Framework for efficient optimal multilevel image thresholding

Martin Luessi
Northwestern University
Department of Electrical Engineering and Computer Science
2145 Sheridan Road
Evanston, Illinois 60208-3118
E-mail: mlu315@eecs.northwestern.edu

Marco Eichmann
Guido M. Schuster
University of Applied Sciences of Eastern Switzerland
Oberseestrasse 10
8640 Rapperswil
Switzerland

Aggelos K. Katsaggelos
Northwestern University
Department of Electrical Engineering and Computer Science
2145 Sheridan Road
Evanston, Illinois 60208-3118

Abstract. *Image thresholding is a very common image processing operation, since almost all image processing schemes need some sort of separation of the pixels into different classes. In order to determine the thresholds, most methods analyze the histogram of the image. The optimal thresholds are often found by either minimizing or maximizing an objective function with respect to the values of the thresholds. By defining two classes of objective functions for which the optimal thresholds can be found by efficient algorithms, this paper provides a framework for determining the solution approach for current and future multilevel thresholding algorithms. We show, for example, that the method proposed by Otsu and other well-known methods have objective functions belonging to these classes. By implementing the algorithms in ANSI C and comparing their execution times, we can also make quantitative statements about their performance.* © 2009 SPIE and IS&T.
[DOI: 10.1117/1.3073891]

1 Introduction

In many image processing applications, the pixels need to be classified as belonging to the foreground or the background. The use of a threshold can often accomplish such a task. In multilevel image thresholding, pixels can be classified into many classes, not just foreground and background. Because of its importance, image thresholding has attracted a considerable amount of attention. In Ref. 1, an extensive taxonomy and comparison of proposed methods is provided. Many methods define the optimal thresholds as the ones that maximize or minimize an objective function.

One method to find the thresholds is exhaustive search, which requires calculating the objective function for every possible placement of the thresholds. Clearly, the problem with this approach is that when the image is segmented into more than two classes, the time needed to find the optimal thresholds increases dramatically with the number of gray levels and the number of classes.

In this paper, we provide a framework for determining the solution approach for current and future multilevel thresholding algorithms. More specifically, we define two classes of objective functions for which the optimal thresholds can be found by efficient algorithms. For the first class, a dynamic programming (DP) algorithm, which has a significantly lower time complexity than exhaustive search, can be employed for finding the thresholds. While this algorithm has been proposed in Ref. 2 for the thresholding method by Otsu[3] and in Ref. 4 for the method of Kittler and Illingworth,[5] we show that this DP algorithm can also be employed for finding the optimal thresholds for the maximum entropy method by Kapur *et al.*[6] For the second class, we prove a theorem based on which more efficient algorithms can be used for finding the optimal thresholds. These algorithms consist of a combination of DP and fast matrix searching that results in algorithms that have lower time complexities than the DP algorithm and can be several orders of magnitude faster. We show that the objective functions of the method proposed by Otsu[3] and the multilevel thresholding extension of the method by Li and Lee are

Paper 08096R received Jun. 22, 2008; revised manuscript received Nov. 10, 2008; accepted for publication Dec. 3, 2008; published online Feb. 2, 2009.

1017-9909/2009/18(1)/013004/10/$25.00 © 2009 SPIE and IS&T.

members of the second class. Consequently, we can find the optimal thresholds for these methods much faster than the previously proposed DP algorithm in Ref. 2. (The multilevel extension of Ref. 7 and its solution are proposed here for the first time.)

To verify the efficiency of the algorithms, execution time measurements of ANSI C implementations are presented. Note that the scope of this paper is not to analyze the segmentation performance of different thresholding algorithms. We do not analyze the segmentation performance since the proposed algorithms provide the same optimal results as the original methods, simply much faster.

An earlier version of this work was presented in Ref. 8. This paper is more comprehensive and contains new material that has not been shown before. For example, in Sec. 3, we show that the objective function of the Kapur method[6] belongs to the first class of objective functions; in Sec. 5, we analyze how the structure of the histogram influences the execution time; and in the Appendix, we show the proof of Theorem 1.

The paper is organized as follows: In Sec. 2, multilevel image thresholding is introduced and the problem is mathematically defined. In Sec. 3, the first class of objective functions is defined and the DP algorithm that can be used for finding the optimal thresholds is presented. In Sec. 4, the second class of objective functions is defined and the algorithms combining DP and fast matrix searching are introduced. In Sec. 5, execution time measurements of the different algorithms are presented. Last, the paper is summarized and conclusions are drawn in Sec. 6.

2 Multilevel Image Thresholding

The pixels of a grayscale image are represented by L gray levels g from 1 to L. Multilevel image thresholding is the task of separating the pixels of the image into M classes $C_1 \ldots C_M$ based on their values, by selecting the integer thresholds $t_1 \ldots t_{M-1}$. Class C_k is defined as $C_k = \{g \mid t_{k-1} < g \leqslant t_k\}$, where $k \in \{i \mid 1 \leqslant i \leqslant M\}$ refers to the class number. The thresholds t_0 and t_M are defined to be equal to 0 and L, respectively.

For the selection of the thresholds, most methods employ the histogram of the image. The histogram $h(g)$ shows the number of occurrences of the gray level g in the image, where $\Sigma_{g=1}^{L} h(g) = N$, with N the total number of pixels in the image. The normalized histogram $p(g) = h(g)/N$ can be considered as an estimate of the probability mass function of the gray levels present in the image.

For each class, statistical properties such as the probability of the class (later called class weight), the mean, and the variance of the class can be calculated as follows:

$$w(t_{k-1}, t_k] = \sum_{i=t_{k-1}+1}^{t_k} p(i), \tag{1}$$

$$\mu(t_{k-1}, t_k] = \sum_{i=t_{k-1}+1}^{t_k} \frac{p(i) \cdot i}{w(t_{k-1}, t_k]}, \tag{2}$$

$$\sigma^2(t_{k-1}, t_k] = \sum_{i=t_{k-1}+1}^{t_k} \frac{p(i) \cdot \big(i - \mu(t_{k-1}, t_k]\big)^2}{w(t_{k-1}, t_k]}. \tag{3}$$

The thresholding methods considered in this paper result from the optimization of an objective function. The optimal thresholds are the ones that either minimize or maximize the objective function. For the sake of conciseness and without loss of generality, derivations are shown only for the case where an objective function $J_{M,L}$ (to be defined later) is maximized, i.e.,

$$[t_1^*, t_2^*, \ldots, t_{M-1}^*] = \arg\max\big(J_{M,L}(t_1, \ldots, t_{M-1})\big), \tag{4}$$

with the following ordering for the positions of the thresholds:

$$0 < t_1 < t_2 \ldots < t_{M-1} < L. \tag{5}$$

The straightforward approach for finding the optimal thresholds is an exhaustive search, i.e., evaluating the objective function for every possible combination of thresholds. However, the number of possible combinations Q is given by (it is assumed that $M \ll L$):

$$Q = \binom{L-1}{M-1} = \prod_{i=1}^{M-1} \frac{L-i}{M-i} \geqslant \left(\frac{L-1}{M-1}\right)^{M-1}. \tag{6}$$

Therefore, an algorithm based on exhaustive search is suitable only for small numbers of gray levels and classes; it quickly becomes unsuitable when the number of gray levels or classes is increased. For example, an ANSI C implementation of an exhaustive search algorithm for the Otsu method requires 42 ms for 128 gray levels and 4 classes. When the number of classes is increased to 5, the execution time increases to 1.8 s. For 256 gray levels, which is the most common number of gray levels for images, the execution time is 350 ms and 28 s for 4 and 5 classes, respectively. Note that this is several orders of magnitude slower than the algorithms introduced in this paper. Please refer to Sec. 5 for execution time measurements of the algorithms proposed in this paper and information about where the ANSI C source code (including exhaustive search algorithm) can be found.

3 Dynamic Programming Solution

In this section, we consider multilevel image thresholding algorithms that are based on dynamic programming (DP). We also show that DP can be employed for finding the optimal thresholds for the maximum entropy method in Ref. 6.

3.1 *General Case*

The required structure for the first class of objective functions that accept a DP solution (also known as the shortest path algorithm) for finding the optimal thresholds is the following:

$$J_{M,L}(t_1, \ldots, t_{M-1}) = \sum_{k=1}^{M} l(t_{k-1}, t_k], \tag{7}$$

where $l(t_{k-1}, t_k]$ is referred to as the class cost of class C_k. Hence, the class cost can depend only on its boundary values, namely, t_{k-1} and t_k. A partial sum up to gray level l for the first m classes is defined as

$$J_m(l) = \sum_{k=1}^{m} l(t_{k-1},t_k], \quad 1 \leqslant t_1 < t_2 < \ldots < t_{m-1} < l. \tag{8}$$

For every gray level l, a subproblem can be defined as finding the optimal thresholds that partition the interval $[1,l]$ into m classes. The optimal solution to the subproblem is given by

$$J_m^*(l) = \max_{1 \leqslant t_1 < \ldots < t_{m-1} < l} (J_m(l)). \tag{9}$$

By rewriting the optimal solution to the subproblem, the following recursive formulation is obtained:

$$\begin{aligned} J_m^*(l) &= \max_{1 \leqslant t_1 < \ldots < t_{m-1} < l} \left(\sum_{k=1}^{m} l(t_{k-1},t_k] \right), \\ &= \max_{1 \leqslant t_1 < \ldots < t_{m-1} < l} \left(\sum_{k=1}^{m-1} l(t_{k-1},t_k] + l(t_{m-1},l] \right), \\ &= \max_{m-1 \leqslant t_{m-1} < l} (J_{m-1}^*(t_{m-1}) + l(t_{m-1},l]). \end{aligned} \tag{10}$$

By setting $m=M$ and $l=L$, this equation provides the maximum of the overall problem. It is clear that if the thresholds of a subproblem are not chosen optimally (and therefore they are not maximizing the objective function for the subproblem), the overall objective function cannot attain its maximum value. By having this recursive formulation, the optimal thresholds can be found by the shortest path algorithm. Pseudocode for the algorithm is depicted in Algorithm 1:

Algorithm 1: DPSEARCH(Trellis, M, L)

```
Trellis[0,0].J := 0
for $m$ := 1 to $M$ do
  for $l$ := $m$ to $L-M-m$ do
    $J_{max}$ := $-\infty$
    for each node in Trellis[$m-1$, :] do
      $g$ := gray level of node
      $J$ := Trellis[$m-1$, $g$].J + $\ell(g,l]$
      if $J > J_{max}$ then
        Trellis[$m$, $l$].J := $J$
        Trellis[$m$, $l$].pos := $g$
        $J_{max}$ := $J$
      end if
    end for
  end for
end for
− − − Backtracking
$l$ := $L$
$m$ := $M$
while $m \geqslant 2$ do
  $t^*_{m-1}$ := Trellis[$m$, $l$].pos
  $l$ := $t^*_{m-1}$
  $m$ := $m-1$
end while
return [$t^*_1, t^*_2, \ldots, t^*_{M-1}$]
```

A trellis is employed to find the optimal thresholds. (A specific example for $M=4$, $L=8$ is shown in Fig. 1.) In the pseudocode, the trellis is represented by the data structure Trellis; a node at stage m and gray level l are accessed by Trellis$[m,l]$. The algorithm proceeds from the bottom of the trellis to the top and compares paths emerging from nodes one stage below and to the left of the current node. The maximal cost is stored in the node, and a back pointer is set to point to the node from which the optimal path emerges. When the algorithm arrives at the end node, the optimal thresholds are found by backtracking to the start node.

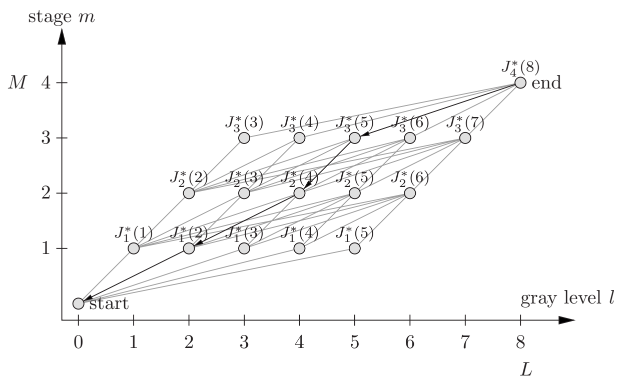

Fig. 1 Trellis for the shortest path algorithm for $M=4$, $L=8$.

Note that the amount of work the DP algorithm needs to perform in the first stage of the trellis is the same as the amount of work performed by an exhaustive search with $M=2$. Consequently, the DP algorithm is more efficient than an exhaustive search only when $M>2$. Since the number of stages is M, the number of nodes per stage equal to $(L-M+1)$, and the number of paths that have to be compared per node is proportional to L, the time complexity of this algorithm is $O(ML^2)$. This assumes that the class cost $l(t_{k-1},t_k]$ can be calculated in $O(1)$ time. This can be the case for many objective functions by introducing a preprocessing step that requires $O(L)$ time. (An example will be shown in Sec. 3.2). In Ref. 2, a dynamic programming scheme based on Eq. (10) has been proposed for the thresholding method by Otsu[3] and in Ref. 4 for the minimum error thresholding method by Kittler and Illingworth.[5]

The shortest path algorithm is also shown here because it forms the basis of more efficient algorithms, which are introduced in Sec. 4. Furthermore, we show that the DP algorithm can be employed for finding the optimal thresholds for the maximum entropy method by Kapur *et al.*,[6] as shown next.

3.2 *Maximum Entropy Method*

Numerous entropy-based thresholding methods have been proposed. (A good overview can be found in Ref. 1.) With the method proposed in Ref. 6, the classes are regarded as separate signal sources. The optimal thresholds maximize the sum of the class entropies, given by

$$J_{M,L}(t_1, \ldots, t_{M-1}) = \sum_{k=1}^{M} l(t_{k-1},t_k], \tag{11}$$

where the class cost $l(t_{k-1},t_k]$ is defined as

$$l(t_{k-1},t_k] = -\sum_{i=t_{k-1}+1}^{t_k} \frac{p(i)}{w(t_{k-1},t_k]} \log\left(\frac{p(i)}{w(t_{k-1},t_k]}\right). \qquad (12)$$

While the objective function for multilevel thresholding is proposed in Ref. 6, no method for selecting the thresholds maximizing the objective function is given. Clearly, the class cost depends only on t_{k-1} and t_k. Therefore, the DP algorithm can be used to find the optimal thresholds. However, the time complexity of the DP algorithm is $O(ML^2)$ only if the class cost can be calculated in $O(1)$ time. This is accomplished by precalculating and storing two arrays defined as

$$H(i) = \begin{cases} p(1)\cdot\log\big(p(1)\big) & \text{if } i=1, \\ H(i-1)+p(i)\cdot\log\big(p(i)\big) & \text{if } 2 \leqslant i \leqslant L, \end{cases} \qquad (13)$$

$$W(i) = \begin{cases} p(1) & \text{if } i=1, \\ W(i-1)+p(i) & \text{if } 2 \leqslant i \leqslant L. \end{cases} \qquad (14)$$

After the arrays have been precalculated, which requires $O(L)$ time, the class cost can be calculated as

$$l(t_{k-1},t_k] = \log\big(W(t_k)-W(t_{k-1})\big) - \frac{H(t_k)-H(t_{k-1})}{W(t_k)-W(t_{k-1})}. \qquad (15)$$

The time needed for this operation does not depend on t_{k-1} and t_k, which means that the time complexity is $O(1)$. Therefore, the optimal thresholds for the method proposed in Ref. 6 can be found in $O(ML^2)$ time using DP.

4 More Efficient Solutions

As shown in the previous section, it is possible for some multilevel thresholding methods to employ a DP solution with time complexity $O(ML^2)$ to find the optimal thresholds. However, the time needed to find the optimal thresholds still increases quadratically with the number of gray levels. In this section, we introduce solutions for finding the optimal thresholds more efficiently, given that the objective function has certain properties.

The problem of finding the optimal paths leading to all the nodes in one of the stages $2,\ldots,M-1$ of the trellis is equivalent to the problem of finding the row-wise maxima in a lower triangular $(L-M+1)\times(L-M+1)$ matrix $\boldsymbol{A}$. At stage m, the matrix is given by

$$\boldsymbol{A}(r,c) = \begin{cases} -\infty, & \text{if } c > r, \\ J^*_{m-1}(c+m-2)+l(c+m-2,r+m-1], & \text{if } c \leqslant r. \end{cases} \qquad (16)$$

In this matrix, the cost of the paths up to all the nodes of one stage in the trellis are treated as matrix elements, where the column c indicates the node from which the path emerges and the row r the node where the path ends. The elements above the main diagonal of the matrix are defined to be equal to $-\infty$, since there are no paths coming from nodes to the right or directly below the current node. We can therefore replace the two inner *for* loops of the DP algorithm with a matrix searching algorithm that finds the row-wise maxima in $\boldsymbol{A}$.

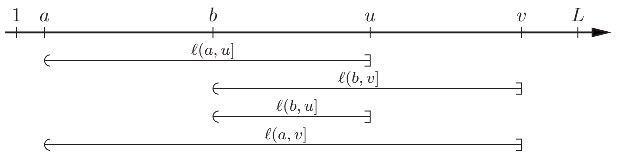

Fig. 2 Intervals of the convex quadrangle inequality.

If the matrix has no special properties, finding the row-wise maxima in the lower triangular region requires calculating all the elements in the region, which is exactly the same as the calculations performed in one stage of the DP algorithm. Depending on the objective function, the matrix can have properties that enable us to find the row-wise maxima without calculating all elements, which results in a reduced time complexity of the overall algorithm.

Assume that for some objective function, the class cost $l(t_{k-1},t_k]$ has the property

$$l(a,u]+l(b,v] \geqslant l(a,v]+l(b,u],$$

$$1 \leqslant a < b < u < v \leqslant L, \qquad (17)$$

which is known as convex quadrangle inequality. The intervals used are illustrated in Fig. 2. If the class cost has this property and we take four elements from the lower triangular region of $\boldsymbol{A}$ such that $1\leqslant r_1 < r_2 \leqslant L-M+1$ and $1 \leqslant c_1 < c_2 \leqslant r_1$, it follows from Eqs. (16) and (17) that

$$\boldsymbol{A}(r_1,c_1)+\boldsymbol{A}(r_2,c_2) \geqslant \boldsymbol{A}(r_1,c_2)+\boldsymbol{A}(r_2,c_1), \qquad (18)$$

which means that $\boldsymbol{A}$ is a lower triangular inverse Monge matrix. Monge matrices have a range of properties that can be exploited when performing combinatorial optimization; refer to Ref. 9 for a thorough discussion of this subject. For the problem at hand, the monotonicity of $\boldsymbol{A}$ can be exploited to find the maxima more efficiently. Assume that we know $\boldsymbol{A}(r_1,c_1)<\boldsymbol{A}(r_1,c_2)$; using Eq. (18), it is easy to show that this implies $\boldsymbol{A}(r_2,c_1)<\boldsymbol{A}(r_2,c_2)$, which means that, $\boldsymbol{A}$ is totally monotone.

The row-wise maxima of a totally monotone matrix can be found using matrix searching algorithms. Note that the matrix introduced here is a theoretical construct, and it is not necessary to calculate it before executing the matrix searching algorithm. (Doing so actually would not result in a reduced time complexity.) If a matrix searching algorithm needs a specific element of the matrix, it uses the cost stored in a node of the trellis and the objective function, as given by Eq. (16), to calculate it.

4.1 *Divide-and-Conquer Algorithm*

The divide-and-conquer algorithm exploits the fact that $\boldsymbol{A}$ is monotone, which means that

$$c_{\max}(r_1) \leqslant c_{\max}(r_2), \quad 1 \leqslant r_1 < r_2 \leqslant m, \qquad (19)$$

where $c_{\max}(r)$ denotes the column index of the leftmost element containing the maximum value of row r. That $\boldsymbol{A}$ is monotone follows directly from the total monotonicity. Assume that $c_{\max}(r_1) \geqslant c_{\max}(r_2)$ for some $1 \leqslant r_1 < r_2 \leqslant m$. The total monotonicity implies that

$$\boldsymbol{A}\big(r_2,c_{\max}(r_2)\big) < \boldsymbol{A}\big(r_2,c_{\max}(r_1)\big), \tag{20}$$

which contradicts the fact that $c_{\max}(r_2)$ is the column index of the maximum in row r_2.

When the divide-and-conquer algorithm is used to find the row-wise maxima of an $m \times n$ matrix $\boldsymbol{A}$, it first searches for the maximum in the middle row $r = \lceil m/2 \rceil$ of the matrix, and it is then recursively applied on the submatrices $\boldsymbol{A}(1,\ldots,r-1;1,\ldots,c_{\max}(r))$ and $\boldsymbol{A}(r+1,\ldots,m;c_{\max}(r),\ldots,n)$, as shown by the pseudocode in Algorithm 2. The algorithm is started by calling DIVCONQ($\boldsymbol{A}$,0), and it obtains a vector MAX containing the maxima positions. The time complexity of this algorithm can be found using the recursion tree method.[10] The worst-case execution time of the algorithm can be written as:

$$T(m,n) = \begin{cases} cn, & \text{if } m = 1, \\ 2T(m/2,n/2) + cn, & \text{if } m > 1, \end{cases} \tag{21}$$

where c is a constant time needed to evaluate an element of the matrix. As the recursion depth is $\log_2(m)$ and the accumulated execution time across all branches at each level of the tree is cn, the time complexity of the algorithm is $O(n \log m)$.

When this algorithm is combined with the DP algorithm, the divide-and-conquer algorithm is executed once for each stage $2,\ldots,M-1$ of the trellis. As the matrix has a size of $(L-M+1)\times(L-M+1)$ and we assume that the class cost can be calculated in $O(1)$ time, the resulting time complexity is $O(ML \log L)$.

Algorithm 2: DIVCONQ($\boldsymbol{M}$, *offset*)

```
[$m,n$]:= size of $\boldsymbol{M}$ {rows, columns}
$r$:= $\lceil m/2 \rceil$
$j$:= position leftmost maximum in row $r$ of $\boldsymbol{M}$
MAX[$offset+r$]:= $j$ {store position}
if m=1 then
  return
else
  if $r \neq 1$ then
    $\boldsymbol{P}$:= $\boldsymbol{M}(1\ldots r-1,1\ldots j)$
    DIVCONQ($\boldsymbol{P}$, $offset$)
  end if
  $\boldsymbol{Q}$:= $\boldsymbol{M}(r+1\ldots m, j\ldots n)$
  DIVCONQ($\boldsymbol{Q}$, $offset+r$)
end if
```

4.2 *SMAWK Algorithm*

Another algorithm, which exploits not only the monotonicity but also the total monotonicity of the matrix, is known as the SMAWK algorithm.[11] Like the divide-and-conquer algorithm, the SMAWK algorithm is recursive, but the total monotonicity of the matrix makes it possible to find the row-wise maxima of an $m \times n$ matrix $(m \leqslant n)$ in $O(n)$ time. Pseudocode of the SMAWK algorithm is shown in Algorithm 3. The algorithm consists of three functions and is started by calling SMAWK($\boldsymbol{A}$). The function REDUCE forms a central part of the algorithm; it removes $n-m$ columns that do not contain row maxima from the matrix. The REDUCE function can do so in $O(n)$ time; please refer to Ref. 11 for the derivation of the time complexity of the SMAWK algorithm. By calling the function REDUCE, the initial problem is transformed into the problem of finding the row-wise maxima in an $m \times m$ matrix. The maxima in the even-numbered rows are found by calling SMAWK recursively on a matrix containing only the even-numbered rows. The maxima in the odd-numbered rows are found by the function MFILL, which can be done very efficiently since the maxima in the even-numbered rows have already been found.

By combining the DP and the SMAWK algorithms, the optimal thresholds are found in $O(ML)$ time, where it is again assumed that the class cost can be calculated in $O(1)$ time.

Algorithm 3: SMAWK (M)

```
  $\boldsymbol{P}$:=REDUCE($M$)
  if $\boldsymbol{P}$ is size $1 \times 1$ then
    $\boldsymbol{P}$ is a maximum in the initial matrix, store
    position
    return
  end if
  $\boldsymbol{Q}$:=matrix with even-numbered rows of $\boldsymbol{P}$
  SMAWK ($\boldsymbol{Q}$) {recursive call}
  MFILL($\boldsymbol{P}$,$\boldsymbol{Q}$) {maxima in odd-numbered rows}
REDUCE ($\boldsymbol{M}$)
  [$m,n$]:=size of $\boldsymbol{M}$ {rows, columns}
  $k$:=1
  while $\boldsymbol{M}$ has more columns than rows do case
      $\boldsymbol{M}(k,k) \geqslant \boldsymbol{M}(k,k+1)$ and $k < m$:
        $m$:=$m+1$
      $\boldsymbol{M}(k,k) \geqslant \boldsymbol{M}(k,K+1)$ and $k = m$:
        Delete column $k+1$ of $\boldsymbol{M}$
      $\boldsymbol{M}(k,K) < \boldsymbol{M}(k,k+1)$:
        Delete column $k$ of $\boldsymbol{M}$
        if $k > 1$ then
          $k$:=$k-1$
        end if
    end case
  end while
  return $\boldsymbol{M}$
MFILL ($\boldsymbol{P}$,$\boldsymbol{Q}$)
  [$m,n$]:=size of $\boldsymbol{P}$ {rows, columns}
  MPOS [$2,4,\ldots,2\lfloor m/2 \rfloor$]:=pos. maxima in even-
    numbered rows of $\boldsymbol{P}$, known from $\boldsymbol{Q}$
  MPOS [0]:=1 MPOS[$m+1$]:=$n$
  for $i$:=$1\ldots\lceil m/2 \rceil$ do
    $r$:=$2i-1$
    $max$:=$-\infty$
    for $c$:=MPOS[$r-1$]...MPOS[$r+1$] do
      if $\boldsymbol{P}(r,c) > max$ then
        $max$:=$\boldsymbol{P}(r,c)$
        MPOS [$r$]:=$c$
      end if
    end for
  end for
```

4.3 *Objective Functions for Efficient Multilevel Thresholding*

In this section, a class of objective functions that fulfill the convex quadrangle inequality and have class costs that can be calculated in $O(1)$ time is introduced. Consequently, if the objective function of a multilevel thresholding method belongs to this class, it is possible to find the optimal thresholds in $O(ML)$ time. It will be shown in this section that the objective functions of two well-known methods are members of this class.

The objective function for this class is calculated as the sum over M classes, as defined in Eq. (7). Furthermore, the class cost must have the structure introduced in the following theorem.

Theorem 1: A class cost $l(p,q]$ of the form

$$l(p,q] = w(p,q] \cdot f\left(\frac{\sum_{p<i\leq q} p(i) \cdot \gamma(i)}{w(p,q]}\right), \tag{22}$$

where $w(p,q]$ is the class weight (probability of the class), $f(\cdot)$ is a convex function on the interval $[\gamma(1),\gamma(L)]$, and function $\gamma(\cdot)$ is either monotonically increasing or decreasing on the interval $[1,L]$, fulfills the convex quadrangle inequality.

Please refer to the Appendix for the proof of this theorem.

As with the maximum entropy method, it is possible to precalculate and store two arrays, which enables us to calculate the class cost in $O(1)$ time. These two arrays are defined as

$$N(i) = \begin{cases} p(1) \cdot \gamma(1), & \text{if } i = 1, \\ N(i-1) + p(i) \cdot \gamma(i), & \text{if } 2 \leq i \leq L, \end{cases} \tag{23}$$

$$W(i) = \begin{cases} p(1), & \text{if } i = 1, \\ W(i-1) + p(i), & \text{if } 2 \leq i \leq L. \end{cases} \tag{24}$$

Both arrays are L elements long, thus calculating and storing their values requires $O(L)$ time. After the arrays have been precalculated, the class cost $l(p,q]$ can be calculated as follows:

$$l(p,q] = [W(q) - W(p)] \cdot f\left(\frac{N(q) - N(p)}{W(q) - W(p)}\right). \tag{25}$$

In the following, it is shown that two existing thresholding methods have objective functions that belong to the class introduced in this section.

4.3.1 *Example A: The Otsu method*

The method proposed by Otsu[3] is one of the most referenced thresholding methods. Also numerous methods for improving the time efficiency of the multilevel case have been suggested. As mentioned earlier, a DP-based solution with time complexity $O(ML^2)$ has been proposed in Ref. 2. In Ref. 12, the execution time of the exhaustive search is reduced by precalculating and storing all possible class costs. In addition, various iterative methods have been proposed. For example, in Ref. 13, the pairwise nearest neighbor method; in Refs. 14 and 15, the zeros of the partial derivatives of the objective function; and in Ref. 16, the Nelder-Mead simplex search combined with particle swarm optimization are used to find the thresholds. However, it is difficult to specify an upper bound on the execution time for these methods because it is not guaranteed that they converge to the optimal thresholds within a given number of iterations.

The optimal thresholds for the method proposed by Otsu are found by minimizing a criterion called within-class variance, which is defined as follows:

$$\sigma_W^2 = \sum_{k=1}^{M} w(t_{k-1},t_k] \cdot \sigma^2(t_{k-1},t_k], \tag{26}$$

$$= \sum_{k=1}^{M} \sum_{i=t_{k-1}+1}^{t_k} p(i) \cdot \big(i - \mu(t_{k-1},t_k]\big)^2. \tag{27}$$

An equivalent problem is encountered when designing an optimal scalar quantizer. An M-level scalar quantizer assigns to each input value one of M reconstruction values. When the input values are discrete values in the range $1 \ldots L$, the quantizer is fully specified by a set of $M-1$ interval boundaries $0 < t_1 < t_2 \ldots < t_{M-1} < L$ and a set of M reconstruction values $r_1 < r_2 \ldots < r_M$. The quantizer assigns the reconstruction value r_k to input values in the interval $(t_{k-1},t_k]$, where $t_0 = 0$ and $t_M = L$. An optimal scalar quantizer uses a set of interval boundaries and reconstruction values such that the mean squared quantization error (MSE) is minimal.[17] The MSE is given by

$$MSE = \sum_{k=1}^{M} \sum_{i=t_{k-1}+1}^{t_k} p(i) \cdot (i - r_k)^2. \tag{28}$$

It can be shown that the reconstruction values have to be the centroids of the corresponding intervals, i.e., $r_k = \mu(t_{k-1},t_k]$, for the quantizer to be optimal. Therefore, the problem of designing an optimal scalar quantizer is equivalent to finding the Optimal thresholds for the Otsu method.

Wu showed in Refs. 18 and 19, that the optimal quantizer can be found in $O(ML \log L)$ and $O(ML)$ time, respectively, by employing algorithms combining DP and divide-and-conquer or SMAWK matrix searching. Note that the result that algorithms for scalar quantizer design with $O(ML)$ time complexity can be used to find the optimal thresholds for the Otsu method has also been found independently by Virmajoki.[20]

The fact that the optimal thresholds for the Otsu method can be found in $O(ML)$ time can also be seen when another objective function, called modified between-class variance in Ref. 12, is used. Maximizing this objective function results in the same thresholds as minimizing (26). The class cost of this objective function is defined as

$$l(p,q] = w(p,q] \cdot \big(\mu(p,q]\big)^2. \tag{29}$$

Note that the class cost has the form given in Eq. (22); therefore, it fulfills the convex quadrangle inequality, and the optimal thresholds can be found in $O(ML)$ time.

4.3.2 *Example B: The minimum cross entropy method*

For the minimum cross entropy method proposed in Ref. 7, the optimal threshold is the one that minimizes the cross entropy between the image and its binarized version, given by

$$\eta(t) = \sum_{i=1}^{t} h(i) i \log\left(\frac{i}{\mu(0,t]}\right) + \sum_{i=t+1}^{L} h(i) i \log\left(\frac{i}{\mu(t,L]}\right), \quad (30)$$

where it is assumed that the means of the two classes are the reconstruction values in the binarized image. It is straightforward to extend this method to multiple thresholds, and as with the Otsu method, the optimal thresholds can also be found by maximizing a modified objective function. The class cost of this objective function is given by

$$l(p,q] = w(p,q] \cdot \mu(p,q] \cdot \log\big(\mu(p,q]\big). \quad (31)$$

Obviously, this class cost has the form given in Eq. (22), which means that the optimal thresholds can be found in $O(ML)$ time.

5 Execution Time Measurements

In this paper, three different algorithms for efficient multilevel thresholding have been presented. Their time complexities of $O(ML^2)$, $O(ML \log L)$, and $O(ML)$ provide an upper bound for their execution time. It is clear that the algorithm that combines DP and SMAWK matrix searching and has a time complexity of $O(ML)$ outperforms the other algorithms if L and M are sufficiently large. However, from the time complexity measure alone, it is not possible to say which algorithm is the fastest for a certain combination of M and L, because the constant factors are unknown.

Quantitative statements about the performance of the algorithms can be made by implementing them and comparing their execution times. The implementations are made for the thresholding method proposed by Otsu.[3] In order to have efficient implementations, ANSI C is used, and no memory is allocated dynamically during the execution of the algorithms. The implementation of the SMAWK algorithm using a low-level programming language like ANSI C is quite involved. In fact, only implementations using high-level languages such as Java or Python can be found on the Internet. For the implementation of the SMAWK algorithm, modifications proposed in Ref. 21 proved to be helpful. For more details, please refer to the source code of the implementations used in this work, which is available online. (The source code of the implementations used in this work is available at http://ivpl.eecs.northwestern.edu/research/projects/thresholding.) The presented execution times were obtained by running the algorithms on a PC with a Pentium 4 2.8-GHz processor.

Since most grayscale images contain 256 gray levels, the execution times for this number of gray levels are of particular interest. The measured times for the different algorithms are shown in Fig. 3. The histogram of the Lena image (converted to gray scale) is used for the measurements. Note that the execution time of all algorithms is proportional to the number of classes. The algorithms that combine DP and matrix searching are both more than 10 times faster than the normal DP algorithm. Even though it has a higher time complexity, the algorithm that uses divide-and-conquer matrix searching is slightly faster than the algorithm that employs SMAWK. This may be explained by the overhead incurred by the complex structure of the SMAWK algorithm. As noted previously, our algorithms are optimal and they obtain the same thresholds as

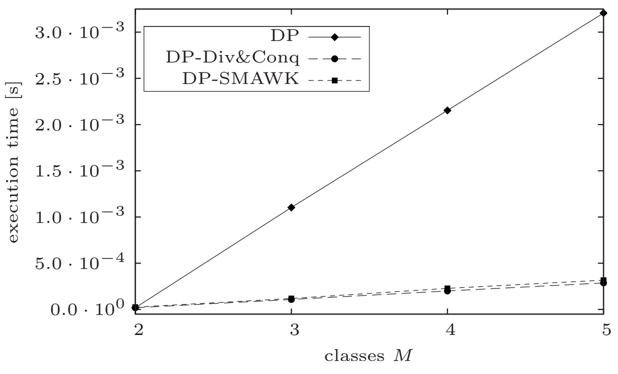

Fig. 3 Execution times for L=256, M=1, ... ,5.

the original method by Otsu,[3] and analyzing the segmentation performance of various methods is not within the scope of this paper. However, we show an example segmentation of the Lena image obtained by the Otsu method with M=5 in Fig. 4(b), where the pixels of each class are set to the mean gray level of the corresponding class.

When the number of gray levels is increased, the advantage of using an efficient matrix searching algorithm becomes more significant, as shown in Fig. 5. The histograms used for measurements with more than 256 gray levels are interpolated versions of the histogram of the Lena image. For M=5 and L=2^{16}, the normal DP algorithm requires about 218 s to find the optimal thresholds, whereas the execution times of the faster algorithms are around 100 ms. The difference between the algorithm that combines DP and divide-and-conquer matrix searching and the one that uses a combination of DP and SMAWK is shown in Fig. 6. It can be seen that the execution time of the algorithm that uses divide-and-conquer grows faster than linear, while the execution time of the algorithm that uses SMAWK is proportional to the number of gray levels, as theoretically predicted. Note that the algorithm that uses SMAWK requires only about 1.5 s to find the optimal thresholds for M=5 and L=2^{20}, while the normal DP algorithm would need about 15 h. Employing an algorithm based on an exhaustive search, as proposed in Ref. 12, is literally impossible for this combination of M and L—it would require millions of years for finding the optimal thresholds! Some of the measured execution times and the relative speedup to the DP algorithm are shown in Table 1.

An interesting question is how the structure of the histogram influences the execution time of the algorithms. It is easy to see that the amount of work the DP algorithm introduced in Sec. 3 needs to perform depends only on M and L and not on the structure of the histogram. Therefore, after the histogram has been calculated, the execution time will be the same for all images that have the same number of gray levels. On the other hand, the execution time of the matrix searching algorithms depends on the structure of the matrix. Consequently, the execution time of the algorithms combining DP and matrix searching will be influenced by the histogram. Three different histograms have been used for measurements, the first is from the Lena image, the second is from the Fishing Boat image, and the third is randomly generated. This means that a random number in the interval [0, 1] is used as the probability for each gray

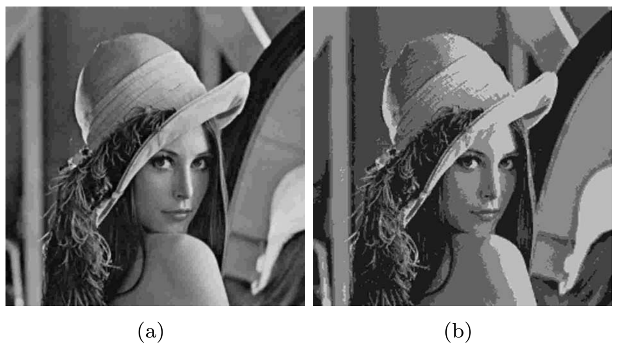

Fig. 4 Original (a) and segmentation (b) with $M=5$; thresholds: $t_1=46$, $t_2=83$, $t_3=118$, and $t_4=163$.

level, and $p(i)$ is then scaled such that $w(0,L]=1$. The execution times for these histograms are shown in Figs. 7 and 8.

As can be seen from these measurements, the execution times are slightly higher when random histograms are used, and the structure of the histogram influences the execution time of the algorithm using SMAWK more than that of the algorithm using divide-and-conquer. We found that the execution times using random histograms are close to the worst-case execution times, which means they can be used as an upper bound when the algorithms are used in real-time systems.

6 Summary and Conclusions

In this paper, we provide a framework for finding the solution approach of multilevel thresholding algorithms by defining two classes of objective functions for which the optimal thresholds can be found by efficient algorithms. In order for an objective function to belong to the first class, the class cost can depend only on its boundary values, as was shown in Sec. 3. We show that a DP scheme can be used for finding the optimal thresholds for this class of objective functions. Even though DP has been proposed in Ref. 2 for the method by Otsu[3] and in Ref. 4 for the method of Kittler and Illingworth,[5] it does not seem to be widely known. For example, for the method proposed in Ref. 6, we show that the optimal thresholds can be found by employing a similar DP algorithm with time complexity $O(ML^2)$. We define the second class of objective funcitons in Theorem 1. For objective functions belonging to this class, more sophisticated algorithms[18,19] from the field of optimal scalar quantization can be employed. As it turns out, Otsu's method also belongs to this second class and so does the multilevel extension of the minimum cross entropy method

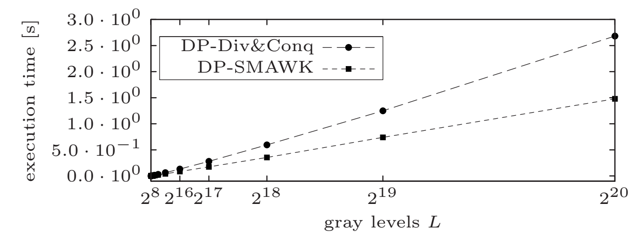

Fig. 6 Execution times for $L=2^8,\ldots,2^{20}$; $M=5$.

proposed in Ref. 7. Hence, we show that the optimal thresholds for the important Otsu method can be found in $O(ML)$ time. However, we note that this result has also been found independently by Virmajoki.[20] Depending on the number of gray levels L and classes M, this results in a speedup of several orders of magnitude relative to previously reported DP algorithms with time complexity $O(ML^2)$.

By comparing the execution times of actual implementations, we can make quantitative statements about the efficiency of the algorithms. The measured execution times are consistent with the theoretically derived time complexities.

Appendix: Proof for Theorem 1

In the following, we prove Theorem 1 for monotonically increasing functions $\gamma(\cdot)$; the proof for monotonically decreasing functions can be found by a symmetric argument. Our proof follows closely the proof of Ref. 22 (Theorem 3.6).

Proof: Every function $\gamma(\cdot)$, which is monotonically increasing on the interval $[1,L]$, maps the values $1\leqslant a<b<u<v\leqslant L$ to the values $\gamma(1)\leqslant\gamma(a)<\gamma(b)<\gamma(u)<\gamma(v)\leqslant\gamma(L)$. In the following derivations, the argument of the function $f(\cdot)$ is denoted as

$$\frac{\sum_{p<i\leqslant q}p(i)\cdot\gamma(i)}{w(p,q]}=\mu_\gamma(q,p]. \tag{32}$$

Since the mean $\mu(p,q]$ is monotonically nondecreasing in p and q and the function $\gamma(\cdot)$ is monotonically increasing, the function $\mu_\gamma(q,p]$ is also monotonically nondecreasing in p and q. Therefore, we have

$$\mu_\gamma(a,\mu]\leqslant\{\mu_\gamma(b,u],\mu_\gamma(a,v]\}\leqslant\mu_\gamma(b,v]. \tag{33}$$

The values $\mu_\gamma(b,u]$ and $\mu_\gamma(a,v]$ can be obtained by a linear combination of $\mu_\gamma(a,u]$ and $\mu_\gamma(b,v]$,

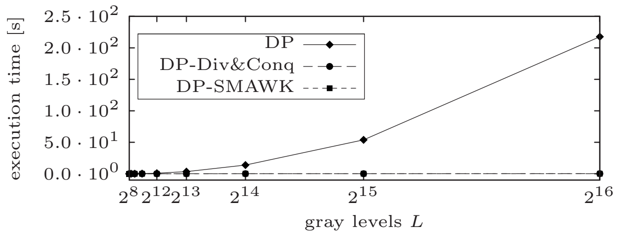

Fig. 5 Execution times for $L=2^8,\ldots,2^{16}$; $M=5$.

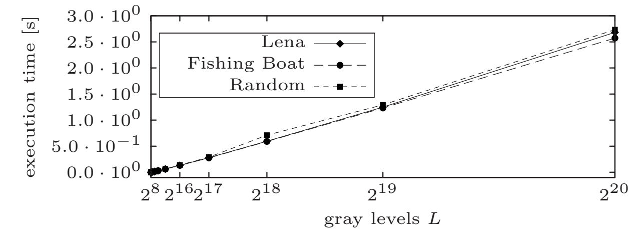

Fig. 7 Execution time *DP* combined with divide-and-conquer; $M=5$.

Table 1 Execution times for M=5, (time/speedup).

L	DP		DP-Div&Conq		DP-SMAWK	
256	3.21 ms	1	0.29 ms	11.1	0.31 ms	10.4
16384	13.8 s	1	29.9 ms	462	19.8 ms	697
65536	218 s	1	132 ms	1650	85.2 ms	2560

$$\mu_\gamma(b,u] = \alpha\mu_\gamma(a,u] + (1-\alpha)\mu_\gamma(b,v], \tag{34}$$

$$\mu_\gamma(a,v] = \beta\mu_\gamma(a,u] + (1-\beta)\mu_\gamma(b,v], \tag{35}$$

where the coefficients α and β are given by

$$\alpha = \frac{\mu_\gamma(b,v] - \mu_\gamma(b,u]}{\mu_\gamma(b,v] - \mu_\gamma(a,u]}, \tag{36}$$

$$\beta = \frac{\mu_\gamma(b,v] - \mu_\gamma(a,v]}{\mu_\gamma(b,v] - \mu_\gamma(a,u]}. \tag{37}$$

Since the function $f(\cdot)$ is convex, upper bounds for $l(b,u]$ and $l(a,v]$ can be found as

$$l(b,u] \leqslant w(b,u]\cdot[\alpha f(\mu_\gamma(a,u]) + (1-\alpha)f(\mu_\gamma(b,v])], \tag{38}$$

$$l(a,v] \leqslant w(a,v]\cdot[\beta f(\mu_\gamma(a,u]) + (1-\beta)f(\mu_\gamma(b,v])]. \tag{39}$$

We prove that the class cost (22) fulfills the convex quadrangle inequality by showing that 0 is a lower bound for d, which is defined as

$$d = l(a,u] + l(b,v] - l(b,u] - l(a,v]. \tag{40}$$

Such a lower bound is found by substituting the upper bounds for $l(b,u]$ and $l(a,v]$ into Eq. (40); after rearranging terms, it results in

$$d \geqslant \zeta\cdot f(\mu_\gamma(a,u]) + \eta\cdot f(\mu_\gamma(b,v]), \tag{41}$$

where ζ and η are given by

$$\zeta = w(a,u] - \alpha w(b,u] - \beta w(a,v], \tag{42}$$

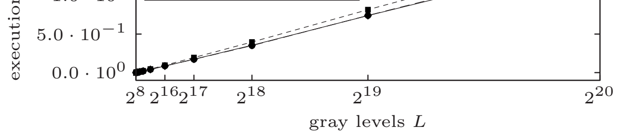

Fig. 8 Execution time *DP* combined with SMAWK; M=5.

$$\eta = w(b,v] - (1-\alpha)w(b,u] - (1-\beta)w(a,v]. \tag{43}$$

A rather long, but simple, computation reveals that ζ=0 and η=0, which means that

$$l(a,u] + l(b,v] - l(b,u] - l(a,v] \geqslant 0, \tag{44}$$

and proves Theorem 1.

References

1. M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *J. Electron. Imaging* **13**, 146–165 (2004).
2. N. Otsu, "An automatic threshold selection method based on discriminant and least squares criteria," *Trans. IECE Japan* **64-D**(4), 349–356 (1980).
3. N. Otsu, "A threshold selection method from gray level histograms," *IEEE Trans. Syst., Man, Cybern., Part B: Cybern.* **9**, 62–66 (1979).
4. T. Kurita, N. Otsu, and N. N. Abdelmalek, "Maximum likelihood thresholding based on population mixture models," *Pattern Recogn.* **25**(10), 1231–1240 (1992).
5. J. Kittler and J. Illingworth, "Minimum error thresholding," *Pattern Recogn.* **19**(1), 41–47 (1986).
6. J. Kapur, P. Sahoo, and A. Wong, "A new method for gray-level picture thresholding using the entropy of the histogram," *Comput. Vis. Graph. Image Process.* **29**, 273–285 (1985).
7. C. H. Li and C. K. Lee, "Minimum cross entropy thresholding," *Pattern Recogn.* **26**, 617–625 (1993).
8. M. Luessi, M. Eichmann, G. M. Schuster, and A. K. Katsaggelos, "New results on efficient optimal multilevel image thresholding," in *IEEE Int. Conf. Image Process. 2006 (ICIP 2006)*, pp. 773–776 (2006).
9. R. E. Burkard, B. Klinz, and R. Rudolf, "Perspectives of Monge properties in optimization," *Discrete Appl. Math.* **70**(2), 95–161 (1996).
10. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*,2nd ed., MIT Press, Cambridge, MA (2001).
11. A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. E. Wilber, "Geometric applications of a matrix-searching algorithm," *Algorithmica* **2**, 195–208 (1987).
12. P. Liao, T. Chen, and P. Chung, "A fast algorithm for multilevel thresholding," *J. Comput. Inf. Sci. Eng.* **17**, 713–727 (2001).
13. O. Virmajoki and P. Fränti, "Fast pairwise nearest neighbor based algorithm for multilevel thresholding," *J. Electron. Imaging* **12**(4), 648–659 (2003).
14. K. C. Lin, "Fast thresholding computation by searching for zero derivatives of image between-class variance," in *IECON '01, 27th Ann. Conf. IEEE*, vol. 1, pp. 393–397, Industrial Electronics Soc. (2001).
15. K. C. Lin, "On improvement of the computation speed of Otsu's image thresholding," *J. Electron. Imaging* **14**(2), 023011 (2005).
16. E. Zahara, S.-K. S. Fan, and D.-M. Tsai, "Optimal multi-thresholding using a hybrid optimization approach," *Pattern Recogn. Lett.* **26**(8), 1082–1095 (2005).
17. J. Max, "Quantizing for minimum distortion," *IEEE Trans. Inf. Theory* **IT-6**, 7–12 (1960).
18. X. Wu and J. G. Rokne, "An o(kn lgn) algorithm for optimum k-level quantization on histograms of n points," in *ACM Conf. Computer Sci.*, 339–343 (1989).
19. X. Wu and K. Zhang, "Quantizer monotonicities and globally optimal scalar quantizer design," *IEEE Trans. Inf. Theory* **39**(3), 1049–1053 (1993).
20. O. Virmajoki, "Pairwise nearest neighbor method revisited," PhD Thesis, Univ. of Joensuu, Finland (2004).
21. J. Hershberger and S. Suri, "Matrix searching with the shortest-path metric," *SIAM J. Comput.* **26**(6), 1612–1634 (1997).

22. A. Said, "On the reduction of entropy coding complexity via symbol grouping: part I—redundancy analysis and optimal alphabet partition," Technical Report HPL-2004–145, HP Labs, Palo Alto, CA (2004).

Martin Luessi received an FH degree in electrical engineering from the University of Applied Sciences of Eastern Switzerland, Rapperswil, in 2006, and an MS degree in electrical engineering from Northwestern University, Evanston, Illinois, in 2007. At the University of Applied Sciences of Eastern Switzerland, he was awarded the UBS innovation award for his thesis on efficient multilevel image thresholding. He is currently pursuing his PhD degree in the Department of Electrical Engineering and Computer Science at Northwestern University, where he is a research assistant with the Image and Video Processing Laboratory. His primary research interests include multimodal neuroimaging, video and image compression, computer vision, and pattern recognition.

Marco Eichmann received his FH degree in electrical engineering from the University of Applied Sciences of Eastern Switzerland, Rapperswil, in 2006. For his thesis on efficient multilevel image thresholding, he was awarded the UBS innovation award. He is currently working as a project manager and software engineer for industrial image processing and quality control for a company in Pfäffikon SZ, Switzerland.

Guido M. Schuster received his PhD in electrical engineering from Northwestern University in 1996. He cofounded the 3Com Internet Communications Business Unit in 1999, and as its chief technical officer and senior director, he developed the first commercially available SIP IP telephony system. He is currently a professor at the University of Applied Sciences of Eastern Switzerland in Rapperswil, where he focuses on digital signal processing and wireless sensor networks. Schuster holds more than 50 patents, is the coauthor of the book *Rate-Distortion Based Video Compression*, and has published over 60 peer-reviewed articles. Furthermore, he is a recipient of the gold medal for academic excellence at the Neu Technikum Buchs, a winner of the Landis & Gyr fellowship competition, a recipient of the 3Com inventor of the year award, a recipient of the IEEE Signal Processing Society Best Paper Award, and three times a a recipient of the FUTUR Technology Transfer Innovation Award.

Aggelos K. Katsaggelos received a Diploma degree in electrical and mechanical engineering from the Aristotelian University of Thessaloniki, Greece, in 1979, and MS and PhD degrees in electrical engineering from Georgia Tech, in 1981 and 1985, respectively. In 1985, he joined the Department of EECS at Northwestern University, where he is currently a professor (and past holder of the Ameritech Chair of Information Technology). He is also the director of the Motorola Center for Seamless Communications and a member of the Academic Affiliate Staff, Evanston Hospital. He has published extensively (5 books, 160 journal papers, 380 conference papers, 12 patents). He is a fellow of the IEEE (1998) and the recipient of the IEEE Third Millennium Medal (2000), the IEEE Signal Processing Society Meritorious Service Award (2001), an IEEE Signal Processing Society Best Paper Award (2001), an IEEE ICME Paper Award (2006), and an IEEE ICIP Paper Award (2007). He is a distinguished lecturer of the IEEE Signal Processing Society (2007 to 2008).