Optimizing Speed For Adaptive Local Thresholding Algorithm Using Dynamic Programing

Duong Anh Duc, Tran Le Hong Du, Tran Duc Duan
Faculty of Information Technology
University of Natural Sciences, VNU-HCMC
227 Nguyen Van Cu Street, Dist 5, HCM City, Vietnam
Tel : +84-8-8308117 Fax : +84-8-8350096
Email: {daduc, tlhdu, tdduan}@fit.hcmuns.edu.vn

Abtract:
Image binarization using a global threshold value [3] performs at high speed, but usually results in undesired binary images when the source images are of poor quality. In such cases, adaptive local thresholding algorithms [1][2][3] are used to obtain better results, and the algorithm proposed by A.E.Savekis which chooses local threshold using foreground and background clustering [1] is one of the best thresholding algorithms. However, this algorithm runs slowly due to its re-computing threshold value of each central pixel in a local window MxM . In this paper, we present a dynamic programming approach for the step of calculating local threshold value that reduces many redundant computations and improves the execution speed significantly. Experiments show that our proposal improvement runs more ten times faster than the original algorithm.

Keywords: Adaptive Local Thresholding, Binarization, Image Processing, Dynamic Programming.

1. INTRODUCTION

Image binarization algorithms are often used in almost problems involving in image processing or computer vision domain. It is the basic procedure that is used in various algorithms; therefore, the speed problem is very important. Since the source images may contain multiple object classes of varying color, non-uniform illumination, and camera distortions, the global threshold algorithms do not work well on those images [2]. In such situations, the adaptive local threshold algorithms are used to obtain better results. The adaptive threshold algorithm using foreground and background clustering has proven its efficiencies [1][2][3]. It computes threshold value of each central pixel in its surround local window $MxM, M = 7,9,11...$ We have found that, the computing threshold of each local window has numerous redundant operators. As moving the window in each step, the overlap area is $(M-1)xM$. Therefore, the number of foreground pixels and background pixels that are recalculated in overlap area is large. Thus, we propose using dynamic programming to reduce redundant computations. This approach can immediately determine threshold value of each pixel on various window sizes as needed.

2. PREVIOUS WORK

In order to classify gray images into bi-level images, there are two approaches: using global threshold and using adaptive local threshold.

Let Img be the source image with size WxH, $\text{Img}[i][j] \in [0..255], \forall i \in [1..H], j \in [1..W]$, and ImgB is the result binary image.

2.1. Global Thresholding Method

According to this method, all pixels are compared to a same value in the thresholding procedure. This value may be constant, or be chosen from image histogram [3]. The implementation of this method is showed as follows.

```
Determine GlobalThreshold (using histogram, or the
average value of pixels, ...)
For each i ∈ [1..H] do
    For each j ∈ [1..W] do
        If Img[i][j] ≤ GlobalThreshold Then
            ImgB[i][j] ← 1
        Else
            ImgB[i][j] ← 0
        End If
    End For
End For
```

Result image ImgB (size WxH) is a binary image that is classified from source image Img. The algorithm complexity is WxH plus the cost of calculating threshold value (entirely about $O(N^2), N$ – image size).

2.2. Adaptive Local Thresholding Method

Unlike global thresholding method, this approach compute an independent threshold for each pixel over a local window whose center is the pixel being binarizing (M – window size). Each algorithm following this approach has a different method of determining threshold values on local window. In [1], each pixel is assigned to a background or foreground level (using global threshold), then the threshold value is determined by the average of two background/foreground clusters. Let consider a brief original algorithm:

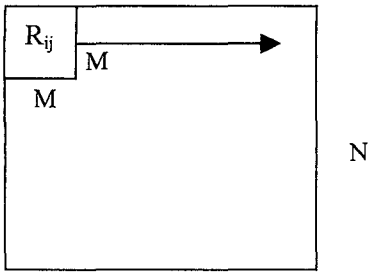

Fig 1. Computing threshold value on local window R_{ij}

We want to compute threshold of pixel (i,j). With each point (i,j), let R_{ij} be the set of pixels in its local window size M. We has,

$$R_{ij} = \left\{(u,v)/u \in \left[i-\frac{M}{2}, i+\frac{M}{2}\right], v \in \left[j-\frac{M}{2}, j+\frac{M}{2}\right]\right\}$$

Using a global threshold to clustering all pixels in R_{ij} into two set, FgR_{ij} – foreground set and BgR_{ij} – background:

$$FgR_{ij} = \left\{\begin{matrix}(u,v) \ / (u,v) \in R_{ij} \wedge \\ \mathrm{Im}\, g[u][v] \le GlobalThreshold\end{matrix}\right\} \quad (1)$$

$$BgR_{ij} = \left\{\begin{matrix}(u,v) \ / (u,v) \in R_{ij} \wedge \\ \mathrm{Im}\, g[u][v] > GlobalThreshold\end{matrix}\right\} \quad (2)$$

Then, compute:

$$SFR_{ij} = \sum_{(u,v)\in FgR_{ij}} \mathrm{Im}\, g[u][v], \text{ and}$$

$$LFV_{ij} = \frac{SFR_{ij}}{Card(FgR_{ij})} \quad (3)$$

$$SBR_{ij} = \sum_{(u,v)\in BgR_{ij}} \mathrm{Im}\, g[u][v], \text{ and}$$

$$LBV_{ij} = \frac{SBR_{ij}}{Card(BgR_{ij})} \quad (4)$$

And finally, local threshold value of pixel (i,j) is:

$$LThresh_{ij} = \frac{LFV_{ij} + LBV_{ij}}{2}$$

The original algorithm is illustrated in the following code:

```
For each (i,j) ∈ [1..W]x[1..H] do
    SF ← 0
    SB ← 0
    CardF ← 0
    CardB ← 0
    For each (u,v) ∈ Rij do
        If Img[u][v] ≤ GlobalTreshold Then
            SF ← SF + Img[u][v]
            CardF ← CradF + 1
        Else
            SB ← SB + Img[u][v]
            CardB ← CradB + 1
        End If
    End For
    LocalThreshold = (SF/CardF + SB/CardB)/2
    If Img[i][j] ≤ LocalThreshold Then
        ImgB[i][j] ← 1
    Else
        ImgB[i][j] ← 0
    End If
End For
```

The complexity of this algorithm is estimated to be about $WxHxM^2 \left(\approx O(N^2xM^2)\right)$. It depends on M, the window size, the larger M is, the slower the total a gorithm speed is.

3. APPLYING DYNAMIC PROGRAMMING TO OPTIMIZE THE ALGORITHM SPEED

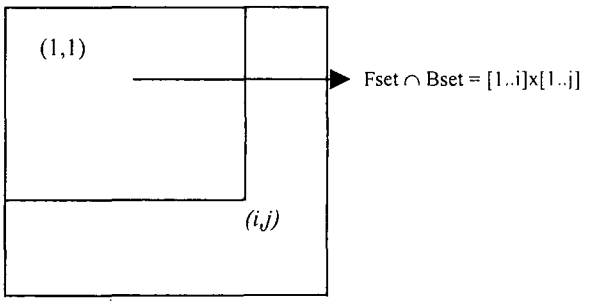

Fig 2. FgSet and BgSet

In the rectangle specified by two points (1,1) and (i,j), there are two sets using clustering by global threshold, $FgSet_{ij}$ – foreground set and $BgSet_{ij}$ – background:

$$FgSet_{ij} = \left\{\begin{matrix}(u,v) \ / \ (u,v)/1 \le u \le i \wedge 1 \le v \le j \wedge \\ \mathrm{Im}\, g[u][v] < GlobalThreshold\end{matrix}\right\} \quad (5)$$

$$BgSet_{ij} = \left\{\begin{matrix}(u,v) \ / \ (u,v)/1 \le u \le i \wedge 1 \le v \le j \wedge \\ \mathrm{Im}\, g[u][v] \ge GlobalThreshold\end{matrix}\right\} \quad (6)$$

Then, compute:

$$SFP_{ij} = \sum_{(u,v)\in FgSet_{ij}} \mathrm{Im}\, g[u][v], \text{ and}$$

$$CFP_{ij} = Card(FgSet_{ij}) \quad (7)$$

$$SBP_{ij} = \sum_{(u,v)\in BgSet_{ij}} \mathrm{Im}\, g[u][v], \text{ and}$$

$$CBP_{ij} = Card(BgSet_{ij}) \quad (8)$$

It can be inferred the following recursive formulae to compute values of $SFP_{ij}, CFP_{ij}, SBP_{ij}, CBP_{ij}$:

$$SFP_{i,j} = \begin{cases} SFP_{i,j-1} + SFP_{i-1,j} - SFP_{i-1,j-1} + \mathrm{Im}g[i][j], \\ \quad if\ \mathrm{Im}g[i][j] \le GlobalThreshold \\ SFP_{i,j-1} + SFP_{i-1,j} - SFP_{i-1,j-1}, otherwise \end{cases} \quad (9)$$

and

$$CFP_{i,j} = \begin{cases} CFP_{i,j-1} + CFP_{i-1,j} - CFP_{i-1,j-1} + 1, \\ \quad if\ \mathrm{Im}g[i][j] \le GlobalThreshold \\ CFP_{i,j-1} + CFP_{i-1,j} - CFP_{i-1,j-1}, otherwise \end{cases} \quad (10)$$

$$SBP_{i,j} = \begin{cases} SBP_{i,j-1} + SBP_{i-1,j} - SBP_{i-1,j-1} + \mathrm{Im}g[i][j], \\ \quad if\ \mathrm{Im}g[i][j] > GlobalThreshold \\ SBP_{i,j-1} + SBP_{i-1,j} - SBP_{i-1,j-1}, otherwise \end{cases} \quad (11)$$

and

$$CBP_{i,j} = \begin{cases} CBP_{i,j-1} + CBP_{i-1,j} - CBP_{i-1,j-1} + 1, \\ \quad if\ \mathrm{Im}g[i][j] > GlobalThreshold \\ CBP_{i,j-1} + CBP_{i-1,j} - CBP_{i-1,j-1}, otherwise \end{cases} \quad (12)$$

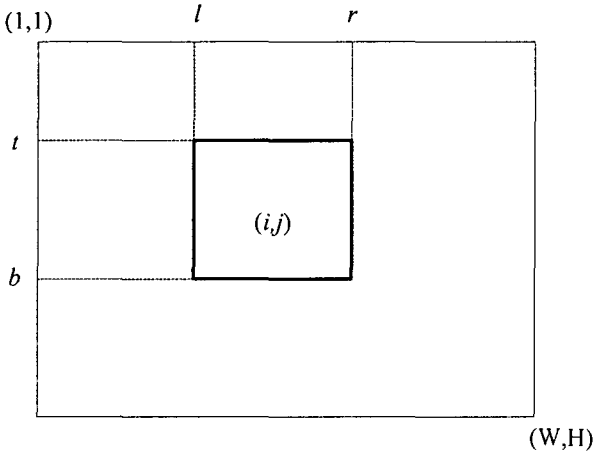

Fig 3. Computing values of window R_{ij}

After all values $SFP_{ij}, CFP_{ij}, SBP_{ij}, CBP_{ij}$ have been computed, $\forall i \in [1..H], \forall j \in [1..W]$, the threshold of each pixel (i, j) on its local window MxM (specified by two points $(j-\frac{M}{2}, i-\frac{M}{2})$ *and* $(j+\frac{M}{2}, i+\frac{M}{2})$) can be quickly determined by computing the values $SFgR_{ij}, Card(FgR_{ij}), SBgR_{ij}, Card(BgR_{ij})$ (3,4) as follows:

Let $l = j-\frac{M}{2}, t = i-\frac{M}{2}, r = j+\frac{M}{2}, b = i+\frac{M}{2}$,
$l: left, t: top, r: right, b: bottom$

With l, t, r, b, they define on the image a number of rectangles:

$R_1 = (1,1,l-1,t-1)$,

$R_2 = (1,1,r,t-1)$, $R_3 = (1,1,l-1,b)$,

$R_4 = (1,1,r,b)$, and $R_5 = (l,t,r,b)$

We can compute values (3, 4) of rectangle $R_5 = (l,t,r,b)$, from R_1, R_2, R_3, *and R4.*

$$R_5 = R_4 - R_3 - R_2 + R_1.$$

Thus, we have:

$$SFgR_{ij} = SFP_{b,r} - SFP_{b,l-1} - SFP_{t-1,r} + SFP_{t-1,l-1} \quad (9)$$

and

$$Card(FgR_{ij}) = CFP_{b,r} - CFP_{b,l-1} - CFP_{t-1,r} + CFP_{t-1,l-1} \quad (10)$$

$$SBgR_{ij} = SBP_{b,r} - SBP_{b,l-1} - SBP_{t-1,r} + SBP_{t-1,l-1} \quad (11)$$

and

$$Card(BgR_{ij}) = CBP_{b,r} - CBP_{b,l-1} - CBP_{t-1,r} + CBP_{t-1,l-1} \quad (12)$$

In order to reduce book-keeping cost, arrays SF, CF, SB, and CB are used from column 0, and row 0.

```
//Firstly, compute all values of SF,CF,SB,CB
For each i ∈ [1,H] do
    For each j ∈ [1,W] do
        If Img[i][j] ≤ GlobalThreshold Then
            SF[i][j] = SF[i][j-1] + SF[i-1][j] –
            SF[i-1][j-1] + Img[i][j]
            CF[i][j] = CF[i][j-1] + CF[i-1][j] –
            CF[i-1][j-1] + 1
            SB[i][j] = SB[i][j-1] + SB[i-1][j] –
            SB[i-1][j-1]
            CB[i][j] = CB[i][j-1] + CB[i-1][j] –
            CB[i-1][j-1]
        Else
            SF[i][j] = SF[i][j-1] + SF[i-1][j] –
            SF[i-1][j-1]
            BF[i][j] = BF[i][j-1] + BF[i-1][j] –
            BF[i-1][j-1]
            SB[i][j] = SB[i][j-1] + SB[i-1][j] –
            SB[i-1][j-1] + Img[i][j]
            CB[i][j] = CB[i][j-1] + CB[i-1][j] –
            CB[i-1][j-1] + 1
        End If
    End For
End For

// Then, compute threshold of each pixel and classify it
For each (i,j) ∈ [1,H]x[1,W] do
    l = j – M/2
    t = i – M/2
    r = j + M/2
    b = i + M/2
    FGg = SF[b][r] – SF[b][l-1] – SF[t-1][r] + SF[t-1][l-
1]
    CFGg = CF[b][r] – CF[b][l-1] – CF[t-1][r] + CF[t-
1][l-1]
    BGg = SB[b][r] – SB[b][l-1] – SB[t-1][r] + SB[t-
1][l-1]
    CBGg = CB[b][r] – CB[b][l-1] – CB[t-1][r] + CB[t-
1][l-1]
    LocalThreshold = (FGg/CFGg + BGg/CBGg)/2
    If Img[i][j] ≤ LocalThreshold Then
```

```
        ImgB[i][j] ← 1
    Else
        ImgB[i][j] ← 0
    End If
End For
```

Our algorithm produces the same binary image result as the original one, but its complexity does not depend on the local window size M, and the complexity is approximately $2xWxH\ (\approx O(N^2))$.

4. EXPERIMENTS AND CONCLUSIONS

4.1. Experiments

We have implemented both the original algorithm and our algorithm on MS Visual C++ 6.0 (HP Workstation X2000 Pentium IV, 1.4 GHz, 512 MB RAM, Windows XP OS), and tested them on the same image set (images size about 800x600) with variety of window size (M). The results are showed in Table 1:

M	Normal (milliseconds)	Optimized (milliseconds)
11	7687.4	621.425
15	9659.825	585.2
21	18704.61	602.9625
25	25939.4	618.8
29	32396	574.9333
33	41996.4	620.9
37	51981.65	601.125

Table 1: Comparison time of performance

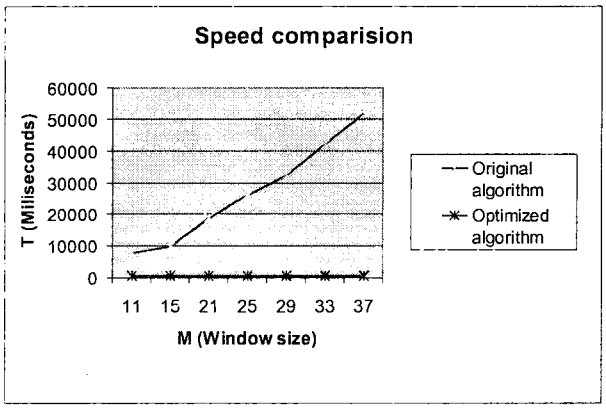

Fig 4. Comparison time of performance in graph

4.2. Conclusions

Our proposed algorithm retains the quality of the resulted binary image, but the execution speed is improved significantly. This improvement can benefit various processes in image processing or computer vision domain.

References

[1] Andreas E. Savakis, "Adaptive document image thresholding using foreground and background clustering", *Proceedings of International Conference on Image Processing ICIP98*, 1998.

[2] Xiaoyi Jiang, "Adaptive Local Threshold by Verification-Based Multithreshold Probing with Application to Vessel Detection in Retinal Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol 25, Jan 2003.

[3] Rafael C.Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall Inc, pp 595-611, 2002