Practical methods for speeding-up the pairwise nearest neighbor method

Olli Virmajoki
Pasi Fränti
University of Joensuu
Department of Computer Science
P.O. Box 111
FIN-80101 Joensuu, Finland

Timo Kaukoranta
University of Turku
Turku Center for Computer Science
Department of Computer Science
Lemminkäisenkatu 14A
FIN-20520 Turku, Finland

Abstract. The pairwise nearest neighbor (PNN) method is a simple and well-known method for codebook generation in vector quantization. In its exact form, it provides a good-quality codebook but at the cost of high run time. A fast exact algorithm was recently introduced to implement the PNN an order of magnitude faster than the original $O(N^3K)$ time algorithm. The run time, however, is still lower bounded by $O(N^2K)$, and therefore, additional speed-ups may be required in applications where time is an important factor. We consider two practical methods to reduce the amount of work caused by the distance calculations. Through experiments, we show that the run time can be reduced to 10 to 15% that of the original method for data sets in color quantization and in spatial vector quantization.
[DOI: 10.1117/1.1412423]

Subject terms: vector quantization; codebook generation; clustering algorithms; pairwise nearest neighbor method.

Paper 200407 received Oct. 12, 2000; revised manuscript received May 15, 2001; accepted for publication May 16, 2001.

1 Introduction

Vector quantization[1] (VQ) is a method for reducing the amount of data. It can be applied in low-bit-rate compression of image and audio data and in image analysis. The problem of generating a good codebook is one of the greatest problems in the design of a vector quantizer. The aim is to find a set of M code vectors (codebook) for a given set of N training vectors (training set) by minimizing the average pairwise distance between the training vectors and their representative code vectors.

The most cited and widely used method for the codebook generation is the generalized Lloyd algorithm[2] (GLA). It starts with an initial codebook, which is iteratively improved until a local minimum is reached. The algorithm is easy to implement but it makes only local changes to the original codebook. The quality of the final codebook is therefore highly dependent on the initialization.

A better result can be achieved by the pairwise nearest neighbor (PNN) method.[3] This method starts by initializing a codebook of size N, where each training vector is considered as its own code vector. Two code vectors are merged in each step of the algorithm and the process is repeated until the codebook reduces to the desired size M. The PNN method can also be combined[4] with the GLA, or used as a component in more sophisticated methods. For example, the PNN method has been used in the merge phase in the split-and-merge algorithm,[5] resulting in to a good time-distortion performance, and as the crossover method in a genetic algorithm,[6] which has turned out to be the best method among a wide variety of algorithms in terms of the codebook quality.

The main drawback of the PNN method is its slowness, as the original implementation requires $O(N^3K)$ time.[7] An order of magnitude faster algorithm was recently introduced,[8] but the method is still lower bounded by $O(N^2K)$, which is more than the $O(NMK)$ time required by the GLA. Additional improvements are therefore required to make the exact algorithm competitive also in speed.

Several speed-up methods have been introduced in the search of nearest code vector in Euclidean space by reducing the computation required by the distance calculations.[9–12] In the PNN method, the distance calculations are also the bottleneck of the algorithm. It is therefore expected that the ideas proposed for the fast search of the nearest code vector could also be adapted to the PNN method. The main problem is that the distance calculations in the PNN method are not made in Euclidean space. It is therefore not self-evident whether the existing ideas can be transferred to the context of the PNN method.

In this paper, we consider two different speed-up methods found in the literature. The first method is the partial distortion search (PDS) proposed by Bei and Gray.[10] It terminates a single distance calculation immediately when the partial distance exceeds the shortest distance found so far. This idea is independent of the chosen metrics and therefore it can also be directly applied to the PNN method. The second method is the mean-distance-ordered partial search (MPS) technique introduced by Ra and Kim.[12] This technique uses the component means of the vectors to derive a precondition for the distance calculations and, in this way, a large number of the distance calculations can be omitted completely. The idea utilizes properties of Euclidean space but we will show that the precondition can be generalized for the distance calculations in the PNN method.

In general, it is not possible to transfer every speed-up method from the nearest code vector search to the context of the PNN. For example, the triangular inequality elimina-

```
PNN(X, M) → C, P
    s_i ← {x_i} ∀ i∈[1,N];
    m ← N;
    REPEAT
        (s_a, s_b) ← NearestClusters();
        MergeClusters(s_a, s_b);
        m ← m-1;
        UpdateDataStructures();
    UNTIL m=M;
```

Fig. 1 Structure of the exact PNN method.

tion (TIE) technique presented by Chen and Tsieh[11] maintains the (Euclidean) distances between all code vectors, and then reduces the number of distance calculations by a condition derived from the triangle inequality. In principle, we could derive a similar precondition for the PNN cost function. In the PNN method, however, the overhead of maintaining a complete distance matrix equals to the original workload of the PNN and, therefore, no speed-up is possible in this way.

The rest of the paper is organized as follows. Section 2 presents the PNN method and its fast exact implementation. We provide a detailed description of the method to enable the reader to implement the exact PNN method accurately. New speed-up methods are then introduced in Sec. 3. In particular, we introduce the PDS and the MPS methods in the context of the PNN method. Simulation results for various training sets are shown in Sec. 4. Experiments show that the run time can be reduced to 10 to 15% in the case of the two favorable data sets, whereas only moderate improvement can be achieved in the case of the unfavorable data set. Conclusions are drawn in Sec. 5.

2 PNN Method

We consider a set of N training vectors ($\mathbf{x}_i$) in a K-dimensional Euclidean space. The aim is to find a codebook C of M code vectors ($\mathbf{c}_i$) by minimizing the average squared Euclidean distance between the training vectors and their representative code vectors:

$$f(C)=\frac{1}{N}\sum_{i=1}^{N}\|\mathbf{x}_i-\mathbf{c}_{p_i}\|^2, \tag{1}$$

where p_i is the cluster (partition) index of the training vector $\mathbf{x}_i$. Cluster is defined as the set of training vectors that belong to the same partition a:

$$s_a=\{\mathbf{x}_i|p_i=a\}. \tag{2}$$

The basic structure of the PNN method is shown in Fig. 1. The method starts by initializing each training vector $\mathbf{x}_i$ as its own code vector $\mathbf{c}_i$. In each step of the algorithm, the size of the codebook is reduced by merging two clusters. The cost of merging two clusters s_a and s_b can be calculated as[3]

$$d_{a,b}=\frac{n_a n_b}{n_a+n_b}\cdot\|\mathbf{c}_a-\mathbf{c}_b\|^2, \tag{3}$$

where n_a and n_b denote to the sizes of the corresponding clusters a and b. The cost function is symmetric ($d_{a,b}=d_{b,a}$) and it can be calculated in $O(K)$ time, assuming that n_a, n_b, $\mathbf{c}_a$, and $\mathbf{c}_b$ are known.

The exact variant of the PNN method applies local optimization strategy: all possible cluster pairs are considered and the one (a,b) increasing the distortion least is chosen:

$$a,b=\arg\min_{\substack{i,j\in[1,N]\\ i\neq j}} d_{i,j}. \tag{4}$$

The clusters are then merged and the process is repeated until the codebook reaches the size M. Straightforward implementation of this takes $O(N^3K)$ time because there are $O(N)$ steps, and in each step there are $O(N^2)$ cost function values to be calculated.

2.1 *Fast Exact PNN Method*

A much faster variant of the PNN method can be implemented by maintaining for each cluster a pointer to its nearest neighbor.[8] The nearest neighbor nn_a for a cluster s_a is defined as the cluster minimizing the merge cost:

$$nn_a=\arg\min_{\substack{j\in[1,N]\\ j\neq a}} d_{a,j}. \tag{5}$$

In this way, only few nearest neighbor searches are needed in each iteration. The method is denoted as fast exact PNN method and its implementation details are given next.*

For each cluster s_j, we also maintain the cluster size n_j, the corresponding code vector $\mathbf{c}_j$, and the pointer to its nearest neighbor nn_j. The nearest neighbor pointer is assigned with the cost value d_j indicating the amount of increase in distortion if the cluster s_j is merged to s_{nn_j}. For each training vector, we maintain the index of the cluster p_i, to which it belongs.

2.2 *Initialization*

In the initialization, each training vector $\mathbf{x}_i$ is assigned to its own cluster. The corresponding cluster size is set to 1, and the code vector $\mathbf{c}_i$ as the training vector itself:

$$p_i\leftarrow i;\quad n_i\leftarrow 1;\quad \mathbf{c}_i\leftarrow\mathbf{x}_i\quad \forall i\in[1,N]. \tag{6}$$

To generate the nearest neighbor table, we must find the nearest neighbor nn_i for every cluster. This is done by considering all other clusters as tentative neighbor and selecting the one that minimizes Eq. (3). There are $O(N^2)$ pairs to be considered, and thus, the initialization phase takes $O(N^2K)$ time in total.

2.3 *Merging the Clusters*

The optimal cluster pair (s_a and s_b) to be merged is the cluster having the minimum d_j value and its nearest neighbor nn_j:

*Pseudocode of the method is available at: http://cs.joensuu.fi/pages/franti/research/pnn.txt.

$$a \leftarrow \arg \min_{j \in [1,N]} d_j ; \quad b \leftarrow nn_a . \tag{7}$$

This pair can be found in $O(N)$ time using linear search for the nearest neighbor table. The merge of the clusters is then performed as follows. First, we update the partition indices so that the combined cluster replaces s_a, and the cluster s_b becomes obsolete:

$$p_i \leftarrow a \quad \forall i | p_i = b. \tag{8}$$

The size of the merged cluster is calculated as:

$$n_a \leftarrow n_a + n_b . \tag{9}$$

The code vector of the combined cluster could be calculated as the weighted average of $\mathbf{c}_a$ and $\mathbf{c}_b$:

$$\mathbf{c}_a \leftarrow \frac{n_a \mathbf{c}_a + n_b \mathbf{c}_b}{n_a + n_b}. \tag{10}$$

However, as we also maintain the partition index of each training vector p_i, it is therefore better to calculate the new code vector as the centroid of the cluster to minimize rounding errors:

$$\mathbf{c}_a \leftarrow \frac{1}{n_a} \cdot \sum_{p_i = a} \mathbf{x}_i . \tag{11}$$

These steps can be performed at most in $O(NK)$ time.

2.4 *Updating the Nearest Neighbor Pointers*

The nearest neighbor nn_a for the merged cluster (now s_a) must be resolved by calculating the distance function values [Eq. (3)] between the new cluster and all other remaining clusters. This can be performed in $O(NK)$ time.

The nearest neighbor function is not symmetrical, i.e., $nn_a = b$ does not imply $nn_b = a$. Therefore, we must also resolve the nearest pointer for all clusters whose nearest neighbor before the merge was either a or b ($nn_i = a$ or $nn_i = b$). This takes $O(NK)$ time for a single cluster and there are approximately three to five clusters on average to be updated in each step of the algorithm, according to Ref. 8. The overall time complexity of the update is therefore $O(\tau NK)$, where τ denotes the number of clusters whose the nearest neighbor pointer must be resolved. To summarize the time complexity of the fast exact PNN method is $O(\tau N^2 K)$.

2.5 *Lazy PNN Method*

The number of distance calculations can be reduced by delaying the update of the nearest neighbor pointers. This idea is based on the monotony property shown in Ref. 13, which says that the minimum cluster distances never decrease due to the merge of the optimal pair. For example, assume that the nearest neighbor for a cluster s_i was s_a before the merge, and s_c after the merge. From the monotony property we know that $d_{i,a} \leq d_{i,c}$. We therefore do not need the exact distance but the previous distance serves as a lower bound. In practice, we can assume that the nearest neighbor after the merge is s_{a+b}, and use the previous cost function value $d_{i,a}$ (or $d_{i,b}$). The distance value is labeled as "outdated," and it will be updated only if it becomes the candidate for being the smallest distance of all. In this way, we can reduce the computation by about 35% while preserving the exactness of the PNN method.[13]

3 Speed-Up Methods for the PNN Method

There are two alternative approaches for speeding-up the PNN method. One approach is to sacrifice the exactness of the PNN method either by using a faster but suboptimal method for selecting the clusters to be merged,[3] or by generating an initial codebook of size $N > M_0 > M$ before the PNN method.[4] However, we take another approach, in which the exactness of the PNN method is preserved in all steps of the algorithm.

The main loop of the PNN in Fig. 1 reduces the number of code vectors from N to M. It seems to be impossible to reduce the number of stages in this loop without sacrificing the optimality of the steps. We therefore aim at reducing the computation inside the loop. The loop consists of the search of the cluster pair, the merge, and the update of the nearest neighbor pointers. The search takes $O(N)$, and the merge $O(NK)$ time. In the update phase, we must find nearest neighbor for τ clusters, and every search requires $O(N)$ distance calculations. Thus, the update phase requires $O(\tau NK)$ in total, and it is clearly the bottleneck of the algorithm.

We consider the following two methods:

1. the PDS technique[10]
2. the MPS technique[12]

These methods are tailored for gaining speed without sacrificing optimality. They are new in the context of the PNN method but are widely used in the encoding phase of VQ, and in the search for the nearest code vector in the GLA. In the PNN method, they can be applied both in the initialization stage and in the main loop of the PNN method. These methods can be considered practical as they achieve speed-up without complicated data structures and without excessive increase of the memory consumption, and they are easy to implement.

3.1 *PDS*

Let s_a be the cluster for which we seek the nearest neighbor. We use full search, i.e., calculate the distance values $d_{a,j}$ between s_a and all other clusters s_j. Let $d_{\min}$ be the distance of the best candidate found so far. The distance is calculated cumulatively by summing up the squared differences in each dimension. In PDS, we utilize the fact that the cumulative summation is nondecreasing, as the individual terms are nonnegative. The calculation is therefore terminated and the candidate rejected if the partial distance value exceeds the current minimum $d_{\min}$.

The implementation of the partial distance calculation is shown in Fig. 2. The distance function of Eq. (3) can be divided into two parts consisting of the squared Euclidean distance ($e_{a,j}$) of the cluster centroids, and a weighting factor ($w_{a,j}$) that depends on the cluster sizes:

```
MergeCost($s_a$, $s_j$, $d_{min}$) $\rightarrow d$;
    $e \leftarrow 0$;
    $k \leftarrow 0$;
    $w \leftarrow n_a \cdot n_j / (n_a + n_j)$;
    REPEAT
        $k \leftarrow k + 1$;
        $e \leftarrow e + (c_{ak} - c_{jk})^2$;
        $d \leftarrow w \cdot e$;
    UNTIL ($d > d_{min}$) OR ($k = K$);
    RETURN $d$;
```

Fig. 2 Pseudocode for the distance calculation in the (simple) PDS method.

$$e_{a,j}=\sum_{k=1}^{K}(c_{ak}-c_{jk})^2, \tag{12}$$

$$w_{a,j}=\frac{n_a\cdot n_j}{n_a+n_j}. \tag{13}$$

Here c_{ak} and c_{jk} refer to the k'th components of the corresponding vectors, and n_a and n_j to the sizes of the particular clusters. The weighting factor $w_{a,j}$ is calculated first, and the squared Euclidean distance $e_{a,j}$ is then cumulated by summing up the squared differences in each dimension. After each summation, we calculate the partial distortion value $(w_{a,j}e_{a,j})$ and compare it to the distance of the best candidate $(d_{\min})$:

$$w_{a,j}e_{a,j}\geqslant d_{\min}. \tag{14}$$

The distance calculation is terminated if this condition is found to be true. The calculations of the partial distortion require an additional multiplication operation and an extra comparison for checking the termination condition. We refer this as the simple variant. Speed-up can be achieved if this extra work does not exceed the time saved by the termination. The extra multiplication can be avoided by formulating the termination condition as:

$$e_{a,j}\geqslant\frac{d_{\min}}{w_{a,j}}. \tag{15}$$

The right part of the equation can now be calculated in the beginning of the function, and only the comparison remains inside the summation loop. We refer this as the optimized variant. As a drawback, there are one extra division due to Eq. (15) and extra multiplication outside the loop.

The computational efficiency of the two variants are compared to that of the full search in Table 1. The simple variant is faster when the dimensions are very small and in cases where the termination happens earlier. Equation (14) is also less vulnerable to rounding errors than Eq. (15). The optimized variant, on the other hand, produce significant improvement when the dimensions are very large.

Overall, the effectiveness of the method depends on the quality of the current candidate. It is therefore important to have a good initial candidate so that the $d_{\min}$ would already be small at the early stages of the calculations. In this way, more distance calculations can be terminated sooner. In the previous iteration, the nearest neighbor for s_a was one of the clusters that were merged. It is expected that the distance to the merged cluster remains relatively small and, therefore, we take this as the first candidate. This minor enhancement turns out to provide significant improvement in the algorithm (see Sec. 4).

Table 1 Summary of the arithmetic operations involved in the distance calculations, where the value $q\in[0,1]$ refers to the proportion of the processed dimensions.

Variant	*	/	+
Full search	$k+2$	1	$2k+1$
Simple variant	$2kq+1$	1	$2kq+1$
Optimized variant	$kq+2$	2	$2kq+1$

3.2 *Mean-Distance Ordered Partial Search*

The mean-distance-ordered partial search[12] applies two different techniques to speed-up the search of the nearest code vector. First, it uses a fast precondition for checking whether the distance calculation to a given candidate cluster can be omitted. Second, it sorts the codebook according to the component means of the code vectors and derives limits for the search.

The method stores the component sums of each code vector. Let s_a be the cluster for which we seek its nearest neighbor, and let s_j be the candidate cluster to be considered. The distance of their corresponding code vectors $\mathbf{c}_a$ and $\mathbf{c}_j$ can be approximated by the squared distance of their component sums:

$$\hat{e}_{a,j}=\left(\sum_{k=1}^{K}c_{ak}-\sum_{k=1}^{K}c_{jk}\right)^2. \tag{16}$$

The component sums correspond the projections of the code vectors to the diagonal axis of the vector space. In typical training sets, the code vectors are highly concentrated along the diagonal axis, and therefore, the distance of their component sums highly correlate to their real distance. The following inequality holds true:[12]

$$\hat{e}_{a,j}\leqslant K\cdot e_{a,j}. \tag{17}$$

This inequality was utilized in the search of nearest neighbor in VQ by deriving the following precondition:

$$K\cdot e_{\min}<\hat{e}_{a,j}. \tag{18}$$

In other words, if the squared Euclidean distance of the component sums exceeds the distance to the best candidate found so far (multiplied by K), the real distance cannot be smaller than $e_{\min}$ according to Eq. (17). This is illustrated in Fig. 3, where the distance from A to B is the current minimum. All potential candidates and their projections must therefore lie inside the circle. The precondition can be calculated fast in an $O(1)$ time as the component sums are

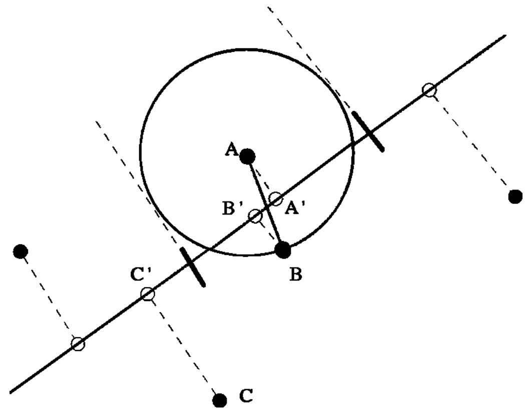

Fig. 3 Code vectors (black dots) and their projections (open dots) according to the component sums.

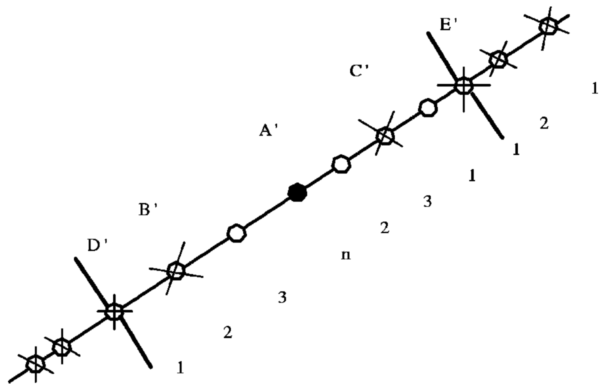

Fig. 4 Example of setting up the search boundaries in the MPS technique. The projections of the clusters are drawn as circles. Cluster *A* (with a black dot) is the one for which we search the nearest neighbor. The numbers represent the cluster sizes, and the crossing indicates that the precondition holds true for the cluster and its distance calculation can be omitted.

already known. If the precondition [inequality (18)] is true, the candidate code vector can be rejected without the distance calculation.

In the PNN method, the distance function consists of the squared Euclidean distance ($e_{a,j}$) of the code vectors and the weighting factor ($w_{a,j}$). As shown in Eqs. (12) and (13), these two can be calculated separately. Inequality (17) can therefore be generalized to the cluster distances as

$$w_{a,j} \cdot \hat{e}_{a,j} \leqslant K \cdot w_{a,j} \cdot e_{a,j} . \tag{19}$$

Given the minimum distance $e_{\min}$, we can then derive a similar precondition for the PNN distance function as:

$$K \cdot d_{\min} < w_{a,j} \cdot \hat{e}_{a,j} . \tag{20}$$

This can be applied as follows. The clusters are processed in any given order. The weighting factor $w_{a,j}$ and the distance of the component sums ($\hat{e}_{a,j}$) are first calculated, and the precondition is evaluated. If it holds true, the calculation of the Euclidean distance can be omitted and the candidate cluster s_j rejected.

Further speed-up can be obtained by sorting the codebook according to their component sums, and then proceed the clusters in the order given by the sorting.[12] The search starts from the cluster s_a and it proceeds bidirectionally along the projection axis. When we find the first cluster for which the precondition is met, we know that all the rest code vectors in that particular direction will meet the precondition of Eq. (18). In the GLA, this gives definite boundaries for the search and the rest of the candidates can be rejected even without calculation of the precondition.

In the PNN method, we cannot make solid bounds for the search because of the weighting factor [Eq. (13)] involved in the distance function. Consider the situation in Fig. 4. The clusters *B* and *C* (with cluster sizes 2 and 3) are the first two that meet the inequality (20). However, they do not necessarily bound the search as there can be smaller clusters further away, for which the distance to *A* is smaller. In the example of Fig. 4, there is one such cluster: the one between *C* and *E*.

The precondition cannot be used as such to provide bounds for the search, but we have found a weaker condition to terminate the search. The precondition guarantees that there are no more potential clusters in the respective direction, whose size is greater than or equal to the cluster that met the precondition. Specifically, if the size of the candidate cluster equals one and inequality (20) holds, we can terminate the search in that direction. For example, in Fig. 4 the search bounds will be the clusters *D* and *E* because they are the first clusters of size 1 in the corresponding directions, and inequality (20) holds for them.

The pseudocode of the algorithm is given in Fig. 5. For simplicity, we assume that the clusters have already been sorted before the call of the routine.

3.3 *Initialization with the Methods*

When special speed-up techniques are not used, the initial nearest neighbors nn_i and the associated distances d_i can be determined by calculating only half of the pairwise distances $d_{i,j}$ because the merge cost function [Eq. (3)] is symmetrical. When we are determining the nearest neighbor for cluster s_a and we have calculated its distance to cluster s_b, we also check if s_a is closer to s_b than s_b's current nearest neighbor nn_b, i.e., $d_{a,b} < d_b$. If that is the case, we update the nearest neighbor of cluster s_b also.

However, this can not be done if the PDS is utilized in the initialization. Because then the real distance between clusters s_a and s_b is not always calculated but only the partial distance, which can be smaller than the real one. Therefore it can not be used for the update of the nearest neighbor of cluster s_b.

Fortunately, it is not necessary that all nearest neighbor pointers have been assigned to really nearest cluster to preserve the exactness of the PNN method. It is enough that for each cluster s_i, we have determined its nearest neighbor among the clusters whose index *j* is greater than *i*. Thus, the nearest neighbor for cluster s_i is

$$nn_i = \arg \min_{i<j \leqslant N} d_{i,j} . \tag{21}$$

This guarantees that nearest neighbor pair of all cluster pairs is stored. The second nearest cluster pair is not nec-

```
SearchNearestNeighborUsingMPS($c_a$, $c_i$, $d_{min}$) → $nn_a$, $d_a$;
    $d_{min} \leftarrow \infty$;
    up ← TRUE;
    down ← TRUE;
    $j_1 \leftarrow a$;
    $j_2 \leftarrow a$;

    WHILE (up OR down) DO

        IF up THEN
            $j_1 \leftarrow j_1 + 1$;
            IF $j_1 > N$ THEN up ← FALSE
            ELSE CheckCandidate($s_a$, $s_{j1}$, $n_a$, $d_{min}$, $nn$, up);

        IF down THEN
            $j_2 \leftarrow j_2 - 1$;
            IF $j_2 < 1$ THEN down ← FALSE
            ELSE CheckCandidate($s_a$, $s_{j2}$, $n_a$, $d_{min}$, $nn$, down);

        END-WHILE;

    RETURN $nn$, $d_{min}$;

CheckCandidate($s_a$, $s_j$, $n_a$, $d_{min}$, $nn$, direction);
    IF PreCondition($s_a$, $s_j$, $d_{min}$) THEN
        IF $n_a = 1$ THEN direction ← FALSE
    ELSE
        $d \leftarrow$ MergeCost($s_a$, $s_j$, $d_{min}$);
        IF $d < d_{min}$ THEN
            $d_{min} \leftarrow d$;
            $nn \leftarrow j$;
    RETURN;

PreCondition($s_a$, $s_j$, $d_{min}$) → BOOLEAN;
    $w \leftarrow n_a \cdot n_j / (n_a + n_j)$;
    $\hat{e} \leftarrow (sum_a - sum_j)^2$;
    RETURN( $K \cdot d_{min} < w \cdot \hat{e}$ );
```

Fig. 5 Pseudocode for the MPS technique used in the PNN method.

essarily stored, but it will not be needed. After the merge of the nearest cluster pair, we update the nearest neighbor pointers for the appropriate clusters by considering all clusters. Therefore, we always have the knowledge on the nearest cluster pair.

4 Test Results

We generated training sets from six images: ''Bridge,'' ''Camera,'' ''Miss America,'' ''Table Tennis,'' ''House,'' and ''Airplane'' (see Fig. 6). The vectors in the first two sets (''Bridge,'' ''Camera'') are 4×4 pixel blocks from the gray-scale images. The third and fourth sets (''Miss America,'' ''Table Tennis'') were obtained by subtracting two subsequent image frames of the original video image sequences, and then constructing 4×4 spatial pixel blocks from the residuals. Only the first two frames were used. The fifth and sixth data sets (''House,'' ''Airplane'') consist of color values of the RGB images. Applications of this kind of data sets is found in image and video image coding (''Bridge,'' ''Camera,'' ''Miss America,'' ''Table Tennis''), and in color image quantization (''House,'' ''Airplane'').

4.1 *Experiments with the PDS Variants*

The effect of the initialization of the PDS is first studied as a function of the vector dimension. For this purpose, we use artificially generated training sets with the following parameters: The number of training vectors is $N=1024$, the number of clusters $M=256$, and the vector size K varies from 16 to 1024. The results are shown in Fig. 7, and they clearly demonstrate the importance of the initial guess for training sets with large vector dimensions. The improvement is less significant for training sets with $K<16$, but it is still large enough to be useful. In the following, we assume that the initial guess is always used.

The performance of the two PDS variants (simple and the optimized variants) are summarized in Fig. 8 for the six training sets of Fig. 6. The results show that the simple variant is better for the training sets (''House'' and ''Airplane'') with small vector dimensions $(K=3)$. This is so because of the extra division operation in the optimized variant. In fact, the optimized variant is even slower than the original PNN. For the other training sets (''Bridge,'' ''Camera,'' ''Miss America,'' ''Table Tennis''), the optimized variants works much better and gives always equal to or better result than the simple variant. The significance of the division operation is much smaller in the case of a vector with a higher dimension.

Note also that the cost of the division operation is hardware dependent. The results indicate that the relative per-

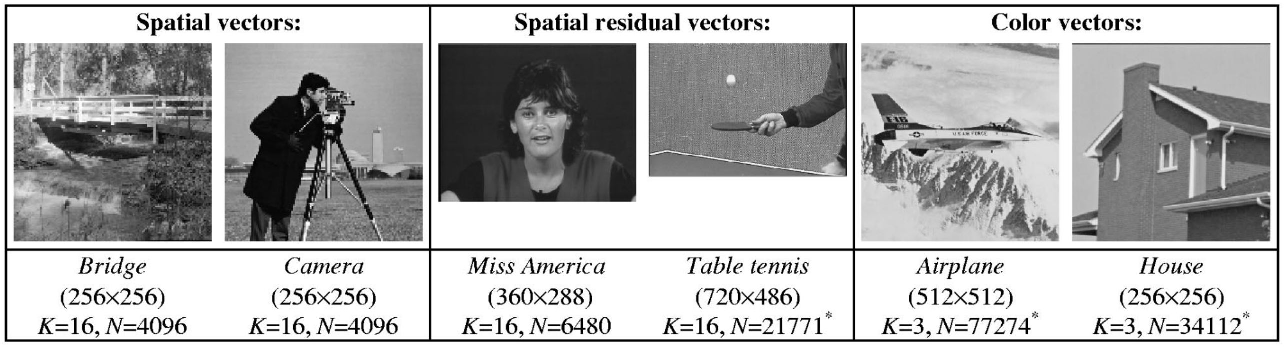

Fig. 6 Sources of the training sets. Here *N* refers to the number of distinctive vectors in the training set, and *K* refers to the dimensionality of the vectors; * indicates that duplicate training vectors are combined and frequency information is stored.

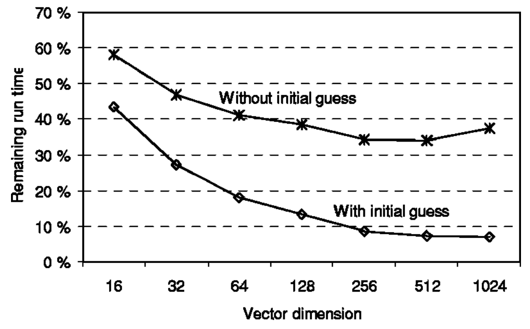

Fig. 7 Remaining run time relative to full search PNN method for the optimized PDS with and without an initial guess.

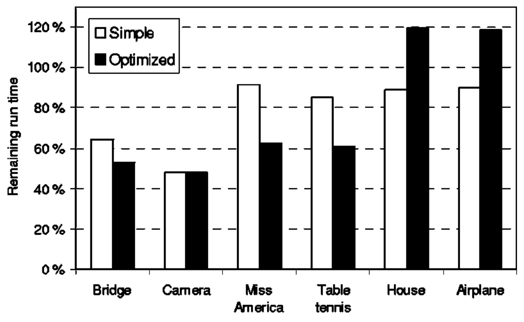

Fig. 8 Remaining run time relative to the full search PNN method for the simple and optimized variants of the PDS.

formance of the optimized variant increases as a function of *K*, but one should not draw any final conclusions about the exact value of *K*, after which the optimized variant outperforms the simple variant.

4.2 *Experiments with the MPS Variants*

The effectiveness of the MPS technique is shown in Table 2 as the number of operations relative to the full search. The first observation is that the use of the precondition is efficient for "Bridge" and "Camera" (run time is decreased to about 30%) but only moderate improvement is achieved with the other four training sets (down to about 70 to 80%). The second observation is that the sorting gives significant improvement for "Bridge," "Camera," "Airplane," and "House." For the color images ("Airplane" and "House"), the sorting is important as it eliminates most of the precondition tests; only about 2% of the tests remains.

Overall, the MPS works very well except for the residual training sets ("Miss America," "Table Tennis"). For these, the MPS can reduce the computation only down to about 2/3 from that of the full search. This originates from the nature of the training sets; the training vectors are relatively evenly distributed in the vector space and there are no clear clusters in the training sets. The projection of the vectors onto the diagonal axis therefore is not as efficient as it would be if the vectors were clustered in the vector space.

4.3 *Joint Experiments*

Table 3 gives the detailed work load of the different PNN variants for three training sets, one from each category. The results show that the joint use of the PDS and MPS techniques is successful. Only in the case of the first category ("Bridge" and "Camera"), the use of the MPS technique somewhat weakens the effect of the PDS method. This is

Table 2 The remaining run times for the MPS variants.

	Relative					
	MPS without Sorting			MPS with Sorting		
	Preconditions/Search (%)	Distance Calculations/Search (%)	Remaining Run Time (%)	Preconditions/Search (%)	Distance Calculations/Search (%)	Remaining Run Time (%)
"Bridge"	100	17.5	32.9	16.6	15.3	16.5
"Camera"	100	9.6	28.8	8.5	7.9	11.0
"Miss America"	100	82.1	72.1	80.5	78.6	66.4
"Table Tennis"	100	85.3	71.4	84.4	83.9	71.3
"Airplane"	100	4.9	77.2	1.7	1.2	18.6
"House"	100	6.9	78.3	2.4	1.8	15.2
	Absolute					
	MPS without Sorting			MPS with Sorting		
	Preconditions/Search	Distance Calculations/Search	Remaining Run Time (%)	Preconditions/Search	Distance Calculations/Search	Remaining Run Time (%)
"Bridge"	2599.1	455.2	32.9	432.6	397.4	16.5
"Camera"	2575.1	247.6	28.8	218.2	202.9	11.0
"Miss America"	3959.2	3252.1	72.1	3187.0	3112.1	66.4
"Table Tennis"	13021.4	11103.4	71.4	10983.7	10922.2	71.3
"Airplane"	48189.5	2373.6	77.2	805.4	560.4	18.6
"House"	21285.1	1464.9	78.3	506.9	373.2	15.2

Table 3 The average number of distance calculations per nearest neighbor search, the average number of processed vector dimension during the distance calculation, and the total number of processed vector dimensions per search on average.

	Distance Calculations/Search	Dimensions/Distance Calculation	Dimensions/Search
"Bridge," $N=4096$, $M=256$, $K=16$			
Full	2208.6	16.0	35338.3
PDS	2208.7	3.0	6534.2
MPS+PDS	397.4	5.7	2261.0
"House," $N=34112$, $M=256$, $K=3$			
Full	16514.8	3.0	49544.5
PDS	16533.6	1.1	17538.3
MPS+PDS	373.2	1.2	434.7
"Miss America," $N=6480$, $M=256$, $K=16$			
Full	3404.8	16.0	54476.9
PDS	3401.5	4.8	16194.3
MPS+PDS	3112.1	4.8	14805.4

because with the MPS, most of the distance calculations are done to nearby clusters and therefore, the PDS is less efficient. Nevertheless, the overall improvement is still very good for these training sets. For the other four training sets, the use of the MPS do not decrease the effectiveness of the PDS.

The overall run times of the PDS and MPS are summarized in Table 4. The methods are also combined with the lazy PNN of Sec. 2.5. The joint use of these three methods reduces the run time to 10 to 15% in the case of the four favorable training sets (''Bridge,'' ''Camera,'' ''House,'' ''Airplane''), but only to about 70% in the case of residual vectors (''Miss America,'' ''Table Tennis''). The use of the lazy update (lazy PNN) gives further reduction of about 15 to 30%.

To sum up, if we combine all the idea presented in this paper (PDS, MPS, lazy update, initialization trick of Sec. 3.3), we can reduce the run time to 8 to 15% in the case of the favorable training sets (''Bridge,'' ''Camera,'' ''House,'' ''Airplane''), and to about 50% in the case of the unfavorable sets (''Miss America,'' ''Table Tennis'').

From the experiments we can see that the results greatly depend on the training set. An important parameter of the training set is the size (dimension) of the vectors. The effect of the vector size is therefore demonstrated in Fig. 9 with the artificial training sets described earlier in this section. Obviously, the overall run time increases as a function of the vector size. On the other, the relative improvement of the speed-up methods also increases, which partly compensates the increase in the vector size.

4.4 *Integration with Other Algorithms*

The improved PNN method benefits also other hybrid codebook generation methods. Here we have tested two of them: (1) GLA-PNN-GLA and (2) GA-PNN. The first method is a hybridization of the PNN method and the GLA due to Ref. 4. It starts with an initial codebook of size

Table 4 Run times (in seconds) for the six training sets ($M=256$).

	"Bridge"		"House"		"Miss America"	
	PNN	Lazy PNN	PNN	Lazy PNN	PNN	Lazy PNN
Full	79	52	1524	1126	229	145
PDS	42	28	1826	1318	143	91
MPS+PDS	13	10	231	195	152	108
	"Camera"		**"Airplane"**		**"Table Tennis"**	
	PNN	Lazy PNN	PNN	Lazy PNN	PNN	Lazy PNN
Full	73	51	8812	6237	2895	1816
PDS	35	25	10460	7145	1756	1109
MPS+PDS	8	6	1636	1295	2063	1462

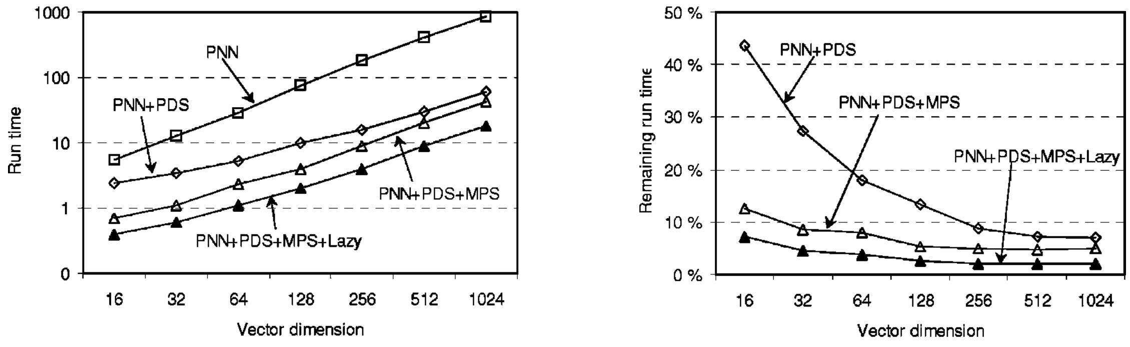

Fig. 9 Run time (in seconds) as a function of the code vector dimension (left), and remaining run time relative to the full search PNN (right).

$M_0 > M$, which is generated by the GLA. The resulting codebook is then reduced to the final size using the exact PNN method, and finally fine-tuned again by the GLA. The method is parametrized by the choice of M_0, and is denoted here as the GLA-PNN-GLA. The GLA is implemented as in Ref. 9.

The second method is the genetic algorithm (GA), as proposed in Ref. 6. The PNN method is used in the implementation of the crossover, as described in Ref. 14. The method is based on evolutionary computing; it uses a crossover to create new candidate solutions, and selection to direct the search toward better quality codebooks. The role of the PNN method is to provide high-quality candidate codebooks instead of using exhaustive trial-and-error approach. We denote this method as the GA-PNN.

The performance of these two methods are compared in Fig. 10 using "Bridge." The results clearly demonstrate that better quality codebooks can be obtained by the hybrid methods than using the PNN method alone. The choice between these two methods depends on whether the user prefers speed or quality. The GA-PNN is capable of providing the best codebooks at the cost of higher run time whereas the GLA-PNN-GLA is the better choice if the time is limited.

5 Conclusions

We have considered two different speed-up techniques for the PNN method: the PDS and the MPS. The methods can be considered practical as they achieve the speed-up without complicated data structures and without excessive increase of memory consumption, and they are easy to implement.

The PDS method works well in most cases and achieves a speed-up similar to that obtained within the GLA; roughly 50% with the favorable training sets. The use of the PDS technique is questionable only with vectors of very small dimension. In this case, the improvement can be overwhelmed by the overhead caused by the additional test. The use of the initial guess was also found to be important. Overall, the PDS is very efficient with vectors of very large dimension. For example, less than 10% run time is required with vectors of size 256 or more.

The MPS technique works also very well in most cases by reducing the run time to 10 to 20%. The exception is the set of residual vectors, for which only moderate improvement was obtained. The improvement is not as good as within the GLA for two reasons: (1) the calculation of the precondition is more complicated and takes more time, and (2) the precondition cannot be used to provide absolute bounds for the search, as in the GLA. Thus, the potential of the MPS technique can be utilized only partially in the context of the PNN method.

To sum up, if we combine all the speed-up methods discussed in this paper, we can reduce the run time to 8 to 15% in the case of the four favorable training sets, and down to about 50% in the case of the unfavorable sets (residual vectors). In the case of vectors with very large dimension (256 or greater) the run time of the favorable sets can be reduced to 2%.

We also demonstrated that the improvements are applicable within more sophisticated hybrid methods, in which the PNN method is used as a component. Two such methods were considered: the GLA-PNN-GLA, and the GA with a PNN crossover.

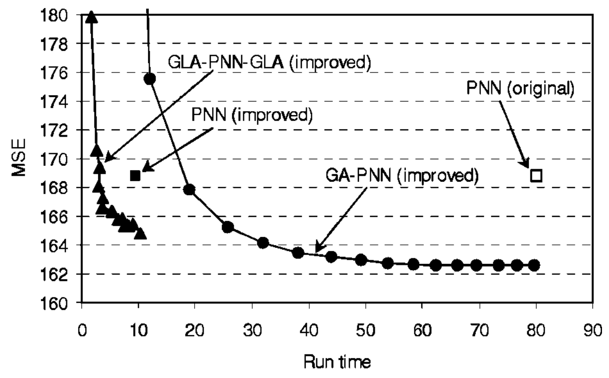

Fig. 10 Time-distortion performance of the hybrid algorithms for "Bridge." The GLA-PNN-GLA is parametrized by changing the size of the initial codebook from 256 to 4096. The results of the GA-PNN are shown from the first 15 iterations.

References

1. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Dordrecht (1992).
2. Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.* **28**(1), 84–95 (January 1980).
3. W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust., Speech, Signal Process.* **37**(10), 1568–1575 (1989).
4. 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 Trans. Circuits Syst. Video Technol.* **5**(2), 83–95 (1995).

5. T. Kaukoranta, P. Fränti, and O. Nevalainen, ''Iterative split-and-merge algorithm for VQ codebook generation,'' *Opt. Eng.* **37**(10), 2726–2732 (1998).
6. P. Fränti, J. Kivijärvi, T. Kaukoranta, and O. Nevalainen, ''Genetic algorithms for large scale clustering problem,'' *Comput. J. (UK)* **40**(9), 547–554 (1997).
7. J. Shanbehzadeh and P. O. Ogunbona, ''On the computational complexity of the LBG and PNN algorithms,'' *IEEE Trans. Image Process.* **6**(4), 614–616 (1997).
8. P. Fränti, T. Kaukoranta, D.-F. Shen, and K.-S. Chang, ''Fast and memory efficient implementation of the exact PNN,'' *IEEE Trans. Image Process.* **9**(5), 773–777 (2000).
9. T. Kaukoranta, P. Fränti, and O. Nevalainen, ''A fast exact GLA based on code vector activity detection,'' *IEEE Trans. Image Process.* **9**(8), 1337–1342 (2000).
10. C.-D. Bei and R. M. Gray, ''An improvement of the minimum distortion encoding algorithm for vector quantization,'' *IEEE Trans. Commun.* **33**(10), 1132–1133 (1985).
11. S.-H. Chen and W. M. Hsieh, ''Fast algorithm for VQ codebook design,'' *Proc. IEEE* **138**(5), 357–362 (1991).
12. S.-W. Ra and J.-K. Kim, ''A fast mean-distance-ordered partial codebook search algorithm for image vector quantization,'' *IEEE Trans. Circuits Syst.* **40**(9), 576–579 (1993).
13. T. Kaukoranta, P. Fränti, and O. Nevalainen, ''Vector quantization by lazy pairwise nearest neighbor method,'' *Opt. Eng.* **38**(11), 1862–1868 (1999).
14. P. Fränti, ''Genetic algorithm with deterministic crossover for vector quantization,'' *Pattern Recogn. Lett.* **21**(1), 61–68 (2000).

Olli Virmajoki received his MSc degree in electrical engineering from the Helsinki University of Technology, Finland, in 1983. He was a software engineer in industry from 1981 to 1985 and for Joensuu City from 1985 to 1996. Since 1998 he has been a lecturer with the Kajaani Polytechnic. He is also a doctoral student in the Department of Computer Science, University of Joensuu. His research interests are in machine vision and clustering algorithms.

Pasi Fränti received his MSc and PhD degrees in computer science in 1991 and 1994, respectively, from the University of Turku, Finland. From 1996 to 1999 he was a postdoctoral researcher with the University of Joensuu (funded by the Academy of Finland), where he has been a professor since 2000. His primary research interests are in image compression, vector quantization, and clustering algorithms.

Timo Kaukoranta received his MSc and PhD degrees in computer science from the University of Turku, Finland, in 1994 and 2000, respectively. Since 1997 he has been a researcher with the University of Turku. His primary research interests are in vector quantization and image compression.