are repartitioned to the nearby clusters. The centroids of the neighbor clusters are updated according to the changes. The general structure of the *IS* algorithm is presented in Fig. 4, and the details are discussed in the following subsections.

```
IS(X, M) → S
    FOR i←1 to N DO
        s_i ← {x_i};
    REPEAT
        s_a ← SelectClusterToBeRemoved(S);
        RemoveCluster(S, s_a);
    UNTIL |S|=M;
```

Fig. 4. Structure of the IS method.

3.1. Finding secondary cluster

For the cluster removal, we need to find the second nearest cluster for each data vector in the selected cluster. We therefore maintain the *secondary partition* $Q=\{q_1, q_2, \ldots, q_N\}$ for every data vector. It can be generated in a similar manner to the primary partition but excluding the current nearest cluster in the search:

$$q_i = \arg\min_{\substack{1\le j\le m \\ j\neq p_i}} \left\| x_i - c_j \right\|^2 . \tag{6}$$

The squared Euclidean distance, however, does not take into account the centroid update, which will take place after the removal process. It is therefore more accurate to apply the merge cost function of (4), and measure the cost of merging the data vector to the neighbor cluster s_j instead of the mere distance to the cluster centroid:

$$q_i = \arg\min_{\substack{1\le j\le m \\ j\neq p_i}} \frac{n_j}{n_j + 1} \left\| x_i - c_j \right\|^2 . \tag{7}$$

Now the cost function will put more weight on larger clusters as their centroids are less likely to move, and less to smaller clusters.

3.2. Selecting cluster to be removed

We adopt the local optimization strategy of the *PNN* method and select the cluster to be removed as the one that increases the cost function least. Because the data vectors of the removed cluster can be divided among several neighbor clusters, we calculate the effect of the removal cost for each data vector separately. We first determine how much the cost function will increase if the data vector x_i is merged to its secondary cluster s_{q_i}, and then how much it will decrease when the data vector is removed from its current cluster s_a. The net effect of the change is the difference:

$$\Delta D_i = \frac{n_{q_i}}{n_{q_i}+1}\left\|x_i - c_{q_i}\right\|^2 - \left\|x_i - c_a\right\|^2 . \tag{8}$$

The removal cost of cluster s_a can now be estimated as the sum of the individual move costs of the data vectors:

$$d_a = \sum_{x_i \in s_a} \Delta D_i . \tag{9}$$

We refer this as the *simple calculation* of the removal cost. It gives a correct result if every data vector moves to a different neighbor cluster. In practice, several data vectors can move to the same neighbor cluster and they all affect on the movement of the cluster centroid. The equation (8) is therefore not accurate because it does not take into account the overall movement of the code vectors. Instead, it tends to over estimate the cost function when more vectors are moving to the same destination cluster.

3.3. Exact calculation of the removal cost

To realize the *exact calculation* for the removal cost, we divide the data vectors x_i in s_a into subclusters $s_{a,j}$ according to their secondary partition q_i:

$$s_{a,j} = \left\{x_i \in s_a \middle| q_i = j\right\}. \tag{10}$$

For example, there are five data vectors of the cluster s_1 divided into four subclusters in Fig. 2. The removal is conceptually considered as a three step process: (1) remove the vectors from the current cluster s_a, (2) form the subclusters $s_{a,j}$, and (3) merge the subclusters to the neighbor clusters s_j. Thus, the removal cost is composed of the three terms corresponding to this process:

$$\begin{aligned} d_a &= -\sum_{x_i \in s_a}\left\|c_a - x_i\right\|^2 + \sum_{j=1}^{m}\sum_{x_i \in s_{a,j}}\left\|c_{a,j} - x_i\right\|^2 + \sum_{j=1}^{m}\frac{n_j \cdot \left|s_{a,j}\right|}{n_j + \left|s_{a,j}\right|}\left\|c_j - c_{a,j}\right\|^2 \\ &= -\sum_{j=1}^{m}\left|s_{a,j}\right| \cdot \left\|c_a - c_{a,j}\right\|^2 + \sum_{j=1}^{m}\frac{n_j \cdot \left|s_{a,j}\right|}{n_j + \left|s_{a,j}\right|}\left\|c_j - c_{a,j}\right\|^2 , \end{aligned} \tag{11}$$

where $\left|s_{a,j}\right|$ is the size of the subcluster $s_{a,j}$. The first term is the sum of the distances to the current cluster centroid c_a, i.e. the cost of the cluster before removal. The second term is the sum of the cost values inside the subclusters, where $c_{a,j}$ represents the centroid of the subcluster. The third term is the sum of the costs of merging the subclusters $s_{a,j}$ to their neighbor clusters s_j.

The Equation (11) gives the exact removal cost, and provides the result of the local optimization strategy as desired. The situation, however, is not as simple as this because the optimality is restricted by the choice of the secondary partition Q. Equation (7), for example, assumes that the vectors are moved independently from each other. The movement of the vector, however, has an effect on the cluster centroid, and therefore, indirectly affects the removal cost values of other data vectors, too.

The problem is that there is no way to determine the best moving sequence without trying all possible combinations. The only reasonable choice is therefore to apply some kind of heuristic. We content ourselves with the one given in (7), in which the partition of all data vectors is determined independently from each other.

3.4. Partition updates

The removal of a cluster s_a affects most of the data structures. The primary partition P is updated for the vectors in the removed cluster by copying the information from the secondary partition:

$$\forall\, x_i \in s_a : \quad p_i \leftarrow q_i \,. \tag{12}$$

The codebook C is then updated by recalculating the centroids of the affected clusters. As a consequence of this, there can be further changes both in the primary and secondary partition due to the movement of the centroids. This can affect the accuracy of the removal cost estimation if the necessary updates are not made.

We consider next different strategies for updating the secondary partition Q. For this purpose, the clusters are classified to three categories according to the location with respect to their removed cluster:

- *Removed* cluster,
- *Neighbor* clusters, and
- *All other* clusters.

A cluster s_j is defined to be a *neighbor cluster* if any data vector from the removed cluster s_a has been reassigned to s_j. The set of neighbor clusters is denoted here as Y_a:

$$Y_a = \left\{ s_j \mid \exists x_i \in s_a : q_i = j \right\}. \tag{13}$$

The secondary partition update of a single vector requires that we search its second nearest cluster among all clusters. This takes O(m) distance calculations per data vector, where m is the current number of clusters. It is therefore vital for the time complexity to make the number of updates as small as possible. We consider three alternative update strategies:

- *Minimum update*,
- *Standard update*, and
- *Extensive update*.

The data vectors that are updated in these strategies are denoted as the sets G_{minimum}, G_{standard} and $G_{\text{extensive}}$. The sets have an increasing amount of updates so that $G_{\text{minimum}} \subseteq G_{\text{standard}} \subseteq G_{\text{extensive}} \subseteq X$. The inclusion of the vectors in these three sets is summarized in Table 1 according to the type of the cluster in the primary and secondary partition. The situation is also illustrated in Fig. 5.

The *minimum update* strategy updates the secondary partition of the vectors that is only absolutely necessary. This includes all vectors in the removed clusters, and also some vectors in the neighbor clusters:

$$G_{\text{minimum}} = \{x_i \,|\, p_i = a \vee q_i = a\}. \tag{14}$$

Firstly, a new secondary cluster must be resolved for the moved vectors (p_i=a) because they have just been reassigned according to their secondary partition, see Eq. (12). Secondly, a vector in the neighboring cluster must be updated if its secondary cluster was the removed one (q_i=a).

The *standard update* includes slightly more data vectors than the minimum update:

$$G_{\text{standard}} = G_{\text{minimum}} \cup \{x_i \mid s_{p_i} \in Y_a \ \wedge \ s_{q_i} \in Y_a\}. \tag{15}$$

In other words, we update the secondary partition also for those vectors in the neighbor clusters whose secondary partition is another neighbor cluster. This provides more accurate maintenance of the secondary partition with only a moderate amount of extra work. On the other hand, the update is not mandatory.

In addition to the previous data vectors, the *extensive update* strategy contains also all data vectors that have any connection to the neighbor clusters. In other words, the update is performed for data vector x_i if either its primary or secondary partition is one of the neighbor cluster:

$$G_{\text{extensive}} = G_{\text{standard}} \cup \{x_i \mid s_{p_i} \in Y_a \ \vee \ s_{q_i} \in Y_a\}. \tag{16}$$

This update strategy covers all vectors that should be updated, and it can be performed using a reasonable amount of computation. It is expected that the number of vectors in $G_{\text{extensive}}$ is still remarkably smaller than the size of the overall data set.

Table 1. Classification of the data vectors according to the type of its primary and secondary clusters.

		Primary partition P		
		Removed	Neighbor	Other
Secondary partition Q	Removed	N/A	Minimum update	Minimum update
	Neighbor	Minimum update	Standard update	Extensive update
	Other	N/A	Extensive update	-

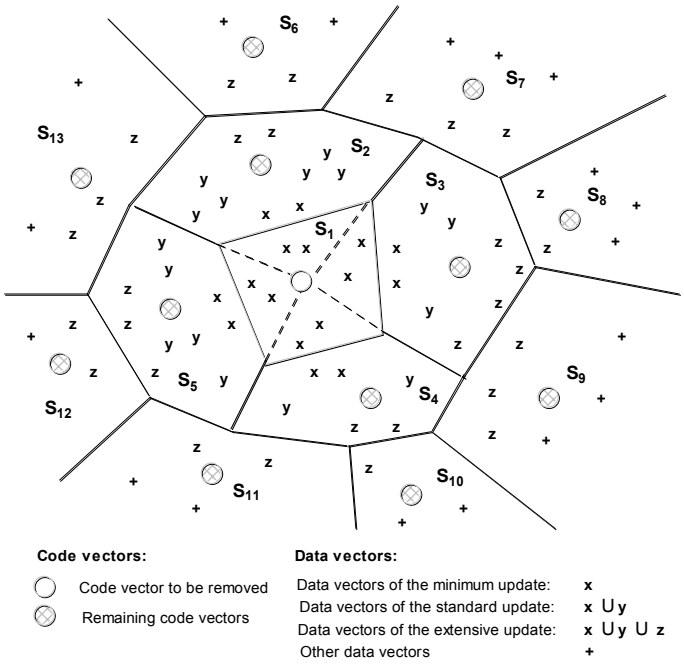


Fig. 5. Illustration of the vectors whose secondary partition is updated at the different levels of the update strategy. The removed cluster is s_1, and the neighbor clusters are s_2, s_3, s_4 and s_5.

3.5. IS versus PNN

The *PNN* method can be seen as a special case of the *IS* method, as it can be simulated by the *IS* method as follows. We first select the cluster to be removed as one of the two clusters (s_a and s_b) selected for the merge. The merge is then performed by moving all the vectors from s_b to s_a, and thus, removing s_b. The centroid of s_a is updated accordingly. The result is equivalent to that of the *PNN* method, and it is easy to see from Fig. 1 and Fig. 2 that some of the vector reassignments could be done better resulting in a smaller increase in the cost function value.

The difference of the merging and removal strategies is illustrated further in Fig. 6. We have six data vectors located symmetrically, and the task is to find a partition of two clusters. After the first three merges, the output of the *PNN* and *IS* methods are equivalent but in the fourth merge the *PNN* method is already restricted by the previous merges and the result is suboptimal. The *IS* method, on the other hand, ends up with the optimal result no matter what is the order of the previous cluster removals.

It is noted, that it is still possible (although rare) to get better result by the *PNN* method than by the *IS* method because locally optimal steps does not necessarily lead to the global optimum. Nevertheless, it is expected that the *IS* method would give better partition than the *PNN* method in most cases.

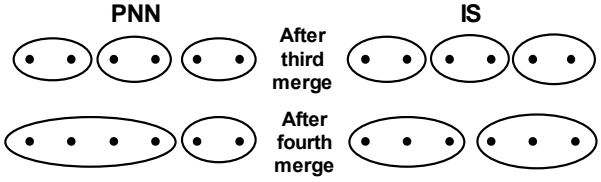


Fig. 6. Example of the different functions of the *PNN* and the *IS* methods.

4. Complexity analysis

Detailed pseudo code of the *IS* method is given in Fig. 7. The initialization phase requires $O(N^2)$ distance calculations due to the construction of the secondary partition. For simplicity, we assume here that the size of the feature vector is constant. The main loop of the algorithm is then repeated by N-M times. The most time consuming operations during the iteration are the calculation of the removal costs, and the update of the secondary partition. The simple calculation of the removal cost requires at most $O(N)$ time per iteration. The exact calculation requires a little bit more than that but still no more than $O(N)$.

```
IS(X, M) → C, P
    m ← N;
    FOR ∀ i∈[1, m]:
        c_i ← x_i;
        p_i ← i;
        n_i ← 1;
    FOR ∀ i∈[1, m]:
        q_i ← FindSecondNearestCluster(C, x_i);
    REPEAT
        CalculateRemovalCosts(C, P, Q, d);
        a ← SelectClusterToBeRemoved(d);
        RemoveCluster(P, Q, a);
        UpdateCentroids(C, P, a);
        UpdateSecondaryPartitions(C, P, Q, a);
        m ← m - 1;
    UNTIL m=M.
```

Fig. 7. Pseudo code of the *IS* method.

The update of the secondary partition requires $n \cdot m$ distance calculations, where n is the number of data vectors to be updated, and m is the number of clusters. We first estimate the number of data vectors within one cluster, and then derive the number of distance calculations in the *minimum update*, *standard update*, and the *extensive update*.

The main question is the number of data vectors in a cluster. After (N-m) removal steps, the N data vectors are divided into the m clusters so that there are N/m vectors per cluster, on average. As the algorithm tends to remove smaller clusters, it is reasonable to estimate that the number of vectors is no more than N/m, on average. If we sum it up from all N-M steps, we get the total number of vectors as:

$$\frac{N}{N}+\frac{N}{N-1}+\ldots+\frac{N}{M}=N\cdot\left(\frac{1}{M}+\frac{1}{M+1}+\ldots+\frac{1}{N}\right). \tag{17}$$

For the case M=1, this gives:

$$N \cdot \left(\frac{1}{1} + \frac{1}{2} + ... + \frac{1}{N} \right) = O(N \cdot \log N). \quad (18)$$

The algorithm actually iterates only N-M steps, so a more accurate upper bound is $N \cdot (\log N - \log M)$. Thus, the average number of vectors in the cluster is approximated by:

$$\frac{N \cdot (\log N - \log M)}{N - M} = O(\log N). \quad (19)$$

The *minimum update* processes only the vectors of the removed cluster, and the vectors that were mapped to the removed cluster in their secondary partition. The number of distance calculations per vector is not equal in all iterations but it varies from N to M, and thus, can be approximated as follows:

$$N \frac{N}{N} + (N-1) \frac{N}{N-1} + ... + M \frac{N}{M} = (N - M) \cdot N. \quad (20)$$

We assume that the same result applies both to the primary partition and secondary partitions. The number of distance calculations in the *minimum update* variant is therefore estimated as $2(N\text{-}M) \cdot N = O(N^2)$.

The *standard update* includes also vectors from the neighbor clusters. The number of distance calculations can be approximated by multiplying the result in (20) by the number of neighbor clusters $|Y|$. The obvious upper bound for $|Y|$ is the number of vectors in a single cluster as in (19). The result from this is:

$$\frac{N \cdot (\log N - \log M)}{N - M} \cdot (N - M) \cdot N = N^2 \cdot \log \frac{N}{M}. \quad (21)$$

Thus, the number of distance calculations is $O(N^2 \cdot \log(N/M))$.

The *extensive update* includes also vectors from other clusters. Suppose that we have $|Y|$ neighbor clusters, and assume that each of them have also $|Y|$ neighbors. Some of the clusters are the same but this anyway gives a simple estimation for the number of affected clusters as $|Y|^2$. The total number of distance calculations is therefore estimated as $O(N^2 \cdot \log^2 N)$.

The overall time complexities of the different *IS* variants are summarized and compared to the main *PNN* variants in Table 2. The *IS* method with *minimum update* is theoretically faster than the *PNN* method but not necessarily so in practice. This is because the number of updates (τ) in the *PNN* method is relatively small, and the removal step in the *IS* method is more complicated. The standard and extensive update variants have somewhat higher time complexities but still smaller than the $O(N^3)$ that would have been the result if all data vectors were updated at every iteration.

Table 2. Summary of the time complexities of the *exact PNN* and the *IS*.

	PNN		*Iterative shrinking*		
	Original	Fast	Minimum	Standard	Extensive
Initialization:	$O(N)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$
Single step:					
• Cluster selection	$O(N^2)$	$O(N)$	$O(N)$	$O(N)$	$O(N)$
• Merge / removal	$O(1)$	$O(1)$	$O(N)$	$O(N)$	$O(N)$
• Update	$O(1)$	$O(\tau N)$	$O(N)$	$O(N \cdot \log N/M)$	$O(N \cdot \log^2 N)$
Algorithm in total:	$O(N^3)$	$O(\tau N^2)$	$O(N^2)$	$O(N^2 \cdot \log N/M)$	$O(N^2 \cdot \log^2 N)$

5. Generalizations of IS

We next generalize iterative shrinking approach to the case when the number of clusters must also be determined. The method is then augmented with the genetic algorithm in the same way as the *PNN* method has been applied in [10, 11]. The proposed combination is denoted as *GAIS* (genetic algorithm with iterative shrinking). We consider both the fixed and variable number of clusters.

5.1. Unknown number of clusters

In many cases, the number of clusters is not known beforehand but finding the number of clusters is part of the problem. The simplest approach is to generate solutions for all possible number of clusters M in a given range $[M_{min}, M_{max}]$, and then select the best partition to a suitable evaluation function f. This multiplies the computation time by $(M_{\max}\text{-}M_{\min})$.

Iterative shrinking, on the other hand, produces solutions in the range $[N, M]$ during the same run. It is therefore enough to replace the distortion function by a suitable clustering validity index. Among the many different indexes [19, 20, 21], it was found out in [22] that *variance-ratio F-test* based on a statistical ANOVA test procedure [23] works pretty well in the case of *Gaussian* clusters. Moreover, it was shown in [24] that the same distance function can be applied with the *F-test* as with the MSE. Therefore, no additional changes are required in the algorithm because of using *F-test*.

5.2. Genetic algorithm

The idea of a genetic algorithm (*GA*) is to maintain a set of solutions which make up a population, which is iteratively improved by genetic operations such as crossover, mutation, and by the selection principle of evolution. Several different crossover algorithms have been considered in [10], and concluded that significantly better results are obtained when the *PNN* method is used as the crossover algorithm instead a straightforward approach of using random crossover and *k-means*. It is therefore logical to consider genetic algorithm with iterative shrinking as the crossover. We refer the method here as *GAIS*.

The sketch of the *GAIS* algorithm is outlined in Fig. 8, and it works as follows. The *GA* is applied here in the problem domain by operating on the codebook and partition (C, P). A set of random solutions are first generated by selecting M random data vectors as the codebook, and by creating optimal partition with respect to this codebook. The best solution survives to the next generation as such, and the rest of the population is

filled by new solutions created by crossover. The process is iterated and the best solution in the final generation is the result of the algorithm.

The crossover starts by merging the parent solutions by taking the union of their centroids (*CombineCentroids*). The partition P^{new} is then constructed on the basis of the existing partitions P^1 and P^2 (*CombinePartitions*). The partition of data vector x_i is either p_i^1 or p_i^2. The one with smaller distance to x_i is chosen. The codebook C^{new} is then updated (*UpdateCentroids*) with respect to the new partition P^{new}. This procedure gives a solution in which the codebook has twice the size desired. Empty clusters are next removed (*RemoveEmptyClusters*), and iterative shrinking is then applied to reduce the number of clusters from $2 \cdot M$ to M. Finally, the solution is fine-tuned by a few iterations of a *k-means* [25]. Note that mutations are not used here as they only slow down the convergence of the optimization process.

The number of generations (T), population size (Z), and the number of *k-means* iterations (G) are the main parameters of the algorithm. Here we consider the following two strategies:

1. *GAIS short*: Create new generations only as long as the best solution keeps improving (T=*). Use a small population size (Z=10), and apply two iterations of *k-means* (G=2).
2. *GAIS long*: Create a large number of generations (T=100) with a large population size (Z=100) and iterate *k-means* relatively long (G=10).

It is expected that the short variant is good enough to compete with the other clustering algorithms in terms of quality. The purpose of the long variant is to squeeze out the best possible result at the cost of a very long computation time.

It is also possible to apply the *GA* with unknown number of clusters. In this case, we take any initial number of clusters M_0, and generate the initial population accordingly. The new solutions in the crossover are reduced from $2M$ to 1 and the intermediate partition that minimizes F-ratio is taken as the new candidate solution. The number of clusters will be automatically determined during the optimization process of the *GA*.

GeneticAlgorithm(X) → (C, P)

FOR i←1 TO Z DO
  C^i ← RandomCodebook(X);
  P^i ← OptimalPartition(X, C^i);
SortSolutions(C,P);
REPEAT
  {C,P} ← CreateNewSolutions({C,P});
  SortSolutions(C,P);
UNTIL no improvement;

CreateNewSolutions({C, P}) → {C^{new}, P^{new} }

C^{new-1}, P^{new-1} ← C^1, P^1;
FOR i←2 TO Z DO
  (a,b) ← SelectNextPair;
  C^{new-i}, P^{new-i} ← Cross(C^a, P^a, C^b, P^b);
  IterateK-Means(C^{new-i}, P^{new-i});

Cross(C^1, P^1, C^2, P^2) → (C^{new}, P^{new})

C^{new} ← CombineCentroids(C^1, C^2);
P^{new} ← CombinePartitions(P^1, P^2);
C^{new} ← UpdateCentroids(C^{new}, P^{new});
RemoveEmptyClusters(C^{new}, P^{new});
IS(C^{new}, P^{new});

CombineCentroids(C^1, C^2) → C^{new}

C^{new} ← $C^1 \cup C^2$

CombinePartitions(C^{new}, P^1, P^2) → P^{new}

FOR i←1 TO N DO
  IF $\left\|x_i - c_{p_i^1}\right\|^2 \le \left\|x_i - c_{p_i^2}\right\|^2$ THEN
    $p_i^{new} \leftarrow p_i^1$
  ELSE
    $p_i^{new} \leftarrow p_i^2$
END-FOR

UpdateCentroids(C^1, C^2) → C^{new}

FOR j←1 TO $|C^{new}|$ DO
  c_j^{new} ← CalculateCentroid(P^{new}, j);

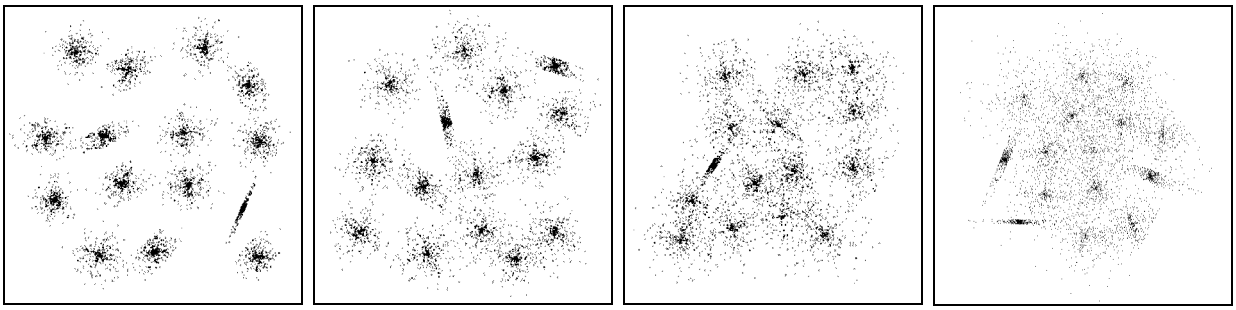
Fig. 8. Pseudo code of the *genetic algorithm* with *iterative shrinking* (*GAIS*).

6. Experiments

We consider three image data sets (Fig. 9), four synthetically generated data sets (Fig. 10), and the *BIRCH* data sets [26]. The vectors in the first set (*Bridge*) are 4×4 non-overlapping blocks taken from a gray-scale image, and in the second set (*Miss America*) 4×4 difference blocks of two subsequent frames in video sequence. The third data set (*House*) consists of color values of the *RGB* image. The number of clusters is fixed to M=256. The data sets S_1 to S_4 are two-dimensional artificially generated data sets with varying complexity in terms of spatial data distributions with M=15 predefined clusters. The summary of the data sets is presented in Table 3. All tests have been performed in Sun Enterprise 450 with 400 MHz UltraSPARC2 processor, 2 GB memory and Solaris 7 (SunOS 5.7) operating system.

Spatial vectors:	Spatial residual vectors:	Color vectors:
		
Bridge (256×256) K=16, N=4096	*Miss America* (360×288) K=16, N=6480	*House* (256×256) K=3, N=34112*

Fig. 9. Source of the data. *Duplicate data vectors are combined and frequency information is stored.



Data set S_1 Data set S_2 Data set S_3 Data set S_4

Fig. 10: Two-dimensional data sets with varying complexity in terms of spatial data distributions. The data sets have 5000 vectors scattered around 15 predefined clusters with a varying degrees of overlap.

Table 3. Summary of the data sets.

Data set	Type of data set	Number of data vectors (N)	Number of clusters (M)	Dimension of data vector (K)
Bridge	Gray-scale image	4086	256	16
House	*RGB* image	34112	256	3
Miss America	Residual vectors	6480	256	16
Data set S_1- S_4	Synthetically generated	5000	15	2
$BIRCH_1$-$BIRCH_3$	Synthetically generated	100000	100	2

6.1. Comparison of the IS variants

The clustering test results of the three data sets are summarized in Table 4 for all the variants of the *IS* method considered here, and for the fast exact *PNN* method as implemented in [17]. In all cases, the *IS* method produces smaller distortion but at the cost of about 2 to 6 times slower running time. The corresponding time-distortion performance is illustrated in Fig. 11.

The extension in the amount of updates of the secondary partition decreases the MSE but also slows down the process. The running time of the standard update is only about 4 to 9% longer than that of the minimum update whereas the extensive update increases the running time about 58 to 140% depending on the data set. In other words, the results of the minimum and standard update are rather similar to each other whereas the extensive update gives a clearer effect both in the MSE and in the running time.

The method of calculating the removal cost (simple or exact) has only a small effect on the MSE but the exact calculation is about 10 to 40% slower than the simple method. The time-distortion performance of the simple variant seems to be marginally better according to Fig. 11. On the other hand, if the MSE is the primary concern, we should use the exact calculation with the extensive update. Thus, we will fix this parameter setup in the following tests.

Table 4. The MSE values and running times (in seconds) of the *PNN* and *IS* variants for the three data sets (M=256).

		Bridge		*House*		*Miss America*	
		Running time	MSE	Running time	MSE	Running time	MSE
PNN		272	168.92	4391	6.27	709	5.36
Simple IS	Minimum update	315	166.18	9614	6.11	824	5.24
	Standard update	324	166.08	9997	6.12	874	5.23
	Extensive update	564	164.22	16043	6.10	1820	5.19
Exact IS	Minimum update	481	165.93	12288	6.15	1283	5.24
	Standard update	499	165.44	13161	6.10	1334	5.23
	Extensive update	705	163.38	19280	6.11	2290	5.19

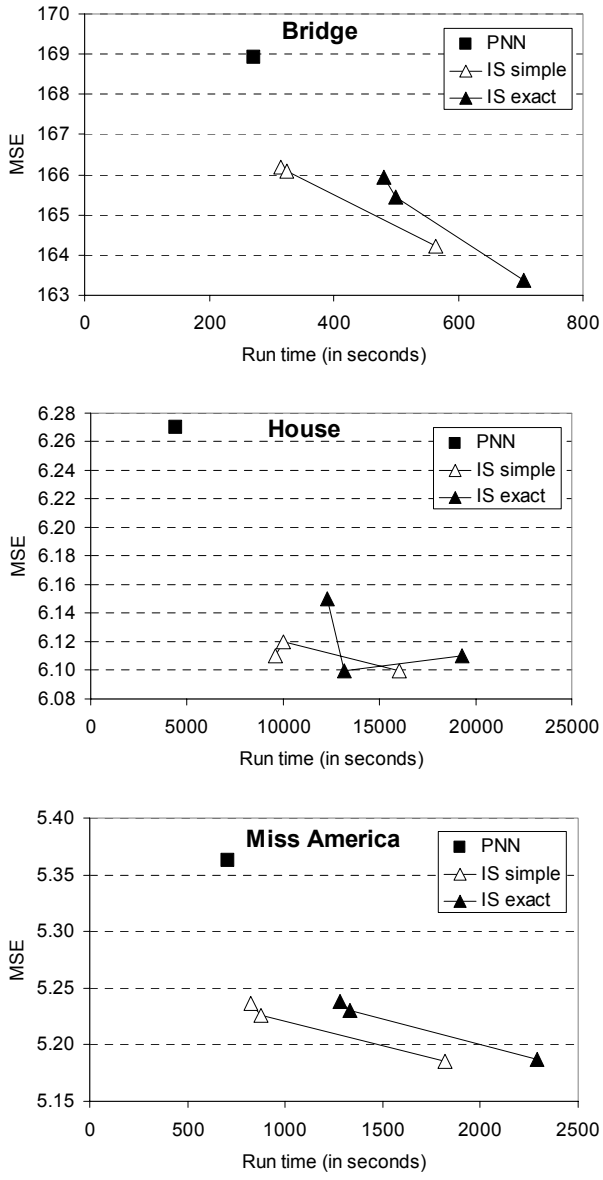


Fig. 11. The MSE-values and running times (in seconds) of the *PNN* and the *IS* methods for *Bridge*, *House* and *Miss America* (*M*=256). The results within a curve are from left to right: the *minimum update*, *standard update* and *extensive update*.

6.2. Running time

We consider next the running time of the *IS* method in more detail. As shown in Section 4, it depends on the size of the data set (N), and on the number of neighbor clusters, which was approximated by the term $\log(N/M)$. We calculated the average number of the neighbor clusters and obtained values 2.0, 2.4, 3.0 for *Bridge*, *House* and *Miss America*. The number is small in the early iterations (mostly 1) because the algorithm removes small clusters. It gradually increases during the process but remains small on average. These numbers support the observation made in Table 4 and Fig. 11 that the choice of the *IS* variant does not have radical effect on the running time. The source of the computation is demonstrated in Table 5 in more detail.

Table 5. Total number of distance calculations made in the case of *Bridge* for the different *IS* variants. The numbers shown are the absolute ($\cdot 10^6$) and relative values (%).

Removal calculation:	Simple calculation			Exact calculation		
Update:	Minimum	Standard	Extensive	Minimum	Standard	Extensive
Initialization phase	16.7 18.1 %	16.7 17.1 %	16.7 9.2 %	16.7 15.6 %	16.7 14.9 %	16.7 8.6 %
Calculation of the removal costs	31.5 33.8 %	31.5 32.1 %	31.5 17.3 %	44.6 41.5 %	44.6 39.6 %	44.4 22.7 %
Update of the secondary partition	44.8 48.1 %	49.7 50.8 %	133.6 73.5 %	46.0 42.9 %	51.4 45.5 %	134.4 68.7 %

6.3. Unknown number of clusters

The *PNN* and *IS* methods were both applied to the *S* data sets using the F-ratio for determining the number of clusters. The results for the *PNN* and *IS* methods are virtually the same with S_1 and S_2 but the *IS* performs better with the sets S_3 and S_4, see Fig. 12. The difference is significant with data set S_4, for which the *IS* method finds the minimum for the number of clusters (M=15) whereas the *PNN* method finds the minimum in the wrong place (M=14).

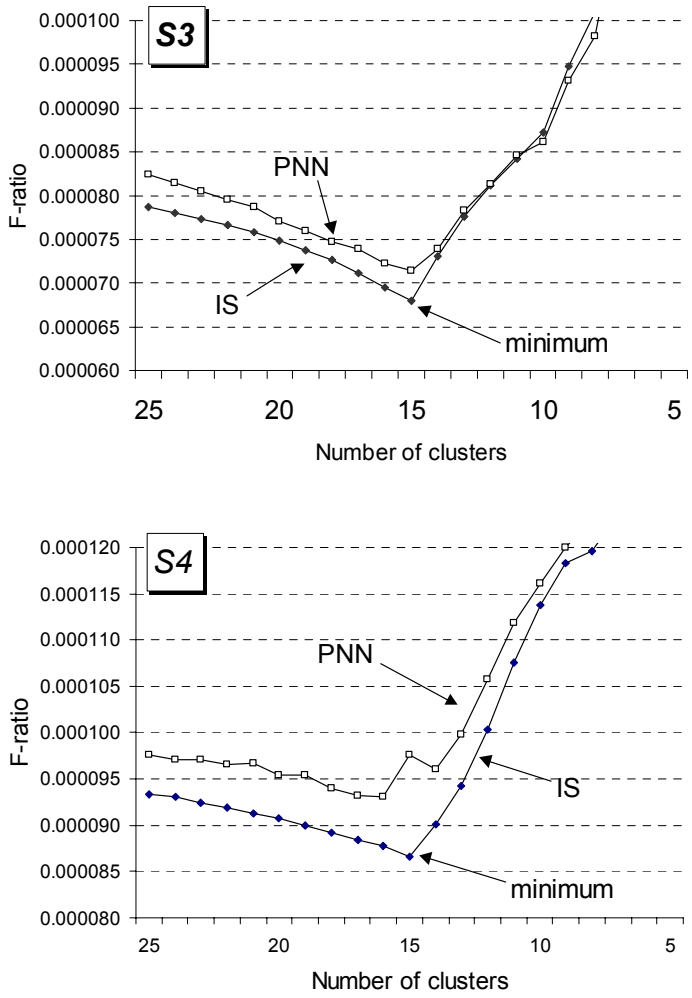


Fig. 12. Comparison of the *PNN* and *IS* methods in the search of the number of clusters.

6.4. Genetic algorithm

We test the *IS* method within the *GA* (denoted as *GAIS*) and the corresponding results are demonstrated in Fig. 13 for *Bridge*. Comparative results are given for the *GA* with PNN crossover – with and without the use of two *k-means* iterations. The first observation is that the *IS* crossover is better than the *PNN*. The experiments also show that improvement can appear during a long time but most remarkable improvement is obtained in the first few iterations. The later improvement is more or less fine-tuning of the solution. In the case of *Bridge*, the first local minimum is reached after 8 iterations with the value of 161.59. The results for the other data sets were similar.

Next we test the *GAIS* method with the F-ratio allowing it dynamically change the number of clusters. The algorithm takes any initial guess for the number of clusters; our random number generator produced M_0=205. With the *S* data sets, the *GAIS* method converges to the correct number of clusters (M=15) in a single iteration. The convergence with the image data sets takes a little bit longer but the number of clusters in the best solution also settles in the first iteration, see Fig. 14. It is also worth noting that the *GAIS* method is actually not much slower than the *IS* method because it starts

process with the initial number of clusters M_0, which is usually much smaller than N, where the *IS* method must start from.

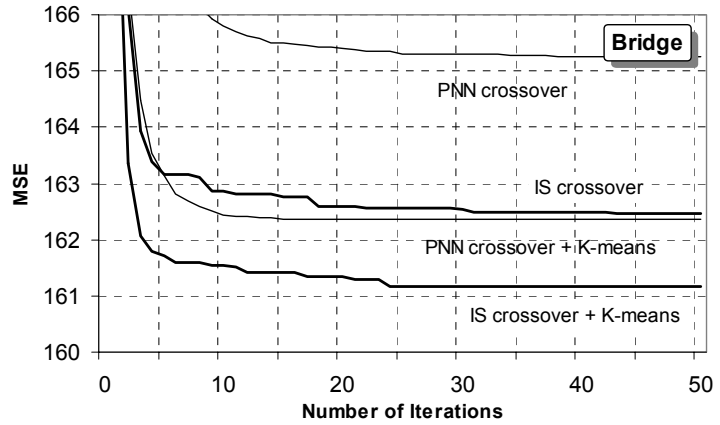


Fig. 13. Performance of the *GA* (Z=10, T=50) with different crossover methods (with and without *k-means* iterations) as a function of the number of iterations.

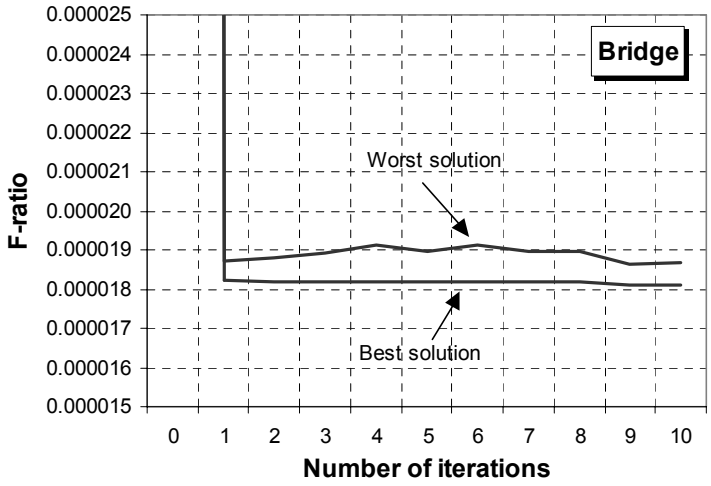


Fig. 14. Convergence of the *GAIS* method using F-ratio for unknown number of clusters. The number of clusters is 205 in the initial population, and then varies from 3 to 4 in the following iterations.

6.5. Comparison

Finally, we compare the performance of proposed methods (*IS* and *GAIS*) to other clustering algorithms in the minimization of the *MSE*. We test the following algorithms:

- Random clustering,
- K-means [25],
- SOM: Self-organizing maps [27],
- FCM: Fuzzy C-means [28],
- Split: Iterative splitting method [29],
- RLS: Randomized local search [30],
- Split-and-merge [9],
- SR: Stochastic relaxation [31],
- PNN: Pairwise nearest neighbor [17],

- GKM: Global k-means [14],
- IS: Iterative shrinking (proposed),
- GA: Genetic algorithm with PNN crossover [10, 11],
- GA: Genetic algorithm with k-means crossover [10],
- GAIS: Genetic algorithm with IS crossover (proposed).
- SAGA: Self-adaptive genetic algorithm [32].

In the comparison, we have included only methods that, according to our experiments, consistently provide high quality partition, and methods that are popular due to their simplicity or for other reasons. The hierarchical approaches are also combined with the *k-means* to get further improvement whereas the other algorithms implicitly include *k-means* iterations in one form or another. The best results of the algorithms are summarized in Table 6. The *Random*, *k-means*, *FCM*, and *SR* have been repeated 10 times. The reported results are the best result found, except random is the average.

Table 6. Performance comparison of the algorithms (for M=256). The results with the S sets have been multiplied by 10^8. The last column gives running times for *Bridge* (in seconds).

	Image sets			Birch data sets			Synthetic data sets				Time
	Bridge	*House*	*Miss America*	B_1	B_2	B_3	S_1	S_2	S_3	S_4	*Bridge*
Random	251.32	12.12	8.34	14.44	35.73	8.20	78.55	72.91	55.42	47.05	<1
K-means (aver.)	179.87	7.81	5.96	5.52	7.99	2.53	20.53	20.91	21.37	16.78	5
K-means (best)	176.95	7.35	5.93	5.13	6.87	2.16	13.23	16.07	18.96	15.71	50
SOM	173.63	7.59	5.92	13.50	10.03	15.18	20.11	13.28	21.10	15.71	376
FCM	178.39	7.79	6.22	5.02	5.29	2.48	8.92	13.28	16.89	15.71	166
Split	170.22	6.18	5.40	4.81	2.29	1.91	8.95	13.33	17.50	16.01	13
Split + k-means	165.77	6.06	5.28	4.64	2.28	1.91	8.92	13.28	16.92	15.77	17
RLS	164.64	5.96	5.28	4.64	2.28	1.86	8.92	13.28	16.89	15.71	1146
Split-n-Merge	163.81	5.98	5.19	4.64	2.28	1.93	8.92	13.28	16.91	15.75	85
SR (average)	162.45	6.02	5.27	4.84	3.39	1.99	9.52	13.68	17.31	15.80	213
SR (best)	161.96	5.98	5.25	4.76	3.12	1.98	8.93	13.28	16.89	15.71	2130
PNN	168.92	6.27	5.36	4.73	2.28	1.96	8.93	13.44	17.70	17.52	272
PNN + k-means	165.04	6.07	5.24	4.64	2.28	1.88	8.92	13.28	16.89	16.87	285
GKM – fast 10	164.12	5.94	5.34	4.64	2.28	1.92	8.92	13.28	16.89	15.71	91721
IS	163.38	6.09	5.19	4.70	2.28	1.89	8.92	13.29	16.96	15.79	717
IS + k-means	162.38	6.02	5.17	4.64	2.28	1.86	8.92	13.28	16.89	15.71	719
GA (k-means)	174.91	6.61	5.54	6.58	5.96	2.45	11.66	15.99	19.22	16.14	654
GA (PNN)	162.37	5.92	5.17	4.98	2.28	1.98	8.92	13.28	16.89	15.71	404
SAGA	161.22	5.86	5.10	4.64	2.28	1.86	8.92	13.28	16.89	15.71	74554
GAIS (short)	161.59	5.92	5.11	4.64	2.28	1.86	8.92	13.28	16.89	15.72	1311
GAIS (long)	160.73	5.89	5.07	4.64	2.28	1.86	8.92	13.28	16.89	15.71	387533

The *K-means*, *SOM* and *FCM* are well known and popular due to their simple implementation. Despite of this, the *k-means* is sensitive to the initialization and the *SOM* is very sensitive to a proper parameter setup. Even a slightest change in the parameter setup can provide noticeable improvement with one data set but turn out to give significant weaker result with another set. With the chosen parameter setup [33] *SOM* finds the best solution with S_2 and S_4 but with significantly weaker results for S_1

and S_3, and for the Birch data sets. The *FCM* finds the best solutions for S_1-S_4 but does perform worse with the image data sets. The *k-means* is implemented as in [34].

The *Split* method [29] always selects the optimal hyper plane dividing the particular cluster along its principal axis, augmented with a local repartitioning phase at each division step. This chosen *Split* variant is optimized for quality rather than speed. Faster *Split* variants also exists but, depending on the variant, the results vary somewhere between *k-means* and *Random*.

The *RLS*, *Split-and-Merge*, and *SR* are all competitive in terms of quality. The *RLS* is the most attractive because of its easy adaptation between speed and quality, even though *Split-and-Merge* sometimes gives slightly better results but with a significantly more complex implementation. The *RLS* and *SR* are both relatively simple to implement but the *SR* is more sensitive to the initialization: it works well for the image data sets but fails to find good partition in about 10-20% of times with the easier *S* data sets.

Among the hierarchical variants, the *PNN* method works rather well in most cases but sometimes (*S3* and *S4*) the results are clearly inferior to that of the *IS* method. The combination with the *k-means* makes sense because the *PNN* and *IS* methods do not do local fine-tuning of the clusters during the process except the partition update operations in the *IS* method. In particular, the *IS* + *k-means* outperforms the other variants except the *genetic algorithms*.

The results of the *GKM* are obtained using the faster $O(gNM^3)$ algorithm with g=10, and by using intermediate codebook of size $2 \cdot M$ to reduce the number of candidate vectors considered at each step of the algorithm. This provides competitive results but with much slower running time. The algorithm can be useful when the number of code vectors M is small.

The proposed genetic algorithm (*GAIS*) gives significantly better than using *k-means* as the crossover method, and slightly better results than the *GA* with *PNN* crossover. It reaches the lowest MSE with only one exception (*House*), thus, effectively matching or even outperforming the previously best known clustering algorithm *SAGA*. The result of the *GAIS* method is also consistent on the initialization as shown in Fig. 15.

The negative side of the genetic algorithm is its slow running time, and the long variant can take several days for the largest data sets. However, much faster convergence can be reached by tuning the parameters of the *GAIS short* as follows. We use the *IS* algorithm with the simple removal calculation and standard update. The *GAIS* method starts with a small population Z=2, which is then increased by one up to Z=100 after every generation. Two *k-means* iterations are applied (g=2). In this way, good solutions are reached much faster but the method is still able to improve in the long run.

Time-distortion performance of the tuned *GAIS* algorithm is compared in Fig. 16 with that of the *k-means* (repeated from new random solutions), *RLS*, and *SAGA*. The *GAIS* method outperforms both the repeated *k-means* and *RLS* when more than 10 seconds is spent in the optimization, and converges approximately to the same result as *SAGA*. The method is inferior to *RLS* and *k-means* only when 10 seconds or less is used for generating the solution.

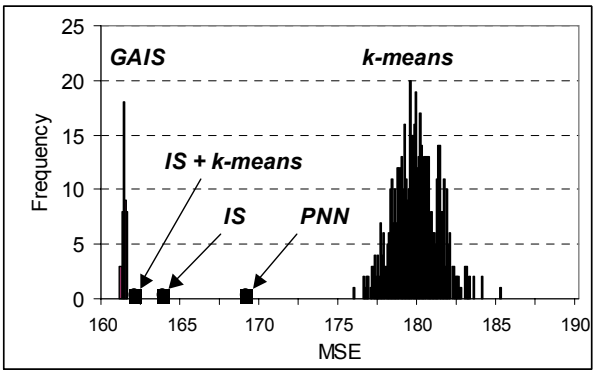


Fig. 15. Histograms of the MSE-values of 50 runs of the *GAIS* method, and 500 runs of the *k-means*. The corresponding standard deviations are $\sigma = 0.11$ (*GAIS*) and $\sigma = 1.41$ (*k-means*).

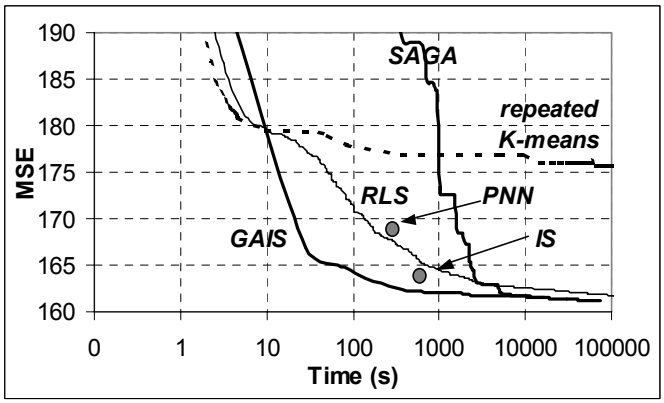


Fig. 16. Time-distortion performance of the selected algorithms.

7. Conclusions

We have proposed the *iterative shrinking* (*IS*) method for the clustering problem. The method generates the clustering hierarchically by removing one cluster at a time. At each step of the algorithm, the cluster to be removed is selected optimally. The merge-based clustering agglomerative can be considered as a special case of the proposed approach. Experimental results show that the method achieves better results than the comparative methods at the cost of slower speed. The time complexity of the method varies from $O(N^2)$ to $O(N^2 \cdot \log^2 N)$ depending on the variant.

The proposed method can also be applied as a crossover method in the *genetic algorithm* (*GAIS*). According to experiments, the genetic combination outperforms all comparative algorithms in terms of minimizing the distortion. Iterative shrinking method extends also to the case where the number of clusters must also be determined simply by changing the optimization function. This does not add to the time complexity as the solutions for a variable number of clusters can be found during a single run of the algorithm.

To sum up, the proposed clustering method (*GAIS*) is capable of providing the best results in minimizing intra cluster variance with competitive time-distortion performance.

Acknowledgements

This work is dedicated to the memory of our colleague Timo Kaukoranta, with whom we had pleasure to work for many years before he suddenly passed away in 2002. His analytical approach and the good sense of humor have inspired us to study the clustering algorithms beyond expectations.

References

1. B.S. Everitt, *Cluster Analysis* (3rd edition), Edward Arnold / Halsted Press, London, 1992.
2. L. Kaufman, P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley Sons, New York, 1990.
3. A.K. Jain, R.C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, Englewood Cliffs, NJ, 1988.
4. A. Gersho, R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Dordrecht, 1992.
5. J.H. Ward, Hierarchical grouping to optimize an objective function, *J. Amer. Statist.Assoc.* 58 (1963) 236-244.
6. W.H. Equitz, A new vector quantization clustering algorithm, *IEEE Trans. on Acoustics, Speech, and Signal Processing* 37 (10) (1989) 1568-1575.
7. O. Virmajoki, P. Fränti, T. Kaukoranta, Practical methods for speeding-up the pairwise nearest neighbor method, *Optical Engineering* 40 (11) (2001) 2495-2504.
8. D.P. de Garrido, W.A. Pearlman, W.A. Finamore, A clustering algorithm for entropy-constrained vector quantizer design with applications in coding image pyramids, *IEEE Trans. on Circuits and Systems for Video Technology* 5 (2) (1995) 83-95.
9. T. Kaukoranta, P. Fränti, O. Nevalainen, Iterative split-and-merge algorithm for VQ codebook generation, *Optical Engineering* 37 (10) (1998) 2726-2732.
10. P. Fränti, J. Kivijärvi, T. Kaukoranta, O. Nevalainen, Genetic algorithms for large scale clustering problem, *The Computer Journal* 40 (9) (1997) 547-554.
11. P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, *Pattern Recognition Letters* 21 (1) (2000) 61-68.
12. O. Virmajoki, P. Fränti, T. Kaukoranta, Iterative shrinking method for generating clustering, *IEEE Int. Conf. on Image Processing (ICIP'02)*, Rochester, New York, USA, vol. 2, 2002, pp. 685-688.
13. P. Fränti, O. Virmajoki, Genetic algorithm using iterative shrinking for solving clustering problems, *Proc. Data Mining Conf. 2003*, Rio de Janeiro, Brazil, 2003, pp. 193-204.
14. A. Likas, N. Vlassis, J.J. Verbeek, The global k-means clustering algorithm, *Pattern Recognition* 36 (2003) 451-461.
15. J. Shanbehzadeh, P.O. Ogunbona, On the computational complexity of the LBG and PNN algorithms, *IEEE Trans. on Image Processing* 6 (4) (1997) 614-616.
16. T. Kurita, An efficient agglomerative clustering algorithm using a heap, *Pattern Recognition* 24 (3) (1991) 205-209.
17. P. Fränti, T. Kaukoranta, D.-F. Shen, K.-S. Chang, Fast and memory efficient implementation of the exact PNN, *IEEE Trans. on Image Processing* 9 (5) (2000) 773-777.

18. T. Kaukoranta, P. Fränti, O. Nevalainen, Vector quantization by lazy pairwise nearest neighbor method, *Optical Engineering* 38 (11) (1999) 1862-1868.
19. J.C. Bezdek, N.R. Pal, Some new indexes of cluster validity, *IEEE Trans. on Systems, Man and Cybernetics* 28 (3) (1998) 302-315.
20. D.L. Davies, D.W. Bouldin,. A cluster separation measure, *IEEE Trans. on Pattern Analysis and Machine Intelligence* 1 (2) (1979) 224-227.
21. M. Sarkar, B. Yegnanarayana, D. Khemani, A clustering algorithm using an evolutionary programming-based approach, *Pattern Recognition Letters* 18 (10) (1997) 975-986.
22. I. Kärkkäinen, P. Fränti, Stepwise algorithm for finding unknown number of clusters, *Advanced Concepts for Intelligent Vision Systems (ACIVS'2002)*, Gent, Belgium, 2002, pp. 136-143.
23. P.K. Ito, Robustness of ANOVA and MANOVA Test Procedures, in: P.R. Krishnaiah (ed), *Handbook of Statistics 1: Analysis of Variance*, North-Holland Publishing Company, 1980, pp. 199-236.
24. M. Xu, Delta-MSE dissimilarity in GLA-based vector quantization, *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, (ICASSP'04),* Montreal, Canada, May 2004.
25. Y. Linde, A. Buzo, R.M. Gray, An algorithm for vector quantizer design, *IEEE Trans. on Communications* 28 (1) (1980) 84-95.
26. T. Zhang, R. Ramakrishnan, M. Livny, BIRCH: A new data clustering algorithm and its applications, *Data Mining and Knowledge Discovery* 1 (2) (1997) 141-182.
27. T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, New York, 1988.
28. J.C. Dunn, A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters, *Journal of Cybernetics* 3 (3) (1974) 32-57.
29. P. Fränti, T. Kaukoranta, O. Nevalainen, On the splitting method for vector quantization codebook generation, *Optical Engineering* 36 (11) (1997) 3043-3051.
30. P. Fränti, J. Kivijärvi, Randomized local search algorithm for the clustering problem, *Pattern Analysis and Applications* 3 (4) (2000) 358-369.
31. K. Zeger, A. Gersho, Stochastic relaxation algorithm for improved vector quantiser design, *Electronics Letters* 25 (1989) 896-898.
32. J. Kivijärvi, P. Fränti, O. Nevalainen, Self-adaptive genetic algorithm for clustering, *Journal of Heuristics* 9 (2) (2003) 113-129.
33. P. Fränti, On the usefulness of self-organizing maps for the clustering problem in vector quantization, *11th Scandinavian Conf. on Image Analysis* (*SCIA'99*), Kangerlussuaq, Greenland, vol. 1, 1999, pp. 415-422.
34. T. Kaukoranta, P. Fränti, O. Nevalainen, A fast exact GLA based on code vector activity detection, *IEEE Trans. on Image Processing* 9 (8) (2000) 1337-1342.