# Solving the Shortest Path Problem By Genetic and Ant Colony Optimization Algorithms

**Eyman Yosef; Ahmed Salama; M. Elsayed Wahed**

Department of Mathematics, Faculty of Sience, Bour Said University; Bour Said, Egypt,
Department of Mathematics, Faculty of Sience, Bour Said University; Bour Said Egypt,
Faculty Of Computers and Informatics, Suez Canal University, Ismailia ,Egypt,
mewahed@yahoo.com

**Abstract:** In this paper I will present two different genetic and ant colony algorithms for solving a classic computer science problem: shortest path problems. I will first give a brief discussion on the general topics of the shortest path problem, genetic and ant colony algorithms. I will conclude by making some observations on the advantages and disadvantages of using genetic and ant colony algorithms to solve the shortest path problem and my opinion on the usefulness of the solutions and the future of this area of computer science
Keywords: genetic programming, ant colony algorithms, shortest path, optimization problems.

## 1 Introduction to Shortest Path

The shortest path problem (SPP) is a classic in the computer science community. It has been studied by many people but the current standard is Djikstra's shortest path algorithm, which utilizes dynamic programming to solve the problem.

Essentially what the shortest path problem deals with is if you have a graph G = (N,V); where N is a set of nodes or locations and V is a set of vertices that connect nodes in N where V is a subset of NxN. In SPP each vertex in V also has a weight associated with it and the problem that needs to be solved it how to get from any node in N to any other node in N with the lowest weight on the vertices used.

A simple example is to let N be all the airports serviced by an airline, and V is the flights for that airline. Additionally let the weight of V be the cost of each flight. The problem to be solved in this situation would be to find the cheapest way to get from any airport serviced to any other airport.

Djikstra developed an algorithm for this that can solve the problem that runs in O(n log n) time. However, it needs to be recalculated every time there is a change. The goal is to find an algorithm that can adapt to a changing graph topology in semi-real time.

## 2 Introduction to GAs

Genetic algorithms are a way to apply what we know about biology and how nature solves problems to the computer science realm. Either they can be used to solve problems where the possible solutions cannot be enumerated or it would be too costly to do so. Some examples of this would be optimal placement of multiple cameras on a map, or in this case the shortest path from point A to point B though nodes in a graph.

The problem is solved by representing each solution as a gene; each gene is made of one or more alleles; just like a DNA sequence. Two examples used in this section are optimal camera placement and a generic SPP. In the camera placement problem for example, each allele might contain a position, a cost to place there or some other information. In the SPP problem I will use as an example, each allele would be a node and there would be a cost associated with getting from the previous node in the gene to the current one; in this example the first allele in the gene would be the start, and the last allele would be the destination.

In some cases encoding can be a big part of designing the algorithm. We will see a couple examples on how to encode an SPP later in this paper.

The basic problem is broken down into four basic steps: initialization, selection, crossover, and mutation. In some cases, a fifth step of recovery is needed to fix any genes that may be invalid or unfeasible. I will discuss all four steps next.

**Initialization**. The first step is to initialize the first population (generation) that you will use as a base. Most times this is done randomly or with some small guidance, but we do not want too much time to be taken for initialization. For SPP most research has shown that random initialization, without errorious genes provide the best results.

**Selection.** In this part of the algorithm we will design a fitness function, this will return a numeric value for how “good” of a solution that particular gene is. This rating is used to see which of the genes of the generation will produce offspring and/or continue to the next generation.

Once you have a value for each gene, there are several ways to select parents for the next generation. Some examples of this are roulette wheel selection; where each gene is given a subsection of integers below a certain number then a random number is generated within that range and the gene that “owns” that range continues. Another is tournament selection, where two or more genes is compared agents each other randomly and the one with the better score is used. While others exist, I have seen these ones most often.

For some algorithms, some of the top “parents” selected are also continued on to the next generation unchanged. This is done so that if the optimal solution is already found it will not be destroyed.

**Crossover.** In crossover, two parents are taken in and two children are returned. This is similar to the DNA crossover procedure taken from biology.

The two parent genes are examined to find a suitable point for crossover. In the example of camera placement, assuming that each allele is a camera position, all pints would be suitable. In contrast, in the shortest path, assuming each node is an allele, only pints where the same node appears in both are suitable.

Once the suitable points are found one point on each gene is selected to cross the solutions over. Below is a simple graphic of how this would work:

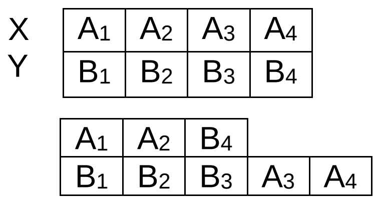


Figure 1 Crossover Example

In the figure above let X be the crossover point selected for parent A, and Y be the crossover point selected in parent B. The children produced contain some genetic information from both parents and these children will be used as part of the next generation.

In most algorithms it is okay for the genes to be of different lengths, but if that is not allowed X and Y would have to be at the same position in both genes before crossover.

**Mutation.** At the mutation phase, some alleles are randomly selected with a given percentage to change. Depending on the algorithm in question this may also change alleles around it. This similar to a genetic mutation in Biology where you may have a mistake in copying DNA and this results in some change in the cell produced, except here is on purpose and is enforced.

In most algorithms, the rate of mutation is very low, so this would happen depending on the size of the population, only a couple times. In the camera placement algorithm, they may just choose a new location for the camera and that would be the end of this stage. It gets more complex as the problem to be solved does the same.

Take for example the SPP algorithm explained in 2C, where each allele is a node, if you randomly change a node this may not be a path anymore, so you may want to instead choose a point on the path and randomly generate the end of the gene from that node on as in initialization. We will see in sections 3 and 4 how this problem is solved.

**Recovery.** This step is not always used, and in most cases is not presented in the explanation of what a GA is. Nevertheless, because so many of the SPP-GAs do have some sort of recovery function I have included it here.

What the recovery stage will do is find any malformed or incorrect genes in the population. For instance if an SPP gene creates a loop back on itself, we would want to identify it, or if it never reaches the destination node.

Once we have identified the problem genes, we can do basically one of two things: destroy it (this includes replacement), or fix it. If we destroy the gene, again we have a couple choices. We can create a new one randomly, create a new child from the last generation, or leave the population size one smaller than the previous populations and possibly recover later.

Some “broken” genes cannot be fixed and must be removed, while others can be, for instance we could remove the loop if it exists, of complete a path that does not reach the destination. These are the decisions that would need to be made when designing the algorithm and we will see some examples of how this is handled later.

## 3 Description of the Ahn-Ramakrishna (AR) Algorithm

This paper was presented in 2002 and has since been used as a resource for other papers and is no longer a real candidate to replace the current mainstream SPP algorithm. With that said I still feel this algorithm provides the basis for many others and it seems to be a good place to start examining how this problem would be solved in this realm.

**Encoding.** The encoding for this algorithm is simple, each allele is a node in the graph, so the edge used to get to any node in the gene (g) is the edge from g(i-1) to g(i); except for the first allele, because this is the source. This marked the fitness function for the gene quite simple. Because they are working with a network, the goal is to try to minimize the latency from source to destination. The fitness function is shown below:

$$f_i = \frac{1}{\sum_{j=1}^{l_i - 1} C_{g_i(j)g_i(j+1)}}$$

**Figure 2 AR calculation of fittness [2]**

where $C_{ij}$ gives the latency between nodes i and j in the directed graph.

**Initialization**. This algorithm uses random initialization to create the first generation. Because purely random generation is not feasible for SPP the algorithm attempts to be as random as possible. They start reach gene by adding the source node. Then they randomly choose a node that has an edge from the source. Then they repeat the process from that node and so on, making sure not to add a node twice. If they get to a point where they cannot add a node without a repeat, they backtrack until there is a new node they can add that is not one they have tired previously. They continue this until the destination is found.

**Selection.** Once the initial population is created, they utilize a “pairwise tournament selection without replacement”[2]. They select two genes at random, then the fittest of the two is selected as a parent. It is not put back in the pool for selection, so it cannot be a parent twice in one generation. Once the desired number of parents is selected, they go to the crossover method.

**Crossover.** The crossover method employed is very simple. They find all the nodes that exist in both parent genes, then they randomly pick one of these sets to be the crossover point. Then the sections are reversed and two new children are born. Because they would only switch at a point where both genes have the same node there should still be a path from source to destination. However, in this process they could create a loop, this will be resolved in the repair function that runs after mutation.

**Mutation.** Their mutation function utilizes some of the properties of the initialization function. If a gene is selected for mutation, then an allele is picked at random to be the mutation point. At this mutation point, it essentially follows the process of initialization where it randomly picks edges until it reaches the destination, but in this case, it will not regress back past the initial point of mutation.

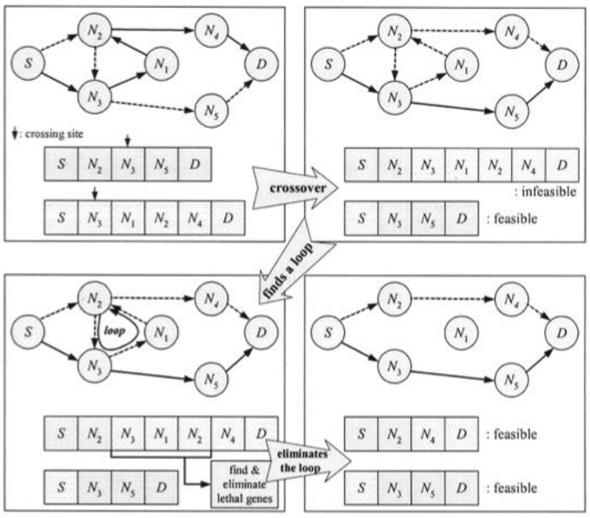


**Figure 3 Diagram of AR algorithm**

**Repair Function.** They use a very simple repair function. It traces the route that the gene encodes, if it encounters the same node twice it cuts out the path that created the loop. This can be seen in Figure 2. You can see in the third box a loop that had been created in the crossover is identified and then removed in the fourth box.

From the state at the end of Figure 2, the algorithm would use the newly created generation to produce more offspring and continue the cycle until an optimal solution is found.

## 4 Description of the Li-He-Guo (LHG) Algorithm

This paper is presented because it uses several major differences from the AR algorithm and provides contrast even within the same field of study.

**Encoding.** The encoding for this algorithm is different from the AR algorithm in that it does not actually encode the path. The gene contains a waiting system for each node in the graph, thus each gene is also the same size. The genes are specifically each of size |N|. This proves to make thing mush easier in later stages of the GA.[3] The way it works is that each node has a weight relative to each other node, no two nodes should have the same weight and this is resolved in the repair function.

To get the path from this priority weight encoded gene you start from the source and then choose the node with an edge from the source that has the highest priority. Then you repeat until you reach the destination without causing a loop. If a node is encountered that has no edge from which you would not create a loop, we would create a virtual link from that node to the destination, and give it a severe penalty (high latency) so that the GA will overcome the problem and evolve past it. The whole process is described in detail in section IV of [4].

**Initialization.** Initialization for the given encoding is quite simple. They assign an importance value to each node in the graph, which essentially is a percentage of the edges that don't come to

5

or from that node. The formula is given below. This value is then used along with a random number (between 0 and 1), and a large integer constant to produce the value of that node in the current gene.

$$\forall i \in V, W(i) = \left[ 1 - \frac{\sum_{j} c_{i,j}}{\sum_{i} \sum_{j} c_{i,j}} \right], (i, j) \in E$$

**Figure 4 LHG calculation of importance**

**Crossover.** Simply select a number of parents from the current generation then randomly generate pairings of them for crossover. Then a random section of each parent (of identical size and position) is exchanged.

**Mutation.** A select number of chromosomes are selected for mutation. Once a chromosome selected, then a gene is selected given its gene weight, some probability, and a random number.

**Fitness & Selection.** Here you calculate the fitness of each chromosome, this is done by calculating the latency of the path, and then the fitness is the inverse of that. Select some of the chromosomes that have a better fitness compared to the others in the generation. The total number of selected chromosomes should be between .6*size & .9*size.

## 5 Analysis of the Algorithms

While the AR algorithm gets close to, 100% accuracy on small networks we can see as the size of the network increases their convergence percentage begins to fall. This is then counteracted with the increase of the population size. On the other hand, the LHG algorithm does not resent any results on any networks as small as those presented in the AC algorithm, they also tended to get higher convergence rates.

While the two algorithms presented here do not present results compared to each other, we can see that from the results presented they both work well, but the AC algorithm seems to do better on small networks, while the LHG algorithm does better on large networks.

## 6 Observations

In looking at many of genetic algorithms for solving the shortest path problem, I have seen that they could be a reasonable solution for use on an Ad-Hoc network but the results are still on the same level as Djikstra. No paper I have read really presents anything that can far surpass the current standard in time complexity and results.

As an overall observation of Genetic algorithms, they are a very interesting class of solutions, but I am still not convinced that they can be used efficiently to solve any problems that are not already completed in realistic time. It seems like in most cases it makes the problem more complex than needed. I also feel that more research could be done in this area to possibly used GAs to solve some unsolved issued in computer science that people have put on the back burner while concentrating on the more “glamorous” topics. While there is some research and development being done in genetic algorithms, i think as the sciences all converge together, there is so much more I feel we can learn from their disciplines and bring to the CS community.

## 7. The Proposed Genetic Algorithm (GA)

In the proposed GA, each candidate path is represented by a binary string with length N that can be used as a chromosome. Each element of the chromosome represents a node in the network topology. So, for a network of N nodes, there are N string components in each candidate solution x. Each chromosome must contain at least two none zero elements.

For example if N = 8, the path of Figure 5 is represented as a chromosome as shown in Figure 6. Figure
In the following subsections we give an explanation of different components (operations) of the presented genetic algorithm.

### 7.1. Initial Population

The generated chromosome in initial population must contain at least two none zero elements to be a real candidate path.  The following steps show how to generate pop_size chromosomes of the initial population:

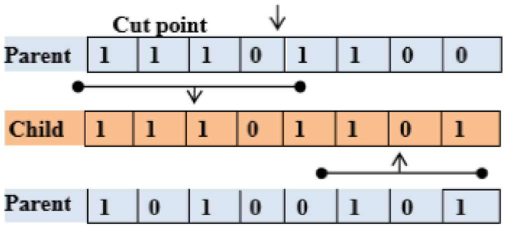
1. Randomly generate a chromosome x. 2. Check if x represents a real candidate path, i.e. contains at least two non zero elements. 3. Repeat steps 1 to 2 to generate pop_size chromosomes.

### 7.2. The objective function

The cost of the candidate path is used as objective function to compare the solutions and find the best one. The cost of the candidate path is calculated when it satisfies the following conditions:

 The chromosome must contain at least two none zero elements.  The chromosome contains a connected candidate path. I.e. each node in the path connects at least one another.
### 7.3. Genetic Crossover Operation

In the proposed GA, we use the single cut point crossover to breed a new offspring from two parents. The crossover operation will be performed if the crossover ratio (Pc=0.90) is verified. The cut point is randomly selected. Figure 7 shows the crossover operation.



7

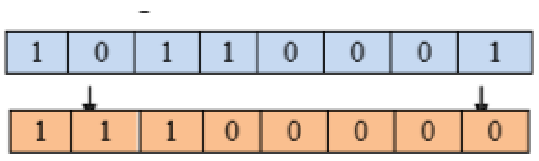


Figure 5. Example of the crossover operation.

7.4. Genetic Mutation Operation

The mutation operation is performed on bit-by-bit basis. In the proposed approach, the mutation operation will be performed if the mutation ratio (Pm) is verified. The Pm in this approach is chosen experimentally to be 0.02. The point to be mutated is selected randomly. The offspring generated by mutation is shown in Figure 8.

Figure 8. An example of the mutation operation.

7.5. The Entire Algorithm

The following pseudocode illustrates the use of our different components of the GA algorithm to generate the minimum-cost paths tree of a given network.

| Algorithm Find minimum-cost paths tree | |
|---|---|
| Input | Set the parameters: pop_size, max_gen, Pm, Pc. |
| Output | Minimum-cost paths tree |
| 1<br>2<br>3<br>4<br>5<br>6<br>7<br>7.1<br>7.2<br>7.3<br>8<br>9<br>10<br>11<br>12<br>13<br><br>14 | Set j = 2, the destination node.<br>Generate the initial population according to the steps in Section 0.<br>gen←1.<br>While (gen < = max_gen) do {<br>P ← 1<br>While (P <= pop_size) do {<br>Apply Genetic operations to obtain new population<br>Apply crossover according to Pc parameter (Pc >=0.90)<br>Apply Mutation as shown in section 4.4.<br>Compute the total cost of the candidate path<br>P ← P+1.<br>}<br>Set gen =gen + 1<br>if gen > max_gen then stop<br>}<br>Save the candidate path for the destination j that has the minimum cost (the shortest path between the root node and the destination node j).<br>Set j = j + 1 15.<br>If j <= N Goto Step 2,<br>otherwise stop the entire algorithm and print out the minimum-cost paths tree. |

## 8 Ant Colony Optimization

As mentioned earlier, the shortest path is expected to be estimated using ant colony algorithm. The coding capable nodes can be detected by examining a set of links. Let the cost path between two nodes (i, j) denoted as the amount of pheromone spread it on that link l. The pheromone information is changed randomly during problem solving progress to reflect the experience gained by ants. Hence, ants randomly choose paths based on the amount of pheromone. The size of deposited pheromone on a path is proportionally equal to the quality of that path. The quality here means the length of that path, more amount of pheromone are deposited on the shortest paths, L, which became the candidate paths for Ant Colony optimization solution.

Lets define, $T_ij^k(t)$ as size of pheromone in the link (L) connects between node i and node j. At each iteration of the Ant Colony algorithm, an ant goes from a joint node, i.e., intermediate node, i to node j. This reflects intermediate node accomplished intermediate solution. Furthermore, in each iteration, an ant (m) calculates a set L of feasible paths, and moves to one of these with a probability of moving $p_{ij}^k$ .

The probability of moving $p_{ij}^k$ depends on two factors: the attractiveness (heuristic value) $n_{ij}$ and trail level $T_{ij}$. The former factor can be calculated using some heuristic methods that are pointing the a priori desirability of that move, while the latter factor reflects the history of making this move in the past. In other words, it acts as a posterior indicator of the desirability of that move. Trails are regularly updated when all ants have a completed list of their solution.

It also indexed as increasing or decreasing to represents the high and bad quality solutions, repressively. The probability $k_{th}$ ant travels from node i to node j at time t, denoted as P, is given by [17]:

$$\rho_{ij} = \begin{cases} \dfrac{T_{ij}^{\alpha}(t) n_{ij}^{\beta}}{\sum_{j \in N_i^k} T_{ij}^{\alpha} n_{ij}^{\beta}(t)} & j \in N_i^k \\ 0 & O/W \end{cases}$$

Where $N_i^k$ is a set of non-visited nodes of ant m. α, 0≤ α ≤ 1, and β are two positive parameters, which can control the influence of relative weight of pheromone trail , and the influence of heuristic value $n_{ij}$. $n_{ij}^{\beta}$ defined as the historical value, calculated by [10]:

$$n_{ij}^{\beta} = \frac{1}{d_{ij}}$$

Where $d_{ij}$ is refereed of the distance that is used to reduce the probability to choice a long path. An ant stops going toward a path when it reach a wrong destination $N_i^k$ is the set of all candidate the neighbour nodes of node i and $n_{ij}$ is heuristic information. Once all ants complete one iteration, the amount of the pheromone of all the links can be given by following[10]:

$$T_{\mathrm{ij}}(t) = (1-\rho) * T_{\mathrm{ij}}(t-1) + \rho \sum_{k=1}^{m} \Delta T_{ij}^{k}(t-1)$$

Where $T_{\mathrm{ij}}(t)$ is the pheromone on the edge (i, j) after modernizing $T_{\mathrm{ij}}(t-1)$ is the pheromone before modernizing, k the number of route in the solution. ρ ∈ {0,1} is the evaporation rate of the pheromone while $\Delta T_{ij}^{k}$ is the size of pheromone spread which can be calculated by [10]:

$$\Delta T_{ij}^{k} = \begin{cases} \dfrac{1}{L^{k}} & if \text{ ant k use curve (i, j)} \in \text{iteration} \\ 0 & O/W \end{cases}$$

Where $L^{k}$ is the length of iteration that generated by ant kth route. The $E^{lk}$ here plays an important role of measuring the quality of each ant's solution.

## 8.1 Description of the algorithm:

**Step 1:** • Paths are implicitly defined by the values of pheromone variables contained in so-called pheromone table playing the role of routing tables.
• Defines the pheromone values in terms of the metrics (minimum delay and average of bandwidth of links).
• The algorithm tries to find paths characterized by minimal delay between S and D, minimal number of hops and maximum average bandwidth links between nodes.
• The use of such composite pheromone values allows to optimize multiple objectives simultaneously, which can be very important in complex networks (Pheromone metrics used to define path quality).

**Step 2:** • From each network node, s agents are launched towards specific destination nodes d at regular intervals and concurrently with the data traffic.
• These agents moving from their source to destination nodes are called forward ants.
• Each forward ant is a random experiment aimed at collecting and gathering at the nodes non-local information about paths and traffic patterns.
**Step 3:** • The specific task of each forward ant is to search for a minimum delay path and maximum average bandwidth link connecting its source and destination nodes.

**Step 4:** • While moving, the forward ant collects information about the traveling time. Once arrived at destination, the forward ant becomes a backward ant and goes back to its source node by moving along the same path.

**Step 5:** • At each visited node, the backward ant updates the local routing information related to each node in the path followed by the forward ant to d, and related to the choice of as next hop to reach each node.

**Step 6:** • When a source node s starts communication session with a destination node d, and no pheromone information is available about how to reach d, the node manager needs to gather long range information about possible paths.
• Therefore, it broadcasts a reactive forward ant.

**Step 7:** • Each forward ant keeps a list of the nodes it has visited.
• Upon arrival at the destination d, it is converted into a backward ant, which travels back to the source retracing the path.
• At each intermediate node, coming from neighbor, the ant information is used to update the entry in the pheromone table.
• The way the entry is updated depends on the minimum delay and average of bandwidth links between nodes used to define pheromone variables.

**Step 8:** • During the course of a communication session, managers at source nodes periodically send out proactive forward ants to update the information about currently used paths and to try to find new and potentially better paths.
• They follow pheromone and update pheromone tables in the same way as reactive forward ants do.

**Step 9:** • The path setup phase together with the proactive path improvement actions create a mesh of multiple paths between source and destination.
• Data are forwarded according to a stochastic policy depending onto the pheromone values (minimum delay path average bandwidth link).
• The forward ant migrates from a node to an adjacent one towards its destination.
• At each intermediate node, a stochastic decision policy is applied to select the next node to move to.

## 9 Numerical Examples:

Solving by using Anti colony

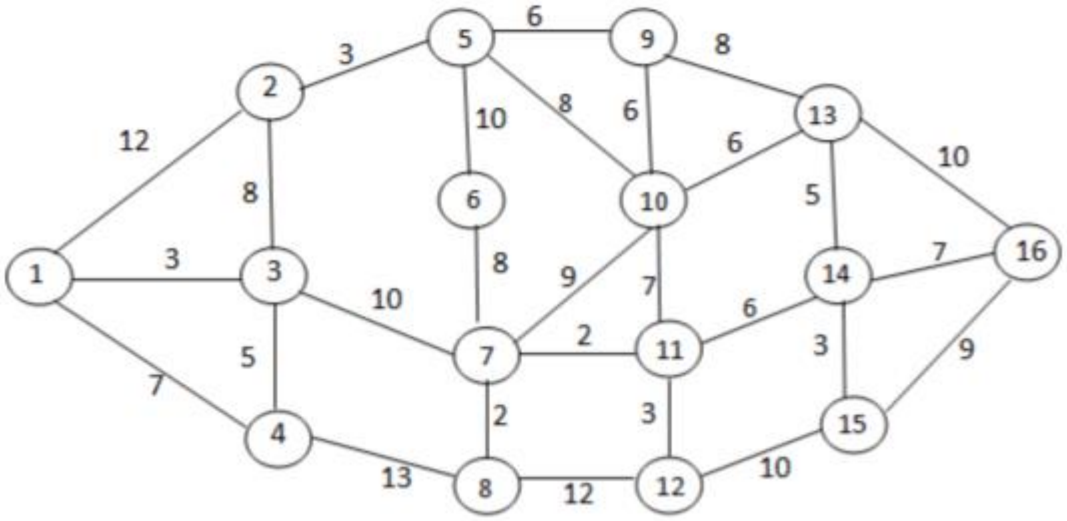
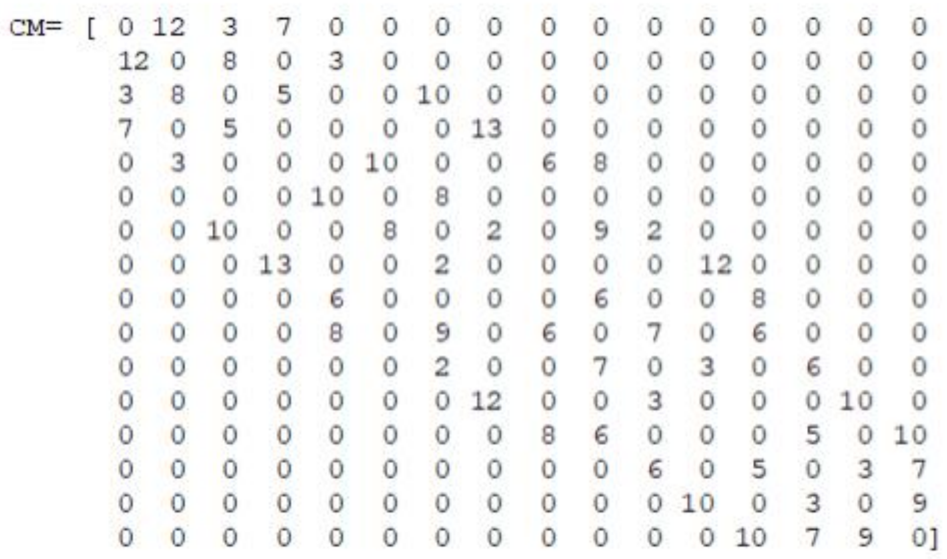


Figure 13: Sixteen nodes network

$$
M=\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0
\end{bmatrix}
$$

Connection matrix of the above network

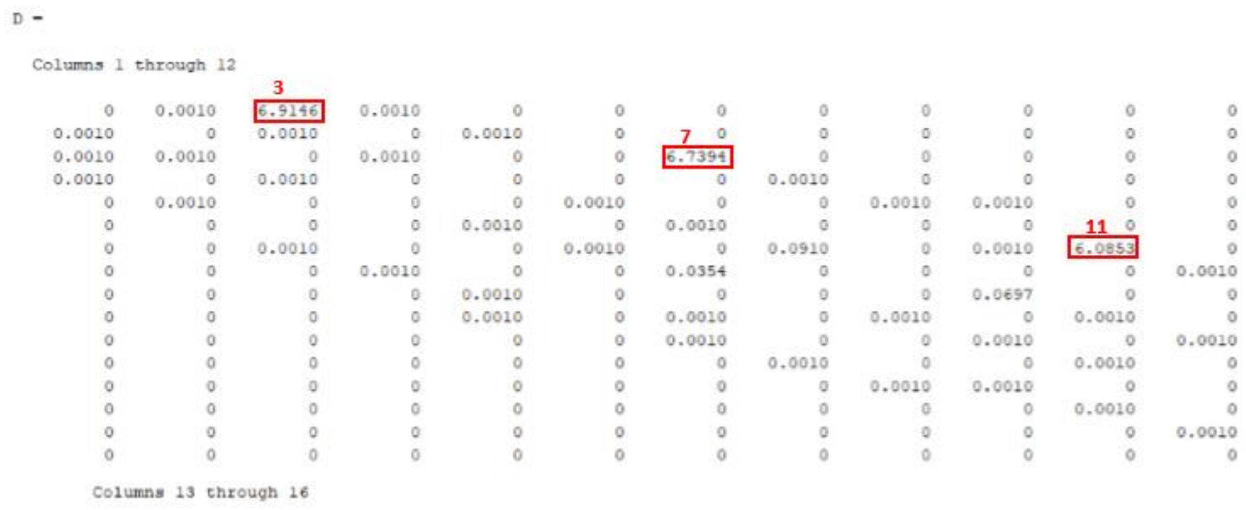
12



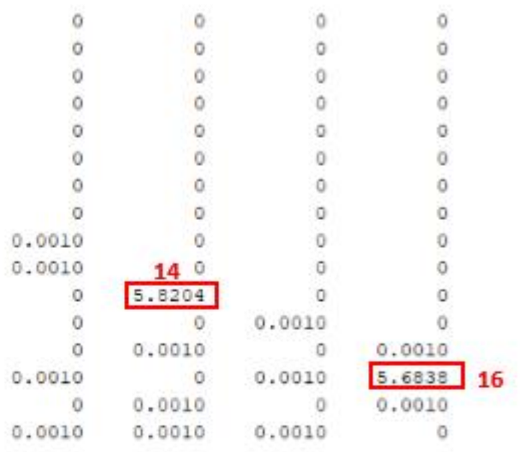
```
CM= [ 0 12  3  7  0  0  0  0  0  0  0  0  0  0  0  0
     12  0  8  0  3  0  0  0  0  0  0  0  0  0  0  0
      3  8  0  5  0  0 10  0  0  0  0  0  0  0  0  0
      7  0  5  0  0  0  0 13  0  0  0  0  0  0  0  0
      0  3  0  0  0 10  0  0  6  8  0  0  0  0  0  0
      0  0  0  0 10  0  8  0  0  0  0  0  0  0  0  0
      0  0 10  0  0  8  0  2  0  9  2  0  0  0  0  0
      0  0  0 13  0  0  2  0  0  0  0 12  0  0  0  0
      0  0  0  0  6  0  0  0  0  6  0  0  8  0  0  0
      0  0  0  0  8  0  9  0  6  0  7  0  6  0  0  0
      0  0  0  0  0  0  2  0  0  7  0  3  0  6  0  0
      0  0  0  0  0  0  0 12  0  0  3  0  0  0 10  0
      0  0  0  0  0  0  0  0  8  6  0  0  0  5  0 10
      0  0  0  0  0  0  0  0  0  0  6  0  5  0  3  7
      0  0  0  0  0  0  0  0  0  0  0 10  0  3  0  9
      0  0  0  0  0  0  0  0  0  0  0  0 10  7  9  0]
```

Cost matrix of the above network

The output from matlab code

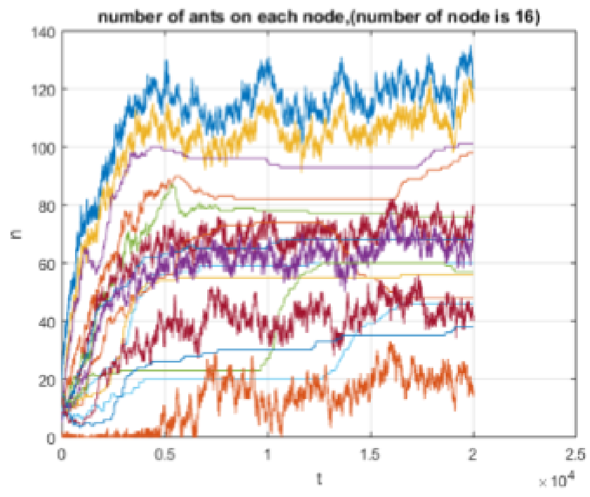


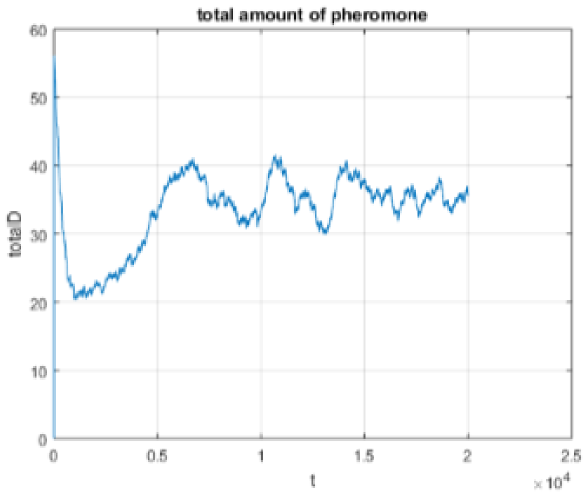
```
D =

  Columns 1 through 12

                              3
         0    0.0010    6.9146    0.0010         0         0         0         0         0         0         0         0
    0.0010         0    0.0010         0    0.0010         0   7     0         0         0         0         0         0
    0.0010    0.0010         0    0.0010         0         0    6.7394         0         0         0         0         0
    0.0010         0    0.0010         0         0         0         0    0.0010         0         0         0         0
         0    0.0010         0         0         0    0.0010         0         0    0.0010    0.0010         0         0
         0         0         0         0    0.0010         0    0.0010         0         0         0  11     0         0
         0         0    0.0010         0         0    0.0010         0    0.0910         0    0.0010    6.0853         0
         0         0         0    0.0010         0         0    0.0354         0         0         0         0    0.0010
         0         0         0         0    0.0010         0         0         0         0    0.0697         0         0
         0         0         0         0    0.0010         0    0.0010         0    0.0010         0    0.0010         0
         0         0         0         0         0         0    0.0010         0         0    0.0010         0    0.0010
         0         0         0         0         0         0         0    0.0010         0         0    0.0010         0
         0         0         0         0         0         0         0         0    0.0010    0.0010         0         0
         0         0         0         0         0         0         0         0         0         0    0.0010         0
         0         0         0         0         0         0         0         0         0         0         0    0.0010
         0         0         0         0         0         0         0         0         0         0         0         0

  Columns 13 through 16
```



```
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
        0         0         0         0
   0.0010         0         0         0
   0.0010      14 0         0         0
        0    5.8204         0         0
        0         0    0.0010         0
        0    0.0010         0    0.0010
   0.0010         0    0.0010    5.6838  16
        0    0.0010         0    0.0010
   0.0010    0.0010    0.0010         0
```

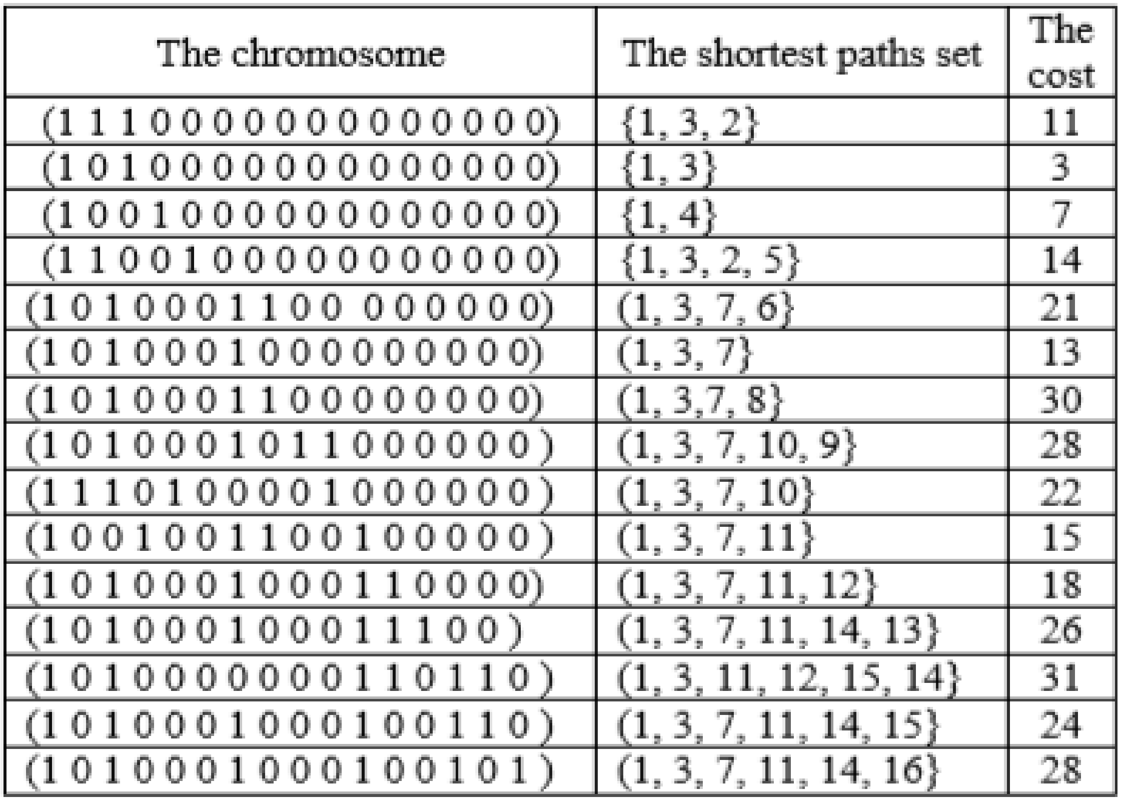
In the outcome matrix of conductivity D, we find that elements $D_{1,3}$, $D_{3,7}$, $D_{7,11}$,$D_{11,14}$,$D_{14,16}$ reinforce the pheromone. So, the shortest path is 1 - 3 - 7 - 11 - 14 - 16. The total cost of the shortest path is 28.



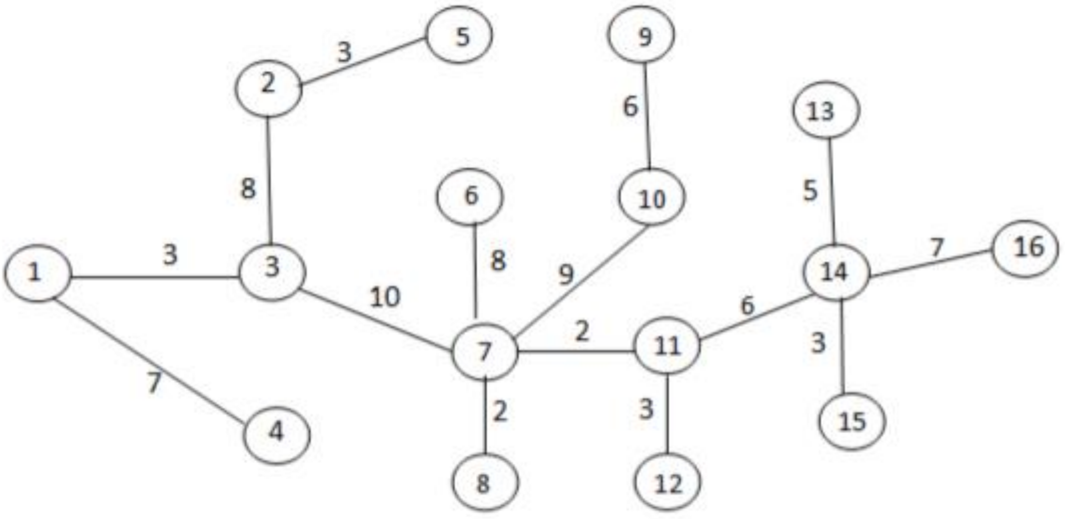


Solving by using Genetic Programming

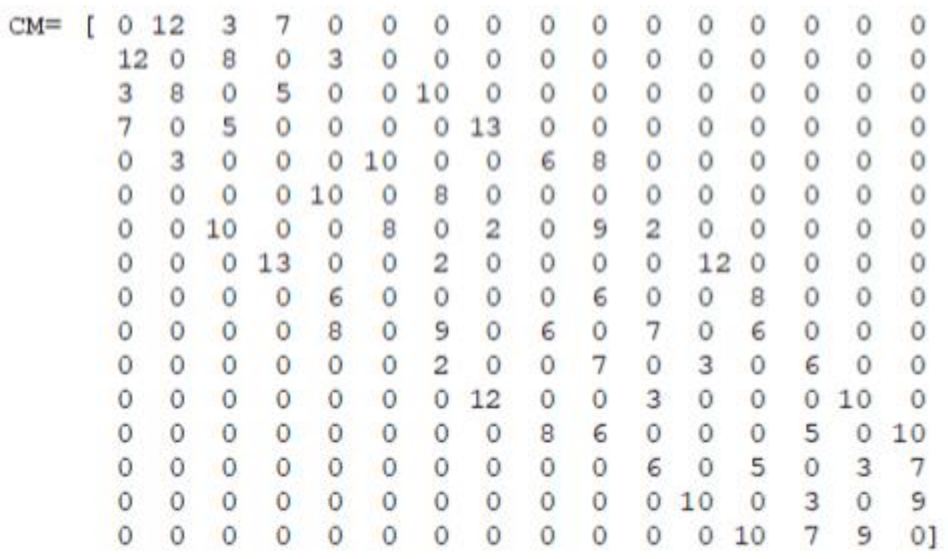
14



| The chromosome | The shortest paths set | The cost |
|---|---|---|
| (1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0) | {1, 3, 2} | 11 |
| (1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0) | {1, 3} | 3 |
| (1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0) | {1, 4} | 7 |
| (1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0) | {1, 3, 2, 5} | 14 |
| (1 0 1 0 0 0 1 1 0 0  0 0 0 0 0 0) | (1, 3, 7, 6} | 21 |
| (1 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0) | (1, 3, 7} | 13 |
| (1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0) | (1, 3,7, 8} | 30 |
| (1 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 ) | (1, 3, 7, 10, 9} | 28 |
| (1 1 1 0 1 0 0 0 0 1 0 0 0 0 0 0 ) | (1, 3, 7, 10} | 22 |
| (1 0 0 1 0 0 1 1 0 0 1 0 0 0 0 0 ) | (1, 3, 7, 11} | 15 |
| (1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 0) | (1, 3, 7, 11, 12} | 18 |
| (1 0 1 0 0 0 1 0 0 0 1 1 1 0 0 ) | (1, 3, 7, 11, 14, 13} | 26 |
| (1 0 1 0 0 0 0 0 0 0 1 1 0 1 1 0 ) | (1, 3, 11, 12, 15, 14} | 31 |
| (1 0 1 0 0 0 1 0 0 0 1 0 0 1 1 0 ) | (1, 3, 7, 11, 14, 15} | 24 |
| (1 0 1 0 0 0 1 0 0 0 1 0 0 1 0 1 ) | (1, 3, 7, 11, 14, 16} | 28 |



The minimum-cost paths tree rooted at node 1



```
CM= [ 0 12  3  7  0  0  0  0  0  0  0  0  0  0  0  0
     12  0  8  0  3  0  0  0  0  0  0  0  0  0  0  0
      3  8  0  5  0  0 10  0  0  0  0  0  0  0  0  0
      7  0  5  0  0  0  0 13  0  0  0  0  0  0  0  0
      0  3  0  0  0 10  0  0  6  8  0  0  0  0  0  0
      0  0  0  0 10  0  8  0  0  0  0  0  0  0  0  0
      0  0 10  0  0  8  0  2  0  9  2  0  0  0  0  0
      0  0  0 13  0  0  2  0  0  0  0 12  0  0  0  0
      0  0  0  0  6  0  0  0  0  6  0  0  8  0  0  0
      0  0  0  0  8  0  9  0  6  0  7  0  6  0  0  0
      0  0  0  0  0  0  2  0  0  7  0  3  0  6  0  0
      0  0  0  0  0  0  0 12  0  0  3  0  0  0 10  0
      0  0  0  0  0  0  0  0  8  6  0  0  0  5  0 10
      0  0  0  0  0  0  0  0  0  0  6  0  5  0  3  7
      0  0  0  0  0  0  0  0  0  0  0 10  0  3  0  9
      0  0  0  0  0  0  0  0  0  0  0  0 10  7  9  0]
```

Cost matrix of the above network

Comparison between the previous two methods

| | The used Optimization Algorithm | Objective | Purpose |
|---|---|---|---|
| First Method | Ant Colony Optimization (ACO) | Single-objective | Find the shortest path between source node s and destination node d which minimize cost |
| Second Method | Genetic Algorithm (GA) | Single-objective | Find shortest paths tree (shortest path between node s and every node in network) |

## Conclusion:

The problem of finding a shorter path in the network has been resolved in many ways by our goals. In this summary, we presented two different ways to solve this problem. The first is using Ant Colony Optimization (ACO) where the routing protocol was searched with network encryption and the proposed network performance was displayed in terms of packet delay, throughput consumption, and bandwidth. The second method is based on GA and uses crossover operators and mututation making it more complex and not always guaranteed giving the right results. Also, the crossover and mutation operators is done on random basis which may lead loose the best chromosome in the new generations. There is need from adding selection operator in the algorithm. Selection operator will help in determining which solutions are to be preserved and allowed to reproduce and which ones deserve to die out. Also, it will help in focusing research in promising areas of the search space.

## References

[1] Dr. A. Wu, “Genetic Algorithms”. *Lecture to COP 4810.0001*. 29 January 2007.

[2] C.W. Ahn and R.S. Ramakrishna, “A Genetic Algorithm for Shortest Path Routing Problem and the Sizing of Populations”. *IEEE Trans. on Evolutionary Computation*, Vol. 6, No 6, pp. 566-579 December 2002.
[3] Y. Li, R. He, Y. Guo, “Faster Genetic Algorithm for Network Paths”. *The Sixth International Symposium on Operations Research and Its Applications.* Pp. 380-389, August 2006.
[4] M. Gen, R. Cheng, D. Wang, “Genetic Algorithms for Solving Shortest Path Problems”. *IEEE*. Pp. 401-406, 2007
[5] M. Gen, L. Lin, “A New Approach for Shortest Path Routing Problem by Random Key-based GA”. *GECCO’06.* Pp. 1411-1412, July 2006.
[6] R.Geetha, and G. Srikanth, "Ant Colony optimization based Routing in various Networking Domains - A Survey ", International Research Journal of Mobile and Wireless Communications, ICICES-2012-SAEC, Chennai, Tamilnadu , Vol 03, Issue 01; January-April 2012.
[7] Dalia Sabri," Performance Analysis for Network Coding Using Ant Colony Routing", Master of Philosophy , Electronic and Computer Engineering, School of Engineering and Design ,Brunel University ,December 2011.
[8] M. Dorigo, V. Maniezzo & A. Colorni. "Ant System: Optimization by a Colony of Cooperating Agents ", IEEE Transactions on Systems, Man, and Cybernetics - Part B, 26 : 29-41.1996
[9] M. Dorigo, E. Bonabeau, and G. Theraulaz, “Ant algorithms and stigmergy “ , Future Gener Comput. Syst., vol. 16, no. 8, pp. 851-871, 2000.
[10]- Xiao-Min Hu and Jun Zhang," Minimum Cost Multicast Routing Using Ant Colony Optimization Algorithm" Hindawi Publishing Corporation, Mathematical Problems in Engineering, Volume 2013, Article ID 432686, 13 pages, 18 April 2013.
[11]- Gianni A. Di Caro, F. Ducatelle and Luca M. Gambardella," Theory and practice of Ant Colony Optimization for routing in dynamic telecommunications networks" the Complex Coevolution of Information Technology Ecosystems, Idea Group, Hershey, PA, USA, 2008.