CMSC 25400 Assignment 3

Rachel Hwang

February 26, 2014

1 Implementation

My loopy belief propagation algorithm is implemented in ”message_passing.c”, closely following the specifications in the assignment. My program reads in a matrix of binary values from a file, adds noise, then runs the the message-passing algorithm to either convergence, or a cap of 30 iterations (This value was chosen based on observed runs. Almost always, when a run of the algorithm failed to converge by message value, it was because message values seemed to be toggling back and forth, despite prediction values already having converged). The resulting matrix of predicted values was compared to the original read-in matrix. Error rate is the number of errors over the total number of pixels in the image. Both the normalized message and prediction expressions were implemented exactly as given in the assignment.

2 Error Rate

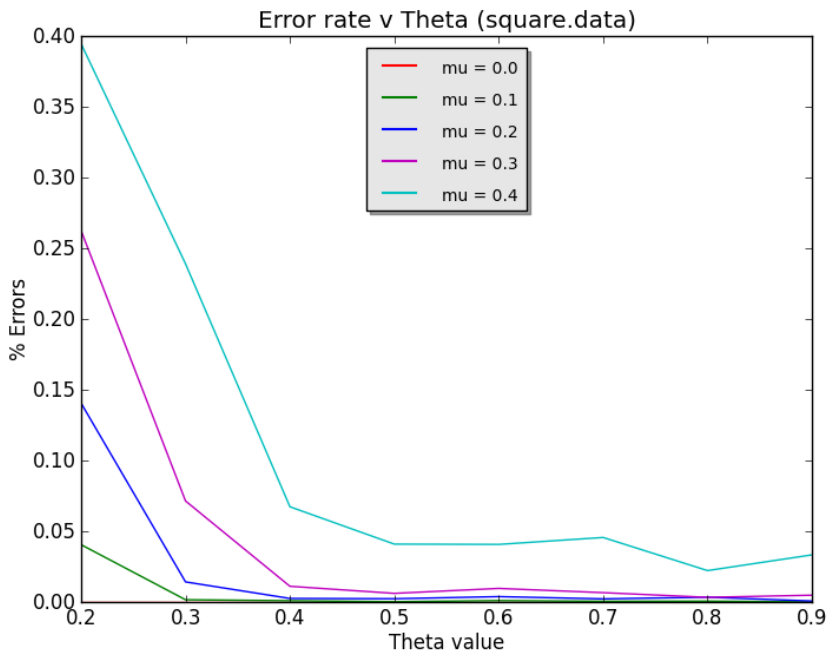

Figure 1:

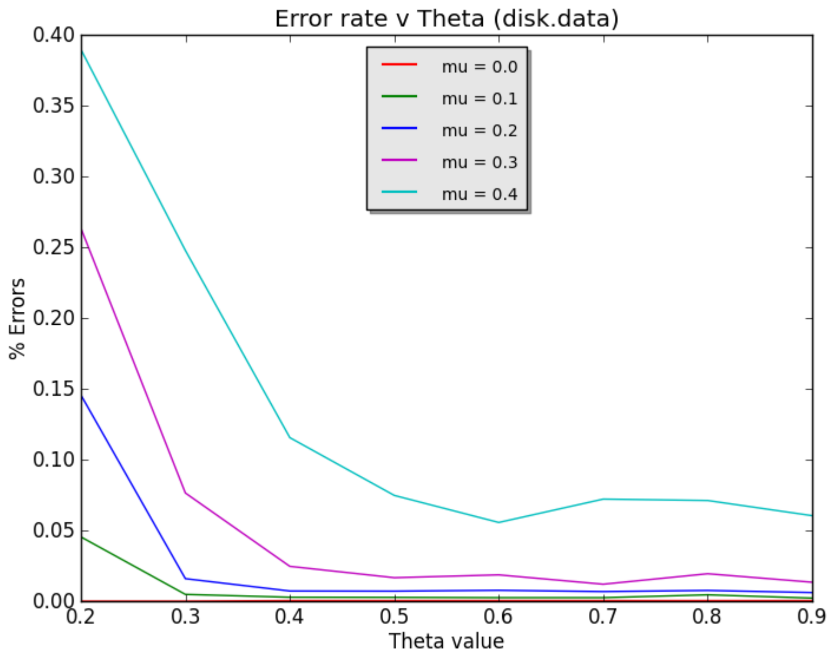

Figure 2:

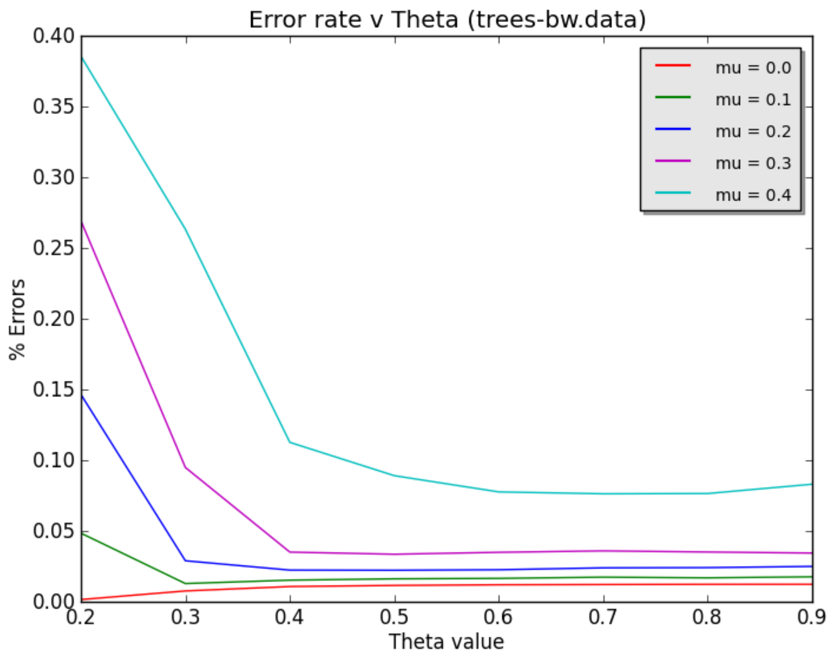

Figure 3:

As can be observed in the plots above, there is a negative correlation between error rate and theta value. For square and disk, the optimal theta value appears to be 0.8 and 0.9 respectively, for all mu values. However, for the trees image, optimal theta is anywhere from 0.4 to 0.9 for mu ≥ 0.2, and 0.3

for mu = 0.1.

Intuitively, the program performs best on the simplest images, those with the most well-defined black-/white boundaries. The square image can be reasonably-well denoised from mu values of up to 0.4, where disk is denoised less accurately, and trees, even less accurately. Note that the error rate of tree-bw with mu=0 *increases*, meaning that the program artificially smoothes the image even without noise. This makes sense given that the original image is fairly noisy. Visual inspection reveals lone pixels and ragged boundaries.

For all three images, the program is able to denoise images noised with mu values of 0.3 and below to an error rate of $< 5\%$, whereas for mu values of 0.4, the program performs noticably worse.

The errors in the output denoised images tend to be clusters of toggled pixels, the larger the flipped cluster, the more stable. This makes sense given that a flipped pixel is 'insulated' from the influence of the correct surrounding pixels if all of its neighbors are also flipped.

3 Iterations

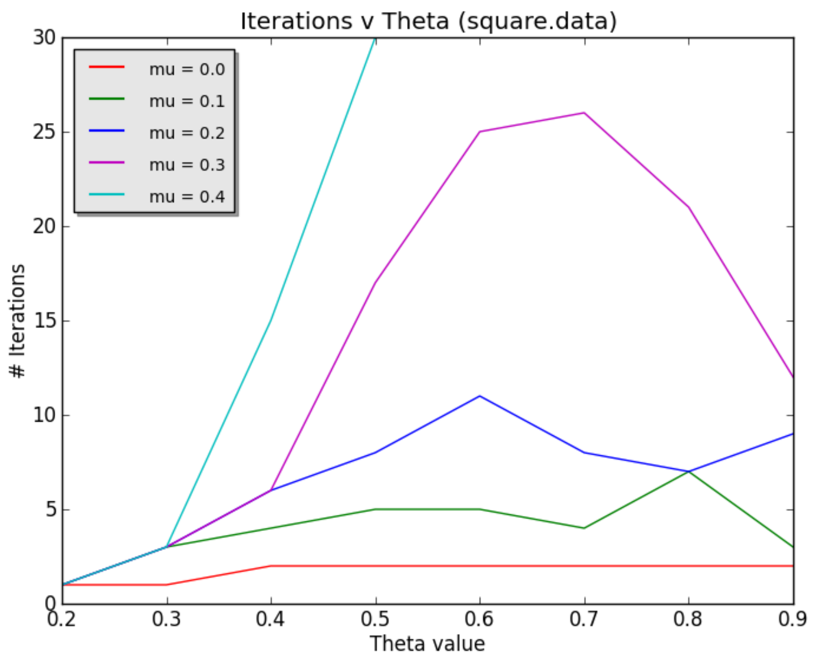

Figure 4:

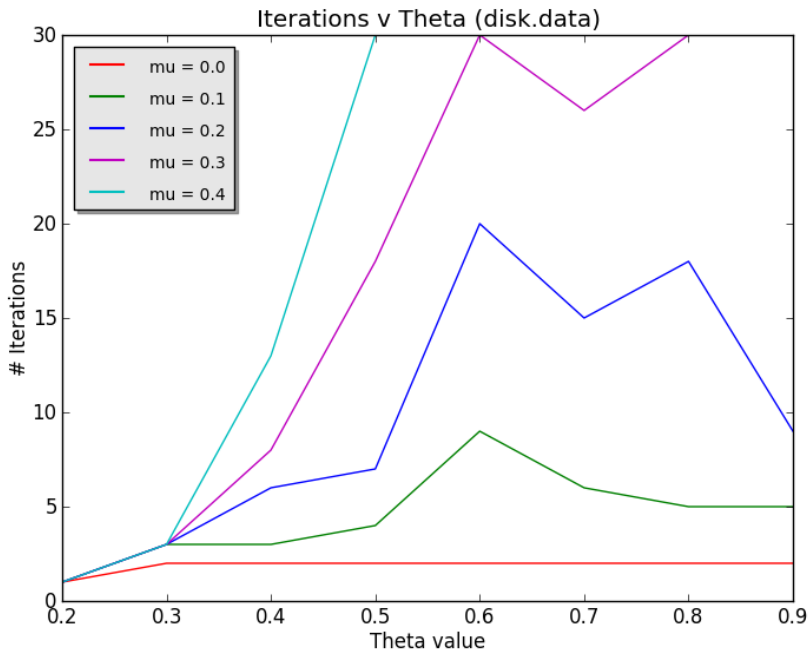

Figure 5:

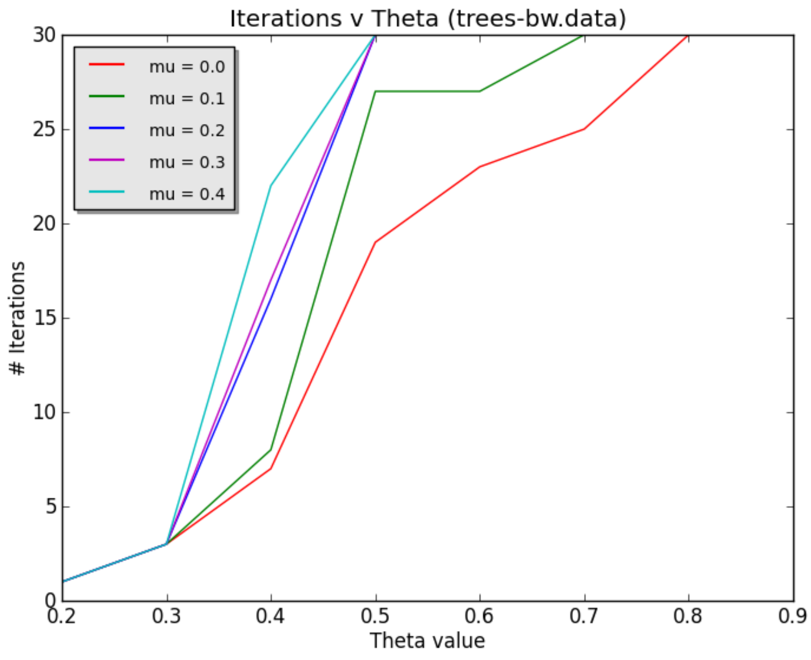

Figure 6:

Generally speaking, there is a positive correlation between theta value and number of iterations required to converge (again, note the cap at 30 iterations. For all runs, prediction matrices have essentially converged even if message values have not, due to message values flipping back and forth). This is because the more sensitive to its neighbors each node is, the more likely its message values are to change as theirs do, resulting in more movement overall, hence more iterations before convergence. By the same logic, small theta values cause the program to converge early, since pixels are influenced only slightly by their neighbors.

For square and disk, we see that at low mu values, the program converges quickly almost regardless of theta value. At higher mu values, number of iterations peaks at theta values of 0.6. Somewhat surprisingly, for mu = 0.2 and mu = 0.3, there is a decline in iterations required again after that peak, which seems to correlate with error rate performance. Because square and disk are fairly straightforward images, flipped pixels stand out as anomolous, large theta values are able to quickly correct the errors.

For the trees image, however, a noisy image, both higher mu and theta values yield worse performance. Iterations rise sharply after the optimal theta value of 0.3.

4 Images

Below are a comparison of noised and denoised images. Pervasive errors tend to be clusters of incorrect pixels. Unsurprisingly, the program is much better at generating straight lines of pixels than it is curves. It does the best with the simpliest images, so errors rate is square < disk < trees.

square.data

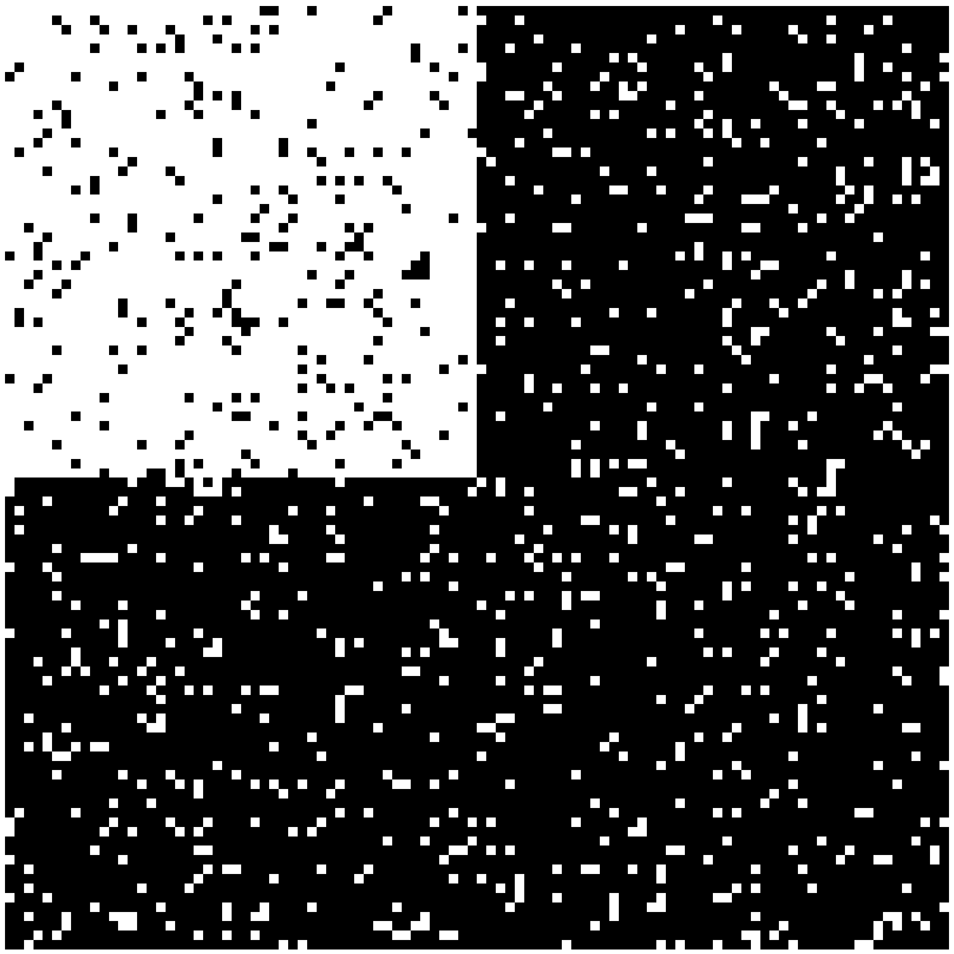
Figure 7:

Figure 8:

disk.data

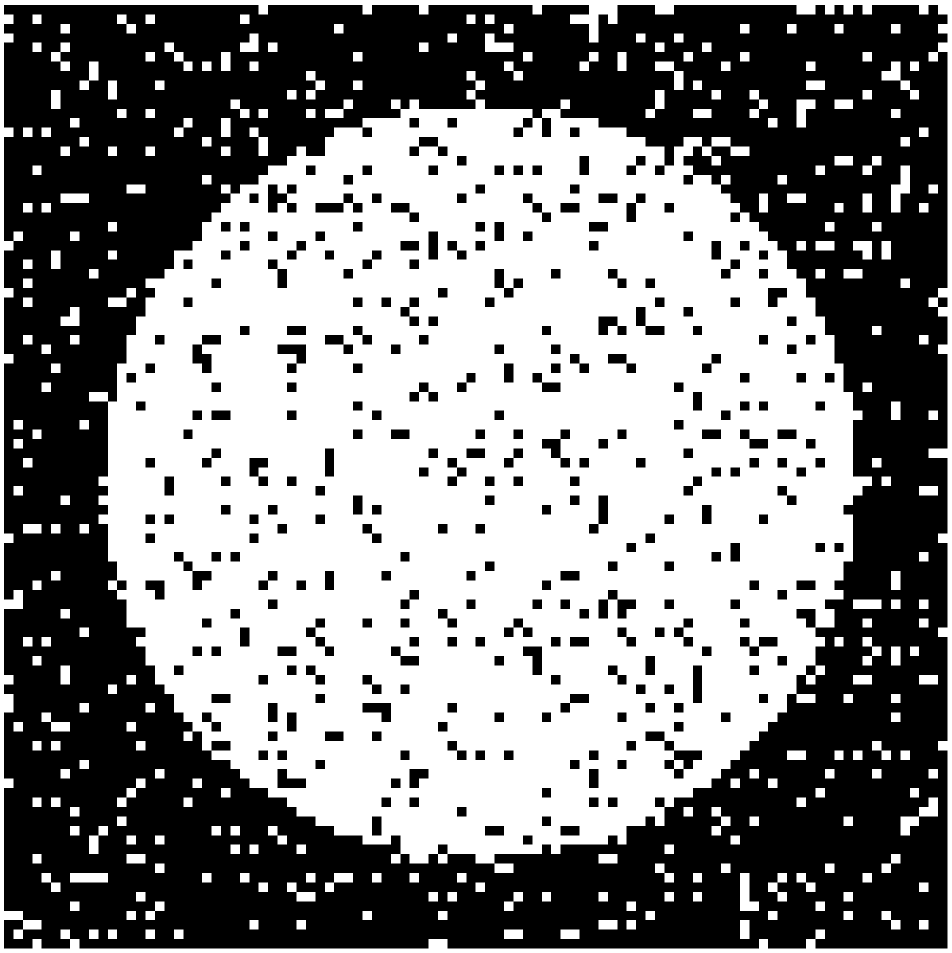
Figure 9:

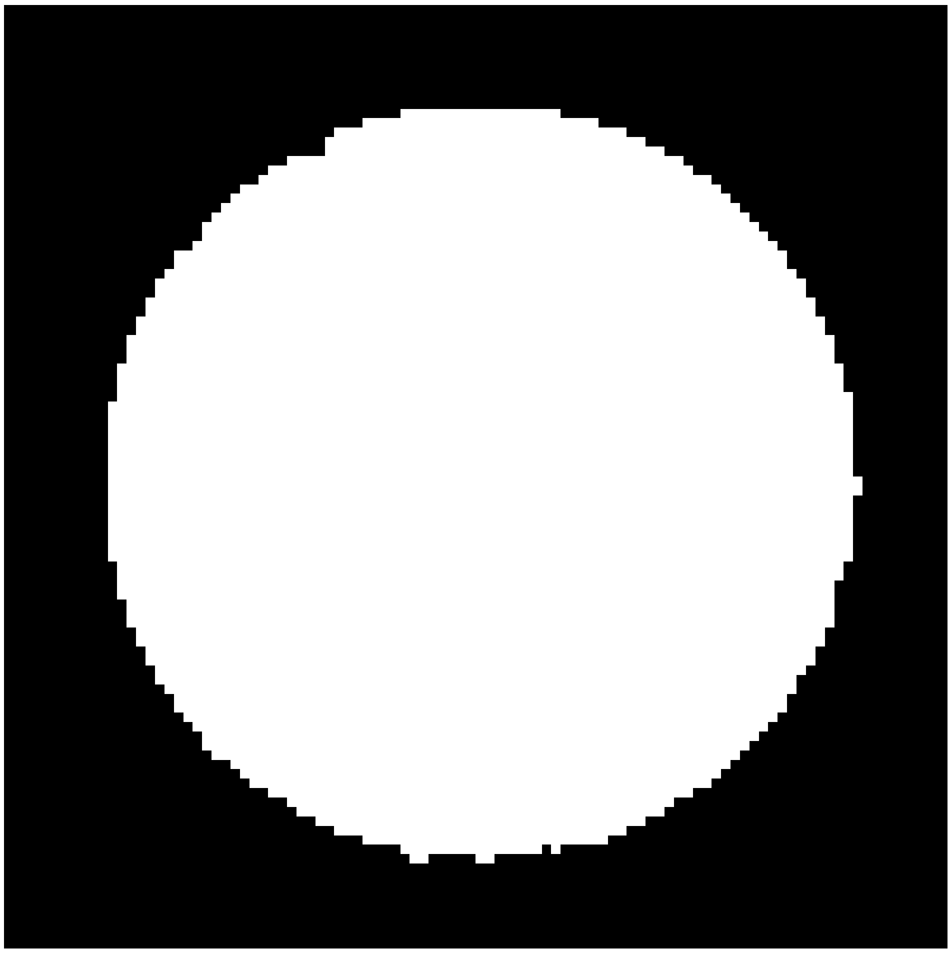
Figure 10:

trees-bw.data

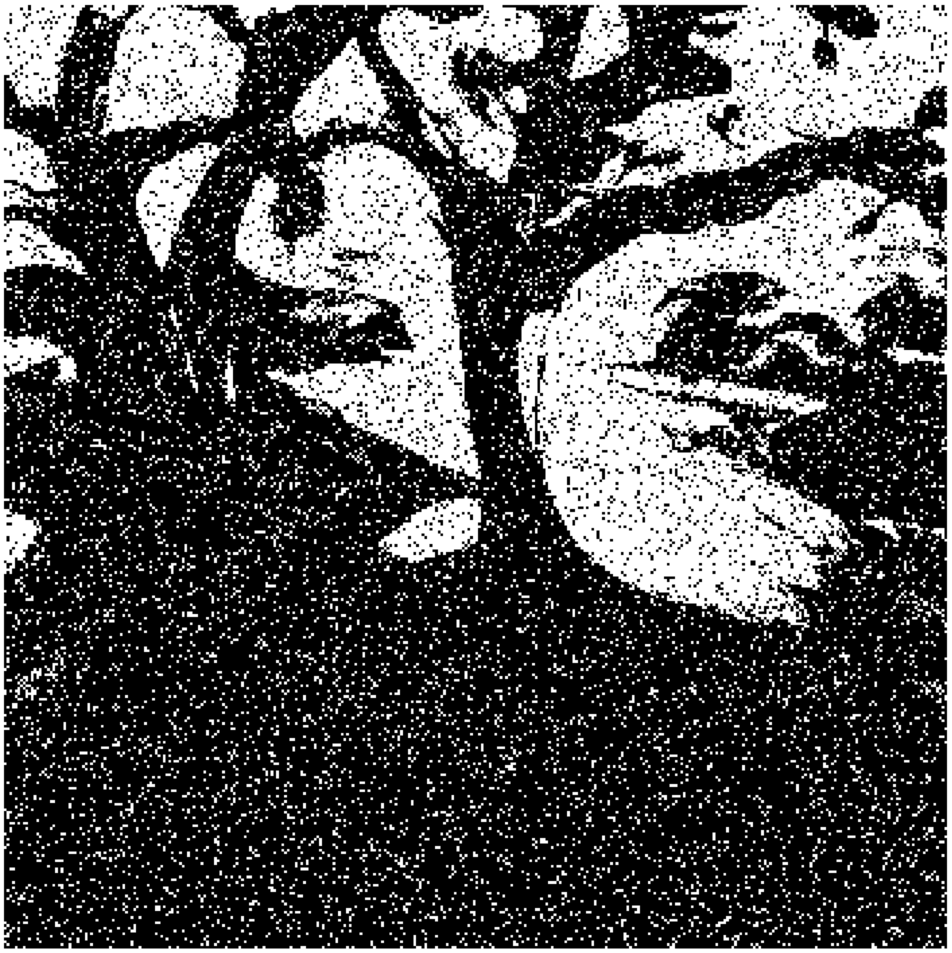
Figure 11:

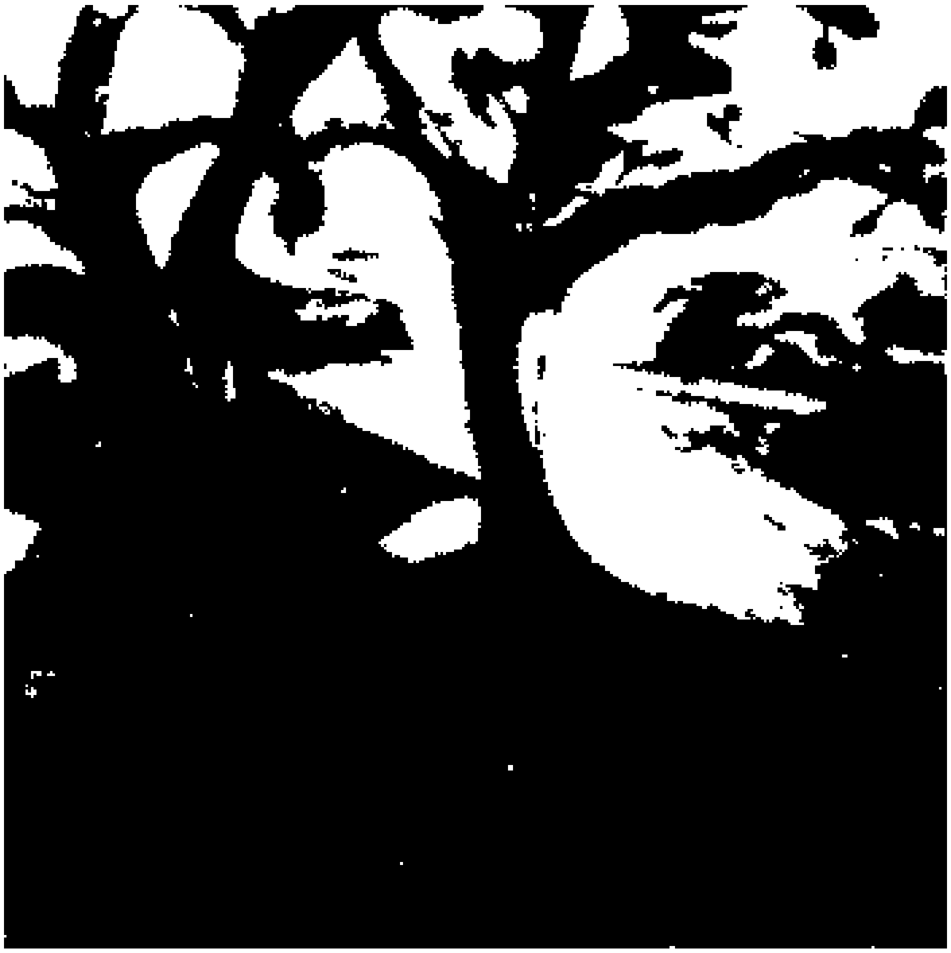
Figure 12:

Note that Tree image was extended to make square.