EDGE DETECTION USING ANT ALGORITHM

MTech. Mini Project Work for AOS

Somnath Paul & Kirti Sharma

SCHOOL OF COMPUTER & INFORMATION SCIENCE

University of Hyderabad

UNIVERSITY OF HYDERABAD

Certificate of Originality

This is to certify that the Project Report entitled "**Edge Detection using Ant Colony System**" submitted to School of Computer & Information Science, University of Hyderabad in partial fulfillment of the requirements for the award of the AOS mini project of MTech, is an authentic and original work carried out under my guidance by the following students with University roll no.

Name	University Roll No
Somnath Paul	**19MCMI09**
Kirti Sharma	**19MCMI25**

The matter embodied in this project is genuine work done by the student and has not submitted whether to this Institute or to any other University /Institute for the fulfillment of the requirements of any course of study.

This project is here by approved as accreditable performance for a group of MTech 1st Year students. It is carried out and presented in a manner will satisfactory to warrant it's acceptance as a prerequisite to the MTech for which it has been submitted. It is understood that by approval of this project Report the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the Project Report for the purpose it has been submitted.

_____ _____

Somnath Paul Kirti Sharma

_____

Signature of Instructor

Dr. M N Naagamani

Assistant Professor, Dept of CIS, UOH

Department of CIS
University of Hyderabad

CERTIFICATE OF APPROVAL

This is to certify that Somnath Paul and Kirti Sharma are the students of School of Computer and Information Science, University of Hyderabad, have successfully completed a project on "**Edge Detection using Ant Algorithm"** for MTech Mini Project work at School of Computer and Information Science.

_____

Signature of Instructor

Dr. M N Naagamani

Assistant Professor, Dept of CIS
University of Hyderabad

ACKNOWLEDGEMENT :

It is our privilege to express our sincerest regards to our project coordinator Dr. M N Naagamani, for her valuable inputs, able guidance, encouragement, whole-hearted cooperation and constructive criticism throughout the duration of our project.

We deeply express our sincere thanks to our Assistant Prof. Dr. M N Naagamani for encouraging and allowing us to present the project on the topic "**Edge Detection using Ant Algorithm**" at our department, premises for the partial fulfillment of the requirements leading to mini project for Advanced Operating System, MTech 1st Year.

We take this opportunity to thank all our lecturers who have directly or indirectly helped our project. We pay our respects and love to our parents and all other family members and friends for their love and encouragement throughout our career. Last but not the least we express our thanks to our friends for their cooperation and support.

Name & Signature of the Student with Date:

_____ _____

Somnath Paul Kirti Sharma

1 INTRODUCTION

Ant colony optimization (ACO) is a nature-inspired optimization algorithm that is motivated by the natural foraging behavior of ant species. Ants deposit pheromone on the ground to mark paths between a food source and their colony, which should be followed by other members of the colony. Over time, pheromone trails evaporate. The longer it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. Shorter – and thus, favorable – paths get marched over faster and receive greater compensation for pheromone evaporation. Pheromone densities remain high on shorter paths because pheromone is laid down faster. This positive feedback mechanism eventually leads the ants to follow the shorter paths. It is this natural phenomenon that inspired the development of the ACO metaheuristic. Dorigo et al. proposed the first ACO algorithm, ant system (AS) . Since then, extensions to AS have been developed. One of the successful ones is ant colony system (ACS) . ACO has been used to solve a wide variety of optimization problems. In this paper, an ACO-based method for image edge detection is proposed.

In recent paper of Image Edge Detection using Ant Colony Optimization, April 2010, ANNA VERONICA BATERINA and CARLOS OPPUS, have developed a paper to improve the algorithm.

1.1. Image Edge Detection

Image edge detection refers to the extraction of the edges in a digital image. It is a process whose aim is to identify points in an image where discontinuities or sharp changes in intensity occur. This process is crucial to understanding the content of an image and has its applications in image analysis and machine vision. It is usually applied in initial stages of computer vision applications. Edge detection aims to localize the boundaries of objects in an image and is a basis for many image analysis and machine vision applications. Conventional approaches to edge detection are computationally expensive because each set of operations is conducted for each pixel. In conventional approaches, the computation time quickly increases with the size of the image. An ACO-based approach has the potential of overcoming the limitations of conventional methods. Furthermore, it can readily be parallelized, which makes the algorithm easily adaptable for distributed systems. Several ACO-based approaches to the edge detection problem have been proposed. Previously reported ACO-based approaches to image edge detection, to the best of the authors' knowledge, all use a decision rule that is based on AS. This paper presents a technique that is derived from improvements introduced in ACS, one of the main extensions to AS. One of the significant aspects of ACS is the form of decision rule used, the pseudorandom proportional rule. The approach presented in this paper uses such rule in the tour construction process.

2 ABSTRACT

In this project a new algorithm for edge detection using ant colony search is proposed. The problem is represented by a directed graph in which nodes are the pixels of an image. To adapt the problem, some modifications on original ant colony search algorithm (ACSA) are applied. A large number of experiments are employed to determine suitable algorithm parameters. We drive an experimental relationship between the size of the image to be analyzed and algorithm parameters. Several experiments are made and the results suggest the effectiveness of the proposed algorithm.

3 INVOKING EDGE DETECTION

We have to change the Code file to Read Image. We have to give the hard coded path to the file in readGraph() function.

As a Example `f = new File("C:\\Users\\Somnath Paul\\Desktop\\Image.jpg");`

And We gate Output on the Pre-set path , which we can change that in code file , PrintImage() function

By default `f = new File("C:\\Users\\Somnath Paul\\Desktop\\Output.jpg");`

4 ABOUT THIS APPROACH

Edge detectors are used to detect and localize the boundaries of objects in an image. The complication of edge detector designing is the difficulty of defining precisely what means an edge. In practice, it is impossible to design an edge detector capable of finding all the true, and the only true edges in an image. Furthermore, edge detectors give ambiguous information about location of object boundaries. Edge detectors are usually subjectively evaluated by observers. Most conventional edge detectors are designed based on some models of edges. For examples, gradient-based methods assume edges as the set of pixels where the gray level has a high rate of change, or in a Canny edge detector, edges are considered as step functions corrupted by Additive White Gaussian Noise (AWGN). Konishi et al. formulate edge detection as statistical inference where the detection of edges depends on the statistics of both “on the edge” and “off the edge” filters. Conventional edge detectors are usually performed by linear filtering operations. An earlier smoothing operation for reducing the noise effect is sometimes applied. Edge operators that combine smoothing and derivation can use first derivative (Canny) or a Laplacian of the Gaussian. A visual model-based edge detector is proposed by Peli. In this model, the visual receptive field like filtering channels are used, and the threshold is the contrast sensitivity of the human eye. Suzuki et al. proposed an edge detector using a multilayer neural network, called a neural edge detector (NED), for detecting the desired edges clearly from noisy images. This paper describes an edge detector using ant colony algorithms. The ant colony system(ACS) algorithm was proposed by Dorigo in 1992, and has been used to solve many complex problems successfully such as the traveling salesman problem, quadratic assignment problem, design of combinational logic circuits, combined heat and power economic dispatch, data mining, classification rule discovery, data clustering, image retrieval, and approximation of plane curves.

5 ANT COLONY SEARCH ALGORITHMS

Real ants follow their own agenda of tasks independent from each other, however, when act as a community, they are capable to solve their daily complex problems, which require sophisticated planning, such as selecting and picking up materials, or, finding and storing foods. However, there is no kind of supervising or controlling. Finding the shortest route between the colony and a food source, is done by an exchange of information about the path should be followed. Ants communicate with each other by means of pheromone trail. They mark the path by leaving a certain amount of pheromone. Ants probabilistically prefer to follow a direction, proportional to the amount of pheromone on it. The more ants follow a trail, the more attractive this trail becomes to be followed by other ants. This process could be considered as a positive feedback loop.

An artificial ant colony system is an agent-based algorithm, which simulates the behavior of real ants. Artificial

ants are like real ants with some major differences:

– Artificial ants have memory;

– They are not completely blind;

– They live in a discrete time environment.

However they have some adopted characteristics from real ants:

– Artificial ants probabilistically prefer paths with a larger

amounts of pheromone.

– Shorter paths have larger rate of growth in the amount of

pheromone.

– The ants communicate to each other by means of the

amount of pheromone laid on each path.

ACS is an iterative algorithm. At each iteration, it performs

a loop containing two basic operations:

(1) construction or modification of solutions of the problem,

(2) updating the pheromone trails

Different steps of an ACS algorithm are the followings:

1. Problem graph representation: problems which could be solved by ACSA are often discrete, so they could be represented by a graph with N nodes and E edges, G = N, E.

2. Initializing ants distribution: a number of ants are placed on the randomly chosen nodes.

3. Node transition rule: the node transition rule specifies how ants must move from node to node. The node transition is probabilistic. The probability of displacing kth ant from node i to node j is given by:

$$p_{ij}^{k} = \begin{cases} \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{h \notin \text{tabu}_k}(\tau_{ih})^{\alpha}(\eta_{ih})^{\beta}} & \text{if } j \notin \text{tabu}_k, \\ 0 & \text{otherwise,} \end{cases}$$

where τij and ηij are the intensity of pheromone and the visibility of edge (i, j), respectively, and, α and β are control parameters. By tabuk we present the set of inaccessible nodes.

4. Pheromone updating rule: a cycle of ACS algorithm is completed when every ant has constructed a solution. At the end of each cycle, the intensity of pheromone is update by a pheromone trail updating rule:

$$\tau_{ij}(\text{new}) = (1-\rho)\tau_{ij}(\text{old}) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \qquad (2)$$

where ρ represents the pheromone evaporation, and is the amount of pheromone laid on edge (i, j) by the kth ant and could be given by:

$$\Delta\tau_{i,j}^{(k)} = \begin{cases} \frac{1}{L_k}, & \text{if ant k used edge (i, j)} \\ 0, & \text{otherwise} \end{cases} \qquad (3)$$

where Lk is the tour value of the solution found by kth ant and 1 is a constant.

5. Stopping criterion: The end of the algorithm could be achieved by a predefined number of cycles, or the maximal number of cycles between two improvements of the global best solution.

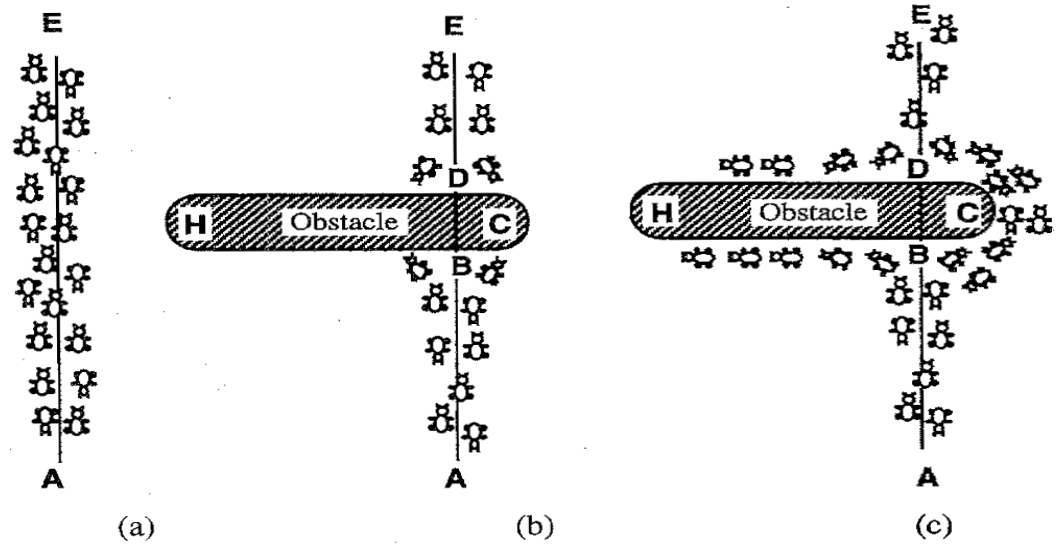


Fig. 1. An example with real ants. (a) Ants follow a path between points A and E. (b) An obstacle is interposed; ants can choose to go around it following one of the two different paths with equal probability. (c) On the shorter path more pheromone is laid down.

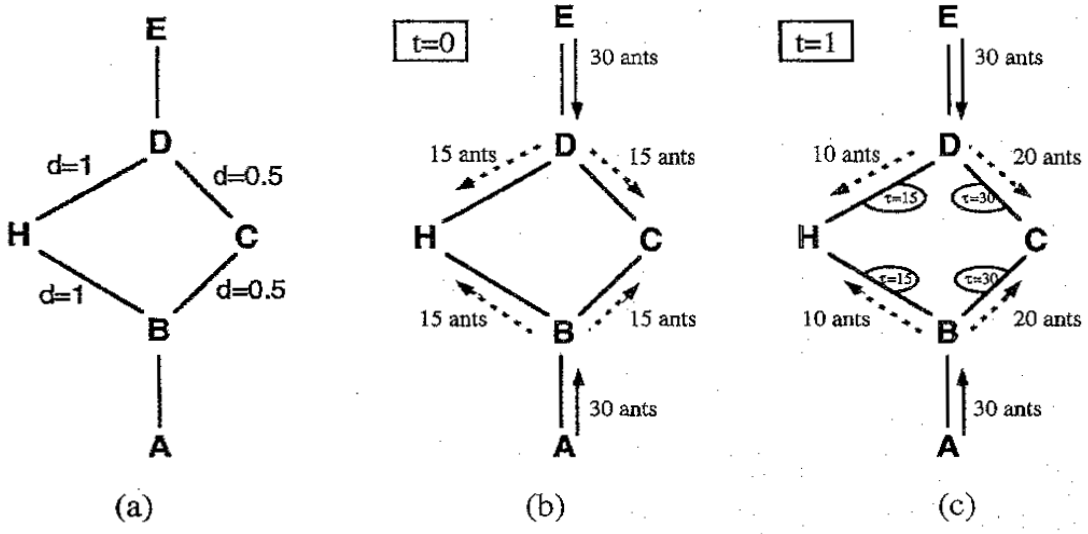


Fig. 2. An example with artificial ants. (a) The initial graph with distances. (b) At time $t = 0$ there is no trail on the graph edges; therefore ants choose whether to turn right or left with equal probability. (c) At time $t = 1$ trail is stronger on shorter edges, which are therefore, in the average, preferred by ants.

6 ANT COLONY SEARCH METHOD FOR EDGE DETECTION

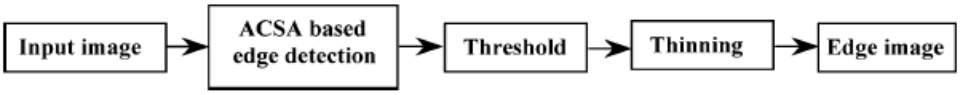
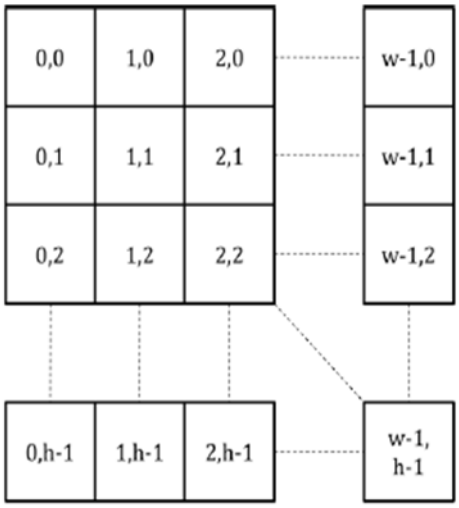
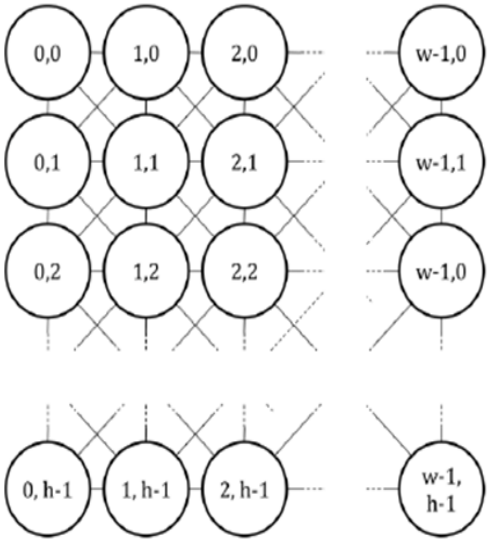
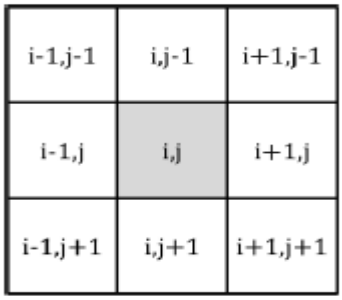


Image edge detection can be thought of as a problem of identifying the pixels in an image that correspond to edges. A **w × h two-dimensional digital image** can be represented as a two-dimensional matrix with the image pixels as its elements.

The graph is defined as follows. The components of the graph are the pixels of the image. The connections of the graph connect adjacent components or pixels together. An **8-connectivity pixel configuration is used**: a pixel is connected to every pixel that touches one of its edges or corners. Ants traverse the graph by moving from one pixel to another, through their connections. An ant cannot move to a pixel if it is not connected to the pixel where the ant is currently located. This means that an ant can move only to an adjacent pixel.





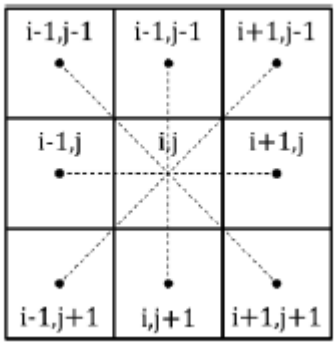


6.1. Initialization Process

In the initialization process, each of the K ants is assigned a random position in the M1 x M2 image. The initial value of each element in the pheromone matrix is set to a constant τ_{init}, which is small but non-zero. Also, the heuristic information matrix is constructed based on the local variation of the intensity values. The heuristic information is determined during initialization since it is dependent only on the pixel values of the image, thus, constant.

$$\eta_{i,j} = \frac{1}{I_{Max}} \times \text{Max} \begin{bmatrix} |I(i-1, j-1) - I(i+1, j+1)|, \\ |I(i-1, j+1) - I(i-1, j+1)|, \\ |I(i, j-1) - I(i, j+1)|, \\ |I(i-1, j) - I(i+1, j)| \end{bmatrix}. \quad (4)$$

In this approach, the value of visibility is determined using the maximum variation of gray level of the image intensity. Therefore, edge pixels are expected to have a greater value of visibility. In addition, all pixels currently registered in ants memory are considered as non-admissible pixels.



7.2. Iterative Construction and Update Process

On every iteration, each ant moves across the image, from one pixel to the next, until it has made L construction steps (a construction step consists of a single movement from one pixel to another). An ant moves from the pixel (i_0, i_1) to an adjacent pixel (i, j) according to the pseudorandom proportional rule. The transition probability for the biased exploration is given by

$$p^{(n)}_{(i_0,j_0),(i,j)} = \frac{\left(\tau^{(n-1)}_{i,j}\right)^{\alpha}(\eta_{i,j})^{\beta}}{\sum_{(i,j)\in\Omega_{(i_0,j_0)}}\left(\tau^{(n-1)}_{i,j}\right)^{\alpha}(\eta_{i,j})^{\beta}} \qquad (9)$$

After every step, the pheromone level is updated according to Eq.

$$\tau_{(i,j)}(\text{new}) = (1-\rho)\tau_{(i,j)}(\text{old}) + \Delta\tau_{(i,j)} \qquad (5)$$

where

$$\Delta\tau_{(i,j)} = \sum_{k=1}^{m}\Delta\tau^{k}_{(i,j)}, \quad \text{and,}$$

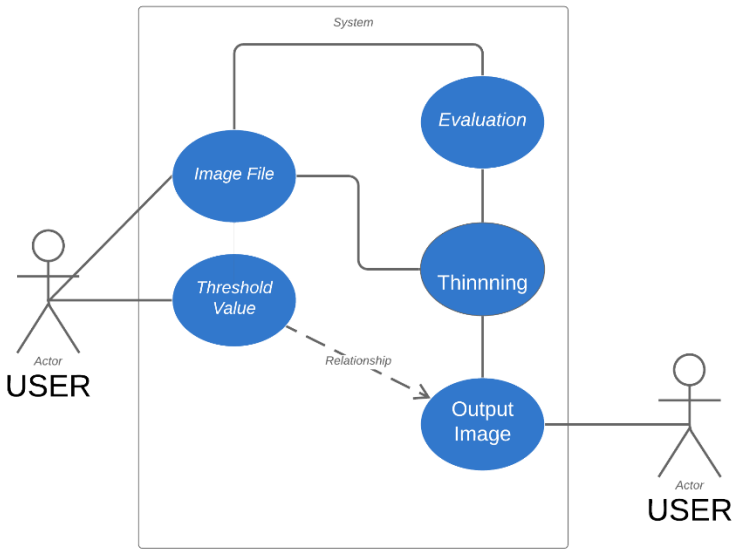
$$\Delta\tau^{k}_{(i,j)} = \begin{cases} \eta_{(i,j)} & \text{if } \eta_{(i,j)} \geq b \,\&\, k\text{th ant displaces to pixel } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

8.3. Stopping criterion:

The end of algorithm achieves by a pre-defined number of cycles, for which each cycle contains a fix number of steps. The number of cycles is chosen to be 3. This value works well enough for all the experiments.

9 UML DIAGRAM

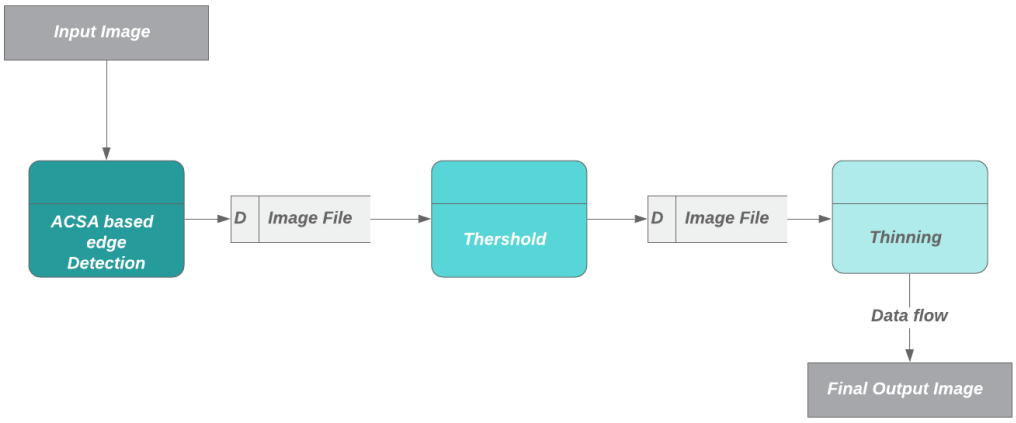
1. BASIC USE CASE DIAGRAM



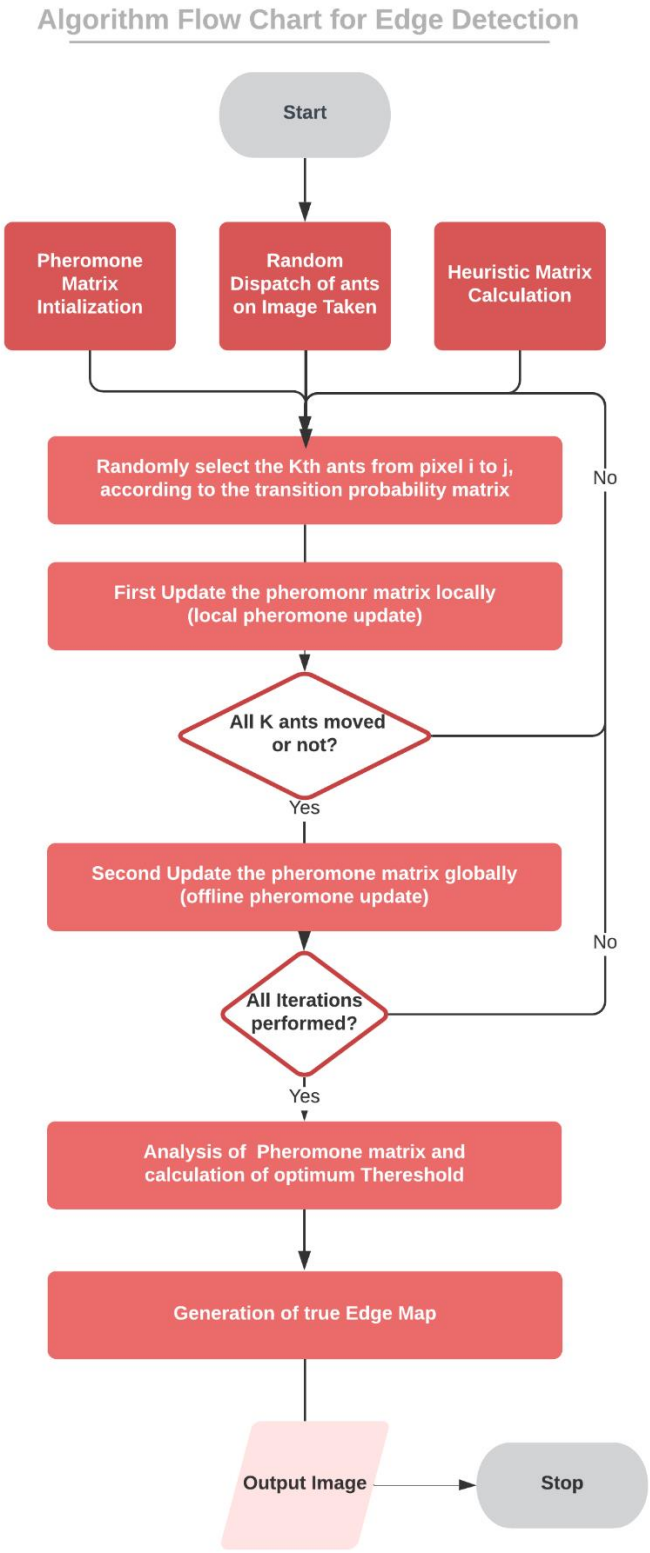
Basic Use Case Diagram For EDGE Detection

2. DATA FLOW DIAGRAM

Data Flow Diagram (Logical)



3. ALGORITHM FLOW CHART



10 CODE (JAVA)

```
import java.io.File;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Random;
import java.lang.*;
import java.awt.*;
import java.lang.Double.*;


public class AntSystem {
    // Algorithm parameters:
    // original amount of trail
    private double c = 1;
    // trail preference
    private double alpha = 5.5;
    // greedy preference
    private double beta = 7.5;
    // trail evaporation coefficient
    private double evaporation = 0.5;
    // new trail deposit coefficient;
    private double Q = 1000000.0;
    // number of ants used = numAntFactor*numTowns
    private double numAntFactor =0.01;
    // probability of pure random selection of the next town
    //private double pr = 0.01;

    // Reasonable number of iterations
    private int maxIterations = 100;

    public int n = 0;
    public static int width=0;
    public static int height=0; // # towns
    public int m = 0; // # ants
    private static double graph[][] = null;
    private double trails[][] = null;
```

```
    private double heuristic[][] = null;
    private Ant ants[] = null;
    private double probs[] = null;
    private int currentIndex = 0;
    public static BufferedImage img = null;
    public static File f = null;
    double vmax=0.0;
    public int[] bestTour;
    private Random rand = new Random();
    public double bestTourLength;
    public double Theresold=0.0083;

    // Ant class. Maintains tour and tabu information ofr every ants
    private class Ant {
        public int tour[][] = new int[n][2];
        public boolean visited[][] = new boolean[width][height];

        public void visitPixel(int i,int j) {
            tour[currentIndex + 1][0] = i;
            tour[currentIndex + 1][1] = j;
            visited[i][j] = true;
        }

        public boolean visited(int i,int j) {
            return visited[i][j];
        }

        public double tourIntensity() {
            double intensity = 0.0;
            for (int i = 0; i < n-1; i++) {
                //System.out.println("I am in 10: i "+i +":"+ tour[i][0]+" "+t
our[i][1] );
                intensity += calcIntensity(tour[i][0],tour[i][1]);
            }
            return intensity;
        }

        public void clear() {
            for (int i = 0; i < width; i++)
                for(int j=0; j<height; j++)
                visited[i][j] = false;
        }
    }

    public  double calcIntensity(int i,int j)
```

```
    {
        double max=0;
        double intensity=0.0;
        //System.out.println("I am in calc intensity"+intensity);
        if(i+1<width && j+1<height && i-1>=0 && j-1 >=0 )
            intensity=(graph[i-1][j-1]-graph[i+1][j+1]);
        else
            intensity=0;

        if(intensity<0)
            intensity*=-1;
        if(intensity>max)
            max=intensity;
           // System.out.println("I am in calc intensity"+intensity+ " "+graph
[i-1][j-1]+" "+graph[i+1][j+1] );
        if(i+1<width && i-1>=0)
            intensity= (graph[i-1][j]- graph[i+1][j]);
        else
            intensity=0;

        if(intensity<0)
            intensity*=-1;
        if(intensity>max)
            max=intensity;
          //  System.out.println("I am in calc intensity"+intensity);
        if(i+1<width && j+1<height && i-1>=0 && j-1 >=0 )
            intensity=(graph[i-1][j+1]-graph[i+1][j-1]);
        else
            intensity=0;

        if(intensity<0)
            intensity*=-1;
        if(intensity>max)
            max=intensity;
           // System.out.println("I am in calc intensity"+intensity);
        if(i+1<width && j+1<height && i-1>=0 && j-1 >=0 )
            intensity=(graph[i][j-1]-graph[i][j+1]) ;
        else
            intensity=0;

        if(intensity<0)
            intensity*=-1;
        if(intensity>max)
            max=intensity;
          //  System.out.println("I am in calc intensity"+intensity);
```

```
                return max;
    }
    public static int calcIntvalue(int red, int green, int blue)
    {
            int rgb = red;
            rgb = (rgb << 8) + green;
            rgb = (rgb << 8) + blue;
        return rgb;
    }

    public static double pow(final double a, final double b) {
        final int x = (int) (Double.doubleToLongBits(a) >> 32);
        final int y = (int) (b * (x - 1072632447) + 1072632447);
        return Double.longBitsToDouble(((long) y) << 32);
    }

    // Read in graph from a file.
    // Allocates all memory.
    // Adds 1 to edge lengths to ensure no zero length edges.
    public void readGraph() throws IOException {
        try{
          f = new File("C:\\Users\\Somnath Paul\\Desktop\\Test\\Untitled.png")
;
          img = ImageIO.read(f);
        }catch(IOException e){
          System.out.println(e);
        }

        //get image width and height
        width = img.getWidth();
        height = img.getHeight();
        graph = new double[width][height];


        System.out.println("Width= "+width+" Height= "+height );

        for(int i=0;i<width;i++){
       // System.out.print("----------------------I am in  "+i+"-------------
---\n");
            for(int j=0;j<height;j++){
                Color c = new Color(img.getRGB(i, j));
                int a= calcIntvalue(c.getRed(),c.getGreen(),c.getBlue());
                graph[i][j]=a;
                /*System.out.print(" "+String.format("0x%08X", graph(i,j)));*/
            }
```

```
        //  System.out.println("");
      }



      vmax=0.0;


      for(int i=1;i<width-1;i++){
          for(int j=1;j<height-1;j++){
              double temp=calcIntensity(i,j);
              if(temp>vmax)
                  vmax=temp;
          }
      }

      if(vmax==0)
          vmax=vmax+0.1;

  System.out.println("Maximum Intensity "+vmax);

      n= 10000;
      // all memory allocations done here
      trails = new double[width][height];
      heuristic = new double[width][height];
      probs = new double[8];

    //  System.out.println("I am here 1");
      for(int i=1;i<width;i++){
          for(int j=1;j<height;j++){
              heuristic[i][j]=calcIntensity(i,j)/vmax;
              if(heuristic[i][j]<0)
                  heuristic[i][j]=0-heuristic[i][j];
          }
      }
     // System.out.println("I am here 3");
      m =(int)(width * height* numAntFactor);
     // System.out.println("number of ants"+m);
      ants = new Ant[m];
      for (int j = 0; j < m; j++)
          ants[j] = new Ant();
     // System.out.println("I am here 4");
  }
```

```
    // Store in probs array the probability of moving to each town
    private void probTo(Ant ant) {
        int i = ant.tour[currentIndex][0];
        int j = ant.tour[currentIndex][1];

        double denom = 0.0;
        /*need an updataion depending on result*/


            if(i-1<width && j-1 <height && i-1>=0 && j-1 >=0 ){
                //System.out.println("heuristic[i-1][j-1]="+heuristic[i-1][j-
1] );

                Double x=pow(trails[i-1][j-1], alpha)* pow(heuristic[i-1][j-
1], beta);
                if(!x.isNaN())
                    denom+=x;
                else
                    denom+=0.0001;

            }
            else
                denom +=0;
           // System.out.println("denom="+denom );
            if(i+1<width && j+1 <height && i+1>=0 && j+1 >=0 ){
                Double x= pow(trails[i+1][j+1], alpha)
                        * pow(heuristic[i+1][j+1], beta);
                        if(!x.isNaN())
                            denom+=x;
                        else
                    denom+=0.0001;
            }
            if(i<width && j+1 <height && i>=0 && j+1 >=0){
                Double x= pow(trails[i][j+1], alpha)
                        * pow(heuristic[i][j+1], beta);
                if(!x.isNaN())
                    denom+=x;
                else
                    denom+=0.0001;

            }

            if(i+1<width && j <height && i+1>=0 && j >=0){
                Double x= pow(trails[i+1][j], alpha)
                        * pow(heuristic[i+1][j], beta);
```

```
                if(!x.isNaN())
                denom+=x;
            else
                denom+=0.0001;
        }

        if(i-1<width && j+1 <height && i-1>=0 && j+1 >=0 ){
            Double x= pow(trails[i-1][j+1], alpha)
                    * pow(heuristic[i-1][j+1], beta);
            if(!x.isNaN())
                denom+=x;
                else
                denom+=0.0001;
        }

        if(i+1<width && j-1 <height && i+1>=0 && j-1 >=0){
           Double x= pow(trails[i+1][j-1], alpha)
                    * pow(heuristic[i+1][j-1], beta);
            if(!x.isNaN())
                denom+=x;
                else
                denom+=0.0001;

        }

        if(i<width && j-1 <height && i>=0 && j-1 >=0){
            Double x= pow(trails[i][j-1], alpha)
                    * pow(heuristic[i][j-1], beta);
            if(!x.isNaN())
                denom+=x;
            else
                denom+=0.0001;
        }

        if(i-1<width && j <height && i-1>=0 && j >=0){
            Double x= pow(trails[i-1][j], alpha)
                    * pow(heuristic[i-1][j], beta);
            if(!(x.isNaN()))
                denom+=x;
                else
                denom+=0.0001;
        }

//  System.out.println("denom="+denom );
```

```
            if ( i-1>=width || j-1 >=height || i-1<0 || j-
1<0 || ant.visited[i-1][j-1] ) {
                probs[0] = 0.0;
            } else {
                Double numerator = pow(trails[i-1][j-1], alpha)
                        * pow(heuristic[i-1][j-1], beta);
                        if(numerator.isNaN())
                            System.out.println(trails[i-1][j-
1]+" "+heuristic[i-1][j-1]);
                probs[0] = numerator / denom;
            }

            if ( i+1>=width || j+1 >=height || i+1<0 || j+1<0  || ant.visited[
i+1][j+1]) {
                probs[7] = 0.0;
            } else {
                double numerator = pow(trails[i+1][j+1], alpha)
                        * pow(heuristic[i+1][j+1], beta);
                probs[7] = numerator / denom;
            }

            if ( i>=width || j-1 >=height || i<0 || j-1<0 || ant.visited[i][j-
1] ) {
                probs[1] = 0.0;
            } else {
                double numerator = pow(trails[i][j-1], alpha)
                        * pow(heuristic[i][j-1], beta);
                probs[1] = numerator / denom;
            }

            if ( i+1>=width || j-1 >=height || i+1<0 || j-
1<0 || ant.visited[i+1][j-1]) {
                probs[2] = 0.0;
            } else {
                double numerator = pow(trails[i+1][j-1], alpha)
                        * pow(heuristic[i+1][j-1], beta);
                probs[2] = numerator / denom;
            }

            if ( i-1>=width || j >=height || i-1<0 || j<0 || ant.visited[i-
1][j] ) {
                probs[3] = 0.0;
            } else {
                double numerator = pow(trails[i-1][j], alpha)
                        * pow(heuristic[i-1][j], beta);
```

```
                probs[3] = numerator / denom;
            }

            if ( i+1>=width || j >=height || i+1<0 || j<0 || ant.visited[i+1][
j] ) {
                probs[4] = 0.0;
            } else {
                double numerator = pow(trails[i+1][j], alpha)
                        * pow(heuristic[i+1][j], beta);
                probs[4] = numerator / denom;
            }

            if ( i-1>=width || j+1 >=height || i-
1<0 || j+1<0 || ant.visited[i-1][j+1]) {
                probs[5] = 0.0;
            } else {
                double numerator = pow(trails[i-1][j+1], alpha)
                        * pow(heuristic[i-1][j+1], beta);
                probs[5] = numerator / denom;
            }

            if ( i>=width || j+1 >=height || i<0 || j+1<0 || ant.visited[i][j+
1]) {
                probs[6] = 0.0;
            } else {
                double numerator = pow(trails[i][j+1], alpha)
                        * pow(heuristic[i][j+1], beta);
                probs[6] = numerator / denom;
            }
        /*
        //To see probablity of every town
        System.out.println();
        for(int k=0;k<n;k++)
            System.out.print(probs[k]+" ");
        */

    }

    // Given an ant select the next town based on the probabilities
    private void selectNextTown(Ant ant) {
        // calculate probabilities for each town (stored in probs)
        probTo(ant);
        int i=ant.tour[currentIndex][0];
        int j=ant.tour[currentIndex][1];
        //System.out.println("I am here 10");
```

```
        //select according to probs
        int next=-1;
        double max=0.0;
      /*  for(int a=0;a<8;a++)
            System.out.println(probs[a]);*/
       for (int k = 0; k < 8; k++) {
                //System.out.println("I am here 9 ");
            if (probs[k]>max){
                    next=k;
                    max=probs[k];
            }
        }

                if(next==0){
                    ant.visitPixel(i-1,j-1);
                    return;
                }
                else if(next==1){
                    ant.visitPixel(i,j-1);
                    return;
                }
                else if(next==2){
                    ant.visitPixel(i+1,j-1);
                    return;
                }
                else if(next==3){
                    ant.visitPixel(i-1,j);
                    return;
                }
                else if(next==4){
                    ant.visitPixel(i+1,j);
                    return;
                }
                else if(next==5){
                    ant.visitPixel(i-1,j+1);
                    return;
                }
                else if(next==6){
                    ant.visitPixel(i,j+1);
                    return;
                }
                else if(next==7){
                    ant.visitPixel(i+1,j+1);
                    return;
                }
```

```
                else{
                   // System.out.println("I am in selectnext, and something wr
ong");
                        ant.visitPixel(rand.nextInt(width),rand.nextInt(height
));
                        return;
                    //throw new RuntimeException("Not supposed to get here.");

                }



    }

    // Update trails based on ants tours
    private void updateTrails() {
        // evaporation
        for (int i = 0; i < width; i++)
            for (int j = 0; j < height; j++)
                trails[i][j] *= evaporation;

        // each ants contribution
        for (Ant a : ants) {
            double contribution = Q / a.tourIntensity();
            for (int i = 0; i < n - 1; i++) {
                trails[a.tour[i][0]][a.tour[i][1]] += contribution;
            }
        }
    }

    // Choose the next town for all ants
    private void moveAnts() {
        // each ant follows trails...
        //System.out.println("I am here 7");
        //This loop will do width*height
        while (currentIndex < n-1) {
            for (Ant a : ants){
                selectNextTown(a);
                //System.out.println("I am here 8 : Current Index"+currentInde
x);
            }
            currentIndex++;
            //System.out.println("I am here 9");
        }
    }
```

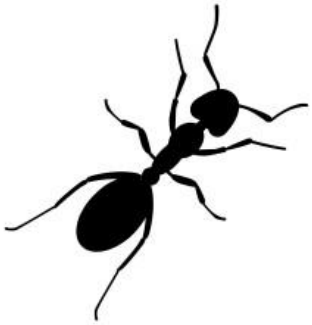
```
    private void setupAnts() {

        currentIndex = -1;
        for (int i = 0; i < m; i++) {
            ants[i].clear(); // faster than fresh allocations.
            ants[i].visitPixel(rand.nextInt(width),rand.nextInt(height));
           // System.out.println("I am here 6");
        }
        currentIndex++;

    }


    public void solve() {
        // clear trails
        for (int i = 0; i < width; i++)
            for (int j = 0; j < height; j++){
                trails[i][j] = c;
                //System.out.println("I am here 5");
            }


        int iteration = 0;
        // run for maxIterations
        // preserve best tour
       while (iteration < maxIterations) {
            System.out.println("----------------------
Iteration "+iteration+" ----------------------");
            setupAnts();
            moveAnts();
            updateTrails();
            //Displaytourlength();



            iteration++;
            System.out.println();
        }
        DisplayPheromoneMat();
        //DisplayHeuristics();
        PrintImage();
        // Subtract n because we added one to edges on load
       // System.out.println("Best tour length: " + (bestTourLength - n));
       // System.out.println("Best tour:" + tourToString(bestTour));
```

```
      //  return bestTour.clone();
    }

    public void PrintImage()
    {
        int a=255;
        int r=0;
        int g=0;
        int b=0;

        int p = (a<<24) | (r<<16) | (g<<8) | b;

        r=g=b=255;
        int q = (a<<24) | (r<<16) | (g<<8) | b;
        for(int i=0; i<width; i++)
            for(int j=0; j<height; j++){
                if(trails[i][j]>Theresold)
                    img.setRGB(i,j,p);
                else
                    img.setRGB(i,j,q);
            }

    try{
      f = new File("C:\\Users\\Somnath Paul\\Desktop\\Test\\Output.jpg");
      ImageIO.write(img, "jpg", f);
    }catch(IOException e){
      System.out.println(e);
    }

    }

    public void DisplayPheromoneMat()
    {
    System.out.println("Phreomone Matrix" );
        for(int i=0;i<width;i++){
        System.out.print("----------------------I am in  "+i+"----------------
-\n");
            for(int j=0;j<height;j++){
                System.out.print(" "+ trails[i][j]);
            }
            System.out.println(" ");
        }
    }

    public void DisplayHeuristics()
```

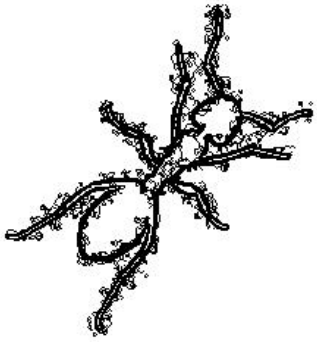
```
    {
     System.out.println("graph Matrix" );
        for(int i=0;i<width;i++){
        System.out.print("----------------------I am in  "+i+"----------------
-\n");
            for(int j=0;j<height;j++){
                System.out.print(" "+ graph[i][j]);
            }
            System.out.println(" ");
        }

    System.out.println("Heuristics Matrix" );
        for(int i=0;i<width;i++){
        System.out.print("----------------------I am in  "+i+"----------------
-\n");
            for(int j=0;j<height;j++){
                System.out.print(" "+ heuristic[i][j]);
            }
            System.out.println(" ");
        }
    }


    public static void main(String[] args) {
        // Load in TSP data file.
       /* if (args.length < 1) {
            System.err.println("Please specify a TSP data file.");
            return;
        }*/
        AntSystem anttsp = new AntSystem();
       try {
            anttsp.readGraph();
        } catch (IOException e) {
            System.err.println("Error reading graph.");
            return;
        }
            anttsp.solve();

    }
}
```

This code is implemented all by the programming language Java.

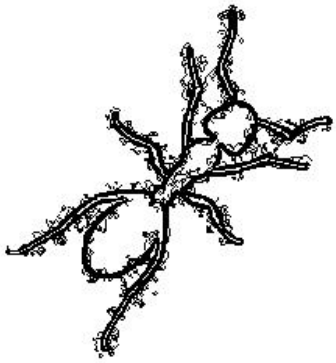
11 OUTPUT



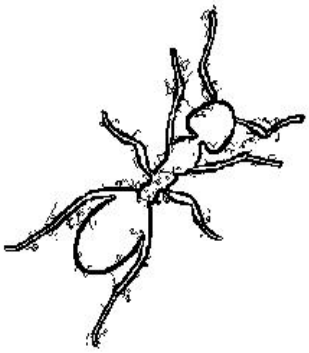
Original Image



Threshold value : 0.018



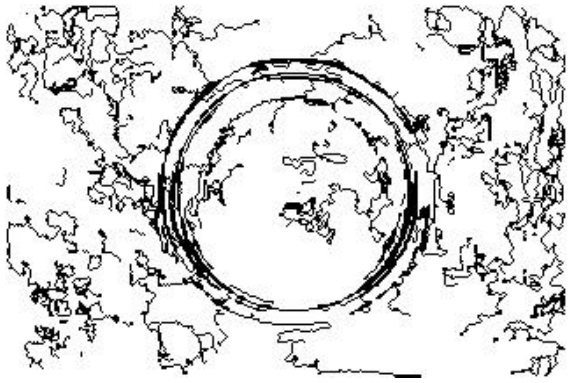
Threshold value: 0.019



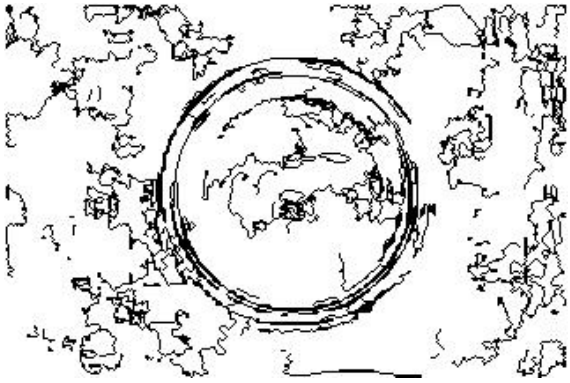
Threshold value: 0.021

Image Size	α	β	ρ	Q	Number of Ant	Number of Steps	Number of Iteration	Time
240*240	5.5	7.5	0.5	1000000	576	10000	100	4.23min

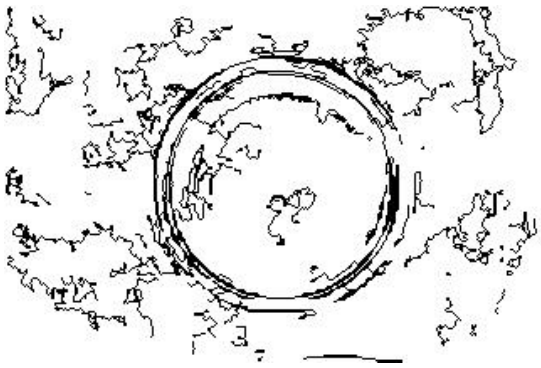
Original Image



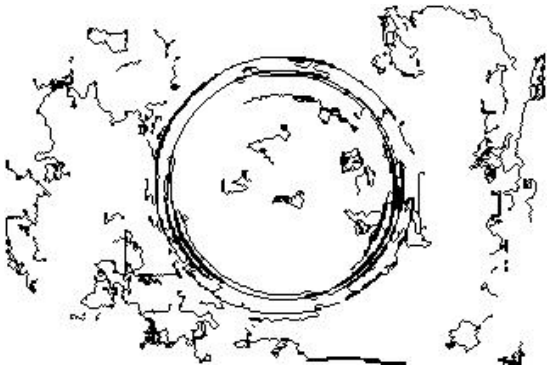
Threshold Value – 0.07



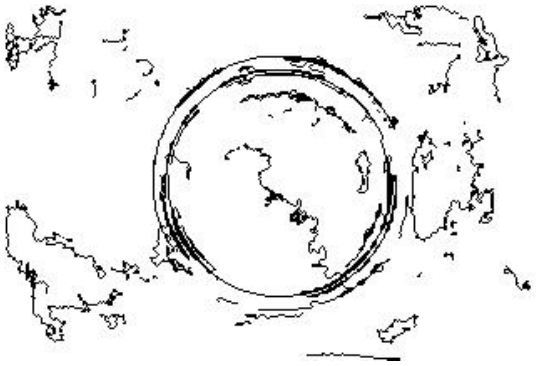
Threshold Value – 0.08



Threshold Value – 0.09



Threshold Value – 0.1



Threshold Value – 0.11

Image Size	α	β	ρ	Q	Number of Ant	Number of Steps	Number of Iteration	Time
320*213	5.5	7.5	0.5	1000000	576	10000	100	2.33min

Input Image

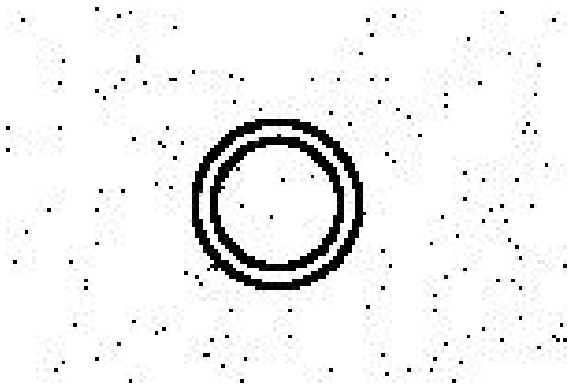
Threshold Value – 0.018

Threshold Value – 0.0175

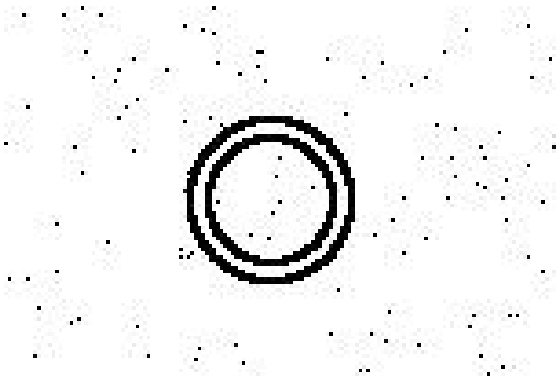
Threshold Value – 0.0179

Image Size	α	β	ρ	Q	Number of Ant	Number of Steps	Number of Iteration	Time
131*180	5.5	7.5	0.5	1000000	576	10000	100	1.03min

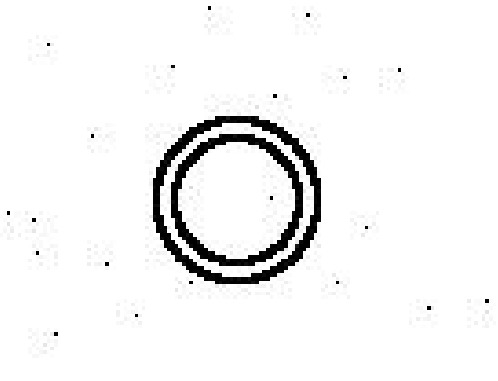
Input Image



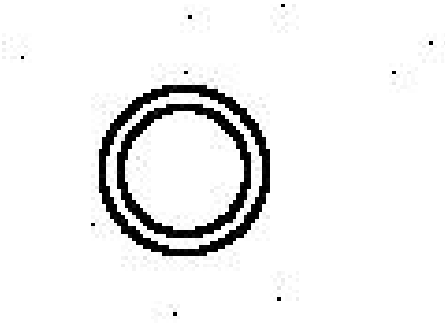
Threshold Value – 0.095



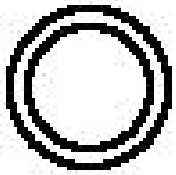
Threshold Value – 0.096



Threshold Value – 0.099



Threshold Value – 0.01



Threshold Value – 0.015

Image Size	α	β	ρ	Q	Number of Ant	Number of Steps	Number of Iteration	Time
151*101	5.5	7.5	0.5	1000000	576	10000	100	1.18min

So we will different Image Result for different parameter value of **alpha, beta, trail evaporation coefficient, trail deposit coefficient, Ant factor, Number of Iteration, Ant memory** and **Threshold**. We used here

12 CONCLUSION

An ACO-based image edge detection algorithm that takes advantage of the improvements introduced in ACS has been successfully developed and tested. Experimental results show the feasibility of the approach in identifying edges in an image. With suitable parameter values, the algorithm was able to successfully identify edges in the canonical test images. It must be noted that the appropriate parameter values depend on the nature of the image, and thus, may vary per application. As a continuation of this research, it is recommended to further examine how the quality of the extracted edges is affected by the parameter values and the functions for obtaining the heuristic information, for quantifying the quality of a solution, and for computing how much pheromone to deposit. In a study on a simplified ACO algorithm, it was shown that the basic properties of ACO are critical to the success of the algorithm, especially when solving more complex problems.

In recent studies, techniques that could enhance the performance of ACS have been explored. Here, ants are assigned different pheromone sensitivity levels, which makes some ants more sensitive to pheromone than the others. Here, multiple ant colonies with new communication strategies were employed. The proposed ACS method for edge detection could be extended and possibly be improved by making use of such techniques.

13 REPORTING BUGS

If you find a bug not listed there, please email it to somanathpaul119@gmail.com to create a new bug report.

14 BIBLIOGRAPHY

- Image Edge Detection Using Ant Colony : Anna Veronica Baterina and Carlos Oppus : January 15, 2010.
- Ant System: Optimization by a Colony of Cooperating : Marco Dorigo, Member, ZEEE, Vittorio Maniezzo, and Albert0 Colorni : December 28, 1994
- Edge detection using ant algorithms: Hossein Nezamabadi-pour · Saeid Saryazdi Esmat Rashedi : Published online: 1 August 2005 © Springer-Verlag 2005
- Edge Detection of an Image based on Ant Colony Optimization Technique, Charu Gupta, Sunanda Gupta : 6 June, 2013