Fast pairwise nearest neighbor based algorithm for multilevel thresholding

Olli Virmajoki
Pasi Fränti
University of Joensuu
Department of Computer Science
Box 111
FIN-80101 Joensuu
Finland
E-mail: franti@cs.joensuu.fi

Abstract. *We propose a fast pairwise nearest neighbor (PNN)-based O(N log N) time algorithm for multilevel nonparametric thresholding, where N denotes the size of the image histogram. The proposed PNN-based multilevel thresholding algorithm is considerably faster than optimal thresholding. On a set of 8 to 16 bits-per-pixel real images, experimental results also reveal that the proposed method provides better quality than the Lloyd-Max quantizer alone. Since the time complexity of the proposed thresholding algorithm is log linear, it is applicable in real-time image processing applications.* © 2003 SPIE and IS&T. [DOI: 10.1117/1.1604396]

1 Introduction

Thresholding is one of the most common operations in image processing. It tries to extract a target from the background on the basis of the distribution of pixel values. Most thresholding techniques are based on the statistics of the 1-D histogram of the gray levels, or on the 2-D co-occurrence matrix of an image.

Many 1-D thresholding methods have been investigated.[1–10] Locating the thresholds can occur in parametric or nonparametric approaches. In parametric approaches,[3,5] the gray-level distribution of an object class leads to finding the thresholds. In Wang and Haralick's study,[3] the pixels of an image are first classified as either edge or nonedge pixels. According to their local neighborhoods, edge pixels are then classified as being relatively dark or relatively bright. Next, one histogram is obtained for the dark-edge pixels and another one for the bright-edge pixels. The highest peaks of these two histograms are chosen as the thresholds. Moment preserving thresholding is another parametric method, which is based on the condition that the thresholded image has the same moments as the original image.[5]

In nonparametric approaches,[2,7,8] the thresholds are obtained in an optimal manner according to some criteria. For instance, Otsu's method chooses the optimal thresholds by maximizing the between-class variance with an exhaustive search.[2] In Pun's method,[7] as modified by Kapur, Sahoo, and Wong,[8] the threshold is found by maximizing the entropy of the histogram of gray levels of the resulting classes. 1-D thresholding techniques can also be generalized from bilevel thresholding to multilevel thresholding.[1–5] In contrast to 1-D thresholding methods, 2-D methods essentially do image segmentation by using spatial information in an image.[11–16] Kirby and Rosenfeld[11] proposed a 2-D thresholding method that simultaneously considers both the pixel gray level and the local statistics of its neighboring pixels. One particular 2-D method is entropic thresholding, which makes use of spatial entropy to find the optimal thresholds. Abutaleb,[12] and Pal and Pal[13] proposed that optimal thresholds can be selected by maximizing the sum of the posterior entropies of two classes. Chen, Wen, and Yang[15] proposed a two-stage approach to search for the optimal threshold of 2-D entropic thresholding, so that the computation complexity can be reduced to $O(N^{8/3})$ for an image with N gray levels. Recently, Gong, Li, and Chen[16] designed a recursive algorithm for 2-D entropic thresholding to further reduce the computation complexity to $O(N^2)$. However, it is still inefficient to apply this algorithm to a 1-D multilevel thresholding selection, owing to their computation of threshold without taking advantage of the recursive structure of entropy measures.

Sahoo *et al.* concluded in their study[6] on global thresholding that Otsu's method was one of the best threshold selection methods for general real-world images with regard to uniformity and shape measures. However, Otsu's method uses an $O(N^{M-1})$ time exhaustive search to solve the thresholds that maximize the between-class variance, where M is the number of classes. As the number of thresholds increases, Otsu's method takes too much time to be practical for multilevel thresholding. Recently, Liao, Chen, and Chung[10] proposed a recursive form of the modified between-class variance and achieved a significant speed up of Otsu's method. However, the faster implementation of Otsu's method still has the same time complexity.

Paper 02074 received Jul. 9, 2002; revised manuscript received Mar. 3, 2003, and May 30, 2003; accepted for publication Jun. 3, 2003.
1017-9909/2003/$15.00 © 2003 SPIE and IS&T.

The optimal multilevel thresholding can still be a bottleneck in real-time machine vision applications. For example, in medical imaging the images can have more than 8 bits per pixel, and if we have ten distinct threshold values, the number of operations in optimal thresholding can be $\sim 65536^{10}$. There are also faster suboptimal methods for calculating the threshold values. The Lloyd-Max quantizer (LMQ)[17] takes only $O(N)$ operations per iteration, but it does not necessarily provide optimal thresholding. Methods from vector quantizer can also be applied for calculating the threshold values, as the thresholding can be considered as a special case of vector quantization.

We propose a new multilevel thresholding algorithm derived from the well-known pairwise nearest neighbor (PNN) method previously used in vector quantization.[18] The PNN has been considered a slow algorithm, as the time complexity of the original method was shown to be $O(N^3)$.[19] Faster approaches have been proposed by Kurita[20] using a heap structure, and by Fränti *et al.*[21] using nearest neighbor pointers. Nevertheless, the time complexity of the PNN is still lower limited by $\Omega(N^2)$ in vector quantization. It is therefore not self-evident that the PNN could be useful in real-time applications.

Our contribution is to show that PNN thresholding can be implemented efficiently. Unlike vector quantization, the one-dimensionality of the histogram can be utilized so that the neighbor classes can be determined by using a simple linked list structure. This allows constant time update of the data structures. At the same time, we use a heap structure for the search of the minimum cost class pair. The time complexity of a single step of the algorithm takes $O(\log N)$ time, and the overall algorithm takes $O(N \log N)$ time. Thus, the proposed method is significantly faster than the optimal multilevel thresholding. By experiments, we also show that the quality of PNN thresholding is closer to the optimal solution than the quality of the Lloyd-Max quantizer.

The rest of the work is organized as follows. In Sec. 2, we first define the thresholding problem, and then recall the relevant thresholding methods. These include the optimal thresholding and the Lloyd-Max quantizer, which we use for comparison. The PNN-based thresholding method is then proposed in Sec. 3. Experimental results are reported in Sec. 4, and conclusions are drawn in Sec. 5.

2 Multilevel Thresholding

Suppose that the image consists of one or more objects and background, each having distinct gray-level values. The purpose of thresholding is to separate these areas from each other by using the information given by the histogram of the image. If the object and the background have clear uniform gray-level values, the object can be detected by splitting the histogram using a suitable threshold value. Then, any point (x,y) for which $f(x,y)>t$ is called an object point; otherwise, the point is called a background point. The thresholding can be performed by:

$$g(x,y)=\begin{cases}1 & \text{if } f(x,y)>t\\ 0 & \text{if } f(x,y)\leq t\text{'}\end{cases} \tag{1}$$

where x and y are the coordinates of the pixel and $f(x,y)$ is the intensity value of the pixel. The time complexity of optimal thresholding with one threshold value is $O(N)$.

If the image histogram is characterized by three or more dominant modes, we need multilevel thresholding to extract the objects from the background. Here with three modes, the same basic approach classifies a point (x,y) as belonging to one object class if $t_1<f(x,y)\leq t_2$, to the other object class if $f(x,y)>t_2$, and to the background if $f(x,y)\leq t_1$. The optimal thresholding with $M-1$ threshold values can be achieved by an exhaustive search with a time complexity of $O(N^{M-1})$.

2.1 *Otsu's Method*

An image is a 2-D grayscale intensity function, and contains N_{pixels} pixels with gray levels from 1 to L. The number of pixels with gray level i is denoted x_i, giving a probability of gray level i in an image of

$$p_i=x_i/N_{\text{pixels}}. \tag{2}$$

In the case of bilevel thresholding of an image, the pixels are divided into two classes, s_1 with gray levels $[1,...,t]$, and s_2 with gray levels $[t+1,...,L]$. Let μ_k and μ_T be the mean intensities for the class s_k and whole image, respectively. Using discriminant analysis, Otsu[2] defined the between-class variance of the thresholded image as:

$$\sigma_B^2=\omega_1(\mu_1-\mu_T)^2+\omega_2(\mu_2-\mu_T)^2, \tag{3}$$

where $\omega_1(t)$ and $\omega_2(t)$ are cumulative sums of the probabilities in each class. For bilevel thresholding, Otsu verified that the optimal threshold t^* is chosen so that the between-class variance σ_B^2 is maximized.

The previous formula, Eq. (3) can be easily extended to multilevel thresholding of an image. Assuming that there are $M-1$ thresholds, $\{t_1,t_2,\ldots,t_{M-1}\}$, which divide the original image into M classes: s_1 for $[1,...,t_1]$, s_2 for $[t_1+1,...,t_2],\ldots,s_i$ for $[t_{i-1}+1,...,t_i],\ldots,$ and s_M for $[t_{M-1}+1,...,L]$, the optimal thresholds $\{t_1^*,t_2^*,\ldots,t_{M-1}^*\}$ are chosen by maximizing:

$$\sigma_B^2(t_1,t_2,\ldots,t_{M-1})=\sum_{k=1}^{M}\omega_k(\mu_k-\mu_T)^2. \tag{4}$$

Regardless of the number of classes being considered during the thresholding process, the sum of the cumulative probability functions of M classes equals one, and the mean of the image is equal to the sum of the means of M classes weighted by their cumulative probabilities. The between-class variance in Eq. (4) of the thresholded image can thus be rewritten as:

$$\sigma_B^2(t_1,t_2,\ldots,t_{M-1})=\sum_{k=1}^{M}\omega_k\mu_k^2-\mu_T^2. \tag{5}$$

Because the second term in Eq. (5) is independent of the choice of the thresholds $\{t_1,t_2,\ldots,t_{M-1}\}$, the optimal thresholds $\{t_1^*,t_2^*,\ldots,t_{M-1}^*\}$ can be chosen by maximizing a modified between-class variance $(\sigma_B')^2$, defined as the summation term on the right-hand side of Eq. (5):[10]

$$(\sigma_B')^2(t_1,t_2,\ldots,t_{M-1})=\sum_{k=1}^{M}\omega_k\mu_k^2. \tag{6}$$

To reduce the computations of Eq. (6), we make a small modification to it by using absolute frequencies x_i instead of the relative frequencies p_i:

$$(\sigma_B')^2(t_1,t_2,\ldots,t_{M-1})=\sum_{k=1}^{M}\left(\sum_{i\in s_k} ix_i\right)^2 \Bigg/ \sum_{i\in s_k} x_i\,. \tag{7}$$

In this way, we can save at least N_{pixels} divisions in comparison to the original implementation. In the rest of this work, we call this Otsu's method. Still a faster algorithm can be achieved by recursive calculation of Eq. (7). Let us define the look-up tables for the $u-v$ interval:

$$P(u,v)=\sum_{i=u}^{v} x_i \quad \text{and} \quad S(u,v)=\sum_{i=u}^{v} ix_i\,. \tag{8}$$

For index $u=1$, Eq. (8) can be rewritten as:

$$\begin{aligned}&P(1,v+1)=P(1,v)+x_{v+1} \quad \text{and} \quad P(1,0)=0,\\ &S(1,v+1)=S(1,v)+(v+1)x_{v+1} \quad \text{and} \quad S(1,0)=0,\end{aligned} \tag{9}$$

where x_{v+1} is the number of pixels of the gray level being $v+1$. From Eqs. (8) and (9), it follows that:

$$P(u,v)=P(1,v)-P(1,u-1)$$

and

$$S(u,v)=S(1,v)-S(1,u-1). \tag{10}$$

Now we can write:

$$P(t_{k-1}+1,t_k)=P(1,t_k)-P(1,t_{k-1})$$

and

$$S(t_{k-1}+1,t_k)=S(1,t_k)-S(1,t_{k-1}). \tag{11}$$

In Eq. (11), note that t_0 and t_M are defined as $t_0=0$ and $t_M=L$, respectively. Now the modified between-class variance $(\sigma_B')^2$ can be rewritten as:

$$\begin{aligned}(\sigma_B')^2(t_1,t_2,\ldots,t_i,\ldots,t_{M-1})=H(1,t_1)+H(t_1+1,t_2)+\ldots\\ +H(t_{M-1}+1,L),\end{aligned} \tag{12}$$

where the modified between-class variance of class s_i is defined as:

$$H(t_{i-1}+1,t_i)=S(t_{i-1}+1,t_i)^2/P(t_{i-1}+1,t_i). \tag{13}$$

In the rest of this work, we call this fast Otsu's method. The algorithm for the fast Otsu's method is shown in Fig. 1.

```
Fast_OTSU($X$, $x_{max}$, $M$) → $T$
    $P(1,0) \leftarrow 0$;
    $S(1,0) \leftarrow 0$;
    FOR $v \leftarrow 0$ TO $x_{max}-1$
        $P(1, v+1) \leftarrow P(1,v) + x_{v+1}$;
        $S(1,v+1) \leftarrow S(1,v) + (v+1)x_{v+1}$;
    END-FOR
    FOR $v \leftarrow 1$ TO $x_{max}$
        FOR $u \leftarrow 1$ TO $v$
            $H(u,v) \leftarrow S(u,v) * S(u,v) / P(u,v)$;
        END-FOR
    END-FOR
    $Smax \leftarrow 0.0$;
    FOR $\forall(t_1,t_2,\ldots,t_i,\ldots,t_{M-1})$  $1 \le t_1 < \ldots < t_{M-1} < x_{\max}$
        $(\sigma_B')^2(t_1,t_2,\ldots,t_i,\ldots,t_{M-1}) \leftarrow H(1,t_1)+H(t_1+1,t_2)+\ldots+H(t_{M-1}+1,x_{\max})$;
        IF $((\sigma_B')^2 > Smax)$
            $Smax \leftarrow (\sigma_B')^2$;
            $T \leftarrow (t_1,t_2,\ldots,t_i,\ldots,t_{M-1})$;
        END-IF
    END-FOR ;
```

Fig. 1 The algorithm for the fast Otsu's method for thresholding.

2.2 *Lloyd-Max Quantizer*

Lloyd[17] has proposed an algorithm known as the Lloyd-Max quantizer (LMQ) for solving M-level scalar quantization problems. The algorithm aims at minimizing the mean square error (MSE), which is equivalent to the multilevel thresholding problem maximizing intercluster variance. The LMQ algorithm starts with a set of initial threshold values, which are modified through a sequence of iterations. The LMQ applies the necessary conditions for an optimal quantizer, which appears to have been first discovered by the Polish mathematicians Lukaszewicz and Steinhaus.[22]

- Nearest neighbor condition: For a given set of class representatives, each pixel value is mapped to its nearest class centroid.
- Centroid condition: For a given mapping, the optimal class representatives are the average values of the classes.

```
LMQ($X$, $x_{min}$, $x_{max}$, $M$) → $T$, $C$
    FOR $i \leftarrow 0$ TO $M$ DO $t_i \leftarrow x_{min} + (i * (x_{max} - x_{min})) / M$ ;
    REPEAT
        Segment the image using $t_i$; $i \leftarrow 1, 2, \ldots, M-1$ ;
        Calculate segment averages:
        FOR $i \leftarrow 1$ TO $M$ DO $c_i \leftarrow \sum_{j=t_{i-1}+1}^{t_i} j x_j \Big/ \sum_{j=t_{i-1}+1}^{t_i} x_j$ ;
        Recalculate thresholds:
        FOR $i \leftarrow 1$ TO $M$-1 DO $t_i \leftarrow (c_i + c_{i+1}) / 2$ ;

    UNTIL $T$ does not change ;
```

Fig. 2 Structure of the Lloyd-Max quantizer.

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

Fig. 3 Structure of the PNN method.

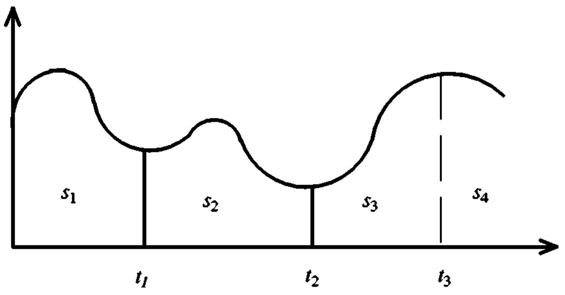

Fig. 4 Removal of the threshold t_3 corresponds to the merge of clusters s_3 and s_4.

The LMQ for thresholding is presented in Fig. 2. The method starts by generating initial thresholding values t_i; $i=1,2,...,M-1$, in which the gray-level values are divided to M partitions, which are the intervals of equal size. We call this initial thresholding the uniform quantizer. On the basis of the two optimality conditions, the Lloyd-Max algorithm for thresholding is formulated as a two-phase iterative process: 1. divide the data vectors x_i to the segments according to the current threshold values, and 2. calculate the segment averages of each thresholded region. The new threshold values are now calculated as an average of two neighbor segment averages.

Both stages of the algorithm satisfy the optimality conditions, thus the resulting thresholding after one iteration can never be worse than the original one. The iterations are continued until no change is achieved. However, the final result is not necessarily global optima, because the iteration can be stopped in a local optima.

The calculation of the new threshold values takes $O(M)$ time because we compute $M-1$ new threshold values as the average values of the neighboring segments. The calculation of the segment averages takes $O(N)$ time, because we compute M new segment averages each having N/M elements on average. The calculation of the segment averages dominates the time complexity of the LMQ, which is thus $O(N)$.

3 PNN-Based Thresholding

Thresholding can be considered as a special case of vector quantization (VQ), where the vectors are 1-D only. It is therefore expected that algorithms developed for VQ can also be applied to the multilevel thresholding problem.

3.1 *Vector Quantization*

The goal of vector quantization is to map a set of K-dimensional input vectors to a reduced set of representative vectors, so that the quantization error is minimized.[23] The set of representative vectors are called code vectors, and set of code vectors is called codebook. The quantization error is usually measured by the MSE between the data vectors and the code vectors. Given a set of N input vectors $X=\{x_1,x_2,\ldots,x_N\}$, and the codebook $C=\{c_1,c_2,\ldots,c_M\}$, the error is calculated as:

$$\mathrm{MSE}(C,P)=\frac{1}{N}\cdot\sum_{i=1}^{N}\|x_i-c_{p_i}\|^2, \tag{14}$$

where $P=\{p_1,p_2,\ldots,p_N\}$ defines the mapping from the input vector x_i to its nearest code vector in the codebook.

There is a clear relationship between the MSE and the criterion used in Otsu's method in Sec. 2.1. Otsu's method maximizes the between-class variance of Eq. (4). It is known that the total variance of the vectors is a sum of between-class and within-class variances:[24]

$$\sigma^2=\sigma_B^2+\sigma_W^2=\sum_{k=1}^{M}\omega_k(\mu_k-\mu_T)^2+\sum_{k=1}^{M}\omega_k\sigma_k^2. \tag{15}$$

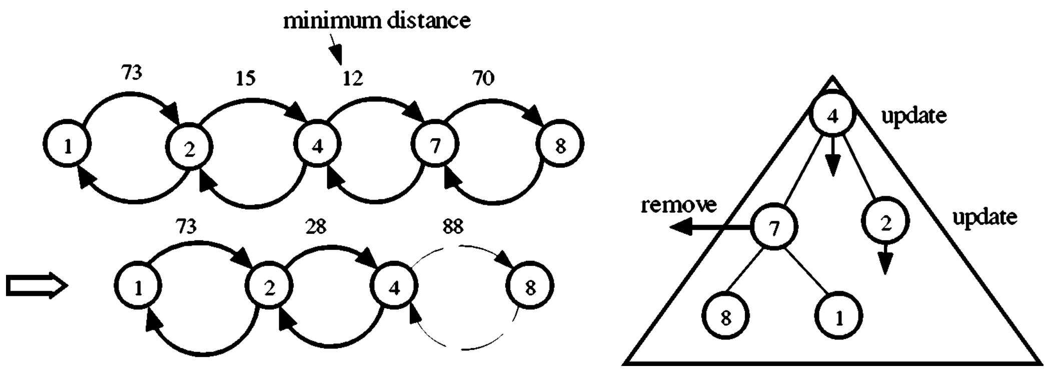

Fig. 5 Changes in the data structures due to the threshold removal.

Index	Index of the previous cluster	Index of the next cluster	Mean gray level	Maximal gray level	Distance to the next cluster	Number of pixels
i	$prev_i$	$next_i$	c_i	t_i	d_i	n_i

Fig. 6 The data structure of the linked list.

Thus, maximizing between-class variance is equivalent to minimizing within-class variance. As the MSE is equivalent to the within-class variance, we can conclude that Otsu's method also minimizes the MSE.

The problem of finding the codebook for vector quantization can be considered as a clustering problem.[25] In general, clustering aims at partitioning the dataset into M clusters, such that similar vectors are grouped together and dissimilar vectors are sent to different groups. A cluster s_a is defined as the set of data vectors that belong to the same partition a:

$$s_a = \{x_i | p_i = a\}. \tag{16}$$

The clustering is then represented as the set $S = \{s_1, s_2, \ldots, s_M\}$. In vector quantization, the codebook is taken as the average vectors (centroids) of the clusters.

The problem in finding the optimal codebook is known to be NP complete.[26] In other words, there is no known polynomial time algorithm for finding the optimal solution. In principle, the optimal codebook can be solved by using a branch-and-bound technique, as proposed in Ref. 27. The method, however, has exponential time complexity and is applicable for small clustering problems only. Therefore, faster but suboptimal methods, such as the generalized Lloyd algorithm (GLA)[28] and the PNN,[18] are used for generating the codebook in vector quantization.

3.2 *PNN Method*

The pairwise nearest neighbor (PNN) method generates the clustering hierarchically using a sequence of merge operations as described in Fig. 3.[18,29] In each step of the algorithm, the number of clusters is reduced by merging two nearby clusters.

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:[18,29]

$$d_{a,b} = \frac{n_a n_b}{n_a + n_b} \cdot \|c_a - c_b\|^2, \tag{17}$$

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

```
PNNThresholding(X, x_max, M) → T

S, H ← Initialize(X, x_max);
m ← x_max;
REPEAT
    (s_a,s_b) ← SearchMinDistFromHeap(H);
    S ← Merge(S, s_a, s_b);
    S, H ← UpdateDataStruct(S, s_prev_a, s_a, s_b, H);
    m ← m−1;
UNTIL m = M;

Initialize(X, N) → S, H
  FOR i ←1 to N
      prev_i ← i-1; next_i ← i+1;
      c_i ← i; t_i ← i;
      Calculate d_i;
      s_i ← { prev_i, next_i, c_i, t_i, d_i, x_i };
      Insert s_i into the heap;
  END-FOR
  RETURN S, H;

Merge(S, s_a, s_b) → S
  n_q ← n_a + n_b;
  c_q ← (n_a c_a + n_b c_b) / n_q;
  t_q ← t_b;
  s_a ← s_q;
  RETURN S;

UpdateDataStruct(S, s_prev_a, s_a, s_b, H) → (S, H)
  next_a ← next_b;
  Set the previous neighbor cluster for next_b to a;
  Recalculate distance of the next neighbor
  cluster for a and prev_a;
  Remove s_b from the heap;
  Shift down clusters a and prev_a in the heap;
  RETURN S, H;
```

Fig. 7 The proposed fast PNN-based algorithm for thresholding.

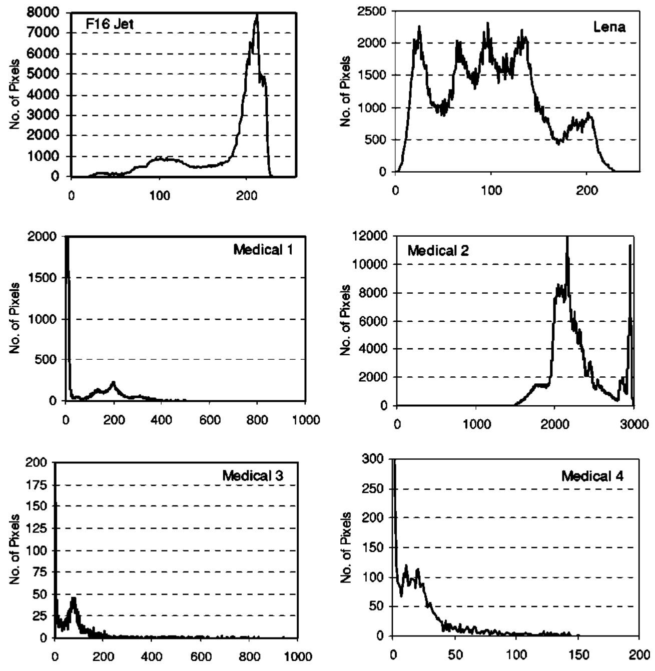

Fig. 8 Histograms of the test images.

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.[21] The cluster pair to be merged can be found in $O(N)$ time, and only a small number (denoted by τ) of the nearest neighbors need to be updated after each merge. The implementation takes $O(\tau N^2)$ time in total. Further speed-up can be achieved by using a lazy update of the merge cost values,[30] and by reducing the amount of work caused by the distance calculations.[31] Kurita's method[20] stores all pairwise distances into the heap structure and it requires $O(N^2 \log N)$ time originating from the update of the heap structure.

3.3 *Adapting PNN to Thresholding*

The PNN can be adapted to multilevel thresholding by implementing the following three steps: 1. find the threshold to be removed, 2. remove the threshold, and 3. update the class parameters. The first step is similar to that of the vector quantization as it considers all possible thresholds to be removed, and then selects the one that increases the MSE least. The difference is that we find a threshold to be removed instead of a cluster pair to be merged. This simplifies the process remarkably, as there are only $O(N)$ thresholds to be considered, whereas in vector quantization, there are $O(N^2)$ cluster pairs in total.

The second step corresponds to the merge of the two clusters. In thresholding, the merge is performed by removing the selected threshold (see Fig. 4). This step is simple both in vector quantization and in thresholding. In vector quantization, the new cluster centroid can be calculated on the basis of the centroids of the merged clusters. In thresholding, the step consists merely of the removal of the class and the corresponding threshold from a list structure.

The third step consists of the update of the respective data structures. In vector quantization, this corresponds to the recalculation of the merge cost values for certain clusters. The variants in Refs. 19 and 20 store all pairwise merge cost values in a 2-D table of size $O(N^2)$, which requires $O(N)$ amount of updates. The variant in Ref. 21 stores only the nearest neighbor pointers and the corresponding cost values for every cluster. The memory requirement of this variant is more realistic for practical implementation, but the amount of necessary updates is somewhat higher. In thresholding, on the other hand, memory consumption is not a problem, as we have only 1-D data space. We can therefore apply any of the earlier variants without problems.

Table 1 Time complexities of the different PNN variants.

	Vector quantization		Thresholding:
	Heap-PNN:	Fast PNN:	PNN:
Initialization:	$O(N^2)$	$O(N^2)$	$O(N \log N)$
Single step:			
● Cluster selection	$O(1)$	$O(N)$	$O(1)$
● Merge/removal	$O(1)$	$O(1)$	$O(1)$
● Update	$O(N \log N)$	$O(\tau N)$	$O(\log N)$
Algorithm in total:	$O(N^2 \log N)$	$O(\tau N^2)$	$O(N \log N)$

Table 2 Time complexities of the Lloyd-Max quantizer[17] and the GLA.[28]

	Lloyd-Max quantizer	GLA
Partition step	$O(M)$	$O(NM)$
Codebook step	$O(N)$	$O(N)$
Whole iteration step	$O(N)$	$O(NM)$

3.4 *Data Structures*

In thresholding, the situation is significantly different from vector quantization in a sense that every cluster has only one neighbor. This will effect the choice of the data structures for two reasons: 1. the amount of memory is never more than $O(N)$ even if we store all cost values, and 2. the number of necessary updates is always a constant.

We maintain a linked list for the class parameters, as shown in Fig. 5. Each class in the list consists of six values: $(\text{prev}_i, \text{next}_i, c_i, t_i, d_i, n_i)$ as shown in Fig. 6. The first two

Table 3 Test images.

Image	Min:	Max:	Difference:	Resolution:	Bpp:
F16 Jet	16	231	216	512×512	8
Lena	3	248	236	512×512	8
Medical 1	0	999	633	256×256	12
Medical 2	200	2998	2068	2392×1792	12
Medical 3	0	940	314	64×64	16
Medical 4	0	179	152	64×64	16

Table 4 Thresholds for the test image F16 Jet.

	Number of thresholds ($M-1$):			
Method:	1	2	3	4
Otsu	152	112, 172	91, 143, 189	83, 128, 171, 202
PNN+LMQ	152	120, 178	90, 141, 188	86, 133, 176, 204
LMQ	152	111, 172	90, 141, 188	80, 122, 167, 201

Table 5 Thresholds for the test image Lena.

	Number of thresholds ($M-1$):			
Method:	1	2	3	4
Otsu	101	77, 145	56, 106, 159	46, 83, 119, 164
PNN+LMQ	103	78, 145	57, 108, 161	47, 84, 120, 164
LMQ	103	80, 148	58, 108, 161	50, 89, 124, 167

are pointers to the neighbor classes to the left and to the right, and c_i is the mean gray-level value of the class. t_i is the maximal gray level, and it also serves as the threshold value between the class and its neighbor class to the right. The pointer to the right is augmented also by the merge cost value (d_i), indicating the increase in the MSE if the two classes are merged. The last value (n_i) is the number of pixels in the class.

The cost values (d_i) are stored also in a heap structure (see Fig. 5). The heap is used only as a search structure. Thus, it includes merely pointers to the linked list. From the linked list, we also have pointers to the heap structure to locate the elements in the heap when the corresponding cost values are updated.

The pseudocode of the proposed PNN-based thresholding algorithm is outlined in Fig. 7. In the beginning of the algorithm, every histogram value i is assigned to its own class. The corresponding mean (c_i) and maximal (t_i) gray levels are set to i for every class i. We thus have N classes to start with. The following steps of the algorithm are then repeated until we reach M classes ($M-1$ thresholds). We first pop the smallest element from the heap and get the corresponding class element from the list. For example, class 4 has the cost value of 12 in Fig. 5. It is then merged with the following class element by removing the latter one from the linked list. New cost values are then calculated for the class (class 4 in the example), and for its preceding class (class 2 in the example). Their locations in the heap are also updated.

3.5 *Complexity Analysis*

The initialization requires $O(N \log N)$ time, as there are N elements for which we must calculate the cost value and insert into the heap.

In the first step of the iteration, we find the removed threshold. This is equal to the minimum-finding problem of the heap, and it can be implemented in $O(1)$ time using the heap structure.

In the second step, we merge the two classes. We calculate a new mean pixel value of the merged class and put it into the place of the first of the two classes in the linked list. Then we remove the second class. These calculations can be performed in $O(1)$ time.

In the third step, we update the data structures. The merge cost value must be recalculated only for the merged class and its preceding class. Their new location in the heap is found by sinking them down in the heap, as the values can only increase. This requires $O(\log N)$ time in total. The removal of the obsolete cluster from the heap requires also $O(\log N)$ time. The update of the list structure is straightforward and it takes only $O(1)$ time.

In total, there are $(N-M)$ iterations to be performed. For each iteration, the three steps together take $O(1+1+\log N)=O(\log N)$ time. Thus, the proposed algorithm takes $O[(N-M)\log N]=O(N\log N-M\log N)=O(N\log N)$ time in total, assuming that $M \ll N$. The time complexities of the PNN method both in vector quantization and in thresholding are summarized in Table 1. The time complexities of the LMQ (for thresholding) and the GLA (for vector quantization) are shown in Table 2 for comparison.

4 Experiments

For evaluating the proposed method, we use six test images: F16 Jet, Lena, Medical 1, Medical 2, Medical 3, and Medical 4. The first two are well known photographic images, and the other four are medical images of various sources (magnetic resonance, computer radiology, and

Table 6 MSE values and run times for the test image F16 Jet.

	MSE					Run time (seconds)				
M	UQ	PNN	PNN+ LMQ	LMQ	Otsu	PNN	PNN+ LMQ	LMQ	Otsu	Fast Otsu
2	501	378	362	362	362	0.00	0.00	0.00	0.00	0.01
3	227	248	208	206	206	0.00	0.00	0.00	0.22	0.01
4	144	150	130	130	129	0.00	0.00	0.00	17	0.12
5	98.4	94.4	85.2	85.5	84.1	0.00	0.00	0.00	966	10
10	29.4	23.6	21.1	25.4	···	0.00	0.00	0.00	···	···
15	14.3	10.5	10.1	10.2	···	0.00	0.00	0.00	···	···
20	9.5	6.3	5.9	7.2	···	0.00	0.00	0.00	···	···

Table 7 MSE values and run times for the test image Lena.

	MSE					Run time (seconds)				
M	UQ	PNN	PNN+ LMQ	LMQ	Otsu	PNN	PNN+ LMQ	LMQ	Otsu	Fast Otsu
2	1021	891	877	877	876	0.00	0.00	0.00	0.00	0.01
3	442	502	403	404	402	0.00	0.00	0.00	0.27	0.01
4	275	214	199	199	198	0.00	0.00	0.00	22	0.23
5	174	151	121	124	120	0.00	0.00	0.00	1378	15
10	45.2	38.3	33.8	39.5	···	0.00	0.00	0.00	···	···
15	21.0	18.9	16.4	19.7	···	0.00	0.00	0.00	···	···
20	12.2	10.8	9.5	12.1	···	0.00	0.00	0.00	···	···

Table 8 MSE values and run times for the test image Medical 1.

	MSE					Run time (seconds)				
M	UQ	PNN	PNN+ LMQ	LMQ	Otsu	PNN	PNN+ LMQ	LMQ	Otsu	Fast Otsu
2	11256	2679	2518	2518	2518	0.01	0.01	0.00	0.03	0.11
3	8934	1040	1013	1013	1012	0.01	0.01	0.00	16	0.17
4	5950	648	627	716	627	0.01	0.01	0.01	5514	22
5	3872	421	380	380	380	0.01	0.01	0.01	···	5788
10	454	137	112	141	···	0.01	0.01	0.02	···	···
15	185	62.1	57.1	82.5	···	0.01	0.01	0.03	···	···
20	108	35.4	32.3	73.1	···	0.01	0.01	0.01	···	···

Table 9 MSE values and run times for the test image Medical 2.

	MSE					Run time (seconds)				
M	UQ	PNN	PNN+ LMQ	LMQ	Otsu	PNN	PNN+ LMQ	LMQ	Otsu	Fast Otsu
2	97360	32713	32110	32109	32109	0.02	0.03	0.01	0.30	0.98
3	60520	16725	15985	31411	15983	0.02	0.02	0.01	445	1.43
4	33332	10487	8675	15397	8674	0.02	0.03	0.01	···	491
5	19590	6480	5368	8233	···	0.02	0.04	0.02	···	···
10	5116	1785	1614	1982	···	0.02	0.03	0.06	···	···
15	2467	797	644	1164	···	0.02	0.03	0.05	···	···
20	1564	434	376	493	···	0.02	0.03	0.11	···	···

Table 10 MSE values and run times for the test image Medical 3.

	MSE					Run time (seconds)				
M	UQ	PNN	PNN+ LMQ	LMQ	Otsu	PNN	PNN+ LMQ	LMQ	Otsu	Fast Otsu
2	3830	3272	3163	3163	3163	0.00	0.01	0.00	0.03	0.10
3	2921	1482	1424	1425	1424	0.00	0.01	0.01	14	0.16
4	2642	782	754	1063	753	0.00	0.01	0.00	4368	20
5	2147	569	500	500	499	0.00	0.01	0.01	···	5061
10	993	140	124	220	···	0.00	0.01	0.01	···	···
15	352	61.9	59.8	130	···	0.00	0.01	0.01	···	···
20	182	34.2	32.4	94.1	···	0.00	0.01	0.01	···	···

Table 11 MSE values and run times for the test image Medical 4.

	MSE					Run time (seconds)				
M	UQ	PNN	PNN+LMQ	LMQ	Otsu	PNN	PNN+LMQ	LMQ	Otsu	Fast Otsu
2	352	237	237	237	237	0.00	0.00	0.00	0.00	0.00
3	198	130	127	127	124	0.00	0.00	0.00	0.10	0.00
4	134	69.8	64.4	66.8	63.8	0.00	0.00	0.00	6	0.07
5	103	46.5	42.9	45.3	42.1	0.00	0.00	0.00	275	3
10	29.0	12.3	11.3	22.0	···	0.00	0.00	0.00	···	···
15	12.9	5.5	5.4	12.9	···	0.00	0.00	0.00	···	···
20	6.8	2.9	3.0	6.8	···	0.00	0.00	0.00	···	···

nuclear medicine).[32] The characteristics of the test images are summarized in Table 3, and their histograms are shown in Fig. 8. All tests are made on a 450-MHz Pentium III computer.

4.1 *Methods in Comparison*

We apply the algorithms for all test images by varying the number of thresholds (from $M=2$ to 20). We use the following methods:

- uniform quantizer (UQ)
- LMQ
- PNN
- PNN+LMQ
- Otsu's method (optimal).

A uniform quantizer distributes the thresholds equally, and it serves as a point of comparison. It is expected that the studied algorithms perform better than the uniform quantizer. The LMQ is the iterative algorithm from Sec. 2.2. We use the output of the uniform quantizer as the input for the LMQ. The PNN is the algorithm proposed in this work, and the PNN+LMQ is the result of the PNN after it is iterated by the LMQ. Otsu's method serves as another point of comparison, since it provides the optimal result in the sense of minimizing MSE.

The threshold values for the F16 Jet and Lena are summarized in Tables 4 and 5. The MSE values and the run times for the test images are reported in Tables 6 through 11. The number of iterations required by the LMQ and the PNN+LMQ are shown in Table 12.

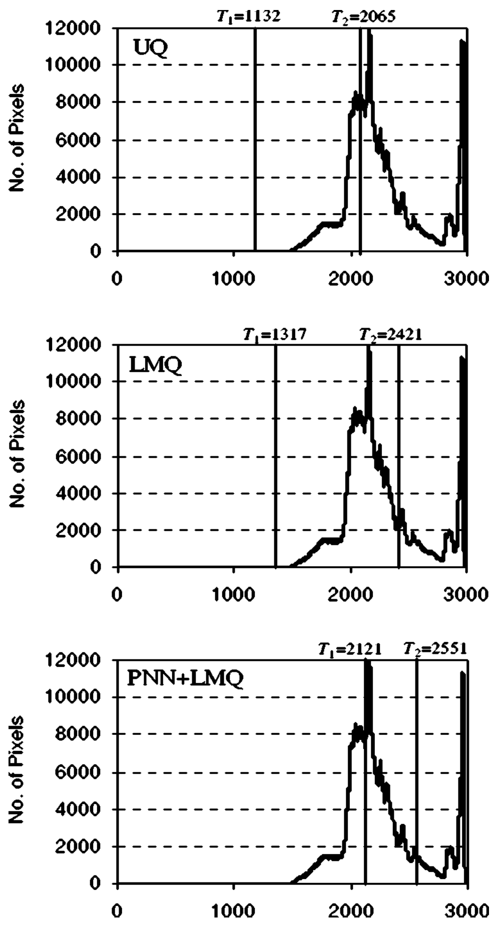

Fig. 9 Illustration of the thresholding of Medical 2 for $M=3$ segments.

4.2 *Quality Comparison*

From the results, we make the following observations. All methods provide significantly lower MSE values than the uniform quantizer. In the case of a small number of thresh-

Table 12 The number of iterations.

	F 16 Jet		Medical 1		Medical 3	
M	PNN+LMQ	LMQ	PNN+LMQ	LMQ	PNN+LMQ	LMQ
2	4	5	4	11	7	8
3	11	11	4	13	4	27
4	7	11	9	28	10	19
5	5	16	9	31	12	25
10	6	5	16	91	14	31
15	4	16	20	112	4	32
20	4	5	7	54	4	30

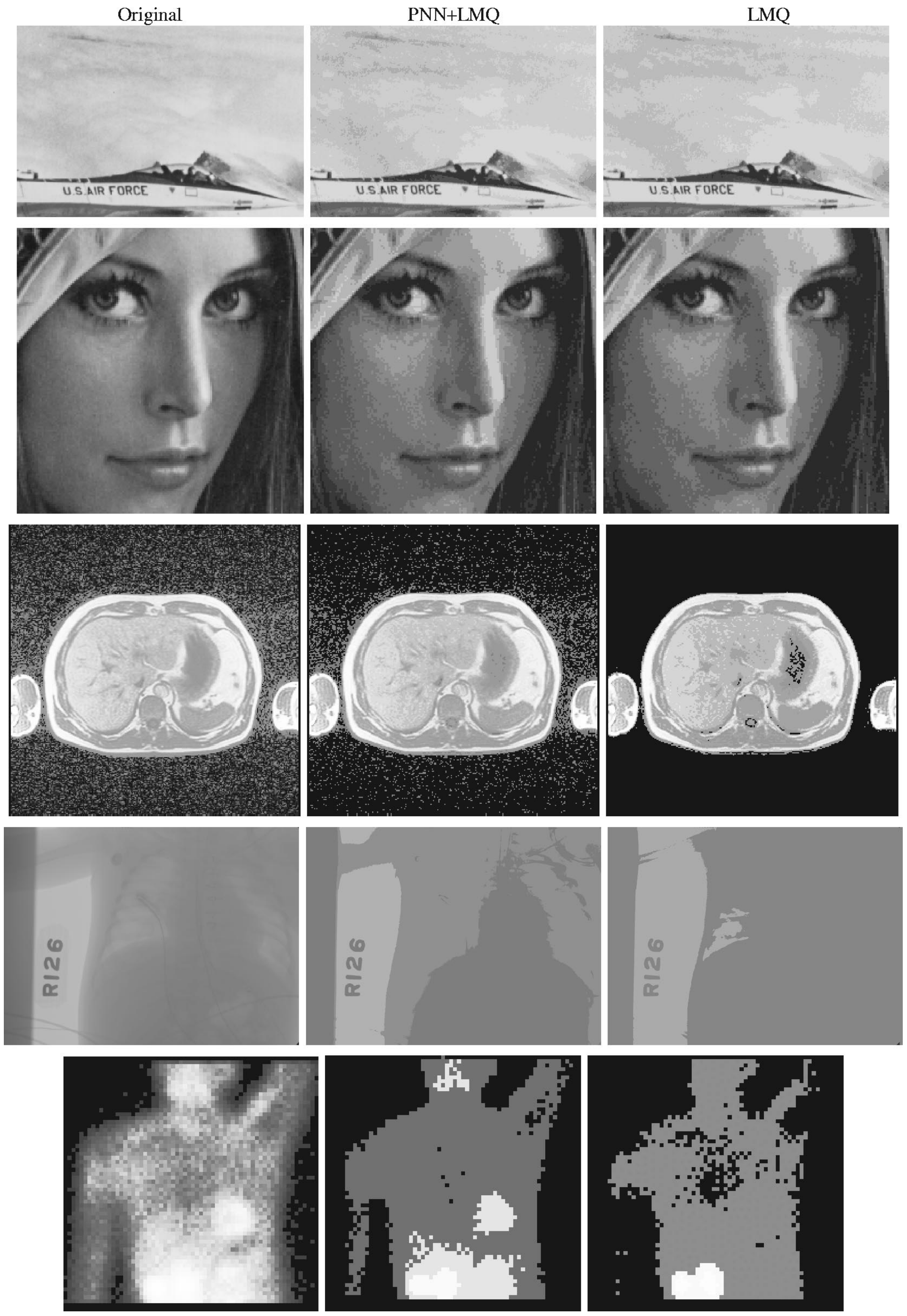

Fig. 10 Sample thresholded images from top down: F16 Jet ($M=10$), Lena ($M=10$), Medical 1 ($M=20$), Medical 2 ($M=3$), and Medical 3 ($M=4$).

olds ($M=2$ and 3), the results are optimal or very close to optimal. For a large number of thresholds, the optimal result is not always found, and there are significant differences in the performance between the methods.

The comparison between LMQ and PNN is two-fold. For small values of M, the LMQ gives lower MSE values, whereas the PNN is better with large values of M. For example, for Lena with $M=3$, the MSE is 502 by the PNN, and 404 by the LMQ (Table 7). On the other hand, for Medical 1 with $M=20$, the MSE is 35.4 by the PNN, and 73.1 by the LMQ (Table 8). We therefore recommend combining the PNN and the LMQ methods and use PNN+LMQ instead of applying the methods as themselves.

The PNN+LMQ combination gives near-optimal results in all cases reported here. Even in the worst case, it reaches the MSE value only 2% away from the optimum; 42.9 by the PNN+LMQ, and 42.1 by the optimal method for Medical 4 (Table 11). The reduction in the MSE, on the other hand, can be as much as 66% from that of the LMQ: MSE values 32.4 versus 94.1 for Medical 3 (Table 10). This corresponds to 4.63-dB improvement in the peak signal-to-noise ratio (PSNR).

Sometimes the deficiency of using LMQ alone appears also with a small value of *M*, as demonstrated in Fig. 9. There is a very low, almost invisible peak, around the values 200 to 220 in the histogram. The LMQ starts with the output of the uniform quantizer and manages to relocate the second threshold ($T_2=2421$), but fails to optimize the first threshold ($T_1=1317$). The PNN+LMQ, on the other hand, finds significantly better solution ($T_1=2121$ and $T_2=2551$).

Visual comparisons of the LMQ and PNN+LMQ are shown in Fig. 10. The differences in quality in F16 Jet and Lena are not so clearly visible, but there are few: the clouds in the F16 Jet are somewhat more natural, and the cheek of Lena is slightly smoother in the image thresholded by the PNN+LMQ. With the medical images the differences are more clearly visible, as can be seen in Medical 2 and Medical 3 even for values as low as $M=3$ and $M=4$. With larger values of *M*, the results favor PNN+LMQ. The example with Medical 1 shows that sometimes worse optimization of the LMQ can have surprising side effects in the form of removal of background noise. Nevertheless, the PNN+LMQ still provides better approximation of the original image, despite the preservation of noise.

The significance of the improvement in quality can also be estimated by calculating the amount of additive noise that would have provided the same MSE value as the improvement from LMQ to PNN+LMQ. The examples of Fig. 10 (F16 Jet, Lena, Medical 1, Medical 2, and Medical 3) correspond to the noise level of (1, 1, 0.5, 4, and 2%) on average. The corresponding PSNR values are (0.81, 0.68, 1.60, 2.93, and 1.49 dB), and the maximum obtained improvement was measured as 4.63 dB for Medical 3 with $M=20$.

4.3 *Run Time Comparison*

Otsu's method is applicable only for small values of *M* because it is too slow in the case of $M>5$. The fast variant of Otsu's method is remarkably faster; 15 s versus 1378 for Lena with $M=5$ (Table 7). The speed-up, however, is not asymptotic and does not help for larger values of *M*. For example, the run times of Otsu's method and its fast variants are 5514 and 22 s for Medical 1 with $M=4$ (Table 10). Despite the remarkable difference, the run time of the fast variant is already 5788 s when $M=5$.

All suboptimal methods are fast in all cases. The highest measured run time is only about 0.11 s by the LMQ for Medical 2 (Table 9). The time complexity of a single iteration of the LMQ is faster than that of the PNN. The LMQ, however, is iterated several times, and the overall run times are similar. With higher values of *M*, the PNN has a moderate time advantage, although both methods are sufficiently fast for real-time applications.

It is also noted that the combination of PNN+LMQ could be even faster than the LMQ alone. This is because the PNN can provide better initial thresholding, and thus, the LMQ uses less iterations (Table 12). We would also like to note that the run times are so small that the implementation details and system level details, such as memory caching, can affect the time measurement. We therefore cannot make any other conclusions between PNN, LMQ, and PNN+LMQ other than that they are all suitable for real-time image processing.

5 Conclusions

A fast PNN-based multilevel thresholding algorithm is proposed. The time complexity of the method is $O(N \log N)$, which is significantly better than that of optimal thresholding. In practice, the proposed method works in real time for any number of thresholds. Experiments also show that the proposed method, when combined with the LMQ, provides MSE values that are much closer to that of optimal thresholding than using LMQ alone. The difference is small when a low number of thresholds are needed ($M=2$ or 3), but the difference is significant when the number of thresholds is higher (from $M=10$ to $M=20$).

References

1. D. M. Tsai and Y. H. Chen, "A fast histogram-clustering approach for multilevel thresholding," *Pattern Recogn. Lett.* **13**(4), 245–252 (1992).
2. N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst. Man Cybern.* **9**(1), 62–66 (1979).
3. S. Wang and R. Haralick, "Automatic multithresholding selection," *Comput. Vis. Graph. Image Process.* **25**, 46–67 (1984).
4. J. C. Yen, F. J. Chang, and S. Chang, "A new criterion for automatic multilevel thresholding," *IEEE Trans. Image Process.* **4**(3), 370–378 (1995).
5. W. H. Tsai, "Moment-preserving thresholding: A new approach," *Comput. Vis. Graph. Image Process.* **29**, 377–393 (1985).
6. P. K. Sahoo, S. Soltani, A. K. C. Wong, and Y. Chen, "A survey of thresholding techniques," *Comput. Vis. Graph. Image Process.* **41**, 233–260 (1988).
7. T. Pun, "A new method for gray-level picture thresholding using the entropy of the histogram," *Signal Process.* **2**, 223–237 (1980).
8. J. N. Kapur, P. K. Sahoo, and A. K. C. Wong, "A new method for gray-level picture thresholding using the entropy of the histogram," *Comput. Vis. Graph. Image Process.* **29**(3), 273–285 (1985).
9. S. U. Lee and S. Y. Shung, "A comparative performance study of several global thresholding techniques for segmentation," *Comput. Vis. Graph. Image Process.* **52**, 171–190 (1990).
10. P. S. Liao, T. S. Chen, and P. C. Chung, "A fast algorithm for multilevel thresholding," *J. Comput. Inf. Sci. Eng.* **17**, 713–727 (2001).
11. R. L. Kirby and A. Rosenfeld, "A note on the use of (gray, local average gray level) space as an aid in thresholding selection," *IEEE Trans. Syst. Man Cybern.* **9**(12), 860–864 (1979).
12. A. S. Abutaleb, "Automatic thresholding of gray-level pictures using two-entropy," *Comput. Vis. Graph. Image Process.* **47**(1), 22–32 (1989).
13. N. R. Pal and S. K. Pal, "Entropic thresholding," *Signal Process.* **16**(2), 97–108 (1989).
14. A. D. Brink, "Thresholding of digital images using two-dimensional entropies," *Pattern Recogn.* **25**(8), 803–808 (1992).
15. W. T. Chen, C. H. Wen, and C. W. Yang, "A fast two-dimensional entropic thresholding algorithm," *Pattern Recogn.* **27**(7), 885–893 (1994).
16. J. Gong, L. Li, and W. Chen, "Fast recursive algorithms for two-dimensional thresholding," *Pattern Recogn.* **31**(3), 295–300 (1998).
17. S. P. Lloyd, "Least squares quantization in PMC," *IEEE Trans. Inf. Theory* **28**(2), 129–137 (1982).
18. W. H. Equitz, "A new vector quantization clustering algorithm,"

IEEE Trans. Acoust., Speech, Signal Process. **37**(10), 1568–1575 (1989).
19. 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).
20. T. Kurita, ''An efficient agglomerative clustering algorithm using a heap,'' *Pattern Recogn.* **24**(3), 205–209 (1991).
21. 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).
22. J. Lukaszewicz and H. Steinhaus, ''On measuring by comparison,'' *Zastos. Mat.* **2**, 225–232 (1955).
23. A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Dordrecht (1992).
24. L. Shapiro and G. Stockman, *Computer Vision*, pp. 83–91, Prentice Hall, Englewood Cliffs, NJ (2001).
25. A. K. Jain, M. N. Murty, and P. J. Flynn, ''Data clustering: A review,'' *ACM Comput. Surv.* **31**(3), 264–323 (1999).
26. M. R. Garey, D. S. Johnson, and H. S. Witsenhausen, ''The complexity of the generalized Lloyd-Max problem,'' *IEEE Trans. Inf. Theory* **28**(2), 255–256 (1982).
27. P. Fränti, O. Virmajoki, and T. Kaukoranta, ''Branch-and-bound technique for solving optimal clustering,'' *Intl. Conf. Patt. Recog. (ICPR'02)*, Quebec, Canada, Vol. 2, pp. 232–235 (Aug. 2002).
28. Y. Linde, A. Buzo, and R. M. Gray, ''An algorithm for vector quantizer design,'' *IEEE Trans. Commun.* **28**(1), 84–95 (1980).
29. J. H. Ward, ''Hierarchical grouping to optimize an objective function,'' *J. Am. Stat. Assoc.* **58**, 236–244 (1963).
30. T. Kaukoranta, P. Fränti, and O. Nevalainen, ''Vector quantization by lazy pairwise nearest neighbor method,'' *Opt. Eng.* **38**(11), 1862–1868 (1999).
31. O. Virmajoki, P. Fränti, and T. Kaukoranta, ''Practical methods for speeding-up the pairwise nearest neighbor method,'' *Opt. Eng.* **40**(11), 2495–2504 (2001).
32. J. Kivijärvi, T. Ojala, T. Kaukoranta, A. Kuba, L. Nyúl, and O. Nevalainen, ''A comparison of lossless compression methods for medical images,'' *Comput. Med. Imaging Graph.* **22**(4), 323–339 (1998).

Olli Virmajoki received his MSc degree in electrical engineering from the Helsinki University of Technology, Finland, in 1983, and PhLic degree in computer science from the University of Joensuu, in 2002. 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 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.