Journal of Data Analysis and Information Processing, 2022, 10, 91-109
https://www.scirp.org/journal/jdaip
ISSN Online: 2327-7203
ISSN Print: 2327-7211

Fast Object Extraction and Euler Number on Block Represented Images

Iraklis M. Spiliotis*, Alexandros S. Peppas, Nikolaos D. Karampasis, Yiannis S. Boutalis

Department of Electrical and Computer Engineering, Democritus University of Thrace, GR, Xanthi, Greece
Email: *spiliot@ee.duth.gr, alexandrospeppas@hotmail.com, nikolaoskarampasis@gmail.com, ybout@ee.duth.gr

How to cite this paper: Spiliotis, I.M., Peppas, A.S., Karampasis, N.D. and Boutalis, Y.S. (2022) Fast Object Extraction and Euler Number on Block Represented Images. *Journal of Data Analysis and Information Processing*, **10**, 91-109.
https://doi.org/10.4236/jdaip.2022.102006

Received: January 2, 2022
Accepted: March 28, 2022
Published: March 31, 2022

Abstract

The identification of objects in binary images is a fundamental task in image analysis and pattern recognition tasks. The Euler number of a binary image is an important topological measure which is used as a feature in image analysis. In this paper, a very fast algorithm for the detection and localization of the objects and the computation of the Euler number of a binary image is proposed. The proposed algorithm operates in one scan of the image and is based on the Image Block Representation (IBR) scheme. The proposed algorithm is more efficient than conventional pixel based algorithms in terms of execution speed and representation of the extracted information.

Keywords

Image Block Representation, Object Detection, Hole Detection, Euler Number, Connected Components Labeling

1. Introduction

In our days, vast amounts of image and video data are generated, transmitted and analyzed, thus the development of fast algorithms able to achieve high processing rates is of great importance for many applications.

Binary images are suitable for a number of image analyses, pattern recognition, document processing, robot vision, and image based industrial applications, especially when the shape of the objects is important and the segmentation from the background is simple and without uncertainty. Object detection and localization are fundamental tasks in various applications. The Euler number E of a binary image is an important topological property that remains invariant under certain image rubber-sheet transformations, such as stretching and under scaling, rotation, or translation [1]. It is defined as the difference between the number of connected object components C and the number of holes H of the binary image, $E = C - H$. It has been used in various image processing and analysis applications such as medical image diagnosis [2], image database retrieval [3], and robot vision. Since C is the number of objects, the operations of object detection and Euler number computation are closely related.

Algorithms for object detection were developed 50 years ago [4]. Due to their mode of operation, they are called Connected Components Labeling (CCL). In recent years, new improved CCL algorithms with reduced complexity have been presented [5] [6] [7].

The CCL operation assigns a unique label to the pixels of each connected component of the image. Each connected component is a different object. After the labeling operation, the separation of objects from the image is feasible. As a result, the output of any CCL algorithm is an array of pixel labels, where each label corresponds to a different object in the image. Subsequently, the feature extraction and object classification tasks can be performed using the above labeling. Therefore, CCL is a significant operation in binary image analysis, pattern recognition, object tracking [8], and computer vision in general.

For the calculation of the connected components labeling, two kinds of algorithms have been presented. The first one is the label equivalence-based algorithms, which process the image with raster scans. A provisional label is assigned to each foreground pixel in the first scan, while the resulting label equivalences are resolved in the subsequent scans [9] [10] [11]. The second kind is the label propagation algorithms which scan the image, locate a foreground pixel, assign to it a new label and then assign this label to its connected foreground pixels [12].

A number of different approaches have been proposed for the calculation of the Euler number on binary images. Dyer [13] and Samet and Tamminen [14] proposed an algorithm based on the quadtree representation of images. Chen and Yen presented a parallel algorithm using graphs of the image [15]. Díaz-De-León *et al.* presented an algorithm using the skeleton of the image [16]. Zenzo *et al.* presented a run-based algorithm using the number of runs and 8-neighbor runs in the image [17]. Also, there are algorithms based on 2 × 2 pixel patterns called bit-quads in the image [6] [18]. Parallel implementation on a multicore computer for the computation of CCL [19] and a VLSI implementation [20] were also proposed.

Recently, He *et al.* proposed the GLC algorithm for integrating connected components labeling and Euler number computation [7]. In the GLC algorithm, a binary image is converted into a graph G, and using Euler's theorem [21], the Euler number can be calculated according to the numbers of vertices, edges, and faces. The advantage of the above GLC algorithms is that they use 16 possible patterns of the connected-component labeling masks, but they take into account only the four of them. So they achieve bigger acceleration than the previous algorithms CHE [22] and ML [6], which were also proposed previously by He *et al.* All of these algorithms compute the Euler number and the connected component labeling in one scan, simultaneously.

He *et al.* relied on Chen's and Yan's algorithm [15] to implement a new algorithm to calculate the Euler number [6]. More specifically, they transform a binary image into a square graph G, where the 8-connectivity of the neighboring pixels is utilized. Each foreground pixel in the image is considered as a vertex in the graph and an edge is added if pixels p and q are 8-neighbors. Suppose v, e, r, and c are the numbers of vertices, edges, squares, and connected components in G, respectively.

According to Euler's theorem [21], $v - e + r = c + 1$, where squares include holes, basic faces, and an infinite square outside of G. Also h and s are respectively the numbers of holes and basic faces in graph G. Then, $r = h + s + 1$. Thus, the Euler number E of G can be represented as:

$$E = c - h = v - e + s \tag{1}$$

The faces are the basic right angle triangles, each of which consists of two right-angle sides of length one, as shown in **Figure 1(a)** and **Figure 1(b)**.

The Euler number can be calculated by counting the numbers of vertices, faces, and edges by adding pixels one-by-one in the raster scan. For each pixel being added, the increments of the numbers of vertices, edges, and faces generated by adding this pixel are calculated. If the pixel is a background pixel, no new vertex, edge, or face is generated, and thus nothing needs to be done. Otherwise, the pixel is a foreground pixel and the number of vertices should be increased by one. Due to the fact that it is not convenient to compute the Euler number by directly counting the numbers of vertices, edges, and faces generated by adding a foreground pixel one-by-one [9], He et al proposed the usage of the Dv, De, and Ds, which are the increments in the numbers of vertices, edges, and faces, respectively, when the current foreground pixel is added. Thus, the corresponding change in the Euler number DE can be calculated as

$$DE = Dv - De + Ds \tag{2}$$

These increments can be calculated by using the mask patterns of **Figure 2**. In fact, the Euler number can be calculated by using the patterns of the masks of **Figure 2**. So, by using the connected component labeling the Euler number can be calculated as:

$$E = w - k \tag{3}$$

where w is the number of provisional labels, such as the number of pattern $P1$ occurrences from **Figure 2**, and k is the number of label equivalence resolutions, that is the total number of occurrences of patterns $P10$-$P12$ from **Figure 2**. The label equivalences, the Euler number and the connected components labeling can be calculated by the HCS algorithm, proposed by He *et al.* [9]. This algorithm is referred to as GLC. He *et al.* also proposed the calculation of the Euler number only, from the DE of the mask patterns in **Figure 2**.

In short, the GLC algorithm operates in one image scan considering only the foreground pixels and is the fastest pixel based CCL and Euler number algorithm to date. The execution of the GLC algorithm requires the checking of four neighboring previous pixels of the current foreground pixel.

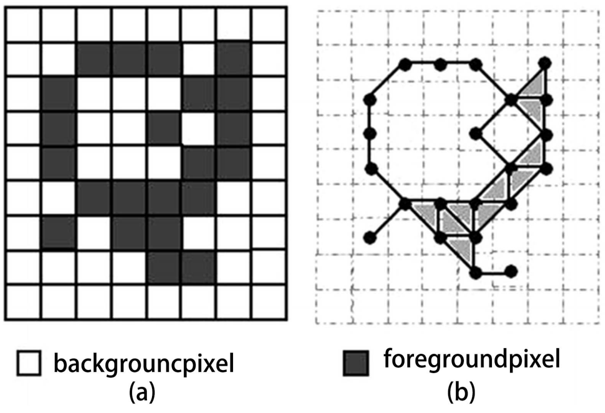


Figure 1. (a) Binary image and (b) the graph of this image.

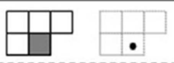
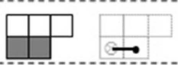
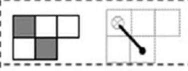
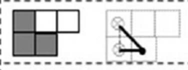
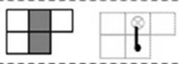
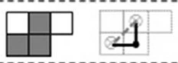
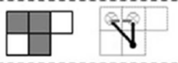
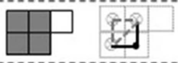
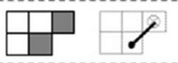
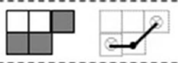
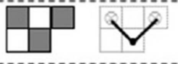
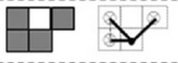
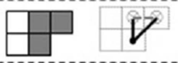
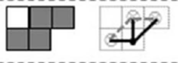
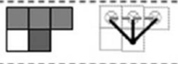
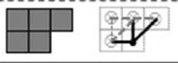
Pattem	Configuration	Δ*v*	Δ*e*	Δ*s*	Δ*E*
*P*1		1	0	0	1
*P*2		1	1	0	0
*P*3		1	1	0	0
*P*4		1	2	1	0
*P*5		1	1	0	0
*P*6		1	2	1	0
*P*7		1	2	1	0
*P*8		1	2	1	0
*P*9		1	1	0	0
*P*10		1	2	0	-1
*P*11		1	2	0	-1
*P*12		1	3	1	-1
*P*13		1	2	1	0
*P*14		1	3	2	0
*P*15		1	3	2	0
*P*16		1	3	2	0

Figure 2. The mask patterns for the execution of the GLC algorithm.

In this paper, a very fast algorithm for the detection of objects and the computation of Euler number in one scan of the image is presented. This algorithm is called Euler number in Block Represented Images (EBRI) and is based on Image Block Representation (IBR) [23], which represents the binary image as a set of non-overlapping rectangular areas with foreground pixels. Each foreground pixel belongs to one block, and the IBR is an information lossless representation equivalent to the 2D array image representation. The presented EBRI algorithm is faster than any CCL algorithm. However, the greater benefit of the EBRI algorithm is that it provides, in one scan, improved machine perception of the binary image. Any object of the image is represented as a list of coordinates, thus all the information concerning the object and its location is directly provided to the machine. In contrast, the CCL algorithms require additional image scans, in order to separate the objects according to their labels.

The rest of the paper is organized as follows. In Section 2 the EBRI algorithm is introduced and analyzed. Experimental results and comparisons are given in Section 3, while conclusions are given in Section 4.

2. The EBRI Algorithm

The proposed EBRI algorithm consists of a number of discrete tasks, which are implemented concurrently. These steps are 1) the Block Representation of the binary image, 2) the extraction of connectivity among the blocks, 3) the objects detection and representation using blocks' coordinates, and 4) the holes detection and Euler number calculation. The first step of block representation is the only step that involves pixel checking, and all the subsequent steps work directly on the derived blocks.

In order to clarify the algorithm, these tasks are initially presented separately and the combined algorithm follows.

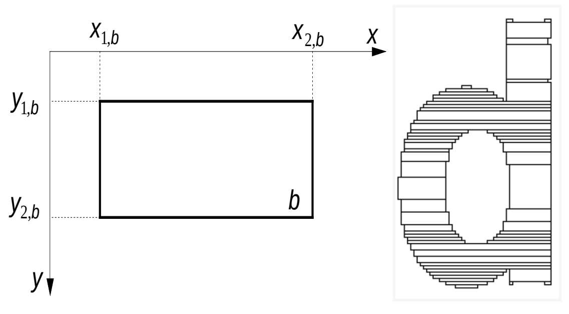
2.1. Image Block Representation

In a binary image, the foreground pixels are represented by a set of non-overlapping rectangles with edges parallel to the axes, in such a way that every object's pixel belongs to only one rectangle. These rectangles are called blocks and this representation is called Image Block Representation (IBR) [23] and is an information lossless representation of the image.

A binary image is called *block represented*, if it is represented by a set of blocks with object level, and if each pixel of the image with object value belongs to one and only one block.

A block represented image is denoted as the set of the blocks, where each block is described by four integers, the coordinates of the upper left and down right corner in vertical and horizontal axes as shown in **Figure 3(a)**. In **Figure 3(b)** the blocks that represent the image of character *d*, as extracted when using **Algorithm 1**, are illustrated. A block represented image is denoted as:

$$f(x,y) = \{b_i : i = 0,1,\cdots,b_{no} - 1\} \tag{3}$$



(a) (b)

Figure 3. (a) A block *b*. (b) Image of the character d and the derived blocks.

where b_{no} is the number of the blocks.

The IBR process requires one image scan and simple pixel checking operations; Algorithm 1 implements the IBR.

Algorithm 1. Image block representation.

```
1  For each row $y$ of the image f
2  Find object level intervals in row $y$
3  Compare intervals of row $y$ with blocks of row $y$-1
4  If an interval matches with a block, the end of the block is in row $y$
5  If an interval does not match with any block, create a new block
```

2.2. Connectivity of the Blocks

In correspondence with the 4 and *D* pixel connectivity, the following definitions clarify block connectivity schemes.

Definition 1. Two blocks are 4-connected, if there exists a pair of 4-connected pixels, one from each block.

Definition 2. Two blocks are D-connected, if they are not 4-connected and exist a pair of D-connected pixels one from each block.

The connectivity among the image blocks may be determined during the image block representation process, or directly on a given block represented image, using the following criteria [24]:

Lemma 1. Two blocks are 4-connected if their projections on one of the x or y axes are overlapped and their projections on the other axis are neighbors.

Lemma 2: Two blocks are D-connected if their projections on both axes are neighbors.

Each block *b* is stored as the structure

$$b = \{y_1,\ x_1,\ y_2,\ x_2,\ nc,\ c[\,]\} \qquad (5)$$

where y_1, x_1, y_2, x_2 the coordinates of the upper left and lower right angular points, *nc* the number of the connected blocks and *c*[] the list with their indices.

2.3. Object Detection

The connectivity information of each block allows the creation of lists of connected blocks that form the objects. A suitable data structure for storing the m objects is the vector $o[]$, where each object o_i, $i = 0, 1, \ldots, m-1$ is the data structure

$$o_i = \{bid_i[],\ nb_i\} \tag{6}$$

where the vector $bid_i[]$ that belongs to the object structure, holds the indices of the blocks that constitute the i-th object o_i, while nb_i is the number of the blocks that form o_i.

In order to extract the objects, the following procedure is used. All blocks of the binary image are examined; if the examined block (current block) has not been assigned to an object, a new object is created and the current block is assigned to the new object (current object). For the current block, every single neighboring block is being processed. If a neighbor is not assigned to an object, then it is assigned to the current object. Otherwise, if the neighbor is already assigned to a different object, a conflict between the two objects occurs and has to be resolved. The algorithm that resolves the label equivalences and presented by He *et al.* [9] is used to handle this object equivalence similar situation. **Algorithm 2** implements the extraction of objects using the block and the connections among the blocks as presented above.

Algorithm 2. Object extraction.

```
1   ono ← 0
2   i ← 0
3   while (i<bno) // for all blocks
4       if (oid[i] == -1)
5       // i-th block has not assigned to an object yet
6           oid[i] ← ono
7           o[ono].bid[0] ← i
8           o[ono].nb ← o[ono].nb+1
9           ono ← 1
10      endif
11      else if
12          m ← oid[i] // it has assigned to an object, save the object id
13      endif
14      k ← 0
15      while(k<b[i].nc)    //for all the current block's neighbors
16          neighbor ← b[i].c[k]
17          if (oid[neighbor] == -1) // neighbor not assigned to any object
18              AddNeighborToObject(i,neighbor)
19          endif
20          else if
21              n ← oid[neighbor]
22              ObjectEquivalenceResolve (m,n)
23          endif
24          k ← k+1
25      endwhile
26      i ← i+1
27  endwhile
```

The auxiliary vector *oid*[] holds the object index for each block, *i.e. oid*[*k*] is the object id in which the *k*-th block belongs. The function *AddNeighborToObject*(), assigns the neighbor block to the object of the current block and is given in **Algorithm 3**.

Algorithm 3. Function AddNeighborToObject.

```
function AddNeighborToObject (current, neighbor)
    k ← oid[current]
    o[k].b[o[k].bno] ← neighbor
    o[k].bno ← o[k].bno + 1
    oid[neighbor] ← k
end function
```

In the case of the equivalence between two objects, the lists of the blocks that make up the two involved objects, are merged into one list, as the two objects become one. The *ObjectEquivalenceResolve*() function is responsible for solving the object equivalence situation and is presented in **Algorithm 4**. The m and n are the representative objects for each object and u and v are the conflicting objects whose equivalence should be resolved.

Algorithm 4. Function ObjectEquivalenceResolve.

```
function ObjectEquivalenceResolve (u,v)
    m←rep_object[u]
    n←rep_object[v]
    if(m<n)
      $equiv\_list_m \leftarrow equiv\_list_m \cup equiv\_list_n$
      while($object\_label\_W \in equiv\_list_n$)
        rep_object[object_label_W] ←m
        object_label_W ←object_label_W+1
      end while
    end if
    else if(m>n)
      $equiv\_list_n \leftarrow equiv\_list_n \cup equiv\_list_m$
      while($object\_label\_W \in equiv\_list_m$)
        rep_object[object_label_W] ←n
        object_label_W ←object_label_W+1
      end while
    end if
end function
```

The vector *rep_object*[] stores the representative object for each object. The 2-D array $equiv\_list_n$ is the equivalence list of the object n and contains all the previous equivalent objects in relation to it.

Considering the application of **Algorithm 2** on the image of **Figure 4(a)**, the extracted objects are $o_0 = \{[0, 2], 2\}$, $o_1 = \{[1, 3, 4], 3\}$. In the example of the **Figure 4(b)** image, two objects $o_0 = \{[0, 2, 3], 3\}$, $o_1 = \{[1, 4, 5], 3\}$ are initially extracted. In the examination of the block b_3 the two north neighbor blocks b_2, b_1 belonging to different objects o_0, o_1 are detected, and the execution of function *ObjectEquivalenceResolve*() sets the equivalence $o_0 \equiv o_1$ and the blocks of o_1 are merged with the blocks of o_0. Thus, only one object is extracted from the image, that is the object $o_0 = \{[0, 2, 3, 1, 4, 5], 6\}$.

A significant feature of the proposed algorithm is that it directly provides the information for the localization of each extracted object using the coordinates of its blocks.

2.4. Hole Detection and Euler Number

Based on the previous approach for the extraction of the objects, a hole is detected when there is a block with two north neighbor blocks that belong to the same or equivalent objects. If the two objects already have the same representative object, it means that the left and right sides belong to the same object, as the top side is common.

In Figure 5 two examples of hole detection are demonstrated. In Figure 5(a) all the extracted blocks are assigned to the same object o_0. Therefore in the examination of block b_4, the two north neighbor blocks b_3, b_5 belong to the same object and a hole is detected.

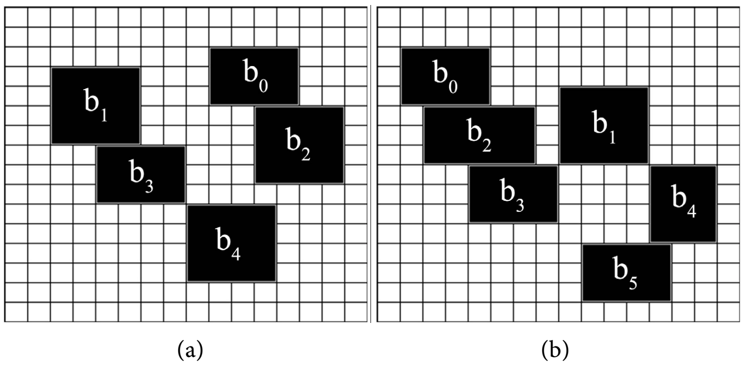


Figure 4. Two examples of the Object Extraction task.

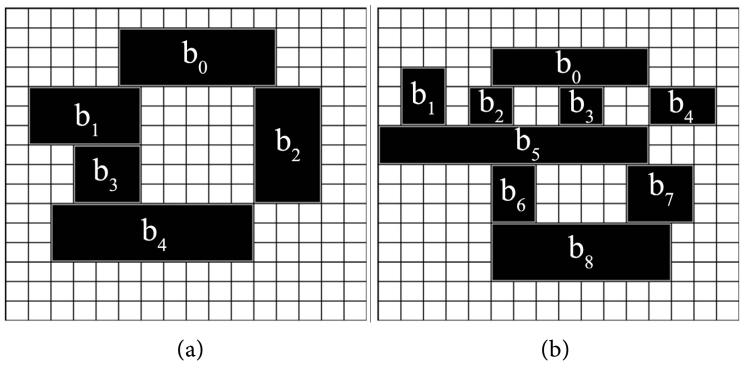


Figure 5. Two examples of hole detection. (a) The hole is detected during the examination of block b_4. (b) The first and second holes are detected during the examination of block b_5, while the third hole is detected during the examination of block b_8.

In Figure 5(b), in the examination of block b_5, the north neighbors b_1, b_2 belong to different objects that are not equivalent yet, thus no hole is detected; instead, the function ObjectEquivalenceResolve() is called and sets the equivalence of objects o_0, o_1. Also in the examination of b_5, the north neighbors b_2, b_3, b_4 belong to the same object, and the two holes are detected. In the examination of block b_8 the north neighbors b_6, b_7 belong to the same object and the third hole of the object is detected.

As the number of Objects C and the number of Holes H of a binary image are known, the Euler number can be found as $E = C - H$.

2.5. The Proposed One Scan EBRI Algorithm

The proposed algorithm's goal is to calculate simultaneously in one image scan the blocks and their location coordinates, the number of the Objects and their location coordinates, the number of holes and the Euler number. The combination of the steps described in the previous subsections constitutes the proposed EBRI Algorithm. The proposed algorithm scans each pixel of the binary image and searches for object level intervals in the image rows. When an interval is found, then the interval is assigned to an existing block or a new block is created; in the case of a new block, the block connectivity, object assignment and hole detection tasks are executed, as presented in Algorithm 5.

Algorithm 5. EBRI.

```
kp← 0 //number of blocks on previous image row
bno← 0 //number of the extracted blocks
ono← 0 //number of the extracted objects
while(y<L)
    kc← 0 //number of blocks on current row
    intervalfound←0
    j_curr← 0 //currently examined block of previous row
    x←0
    while(x<W)
        try2match←0
        if(img(y,x) AND intervalfound=0) // block's x1 coordinate
            intervalfound←1; x1←x
        end if
        if (img(y,x)=0 AND intervalfound=1)
            intervalfound←0; x2←x-1; try2match←1
        end if
        if (x=W-1 AND img(y,x) AND intervalfound=1) // row end
            x2←x; try2match←1
        end if
        if (try2match=1) //match interval with blocks of previous row
            intervalmatched←0
            j_last ←j_curr // last examined block of the previous line
            j←j_last
```

```
          while (j<kp AND x1>=b[p[j]].x1)
          j_curr ← j
          if (x1==b[p[j]].x1 AND x2==b[p[j]].x2)
          // interval matched with block from previous row
              c[kc] ←p[j]; b[p[j]].y2←y; intervalmatched←1
          end if
          j←j+1
      end while
      if (intervalmatched=0)
          NewBlock(bno, x1, x2, y) //creation of block bno
          j_last = j_curr; ii←j_last
          while (ii<kp AND x2>=b[p[ii]].x1-1)
              j_curr←ii
              // connectivity check with the block of previous row
              if((b[bno].x1<=b[p[ii]].x2+1) AND (b[bno].x2>=b[p[ii]].x1-1))
                 ConnectivityRegistration(bno, p[ii])
                 if (oid[bno] <0)    // block has no object
                   oid[bno] ← oid[p[ii]] // of left northern neighbor
                   BlockToObjectRegistration (oid[bno], p[ii])
                   //add block p[ii] into blocks that make
                   //up the object with tag bno (oid[bno])
                 end if
                 else // resolve the equivalence
                   ObjectEquivalenceResolve(oid[bno], oid[p[ii])
                 end if
              end if
              ii←ii+1
          end while
          if (oid[bno] <0)       // create new object
              oid[bno]    ← ono+1
              BlockToObjectRegistration (oid[bno], b[p[ii]])
          end if
          c[kc] ← bno+1
        end if
        kc ← kc+1
      end if
      x←x+1
    end while
    p←c ; kp←kc
    y←y+1
end while
```

The function *NewBlock*() creates a new block and is presented in Algorithm 6. Algorithm 7 presents the function *ConnectivityRegistration* (u, v) which registers the connectivity of the blocks $b[u]$ and $b[v]$. The function *BlockToObjectRegistration* (u, v) performs the registration of the v-th block $b[v]$ in the list of the blocks that constitute the u-th object $o[u]$ and is presented in Algorithm 8.

Algorithm 6. Function NewBlock.

```
function NewBlock(v, x1, x2, y)
//creates the v-th block with coordinates x1, x2, y, y
    oid[v] ← -1 //not associated with an object yet
    b[v].x1← x1
    b[v].x2← x2
    b[v].y1← y
    b[v].y2← y
    b[v].nc← 0
end function
```

Algorithm 7. Function ConnectivityRegistration.

```
function ConnectivityRegistration(u, v)
// registration of connectivity of blocks b[u] and b[v]
    b[u].c[b[u].nc] ← v
    b[u].nc← b[u].nc+1
    b[v].c[b[v].nc] ← u
    b[v].nc← b[v].nc+1
end function
```

Algorithm 8. Function BlockToObjectRegistration.

```
function ObjectRegistration(u, v)
//registers the v-th block into the list of the blocks of the u-th object
    o[u].bid[u].nb] ← v
    o[u].nb ← o[u].nb+1
end function
```

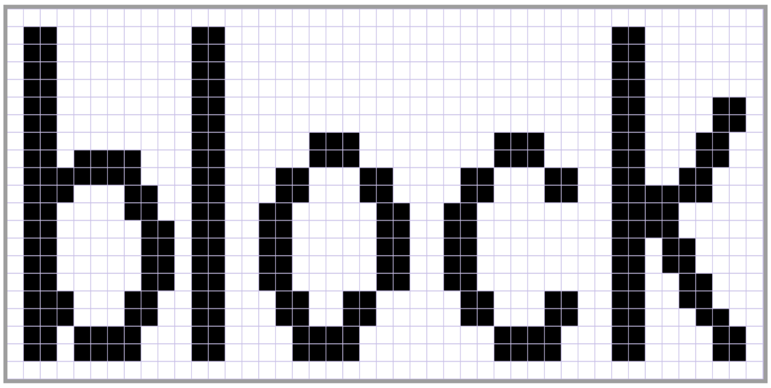
The variables W and L are the image width and length, the flag *intervalfound* detects the beginning of an object level interval in an image row, the flag *try2match* detects the completion of an object level interval and enables the matching of the interval with the blocks of the previous image row. The value 1 in the flag *intervalmatched* indicates the matching of the interval with a block of the previous image row, while the value 0 enables the creation of a new block. The vectors $p[]$, $c[]$ hold the blocks of the previous and current image row, while kp, kc are their number. The vector $oid[]$ holds the object indices for each extracted block, *i.e.* oid_k is the index of the object in which the k-th block $b[k]$ belongs to; its initial value is −1 and the negative value indicates that the block is not yet associated with an object.

3. Experimental Results

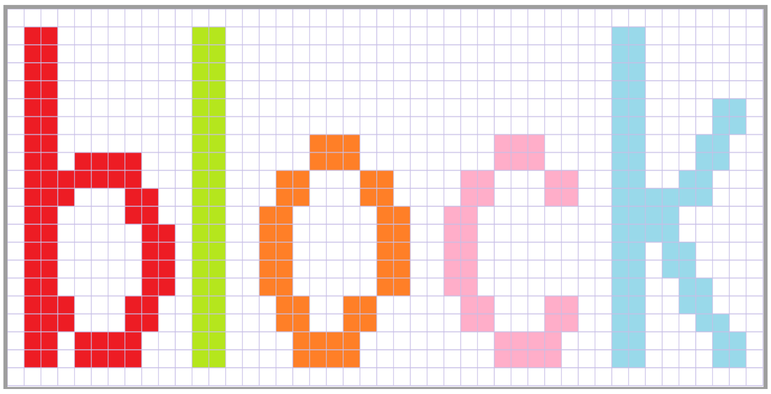
In this section, the experimental results for the execution of the proposed EBRI algorithm are presented. Also, experimental results and comparisons with the fast algorithm GLC are provided. The results provided by the two algorithms are equivalent for all the test images used, that is, the number of objects and their location as well as the number of holes are always the same. The location of the objects in the proposed method is described by the blocks' coordinates.

3.1. Qualitative Results

It has to be stressed that the EBRI algorithm provides all the necessary information of objects and their location in a compact form, in one image scan. Moreover, it requires less execution time, and usually real world images require less information than the pixel based algorithms. In order to clarify the qualitative superiority of the EBRI algorithm in comparison with CCL algorithms, consider the simplified image of Figure 6(a), which depicts text. The image of Figure 6(a) contains 5 characters or 5 objects according to the EBRI algorithm or 5 connected components represented by labels according to CCL algorithms. Figure 6(b) demonstrates the 5 connected components as extracted by the GLC algorithm. In the CCL case for the extraction of each labeled object, an additional image scan is required, so that each object pixel is copied to an image f_L, where L is the label.



(a)



(b)

Figure 6. (a) An example of a binary image used as input to GLC and EBRI algorithm. (b) The result of the GLC algorithm.

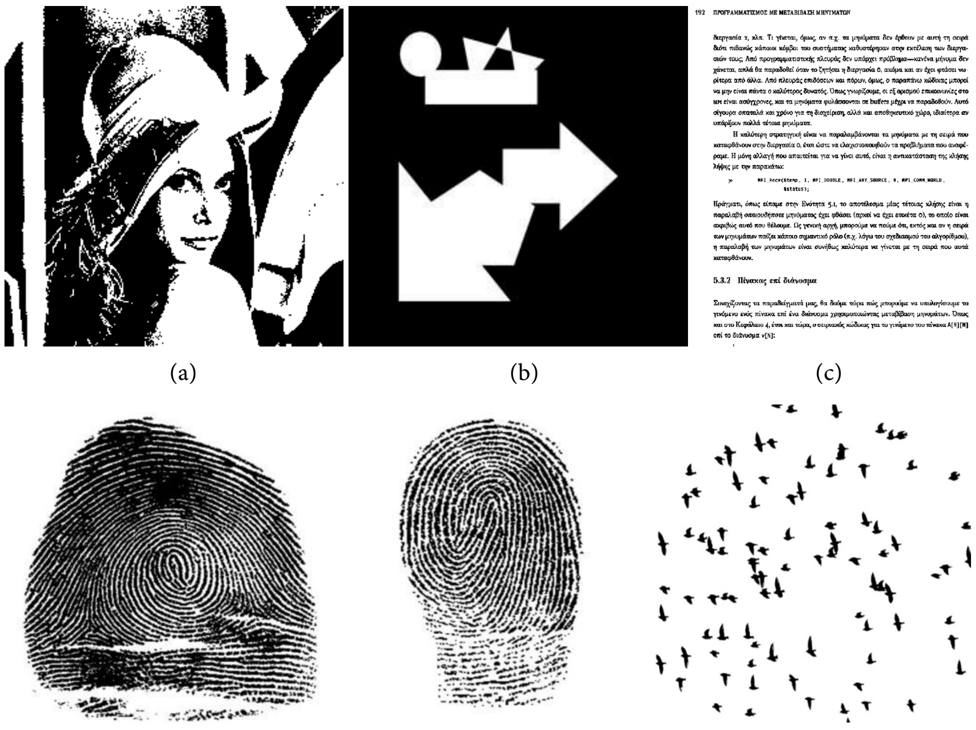
Table 1 demonstrates the information of objects and blocks as extracted by the EBRI algorithm. The image contains 5 objects, 38 blocks and each object is represented by the corresponding blocks without any necessity for an additional image scan. Obviously, this is a compact representation of the binary image, as it includes the number of objects and their localization information is determined by the coordinates of the blocks. This qualitative feature constitutes the first advantage of the proposed method in comparison with the CCL methods. The smaller execution time of EBRI complements and enhances this qualitative characteristic.

3.2. Time Complexity

For the experimental evaluation, a computer with total 8 AMD Opteron cores at 2.2GHz and 16 GB of memory was used. The operating system was Linux CentOS 7, all the programs implemented in C programming language, compiled with gcc for serial execution using one CPU core. To decrease random variation, all the execution time complexities were measured as the average of 1000 runs. Figure 7 demonstrates a sample of the test binary images in different sizes that have been used in the experiments.

Table 1. The result of EBRI algorithm for the input image of Figure 6(a).

O_0 = {11, [0, 7, 8, 14, 15, 22, 26, 27, 33, 34]}	
O_1 = {1, [1]}	
O_2 = {8, [4, 9, 10, 18, 19, 28, 29, 35]}	
O_3 = {7, [5, 11, 12, 20, 30, 31, 36]}	
O_4 = {11, [2, 3, 6, 13, 16, 21, 23, 24, 25, 32, 37]}	
b_0 = {1, 1, 8, 2, 1, [8]}	b_{19} = {11, 22, 15, 23, 2, [10, 29]}
b_1 = {1, 11, 19, 12, 0, []}	b_{20} = {11, 26, 15, 27, 2, [11, 30]}
b_2 = {1, 36, 9, 37, 1, [16]}	b_{21} = {11, 36, 12, 39, 3, [16, 23, 24]}
b_3 = {5, 42, 6, 43, 1, [6]}	b_{22} = {12, 8, 15, 9, 2, [15, 27]}
b_4 = {7, 18, 8, 20, 2, [9, 10]}	b_{23} = {13, 36, 19, 37, 1, [21]}
b_5 = {7, 29, 8, 31, 2, [11, 12]}	b_{24} = {13, 39, 14, 40, 2, [21, 25]}
b_6 = {7, 41, 8, 42, 2, [3, 13]}	b_{25} = {15, 40, 16, 41, 2, [24, 32]}
b_7 = {8, 4, 8, 7, 1, [8]}	b_{26} = {16, 1, 17, 3, 3, [17, 33, 34]}
b_8 = {9, 1, 9, 7, 2, [7, 14]}	b_{27} = {16, 7, 17, 8, 2, [22, 34]}
b_9 = {9, 16, 10, 17, 2, [4, 18]}	b_{28} = {16, 16, 17, 17, 2, [18, 35]}
b_{10} = {9, 21, 10, 22, 2, [4, 19]}	b_{29} = {16, 20, 17, 21, 2, [19, 25]}
b_{11} = {9, 27, 10, 28, 2, [5, 20]}	b_{30} = {16, 27, 17, 28, 2, [20, 6]}
b_{12} = {9, 32, 10, 33, 1, [5]}	b_{31} = {16, 32, 17, 33, 1, [36]}
b_{13} = {9, 40, 9, 41, 2, [6, 16]}	b_{32} = {17, 41, 17, 42, 2, [25, 37]}
b_{14} = {10, 1, 10, 3, 2, [8, 17]}	b_{33} = {18, 1, 19, 2, 1, [26]}
b_{15} = {10, 7, 11, 8, 2, [8, 22]}	b_{34} = {18, 4, 19, 7, 2, [26, 27]}
b_{16} = {10, 36, 10, 41, 3, [2, 13, 21]}	b_{35} = {18, 17, 19, 20, 2, [28. 29]}
b_{17} = {11, 1, 15, 2, [14, 26]}	b_{36} = {18, 29, 19, 32, 2, [30, 31]}
b_{18} = {11, 15, 15, 16, 2, [9, 28]}	b_{37} = {18, 42, 19, 43, 1, [32]}



192 ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΜΕ ΜΕΤΑΒΙΒΑΣΗ ΜΗΝΥΜΑΤΩΝ

διεργασία 2, κλπ. Τι γίνεται, όμως, αν π.χ. τα μηνύματα δεν έρθουν με αυτή τη σειρά διότι πιθανώς κάποιοι κόμβοι του συστήματος καθυστέρησαν στην εκτέλεση των διεργασιών τους; Από προγραμματιστικής πλευράς δεν υπάρχει πρόβλημα—κανένα μήνυμα δεν χάνεται, απλά θα παραδοθεί όταν το ζητήσει η διεργασία 0, ακόμα και αν έχει φτάσει νωρίτερα από άλλα. Από πλευράς επιδόσεων και πόρων, όμως, ο παραπάνω κώδικας μπορεί να μην είναι πάντα ο καλύτερος δυνατός. Όπως γνωρίζουμε, οι εξ ορισμού επικοινωνίες στο MPI είναι ασύγχρονες, και τα μηνύματα φυλάσσονται σε buffers μέχρι να παραδοθούν. Αυτό σίγουρα σπαταλά και χρόνο για τη διαχείριση, αλλά και αποθηκευτικό χώρο, ιδιαίτερα αν υπάρξουν πολλά τέτοια μηνύματα.

Η καλύτερη στρατηγική είναι να παραλαμβάνονται τα μηνύματα με τη σειρά που καταφθάνουν στην διεργασία 0, έτσι ώστε να ελαχιστοποιηθούν τα προβλήματα που αναφέραμε. Η μόνη αλλαγή που απαιτείται για να γίνει αυτό, είναι η αντικατάσταση της κλήσης λήψης με την παρακάτω:

```
30          MPI_Recv(&temp, 1, MPI_DOUBLE, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD,
                     &status);
```

Πράγματι, όπως είπαμε στην Ενότητα 5.1, το αποτέλεσμα μίας τέτοιας κλήσης είναι η παραλαβή *οποιουδήποτε* μηνύματος έχει φθάσει (αρκεί να έχει ετικέτα 0), το οποίο είναι ακριβώς αυτό που θέλουμε. Ως γενική αρχή, μπορούμε να πούμε ότι, εκτός και αν η σειρά των μηνυμάτων παίζει κάποιο σημαντικό ρόλο (π.χ. λόγω του σχεδιασμού του αλγορίθμου), η παραλαβή των μηνυμάτων είναι συνήθως καλύτερα να γίνεται με τη σειρά που αυτά καταφθάνουν.

5.3.2 Πίνακας επί διάνυσμα

Συνεχίζοντας τα παραδείγματά μας, θα δούμε τώρα πώς μπορούμε να υπολογίσουμε το γινόμενο ενός πίνακα επί ένα διάνυσμα χρησιμοποιώντας μεταβίβαση μηνυμάτων. Όπως και στο Κεφάλαιο 4, έτσι και τώρα, ο σειριακός κώδικας για το γινόμενο του πίνακα A[N][N] επί το διάνυσμα v[N]:

Figure 7. A sample of test images used in experiments. (a) Lena with size 256 × 256, (b) Shapes with size 1024 × 1024, (c) negative of Text page with size 1024 × 1024, (d) negative of Fingerprint1 with size 1024 × 1024, (e) negative of Fingerprint2 with size 1024 × 1024 and (f) negative of Birds with size 2048 × 2048 pixels.

Table 2 presents the number of blocks and the number of foreground pixels for each test image. It should be noticed that the number of blocks is significantly smaller than the number of foreground pixels. **Table 3** demonstrates the required execution times in milliseconds of the proposed EBRI algorithm and of the GLC which is the fastest CCL and Euler number calculation algorithm to date. In **Figure 8** the mean execution times of the two algorithms for a number of test images with sizes from 128 × 128 up to 2048 × 2048 pixels are presented.

Both EBRI and GLC algorithms operate in real time, but the proposed EBRI algorithm has lower time complexity than the GLC algorithm, especially when the image size increases. The EBRI algorithm locates the foreground pixels only for the extraction of object level intervals in image rows; the rest of the processing takes place at a higher level since it deals with the blocks and objects. On the contrary, the GLC algorithm deals with 4 neighbor pixels for each foreground pixel, which means memory accesses and execution of logical operations. Since the number of blocks is significantly reduced in comparison to the foreground pixels as demonstrated in **Table 2**, the complexity of the EBRI algorithm is lower than the complexity of the GLC algorithm.

It should be noticed that execution times of the GLC algorithm are measured only for the creation of the label images (see **Figure 6(b)**), where each label represents a different object. The label images are not directly useful for any image analysis and object recognition task. In order to achieve object recognition, object segmentation and perhaps the creation of a bounding box for each object are usually required; these procedures require one or more image scans, which are not taken into account in Table 3 and Figure 8.

Table 2. The size, number of blocks and number of foreground pixels of test images.

Image	Size	Blocks	Foreground Pixels
Lena	256 × 256	2102	33,513
Shapes	1024 × 1024	927	239,421
Text page	1024 × 1024	18,753	99,606
Fingerprint1	1024 × 1024	13,812	428,616
Fingerprint2	1024 × 1024	7362	181,147
Birds	2048 × 2048	4429	135,158

Table 3. Execution time (msec) of EBRI and GLC algorithms for test images.

Image	Size	EBRI	GLC
Lena	256 × 256	1.914	1.473
Shapes	1024 × 1024	7.809	9.791
Text page	1024 × 1024	15.840	16.640
Fingerprint1	1024 × 1024	14.010	16.620
Fingerprint2	1024 × 1024	13.430	14.440
Birds	2048 × 2048	21.005	24.630

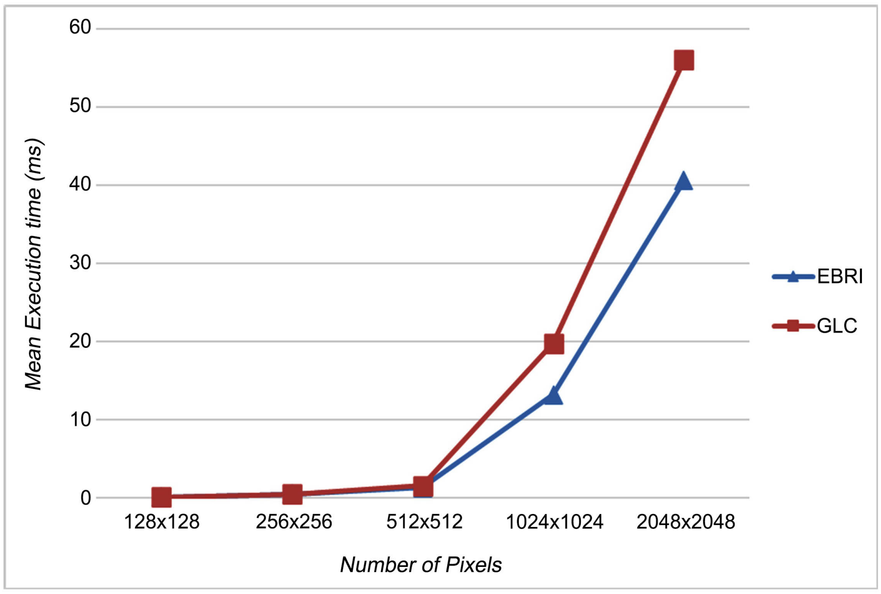


Figure 8. Mean execution time (in msec) for images with different number of pixels.

4. Conclusions

In this paper the EBRI algorithm is presented; which aims to provide block representation of binary images, connectivity information among the blocks, extraction of objects in a compact form using blocks, holes detection, and Euler number computation in one image scan.

From the experimental results, the proposed EBRI algorithm is evaluated as faster than the GLC algorithm and achieves a higher processing rate.

Additionally to the execution time, there is a qualitative perspective that is also significant. The proposed algorithm produces results in the compact form of blocks and objects, which provide to a vision system the perception of image areas that are greater than a pixel. Various feature extraction fast algorithms on block represented images have been presented in the past, specifically the skeletonization [24], the moment computation on binary images [23] and on gray images [25] [26] and the Hough transform [27].

On the other hand, the GLC algorithm requires an additional image scan to extract each labeled object and copy each object pixel to a new image fL, where L is the label of the object pixels. In contrast, the proposed EBRI algorithm extracts all objects in the same image in one scan and in a shorter time. This feature is very important for real-time pattern recognition applications.

The development of more feature extraction algorithms using block representation is a direction of our research. Also, the parallel implementation of the related algorithms is another interesting research direction. Recently, the parallel implementation of the IBR algorithm [28] and the parallel computation of discrete orthogonal moments on block represented images [29] using OpenMP API on shared memory computers have been presented. The development of a parallel EBRI algorithm is also a future research direction.

Acknowledgements

This work was partially supported by Special Account for Research Funds, Democritus University of Thrace, Project KE 82319 and Project KE 82742.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Gonzalez, R.C. and Woods, R.E. (2018) Digital Image Processing. 4th Edition, Pearson Education, London.

[2] Hashizume, A., Suzuki, R. and Yokouchi, H. (1990) An Algorithm of Automated RBC Classification and Its Evaluation. *Japanese Journal of Medical Electronics and Biological Engineering*, **28**, 25-32.

[3] Bishnu, A., Bhattacharya, B.B., Kundu, M.K., Murthy, C.A. and Acharya, T. (2005) Euler Vector for Search and Retrieval of Gray-Tone Images. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **35**, 801-812.

https://doi.org/10.1109/TSMCB.2005.846642

[4] Rosenfeld, A. and Kak, A.C. (1982) Digital Picture Processing. 2nd Edition, Academic Press, San Diego, CA.

[5] Grana, C., Borghesani, D. and Cucchiara, R. (2010) Optimized Block-Based Connected Components Labeling with Decision Trees. *IEEE Transactions on Image Processing*, **19**, 1596-1609. https://doi.org/10.1109/TIP.2010.2044963

[6] He, L.F., Zhao, X., Yao, B., *et al.* (2017) A Combinational Algorithm for Connected-Component Labeling and Euler Number Computing. *Journal of Real-Time Image Processing*, **13**, 703-712. https://doi.org/10.1007/s11554-014-0433-y

[7] He, L.F., Zhao, X., Yao, B., *et al.* (2018) A Fast Algorithm for Integrating Connected-Component Labeling and Euler Number Computation. *Journal of Real-Time Image Processing*, **15**, 709-723. https://doi.org/10.1007/s11554-015-0499-1

[8] Dung, L., Wang, S. and Wu, Y. (2018) A Multiple Random Feature Extraction Algorithm for Image Object Tracking. *Journal of Signal and Information Processing*, **9**, 63-71. https://doi.org/10.4236/jsip.2018.91004

[9] He, L.F., Chao, Y.Y. and Suzuki, K. (2008) A Run-Based Two-Scan Labeling Algorithm. *IEEE Transactions on Image Processing*, **17**, 749-756. https://doi.org/10.1109/TIP.2008.919369

[10] He, L.F., Chao, Y.Y. and Suzuki, K. (2009) Fast Connected-Component Labeling. *Pattern Recognition*, **42**, 1977-1987. https://doi.org/10.1016/j.patcog.2008.10.013

[11] He, L.F., Chao, Y.Y. and Suzuki, K. (2010) An Efficient First-Scan Method for Label Equivalence-Based Labeling Algorithms. *Pattern Recognition Letters*, **31**, 28-35. https://doi.org/10.1016/j.patrec.2009.08.012

[12] Hu, Q., Qian, G. and Nowinski, W.L. (2005) Fast Connected-Component Labelling in Three-Dimensional Binary Images based on Iterative Recursion. *Computer Vision and Image Understanding*, **99**, 414-434. https://doi.org/10.1016/j.cviu.2005.04.001

[13] Dyer, C.R. (1980) Computing the Euler Number of an Image from Its Quadtree. *Computer Graphics and Image Processing*, **13**, 270-276. https://doi.org/10.1016/0146-664X(80)90050-7

[14] Samet, H. and Tamminen, H. (1985) Computing Geometric Properties of Images Represented by Linear Quadtrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **7**, 229-240. https://doi.org/10.1109/TPAMI.1985.4767646

[15] Chen, M.-H. and Yan, P.-F. (1988) A Fast Algorithm to Calculate the Euler Number for Binary Images. *Pattern Recognition Letters*, **8**, 295-297. https://doi.org/10.1016/0167-8655(88)90078-5

[16] Díaz-De-León, J.L. and Sossa-Azuela, J.H. (1996) On the Computation of the Euler Number of a Binary Object. *Pattern Recognition*, **29**, 471-476. https://doi.org/10.1016/0031-3203(95)00098-4

[17] Zenzo, S., Cinque, L. and Levialdi, S. (1996) Run-Based Algorithms for Binary Image Analysis and Processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **18**, 83-89. https://doi.org/10.1109/34.476016

[18] Gray, S.B. (1971) Local Properties of Binary Images in Two Dimensions. *IEEE Transactions on Computers*, **20**, 551-561. https://doi.org/10.1109/T-C.1971.223289

[19] Cabaret, L., Lacassagne, L. and Etiemble, D. (2018) Parallel Light Speed Labeling: An Efficient Connected Component Algorithm for Labeling and Analysis on Mul-

ti-Core Processors. *Journal of Real-Time Image Processing*, **15**, 173-196. https://doi.org/10.1007/s11554-016-0574-2

[20] Dey, S., Bhattacharya, B.B., Kundu, M.K. and Acharya, T. (2000) A Fast Algorithm for Computing the Euler Number of an Image and Its VLSI Implementation. *Proceedings of* 13*th International Conference on VLSI Design*, Calcutta, 3-7 January 2000, 330-335. https://doi.org/10.1109/ICVD.2000.812628

[21] West, D.B. (2001) Introduction to Graph Theory. 2nd Edition, Prentice Hall, Hoboken.

[22] He, L.F., Chao, Y.Y. and Suzuki, K. (2013) An Algorithm for Connected-Component Labeling, Hole Labeling and Euler Number Computing. *Journal of Computer Science and Technology*, **28**, 468-478. https://doi.org/10.1007/s11390-013-1348-y

[23] Spiliotis, I. and Mertzios, B. (1998) Real-time Computation of Two-Dimensional Moments on Binary Images Using Image Block Representation. *IEEE Transactions on Image Processing*, **7**, 1609-1615. https://doi.org/10.1109/83.725368

[24] Spiliotis I.M. and Mertzios, B.G. (1997) A Fast Skeleton Algorithm on Block Represented Binary Images. 13*th International Conference on Digital Signal Processing*, Santorini, 2-4 July, 1997, 675-678.

[25] Spiliotis, I. and Boutalis, Y. (2011) Parameterized Real-Time Moment Computation on Gray Images Using Block Techniques. *Journal of Real-Time Image Processing*, **6**, 81-89. https://doi.org/10.1007/s11554-009-0142-0

[26] Spiliotis, I.M., Karampasis, N.D. and Boutalis, Y.S. (2020) Fast Computation of Hahn Moments on Gray Images Using Block Representation. *Journal of Electronic Imaging*, **29**, Article ID: 013020. https://doi.org/10.1117/1.JEI.29.1.013020

[27] Gatos, B., Perantonis, S. and Papamarkos, N. (1996) Accelerated Hough Transform Using Rectangular Block Decomposition. *Electronic Letters*, **32**, 730-732. https://doi.org/10.1049/el:19960510

[28] Spiliotis, I.M., Bekakos, M.P. and Boutalis, Y.S. (2020) Parallel Implementation of the Image Block Representation Using OpenMP. *Journal of Parallel and Distributed Computing*, **137**, 134-147. https://doi.org/10.1016/j.jpdc.2019.11.006

[29] Spiliotis, I.M., Sitaridis, C. and Bekakos, M.P. (2021) Parallel Computation of Discrete Orthogonal Moment on Block Represented Images Using OpenMP. *International Journal of Parallel Programming*, **49**, 440-446. https://doi.org/10.1007/s10766-021-00713-2