# NEW RESULTS ON EFFICIENT OPTIMAL MULTILEVEL IMAGE THRESHOLDING

*M. Luessi, M. Eichmann, and G. M. Schuster*

Abteilung Elektrotechnik
Hochschule Rapperswil, Switzerland

*Aggelos K. Katsaggelos*

Department of EECS
Northwestern University, USA

## ABSTRACT

Image thresholding is one of the most common image processing operations, since almost all image processing schemes need some sort of separation of the pixels into different classes. In order to find the thresholds, almost all methods analyze the histogram of the image. In most cases, the optimal thresholds are found by either minimazing or maximazing an objective function, which depends on the positions of the thresholds. We identify two classes of objective functions for which the optimal thresholds can be found by algorithms with low time complexity. We show, that for example the method proposed by Otsu [1] 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 make a quantitative statement about their performance.

***Index Terms***— Image segmentation, Dynamic Programming

## 1. INTRODUCTION

In many image processing applications the pixels need to be classified as belonging to the foreground or the background. Because of its importance, image thresholding has attracted a considerable amount of attention and in [2] an extensive taxonomy and comparison of prior proposed methods has been published. Many methods define the optimal threshold as the one which maximizes or minimizes an objective function. In multilevel image thresholding pixels can be classified into many classes, not just foreground and background.

One method to find the thresholds is an exhaustive search, which means calculating the objective function for every possible placement of the thresholds. 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. One way to overcome this shortcoming is to employ an iterative algorithm. However, it is difficult to guarantee optimality for these algorithms. An algorithm, which is fast and optimal, is therefore desireable.

In this paper, algorithms are studied which can be employed to find the optimal thresholds efficiently. As a result, two classes of objective functions are identified. For the first class, an efficient dynamic programming algorithm can be used for finding the thresholds, whereas for the second class a combination of dynamic programming and fast matrix searching can be employed. Furthermore, it is shown that some well known thresholding techniques are members of these classes. To verify the efficiency of the algorithms, execution time measurements of ANSI C implementations are presented. Figures depicting the segmentation performance are not shown because the proposed algorithms provide the same results as the original methods.

The paper is organized as follows. In section 2, multilevel image thresholding is introduced and the problem is mathematically defined. In section 3, the dynamic programming scheme is presented. In section 4, a class of objective functions is presented for which the optimal thresholds can be found even more efficiently. In section 5, execution time measurements of the different algorithms are presented. Finally, the paper is summarized and conclusions are drawn in section 6.

## 2. MULTILEVEL IMAGE THRESHOLDING

The pixels of a gray scale 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 $C_1 \dots C_M$, by setting the thresholds $t_1 \dots t_{M-1}$. Class $C_k$ is defined as $C_k = \{g | t_{k-1} < g \leq t_k\}$ , where $k$ refers to the class number $k \in \{i | 1 \leq i \leq M\}$. The thresholds $t_0$ and $t_M$ are defined to be 0 and $L$, respectively.

For the placement of the thresholds most methods employ the histogram. The histogram $h(g)$ shows the number of occurrence of the gray level $g$ in the image, where $\sum_{g=1}^{L} h(g) = N$, and $N$ is 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

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

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

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

The thresholding methods considered in this paper are the result of the optimization of an objective function. The value of the objective function dependens on the positions of the thresholds. The optimal thresholds are the ones which either minimize or maximize the objective function. For the sake of conciseness and without loss of generality, derivations are only shown for the case where the objective function is maximized

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

with the following constraint for the positions of the thresholds

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

The straightforward approach for finding the optimal thresholds is an exhaustive search which means evaluating the objective function for every possible combination of thresholds. However, the number of possible combinations $N$ is given by (it is assumed that $M << L$)

$$N = \binom{L-1}{M-1} = \prod_{i=1}^{M-1} \frac{L-i}{M-i} \geq \left(\frac{L-1}{M-1}\right)^{M-1}. \quad (6)$$

Therefore, an algorithm based on an exhaustive search is not suitable when the image is segmented into more than two classes.

## 3. DYNAMIC PROGRAMMING ALGORITHM

For the first class of objective functions with the structure below, a dynamic programming algorithm, known as the shortest path algorithm, can be employed to find the optimal thresholds. The required structure is

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

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

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

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

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

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

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

By setting $m = M$ and $l = L$, this equation maximizes the overall problem. It is clear that if the thresholds of a subproblem are not set optimally and therefore are not maximizing the subproblem, the objective function can never be maximal. By having this recursive formulation, the optimal thresholds can be found by a shortest path algorithm. The algorithm employs a trellis to find the optimal thresholds, as shown in Fig. 1 ($M = 4$, $L = 8$). It 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 the backpointer 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. For more than two classes this algorithm is more efficient than an exhaustive search. Since the number of stages is $M$, the number of nodes per stage $L - M + 1$ and the

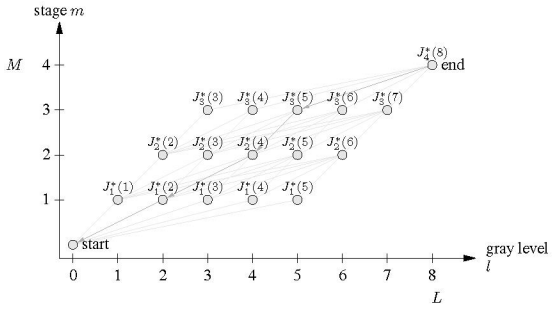


**Fig. 1**. Trellis for the shortest path algorithm.

number of paths which have to be compared per node proportional to $L$, the time complexity of this algorithm is $O(ML^2)$. This assumes, that the class cost $\ell(t_{k-1}, t_k]$ can be calculated in $O(1)$ time. This is the case for many objective functions by introducing a preprocessing step which requires $O(L)$ time, as will be shown later. In [3] a dynamic programming scheme based on (10) has been proposed for the thresholding method by Otsu [1] and in [4] for the minimum error thresholding method by Kittler and Illingworth [5]. However, since [3] was published in a Japanese journal the knowledge that this method can be used does not seem to be widespread. In [6] a suboptimal algorithm was proposed and the authors wrote that an exhaustive search is the only possibility to find the optimal thresholds. In [7] an algorithm based on an exhaustive search is proposed. The shortest path algorithm is also shown here because it builds the basis of more efficient algorithms, which are introduced in the next section. Furthermore we propose that the DP algorithm can be employed for finding the optimal thresholds for the maximum entropy method by Kapur [8]. Due to lack of space, the derivations are not shown here, but they can be found in [9].

## 4. MORE EFFICIENT ALGORITHMS

The problem of finding the optimal paths to all the nodes in one stage in the trellis, is equivalent to the problem of finding the row-wise maxima in a lower triangular matrix defined as

$$M(r,c) = \begin{cases} -\infty, & \text{if } c > r, \\ J_{m-1}^*(c+m-2) + \ell(c+m-2, r+m-1], & \text{if } c \leq r. \end{cases} \quad (11)$$

In this matrix, the cost of the paths up to all the nodes in 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 in the upper triangular region of the matrix are defined to be $-\infty$, since there are no paths coming from nodes to the right or directly below the current node. The resulting size of the search matrix is $(L - M + 1) \times (L - M + 1)$.

For a second class of objective functions with a class cost of the form

$$\ell(p,q] = w(p,q] \cdot f\left(\frac{\sum_{p<i\leq q} p(i) \cdot \gamma(i)}{w(p,q]}\right), \quad (12)$$

where $w(p,q]$ is the class weight, $f(x)$ is a convex function on the interval $[\gamma(1), \gamma(L)]$ and the function $\gamma(x)$ is either monotone increasing or decreasing on the interval $[1, L]$, the matrix (11) is totally monotone. This follows, because the class cost shown in (12) fulfills the convex quadrangle inequality. The proof is omitted here

and can be found in [9]. In addition, the class cost can be calculated in $O(1)$ time after the following preprocessing step. For this two arrays $W(i)$ and $N(i)$ are defined recursively as

$$N(i) = \begin{cases} p(1) \cdot \gamma(1), & \text{if } i = 1, \\ N(i-1) + p(i) \cdot \gamma(i), & \text{if } 2 \le i \le L, \end{cases} \tag{13}$$

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

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

$$\ell(p,q] = [W(q) - W(p)] \cdot f\left(\frac{N(q) - N(p)}{W(q) - W(p)}\right). \tag{15}$$

The total monotonicity of the matrix makes it possible, to employ algorithms, which find the row-wise maxima more efficiently.

**Divide-and-Conquer Algorithm:** The divide-and-conquer algorithm exploits the fact, that the row maxima in a monotone matrix build a staircase. It first finds the maximum in the middle row of the matrix and is then executed recursively on two submatrices. This algorithm can find the row maxima of a monotone $m \times n$ matrix in $O(n \log m)$ time. By combining it with DP the optimal thresholds are found in $O(ML \log L)$ time.

**SMAWK Algorithm:** Another algorithm, which exploits not only the monotonicity but the total monotonicity of the matrix, is called SMAWK algorithm [10] and finds the row maxima of a $m \times n$ matrix ($m \le n$) in $O(n)$ time. The combination of DP and the SMAWK algorithm, finds the optimal thresholds in $O(ML)$ time.

**Examples:**

**A)** The optimal thresholds for the method proposed by Otsu [1] are found by minimizing a criterion called within-class variance, which is defined as follows

$$\sigma_W^2 = \sum_{k=1}^{M} w(t_{k-1}, t_k] \cdot \sigma^2(t_{k-1}, t_k], \tag{16}$$

$$= \sum_{k=1}^{M} \sum_{i=t_{k-1}+1}^{t_k} p(i) \cdot (i - \mu(t_{k-1}, t_k])^2. \tag{17}$$

Note, that an equivalent problem is encountered when designing an optimal scalar quantizer (minimum mean squared quantization error). For optimal scalar quantization, where a histogram with $N$ points is divided into $K$ intervals, Wu showed in [11] and [12], that the optimal quantizer can be found respectively in $O(KN \log N)$ and $O(KN)$ time, by employing the above mentioned algorithms. The fact that the optimal thresholds can be found in $O(ML)$ time, can also be seen, when another objective function, called modified between-class variance in [7], is used. Maximizing this objective function results in the same thresholds as minimizing (16). The class cost of this objective function is defined as

$$\ell(p,q] = w(p,q] \cdot \big(\mu(p,q]\big)^2. \tag{18}$$

It is obvious, that the class cost of this objective function has the form defined in (12).

**B)** The minimum cross entropy method proposed in [13], can be extended to multiple thresholds. The optimal thresholds can be found by maximizing a modified objective function. The class cost of this objective function is

$$\ell(p,q] = w(p,q] \cdot \mu(p,q] \cdot \log\big(\mu(p,q]\big). \tag{19}$$

Since the class cost is of the form (12) 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 $O(ML)$ give an upper bound for their execution time. It is clear, that the algorithm which combines DP and SMAWK matrix searching and has a time complexity of $O(ML)$ outperforms the other algorithms if $L$ and $M$ are sufficiently high. However, from the time complexity 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 [1]. 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 [14] prove to be helpful. Detailed information about the implementation and the ANSI C code can be found in [9]. Since most gray scale 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. 2. The histogram of the Lenna image (converted to gray scale) is used for the measurements. Note, that the execution time of all algorithms is

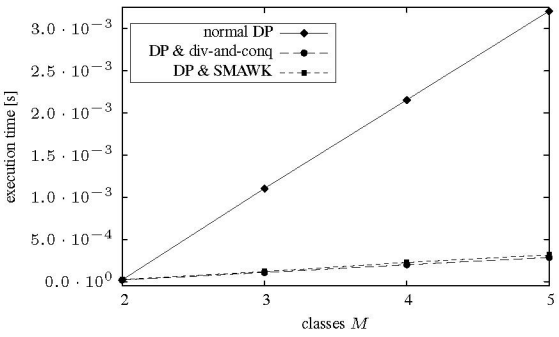


**Fig. 2**. Execution times for $L = 256$, $M = 1 \ldots 5$

proportional to the number of classes. The algorithms which combine DP and matrix searching are both about 10 times faster than the normal DP algorithm. Even though it has a higher time complexity, the algorithm which uses divide-and-conquer matrix searching is slightly faster than the algorithm which employs SMAWK. This may be explained by the overhead incurred by the complex structure of the SMAWK algorithm.

When the number of gray levels is increased, the advantage of using an efficient matrix searching algorithm becomes more significant, as shown in Fig. 3. The histograms used for measurements with more than 256 gray levels are interpolated versions of the histogram of the Lenna image. For $M = 5$ and $L = 2^{16}$ the normal DP algorithm requires about 218s to find the optimal thresh-

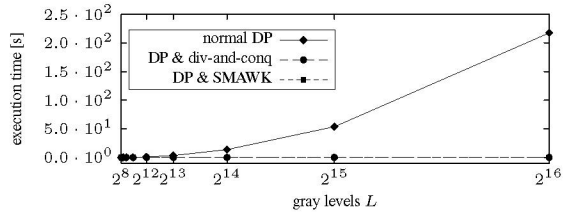


**Fig. 3**. Execution times for $L = 2^8 \dots 2^{16}$, $M = 5$

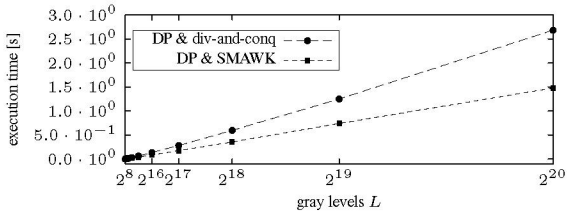


**Fig. 4**. Execution times for $L = 2^8 \dots 2^{20}$, $M = 5$

olds, whereas the execution times of the faster algorithms are around 100ms. The difference between the algorithm which combines DP and divide-and-conquer matrix searching and the one which uses a combination of DP and SMAWK is shown in Fig. 4. It can be seen, that the execution time of the algorithm which uses divide-and-conquer grows faster than linear while the execution time of the algorithm which uses SMAWK is proportional to the number of gray levels, as theoretically predicted. Note, that the algorithm which uses SMAWK only requires about 1.5s to find the optimal thresholds for $M = 5$ and $L = 2^{20}$, while the normal DP algorithm would need about 15 hours. Employing an algorithm based on an exhaustive search, as proposed in [7], 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.

| $L$ | DP | | DP & div-and-conq | | DP & SMAWK | |
|---|---|---|---|---|---|---|
| 256 | 3.21ms | 1 | 0.29ms | 11.1 | 0.31ms | 10.4 |
| 16384 | 13.8s | 1 | 29.9ms | 462 | 19.8ms | 697 |
| 65536 | 218s | 1 | 132ms | 1650 | 85.2ms | 2560 |

**Table 1**. Execution times for $M = 5$, (time | speedup)

## 6. SUMMARY AND CONCLUSIONS

In this paper, we identify two classes of objective functions for which the optimal thresholds can be found by efficient algorithms. For the first class, we show that a dynamic programming scheme can be used for finding the optimal thresholds. Even though this scheme has been proposed for the method by Otsu [3] and for the method of Kittler and Illingworth [4], it does not seem to be widely known. For example, for the method proposed in [8] we reduce the time complexity using this dynamic programming approach to $O(ML^2)$. For the second class, more sophisticated algorithms [11][12] 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 minimum cross entropy method proposed in [13]. Hence, we improve the state of the art for the well known Otsu method from $O(ML^2)$ to $O(ML)$. Depending on number of gray levels $L$ and classes $M$, this results in a speedup of several magnitudes.

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.

The question of whether or not the thresholds found by the algorithms are meaningful for the segmentation of images is not addressed. Finding an objective function, which is a member of the second class and outperforms current thresholding methods, could be the topic of further research.

## 7. REFERENCES


[1] N. Otsu, "A threshold selection method from gray level histograms," *IEEE Trans. Systems, Man and Cybernetics*, vol. 9, pp. 62–66, Mar. 1979.

[2] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic Imaging*, vol. 13, no. 1, pp. 146–165, Jan. 2004.

[3] N. Otsu, "An automatic threshold selection method based on discriminant and least squares criteria," *Transactions of the IECE of Japan*, vol. 64-D(4), pp. 349–356, 1980.

[4] T. Kurita, N. Otsu, and N. N. Abdelmalek, "Maximum likelihood thresholding based on population mixture models.," *Pattern Recognition*, vol. 25, no. 10, pp. 1231–1240, 1992.

[5] J. Kittler and J. Illingworth, "Minimum error thresholding," *Pattern Recogn.*, vol. 19, no. 1, pp. 41–47, 1986.

[6] O. Virmajoki and P. Fränti, "Fast pairwise nearest neighbor based algorithm for multilevel thresholding," *Journal of Electronic Imaging*, vol. 12(4), pp. 648–659, Oct. 2003.

[7] P. Liao, T. Chen, and P. Chung, "A fast algorithm for multilevel thresholding," *Journal of Information Science and Engineering*, vol. 17, pp. 713–727, Sept. 2001.

[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," *Computer Vision, Graphics, and Image Processing*, vol. 29, pp. 273–285, 1985.

[9] M. Eichmann and M. Luessi, "Efficient multilevel image thresholding," Dec. 2005, Hochschule Rapperswil, Switzerland, `http://www.medialab.ch/ICIP2006/`.

[10] A. Aggarwal, M. M. Klawe, S. Moran, P. W. Shor, and R. E. Wilber, "Geometric applications of a matrix-searching algorithm.," *Algorithmica*, vol. 2, pp. 195–208, 1987.

[11] X. Wu and J. G. Rokne, "An o(kn lgn) algorithm for optimum k-level quantization on histograms of n points.," in *ACM Conference on Computer Science*, 1989, pp. 339–343.

[12] X. Wu and K. Zhang, "Quantizer monotonicities and globally optimal scalar quantizer design.," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 1049–1053, 1993.

[13] C. H. Li and C. K. Lee, "Minimum cross entropy thresholding," *Pattern Recognition*, vol. 26, no. 4, pp. 617–625, Apr. 1993.

[14] J. Hershberger and S. Suri, "Matrix searching with the shortest-path metric.," *SIAM J. Comput.*, vol. 26, no. 6, pp. 1612–1634, 1997.