APPLICATION OF ANT COLONY OPTIMIZATIONTECHNIQUE FOR MANETS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology
In
Computer science and Engineering
By

Sthitaprajna Jena

(10506035)

&

Jyoti prakash Lakra

(10506014)

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela-769008

2009

APPLICATION OF ANT COLONY OPTIMIZATIONTECHNIQUE FOR MANETS

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Bachelor of Technology
In
Computer science and Engineering
By

Sthitaprajna Jena

(10506035)

&

Jyoti prakash Lakra

(10506014)

Under the guidance of

Dr. S.K.RATH

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela-769008

2009

National Institute of Technology Rourkela

CERTIFICATE

This is to certify that the thesis entitled "**APPLICATION OF ANT COLONY OPTIMIZATION TECHNIQUE FOR MANETS**" submitted by Sri Sthitaprajna jena(Roll No.10506035) & Sri Jyoti prakash Lakra(Roll NO .10506014) in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Computer Science & Engineering at the National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by them under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Date: **Prof. Santanu kumar Rath**

ACKNOWLEDGEMENT

This project has been a dedicated effort towards development of a highly autonomous artificial intelligence, which primarily would not have been possible at the first place without the apt guidance of the respected Prof. S.K. Rath . His motivation and encouragement during the project made me go further than expected in the development process and the redaction of this current report.

I would also like to thank Prof .S. Chinara who the first explained to me the concept of Ants Algorithm and provided me with a relevant example of implementation.

I am also grateful to Prof. Banshidhar Majhi (Head of the Department) for assigning me this interesting project and for his valuable suggestions and encouragements at various stages of the work.

This dedication would not be complete without expressing my gratitude towards all my dear classmates for their interest and curiosity toward my project.

Sthitapraja Jena
Roll – 10506035
Dept. of Computer Science
& Engineering
National Institute of Technology
Rourkela- 769008

Jyoti prakash Lakra
Roll - 10506014
Dept. of Computer Science
& Engineering
National Institute of Technology
Rourkela- 769008

ABSTRACT

All networks tend to become more and more complicated. They can be wired, with lots of routers, or wireless, with lots of mobile nodes… The problem remains the same: in order to get the best from the network, there is a need to find the shortest path. The more complicated the network is, the more difficult it is to manage the routes and indicate which one is the best.

The Nature gives us a solution to find the shortest path. The ants, in their necessity to find food and brings it back to the nest, manage not only to explore a vast area, but also to indicate to their peers the location of the food while bringing it back to the nest. Thus, they know where their nest is, and also their destination, without having a global view of the ground. Most of the time, they will find the shortest path and adapt to ground changes, hence proving their great efficiency toward this difficult task.

The purpose of this project is to provide a clear understanting of the Ants-based algorithm, by giving a formal and comprehensive systematization of the subject. The simulation developed in Java will be a support of a deeper analysis of the factors of the algorithm, its potentialities and its limitations.

LIST OF CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1SWARM INTELLIGENCE

Swarm intelligence (SI) is a type of artificial intelligence based on the collective behavior of decentralized, self-organized systems. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems.[1].

SI systems are typically made up of a population of simple agents or boids interacting locally with one another and with their environment. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents.

PARTICLE SWARM OPTIMISATION

Particle swarm optimization (PSO) is a swarm intelligence based algorithm to find a solution to an optimization problem in a search space, or model.

ANT COLONY OPTIMISATION

The ant colony optimization algorithm (ACO), is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs.

This algorithm is a member of ant colony algorithms family, in swarm intelligence methods,the first algorithm was aiming to search for an optimal path in a graph; based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of Numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants.

1.2 OBJECTIVES

§ Propose an easy approach to the Ant Colony Algorithm, with appropriated vocabulary and global explanation, as well as details about its behaviour.

§ Develop a Java application which shows the working of the algorithm and gives a better understanding.

§ Give a straightforward analysis of the state-of-the-art studies on Ants-based Routing Algorithms and the implementations which have been done.

1.3 THE SOURCE OF INSPIRATION: THE ANTS

Ant as a single individual has a very limited effectiveness. But as a part of a well-organised colony, it becomes one powerful agent, working for the development of the colony. The ant lives for the colony and exists only as a part of it.

Each ant is able to communicate, learn, cooperate, and all together they are capable of develop themselves and colonise a large area. They manage such great successes by increasing the number of individuals and being exceptionally well organised. The self organising principles they are using allow a highly coordinated behaviour of the colony.

Pierre Paul Grassé, a French entomologist, was one of the first researchers who investigate the social behaviour of insects. He discovered[i] that these insects are capable to react to what he called “significant stimuli," signals that activate a genetically encoded reaction. He observed that the effects of these reactions can act as new significant stimuli for both the insect that produced them and for the other insects in the colony. Grassé used the term *stigmergy* to describe this particular type of indirect communication in which *“the workers are stimulated by the performance they have achieved”*.

Stigmergy is defined as a method of indirect communication in a self-organizing emergent system where its individual parts communicate with one another by modifying their local environment.

Ants communicate to one another by laying down pheromones along their trails, so where ants go within and around their ant colony is a stigmergic system.

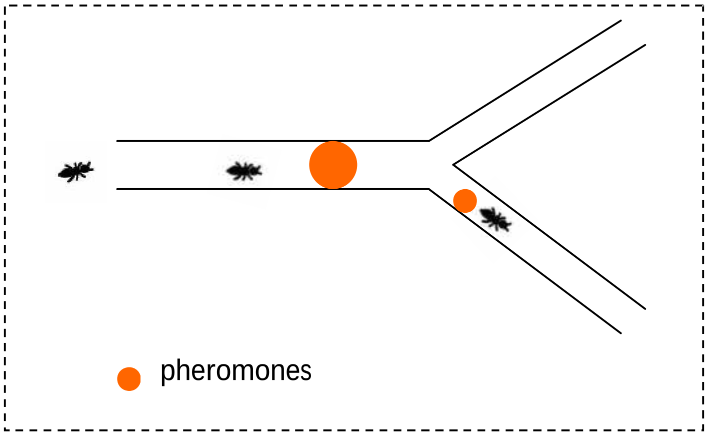
In many ant species, ants walking from or to a food source, deposit on the ground a substance called *pheromone*. Other ants are able to smell this pheromone, and its presence influences the choice of their path, that is, they tend to follow strong pheromone concentrations. The pheromone deposited on the ground forms a pheromone trail, which allows the ants to find good sources of food that have been previously identified by other ants.

Using random walks and pheromones within a ground containing one nest and one food source, the ants will leave the nest, find the food and come back to the nest. After some time, the way being used by the ants will converge to the shortest path.

1.4 THE DOUBLE BRIDGE EXPERIMENT

The ants begin by walking randomly. They cannot see the ground and have a very limited view of what is around them. Therefore, if the ground has not been explored yet, they will just wander and take random decision at each crossroads.

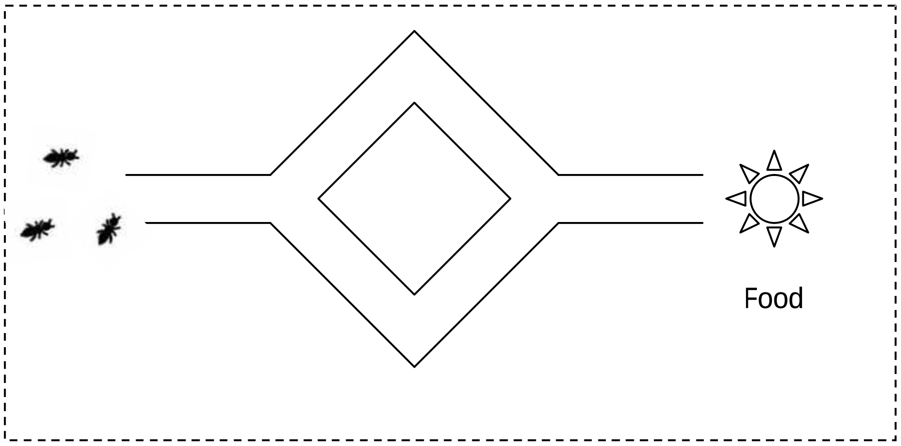
After a while, the places around the nest will be all explored. The ants will get to know that by the marking done by the previous ants. Indeed, they will leave behind them the famous pheromones and inform the other ants that the way is already explored.



FigError! No text of specified style in document.**.1 Ants and pheromones**

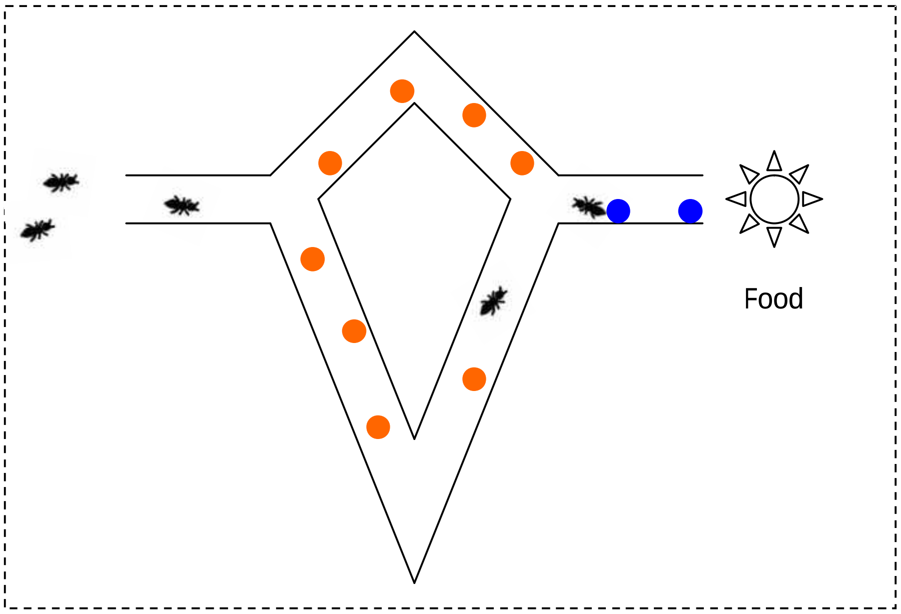
The concentration of pheromones depends on the number of ants who took the way, the more ants taking the way, the more pheromones.

The configuration is as shown in *figure 1.1*: the nest of a colony of ants is connected to the food via two bridges of the same length. In such a setting, ants start to explore the surroundings of the nest and eventually reach the food source. Along their path between food source and nest, ants deposit pheromones. Initially, each ant randomly chooses one of the two bridges. However, due to random fluctuations, after some time one of the two bridges presents a higher concentration of pheromones than the other and, therefore, attracts more ants. This brings a further amount of pheromone on that bridge making it more attractive with the result that after some time the whole colony converges toward the use of the same bridge.



FigError! No text of specified style in document.**.2 The Double Bridge Experiment**

The second experimentation, *figure1.3* gives also two paths to the food source, but one of them is twice longer than the other one. Here again the ants will start to move randomly and explore the ground. Probabilistically, 50% of the ants will take the short way while the 50% others will take the long way, as they have no clue about the ground configuration.



FigError! No text of specified style in document.**.3 Ants coming back to the nest**

The ants taking the shorter path will reach the food source before the others, and leave behind them the trail of pheromones. After reaching the food, they will turn back and try to find the nest. At the cross, one of the paths will contain pheromones although the other one will be not explored. Hence the ant which carries the food will take the path already explored, as it means it is the way to the nest.

As the ant is choosing the shortest way and will continue to deposit pheromones, the path will therefore become more attractive for others. The ants who took the long way will have more probability to come back using the shortest way, and after some time, they will all converge toward using it.

Consequently, the ants will find the shortest path by themselves, without having a global view of the ground. By taking decision at each cross according to the pheromones amount, they will manage to explore, find the food, and bring it back to the nest, in an *optimized* way.

1.5 THE PHEROMONES

Pheromones represent in some ways the common memory. The fact that it is external and not a part of the ants / agents, confers to it an easy access for everyone. The memory is saved in without regarding the configuration of the ground, the number of ants, etc. It is totally independent, and still remains extremely simple.

In our implementation we will see that two different types of pheromones are used. The first one is represented in red and is let by the ants which do not carry the food. We will call it the *Away* pheromone, as it means that the ant is going away from the nest. Oppositely, the ants which carry the food to bring it back to the nest let a blue trace behind them, the *Back* pheromone.

Pheromones just proceed to one task: nature will take care of it in the real life, although it is a simple process in algorithms. In course of time, a global reduction of the pheromones by a certain factor is applied, simulating the *evaporation* process. Thus the non-succeeding path will see their concentration of pheromones reduced, although good solutions will stay full of pheromones as the ants keep using it.

CHAPTER 2

IMPLEMENTATION OF THE ANTS-BASED ALGORITHM IN JAVA

2.1 THE IDEAS BEHIND THE SIMULATION

2.1.1 SELECTION OF THE LANGUAGE

After deciding to code the algorithm, the first step is to decide which language to use. The algorithm is entirely based on objects which are independent one another. Therefore the choice goes straight to any Object-Oriented language, the most common ones being C++, C# and Java.
Since the application needs a GUI, the language needs to provide an easy access to the graphical environment with a simple interface with the mouse or keyboard.
Java is very polyvalent and really applies to these requirements. It is also multi-platform and with the applet helps, the application can be available on any internet webpage.

2.1.2 OBJECT MODELLING

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. The rules are used for interpretation of the meaning of components in the structure.
In order to develop an application which really suits to the definitions, the first action is to give a complete modeling of the specifications. First the problem will be divided into classes, linked to one another and resuming the simulated world.
We can easily define 3 main classes:

§ *Ant*. Agent which can smell the pheromones and make a move on the ground.
§ *Ground*. Here we will call it Playground; it is the 2-dimension space which contains the ants and the pheromones.
§ *Pheromones*. Or traces, they are the trails which partially lead the ants

The next modeling step is to define the relationships between these objects. There are 4 main types of link, each of them basically defining the existence of a class when the other class is destroyed. UML shows the following relationships:

§ Link. Basic relationship among objects.
§ Association. Family of links whose multiplicity is unlimited.
§ Aggregation. When a class is a collection of other classes, but where, if the container is destroyed, its content is not.

Composition. The multiplicity has to be 0…1 or 1, and when the container is destroyed, its content is destroyed as well.

The Playground is the central class, the ants and the traces are around it. We can precise that there is no trace without playground, but there can be ant. Traces are linked to the playground in the Composition way; if the playground is destroyed the traces are destroyed as well. The ants are a collection of the playground, but they are independent from it. They remain ants if the playground is destroyed. They are linked by an Aggregation link.

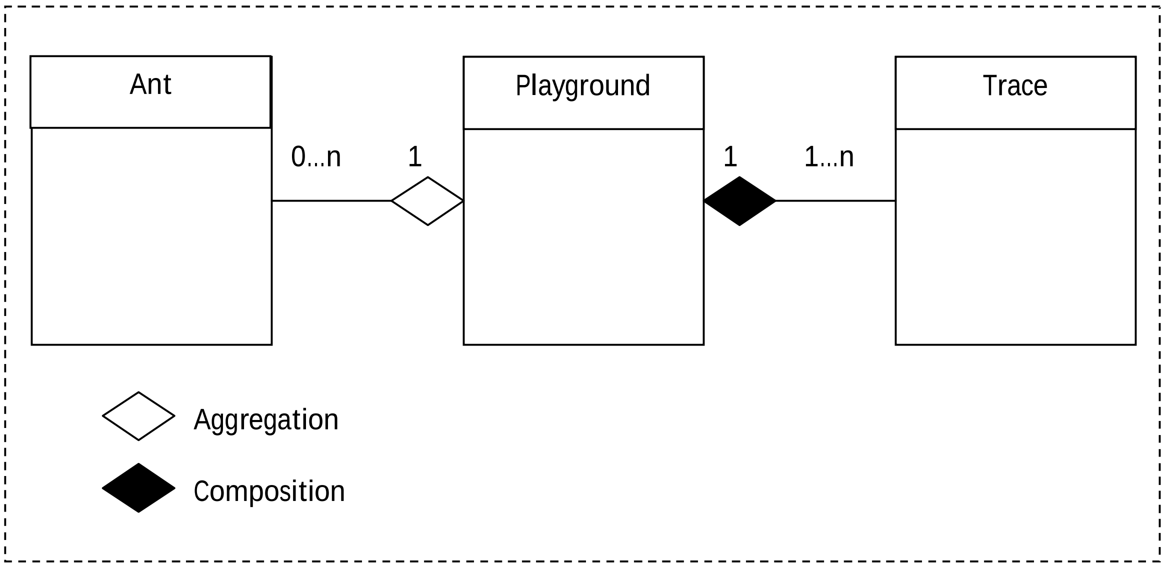


Fig 2.1. Basic Class Diagram

One can notice that there is no link between the ants and the pheromones. This denotes the independence between them; the traces are the memories of the ants but the ants do not carry them. The ants remain simple agents which do not have memory, and simply take decision according to their position and what surround them.

If we go deeper in the modeling process, we need to define packages. They are sets of classes which realize the same kind of tasks, or take care of the same problem. Here we have 3 packages, including the default one:

- Default Package
- Object Package
- GUI Package

The default package which simply contains the starting point of the application, that is the main method. In the Object package we can find the ants, the playground, the traces, and other classes which are abstractions of acting systems. The GUI package includes all the graphical and controlling part.

CLASS DIAGRAM

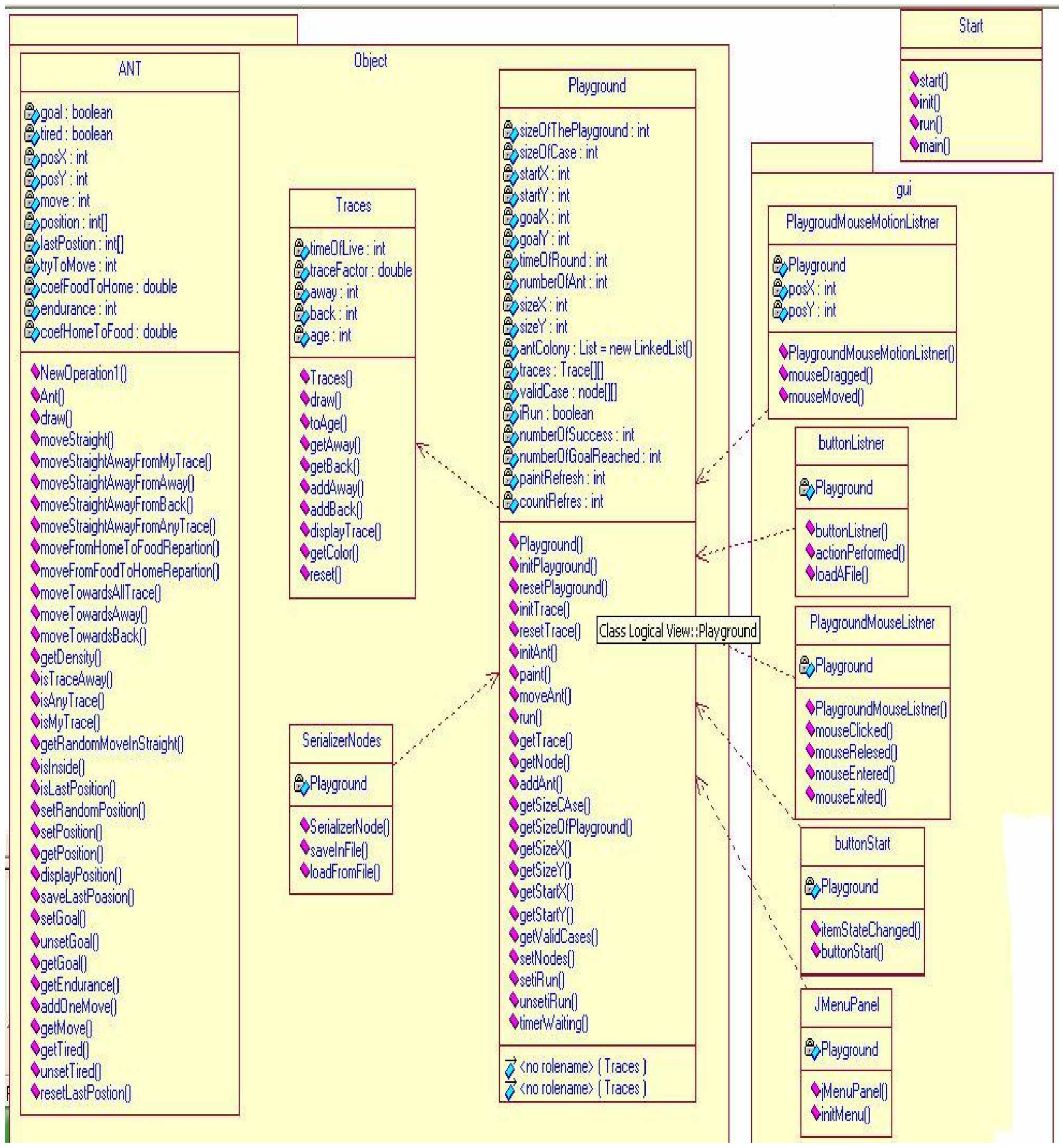


Fig 2.2 Class Diagram

2.2 THE TWO MAIN CLASSES

The code works with 2 main classes: the ant and the playground, plus a very simple one, the traces. After defining these classes, we need to define their attributes and their methods. That means what they own and what they are capable of.

2.2.1 THE CLASS ANT

Attributes

An ant is an independent and autonomous object which is capable to move. First let's have a look to its attribute: each ant is having a position, and two booleans tired and goal. This way, we get to know if the ant is tired and need to come back to the nest, and also if it has reached the goal or not. The position is stored in 2 integers, posX and posY, as well as in an array of 2 integers position[]. The last position is also saved, as the ant even if not the cleverest creature, still remembers its last position and does not go back at every step.

```
        private boolean goal, tired;
        private int posX, posY, move;
        private int[] position, lastPosition;
```

The static attributes are fixed values which define the factors being used. We will find the endurance of an ant – understand the number of moves it will make before being tired and having to come back to the nest, the number of tries for a new position – that is the time taken to decide which way to take, and also the factors regarding the pheromones impact on the decision process. These all values will be explained at the time they are used, in the Methods part of this chapter.

```
        private static int tryToMove = 7;
//   Number   of   tries   to   find   the   best   next   position   {8}
        private static int endurance = 80;
//    Number    of    moves    before    the    ant    get    tired    {80}
        private static double coefFoodToHome = 1.2;
//   Privileges   the   Back   amount   over   the   Away   amount   {2}
        private static double coefHomeToFood = 2;

// Privileges the Away amount over the Back amount {2}
```

The comments explain the meaning and also give the default values.

All the attributes are given as private; neither the playground nor any other class can access them. This explains the creation of the accessors, which are simple methods allowing the other class to access these values, to change it or just read it:

```
public void setPosition(int posX, int posY){
        public int getPosition()[]{
        public void displayPosition(){
        public void saveLastPosition(){
        public void setGoal(){
        public void unSetGoal(){
        public void getGoal(){
        public int getEndurance(){
        public void addOneMove(){
        public int getMove(){
        public void setTired(){
        public boolean getTired(){
        public void unsetTired() {
        public void resetLastPosition() {
```

The effects of these accessors are directly explained in the name of each method. By example, setTired() will change the boolean *tired* to **true**, unSetTired() will change it to **false**. If another class wants to know if a particular ant ant1 is tired, this class can call the getTired() method by ant1.getTired(). The return value is a boolean, which value is **true** if the ant ant1 is tired.

METHODS

The ants have some capabilities that are reflected by their methods. Some of them are private and can be called by the ant itself and no one else, and some are public and are called by the playground.

The private methods are there to help the public methods in the move process. *In fact, the algorithm for the ants to move is the central idea of the whole program*. The more effective the algorithm is, the more acceptable the obtained solutions are. These private methods give random moves, analyses of the playground and the traces, or the validity of a move. The private methods are shown here:

```java
private boolean isTraceAway(int x, int y, Playground p){
private boolean isTraceBack(int x, int y, Playground p){
private boolean isAnyTrace(int x, int y, Playground p){
private boolean isMyTrace(int x, int y, Playground p){
private int[] getRandomMoveInStraight(int x,int y, Playground p){
private boolean isLastPosition(int x, int y){
private boolean isInside(int x, int y, Playground p){
```

The arguments are always the position, int x and int y, and also the playground p. Indeed the methods need information of the current playground, like its size, the traces it contains…

The getRandomMoveInStraight() method is very important, it is used many times in all movements. It generates a new position following this algorithm:

```
//GetRandomMoveInStraight()

DO{
Generate a random move.
} WHILE new position is a diagonal move OR not inside the playground OR
not on a Valid Case OR the last Position.
Return the new position[].
```

This way is much better than getting all the possibilities and picking one of them, at it gives equal results and keeps a strong random part. This random part is a base for the whole movement algorithm, as no direction should be privileged.

To go through all the possibilities and get the best one is one solution which forgets the natural part of the move, it should be proscribed.

The algorithm being actually used take consideration of one more detail: if the ant went into a dead end, the number of tries should be limited. We use a threshold of 60 tries before removing the condition of the last position.

To get a valid move inside the playground which respects the chances is one step. To really get the new position, various algorithms are used according to whether the ant found the food or not, the ant is tired, the food has been found… The decision of the algorithm to be used is made in the Playground class, which manages all the moves.

The following methods are the fourth kind of move used by the ants. Three factors decide which move to perform, that is, which algorithm to use: if the ant is tired, if the ant is carrying the food, and also if the goal has ever been reached. All combine it gives 4 possibilities:

- From Home to Food:
    - The goal has never been reached
    - The goal has been reached
- Back Home:
    - The ant is tired
    - The ant has found the food

The 4 algorithms are described in the following.

If the goal has never been reached, that means, the starting point:

The ants have to explore the largest possible area. In order to do that, the ants have to move away from their traces. If the nest is in the middle and there is no obstacle, they will move in a circle which will become larger at each step.

We again want to keep a strong random part in the decision process of each movement. The MoveStraightAwayFromAway() method uses the precedent algorithm and take in consideration the presence of traces:

```
//MoveStraightAwayFromAway()
count = 0;
DO{
Get a generated random move.
Increase count.
If count > tryToMove, then break.
} WHILE new position is on pheromones
Set the position to the new values
```

tryToMove is a static value defined in the Attributes part (see first page of Chapter 2.2.1). The purpose is to explore the bigger area possible, so we just generate new positions and see if these positions are already marked or not. If not, the position is set to the current ant.

This method also respects the probability's rules as much as possible.

If the ant is tired and needs to come back to the nest

Then the ant needs to come back to the nest, using the shortest path. The algorithm we are using is different from the others as it analyses all the possibilities and pick the best one. The algorithm and its implementation show good results, that is why it has been chosen to keep it.

The method is named moveTowardAway(), as we go with the principle that the density of Away pheromones is higher when the ant is close to the nest. When all the ants leave the nest, they leave the red pheromones, called Away pheromones, and then all go in various directions. So, as the ants advance, the pheromones concentration will be less around them. When they want to come back, the best way is to follow the pheromones and move toward the higher concentration.

The algorithm is as follow:

```
// MoveTowardAway()
Maximum = 0;
For all the cases around the ant, horizontally and vertically {
  Get the pheromones amount;
  If the amount is higher than maximum, keep its value and memorize the
position;
}
If the position is the same than last position or no position memorized
{
  Give a random new position.
}
```

This algorithm shows good effectiveness: whenever the ants are tired they are able to come back to the nest, depending on the playground configuration.

If the ant found the food and needs to come back to the nest

Subsequently the ant will follow the same red traces – away traces, but it will use the blue traces as well. The ants can let two types of pheromones behind it, whether it got the food or not.

By using the *back* pheromones, we insure convergence to one single path. However, the convergence should not be too fast and both amounts of pheromones back and away should be analysed. The algorithm is quite similar to the others, and the factor coefFoodToHome plays a very important role.

```
//MoveFromFoodToHome()
maximum = 0;
count = 0;
DO{
  Increase count;
  Generate a random move;
  Get the value of both traces away and back of the new position;
  Calculate Away * coefFoodToHome – Back;
  IF the result is higher than maximum and new position is not last
position, remember the new position and update the value of maximum;
} WHILE count < tryToMove;
IF no new position found, moveAwayFromBack();
```

moveAwayfromBack() is very similar to moveAwayfromAway(), just the condition changes: if there is a Back trace on the new position, we draw a new random position. Following is the Java code of the methods.

```
//MoveFromFoodToHomeRepartition
//MoveFromFoodToHomeRepartition
public void moveFromFoodToHomeRepartition(int x, int y, Playground p){
         int sizeCase = p.getSizeCase();
         Traces t;
         int count = 0;
         double value = 0, max = 0;
         int[] newPos = new int[2];
         int[] tryNewPos = new int[2];

         do{
            tryNewPos = getRandomMoveInStraight(x,y,p);
            count++;
            t = p.getTrace()[tryNewPos[0]][tryNewPos[1]];
            value = t.getAway() * coefFoodToHome - t.getBack();
            if((value > max)&&(!isLastPosition(tryNewPos[0],
tryNewPos[1]))){
               max = value;
               newPos = tryNewPos;
            }
         } while(count < tryToMove);

         if((newPos[0] == 0)&&(newPos[1] == 0))
             this.moveStraightAwayFromBack(x,y,p);
         else this.setPosition(newPos[0], newPos[1]);
        }
```

The best solution is given as the one whose *value* is maximal, i.e. the one where the amount of Away traces multiplied by the factor *coefFoodToHome* and reduced by the amount of Back traces is the higher. Therefore, we try to follow as much as possible the Away pheromones while trying to avoid the back pheromones. We use the idea that the density of back pheromones is higher when the ant is close to the source of the food, and less when coming to the nest. However, the shortest path will be covered by the Back pheromones as all ants should take it to come back. That is the reason why we use the factor *coefFoodToHome*, which value is around 2 – depending on the configuration of the playground.
The optimal value depends on many parameters and "2" seems to give good results.

If the goal has been found and the ant carrying the food back to the nest
The algorithm is exactly the same than the previous *MoveFromFoodToHomeRepartition()*. In the same way the algorithm will try a number of times equal to *tryToMove* and the best solution will be picked among them.

2.2.2THE PLAYGROUND CLASS

The playground plays the role of the manager in our application. It will be the one which manages the moves and the output in a graphical mode. We will not go into the details of the code and stick to the algorithm part. The full code of the class can be found in the Appendix part.

Attributes

A playground is defined by a size and the position of the start point and the source of the food point. As it manages the main processes, it also contains the number of Ants being used and the time between each round, and the size of the case for the graphical representation.The ants are stored into a Linked List instead of an array. It gives the advantage of the declaration while instancing the colony that they represent. One more solution would have been to define the colony as an object instead of defining a single ant. However, we have seen that each ant should be fully independent from the others. Consequently an ant is a simple object and the playground includes a list of them.Regarding the pheromones, we assign the values to an array of two dimensions. However, we use a special class to keep the pheromones, called Trace. A trace is an object containing two values, away and back, which is able to age with the time. Hence, the playground will contain an array of traces, which can be represented like a 3-dimensional array.

```
private static int sizeOfThePlayground = 40;  // 40
    private static int sizeOfCase = 14;           // 14
    private static int startX = 10, startY = 10;  //10,10
    private static int goalX = 25, goalY = 25;    // 25,25
    private static int timeOfRound = 150;         // in ms, 150
    private static int numberOfAnts = 40;         // 40
    private List antColony = new LinkedList();
    private Traces[][] traces;
    private Nodes[][] validCase;
    private boolean goalReached, iRun;
  private int numberOfSuccess, numberOfGoalReached;
```

The boolean *iRun* is used to control the execution of the movement process. In fact, the application uses thread in order to keep the access to the buttons while it is running. *iRun* starts and stops the movement process.

goalReached is used to inform all the ants that the source of the food has been found. In the real life, ants will communicate about this information while meeting one another. This phenomena is approximated by giving a binary value to *goalReached*.

Number Of Success and number Of Goal Reached are kept to be displayed in real time.

METHODS

The Playground class is having all the contructors which are called when starting the application: initPlayground(), initTrace() and initAnt();It also contains all the accessors, which as seen before are called by the other classes to get a value or to change it.More important, the Playground class contains three major methods. The first one manages the display and is called after every move. It draws the playground, the ants and the traces and refreshes them whenever required.

```
//Paint

Draw the white background
FOR each case in the playground {
   If it is not a valid case, draw a wall;
   Else draw the traces back and away;
}
For each Ant
   Calls the draw methods of the ant itself;
Draw the Nest and the source of Food;
Display the numberOfSuccess and numberOfGoalReached values;
```

Thus the paint() method manages all the display part. The second really important method is the one which manages the movements of the ants. Each step is defined in it:

```
//moveAnts() – equivalent to one step
Age the traces;
For each ant {
  Check if the ant is on Start or Goal point;
  Add a Back or Away trace depending on getGoal() binary value;
  Increase the numberOfMove by one and check whether the ant is tired or
not;
  If the ant is tired: moveTowardAway();
  Else if the goal has never been reached: moveStraightAwayFromAway();
  Else if the ant is carrying the food: moveFromFoodToHomeRepartition();
  Else moveFromHomeToFoodRepartition();}
```

This method which seems simple actually calls methods more complicated, which has been seen before. The whole process is managed here, but this moveAnts() method and that paint() method are themselves called by the run() function. This run() function is mandatory when using the threads: after creating the thread, the process is free to run by itself as well as its father. The run() is then executed. However, we want to be able to stop the process, and as seen before, we have the boolean *iRun* whose binary value can be changed, through an accessor. Therefore, the run() method's code is quite simple and actually manages the whole application:

```
    public void run() {
        while(iRun){
            timerWaiting(timeOfRound);
            moveAnts();
            repaint();
        }
    }
```

timerWaiting() makes the thread wait for a time period equal to *timeOfRound*. Then the ants make one new step and the display is refreshed by using repaint(), which will actually call the paint() method.

CHAPTER 3

APPLICATION

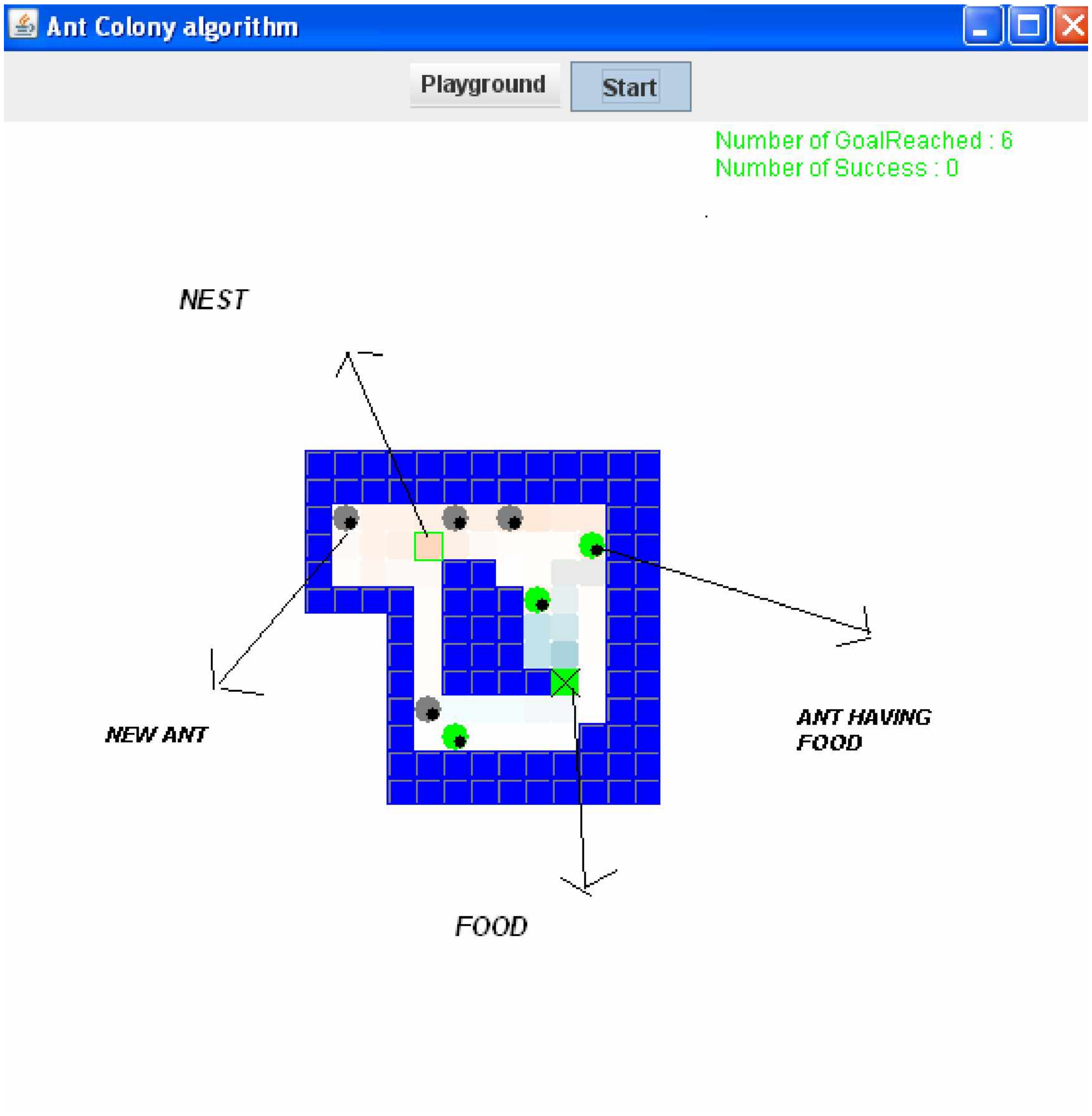
3.1 HOW TO USE, HOW IT WORKS

Java provides the capability to use different panels into a same frame. The whole application is included in a JFrame, itself containing two JPanels. The first one is the controller part, with the menu and the Start button. The second is the playground, started as an applet.

The figure 3.1 shows the look of the application. At the top, the controller part allows the user to start / stop the process and a menu gives the ability to manipulate the playground.

The playground is implementing the mouseListener methods. We use these methods to draw the nodes on the playground with the mouse, by simply clicking on the playground. If the case is already a wall, then the wall will be removed. This way, we can create multiple scenarios and tests the algorithms for various configurations, like the one shown in Figure 3.1.

The Start/Stop toggleButton controls the execution of the thread by changing the boolean value of iRun, as seen previously



Number of GoalReached : 6
Number of Success : 0

Fig 3.1. **The Application's Look**

APPLICATION

Ant colony optimization algorithms have been applied to many combinatorial optimization problems. Ant colony optimization algorithms have been used to produce near-optimal solutions to the travelling salesman problem. The first ACO algorithm was called the Ant system [5] and it was aimed to solve the travelling salesman problem, in which the goal is to find the shortest round-trip to link a series of cities. The general algorithm is relatively simple and based on a set of ants, each making one of the possible round-trips along the cities. At each stage, the ant chooses to move from one city to another according to some rules:

1. It must visit each city exactly once;
2. A distant city has less chance of being chosen (the visibility);
3. The more intense the pheromone trail laid out on an edge between two cities, the greater the probability that that edge will be chosen;
4. Having completed its journey, the ant deposits more pheromones on all edges it traversed, if the journey is short;
5. After each iteration, trails of pheromones evaporate.

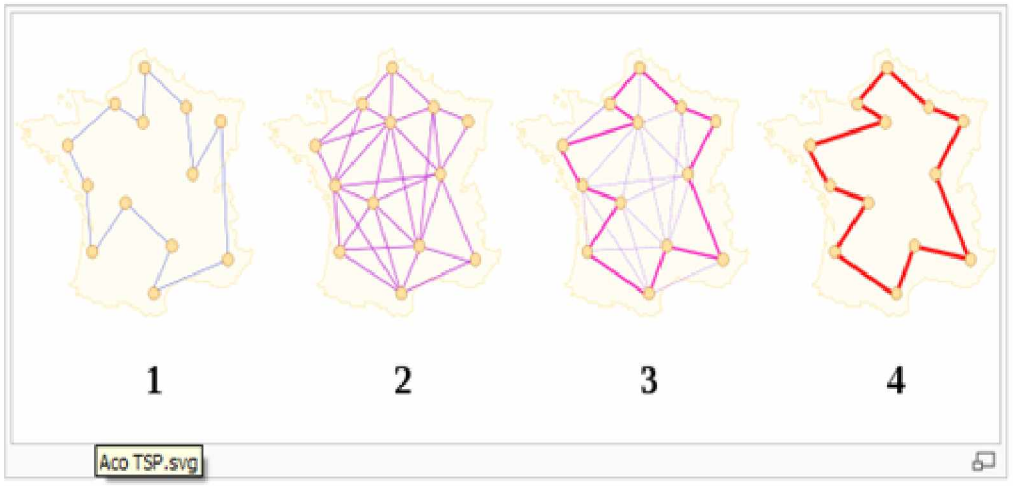
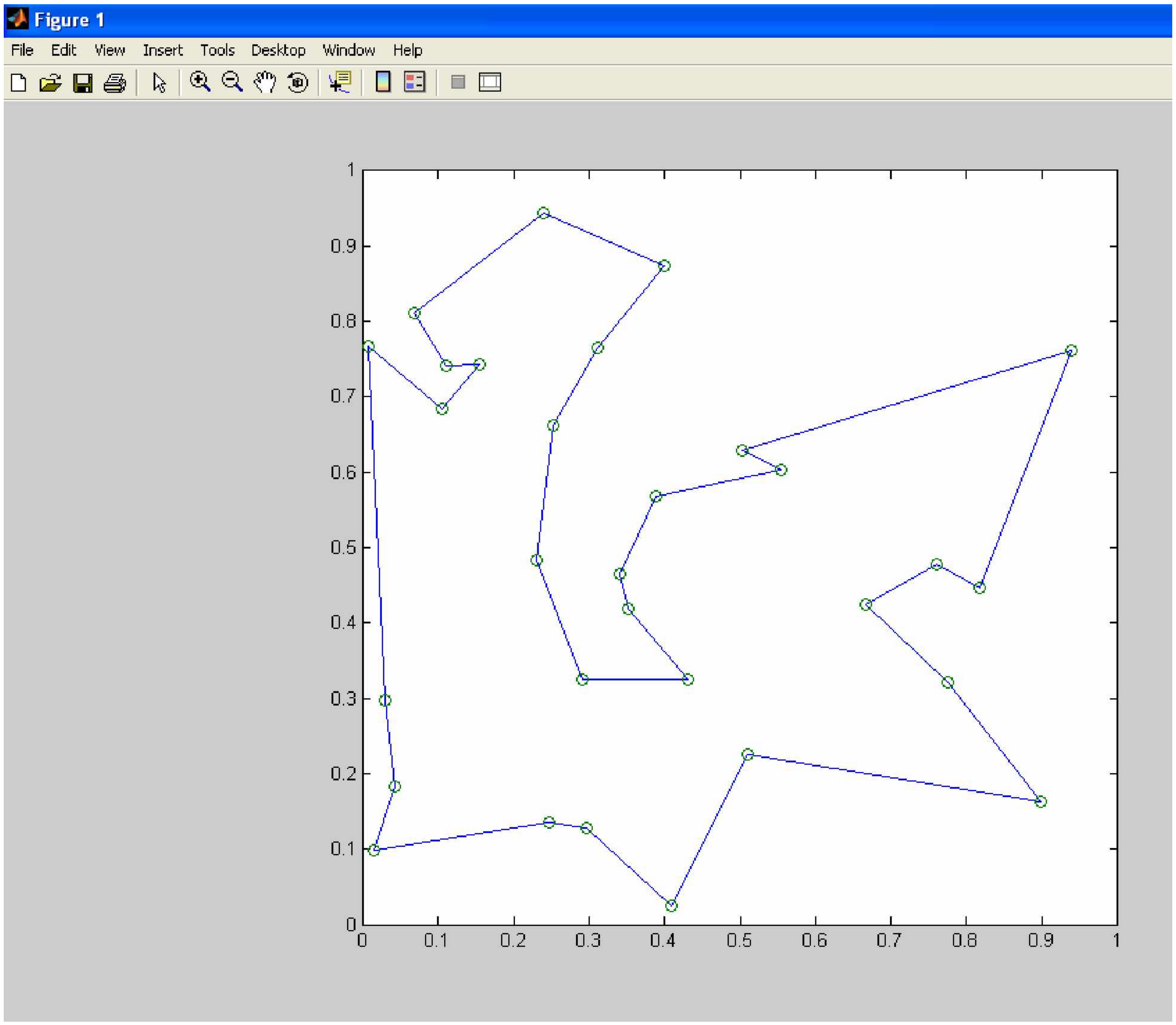


Fig 3.2 TSP



3.3 ant colony optimization for solving the traveling salesperson in matlab

CHAPTER 4

USE OF ACO IN MANETS

Ant Colony Optimization (ACO) is a paradigm for designing met heuristic algorithms for combinatorial optimization problems. The first algorithm which can be classified within this framework was presented in 1991 [21, 13] and, since then, many diverse variants of the basic principle have been reported in the literature.
The essential trait of ACO algorithms is the combination of a priori information about the structure of a promising solution with a posteriori information about the structure of previously obtained good solutions.

Ant Colony Optimization algorithms and their applications in Mobile Ad-hoc Networks have been studied for long. A couple of popular algorithms which aim to alleviate the problems in Mobile Ad-hoc Networks are the AntHocNet Algorithm and ARA (Ant colony-based Routing Algorithm). Both the routing algorithms extensively use the concept of pheromones and pheromone tables. AntHocNet, a more lucrative solution of the two, is a hybrid algorithm, which has both proactive and reactive features. Hybrid Algorithms, despite their better performance are not very popular due to the heavy overhead of the proactive algorithm (continuous
Exchange of routing information, etc between neighbors).Also AntHocNet uses end-to-end delay as a metric to
Calculate congestion at a node, which may not yield accurate results as end-to-end is affected by both congestion as well as the length of the route from source to destination.
ARA is an on-demand routing algorithm similar to DSR,AODV and uses pheromone tables at each node to calculate
the best route to reach the destination. However the pheromones merely enhance routes without any dependency
on congestion, load or energy metrics. Such a mechanism always enhance the shortest routes although it might be
Congested or non-efficient in terms of energy/load distribution.
Hence the suggested algorithm, being on-demand, has low overhead and at the same time tries to optimize all the
Parameters (energy, congestion, load balancing) of the network along-with providing efficient routing.

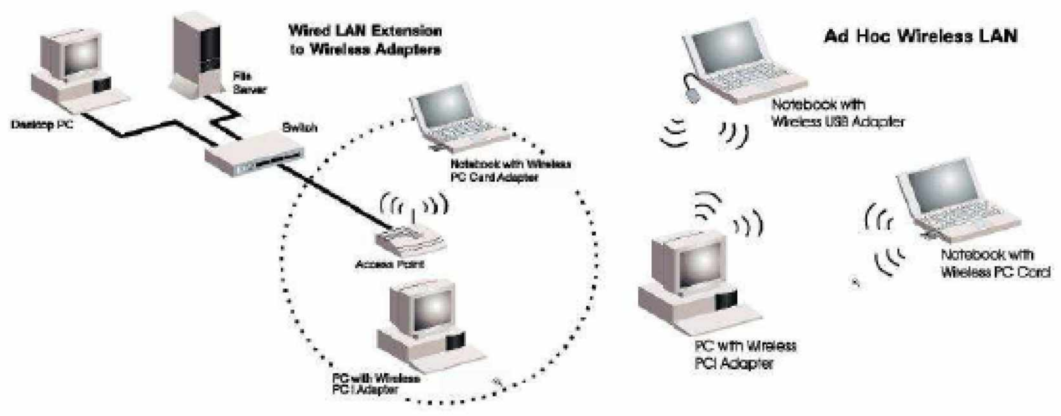

Wired LAN Extension to Wireless Adapters
File Server
Desktop PC
Switch
Notebook with Wireless PC Card Adapter
Access Point
PC with Wireless PCI Adapter
Ad Hoc Wireless LAN
Notebook with Wireless USB Adapter
Notebook with Wireless PC Card
PC with Wireless PCI Adapter

Figure 3.1: Operation modes of IEEE 802.11

Short algorithm description

AntHocNet is an adaptive routing algorithm for mobile ad hoc networks (MANETs) inspired by ideas from Ant Colony Optimization (ACO). In common MANET terminology, AntHocNet could be called a hybrid algorithm, as it combines both reactive and proactive routing strategies. Specifically, the algorithm is reactive in the sense that it does not try to maintain up-to-date routing information between all the nodes in the network, but instead concentrates its efforts on the pairs of nodes between which communication sessions are taking place. It is proactive in the sense that for those ongoing communication sessions, it continuously tries to maintain and improve existing routing information.

To gather routing information, the AntHocNet algorithm uses two complementary processes. One is the repetitive end-to-end path sampling using artificial ant agents. The other is what we call pheromone diffusion, an information bootstrapping process that allows to spread routing information over the network in an efficient way. While the ant-based path sampling is the typical mode of operation of ACO routing algorithms, the pheromone diffusion process is in its working more similar to Bellman-Ford routing algorithms. AntHocNet combines both processes in order to obtain an information gathering process that is at the same time efficient, adaptive and robust.

4.2 METHODOLOGY

4.2.1Reactive path setup

Reactive forward ants looking for a destination *d* are either broadcasted or uni-casted, according to whether or not the node they are currently in has routing information for *d*. Due to the broadcasting, ants can proliferate quickly over the network, following different paths to the destination. When a node receives several ants of the same generation (i.e., they started as the same original forward
ant at the source), it will compare the path travelled by the ant to that of the previously received ants of this generation: only if its number of hops and travel time are both within a certain factor (a parameter which we empirically set to 1.5) of that of the best ant of the generation, it will forward the ant. Using this
policy, overhead is limited by removing ants which follow bad paths, while the possibility to find multiple good paths is not hindered.

The main task of the reactive forward ant is to find a path connecting *s* and *d*. It keeps a list *P* of the nodes [1*; : : : ; n*] it has visited. Upon arrival at the destination *d*, the forward ant is converted into a *backward ant*, which travels back to the source retracing *P*. The backward ant incrementally computes an
estimate ^ *TP* of the time it would take a data packet to travel over *P* towards the destination, which is used to update routing tables.

4.2.2 Proactive path maintenance and exploration

While a data session is running, the source node sends out proactive forward ants according to the data sending rate (one ant every *nth* data packet). They follow the pheromone values in the same way as the data (although the pheromone values are not squared, so that they sample the paths more evenly), but have a small probability at each node of being broadcasted. In this way they serve two purposes. If a forward ant reaches the destination without a single broadcast it simply samples an existing path. It gathers up-to-date quality estimates of this path, and updates the pheromone values along the path from source to destination. A backward ant does the same for the direction from the destination back to the source. If on the other hand the ant got broadcasted at any point, it will leave the currently known pheromone trails, and explore new paths. After a broadcast the ant will arrive in all the neighbors of the broadcasting node. It is possible that in this neighbor it does not find pheromone pointing towards the destination, so that it will need to be broadcasted again. The ant will then quickly proliferate and good the network, like a reactive forward ant does

In order to avoid this, we limit the number of broadcasts to two. If the proactive ant does not find routing information within two hops, it will be deleted. The effect of this mechanism is that the search for new paths is concentrated around the current paths, so that we are looking for path improvements and variations.

In order to guide the forward ants a bit better, we use *hello messages*1: using these messages, nodes know about their immediate neighbors and have pheromone information about them in their routing table. So when an ant arrives in a neighbor of the destination, it can go straight to its goal. Looking back at the ant colony inspiration of our model, this can be seen as pheromone diffusion: pheromone deposited on the ground diffuses, and can be detected also by ants further away. In future work we will extend this concept, to give better guidance to the exploration by the proactive ants. Hello messages also serve an-other purpose: they allow to detect broken links.

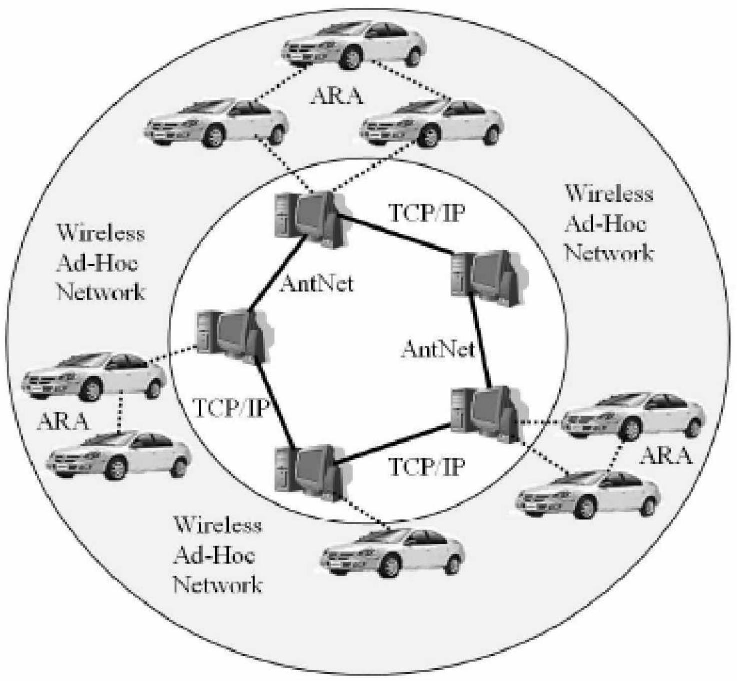


Fig. 2. Hybrid network

4.3FlowChart representation

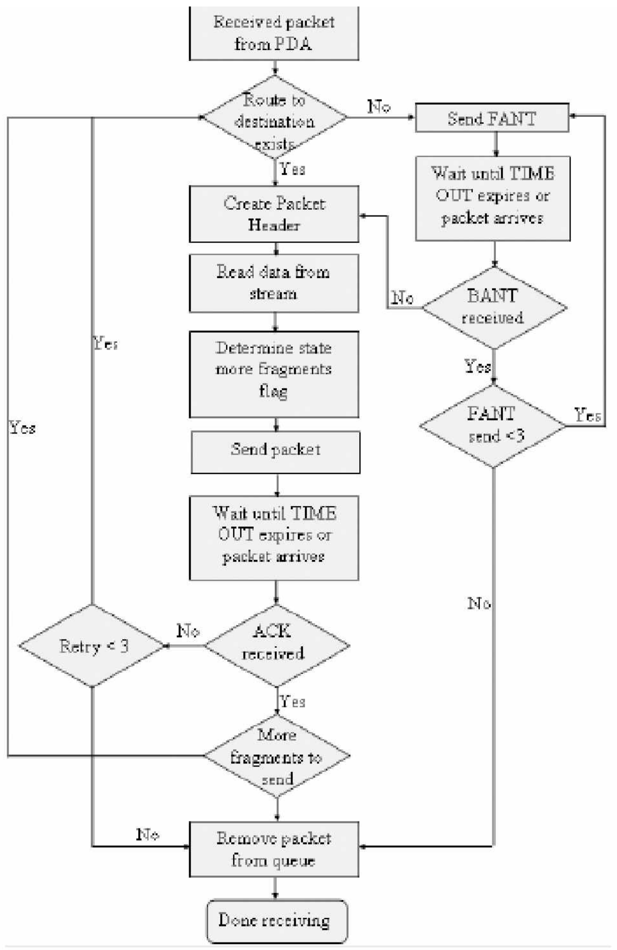


Fig. 6. Outgoing data packet

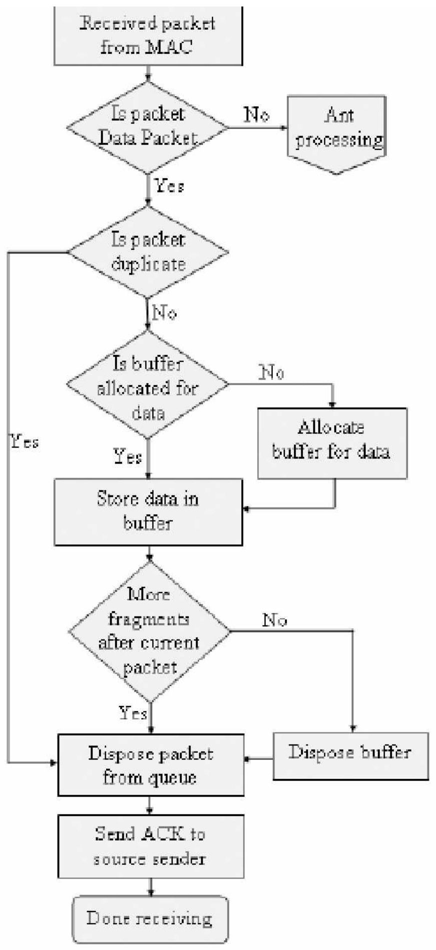


Fig. 7. Incoming data packet

Simulation Experiments

We evaluate our algorithm in a number of simulation tests. We compare its performance with AODV [(with route repair), a state-of-the-art MANET routing algorithm and a de facto standard.

4.4Simulation Environment

As simulation software, we use Qualnet, a commercial simulation package.30 We run simulation tests on three different scenarios. All three scenarios are run for 900 seconds. Nodes are initially placed randomly on the surface area and then they move according to the random waypoint mobility model (RWP),13 with speed ranging between 0 and 20 m/s (under RWP, nodes iteratively choose a random destination,move there in a straight line with a randomly chosen speed, and then wait there for a certain pause time). The pause time is set differently in different experiments. Data traffic is generated by 20 constant bit rate (CBR) sources using UDP at the transport layer. The sources start sending at a random time between 0 and 180 seconds after the start of the simulation, and keep sending until the end. At the physical layer a two-ray signal propagation model is used. At the MAC layer we use the popular 802.11b DCF protocol, with only one channel for communications.

In the first scenario, 100 nodes move in an area of 3000 x 1000 m2. Each data source sends one 64-byte packet per second. The data rate at the physical layer is 2Mbit/s and the radio range 300 meters. We ran tests for different pause times, ranging between 0 and 480 seconds (under RWP, higher pause time means higher mobility). In the second scenario, we investigate what happens under more intense

data traffic, using higher bandwidth: each data source sends eight 64-byte packets per second. The bandwidth of the wireless interface is raised to 11Mbit/s. Since higher bandwidth implies shorter radio ranges (now 110 meters), we reduce the area of the MANET: 100 nodes move in an area of 1000 x 1000 m2. We use the same pause times as before. In the third scenario, we investigate the scalability of the algorithm. We keep the same data rate and bandwidth as in the second scenario, and vary the number of nodes, ranging from 50 to 500. The MANET area is adapted accordingly, ranging from 750 x 750 m2 to 2250 x 2250 m2, to keep the node density constant on 1 node per 100 m2. The pause time is kept constant on 30 seconds.

4.5 Simulation Results

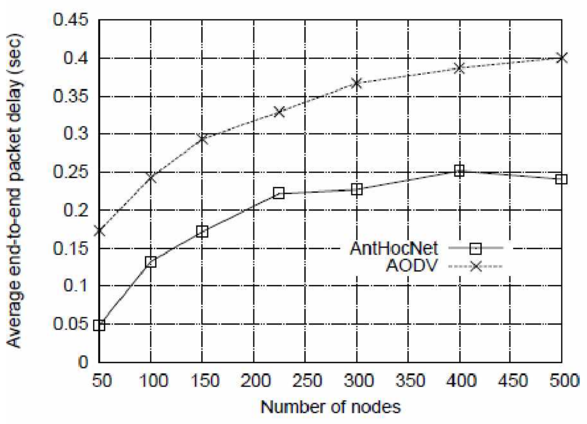
In these tests, we measure the average end-to-end delay for data packets and the ratio of correctly delivered versus sent packets. These are standard measures of effectiveness in MANETs. We also report delay jitter, the average difference in inter-arrival time between packets. This is an

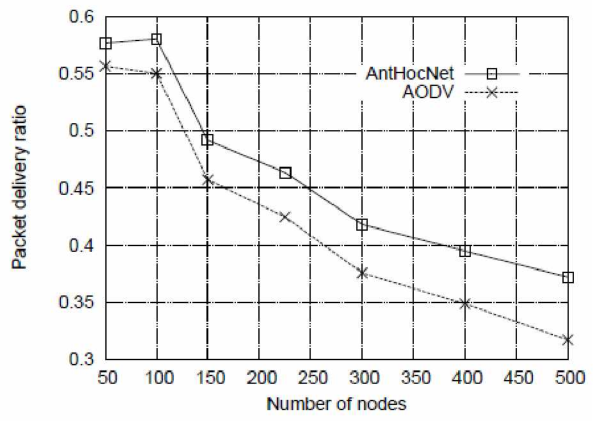
important metric in quality-of-service networks and also provides a measure of the stability of the algorithm's response to
topological changes in MANETs. Finally, we consider routing overhead, as measure of efficiency.

In the first scenario, with 100 nodes and light traffic load (figure 1 and tables 1a and 2a), AntHocNet shows an average end-to-end delay which is about half that of AODV for low pause times, and around one third for high pause times. In terms of delivery ratio, the difference is less striking but still significant. The drop in delivery ratio for the highest pause times is due to connectivity issues.

In the second scenario, with 100 nodes and heavier traffic load (figure 2 and tables 1b and 2b), we can see the same trends. The difference in delay is smaller for low pause times, but for high pause times AntHocNet's average delay is again three times lower than AODV's.

The third scenario (figure 3 and tables 1c and 2c) shows the scalability of AntHocNet: with increasing network size, the differences in terms of average end-to- end delay, delivery ratio and jitter grow. More importantly, while AntHocNet has a slightly higher overhead than AODV for the small scenarios, it actually generates less overhead for the larger scenarios.





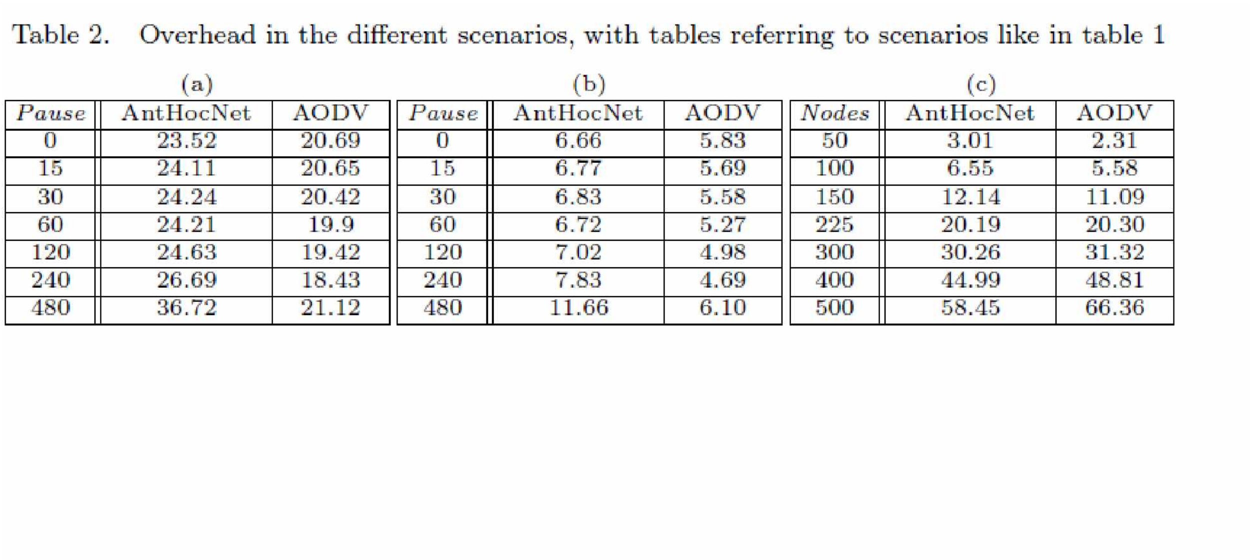


Table 2. Overhead in the different scenarios, with tables referring to scenarios like in table 1

(a)

Pause	AntHocNet	AODV
0	23.52	20.69
15	24.11	20.65
30	24.24	20.42
60	24.21	19.9
120	24.63	19.42
240	26.69	18.43
480	36.72	21.12

(b)

Pause	AntHocNet	AODV
0	6.66	5.83
15	6.77	5.69
30	6.83	5.58
60	6.72	5.27
120	7.02	4.98
240	7.83	4.69
480	11.66	6.10

(c)

Nodes	AntHocNet	AODV
50	3.01	2.31
100	6.55	5.58
150	12.14	11.09
225	20.19	20.30
300	30.26	31.32
400	44.99	48.81
500	58.45	66.36

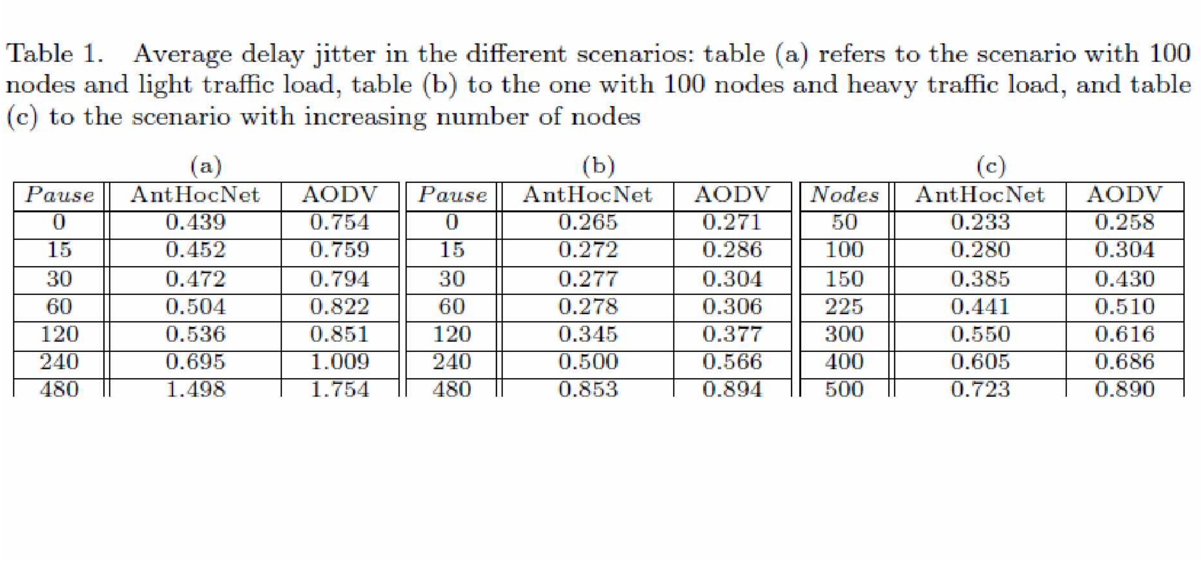


Table 1. Average delay jitter in the different scenarios: table (a) refers to the scenario with 100 nodes and light traffic load, table (b) to the one with 100 nodes and heavy traffic load, and table (c) to the scenario with increasing number of nodes

(a)

Pause	AntHocNet	AODV
0	0.439	0.754
15	0.452	0.759
30	0.472	0.794
60	0.504	0.822
120	0.536	0.851
240	0.695	1.009
480	1.498	1.754

(b)

Pause	AntHocNet	AODV
0	0.265	0.271
15	0.272	0.286
30	0.277	0.304
60	0.278	0.306
120	0.345	0.377
240	0.500	0.566
480	0.853	0.894

(c)

Nodes	AntHocNet	AODV
50	0.233	0.258
100	0.280	0.304
150	0.385	0.430
225	0.441	0.510
300	0.550	0.616
400	0.605	0.686
500	0.723	0.890

CHAPTER 5

RESULTS & DISCUSSIONS

5.1 POSITIVE RESULTS

The application does not contain any bug and the ants can evolved in any configuration. However, as the factors cannot be changed by the user, the ground configuration should always be limited. The number of nodes and the size of the playground are chosen by the user, but the number of ants, the speed at which they become tired or the evaporation of the traces always remain the same.

The obtained results are good when the ants are limited to simple ways, like in the wired or wireless networks: they can go only in two directions, except for the nodes where they need to decide which path to take.

The length of the paths is taken in consideration, but the number of nodes is also important.

The algorithms used give in most of the cases good results. The two-bridge experiment explained in the first part of the report is verified, and the ants will converge to the shortest path for more complicated ground.

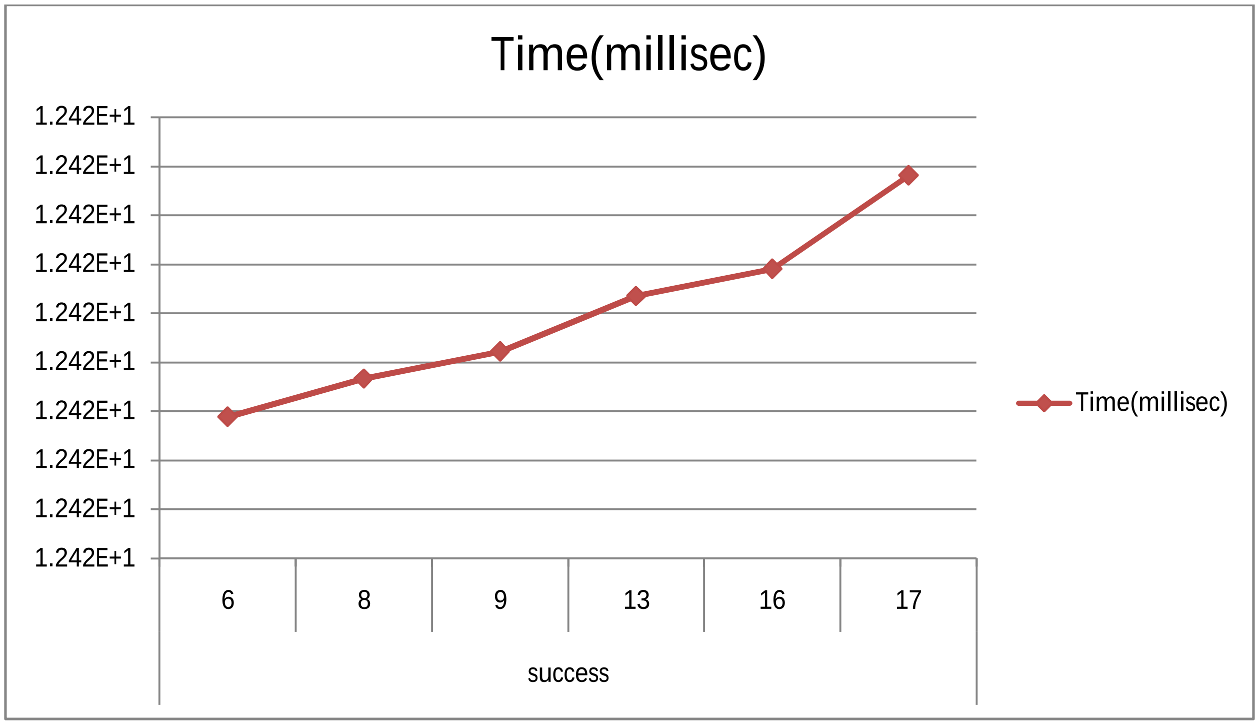


Fig.4.1 **Result**

5.2 Analysis of the state-of-the-art Adaptations of the Ants-based Algorithms to routing protocols :

Why and How to apply an Ant-Based algorithm to routing protocols

The networks become nowadays more and more complicated, with moving nodes, varying loads, etc. The users however expect more quality and more services despite the growing complexity of the networks. The theses which will be analysed in the following study some adaptations of the Ant Colony Optimization to routing protocols, and often compare its efficacy to the current routing algorithms.

Most of the papers see in the ACO a great tool for wireless Ad Hoc networks as it has a strong capacity to adapt to changes. However, some new algorithms based on ACO are also analysed for wired networks and are giving encouraging results.

The comparison between ACO and traditional routing algorithms is done with analysing:

§ The routing information;
§ The routing overhead;
§ The adaptivity.

Routing Information

The routing information consists of what is exchanged to get to know the architecture of the network, hence forward the data packets to the best path. For RIP, the nodes exchange the distance-vector information, each node giving to the other their neighbours and so on. In OSPF, the nodes tables need on the link-state information of all the links in every path to compute the shortest path.

In ACO, the paths from a source to a destination are explored independently and in parallel. The figure 4.2 shows a simple configuration of 6 nodes.

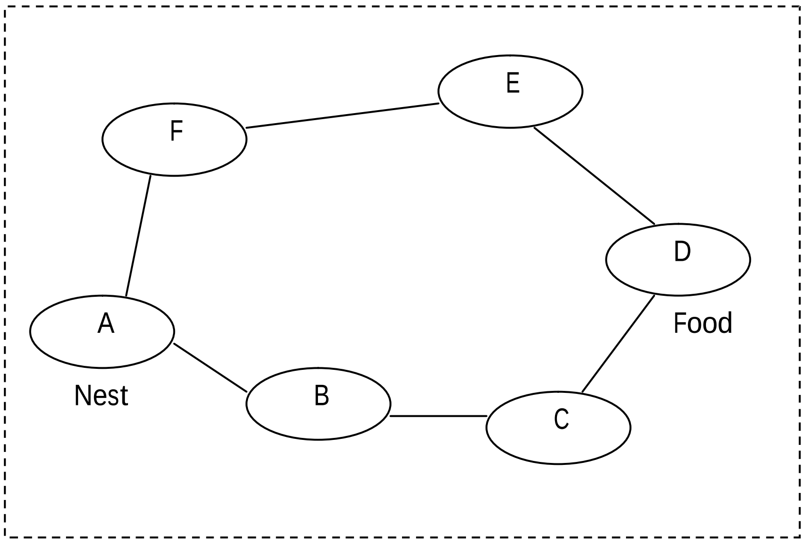


Fig4.2 Nodes

For RIP, the nest A depends on routing tables sent by B and F, as well as the Food depends on C and E’s routing tables.

In OSPF, A needs to know all the link-state between itself and the food to find the shortest path.

In ACO, the paths from the source to the food are explored by using n number of ants, the ants leaving the nest at the same time and taking a random first path. n/2 ants will go through B while the other half will take the way to F. The ants which reach the first the food indicates which way is the shortest without having to wait for the second half of ants to reach. As soon as an ant arrives at a node, the corresponding pheromones value for a path is updated. Hence, each entry of the pheromone table in a node can be updated independently.

In the figure 4.2 the Food point, node D, can immediately use the information in its pheromone table to route data packets to the nest when any ant from either path arrives (and updates its pheromone table).

Information overhead

Routing in RIP involves the transmission of routing tables of each node to every one of its neighbours. For a large network, the routing table of each node, which consists of a list of cost vectors to all other nodes, is large. Since each node needs to transmit its routing table to all of its neighbours, the routing overhead can be very large.

In OSPF, routing is achieved by having each node transmit a link-state packet (LSP) to every other node in a network through a *flooding* processing. Although an LSP, which carries information about the costs to all the neighbours of a node, is generally smaller than a routing table, the flooding process ensures that every node receives a copy of the LSP. Since an LSP from a node can be disseminated via different paths to other nodes, multiple identical copies of the same LSP may be transmitted to the same node.

Routing in ACO is achieved by transmitting ants rather than routing tables or by flooding LSPs. Even though it is noted that the size of an ant may vary in different systems/implementations, depending on their functions and applications, in general, the size of ants is relatively small, in the order of 6 bytes[ii]. This is because ants are generally very simple agents. The following table summarizes the differences between ACO and traditional routing algorithms.

Table 4.1

	RIP / OSPF	ACO algorithm
Routing preference	Based on transmission time / delay	Based on pheromones concentration
Exchange of routing information	Routing information and data packet transmitted separately	Can be attached to data packets
Adapting to topology change	Transmit routing table / Flood LSPs at regular interval	Frequent transmission of ants
Routing overhead	High	Low
Routing update	Update entire routing table	Update an entry in a pheromone table independently

Adaptivity

In dynamic networks, transmitting large routing table (in RIP) or flooding multiple copies of LSPs (in OSPF) in short or regular intervals may incur large routing overhead. However, flooding LSPs and transmitting routing table in longer intervals may result in slower responses to changes in network topology. Since ants are relatively small they can be piggybacked in data

packets, more frequent transmission of ants to provide updates of routing information may be possible. Hence, using ACO for routing in dynamic network seems to be appropriate.

Related to the issue of adaptivity is stagnation. Stagnation occurs when a network reaches its convergence; an optimal path ρ is chosen by all ants and this recursively increases an ant's preference for ρ. This may lead to: 1) congestion of ρ, 2) dramatic reduction of the probability of selecting other paths. The two are undesirable for a dynamic network since:

1) ρ may become nonoptimal if it is congested;

2) ρ may be disconnected due to network failure;

3) other nonoptimal paths may become optimal due to changes in network topology, and iv) new or better paths may be discovered

CHAPTER 6

CONCLUSION

CONCLUSION

This project tried to cover the state-of-the-art studies about Ant Colony Optimisation (ACO) algorithm and its application to routing protocols. It has been a great pleasure to study these papers.

At the beginning of this project I developed my own application which simulates the ACO, which gave me the best understanding of the algorithm and its issues.

Thus, the applications to the routing protocols are easier to understand since the main ideas behind them have always been inspired by the ants.

REFERENCES

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence:from natural to artificial intelligence*. Oxford University Press, 1999. ISBN 0-19-513158-4.

[2] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multihop wireless ad hoc network routing protocols. *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'98)*, pages 85– 97, 1998.

[3] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw- Hill, London, 1999.

[4] C. E. P. (Editor). *Ad Hoc Networking*. Addison-Wesley, 2001. ISBN 0-201-30976-9.

[5] K. Fall and K. Varadhan. *The ns Manual*, Nov 2000.

[6] D. B. Johnson, D. A. Maltz, Y.-C. Hu, and J. G. Jetcheva. The dynamic source routing protocol for mobile ad hoc networks. IETF Internet draft, draft-ietf-manet-dsr-04.txt, November 2000.

[7] J. P. Macker and M. S. Corson. Mobile ad hoc networking and the IETF. *Mobile Computing and Communications Review*, 2(1):9–14, 1998.

[8] C. E. Perkins and P. Bhagvat. Hihgly dynamic destinationsequenced distnace-vector routing (dsdv) for mobile computers.*Computer Communications Rev.*, pages 234–244, October 1994.

[9] C. E. Perkins, E. M. Royer, and S. R. Das. Ad hoc ondemand distance vector (aodv) routing. IETF Internet draft, draft-ietf-manet-aodv-07.txt, November 2000.

ANT COLONY OPTIMIZATION TECHNIQUE

Author: Sthitaprajna Jena and Jyoti prakash Lakra

Supervisor: S.K. Rath

Introduction: Swarm intelligence (SI) is a type of artificial intelligence based on the collective behavior of decentralized, self-organized systems. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems.[1].SI systems are typically made up of a population of simple agents or boids interacting locally with one another and with their environment. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents. Particle swarm optimization (PSO) is a swarm intelligence based algorithm to find a solution to an optimization problem in a search space, or model. The ant colony optimization algorithm (ACO), is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs.

Objectives:

§ Propose an easy approach to the Ant Colony Algorithm, with appropriated vocabulary and global explanation, as well as details about its behaviour.

§ Develop a Java application which shows the working of the algorithm and gives a better understanding.

§ Give a straightforward analysis of the state-of-the-art studies on Ants-based Routing Algorithms and the implementations which have been done.

Implementation:

The code works with 2 main classes: the ant and the playground, plus a very simple one, the traces. After defining these classes, we need to define their attributes and their methods. That means what they own and what they are capable of. An ant is an independent and autonomous object which is capable to move. First let's have a look to its attribute: each ant is having a position, and two booleans tired and goal. This way, we get to know if the ant is tired and need to come back to the nest, and also if it has reached the goal or not. The position is stored in 2 integers, posX and posY, as well as in an array of 2 integers position[]. The last position is also saved, as the ant even if not the cleverest creature, still remembers its last position and does not go back at every step. The playground plays the role of the manager in our application. It will be the one which manages the moves and the output in a graphical mode. We will not go into the details of the code and stick to the algorithm part.

Application :

problems. Ant colony optimization algorithms have been used to produce near-optimal solutions to the travelling salesman problem. The first ACO algorithm was called the Ant system [5] and it was aimed to solve the travelling salesman problem, in which the goal is to find the shortest round-trip to link a series of cities. The general algorithm is relatively simple and based on a set of ants, each making one of the possible round-trips along the cities. At each stage, the ant chooses to move from one city to another according to some rules:

1. It must visit each city exactly once;
2. A distant city has less chance of being chosen (the visibility);
3. The more intense the pheromone trail laid out on an edge between two cities, the greater the probability that that edge will be chosen;
4. Having completed its journey, the ant deposits more pheromones on all edges it traversed, if the journey is short;
5. After each iteration, trails of pheromones evaporate.

Results :

The application does not contain any bug and the ants can evolved in any configuration. However, as the factors cannot be changed by the user, the ground configuration should always be limited. The number of nodes and the size of the playground are chosen by the user, but the number of ants, the speed at which they become tired or the evaporation of the traces always remain the same.The obtained results are good when the ants are limited to simple ways, like in the wired or wireless networks: they can go only in two directions, except for the nodes where they need to decide which path to take.The length of the paths is taken in consideration, but the number of nodes is also important. The algorithms used give in most of the cases good results. The two-bridge experiment explained in the first part of the report is verified, and the ants will converge to the shortest path for more complicated ground.

Conclusion

This project tried to cover the state-of-the-art studies about Ant Colony Optimisation (ACO) algorithm and its application to routing protocols. It has been a great pleasure to study these papers. At the beginning of this project I developed my own application which simulates the ACO, which gave me the best understanding of the algorithm and its issues. Thus, the applications to the routing protocols are easier to understand since the main ideas behind them have always been inspired by the ants.

References:

§ [1] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence:from natural to artificial intelligence*. Oxford University Press, 1999. ISBN 0-19-513158-4.

§ [3] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.