A Fast Exact GLA Based on Code Vector Activity Detection

Timo Kaukoranta, Pasi Fränti, and Olli Nevalainen

***Abstract*—This paper introduces a new method for reducing the number of distance calculations in the generalized Lloyd algorithm (GLA), which is a widely used method to construct a codebook in vector quantization. Reduced comparison search detects the activity of the code vectors and utilizes it on the classification of the training vectors. For training vectors whose current code vector has not been modified, we calculate distances only to the active code vectors. Large proportion of the distance calculations can be omitted without sacrificing the optimality of the partition. The new method is included in several fast GLA variants reducing their running times over 50% on average.**

***Index Terms*—Clustering algorithms, codebook generation, image compression, vector quantization.**

NOMENCLATURE

T_i	Training vector.
C_j	Code vector.
P_i	Partition index of training vector T_i.
K	Dimension of the vectors.
N	Size of the training set ($N = \|T\|$).
M	Size of the codebook ($M = \|C\|$); typically $M \ll N$.

I. INTRODUCTION

We consider the codebook generation problem involved in the design of a *vector quantizer* [1]. The aim is to find 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 vectors are assumed to belong to a K-dimensional Euclidean space. There are several known methods for generating a codebook [2]–[6]. The *generalized Lloyd algorithm* (GLA) is probably the most cited and widely used method [7] because it is easy to implement and produces relatively good codebooks in short time. The algorithm starts with an initial solution, which is then iteratively improved until the process converges. Each iteration consists of partition step, where the vectors of the training set are partitioned into a set of M disjoint clusters, and of *codebook step*, where new code vectors are calculated.

Manuscript received December 1, 1998; revised January 11, 2000. The work of P. Fränti was supported under by a grant from the Academy of Finland. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Antonio Ortega.

T. Kaukoranta and O. Nevalainen are with the Turku Centre for Computer Science (TUCS), Department of Computer Science, University of Turku, Lemminkäisenkatu 14A, FIN-20520 Turku, Finland.

P. Fränti is with the Department of Computer Science, University of Joensuu, FIN-80101 Joensuu, Finland.

Publisher Item Identifier S 1057-7149(00)06142-X.

A straightforward implementation requires $O(NMK)$ time for a single iteration, which restricts the usefulness of the GLA especially in online applications. The GLA method is also commonly used as an integral part of other algorithms such as genetic algorithms [8], iterative split and merge [9], and tabu search [10]. It is characteristic of these algorithms that the GLA is repeated several times during a run of the algorithm and it has been reported that the GLA is the bottleneck of the methods what it comes to speed. Most of the computation in the GLA originates from the distance calculations between the training vectors and the code vectors. A single distance calculation takes $O(K)$ time only but there are MN distance calculations in total.

We introduce a new method for reducing the number of distance calculations in the partition step. The new method records the activity of the code vectors by considering whether they have been changed in the last codebook step or not. According to the activity of the code vectors, the clusters are classified into *active* and *static clusters*. For training vectors in the static clusters, it is sufficient to search for the nearest code vector among the active code vectors, only. A remarkable speed-up can therefore be achieved because the number of static clusters is usually high as the number of iteration grows. In addition, it turns out that we can also omit the recalculation of the distances for a large part of training vectors in the active clusters. The speed advantage of the new method depends on the stage of the iteration: the number of static clusters is rather small at the beginning but it increases rather quickly with the iterations when more and more clusters start to stabilize. The new method can be integrated to the original GLA and to several fast GLA variants.

II. VARIANTS OF THE FAST EXACT GLA

We consider a set of N training vectors T_i in a K-dimensional Euclidean space. The aim is to find a codebook C of M code vectors (C_j) by minimizing the *average squared distance between* the training vectors and their representative code vectors. The distance between two vectors is defined by their squared Euclidean distance

$$d(T_i, C_j) = ||T_i - C_j||^2 = \sum_{k=1}^{K} (T_{i,k} - C_{j,k})^2. \qquad (1)$$

Let C be a codebook and P the *partition of the training set*. Under the usual ergodicity assumption, the distortion of

```
Generate a codebook C^0 by any algorithm and set i ← 0;
REPEAT
    Generate optimal partition P^{i+1} for codebook C^i;
    Generate optimal codebook C^{i+1} for partition P^{i+1};
    Fill empty clusters;
    Set i ← i + 1;
UNTIL no improvement achieved.
```

Fig. 1. Pseudo-code of the GLA.

the codebook C is then determined using the presented training samples as

$$distortion(C) = \frac{1}{N} \sum_{i=1}^{N} d(T_i, C_{Pi}). \tag{2}$$

The basic structure of the GLA is shown in Fig. 1. The method starts by generating an initial codebook, which is then improved iteratively using the following two steps. In the *partition step*, each training vector T_i is mapped to its nearest code vector C_j in the current codebook

$$P_i = \arg \min_{1 \le j \le M} d(T_i, C_j). \tag{3}$$

The resulting partition P is optimal for the given codebook C according to (2). In the *codebook step*, a new codebook C' is constructed by calculating the centroids of the partitions

$$C'_j = \frac{\sum_{Pi=j} T_i}{\sum_{Pi=j} 1}, \qquad 1 \le j \le M. \tag{4}$$

The resulting codebook C' is optimal for the given partition P. The optimality of the two steps guarantees that the distortion value for the new solution is always equal to or smaller than for the previous one. The process is iterated as long as the distortion (2) reduces, or the number of iterations reaches a predefined limit. The number of iterations depends on the training set, and on the quality of the initial codebook; ten to 50 iterations are usually needed when starting from a randomly generated codebook.

The partition step dominates the running time of the GLA. For each training vector, the nearest code vector is found by an exhaustive search from the codebook. There are MN distances to be calculated in total, and each distance calculation takes $O(K)$ time. Thus, the total running time of the partition step is $O(NMK)$, which is also the time complexity of the whole iteration step. In the codebook step a new codebook is calculated by summing the training vectors in each cluster and dividing the sums by the sizes of the clusters. This takes $O(NK)$ time, which is negligible in comparison to the time required by the partition step.

The methods for speeding up the time expensive partition step can be categorized to *exact* and *approximate*. The exact methods always select the nearest code vector whereas the approximate methods use some heuristics for finding a close (not necessary the closest) code vector for the training vector. In the sequel, we concentrate on the exact methods that preserve the optimality of the partition step. We will briefly recall three known techniques of this type, and later show that our technique can be successfully integrated with each of these.

Speed-up is usually achieved by utilizing information of the candidate for being the nearest code vector. This candidate may be the code vector from the previous partition (which is usually a good initial guess), or it may be the nearest code vector found so far. Approaches are characteristic of utilizing the distance to the current candidate vector. The first approach to the utilization reduces the number of dimensions processed in distance calculations, and the second one reduces the number of distance calculations by excluding certain code vectors from the search. Of the following three fast techniques, the first one utilizes the first approach whereas the other two take both approaches at the same time.

In the *partial distortion search* (PDS) by Bei and Gray [11], the distance between two vectors is calculated cumulatively by summing up the squared differences in each dimension. If the cumulative distance exceeds the current minimum distortion at any time, the rest of the distortion calculation is omitted and the candidate is rejected. The effectiveness of this action depends on the quality of the current candidate. It is therefore advisable to calculate first the distance of the training vector to the (possibly updated) code vector C_{Pi}, which was the nearest at the previous iteration. In the rest of the paper, we suppose that this modification is applied in the PDS.

In the *triangular inequality elimination* technique (TIE) by Chen and Tsieh [12] the number of distance calculations is reduced by a condition derived from the triangle inequality. As the vectors are in an Euclidean space they satisfy the inequality

$$||T_i - C_a|| + ||T_i - C_b|| \ge ||C_a - C_b|| \tag{5}$$

where T_i is the training vector, C_a is the nearest code vector found so far, and C_b is another code vector. We can thus avoid the distance calculation between T_i and a code vector C_b if:

$$d(C_a, C_b) > 4d(T_i, C_a). \tag{6}$$

A practical implementation of this idea utilizes an $M \times M$ matrix of the distances between all code vectors. The entries of the matrix are recalculated at the beginning of each partition step. Efficient use of the distance matrix is possible if the rows are sorted into an increasing order by the distances. A drawback of the method is that it requires $O(M^2)$ extra space.

In TIE, the search for the nearest code vector proceeds as follows. The nearest code vector C_a of the previous partition is chosen as the initial guess. The rest of the code vectors are then checked using the PDS in the order given by the row a of the distance matrix. Only the first code vectors up to the limit given by the rule (6) are checked, and the rest can be eliminated. The overall running time is reduced if the speed-up is greater than the overhead from generating the distance matrix. This is usually the case (assuming that $M \ll N$) since the calculation of the distance matrix takes $O(M^2K)$ time whereas the time complexity of the partition step is $O(NMK)$.

The *mean-distance-ordered partial search* (MPS) technique introduced by Ra and Kim [13] is designed for the encoding part of a vector quantizer but it also applies to the partition step of

the GLA. They show that the calculation of $d(T_i, C_j)$ can be avoided if

$$\left(\sum_{k=1}^{K} T_{i,k} - \sum_{k=1}^{K} C_{j,k}\right)^2 > K \cdot d(T_i, C_a) \qquad (7)$$

where C_a is the current nearest code vector. The left side of the condition is very fast to evaluate using preprocessing.

An efficient implementation of MPS sorts the codebook into an ascending order of component means of the code vectors. An initial guess for C_a is the code vector whose mean value is closest to that of T_i. This is found by using binary search. Equation (7) gives lower and upper limits for the bi-directional search, which utilizes the PDS technique. The method reduces the running time if the improvement from the rejection of the vectors exceeds the overhead caused by the sorting, which takes $O(M \log M)$ time.

III. Reduced-Comparison Search

It is observed that the GLA makes only local changes in the codebook and the amount of changes differs from vector to vector. Some code vectors will therefore stabilize within few iterations while others develop much longer before stabilizing. This observation is utilized by detecting the *activity* of the code vectors indicating whether the code vector was changed during the previous iteration. The code vectors are classified into two groups: *active* and *static vectors*. The cluster of a static code vector is called a *static cluster*. The number of static code vectors increases with the iteration.

The activity information is used for reducing the number of distance calculations in the case of vectors in static clusters. Consider a training vector T_i in a static cluster P_i, and denote the code vector of this cluster by C_{Pi}. Our first observation is that if another static code vector C_j was not the nearest code vector for T_i in the previous iteration, it cannot be the nearest code vector in the current iteration either. The partition of the training vectors in a static cluster cannot therefore change to another static cluster. For these vectors, it is sufficient to calculate the distances only to the active code vectors because only they may become closer than the code vector of the current cluster.

Our second observation is that the reduced-comparison search can also be applied for certain training vectors in an active cluster. Consider a training vector T_i in an active cluster a. If the code vector C_a moves closer to T_i than it was before the update, then T_i can be treated as it was in a static cluster. In practice, we do not classify the training vectors according to their cluster status but according to the direction of the movement of the code vector [see Fig. 2]. To implement this extension we store for each training vector T_i its distance to the current code vector C_j. For the remaining training vectors (whose distance to current code vector was increased) we must perform full search by calculating the distances to all code vectors in the codebook.

To implement the reduced comparison search we maintain a set of the active code vectors. This is done in the codebook step by comparing the new and the previous code vectors with at most $O(MK)$ extra work and $O(M)$ extra space. In the partition step, we first check for each training vector T_i if its current code vector C_{Pi} is in the set of active vectors. If this is not the case, we perform the search for the nearest code vector from the set of the active code vectors only by using any fast search method. Otherwise, when T_i is mapped to active code vector C_{Pi}, we calculate the distance $d(T_i, C_{Pi})$. If this distance is greater than the distance in previous iteration, we perform a full search using any fast search method. If, however, the distance does not increase we perform the search from the set of the active code vectors only. To store the distances of the previous iteration we need $O(N)$ extra space.

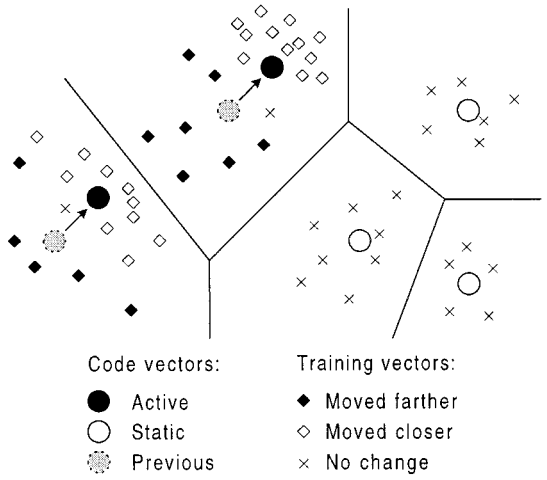

Fig. 2. Example of the classification of the vectors.

Denote the number of active clusters by M_a and the number of training vectors in these clusters by N_a. The proportion of the training vectors (in active clusters) whose distance to the current code vector is increased is denoted by p. The number of distance calculations is thus $(1-p)\cdot N_a M_a + pN_a M$ in the case of active clusters, and $(N - N_a) \cdot M_a$ in the case of static clusters. It is expected that $p \approx 0.5$ because of the centroid operation. The total number of distance calculations in a single partition step is therefore

$$\begin{aligned}(N - N_a)M_a + (1-p)N_a M_a + pN_a M \\ = NM_a + pN_a(M - M_a).\end{aligned} \qquad (8)$$

In other words, the distances to the active code vectors must be calculated for all training vectors, but the distances to the static code vectors only for a part (pN_a) of the training vectors. These are the training vectors of the active clusters for which the distance to their current code vector has increased. The benefit of the reduced comparison search depends directly on the number of active clusters M_a.

The reduced comparison search can be implemented with any fast GLA variant as follows. The partition step contains two different kind of searches. Some search operations are performed for the full codebook as usual, and the other searches for the subcodebook of the active code vectors. This must be considered in the implementation because some GLA variants (like TIE and MPS) use special data structures for the search. In practice, we must generate search structures both for the full codebook and for the subcodebook. We generate first the structure for the full codebook as usual and then eliminate the static vectors for ob-

```
Generate an initial codebook C;
Mark all code vectors as changed, i.e. let the subcodebook be Ĉ←C;
REPEAT
    Generate the search structure for the full codebook C;
    Generate the search structure for the subcodebook Ĉ of active vectors;
    For each T_i
        IF C_Pi did not change or it become closer
        THEN Search from the subcodebook Ĉ;
        ELSE Search from the full codebook C;
    Calculate a new codebook C;
    Generate a subcodebook Ĉ;
UNTIL convergence.
```

Fig. 3. Pseudo-code of the GLA with reduced comparison search.

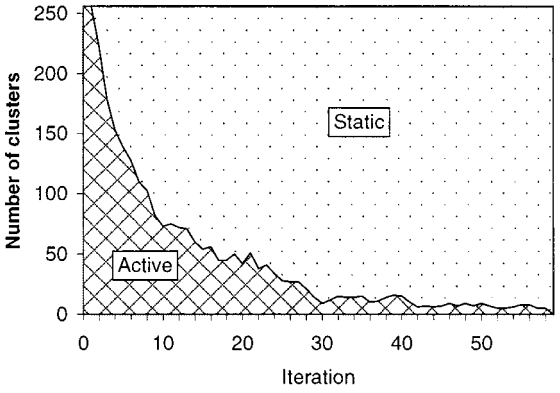

Fig. 4. Number of active and static clusters during the iteration (for *Miss America*, $N = 19\,940$, $M = 256$).

taining the search structure for the subcodebook. Fig. 3 summarizes the control flow of the GLA with the reduced comparison search.

IV. Test Results

We generated training sets from three different images: *bridge* (256 × 256 pixels, 8 bits per pixel), *Miss America* (360 × 288 pixels, 8 bits per pixel), and *house* (256 × 256 pixels, 24 bits per pixel). The vectors in the first set (*bridge*, $N = 4096$, $K = 16$) are 4 × 4 pixel blocks from the image. The second set (*Miss America*, $N = 19\,940$, $K = 16$) has been obtained by constructing 4 × 4 spatial pixel blocks from the first three frames of the video sequence. The third data set (*House*, $N = 65\,536$, $K = 3$) consists of color values of the RGB image. Applications of this kind of data sets are found in image and video image coding (*bridge, Miss America*), and in color image quantization (*house*).

In the following we study the behavior of the proposed method using *Miss America* as a test case, and then give a summary of the main results for the three test sets. The classification of the clusters is illustrated in Fig. 4, and the classification of the training vectors in Fig. 5. We observe that the number of active clusters decreases rapidly and during the last thirty iterations it is close to zero. The proportion of static clusters was 82% in total, when observing all iteration steps. The number of favorable training vectors develops in similar way. In total, 83% of the training vectors were in static clusters ("no change"), and 8% of the vectors have decreased distance to

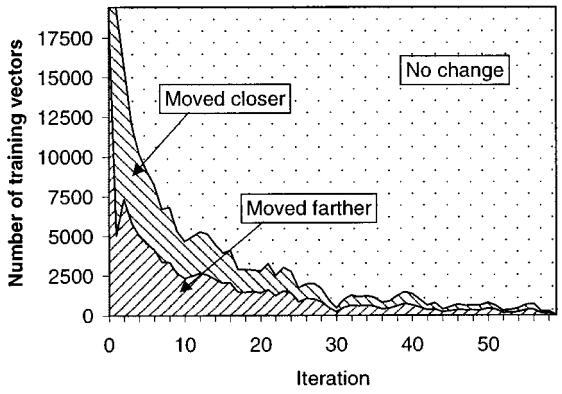

Fig. 5. Classification of the training vectors in respect to their distance to the code vector of their current cluster (for *Miss America*, $N = 19\,940$, $M = 256$).

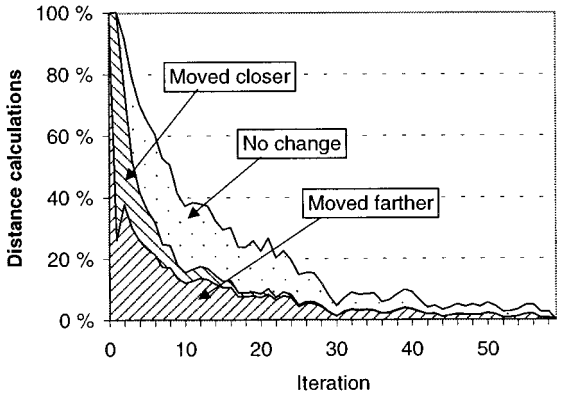

Fig. 6. Number of distance calculations (relative to the full search) when activity grouping is applied to the original GLA (for *Miss America*, $N = 19\,940$, $M = 256$).

their code vector ("moved closer"). Full search was performed for the rest 9% of the training vectors ("moved farther").

Fig. 6 illustrates from which type of vectors the computation originates. Almost 40% of all distance calculations of the improved GLA are caused by the training vectors in the "moved farther" group. The favorable training vectors are not completely free of computation either. In total, over 40% of all distance calculations comes from the training vectors in the "no change" group, and less than 20% from the "moved closer" group. Over half of the distance calculations originate from the first ten iterations.

Next we study how the new method applies for different GLA variants. The amount of work generated by the distance calculations is summarized in Table I. Results are given for the original methods (without activity grouping) and for the methods improved by our technique (with activity grouping). In all cases, initial codebook is generated by selecting random training vectors omitting duplicates. For the full search the number of distance calculations is sometimes below the size of the codebook $(M = 256)$ because we have stopped the search when an equal code vector (distance 0) was found. The number of distance calculations performed by the TIE and MPS methods depends on the training set, and it varies from 4% to 74% of full search. Our grouping technique gives further reduction, which is more than 50%, within all GLA variants for the tested training sets.

TABLE I
AVERAGE NUMBER OF DISTANCE CALCULATIONS PER TRAINING VECTOR AND THE AVERAGE NUMBER OF PROCESSED VECTOR DIMENSIONS DURING THE DISTANCE CALCULATIONS. THE RESULTS ARE AVERAGES OF TEN RUNS (FOR *MISS AMERICA*, $N = 19\,940$, $M = 256$, $K = 16$)

	Without activity grouping			With activity grouping		
	Distance calculations / search	Dimensions / distance calculation	Dimensions / search	Distance calculations / search	Dimensions / distance calculation	Dimensions / search
Full	255.97	16.00	4095.48	61.44	16.00	983.07
PDS	255.97	2.34	598.96	61.44	2.08	127.98
TIE	37.90	7.70	291.64	8.26	8.57	70.80
MPS	26.97	8.07	217.60	5.35	6.72	35.97

TABLE II
RUNNING TIMES (IN SECONDS) FOR THE THREE TRAINING SETS. RESULTS ARE AVERAGES OF 50 RUNS ($M = 256$)

	Bridge		Miss America		House	
	Without grouping	With grouping	Without grouping	With grouping	Without grouping	With grouping
Full	127.6	46.1	1344.5	336.2	874.8	138.8
PDS	33.4	13.0	311.1	75.8	460.7	85.0
TIE	21.4	13.5	135.2	44.6	43.2	28.8
MPS	12.4	4.8	97.3	21.5	55.2	15.5

The second columns in Table I demonstrate the effect of the PDS technique. The PDS alone reduces the number of processed dimensions to the range from 15 to 37 per cent depending on the training set. Both the TIE and the MPS utilize the PDS in distance calculations. However, they have to process roughly 50% of the dimensions because they calculate distances mainly to nearby code vectors and these calculations proceed quite further before exiting due to the PDS. It is observed that our method has no significant effect on the number of processed dimensions. This indicates that the grouping technique utilizes different properties than the PDS. The total amount of distance calculations (per training vector per iteration) is shown in the third columns of Table I.

The overall running times of the GLA variants are summarized in Table II. The speed-up achieved by our method ranges from 30% to 80% depending on the GLA variant and on the training set. Among the different GLA variants, the TIE is the fastest method for *House* and the MPS for the other two sets, when our technique is not applied. With our grouping technique, the MPS is the best for all test sets. This variant takes only about 2% to 4% of the time required by the full search.

Fig. 7 illustrates the effect of the codebook size M on the running time. The MPS is relatively independent on the size of the codebook. The TIE, on the other hand, performs worse for larger values of M because the maintenance of the distance matrix still requires $O(M_a M K)$ time, where M_a is the number of active code vectors. The relative improvement of the activity grouping tends to increase as a function of the codebook size, but it stabilizes for large codebooks, see Fig. 8. In TIE, however, the time required by the maintenance of the distance matrix increases faster than the saving achieved by the activity grouping technique. This is visible in the relatively weak results of TIE in Fig. 8. Note that the activity grouping is used to reduce the ordering operation of the algorithm.

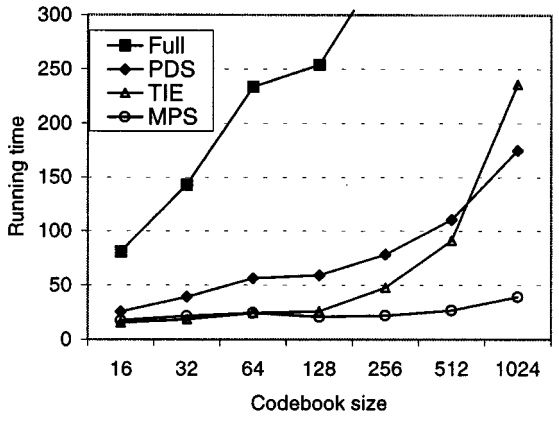

Fig. 7. Running time (in seconds) as a function of the codebook size. The grouping technique is applied in all GLA variants. The methods marked by an open symbol (TIE and MPS) reduce the number of distance calculations. The results are averages of ten runs (for *Miss America*, $N = 19\,940$).

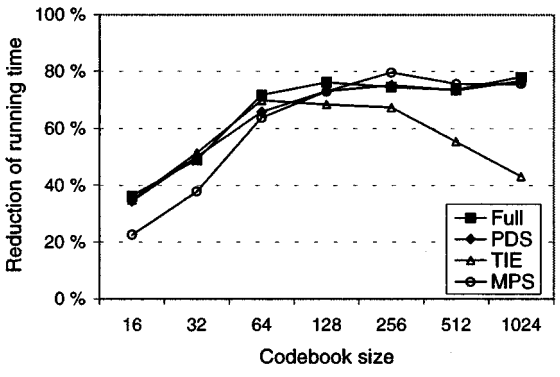

Fig. 8. Improvement obtained by the grouping technique within the GLA variants as a function of the codebook size. The results are averages of ten runs (for *Miss America*, $N = 19\,940$).

V. Conclusions

We introduced an improvement to the exact GLA that preserves the optimality of partition. The key observation is that a large proportion of the distance calculations is unnecessary because only few of the code vectors remain active to the end of the iterations. Significant speed-up is therefore obtained in the partition step by detecting the activity of the clusters. The proposed method, however, is only for speeding up the codebook generation and the technique cannot be used in the encoding part of vector quantization.

An important property of the new method is that the speed-up is not limited only to the full-search GLA but it applies also to other faster GLA variants. The method is capable of improving the running time by 60% to 80% in the case of the PDS and MPS methods, and by 30% to 60% in the case of the TIE method. Overall, the fastest method was the MPS augmented with our grouping technique. Moreover, the speed-up is based on a general property of the GLA and not on a property of the distance metric such as the triangular inequality. We therefore expect that the method is suitable also for applications using different, possibly nonmetric distance function.

As noted by one of the reviewers, the above organization of the reduced comparison GLA is not the only possible. The method could be enhanced by maintaining an $N \times M$ matrix of the distances between all training vectors and the code vectors. The remaining distance calculations to static code vectors for the "moved farther" training vectors can then be saved. In this way, the number of distance calculations of full search GLA can be further reduced from 23% to 19% in case of *Miss America* at the cost of $O(NM)$ extra space. This is a point of future studies.

References

[1] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Dordrecht, The Netherlands: Kluwer, 1992.

[2] N. Akrout, R. Prost, and R. Goutte, "Image compression by vector quantization: A review focused on codebook generation," *Image Vis. Comput.*, vol. 12, pp. 627–637, Dec. 1994.

[3] W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1568–1575, Oct. 1989.

[4] P. Fränti, T. Kaukoranta, and O. Nevalainen, "On the splitting method for VQ codebook generation," *Opt. Eng.*, vol. 36, pp. 3043–3051, Nov. 1997.

[5] N. M. Nasrabadi and Y. Feng, "Vector quantization of images based upon the Kohonen self-organization feature maps," *Neural Networks*, vol. 1, no. 1, p. 518, 1988.

[6] K. Zeger and A. Gersho, "Stochastic relaxation algorithm for improved vector quantiser design," *Electron. Lett.*, vol. 25, pp. 896–898, July 1989.

[7] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. Commun.*, vol. 28, pp. 84–95, Jan. 1980.

[8] P. Fränti, J. Kivijärvi, T. Kaukoranta, and O. Nevalainen, "Genetic algorithms for large scale clustering problem," *Comput. J.*, vol. 40, no. 9, pp. 547–554, 1997.

[9] T. Kaukoranta, P. Fränti, and O. Nevalainen, "Iterative split-and-merge algorithm for VQ codebook generation," *Opt. Eng.*, vol. 37, pp. 2726–2732, Oct. 1998.

[10] P. Fränti, J. Kivijärvi, and O. Nevalainen, "Tabu search algorithm for codebook generation in VQ," *Pattern Recognit.*, vol. 31, pp. 1139–1148, Aug. 1998.

[11] C.-D. Bei and R. M. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Trans. Commun.*, vol. 33, pp. 1132–1133, Oct. 1985.

[12] S.-H. Chen and W. M. Hsieh, "Fast algorithm for VQ codebook design," *Proc. Inst. Elect. Eng.*, vol. 138, pp. 357–362, Oct. 1991.

[13] S.-W. Ra and J.-K. Kim, "A fast mean-distance-ordered partial codebook search algorithm for image vector quantization," *IEEE Trans. Circuits Syst.*, vol. 40, pp. 576–579, Sept. 1993.

Timo Kaukoranta received the M.Sc. and Ph.D. degrees in computer science from the University of Turku, 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.

Pasi Fränti received the M.Sc. and Ph.D. degrees in computer science in 1991 and 1994, respectively, from the University of Turku, Turku, Finland.

From 1996 to 1999, he was a Postdoctoral Researcher with the University of Joensuu, funded by the Academy of Finland. Since 2000, he has been a Professor. His primary research interests are in image compression, vector quantization, and clustering algorithms.

Olli Nevalainen received the M.Sc. and Ph.D. degrees in 1969 and 1976, respectively.

From 1972 to 1979, he was a Lecturer with the Department of Computer Science, University of Turku, Turku, Finland. From 1976 to 1999, he was an Associate Professor, and since 1999 a Professor in the same department. He lectures in the areas of data structures, algorithm design and analysis, compiler construction, and operating systems. His research interests are in the broad field of algorithm design, including image compression, scheduling algorithms, and production planning.