Iterative shrinking method for the clustering problem

Olli Virmajoki[1], Pasi Fränti[1] and Timo Kaukoranta[2]

[1] *Department of Computer Science*
University of Joensuu
P.O. Box 111, FIN-80101 Joensuu
FINLAND

[2] *Turku Centre for Computer Science (TUCS)*
Dept. of Computer Science, University of Turku
Lemminkäisenkatu 14A, FIN-20520 Turku
FINLAND

Abstract: The pairwise nearest neighbor method (PNN) generates the clustering of a given data set by a sequence of merge steps. In this paper, we propose an alternative solution for the merge-based approach by introducing an iterative shrinking method. The new method removes the clusters iteratively one by one until the desired number of clusters is reached. Instead of merging two nearby clusters, we remove one cluster by reassigning its data vectors to the neighbor clusters. We retain the local optimization strategy of the PNN by always removing the cluster that increases the cost function least. We give six alternative implementations, which all outperform the PNN in quality.

Keywords: Vector quantization, codebook generation, clustering, PNN.

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 aims at partition a given set of N data vectors into M groups so that similar objects are grouped together and dissimilar objects to different groups. Clustering is especially needed for designing a *codebook* in vector quantization [4].

Agglomerative clustering is one approach for generating the clustering 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 interesting method for the clustering task because of its conceptual simplicity and good results [7]. The *PNN* can also be combined with the *Generalized Lloyd algorithm* as proposed in [8], or used as a component in more sophisticated methods. For example, the *PNN* has been used in the merge phase in the *split-and-merge* algorithm [9] resulting in to 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 quality of the codebook [11].

The main restriction of the *PNN* is that the clusters are always merged as 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 process when merging smaller

clusters but it can deteriorate the clustering performance at the later stages when merging larger clusters. This can be observed as a weaker performance of the *PNN* when the final number of clusters is very small.

In this paper, we broaden the viewpoint of the *PNN* and propose more general approach for the merge step. Basically, the merge step removes two existing clusters and adds one new. The effective result is that the number of the clusters is reduced by one. The same result can be achieved more directly by removing one cluster and repartition the vectors of the removed cluster to its neighbor clusters. Based on this idea, we propose a new *iterative shrinking* method that generates the clustering by a sequence of cluster removal operations.

The iterative shrinking (*IS*) starts by initializing each data vector as its own cluster. It then removes one cluster at a time until the desired number of clusters has been reached. The vectors of the removed cluster are reassigned to the remaining nearby clusters. The *PNN* can be considered as a special case of the *IS*; we remove one cluster but force the vectors to move to the same neighbor cluster, see Fig. 1. In the *IS*, the vectors can be reassigned more freely, see Fig. 2. Apart from the difference in the removal operation, we follow the local optimization strategy of the *PNN* and always remove the cluster that increases the cost function value least.

The rest of the paper is organized as follows. In Section 2, we give formal definition of the clustering problem and recall the *PNN* method. The iterative shrinking method is then introduced in Section 3. We first present the definition of the secondary partition in Section 3.1. Straightforward solution to the local optimization is given in Section 3.2 and the 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. Experimental results are reported in Section 5, and conclusions drawn in Section 6.

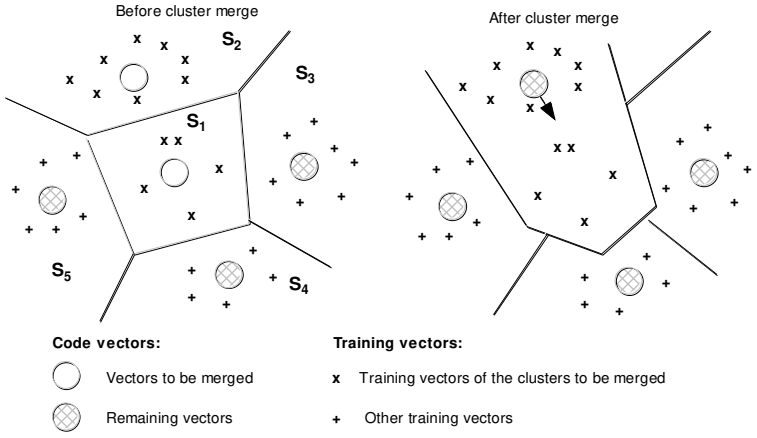

Fig. 1. The merging process of the *PNN*.

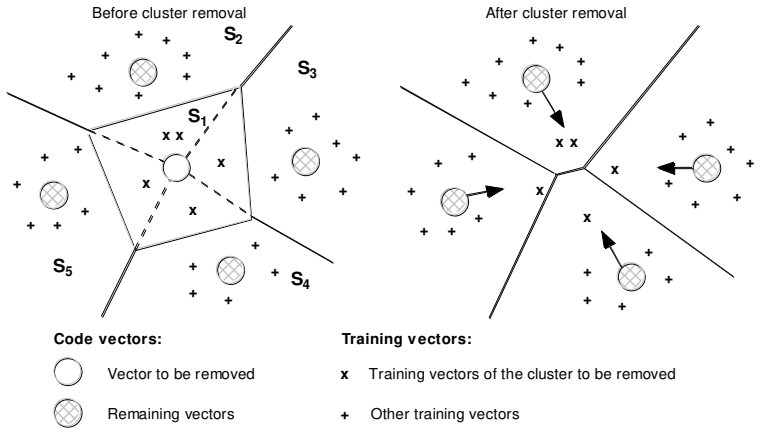

Fig. 2. The cluster removal process of the iterative shrinking.

2. Pairwise nearest neighbor

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 similar objects are grouped together and dissimilar objects to different groups. Partition $P=\{p_1, p_2, \ldots, p_N\}$ defines the clustering by giving for each data vector the index of the cluster where it is assigned to. *A 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. In this paper, the code vectors are always cluster centroids.

The most important choice in the clustering is the cost function f for evaluating the goodness of the clustering. When the data vectors belong to the 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:

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

The *pairwise nearest neighbor* (*PNN*) method [5, 6] generates the clustering hierarchically using 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:

$$s_a \leftarrow s_a \cup s_b \tag{3}$$

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 [6]:

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

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

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

where m is the current number of clusters. There exist many variants of the *PNN* method. 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 [12]. Kurita's implementation maintains an $N\times N$ matrix but it also utilizes a heap structure for searching the minimum distance. The method thus runs in $O(N^2 \log N)$ 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 is obtained by maintaining a pointer from each cluster to its nearest neighbor, and the corresponding merge cost value [13]. 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 [14], 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 still bounded by the fundamental restriction caused by the merge step. Thus, the only way to improve this part, 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

The *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 among 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 will 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 information of the *secondary partition* $Q=\{q_1, q_2, \ldots, q_N\}$ for every data vector. It can be solved in a similar manner to the primary partition but excluding the current nearest 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 account of 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 to 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* 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 to 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 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 vector. Instead, it tends to over estimate the cost function when there are more vectors 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 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:

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

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 results to the local optimization strategy as desired. The situation, however, is not as simple as this because the optimality is restricted by the choice of secondary partition Q. The 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 to 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 solved independently from each other.

3.4. Clustering 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 in respect to the removed cluster:

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

A cluster s_j is defined to be *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:

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

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 limit the amount updates as small as possible. We consider three alternative update strategies:

- *Minimum update*,
- *Standard update*, and
- *Maximum 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 are increasing by the level of update as follows: $G_{\text{minimum}} \subseteq G_{\text{standard}} \subseteq G_{\text{extensive}} \subseteq X$. The inclusion of the vectors in these three sets is summarized in Table 1 according to the type of the clusters of the primary and secondary partition of the data vector. The situation is also illustrated in Fig. 5.

The *minimum update* strategy processes only the vectors that are absolutely necessary to be updated. This includes all vectors in the removed clusters, and also some vectors in the neighbor clusters:

$$G_{\text{minimum}} = \left\{ x_i \middle| p_i = a \vee q_i = a \right\} \tag{14}$$

Firstly, 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, vector in a 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:

$$G_{\text{standard}} = G_{\text{minimum}} \cup \left\{ x_i \mid s_{p_i} \in Y \ \wedge \ s_{q_i} \in Y \right\} \tag{15}$$

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.

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:

$$G_{\text{extensive}} = G_{\text{standard}} \cup \left\{ x_i \mid s_{p_i} \in Y \ \vee \ s_{q_i} \in Y \right\} \tag{16}$$

This update strategy covers practically all vectors that should be updated, and that 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 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	-	Minimum update	Minimum update
	Neighbor	Minimum update	Standard update	Extensive update
	Other	Minimum update	Extensive update	-

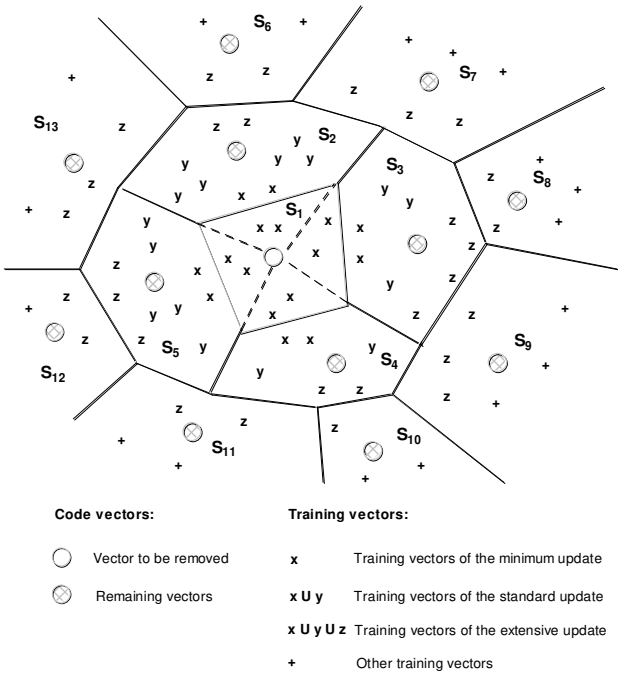

Fig. 5. Illustration of the vectors whose secondary partition is updated in the different update strategies. The removed cluster is s_1, and neighbor clusters are s_2, s_3, s_4 and s_5.

3.5. IS versus PNN

The *PNN* is a special case of the *IS*, as it can be emulated using the *IS* 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, 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 to two clusters. After the first three merges, the output of the PNN and IS procedures are equivalent. However, the fourth merge of the PNN is already restricted by the previous merges and the result is suboptimal. The IS does not have any problems and it will always end up with the same optimal result no matter what was the order of the previous cluster removals.

It is noted that the IS does not guarantee better result than the *PNN* in all occasions. Counter examples exist, in which a single merge step can give better result than the corresponding cluster removal. The reason is that the rule of Eq. (7) does not guarantee locally optimal repartition of the cluster but only approximates it. It is therefore possible, that a sequence of the *PNN* merges produces a better solution than a sequence of *IS* cluster removals for some data sets. Moreover, it is also known that a sequence of locally optimal steps does not necessarily lead to the global optimum.

To sum up, it is possible that the PNN sometimes gives a better clustering result than the IS. But as the definition of the removal step of the IS is more sound than the merge step of the PNN, it is expected that the IS gives better clustering in most cases.

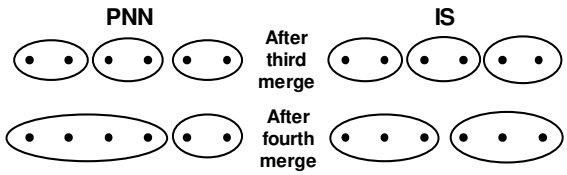

Fig. 6. Example of the different functions of the PNN and the IS methods.

4. Complexity analysis

More detailed pseudo code of the iterative shrinking algorithm is shown in Fig. 7. The initialization phase requires $O(N^2)$ distance calculations due to the construction of the secondary partition. 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 little bit more than that but still no more than $O(N)$. Next, we focus more detailed on the update of the secondary partition.

```
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, , 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 can be shown to be (proof omitted):

$$N\cdot\left(\frac{1}{1}+\frac{1}{2}+\ldots+\frac{1}{N}\right)\le O\left(N\cdot\log N\right) \tag{18}$$

The algorithm actually iterates only N-M steps, so a more accurate upper bound is $N\cdot(\log N - \log M)$. Note that the base of the logarithms in this and the following equations is 2. Thus, the average number of vectors in the cluster is approximated by:

$$\frac{N\cdot\left(\log N-\log M\right)}{N-M}\le O\left(\log N\right) \tag{19}$$

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

$$N\frac{N}{N}+(N-1)\frac{N}{N-1}+\ldots+M\frac{N}{M}=(N-M)\cdot N \tag{20}$$

We assume that the same result applies both for the primary partition and to the secondary partition. 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 straightforward result from this is:

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

Thus, the number of distance calculations in the standard update group 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 simple estimation for the number of affected clusters as $|Y|^2$. The total number of distance calculations can therefore be 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 with *simple update* is theoretically faster than the PNN but probably not so in practice. This is because the number of updates (τ) in the PNN is relatively small, and the removal step in the IS 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.

	Original PNN:	Fast PNN:	Iterative shrinking		
			Simple	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	$O(1)$	$O(\tau N)$	$O(N)$	$O(N\log N/M)$	$O(N\log^2 N)$
Algorithm in total:	$O(N^3)$	$O(\tau N^2)$	$O(N^2)$	$O(N^2\log N/M)$	$O(N^2\log^2 N)$

5. Experiments

We generated training sets from three images: *Bridge*, *Miss America* and *House*, see Fig. 8. The vectors in the set (*Bridge*) are 4x4 pixel blocks from the gray-scale image. The second set (*Miss America*) has been obtained by subtracting two subsequent image frames of the original video image sequences, and constructing 4x4 spatial pixel blocks from the residuals. Only the first two frames have been used. The third data set (*House*) consists of color values of the *RGB* image. Applications of this kind of data sets is found

in image and video image coding (*Bridge*, *Miss America*), and in color image quantization (*House*).

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. 8. Source of the training sets. *Duplicate training vectors are combined and frequency information is stored.

Test clustering results of the three data sets are summarized in Table 3 for all the variants of the *IS* considered here, and for the fast exact *PNN* as implemented in [13]. In all cases, the *IS* produces lower MSE-values but at the cost of about 2 to 6 times longer run times. The time-distortion performance of the variants is also illustrated in Fig.10.

The extension in the amount of updates of the secondary partition decreases the MSE but also slows down the process. The run time of the standard update is only about 4 to 9% longer than that of the minimum update whereas the extensive update increases the run 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 run 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.10. On the other hand, if the MSE is the primary concern, we should use the exact calculation with extensive update. Thus, we will fix this parameter setup in the following tests.

Table 3. The MSE values and run times (in seconds) of the PNN and IS variants for the three training sets (M=256).

		Bridge		House		Miss America	
		Run time	MSE	Run time	MSE	Run time	MSE
PNN		79	168.92	1506	6.27	228	5.36
Simple IS	Minimum update	163	166.18	4842	6.12	456	5.24
	Standard update	171	166.08	5096	6.12	496	5.23
	Extensive update	298	164.22	8384	6.09	1094	5.19
Exact IS	Minimum update	232	166.12	5869	6.13	642	5.24
	Standard update	241	165.44	6117	6.11	675	5.23
	Extensive update	366	163.38	9396	6.09	1268	5.19

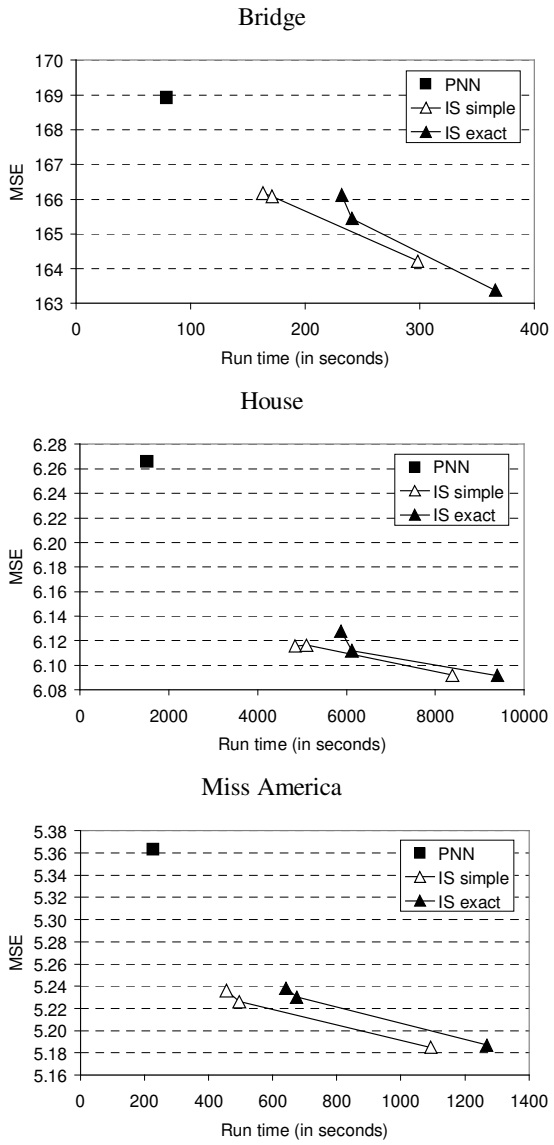

Fig. 10. The MSE-values and run times (in seconds) of the PNN and the IS for *Bridge*, *House*, and *Miss America* (M=256). The results within a curve are from left to right: the *simple update*, *standard update* and *extensive update*.

We consider next the run time of the IS in more detail. As shown in Section 4, the run time 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 the values 2.0, 2.4, 3.0 in the case of *Bridge*, *House*, *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 keeps very small on average. These numbers supports the observation already made from Table 3 and Fig. 10: the choice of the IS variant does not have a considerable effect on the run time. The source of the computation is demonstrated in Table 4 in more detail.

Table 4. 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 the 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 %

The run times of the IS and PNN methods are compared in Fig. 11 by varying the number of clusters (M). Both the IS and PNN spend most of the time at the early stage of iteration. The corresponding MSE-values are summarized in Table 5 for five different algorithms: *GLA* is the well-known iterative algorithm [15]. The PNN and the IS are the same than previous tests. The PNN+GLA and the IS+GLA are the results when the output of the PNN and IS have been fine-tuned by the GLA.

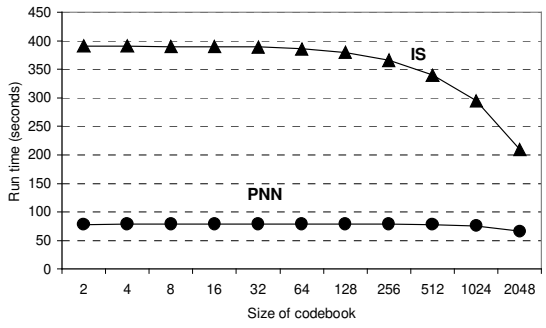

Fig. 11. The run times of the Exact IS with *extensive update*, and the PNN as a function of codebook size (*Bridge*).

Table 5. Comparison of different clustering methods (for *Bridge*) when varying the number of clusters from M=64 to 2048.

M:	GLA	PNN	PNN+GLA	IS	IS+GLA
64	262.17	264.91	257.35	258.36	256.54
128	219.94	214.88	209.02	209.42	207.67
256	180.13	168.92	165.04	163.38	162.38
512	140.29	121.68	120.04	117.41	117.03
1024	98.61	72.61	72.03	70.52	70.45
2048	50.77	27.06	27.00	26.47	26.47

6. Conclusions

We have proposed a new iterative shrinking method for the clustering problem. The method generates the clustering hierarchically by removing one cluster at a time. This is a more general approach than the PNN. The method achieves clearly better results than the GLA, and slightly better than the PNN but at the cost of slower algorithm. The time complexity of the method varies from $O(N^2)$ to $O(N^2 \log^2 N)$ depending on the variant. In practice, the method is about 2 to 6 times slower than the PNN.

References

1. B.S. Everitt, *Cluster Analysis* (3rd edition), Edward Arnold / Halsted Press, London, 1992.
2. L. Kaufman and P.J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley Sons, New York, 1990.
3. R. Dubes and A. Jain, *Algorithms that Cluster Data*, Prentice-Hall, Englewood Cliffs, NJ, 1987.
4. A. Gersho and 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**, 236-244, 1963.
6. W.H. Equitz, "A new vector quantization clustering algorithm", *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37** (10), 1568-1575, October 1989.
7. O. Virmajoki, P. Fränti, T. Kaukoranta, "Practical methods for speeding-up the pairwise nearest neighbor method", *Optical Engineering*, **40** (11), 2495-2504, November 2001.
8. D.P. de Garrido, W.A. Pearlman and W.A. Finamore, "A clustering algorithm for entropy-constrained vector quantizer design with applications in coding image pyramids". *IEEE Transactions on Circuits and Systems for Video Technology* **5** (2), 83-95, April 1995.
9. T. Kaukoranta, P. Fränti and O. Nevalainen, "Iterative split-and-merge algorithm for VQ codebook generation", *Optical Engineering*. **37** (10), 2726-2732, October 1998.
10. P. Fränti, J. Kivijärvi, T. Kaukoranta and O. Nevalainen, "Genetic algorithms for large scale clustering problem", *The Computer Journal*, **40** (9), 547-554, 1997.
11. P. Fränti, "Genetic algorithm with deterministic crossover for vector quantization", *Pattern Recognition Letters*, **21** (1), 61-68, January 2000.
12. J. Shanbehzadeh and P.O. Ogunbona, "On the computational complexity of the LBG and PNN algorithms". *IEEE Transactions on Image Processing* **6** (4), 614-616, April 1997.
13. P. Fränti, T. Kaukoranta, D.-F. Shen and K.-S. Chang, "Fast and memory efficient implementation of the exact PNN", *IEEE Transactions on Image Processing*, **9** (5), 773-777, May 2000.
14. T. Kaukoranta, P. Fränti and O. Nevalainen, "Vector quantization by lazy pairwise nearest neighbor method", *Optical Engineering*, **38** (11), 1862-1868, November 1999.
15. Y. Linde, A. Buzo and R.M. Gray, "An Algorithm for Vector Quantizer Design". *IEEE Transactions on Communications*, **28** (1), 84-95, January 1980.