A (Practical Training Report/Seminar Report) on

An Efficient Scan Algorithm for Block-Based Connected Component Labeling

undergone at

National Institute of Technology Karnataka

under the guidance of

Mrs. Uma Priya

Submitted by

Aparna R Joshi
14CO204
VII Sem B.Tech (CSE)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER ENGINEERING

Department of Computer Science & Engineering Technology
National Institute of Technology Karnataka, Surathkal.

March 2017

ABSTRACT

The paper provides an improved block-based connected-component labeling algorithm that enhances the labeling speed. The algorithm makes use of a block-based scan mask to consider the neighbouring operations and so the proposed algorithm successfully decreases the scan mask from the original 20 pixels to 10 pixels. The simplifying neighborhood operations also produce the decision tables of the new scan mask for the block-based connected-component labeling algorithm. Furthermore, the proposed algorithm efficiently integrates the new block-based scan mask into the original scan through two procedures for different situations in order to enhance performance and to reduce unnecessary memory access. This greatly simplifies the pixel locations of the block-based scan mask. Furthermore, the algorithm significantly reduces the number of leaf nodes and depth levels required in the binary decision tree. A relabelling is then performed to update the provisional labels.

A comparison is also made of the proposed block-based scan with the Suzuki algorithm. The Suzuki algorithm scans through the image in forward and backward directions using set of masks shown later.

The proposed algorithm efficiently uses binary decision trees to label the components and uses a single mask and *can* thus obtain better performance than Suzuki.

TABLE OF CONTENTS

TECHNOLOGIES USED **4**

WORK DONE **5**

Pseudo Code for Block Based 6

CODE **8**

RESULTS AND SNAPSHOTS **19**

Comparison Graph 19

Block-based 19

Suzuki 20

CONCLUSION **22**

REFERENCES **22**

1. TECHNOLOGIES USED

While the authors of the paper use Microsoft Visual Studio 2010 and implement their codes in C++, we use C language to simplify implementation and to observe the binary decisions clearly. Also since the focus of the paper is the efficiency of the algorithm, the inputs and outputs are matrices (representing the pixels of the images, 1 for foreground, and 0 for background pixels) to satisfy our purpose. The output matrix exudes the connected components by labelling them in increasing order of integers starting from 1. Pixels belonging to the same component are given the same label (or an integer value).

MATLAB can be used to convert images to binary files (of 0's and 1's):

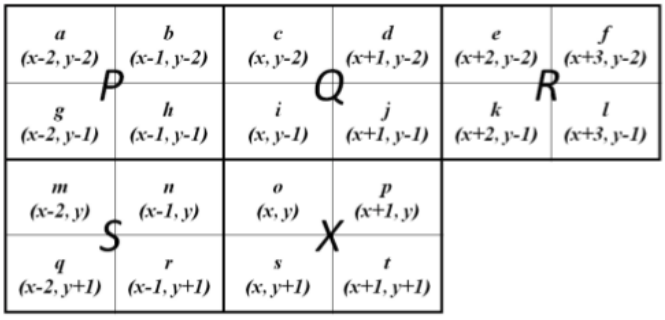
```
image = im2bw('test.png');
```

The matrix can then be saved as a text file, and the program will generate a matrix from the input text file. The output obtained will be a matrix. Comparison is made for different number_of_pixels between block based and suzuki's algorithm.

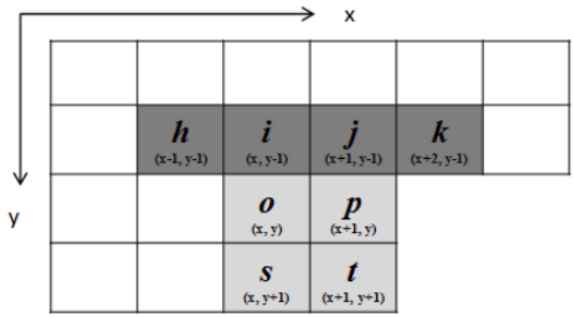
A graph is plotted showing comparison between Suzuki and Block-based labelling using matplotlib library of Python.

2. WORK DONE

Connected-component labeling algorithms form the basis of research in areas of computer and machine vision that involve locating objects for visual applications. The mask used by the BBDT algorithm is as follows:



The algorithm proposed selects the 10 pixels h, i, j, k, n, o, p, r, s, and t from the 20 obtained in the block-based scan mask.



Two checking sequences are designed and two resulting binary decision trees, called “procedure 1,” and use “procedure 2” to apply suitable block-connected relationships individually. In terms of the block-connected relationship, the connectivity between blocks S and X is judged according to pixels n, r, o, and s. When pixels n and r are background pixels, the proposed algorithm directly limits the consideration of the block-connected relationship between block X and blocks P, Q, and R in procedure 1. Otherwise, the proposed algorithm considers block-connected relationships between block X and blocks S, P, Q, and R in procedure 2. The actions performed depending on the cases:

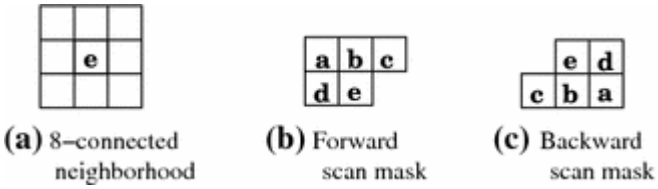
- Assigning same provisional labels of some block to the block under consideration.
- Resolving the conflict.
- Incrementing the serial number and assigning a new provisional label to the block under consideration.
- Merging of two blocks.

PSEUDO CODE FOR BLOCK BASED

```
1.  Provisional label = 1;
2.  for(y = 0; y<=H; y+=2) // begin the first-scan method
3.    for(x = 0; x<=W; x+=2) {
4.              Procedure1;
5.              while(keepinprocedure2==true&&x+2<W) {
6.                      x+=2;
7.                      Procedure2;
8.               }
9.       }
10.     Update provisional labels to representative labels in the second-scan method;
end
```

At the end of the raster scan over an image, provisional labels are assigned to all foreground pixels by applying procedure 1 or procedure 2 in the first-scan method. Having collected all

provisional labels, the second-scan method is used to sequentially update all provisional labels block by block to representative labels in the labeling table. On the other hand, the mask used by Suzuki's algorithm:



(a) 8–connected neighborhood **(b)** Forward scan mask **(c)** Backward scan mask

The forward scan procedure assigns a temporary label to the pixels using the forward mask. Then, the backward scan procedure re-labels them. Also, it stores the label equivalences in an additional one dimensional array called label connection table. The provisional labels propagate through the image as well as the label connection table that reduces the number of scans needed to complete the labeling. In the first scan, the Suzuki algorithm assigns a provisional label to each pixel at position "e" according to the following equation:

$$g(x,y)=\begin{cases} F_B & \text{if } b(x,y)=F_B, \\ m,(m=m+1) & \text{if } \forall\{i,j\in M_s\}g(x-i,y-j)=F_B, \\ T_{\min}(x,y) & \text{Otherwise} \end{cases}$$

$$T_{\min}(x,y)=\min\left[\{T[g(x-i,y-j)]\mid i,j\in M_S\}\right]$$

where F_B indicates the background pixels, g(x,y) stores the provisional labels, T[m] is the label connection table and M_S the region of the mask except the object pixel. Also, the label connection table T[m] is updated at the same time as g(x,y) according to the following equation:

$$\begin{cases} \text{non-operation} & \text{if } b(x,y)=F_B, \\ T[m]=m & \text{if } \forall\{i,j\in M_s\}g(x-i,y-j)=F_B, \\ T[g(x-i,y-j)]=T_{\min}(x,y) & \text{if } g(x-i,y-i)\neq F_B. \end{cases}$$

Thus the forward and backward scans will be performed repeatedly and alternately while the following condition is not satisfied:

$$g(x-i,y-j)\neq T_{\min}(x,y) \quad \text{if } g(x-i,y-j)\neq F_B,$$

$$i,j\in M_S.$$

PSEUDO CODE FOR SUZUKI

The conceptual algorithm is given as follows :

```
1. Algorithm_Suzuki (data)
2.       First pass
3.           for row in data
4.               for column in row
5.                   assign a label to data[row][col] using Forward Mask
6.       Next passes
7.          while there is a change in labels do
8.              for row in data
9.                for column in row
10.                   update the labels[row][column] using Backward Mask
11.             for row in data
12.              for column in row
13.                  update the labels[row][column] using Forward Mask
14. return labels
```

Both the algorithms are implemented in C and results are obtained and compared.

3. CODE

```
#include<stdio.h>
#include <time.h>
#define ll long long
ll a[10000][10000],pi,pj,qi,qj,ri,rj,si,sj,xi,xj;
ll provLabel=1;
ll max(ll a,ll b) {
	return a>b?a:b;
}
ll min(ll a,ll b) {
	return a>b?b:a;
}
void printMatrix(ll a[][10000],ll m,ll n) {
	ll i,j;
	for(i=2;i<m+2;i++)
	{
		for(j=2;j<n+2;j++)
			printf("%lld ",a[i][j]);
		printf("\n");
	}
```

```
}
void action11() {
	if(a[xi][xj]!=0) a[xi][xj]=provLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=provLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=provLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=provLabel;
	provLabel++;
}
void action12() {
	int pLabel = max(max(a[pi][pj],a[pi+1][pj]),max(a[pi][pj+1],a[pi+1][pj+1]));
	if(a[xi][xj]!=0) a[xi][xj]=pLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=pLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=pLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=pLabel;
}
void action13() {
	int qLabel = max(max(a[qi][qj],a[qi+1][qj]),max(a[qi][qj+1],a[qi+1][qj+1]));
	if(a[xi][xj]!=0) a[xi][xj]=qLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=qLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=qLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=qLabel;
}
void action14() {
	int rLabel = max(max(a[ri][rj],a[ri+1][rj]),max(a[ri][rj+1],a[ri+1][rj+1]));
	if(a[xi][xj]!=0) a[xi][xj]=rLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=rLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=rLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=rLabel;
}
void action15() {
	int pLabel = max(max(a[pi][pj],a[pi+1][pj]),max(a[pi][pj+1],a[pi+1][pj+1]));
	if(a[xi][xj]!=0) a[xi][xj]=pLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=pLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=pLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=pLabel;
	if(a[qi][qj]!=0) // resolve the conflict a[qi][qj]=pLabel;
	if(a[qi+1][qj]!=0) a[qi+1][qj]=pLabel;
	if(a[qi][qj+1]!=0) a[qi][qj+1]=pLabel;
	if(a[qi+1][qj+1]!=0) a[qi+1][qj+1]=pLabel;
}
void action16() {
	int pLabel = max(max(a[pi][pj],a[pi+1][pj]),max(a[pi][pj+1],a[pi+1][pj+1]));
```

```
        if(a[xi][xj]!=0) a[xi][xj]=pLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=pLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=pLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=pLabel;
        if(a[ri][rj]!=0) // resolve the conflict a[ri][rj]=pLabel;
        if(a[ri+1][rj]!=0) a[ri+1][rj]=pLabel;
        if(a[ri][rj+1]!=0) a[ri][rj+1]=pLabel;
        if(a[ri+1][rj+1]!=0) a[ri+1][rj+1]=pLabel;
}
void action17() {
        int qLabel = max(max(a[qi][qj],a[qi+1][qj]),max(a[qi][qj+1],a[qi+1][qj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=qLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=qLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=qLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=qLabel;
        if(a[ri][rj]!=0) // resolve the conflict a[ri][rj]=qLabel;
        if(a[ri+1][rj]!=0) a[ri+1][rj]=qLabel;
        if(a[ri][rj+1]!=0) a[ri][rj+1]=qLabel;
        if(a[ri+1][rj+1]!=0) a[ri+1][rj+1]=qLabel;
}
void action21() {
        int sLabel = max(max(a[si][sj],a[si+1][sj]),max(a[si][sj+1],a[si+1][sj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=sLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=sLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=sLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=sLabel;
}
void action22() {
        int sLabel = max(max(a[si][sj],a[si+1][sj]),max(a[si][sj+1],a[si+1][sj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=sLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=sLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=sLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=sLabel;
        // Resolve the conflict with block P by assigning the provisional label of block S
        if(a[pi][pj]!=0) a[pi][pj]=sLabel;
        if(a[pi+1][pj]!=0) a[pi+1][pj]=sLabel;
        if(a[pi][pj+1]!=0) a[pi][pj+1]=sLabel;
        if(a[pi+1][pj+1]!=0) a[pi+1][pj+1]=sLabel;
}
void action23()
{
        //Assign the same provision label as block S for the current block X.
```

```
        int sLabel = max(max(a[si][sj],a[si+1][sj]),max(a[si][sj+1],a[si+1][sj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=sLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=sLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=sLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=sLabel;
        //Resolve the conflict with block Q by assigning the provisional label of block S.
        if(a[qi][qj]!=0) a[qi][qj]=sLabel;
        if(a[qi+1][qj]!=0) a[qi+1][qj]=sLabel;
        if(a[qi][qj+1]!=0) a[qi][qj+1]=sLabel;
        if(a[qi+1][qj+1]!=0) a[qi+1][qj+1]=sLabel;
}
void action24() {
        //Assign the same provision label as block S for the current block X.
        int sLabel = max(max(a[si][sj],a[si+1][sj]),max(a[si][sj+1],a[si+1][sj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=sLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=sLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=sLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=sLabel;
        //Resolve the conflict with block R by assigning the provisional label of block S
        if(a[ri][rj]!=0) a[ri][rj]=sLabel;
        if(a[ri+1][rj]!=0) a[ri+1][rj]=sLabel;
        if(a[ri][rj+1]!=0) a[ri][rj+1]=sLabel;
        if(a[ri+1][rj+1]!=0) a[ri+1][rj+1]=sLabel;
}
void action25() {
        //Assign the same provision label as block S for the current block X.
        int sLabel = max(max(a[si][sj],a[si+1][sj]),max(a[si][sj+1],a[si+1][sj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=sLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=sLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=sLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=sLabel;
        //Resolve the conflict with blocks P and Q by assigning the provisional label of
block S
        if(a[qi][qj]!=0) a[qi][qj]=sLabel;
        if(a[qi+1][qj]!=0) a[qi+1][qj]=sLabel;
        if(a[qi][qj+1]!=0) a[qi][qj+1]=sLabel;
        if(a[qi+1][qj+1]!=0) a[qi+1][qj+1]=sLabel;
        if(a[pi][pj]!=0) a[pi][pj]=sLabel;
        if(a[pi+1][pj]!=0) a[pi+1][pj]=sLabel;
        if(a[pi][pj+1]!=0) a[pi][pj+1]=sLabel;
        if(a[pi+1][pj+1]!=0) a[pi+1][pj+1]=sLabel;
}
```

```
void action26() {
	//Assign the same provision label as block S for the current block X.
	int sLabel = max(max(a[si][sj],a[si+1][sj]),max(a[si][sj+1],a[si+1][sj+1]));
	if(a[xi][xj]!=0) a[xi][xj]=sLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=sLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=sLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=sLabel;
	//Resolve the conflict with blocks P and R by assigning the provisional label of
block S
	if(a[pi][pj]!=0) a[pi][pj]=sLabel;
	if(a[pi+1][pj]!=0) a[pi+1][pj]=sLabel;
	if(a[pi][pj+1]!=0) a[pi][pj+1]=sLabel;
	if(a[pi+1][pj+1]!=0) a[pi+1][pj+1]=sLabel;
	if(a[ri][rj]!=0) a[ri][rj]=sLabel;
	if(a[ri+1][rj]!=0) a[ri+1][rj]=sLabel;
	if(a[ri][rj+1]!=0) a[ri][rj+1]=sLabel;
	if(a[ri+1][rj+1]!=0) a[ri+1][rj+1]=sLabel;
}
void action27() {
	//Assign the same provision label as block S for the current block X.
	int sLabel = max(max(a[si][sj],a[si+1][sj]),max(a[si][sj+1],a[si+1][sj+1]));
	if(a[xi][xj]!=0) a[xi][xj]=sLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=sLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=sLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=sLabel;
	//Resolve the conflict with blocks Q and R by assigning the provisional label of
block S
	if(a[ri][rj]!=0) a[ri][rj]=sLabel;
	if(a[ri+1][rj]!=0) a[ri+1][rj]=sLabel;
	if(a[ri][rj+1]!=0) a[ri][rj+1]=sLabel;
	if(a[ri+1][rj+1]!=0) a[ri+1][rj+1]=sLabel;
	if(a[qi][qj]!=0) a[qi][qj]=sLabel;
	if(a[qi+1][qj]!=0) a[qi+1][qj]=sLabel;
	if(a[qi][qj+1]!=0) a[qi][qj+1]=sLabel;
	if(a[qi+1][qj+1]!=0) a[qi+1][qj+1]=sLabel;
}
void action28() {
	//Plus serial number and New a provisional label for block X as temporary label
	if(a[xi][xj]!=0) a[xi][xj]=provLabel;
	if(a[xi+1][xj]!=0) a[xi+1][xj]=provLabel;
	if(a[xi][xj+1]!=0) a[xi][xj+1]=provLabel;
	if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=provLabel;
```

```
        provLabel++;
}
void action29() {
        //Assign the same provision label as block Q for the current block X
        int qLabel = max(max(a[qi][qj],a[qi+1][qj]),max(a[qi][qj+1],a[qi+1][qj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=qLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=qLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=qLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=qLabel;
}
void action210() {
        //Assign the same provision label as block R for the current block X
        int rLabel = max(max(a[ri][rj],a[ri+1][rj]),max(a[ri][rj+1],a[ri+1][rj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=rLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=rLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=rLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=rLabel;
}
void action211() {
        //Assign the provisional label by block Q then merge blocks Q and R
        int qLabel = max(max(a[qi][qj],a[qi+1][qj]),max(a[qi][qj+1],a[qi+1][qj+1]));
        if(a[xi][xj]!=0) a[xi][xj]=qLabel;
        if(a[xi+1][xj]!=0) a[xi+1][xj]=qLabel;
        if(a[xi][xj+1]!=0) a[xi][xj+1]=qLabel;
        if(a[xi+1][xj+1]!=0) a[xi+1][xj+1]=qLabel;

        if(a[ri][rj]!=0) a[ri][rj]=qLabel;
        if(a[ri+1][rj]!=0) a[ri+1][rj]=qLabel;
        if(a[ri][rj+1]!=0) a[ri][rj+1]=qLabel;
        if(a[ri+1][rj+1]!=0) a[ri+1][rj+1]=qLabel;
}
void p1c1() {
        if(a[qi+1][qj]!=0) {
                if(a[qi+1][qj+1]!=0) action13();
                else {
                        if(a[ri+1][rj]!=0) action17();
                        else action13();
                }
        }
        else
        {
                if(a[qi+1][qj+1]!=0) {
```

```
                        if(a[pi+1][pj+1]!=0) action15();
                        else action13();
                }
                else
                {
                        if(a[pi+1][pj+1]!=0) {
                                if(a[ri+1][rj]!=0) action16();
                                else action12();
                        }
                        else {
                                if(a[ri+1][rj]!=0) action14();
                                else action11();
                        }
                }
        }
        return;
}
void p1c2() {
        if(a[qi+1][qj]!=0) action13();
        else {
                if(a[pi+1][pj+1]!=0) {
                        if(a[qi+1][qj+1]!=0) action15();
                        else action12();
                }
                else {
                        if(a[qi+1][qj+1]!=0) action13();
                        else action11();
                }
        }
        return;
}
void p1c3() {
        if(a[qi+1][qj+1]!=0) action13();
        else {
                if(a[qi+1][qj]!=0) {
                        if(a[ri+1][rj]!=0) action17();
                        else action13();
                } else {
                        if(a[ri+1][rj]!=0) action14();
                        else action11();
                }
        }
```

```
	return;
}
void p1c4() {
	action11();
	return;
}
void p1c5() {
	action11();
	return;
}
void p1c6() {
	// do nothing
	return;
}
void procedure1() {
	if(a[xi][xj]!=0) {
		if(a[xi][xj+1]!=0) p1c1();
		else p1c2();
	}
	else if(a[xi][xj+1]!=0) p1c3();
	else if(a[xi+1][xj]!=0) p1c4();
	else if(a[xi+1][xj+1]!=0) p1c5();
	else p1c6();
}
void p2c1()
{
	if(a[qi+1][qj]!=0) {
		if(a[qi+1][qj+1]!=0) action23();
		else  {
			if(a[ri+1][rj]!=0) action27();
			else action23();
		}
	}
	else {
		if(a[qi+1][qj+1]!=0) {
			if(a[pi+1][pj+1]!=0) action25();
			else action23();
		} else {
			if(a[pi+1][pj+1]!=0) {
				if(a[ri+1][rj]!=0) action26();
				else action22();
			} else {
```

```
                        if(a[ri+1][rj]!=0) action24();
                        else action21();
                }
        }
        return;
}
void p2c2() {
        if(a[qi+1][qj]!=0) action23();
        else {
                if(a[pi+1][pj+1]!=0) {
                        if(a[qi+1][qj+1]!=0) action25();
                        else action22();
                } else {
                        if(a[qi+1][qj+1]!=0) action23();
                        else action21();
                }
        }
        return;
}
void p2c3() {
        if(a[qi+1][qj+1]!=0) action23();
        else {
                if(a[qi+1][qj]!=0) {
                        if(a[ri+1][rj]!=0) action27();
                        else action23();
                } else {
                        if(a[ri+1][rj]!=0) action24();
                        else action21();
                }
        }
        return;
}
void p2c4() {
        if(a[qi+1][qj+1]!=0) action29();
        else {
                if(a[qi+1][qj]!=0) {
                        if(a[ri+1][rj]!=0) action211();
                        else action29();
                } else {
                        if(a[ri+1][rj]!=0) action210();
                        else action28();
```

```
                }
        }
        return;
}
void p2c5() {
        action21();
        return;
}
void p2c6() {
        action28();
        return;
}
void p2c7() {
        // do nothing
        return;
}
void procedure2() {
        if(a[xi][xj]!=0) {
                if(a[xi][xj+1]!=0) p2c1();
                else p2c2();
        }
        else if(a[xi][xj+1]!=0) {
                if(a[xi+1][xj]!=0) p2c3();
                else p2c4();
        }
        else if(a[xi+1][xj]!=0) p2c5();
        else if(a[xi+1][xj+1]!=0) p2c6();
        else p2c7();
}
int main(int argc, char *argv[]) {
        ll n,m,i,j,oddrow=0,oddcolumn=0,row = 0, column = 0, number_of_pixels = 0;
        int num;
        FILE *file_pointer;
        double time_spent,t1,t2;
        struct timespec tp,tp1;
        file_pointer = fopen(argv[1],"r");
        while(fscanf(file_pointer,"%d",&num) != EOF )
                number_of_pixels++;
        fseek(file_pointer,0,SEEK_SET);
        m=425;
        n=640;
        for(i=2;i<m+2;i++) {
```

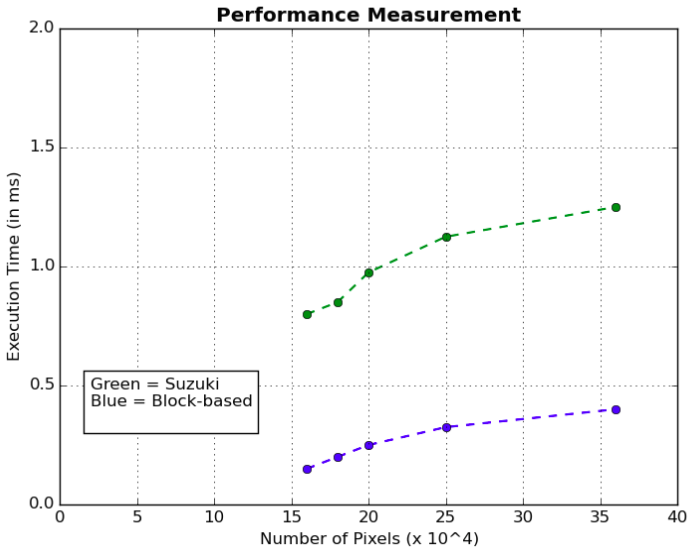
```
        for(j=2;j<n+2;j++) {
                fscanf(file_pointer,"%d",&num);
                a[i][j]=num;
        }
}
for(i=0;i<2;i++)
        for(j=0;j<n;j++) a[i][j]=0;
for(j=0;j<2;j++)
        for(i=0;i<m;i++) a[i][j]=0;
for(i=m+2;i<m+4;i++)
        for(j=0;j<n;j++) a[i][j]=0;
for(j=n+2;j<n+4;j++)
        for(i=0;i<m;i++) a[i][j]=0;
if(m%2!=0) {
        for(j=0;j<n+4;j++) a[m+4][j]=0;
        oddrow=1; m++;
}
if(n%2!=0) {
        for(i=0;i<m+4;i++) a[i][n+4]=0;
        oddcolumn=1; n++;
}
pi=0,pj=0,qi=0,qj=2,ri=0,rj=4,si=2,sj=0,xi=2,xj=2;
ll cnti=0,cntj=0;
clock_gettime(CLOCK_REALTIME,&tp);
t1 = (((double)tp.tv_sec) * 1000000) + (((double)tp.tv_nsec) / 1000) ;
while(cnti<m-1) {
                pj=0; qj=2; rj=4; sj=0; xj=2; cntj=0;
        while(cntj<n-1) {
                // When n and r are both background pixels
                if(a[si][sj+1]==0 && a[si+1][sj+1]==0) {
                        procedure1();
                } else procedure2();
                pj+=2; qj+=2; rj+=2; sj+=2; xj+=2; cntj+=2;
        }
        pi+=2; qi+=2; ri+=2; si+=2; xi+=2; cnti+=2;
}
clock_gettime(CLOCK_REALTIME,&tp1);
t2 = (((double)tp1.tv_sec) * 1000000) + (((double)tp1.tv_nsec) / 1000) ;
time_spent = t2 -t1;
if(oddrow) m--;
if(oddcolumn) n--;
printf("\nMatrix after labeling : \n");
```

```
	//Second scan for relabelling the required provisional labels
	ll arr[100]={0};
	for(i=2;i<m+2;i++)
		for(j=2;j<n+2;j++) arr[a[i][j]]++;
	ll val=0,k=2;
	for(i=2;i<m+2;i++) {
		for(j=2;j<n+2;j++) {
			if(a[i][j]!=1 && a[i][j]!=0)
			{
				if(a[i][j]!=val) while(arr[k]!=0) k++;
				val=a[i][j];
				if(k<a[i][j]) {
					a[i][j]=k;
					arr[a[i][j]]++;
				}
			}
		}
	}
	printMatrix(a,m,n);
	printf("Time Spent : %f\n",time_spent);
	return 0;
}
```

4. RESULTS AND SNAPSHOTS

The comparison graph between Suzuki's and block based algorithm was obtained for matrices with number_of_pixels varying as 16, 18, 20, 25, 36 is shown.

COMPARISON GRAPH



Snapshots of outputs obtained on some inputs by the two algorithms:

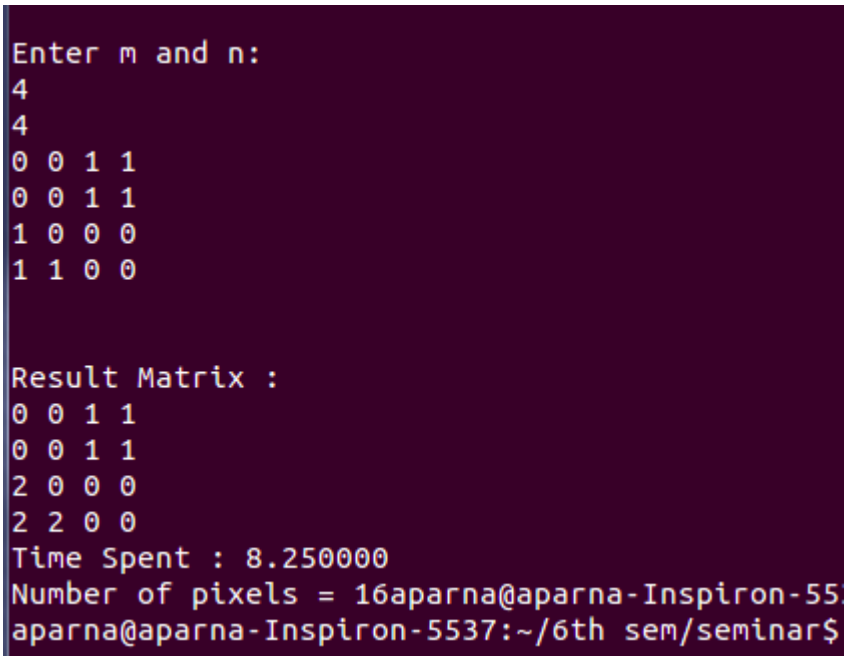
BLOCK-BASED SNAPSHOTS

```
aparna@aparna-Inspiron-5537:~/6th sem/seminar$ ./a.out

Enter m and n :
6
6

Enter the matrix :
0 0 1 1 0 0
0 0 1 0 0 1
0 0 1 1 0 0
0 0 0 0 0 1
0 1 1 0 0 0
0 0 1 1 0 0

Matrix after labeling :

Matrix :
0 0 1 1 0 0
0 0 1 0 0 2
0 0 1 1 0 0
0 0 0 0 0 3
0 4 4 0 0 0
0 0 4 4 0 0
Time Spent : 3.250000
Number of pixels = 36
```

```
aparna@aparna-Inspiron-5537:~/6th sem/seminar$ ./a.out

Enter m and n :
4
4

Enter the matrix :
0 0 1 1
0 0 1 1
1 0 0 0
1 1 0 0

Matrix after labeling :

Matrix :
0 0 1 1
0 0 1 1
2 0 0 0
2 2 0 0
Time Spent : 1.500000
Number of pixels = 16
```

SUZUKI SNAPSHOTS

```
aparna@aparna-Inspiron-5537:~/6th sem/seminar$ cc Suzuki.c
aparna@aparna-Inspiron-5537:~/6th sem/seminar$ ./a.out

Enter m and n:
6
6
0 0 1 1 0 0
0 0 1 0 0 1
0 0 1 1 0 0
0 0 0 0 0 1
0 1 1 0 0 0
0 0 1 1 0 0

Result Matrix :
0 0 1 1 0 0
0 0 1 0 0 2
0 0 1 1 0 0
0 0 0 0 0 3
0 4 4 0 0 0
0 0 4 4 0 0
Time Spent : 8.000000
Number of pixels = 36aparna@aparna-Inspiron-5537:~/6th sem/se
```



5. CONCLUSION

The task of connected components labelling is an important step in multiple image processing and computer vision projects.

Reducing the speed of computation of this task while maintaining it's accuracy is indeed a necessity.

Analysis shows that a block based algorithm using the strategies implemented has the optimal worst-case time complexity which is still better than other conventional algorithms. It was observed that the time complexity significantly reduced for the block-based algorithm for the provided inputs than for the Suzuki's algorithm. The paper proposes an efficient scanning algorithm for block-based connected-component labeling. The proposed method is expected to assist the labeling process in the field of computer vision. Using eight pixels selected from the original 20 pixels in the block-based scan mask, a simplified block-based scan mask using efficient strategies involving two procedures in different scenarios is developed. The two procedures are efficiently combined into the first-scan method, and are processed using binary decision trees through the simplified block-based scan mask. Experimental results showed that the proposed algorithm is superior to conventional methods in terms of improved labelling and execution time. Thus it can be implemented in practical scenarios.

6. REFERENCES

[1]. K. Suzuki, I. Horiba, and N. Sugie, “Linear-time connected-component labeling based on sequential local operations,” Comput. Vis. Image Underst. 89(1), pp. 1–23, 2003.

[2].K. Wu, E. Otoo, and A. Shoshani, Optimizing Connected Component Labeling Algorithms, In Proceedings of SPIE Medical Imaging Conference, pp. 1965-1976, Apr. 2005

[3] Costantino Grana, Member, IEEE, Daniele Borghesani, and Rita Cucchiara, Member, IEEE, Optimized Block-Based Connected Components Labeling With Decision Trees

[4] Wan-Yu Chang, Chung-Cheng Chiu and Jia-Horng Yang, Block-Based Connected-Component Labeling Algorithm Using Binary Decision Trees

[5] Wan-Yu Chang, Student Member, IEEE and Chung-Cheng Chiu, Member, IEEE, An Efficient Scan Algorithm for Block-Based Connected Component Labeling