Mef University

Computer Engineering Department

Object Counting

Muhammed Rahmetullah Kartal- 041701008

Programming Studio – Comp204

Instructor: Muhittin Gökmen

04.03.2019

Abstract

Objects on an image are countable by shrinking algorithms. Shrinking algorithms narrow down the image by deleting and augmenting pixels of the image on every iteration. This paper contains a comparison of the number of iterations of two shrinking algorithms named Levialdi Shrinking Algorithm and Two Subfields Algorithm (Gokmen & Hall). Also, it contains an explanation of these algorithms. Besides these, on the project, a Graphical User Interface is designed to better explain and compare both algorithms.

Introduction and Problem Definition

To get an enhanced image or to get useful information from the image we can make image processing. Image processing has some purposes, we can divide that processes into five groups

1. Visualization
2. Image Sharpening and Visualization
3. Image Retrieval
4. Measurement of Pattern
5. Image Recognition

This project is prepared to focus on Measurement of Pattern, the measurement is about counting the number of conic components on an image. Object counting can be used in urban planning[1] like event management, security, and emergency response or in biology to count the number of microorganisms[2] and situations like this. There are lots of algorithms about object counting. One type of these algorithms is shrinking algorithms, in the 21st century time is the most important thing in our lives. On the way of this purpose, we compared two shrinking algorithms named Levialdi Shrinking Algorithm and Two Subfields Algorithm (Gokmen & Hall). These two algorithms are kind of parallel algorithms, in other words, they shred the work and works on parts at the same time[3].

Solution Methods

Levialdi Algorithm

The working principle of this algorithm is deleting or augmenting every pixel by looking at their left, bottom-left, bottom neighbors. The algorithm starts changes from bottom left of every conic component and finishes them on the right top of the component.

To check left, bottom-left and bottom neighbors we need to convert the original image to binary image (0,1 or 0,255). The deletion condition works like if left, bottom and bottom left pixel of the current pixel is zero, make the current pixel zero too. The augmentation condition works like if the left and bottom pixels of the current pixel are ones, make the current pixel one too. By that processes, any conic component isn't uniting or decaying.

The algorithm starts checking the pixels from the top right of the image (right to left on every row), by doing that, on every iteration, it deletes the leftmost pixels of every conic component. After a couple of iterations, if one pixel is surrounded by zeroes, in other words, is isolated that means a conic component is found. After increasing the conic component counter, it deletes it and looks for other iterations. As an important notation, we need to hold the image on a temp image on every iteration because the code will change the pixel when it sees it, we need to do our processes by looking old version of the image.

Two Subfields Algorithm (Gokmen & Hall)

The working principle of this algorithm is dividing the image into two subfields and deleting or augmenting pixels. Like a parallel algorithm, it starts changes from the top left and bottom left and finishes them on the middle right of every conic component.

The first subfield is even-row even column odd row-odd column field, and the second subfield is even row-odd column - odd row even column field.

There are some special variables to check deletion or augmentation conditions first one of them is the number of one's in pixel's 8-neighborhood(all around the pixel) [B(pixel)], the number of distinct 8-connected components of one's in pixel's 8-neighborhood [C(pixel)], the number of 0-1 transitions in pixel's 8-neighborhood [T(pixel)].

The algorithm has 2 deletion conditions one of them is B(pixel) is 0 situation, on this condition also we need to increment conic component counter because if B(pixel) is 0, that

means the point is isolated (surrounded by zeroes), the other condition requires 3 conditions at the same time, first one of them is C(pixel) is 0, second one is if B(pixel) is 1 then top left and bottom left pixel needs to be 0, third condition is length of connected zeroes in 8-neighborhood needs to be bigger then 3. If all these conditions satisfied, we can delete pixel (1 to 0).

The algorithm has 1 augmentation condition, but this condition requires 2 conditions, first, one of them is C(pixel) is 1 and the second one is left, and top or left and bottom pixels needs to be 1. If both conditions satisfied, we can augment the pixel (0 to 1).

On every iteration, this algorithm changes the top left and bottom left sides of conic components, as an important notation we need to hold the change whole image at the same time, so we need to hold the image on a temp image at the start of every iteration and every subfield. After that, we can change the pixels by looking at that temp image.

My Work

The start was about implementing the Levialdi algorithm, to use in this algorithm we need a bordered binary image, to achieve this image firstly we are uploading an image to program by using Pillow package, after that with “colorToGray” method we are changing the image to gray image, we are doing that because RGB image has 3 attributes for each pixel but gray image has just 1, so it will be easy to convert binary image if we make it gray. After that by using “grayToBinary” method we are obtaining the binary image. In order to add border firstly we need to convert that image to an array, to do that we are using “imageToArray” method, after all, we are using the “borderImage” method to obtain bordered binary image.

In the implementation of Levialdi algorithm, an image array is given to method as a parameter. To implement the iterations a while loop is used, the break condition of that is no change situation, two Boolean flags used to check it because if one of the flags becomes false at the end of iteration it can break the loop in wrong time. At the start of every iteration, the current array is deep copied to a temp array to use in the next iteration. To implement a single iteration the checks starts from top right of the temp array because if it starts from other directions it has some problems because augmentation and deletion conditions are just looking for left, bottom left and bottom pixels, as an example if it starts from bottom left it will delete row by row, not the bottom left of every component. While the algorithm is

working some variables like outputs array, the number of conic components and iteration number also created to use in the GUI or to write CSV.

In the implementation of Two Subfields Algorithm (Gokmen & Hall) an image array is given to method as a parameter. To implement the iterations same while loop with the Levialdi algorithm is used. Every iteration contains two nested for loops because there are two subfields, starts of nested loops are controlled by if statements to satisfy even-row even column odd row-odd column, and even row-odd column - odd row even column harmony. To get B(pixel), C(pixel), T(pixel) and length of connected zeroes “neighbors” method is used. To calculate them 8-neighbors and 4-neighbors stored on two 1-D arrays. To calculate B(pixel) 8-neighbors array used if the element is 255 increase counter 1. To calculate T(pixel) 8-neighbors array checked again if the current element is 0 and the next element is 255 increase counter 1, a special case for T(pixel) is P8 to P1 jump. To calculate C(pixel) 8-neighbors array used again but some changes are done on it, if the corner pixel’s left and the bottom are 255 convert it to 255 too, after doing that calculate 0-1 transitions again. A special case for that is full of 255s if it is, increase the counter. To calculate the length of connected zeroes 8-neighbors array used if the current element of the array is 0 and the next element also zero increase counter 1, the 3 is enough so we can break the loop. A special case for that is if p8 to p1 jump.

On the GUI (Graphical User Interface) part Tkinter package of Python is used. At first, a window is created for selecting the resolution, after that with the selected resolution main window created, on the main window 6 proportioned canvasses with scrollbars created, the top rightmost canvas is for Levialdi algorithm, the bottom rightmost canvas is for Two Subfields Algorithm (Gokmen & Hall). To upload a file we have 2 options, an artificially created image or selected image its asked with a message box. To upload results to CSV there are buttons on the Algorithm canvasses when save button is pressed program asks for select a CSV or create a new one, after selecting it, upload the results to our CSV.

On every iteration of algorithms one of the middle canvasses content changes, in other words, it prints the image array to the window on every iteration. On the left top canvas, we can see the original image, on the left bottom canvas we can see the binary bordered image.

Results

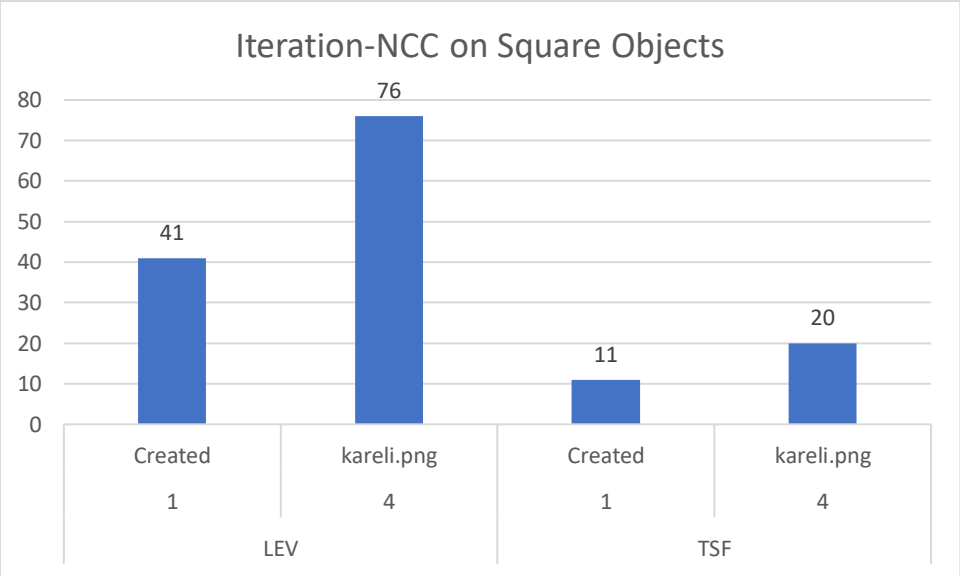

Figure 1 Iteration-NCC on Square Objects

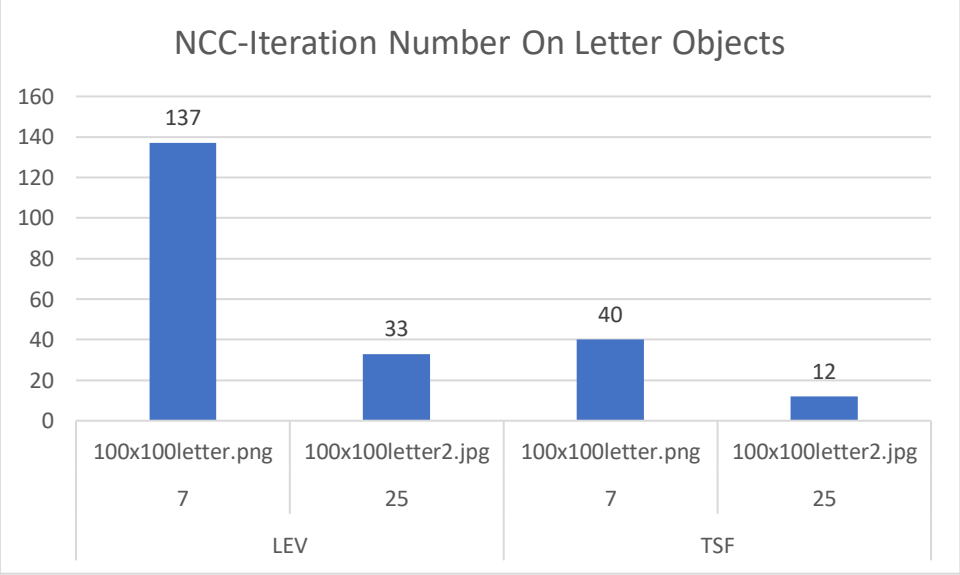

Figure 2: Iteration-NCC on Letter Objects

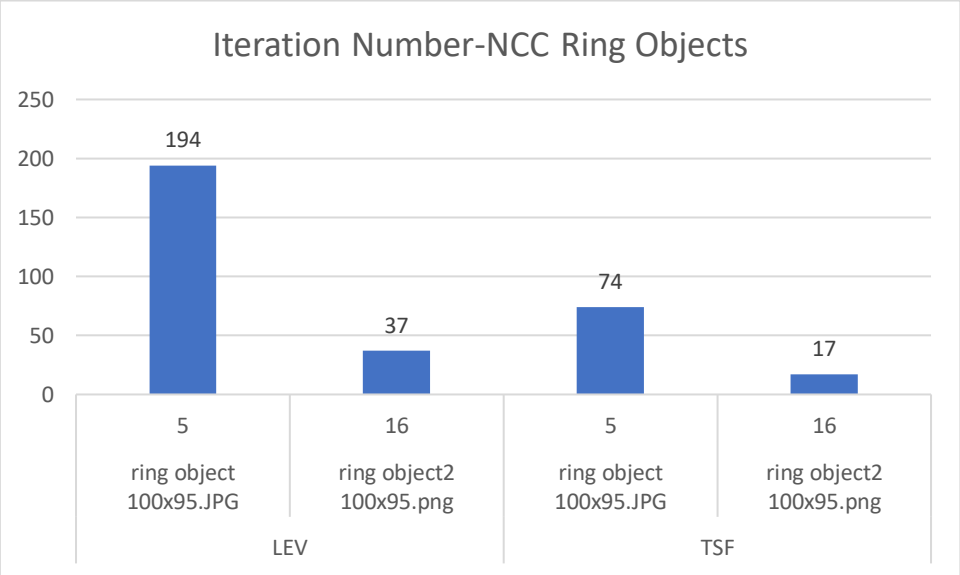

Figure 3: Iteration Number-NCC Ring Objects

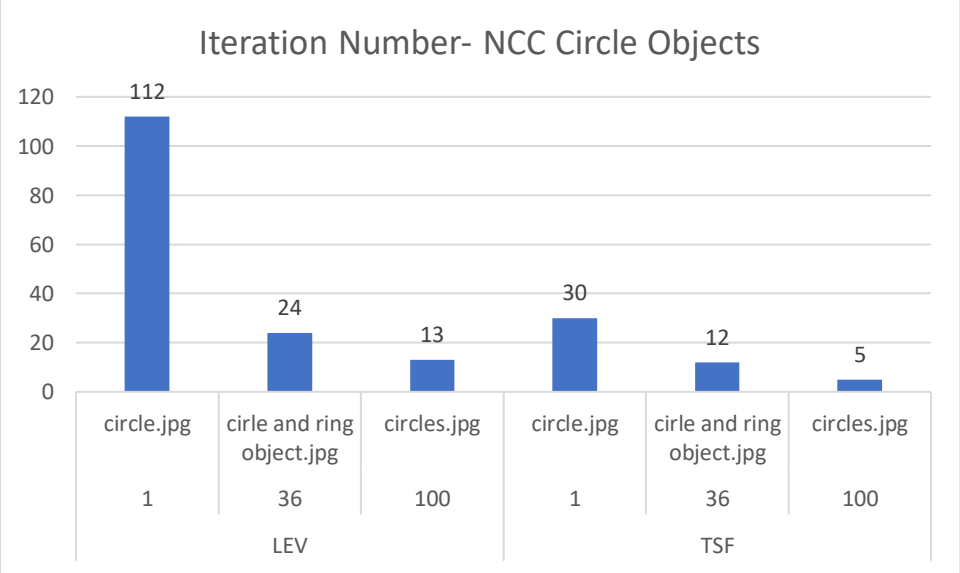

Figure 4: Iteration Number-NCC Circle Objects

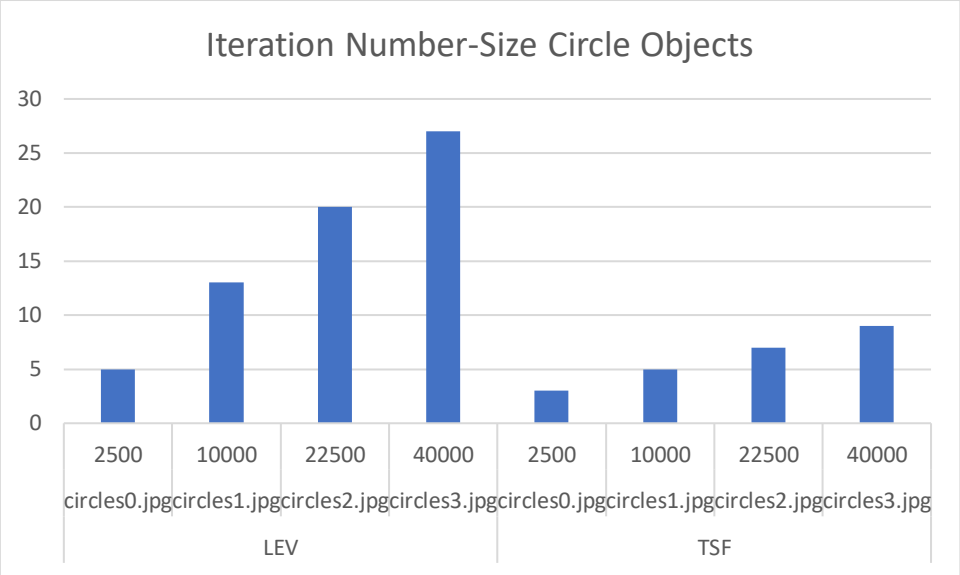

Figure5: Iteration Number-Size Circle Objects

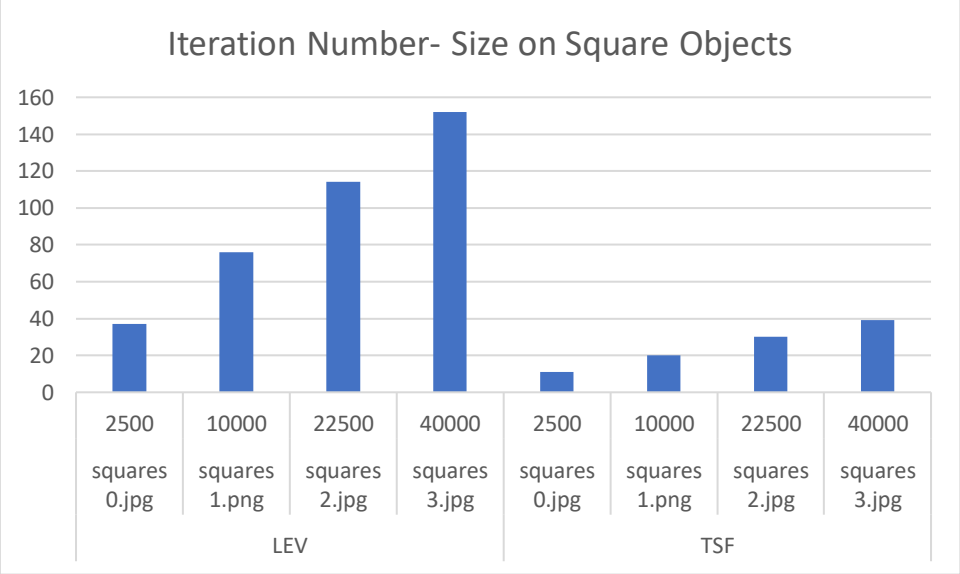

Figure 6: Iteration Number-Size Square Objects

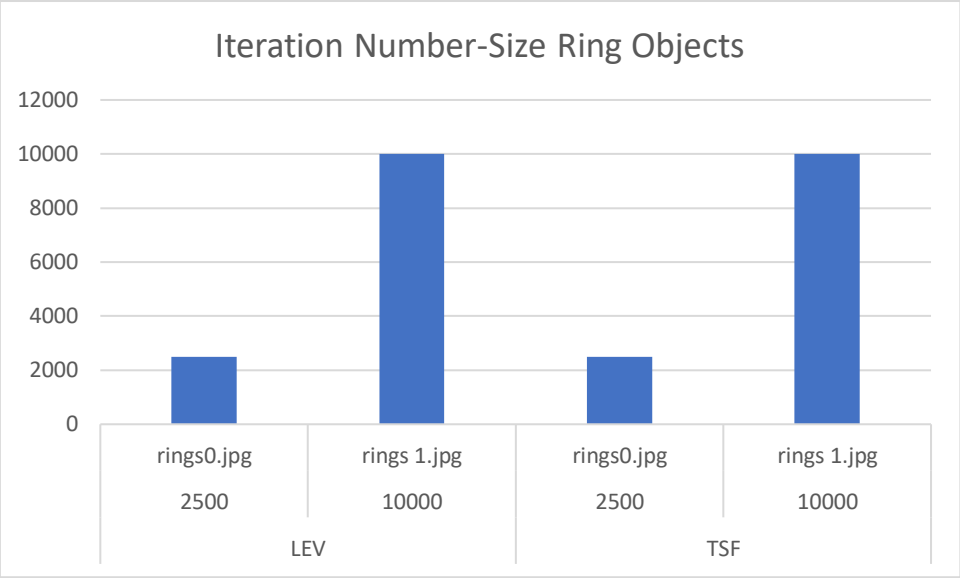

Figure 7: Iteration Number-Size Ring Objects

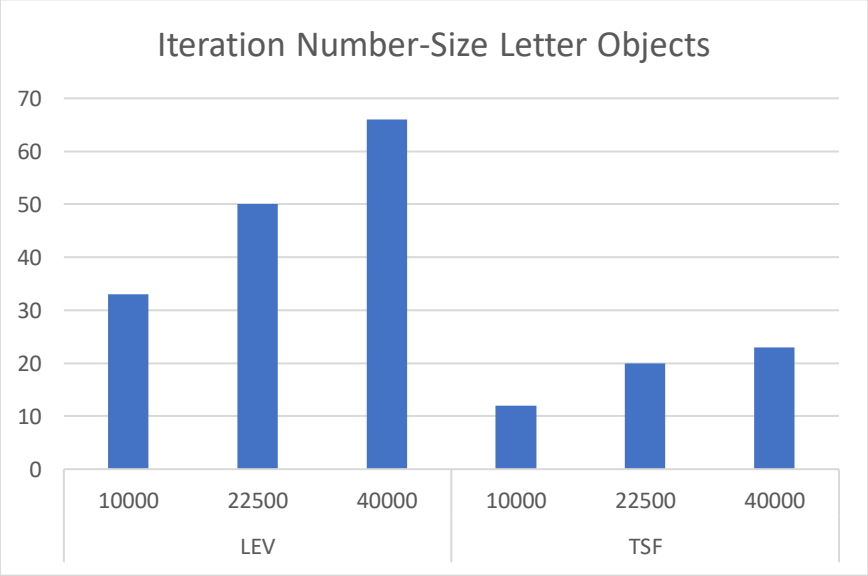

Figure 8: Iteration Number-Size Ring Objects

The Contribution of Project:

On this project, I learned how to create a Graphical User Interface on Python and I improved my Python knowledge. I realized that we need to write our programs ready to changes.

Conclusion:

As we can see at the first 4 figures Two Subfield Algorithm (TSF) is always faster than Levialdi Algorithm because it is like doing two iterations in one iteration and that's the reason behind that TSF algorithm seems like work slower than Levialdi Algorithm. If we look at the Figure 5, Figure 6, Figure 7, Figure 8 iteration number on TSF is getting lesser against Levialdi algorithm when the size of file increases.

References:

[1] N. Elassal and J. Elder, "Unsupervised Crowd Counting", *Elderlab.yorku.ca*, 2019. [Online]. Available: http://www.elderlab.yorku.ca/wp-content/uploads/2013/08/ElassalACCV16.pdf. [Accessed: 03- Mar- 2019].

[2] V. Lempitsky and A. Zisserman, "Learning To Count Objects in Images", *Robots.ox.ac.uk*, 2019. [Online]. Available: http://www.robots.ox.ac.uk/~vgg/publications/2010/Lempitsky10b/lempitsky10b.pdf. [Accessed: 03- Mar- 2019].

[3] S. Poonia, "Parallel Algorithms", *Slideshare.net*, 2019. [Online]. Available: https://www.slideshare.net/sandpoonia/lecture19-33728772. [Accessed: 03- Mar- 2019].