COLLISION FREE PATH PLANNING ALGORITHMS FOR ROBOT NAVIGATION PROBLEM

A Thesis presented to the faculty of the Graduate School
University of Missouri-Columbia

In Partial Fulfillment
of the Requirement for the Degree
Master of Science

By

Kyung min Han

Dr. Robert W. McLaren, Thesis Supervisor

AUGUST 2007

The undersigned, appointed by the dean of the Graduate School, have examined the [thesis] entitled

COLLISION FREE PATH PLANNING ALGORITHMS FOR ROBOT NAVIGATION PROBLEM

Presented by Kyung min Han,

A candidate for the degree of Masters of Science

And hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Robert W. McLaren, Professor Emeritus, Electrical and Computer Engineering

---

Dr. Guilherme Desouza, Assistant Professor, Electrical and Computer Engineering

---

Dr. Roger, Fales, Assistant Professor, Mechanical and Aerospace Engineering

ACKNOWLEDGEMENTS

Looking back on my master's research years, I realized that I have gone through a great program which strengthened my academic knowledge and gave me a broader scope of what the electrical engineering discipline really is. Needless to say, I have faced a number of situations that seem to hard to overcome. However, I was lucky enough to have great faculty members and friends who always are ready to help me out.

First of all, I deeply thank my Lord for that He always listens to me whenever I pray for myself.

Especially, I feel a great thankfulness for my adviser Dr. Robert Mclaren. His precious advice and support always encouraged me in the right direction of my research goal. It would not have been possible for me to finish my work without his kind helps and wise suggestions through all my master's years.

I am also grateful to Dr. Guilherme DeSouza for his valuable teaching and encouragement that always led me to make a right decision whenever I faced difficult situations.

I would like to express my acknowledgement to Dr. Roger Fales for kindly agreeing to join my thesis committee.

I am also grateful to Dr. Keller who offered me such a useful teaching and study opportunity which helped me to have a better idea in my research topic

I am grateful to Dr. Tai seung Jang for his generous help and advice as my senior. I am pleased to thank to my friend Bon seok Koo for his valuable help. I am thankful my lab mates: Yuanqiang Dong, Hui Peng, Vishal Rijhwani, Youyou Wang and Thomas Konig.

Last, but not the least, I deeply thankful to my family members in South Korea: My father Haeng yong Han, my mother Jung shim Yoo, my sister Hye jin Han for cheering me up to confront new challenges in my life.

TABLE OF CONTENTS

ABSTRACT

Path planning problem, including maze navigation is a challenging topic in robotics. Indeed, a significant amount of research has been devoted to this problem in recent years.

Genetic algorithm is a popular approach that searches for an optimal solution in given set of solutions. Considering via points as genes in a chromosome will provide a number of possible solutions on a grid map of paths. In this case, path distances that each chromosome creates can be regarded as a fitness measure for the corresponding chromosome. In some cases, a solution path passes through an obstacle. Assuming that the shape of an obstacle is a circle, such random solutions can easily be eliminated by setting-up simple equation between a line created by two via points and the obstacle.

The ant colony optimization algorithm is another approach to solve this problem. Each ant drops a quantity of artificial pheromone on every point that the ant passes through. This pheromone simply changes the probability that the next ant becomes attracted to a particular grid point. Since each ant will make a decision at every grid point that it encounters, it is possible that an ant may wander around the grid map or may become stuck among local grid points. In order to prevent this phenomena the proposed solution adapted a global attraction term which guides ants to head toward the destination point.

This thesis addresses methods of the path finding problem using these two different approaches. Both algorithms are tested and compared in the result section. The experiment results demonstrate that these two methods have a great potential to solve the proposed problem.

1.0 Introduction

1.1 Path Planning

The robot path planning problem is a very challenging problem in robotics. The main goal of this problem is to construct a collision-free path from a starting position to an end or destination position. However, this navigation problem includes several difficult phases that need to be overcome, such as obstacle avoidance, position identification, and so forth. As Ibrahim [17] pointed out, this problem can be broken in to several subtasks. A reliable navigation algorithm must be able to 1) Identify the current location of the robot, 2) Avoid any collisions, 3) Determine a path to the object. For this reason, mobile robot navigation problem is a challenging problem, and a number of studies have been attempted, resulting in a significant number of solutions. Three major concerns in regard to robot navigation problems are efficiency, safety and accuracy. The efficiency of the algorithm is considered as an important matter since one of the main concerns is to find the destination in a short time. Accordingly, a desirable path should result from not letting the robot waste time taking unnecessary steps or becoming stuck in local minimum positions. Furthermore, a desirable path should avoid all known obstacles in the area. This safety issue is another critical part of the algorithm. Once the optimum collision-free path is constructed, then it is a matter for the robot to accurately follow the pre-determined path. The main scope of the path finding problem involves the efficiency and safety issues. A number of algorithms have been proposed to address these two important issues. D Huh, J. Park, U. Huh and H. Kim [4] approached the path finding problem by combining global path planning and local path planning. They used the Dijkstra algorithm for global path planning and the potential field method for local path

planning. S. Lee and G. Kardaras, [5] used via points (VP) to find the optimum path. They developed a “smart” algorithm which could change the number of via points in response to a different level of complexity of the map upon which paths would be generated. N. G. Bourbakis and L. Vlachavas [6] presented a path planning algorithm that uses a neural network and a skeletonization technique. N. Sadati and J. Taheri [7] presented a combination method consisting of a Hopfield Neural Net ((NN) and a genetic algorithm (GA). Thus, numerous approaches to solving the path finding problem have been published, including those indicated previously. This thesis will apply two approaches to solving the path finding problem: 1) a genetic algorithm (GA) and 2) an ant colony optimization algorithm (ACO).

The remainder of this thesis is organized into five major chapters. In the following chapter, Chapter 2 discusses historical reviews of the Genetic Algorithm and the Ant Colony Optimization algorithm. Chapter 3 describes the proposed problem. Chapter 4 introduces two proposed solutions (GA, ACO) for the proposed problem. In Chapter 5, the proposed solutions are implemented and demonstrated through computer simulations. Then, the simulation results are presented. Chapter 6 provides a conclusion.

2.0 Historical overview of the GA and ACO algorithm

2.1 Genetic Algorithm

An evolutionary computational strategy was first proposed by Rechenberg and Schwefel in 1965 [1] as a numerical optimization technique. In the mid 1960's, Fogel proposed the first evolutionary program. These two steps can be considered as pioneering works in the discipline of evolutionary computation. However, the current frame-work of the so-called "genetic algorithm" or GA was developed by John Holland in 1975 [2]. His pioneering book, *Adaptation in Natural and Artificial Systems* presented an evolutionary method that involves natural selection, crossover, and mutation. The genetic algorithm has become a well-known technique for optimization, intelligent search and machine learning. For example, the GA represents a feasible approach to the classical traveling salesman problem, flowshop optimization, job shop scheduling, and the like [3].These problems bear a strong similarity in that the main objective of these problems is that of optimizing or selecting the best solution out of a number of possible solutions. Thus, GA has a reasonable motive of being employed in a path optimizing problem.

2.3 Ant Colony Optimization

The Ant Colony Optimization (ACO) method is a more recent, but very active research area in the discipline of computational intelligence. The first ACO algorithm, called the ANT System, was developed by Marco Dorigo and his colleagues [10]. This algorithm takes inspiration from the social behavior of ants. The central concept of this algorithm is based on the pheromone trail and the following behavior of real ants. Based on biological studies, some species, such as ants are stimulated by how well they have

performed (certain) tasks. The stimulation in the world of the ant can be equivocated to their pheromone trail. In the ACO algorithm, a collection of artificial ants construct potential solution to a problem requiring optimization based on this pheromone (feedback) information. The constructed solutions are then evaluated by their quality in terms of the performance achieved. Then, the pheromone trail is updated in accordance with this evaluation. The so-called "ant cycle algorithm" of Dorigo [8] was based on the above principle. His algorithm was applied to the classical traveling salesman problem along with the asymmetrical traveling salesman problem (ATSP) and the job scheduling problem. The results achieved, as reported in his paper [14], demonstrated the algorithm's versatility as well as its robustness. Similar to other biologically inspired algorithms, such as Particle Swarm Optimization [18], the ACO algorithm performs well in several different optimization problems. From the early 90's, after the first ACO or AS [10] algorithm(s) were proposed, a significant number of successful applications became available. Examples of these applications include a classical TSP, telecommunication routing problems, and the "single machine total weighted tardiness scheduling problem" (SMTWTP) [8]. The algorithm performs well in finding an optimal path or an optimal sequence of ants' steps that defines a path. In fact, M. Dorigo and M. Birattari [9] presented a theoretical approach to a classical traveling salesman problem. Thus, it is claimed that the ACO algorithm is a feasible approach to the proposed path planning problem. A significant portion of this report shows how the ACO meets such expectation.

3.0 Problem description

Consider a 2-D square map overlaid with a uniform pattern of grid points. The size of a map can be changed arbitrarily; here, the map consists of a 20 x 20 grid. The left bottom corner of the map is the starting point for a path while the right top corner of the map is the destination point for a path. The shape of an obstacle is always a circle, but the size of the obstacle is variable from 1 to 5 grid points. The positions of the obstacles are randomly selected and can be located at any grid point in the map except at points close to the starting point region or close to the goal point region (say 5 grid points away.) Furthermore, multiple obstacles are possible. Figure 3.1 shows an example of such an arrangement.

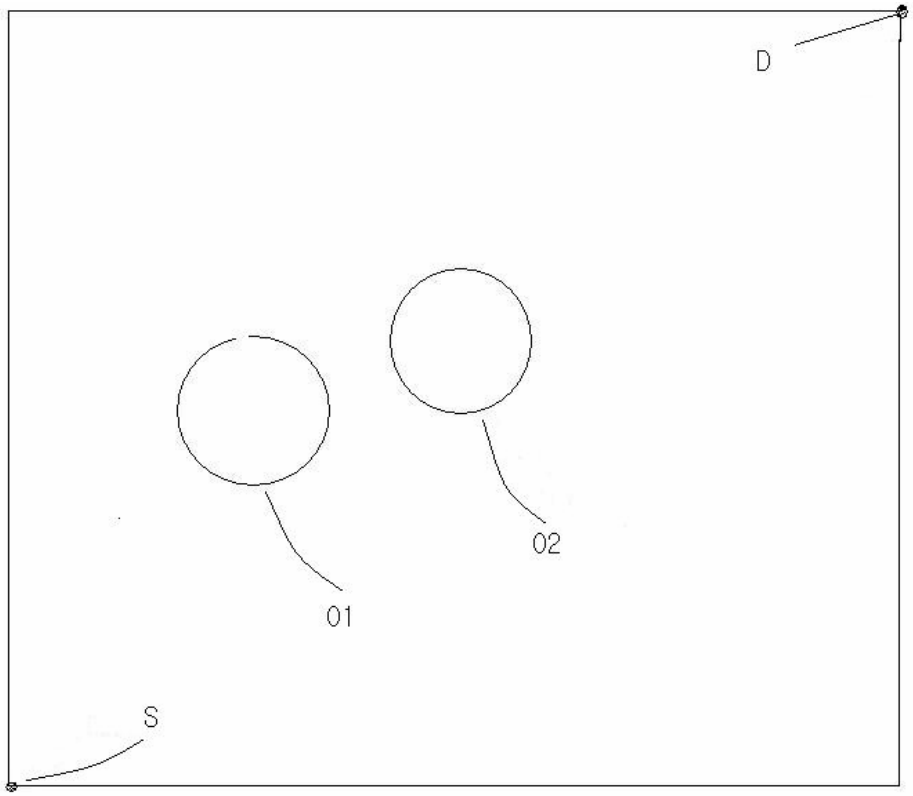


Fig 3.1: problem configuration (S: starting point, O1, O2: obstacles, D: destination point)

The goal is to construct a shortest path from the starting point, S, to the destination (goal point,) D, which avoids every obstacle in the map; (a path cannot touch any obstacle) In order to solve this problem, two approaches, 1) the genetic algorithm and 2) the ACO algorithm will be proposed in the next section.

4.0 Proposed solutions

4.1 Genetic algorithm

- *Definitions of keywords [16]*

Chromosome: a set of parameters (genes) which define a possible solution of the proposed problem.

Fitness: Performance of each chromosome in terms of its output performance evaluated by a fitness function.

Selection: Survival of chromosome that is the best fit in current generation.

Crossover: Exchange of bits between two parents.

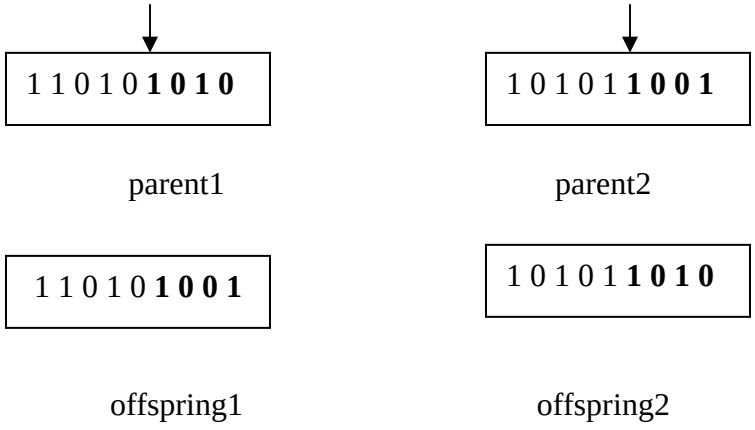


Fig 4.1a: Example of a single point crossover in third bit position from the LSB

Mutation: Random change of one or more bits in chromosome.

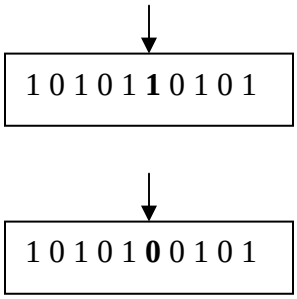


Fig 4.1b Example of a mutation (mutation of a chromosome at the 6^{th} bit)

In the proposed algorithm via points for a path represent genes or bits in each chromosome. Since the algorithm searches for the best path, these bits are changeable. However, the left most chromosome is always the staring point (0, 0) and the right most chromosome is the destination point (20, 20). For example, if a path has 3 via points (1, 1) (5, 5) (10, 5), the 5 genes chromosome for this path is [(0, 0) (5, 4) (9, 7) (18, 17) (20, 20)]. Each grid point, (x, y) represent a gene of a chromosome. Fig 4.1c shows this configuration.

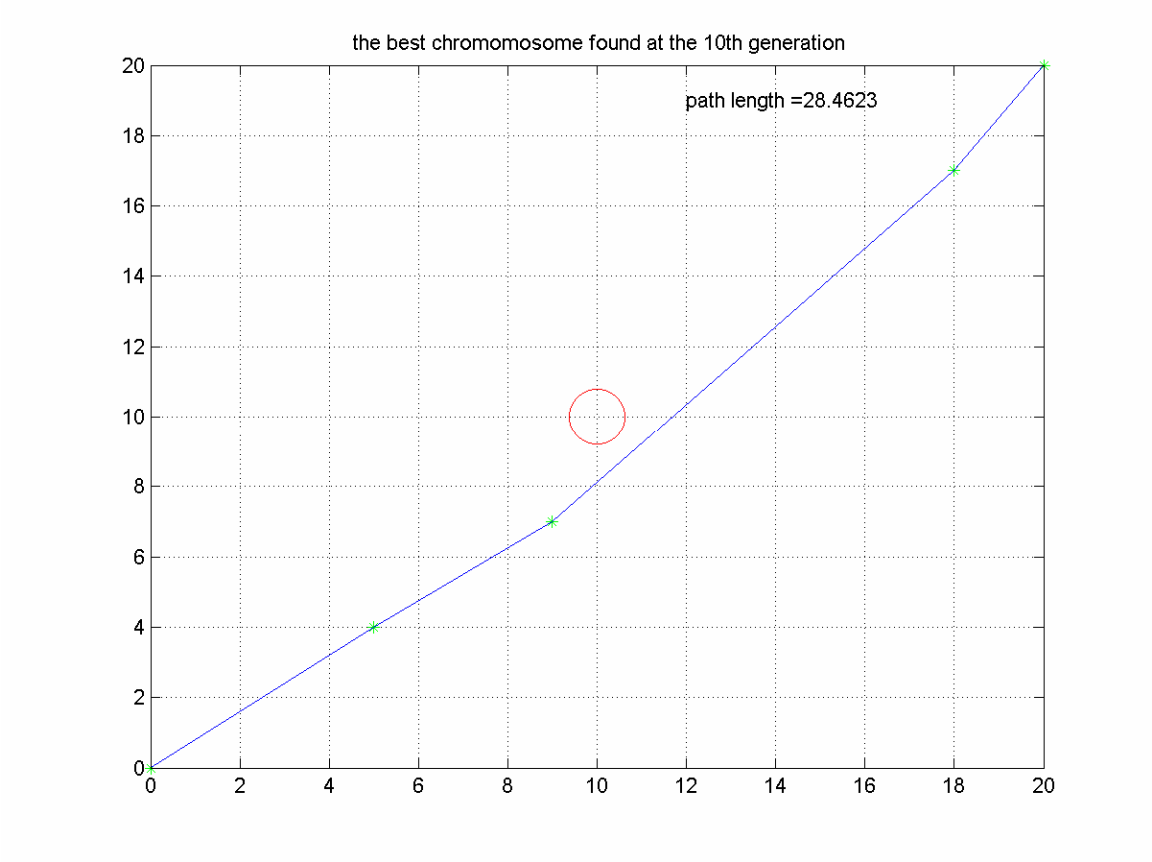


Fig 4.1c: A chromosome (path) with 3 genes (via points)

The proposed genetic algorithm consists of 3 main steps: natural selection, crossover, and mutation. The methodologies associated with these steps are described in subsection i) through subsection v). Subsection vi) explains the obstacle avoidance technique. Subsections vii) through viii) show other important features of the proposed algorithm such as the rules and assumptions, and the entire program sequence.

i) Natural selection

At each generation (iteration), all the chromosomes will be updated by their fitness. In other words, if a particular chromosome has better fitness (shorter path distance) than other chromosomes, then that particular chromosome is more likely to win the competition and clone itself. Thus, a chromosome with good fitness has a much higher probability than other inferior chromosomes to appear in the next generation.

ii) Crossover

A group of chromosome undergoes crossover at each generation. Furthermore, all the crossover events are controlled by a predetermined Pc (crossover rate). In other words, the algorithm creates a random number in [0 1] for each chromosome. If the generated number is less than Pc, the chromosome is a candidate for the crossover event. The left most genes and the right most genes will avoid the crossover event since these two points cannot be eliminated. For the purpose of diversity, the crossover point bit is randomly selected in each generation.

iii) Mutation

Unlike the crossover event, mutation is performed on a bit by bit basis. That is, if the given mutation rate is Pm=0.01, the population is expected to perform a 1% bit mutation. For example, if 20 chromosomes exist with 5 bits in each one, then a 20 X 5 X 0.01=1 bit mutation is expected to happen.

iv) Fitness function

The fitness of each chromosome is evaluated in terms of its path distance. Thus, the smaller that the distance is, the better that corresponding chromosome will be. The fitness function used in this particular problem is $ft(k) = \frac{1}{d(k)}$, where d(k) is the path length for the kth chromosome.

v) Elitism

In order to keep the best chromosome from each generation, the elitism method is employed. The main goal of the elitism rule is to keep the best chromosome from the current generation. Thus, under this rule, the best chromosome from each generation will

not undergo any mutation or crossover event and will safely move onto the next generation.

vi) Obstacle avoidance

New chromosomes appear in every generation. Even if the performance based on (path distance) of a new chromosome is acceptable, it could be a useless solution if the corresponding path passes through any of the obstacles. Since via points cannot lie on any of obstacles, a line created by two via points either passes through an obstacle or not. Knowing this fact, one can check all the lines created between via points. For example, consider a chromosome with 1 via point: (0, 0) (10, 10) (20, 20). Also, say an obstacle is located at a position of (12, 12) with a radius of 2. Then, starting from the first line created by two via points: (0, 0) (10, 10), the algorithm checks if the line passes through the obstacle. The obstacle can be expressed as $(x-12)^2+(y-12)^2=4$ (4.1.1). Furthermore, the line equation is given by y = x, where 0<x<10 (4.1.2) Combining Eqs. (4.1.1) and (4.1.2) and will yield two possible solutions. If the X coordinate of the obstacle lies between two given via points and the two solutions are real numbers, then the algorithm declares that this line will pass through the obstacle and will then eliminate this chromosome. Fig 4.1.2 depicts this concept. Although chromosome 1 has better performance compared to chromosome 2, only chromosome 2 will survive because the chromosome 2 passes through the obstacle.,

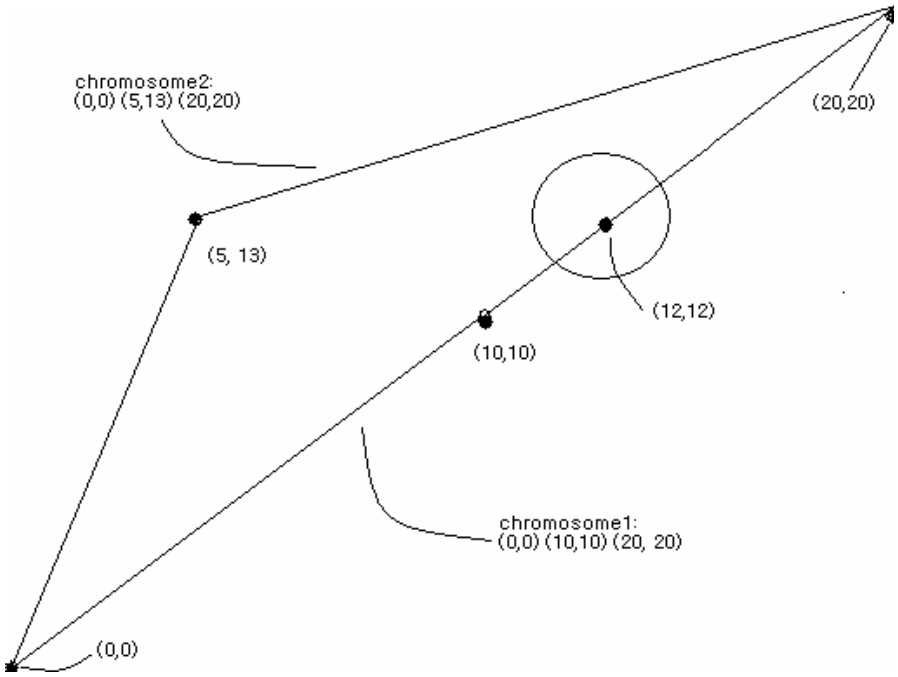


Fig 4.1.1: Obstacle avoidance example

vii) Rules and assumptions

- Only integers are employed to locate positions on the map. The size of the map is 20 X 20 grid points. The starting point is (0, 0), the goal point is (20, 20), and none of obstacles can include the starting point or the goal point.
- The first bit and the last bit of a chromosome is (0, 0) and (20, 20) respectively. If 3 via points are chosen, then the chromosome may look like [(0, 0), (x1, y1), (x2, y2), (x3, y3), (20, 20)].
- The shape of an obstacle is a circle with a radius of 1 or 1.5 grid points most of the time. However, a few of the experiments are done with a radius of 5 grid points.
- A via point cannot be the starting point, the end point, and any point inside of an obstacle.

viii) Program sequence

Choose the size of the population (the number of chromosomes)

Choose the size of a chromosome

Create the number of obstacles and their positions (Xo, Yo)

Choose the mutation rate (Pm) and the crossover rate (Pc)

Initialize the population at the chosen size, 100, for example

Do For generation=1, 2, ….. N

Compute the fitness (the distance of the path from (0,0) to (20,20)) of the chromosomes

Apply cloning and natural selection to each chromosome

Apply a crossover technique to each chromosome

Apply mutation

Eliminate all chromosomes (paths) that pass through any of the obstacles.

If the size of current population < the size of the original population

Generate extra chromosomes which avoid all obstacles.

End if

End for

4.2 ACO (Ant Colony Optimization) algorithm

- *Definitions of keywords [10]*

Artificial ant: Ant artificial agent which makes movements based on the attraction of pheromone.

Pheromone: Chemical substance deposited by an ant when walking; each ant probabilistically prefers to follow a direction rich in pheromone rather than a poorer one.

i) Methodology

The TSP (Travel salesman problem) is a paradigmatic optimization problem since it is used to demonstrate the original AS (Ant System) problem. Since then it has often been used as a benchmark to test the new ACO concept. The proposed path-finding algorithm employs the original concept of the ACO algorithm with some modification. The idea of the proposed algorithm is as follows.

Starting from the grid point (0, 0), an ant iteratively moves from a grid point to one of its neighboring grid points. When at the ith grid point (x,y), ant k can choose the next (jth) grid point by choosing one of its 8 neighbor locations: [(x+1, y+1), (x+1,y), (x, y-1), …..] $\in$ N, where N in general is the set of all neighboring locations of the current location. The ant takes its next step randomly, based on the probability given by

$$\varphi_{ij}^{k}(t) = \frac{\tau_{ij}(t)}{\sum_{l \in N} \tau_{il}} + \alpha * \Omega_{ij}(t) \qquad (4.2.1)$$

Where $\tau_{ij}(t)$ is accumulated pheromone on the jth grid point when the ith grid point is the ant's current location at time t. The quantity τ_{il} indicates every possible lth neighbor point when the ant is in ith position. $\Omega_{ij}(t)$ is the dot product of the vector from i to j with the vector i to destination point. The first term is associated with the pheromone amount in the 8 neighboring grid points (a local pheromone). The second term is associated with a global attraction, where α is a scale parameter. The global term is

defined as the dot product of the agent's heading direction and the food (destination) direction. The purpose of the global term is to guide the agent in the desired direction so that a large number of ants can reach the goal point in a limited number of steps

(x-1, y+1)	*(x, y+1)*	*(x+1,y+1)*
(x-1, y)	**(x, y)**	*(x+1, y)*
(x-1, y-1)	*(x, y-1)*	*(x+1, y-1)*

Fig 4.2.1: An agent's current position (bold) with 8 possible next positions (italic)

The solution construction ends after each ant reaches the destination or the ant took too many steps (50 or 100 steps, for example). However, the pheromone will be deposited only if the ant reaches to the destination position in less than a certain number of steps. Thus, after a group of agents finished its tour, the pheromone in the entire map will be updated by

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\gamma_{ij}^{k}(t) \qquad (4.2.2)$$

Where $0 < \rho < 1$ is the pheromone forgetting parameter. This parameter prevents the map from unlimited accumulation of the pheromone. The quantity $\Delta\gamma_{ij}^{k}$ is the amount of pheromone that ant k deposits on the map. It is defined as

$$\Delta\gamma_{ij}^{k}(t) = 1/L^{k}(t) \qquad (4.2.3)$$

Where $L^{k}(t)$ is the length of the kth ant's tour (path). When the kth ant cannot reach the destination in a certain number of steps, let the $L^{k}(t)$ =Infinity.

i) The elitist strategy

The main idea of the elitist strategy is to give a significant additional weight to the best tour constructed from a group of agents. In other words, each time the pheromone trails are deposited, those belonging to the grid points of the group's best tour get an additional amount of pheromone. For these grid points, equation (4.2.4) becomes:

$$\Delta\gamma_{ij}^{gb}(t) = e / L^{gb}(t) \qquad (4.2.4)$$

Again, the L^{gb} represents the length of the tour, and e is a positive integer which reinforces the global best path.

Algorithm pseudo code

```
Do For iteration=1, 2, ….. N
        Do For ant=1,2, …. K
                Do For step=1,2, …. M
                        Compute the probability of the kth ant's next locaton.
                        Move to a next grid point by the computed probability
                        Store the history of past location points in an array
                        If the current location is equal to the destination
                                Break the step loop
                        End if
                End For
                Store the tour distance made up by kth ant
                Compute the pheromone amount which is generated by the kth ant
        End For
        Update pheromone amount of the entire map
End For
```

5.0 Results

The results of applying the GA (Genetic Algorithm) and ACO (Ant Colony Optimization) are presented in this section. This section is divided into two subsections. In order to make objective conditions regarding the proposed algorithms, different test conditions were set up to create variety of experiments. All simulations were done with MATLAB. Appropriate diagrams and graphs are included to help illustrate results.

5.1 Results with the genetic algorithm.

Experiments were performed with a different number of obstacles, mutation rates, crossover rates, and the number of chromosome bits. Furthermore, these studies include the effect of elitism. In every figure, the circular objects represent the obstacles, and the star objects represent the chromosome bits. However, the size of an object in diagrams could be smaller than its actual size.

Experiment 1: When obstacles do not block the optimum path of the map

- Parameter specifications for the Experiment 1

Pc (crossover rate)	0.25
Pm (mutation rate)	0.2
No. of obstacle	2
Radius of the obst acles	1
No. of via points	3
Elitism usage	yes

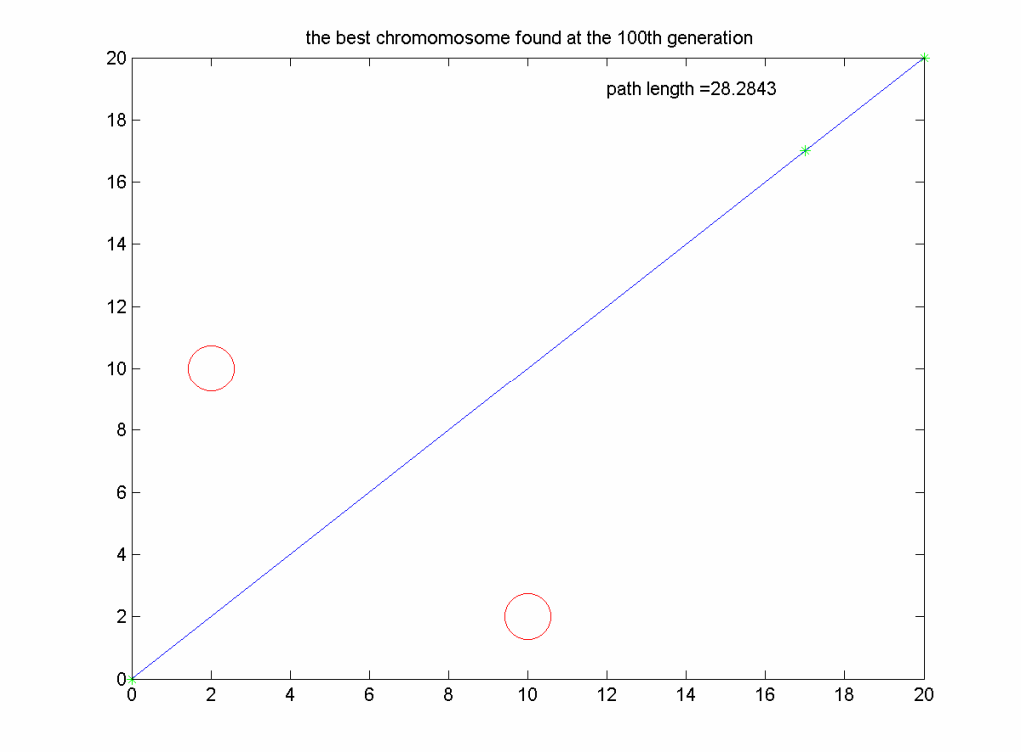


Fig 5.1.1: The best chromosome (solution) at the 10^{th} generation

- Discussion

The interpretation of the above figures is straightforward. Since neither of the obstacles have any influence on finding the best solution, the algorithm found an optimum solution at only the 10^{th} generation.

Experiment 2: When two obstacles block the shortest path

- Parameter specifications for the scenario 2

Pc	0.25
Pm	0.2
No. of obstacle	2
Radius of the obst acles	1
No. of via points	3
Elitism usage	yes

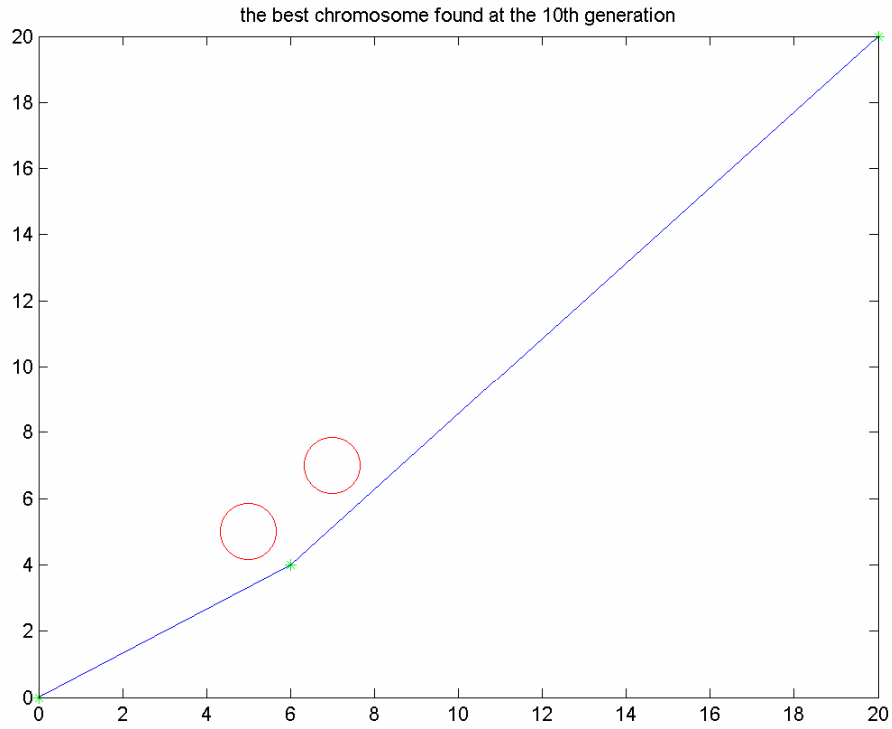


Fig 5.1.2: The best solution at the 10th generation

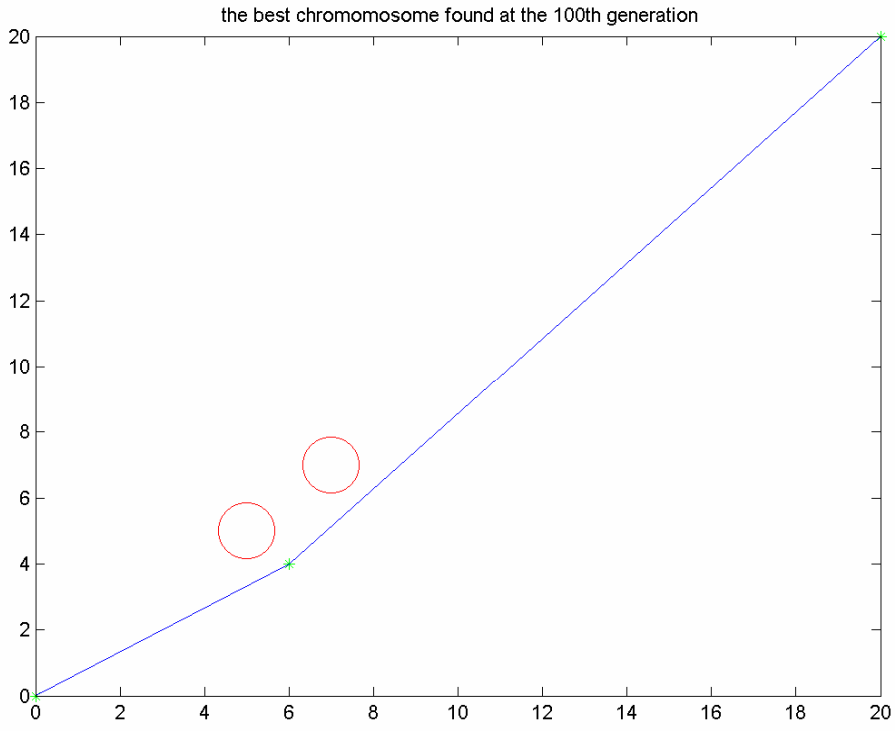


Fig 5.1.3: The best solution at the 100th generation

- Discussion

  Again, an optimum solution was found in a relatively few numbers of generations. Note that there is no difference between the 10^{th} generation solution and the 100^{th} solution.

Experiment 3: When a multiple obstacle exists

- Parameter specifications for experiment 3

Pc	0.25
Pm	0.4
No. of obstacle	9
Radius of the obstacle	1
No. of via points	4
Elitism usage	Yes

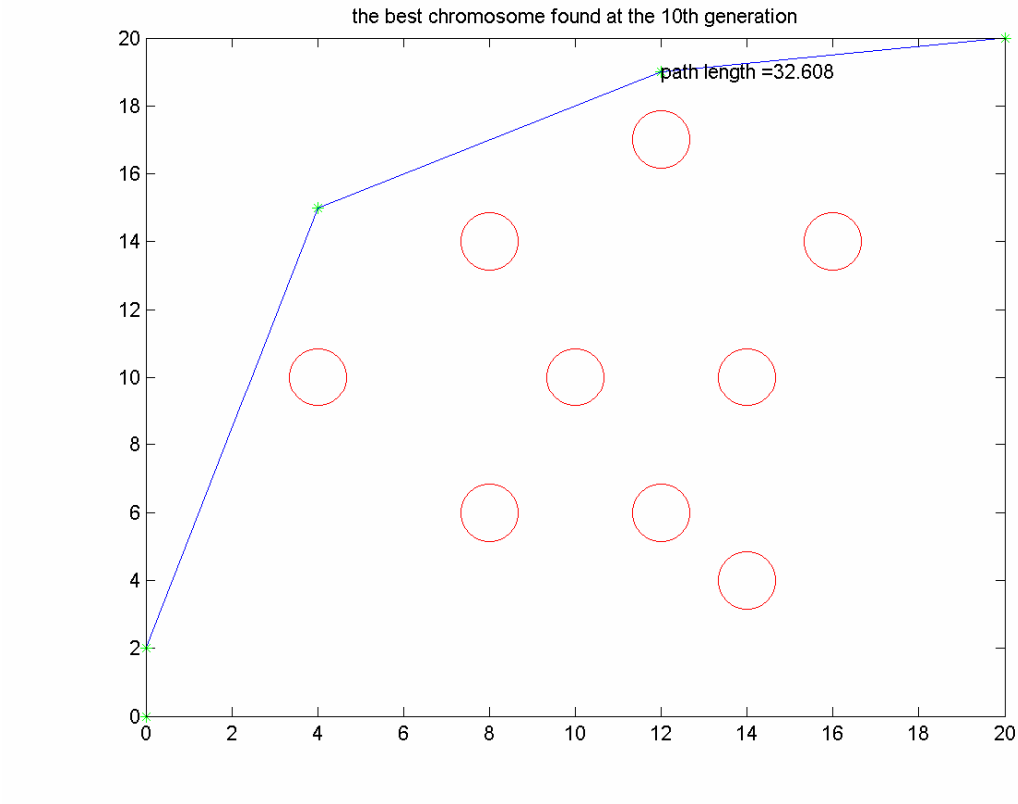


Fig 5.1.4: The best path at the 10^{th} generation

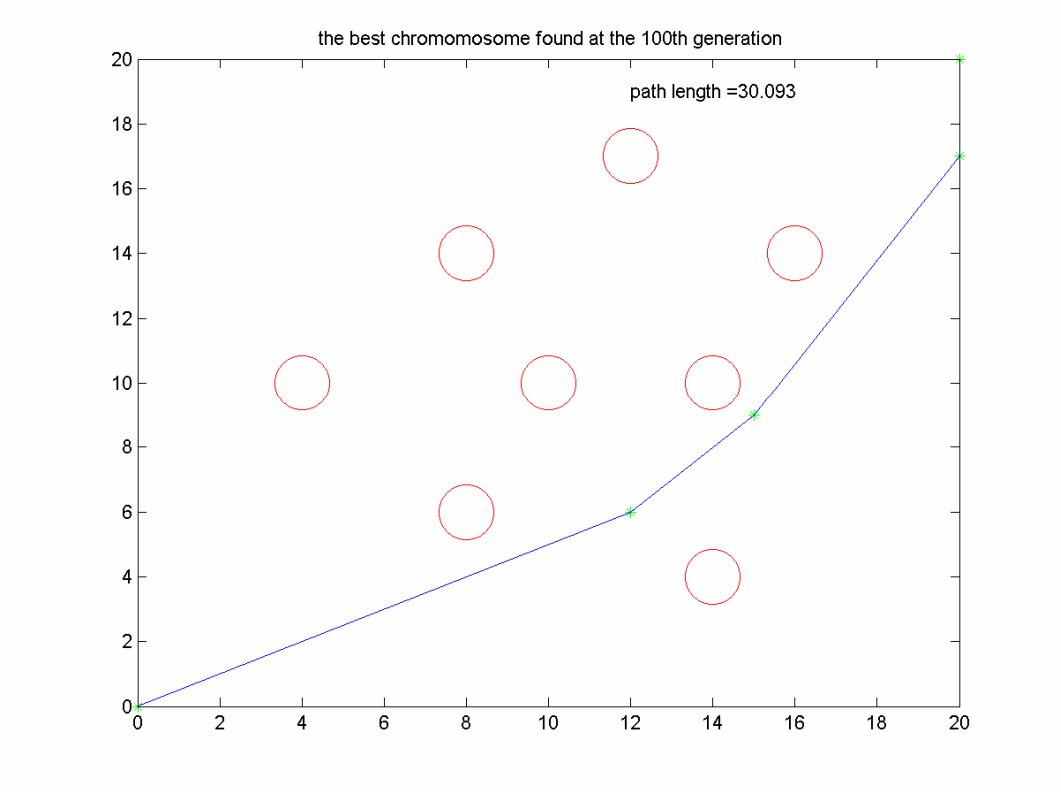


Fig 5.1.5: The best path at the 100th generation

- Discussion

As the number of obstacle increases, the algorithm needs more generations to find the best solution. In general, if the map complexity increases in terms of the number of obstacles as well as its distribution pattern, it is necessary to increase the number of bits in a chromosome. Such a consequence was expected in the sense that similar characteristics are easily found in other computational intelligence algorithms. For example, an artificial neural network requires a large number of hidden neurons if the network has to deal with high uncertainty situations or nonlinearities.

It is verified that the best solution at the 100th generation is better than that at the 10th generation.

Experiment 4: A large obstacle with a different number of chromosomes.

- Parameter specifications for the Experiment 4

Pc	0.25
Pm	0.2
No. of obstacle	1
Radius of the obstacle	5
No. of via points	1, 2, 3
Elitism usage	yes

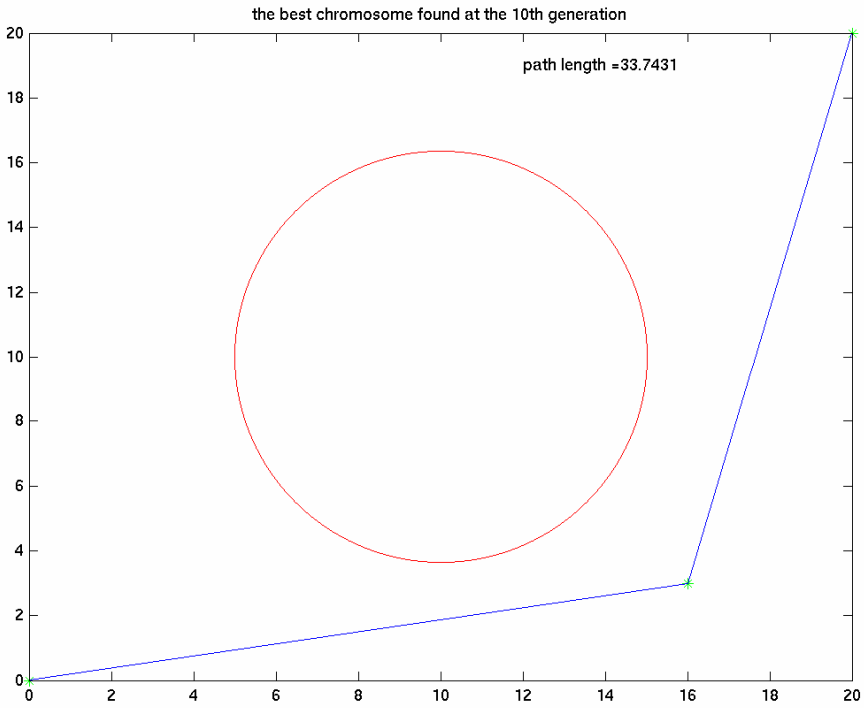


Fig 5.1.6: One via point

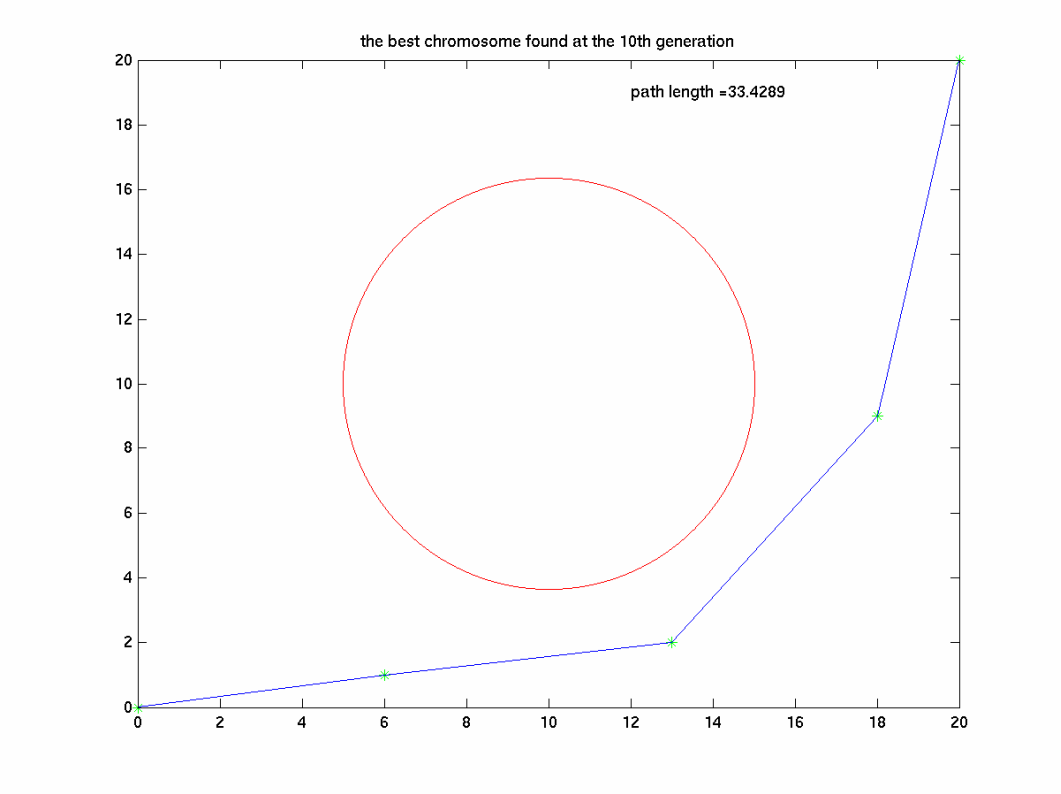


Fig 5.1.7: Two via points

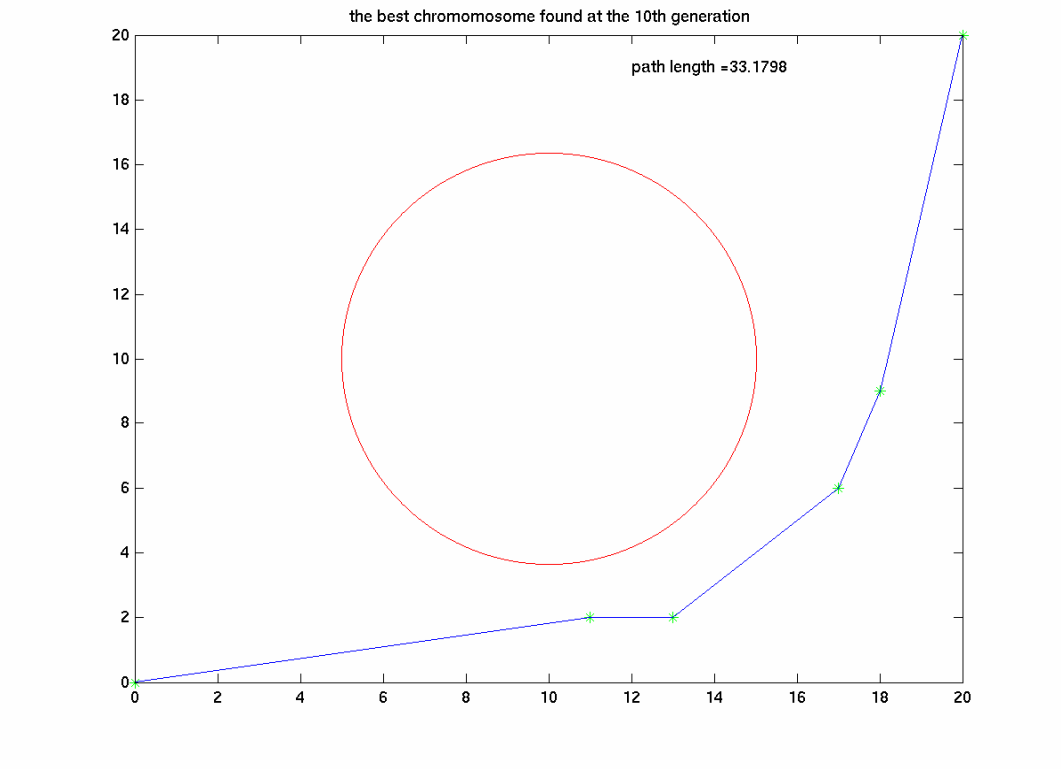


Fig 5.1.8: Three via points

- Discussion

The purpose of the experiment 4 is to see if the number of via points has an effect on the performance. Theoretically, more via points should lead to shorter paths. Indeed, as shown in the above figures (fig 5.1.6 through 5.1.8), the performance with three via points (33.1798) is better than for the other two cases. However, as the number of via points increases, the algorithm requires a larger number of generations to converge.

Experiment 5: Elitism effects on the performances

Pc range	0 ~ .5
Pm range	0 ~ .8
No. of obstacle	1
Radius of the obstacle	5
No. of via points	1, 2, 3
Elitism usage	Yes

It is interesting to see how the performance of the algorithm varies due to employing elitism. The following two Figures demonstrate the effect on the performance of the algorithm due to changing the crossover rate or the mutation rate. Both results are achieved by averaging 10 experiments for each case.

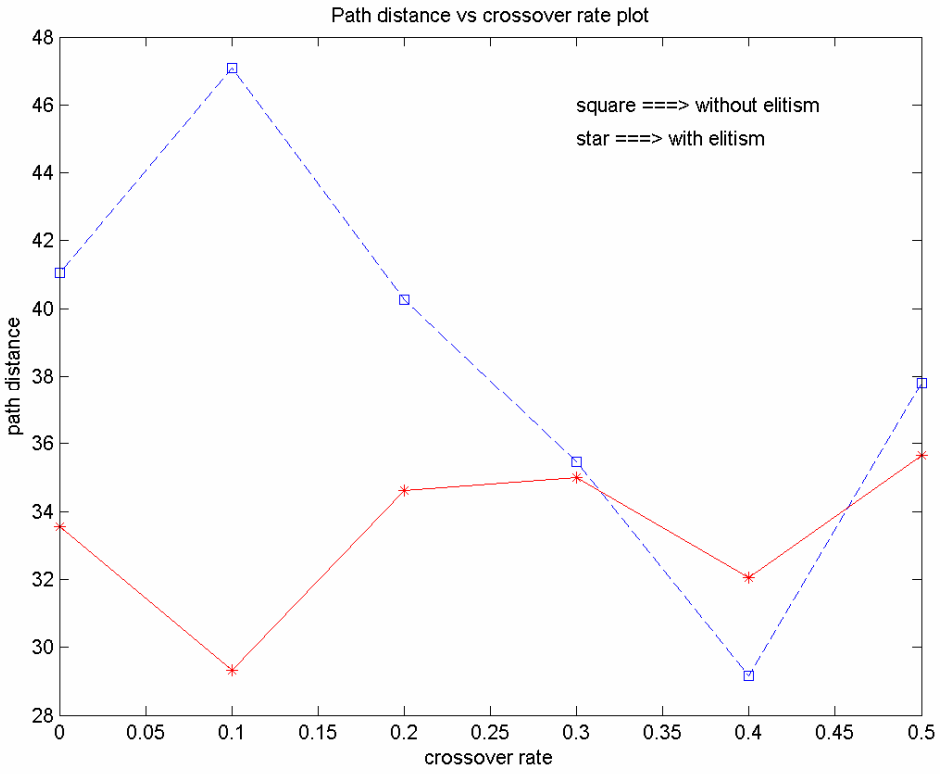


Fig 5.1.9: Path distance vs. crossover rate, Pc

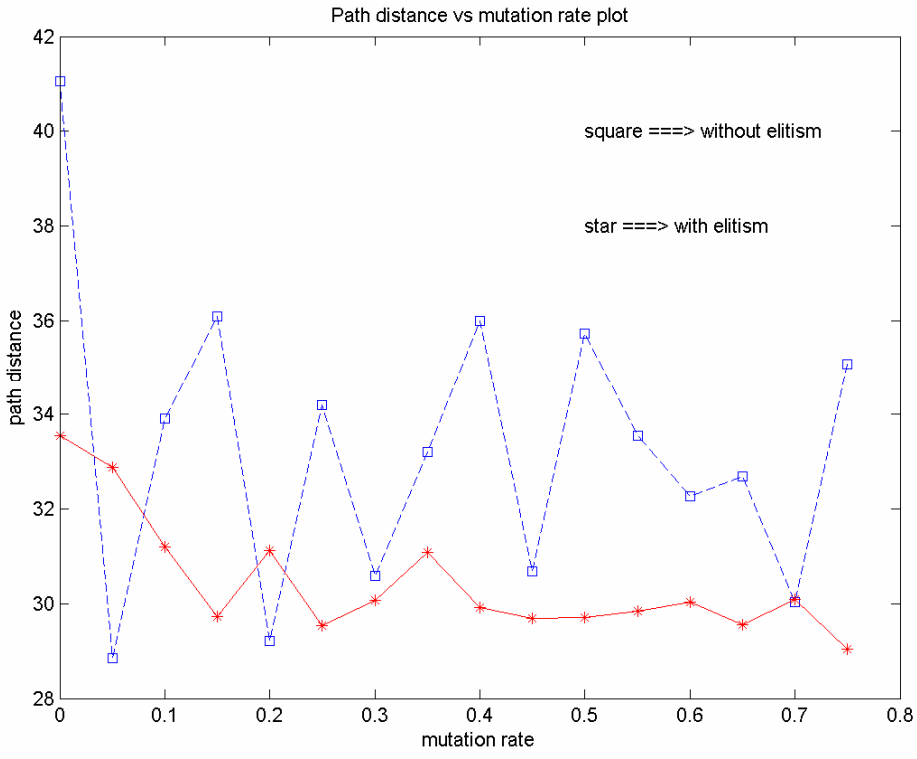


Fig 5.1.10: Path distance vs. mutation rate, Pm

- Discussion

The size and the number of obstacles in this experiment were the same as in Experiment 4. Fig 5.1.9 shows how the performance (path distance) varies as the crossover rate increases. Fig 5.1.10 shows how the performance varies as the mutation rate increases. Pm=0 in the Fig 5.1.19 and Pc=0 in Fig 5.1.20. Both of the figures compare the case with elitism and the case without elitism. According to Fig. 5.1.19, the crossover rate has little influence on the elitism. Figure 5.1.20 shows that a higher mutation rate will lead to better performance, which is not true for the case without elitism. Notice that employing elitism gives stable and better results in both cases in terms of its path distance.

5.2 Results with the ant colony optimization

With the ACO algorithm, the staring point is (1, 1) and the destination point is (20, 20). The obstacles can be located at any place on the map except at the starting point and at the destination point. The experiments were done with a different number of obstacles, different population sizes, and a different number of iterations. Furthermore, these studies include the effects of elitism. In every Figure, the circular objects represent the obstacles (there is some discrepancy between the actual obstacle and the apparent obstacle size in the figure.)

Experiment 1: When obstacles do not exist in the map (W/O elitism)

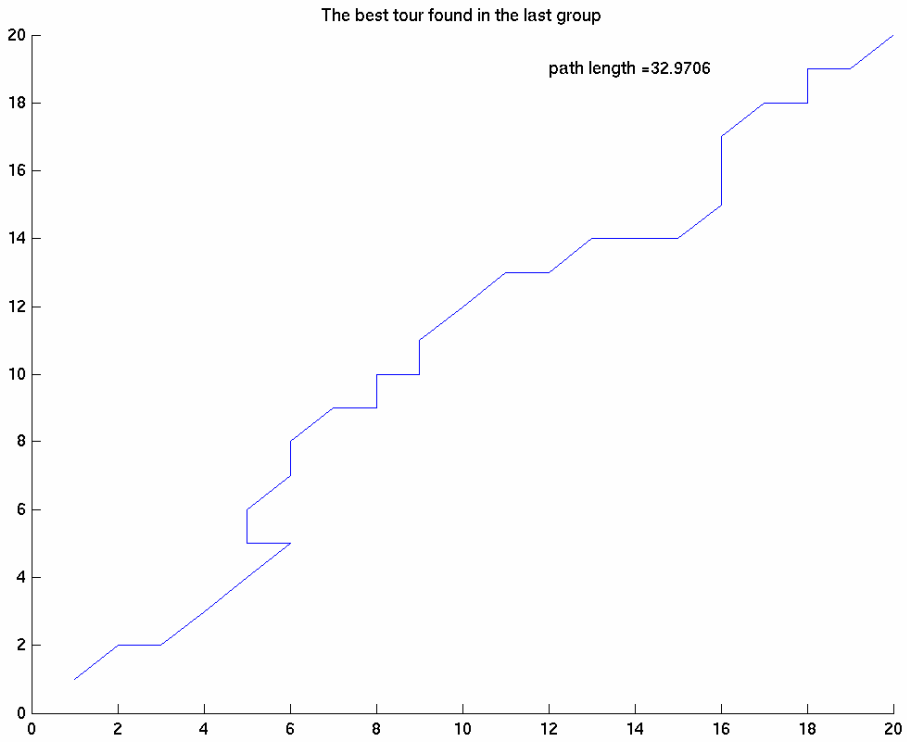


Fig 5.2.1: The best tour found at the 50th iteration

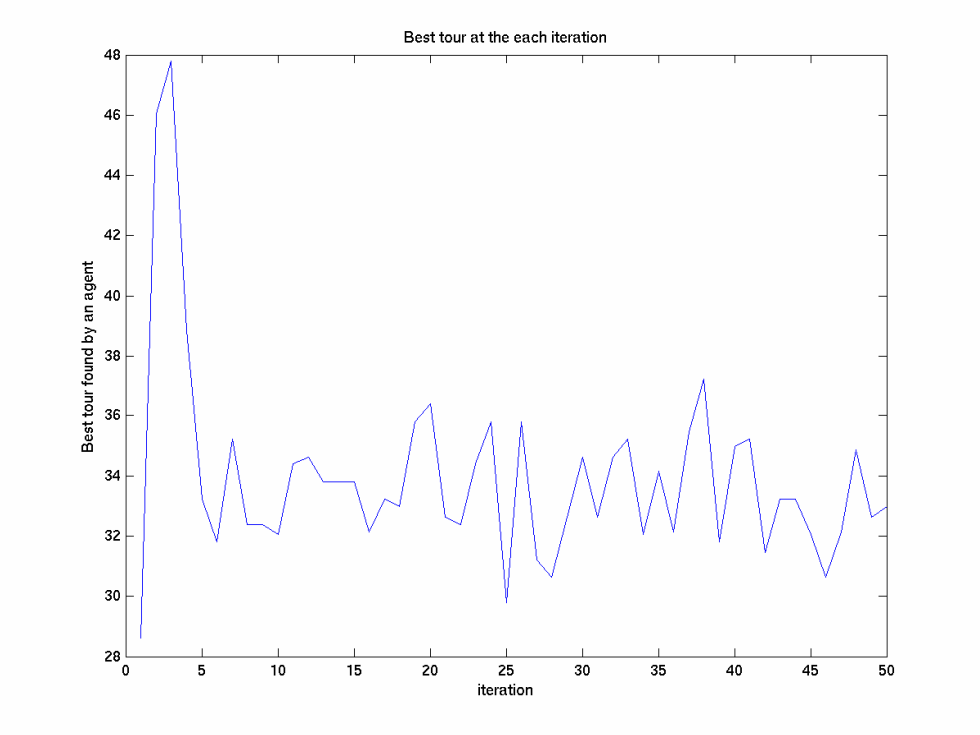


Fig 5.2.2: Best tours at each iteration (minimum distances at each iteration)

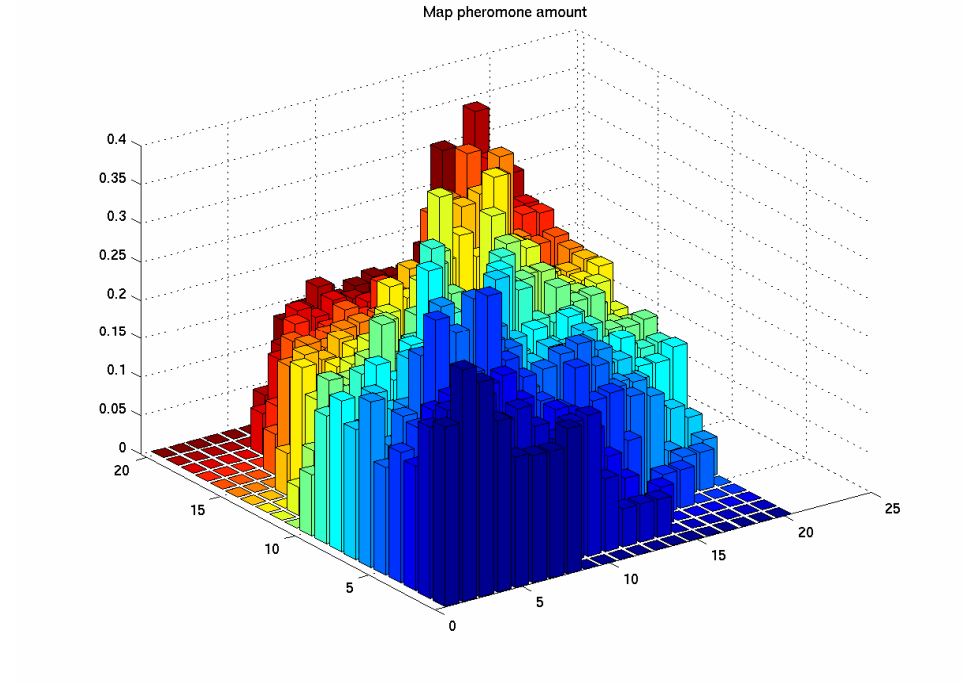


Fig 5.2.3: Deposited pheromone in 3D space

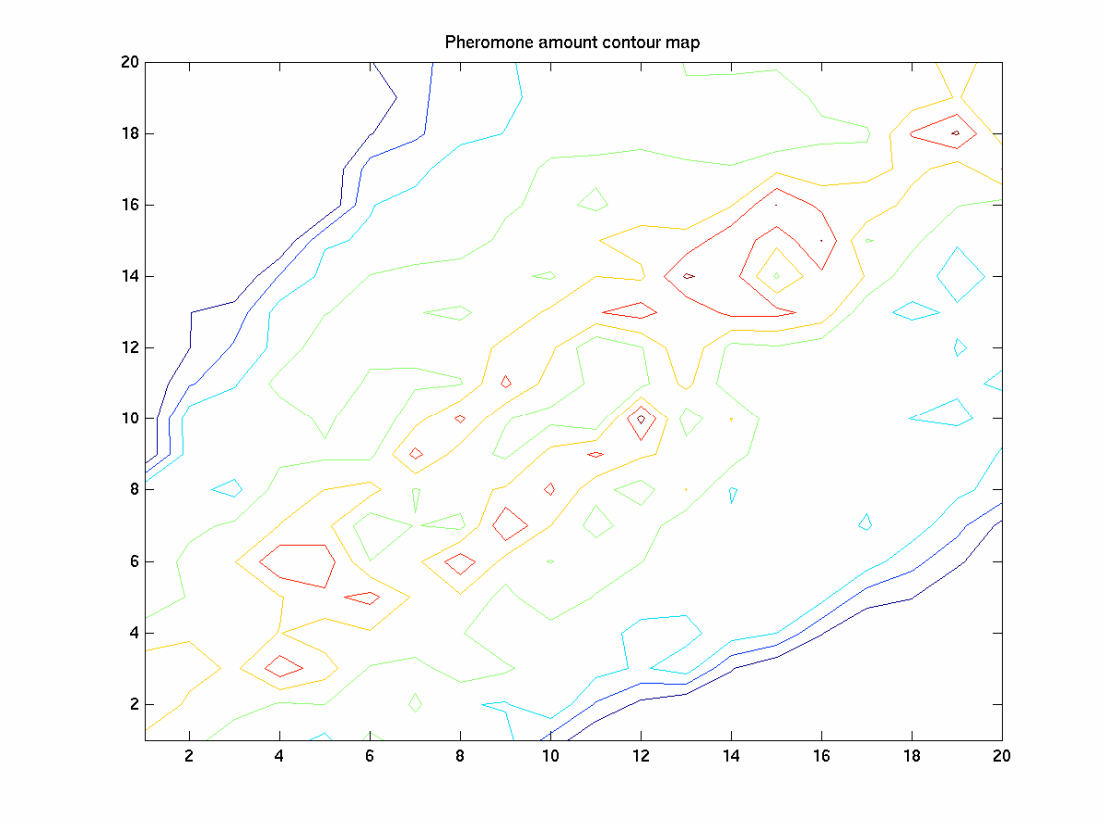


Fig 5.2.4: Deposited pheromone in 2D contour map (Red > yellow > blue)

Experiment 2: When obstacles does not exist in the map (W/ elitism)

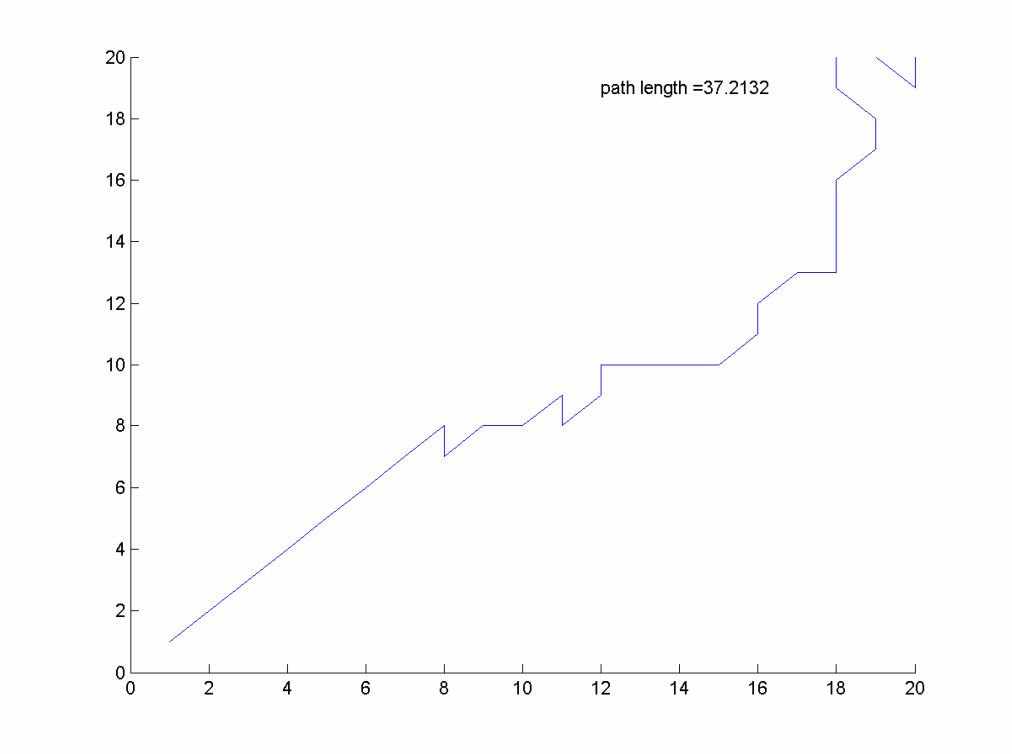


Fig 5.2.5: The best tour found at the 50th iteration

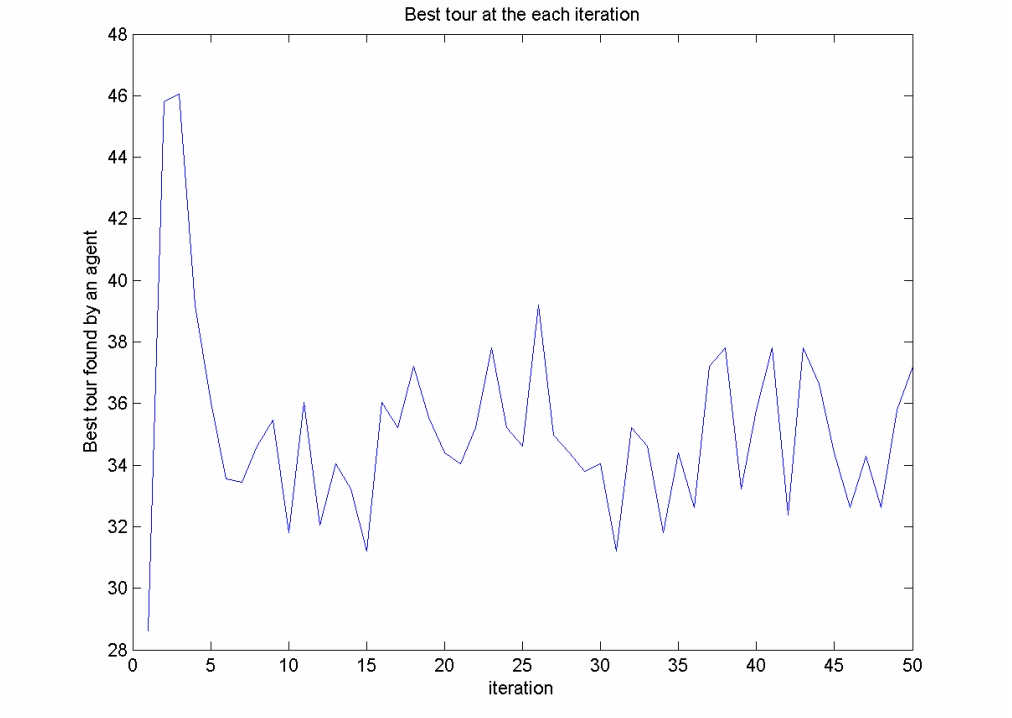


Fig 5.2.6: Best tours at each iteration

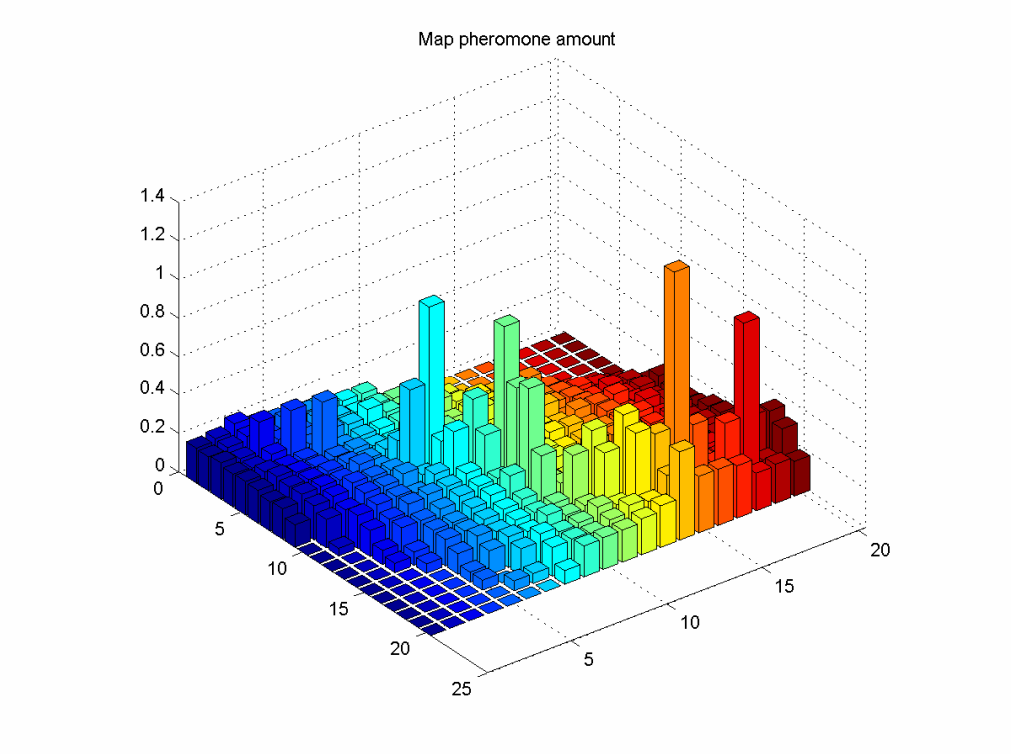


Fig 5.2.7: Pheromone accumulation in 3D

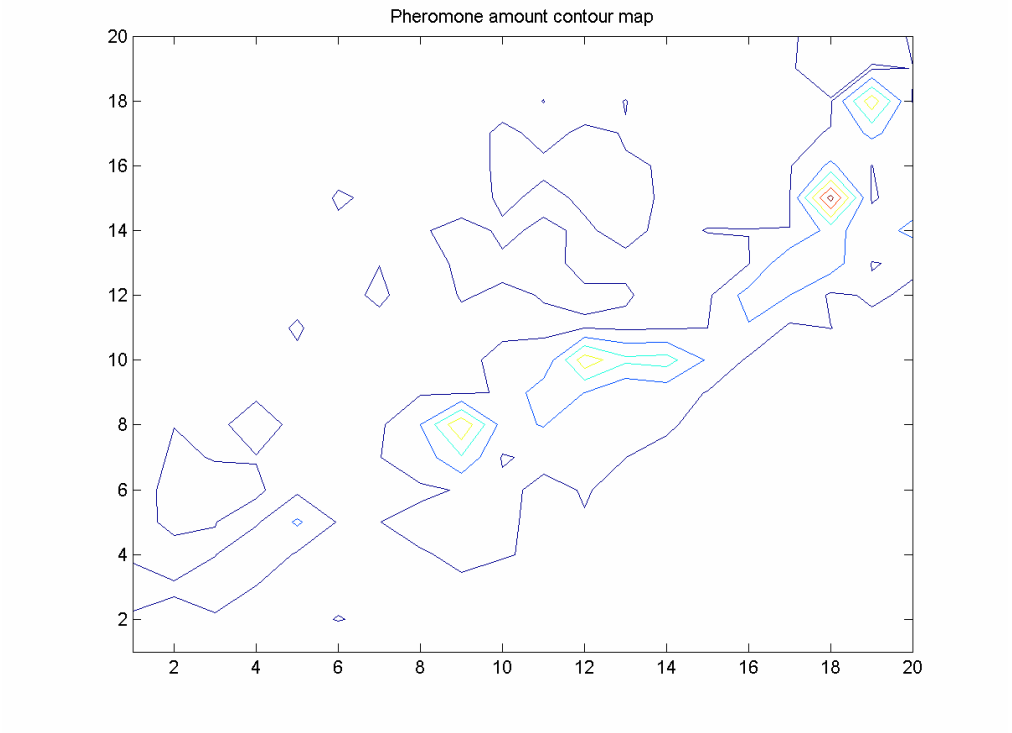


Fig 5.2.8: Map pheromone contour map (red > yellow > blue)

Discussion of the experiment 1 and 2:

It is noted that a significant difference exists between the maximum value and the minimum value in the pheromone map employing in which elitism is employed while the pheromone map for the original ACO is more evenly distributed. Also, the variation in the optimum solution settles down as the number of iterations increases as shown in the Fig 5.2.2.

Experiment 3: When an obstacle blocks the optimum path in the map (W/O elitism)

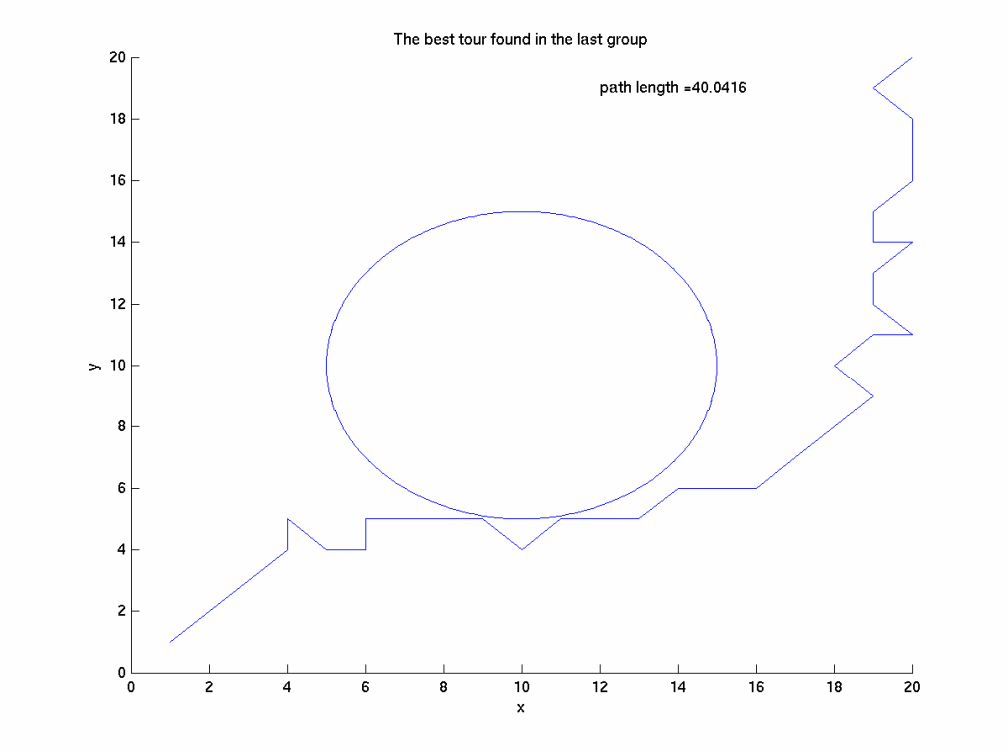


Fig 5.2.9: The best tour found at the last iteration

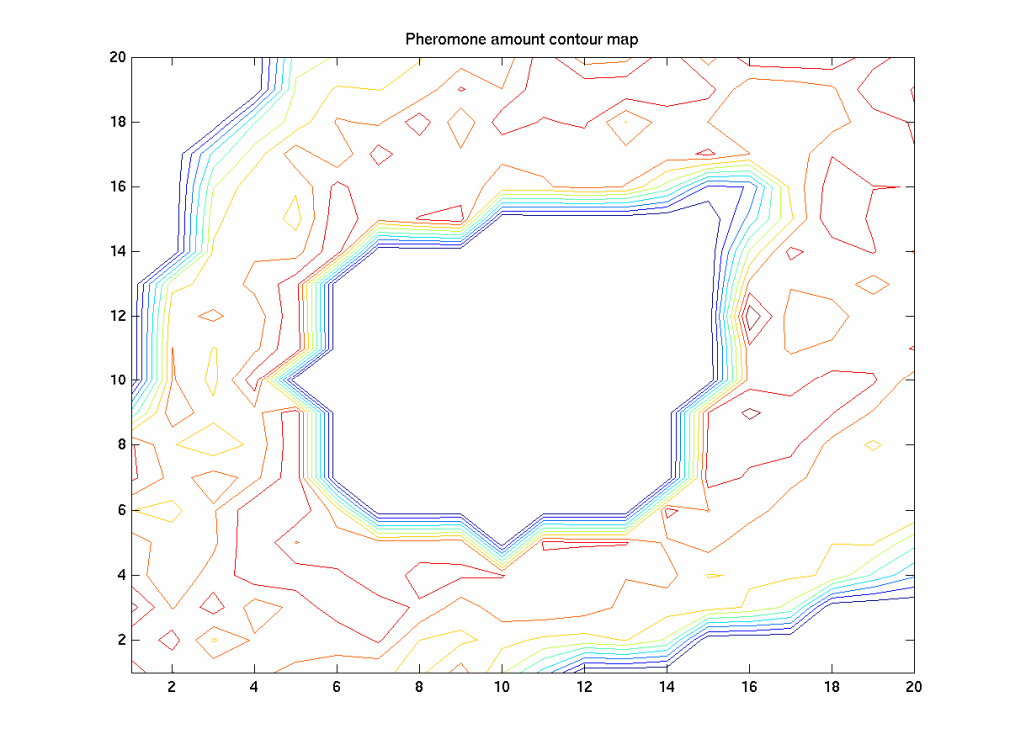
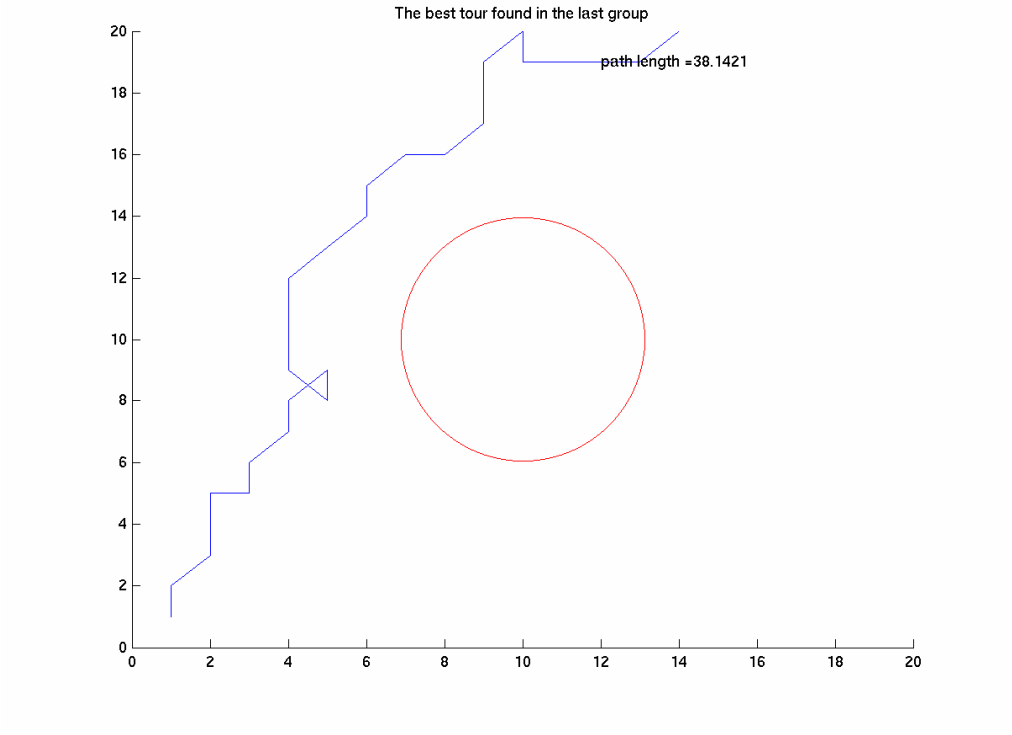


Fig 5.2.10: Deposited pheromone in 2D contour map

Experiment 4: When an obstacle blocks the optimum path in the map (W/ elitism)



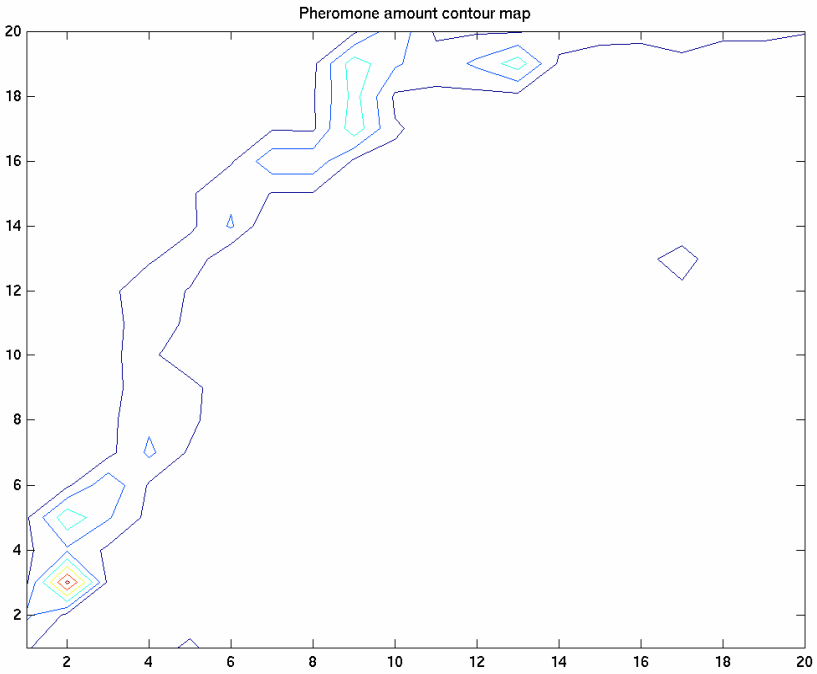


Fig 5.2.11: Deposited pheromone in 2D contour map (red > yellow > blue)

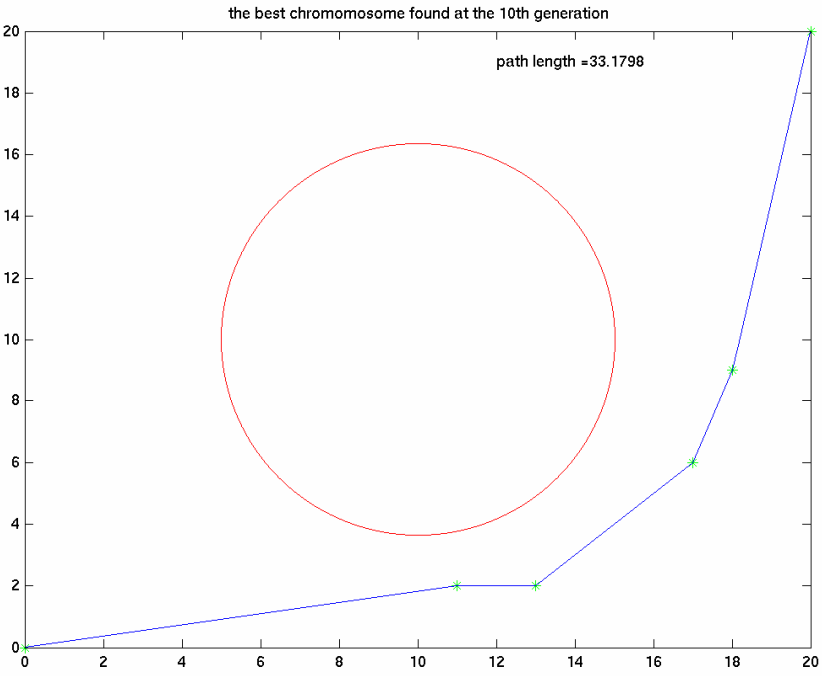


Fig 5.2.12: The best path found by Genetic Algorithm

Discussion of the Experiment 3 and 4:

It is clearly shown that employing the elite strategy reinforces narrow region of the map. This means that the pheromone is mostly distributed on the best path of the map while the other regions on the map receive little or nothing. Fig 5.2.12 is the result achieved from the previous algorithm (GA). Comparing Fig 5.2.12 with Fig 5.2.9, the GA produced a better result in terms of the path distance. This is because the ACO algorithm in this application has to deal with much more uncertainty than in the GA case. In other words, the GA method chooses the best via points (most of the time 3 or 4, points at most 6) while the ACO algorithm makes a decision at every grid point.

Experiment 5: When multiple obstacles block the optimum path on the map (W/ elitism)

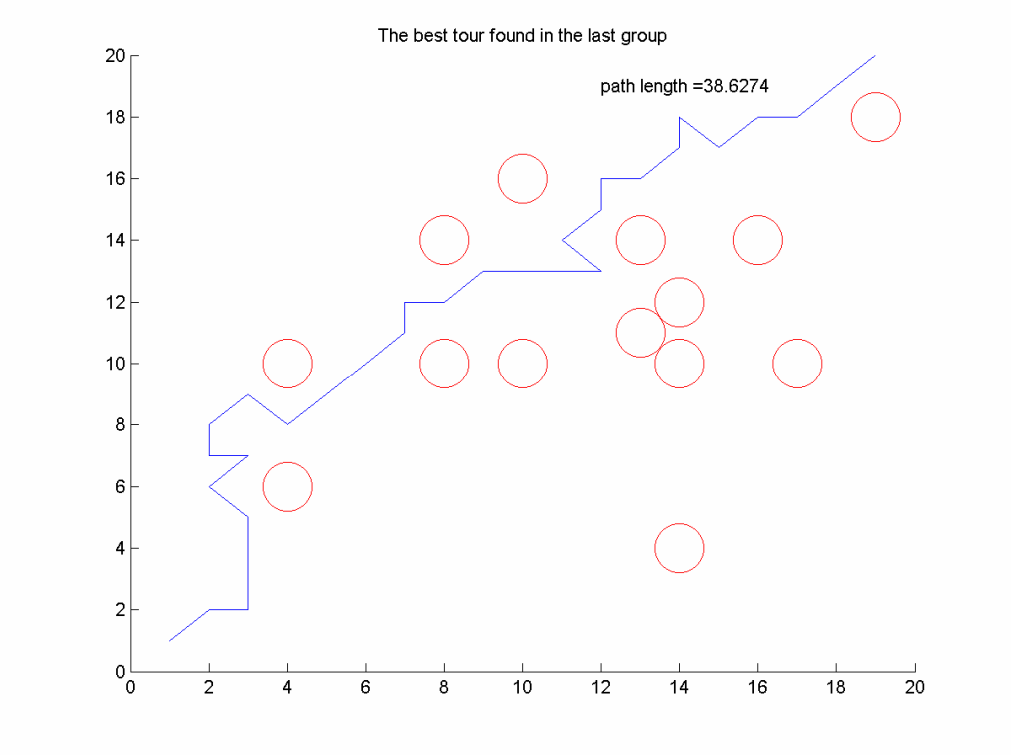


Fig 5.2.13: Best tour found at the last iteration

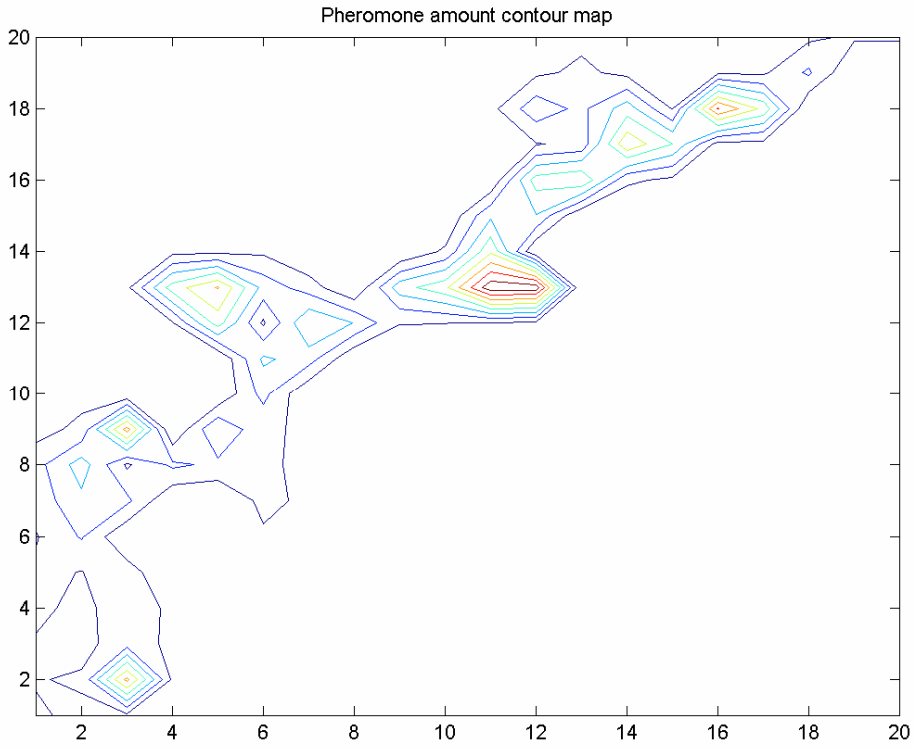


Fig 5.2.14: Map pheromone configuration

Discussion of the Experiment 5: The problem becomes more complicated as the number of obstacles increases. However, since every decision is made at the grid point level, the algorithm always finds an optimized solution which is not necessarily true for the GA method. For example, in the GA case, if the user does not choose a sufficient number of via points, the algorithm will require a significant amount of time to find a best solution or even may fail to find a solution. Thus, the proposed ACO algorithm is better in that sense since the proposed ACO algorithm will find a solution in any case.

Experiment 6: Effect of the number of ants (agents)

The number of ants	10	100	500
Average path distance of 10 trials	45.357	38.6223	36.237

Table 5.2.1

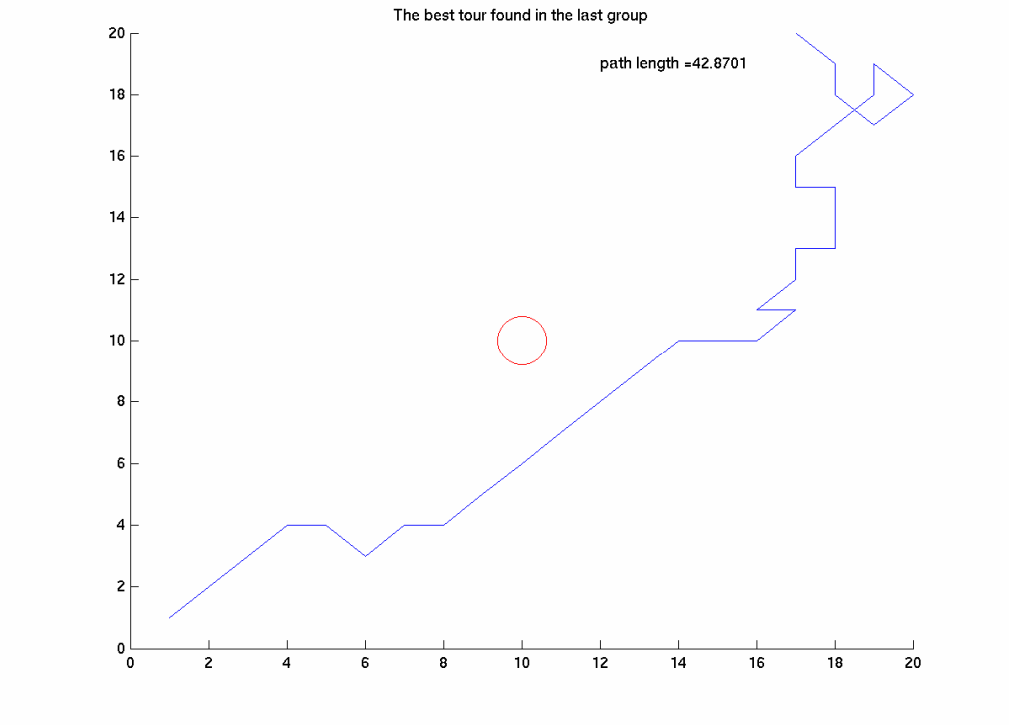


Fig 5.2.15: An example path with the number of ant = 10

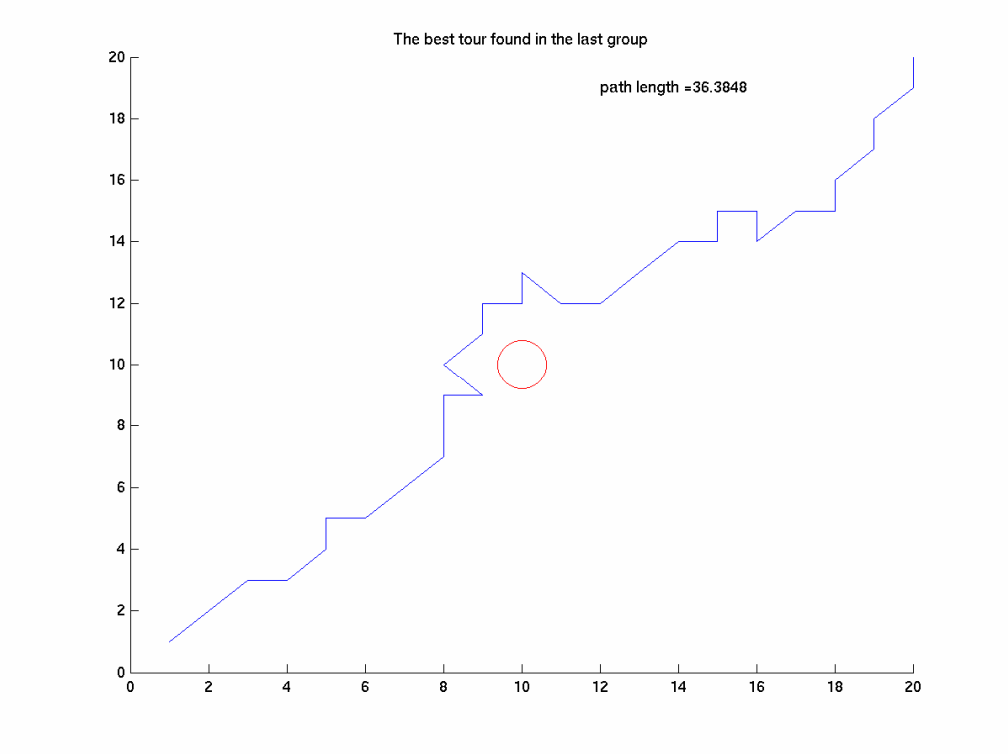


Fig 5.2.16: A path example with the number ant = 100

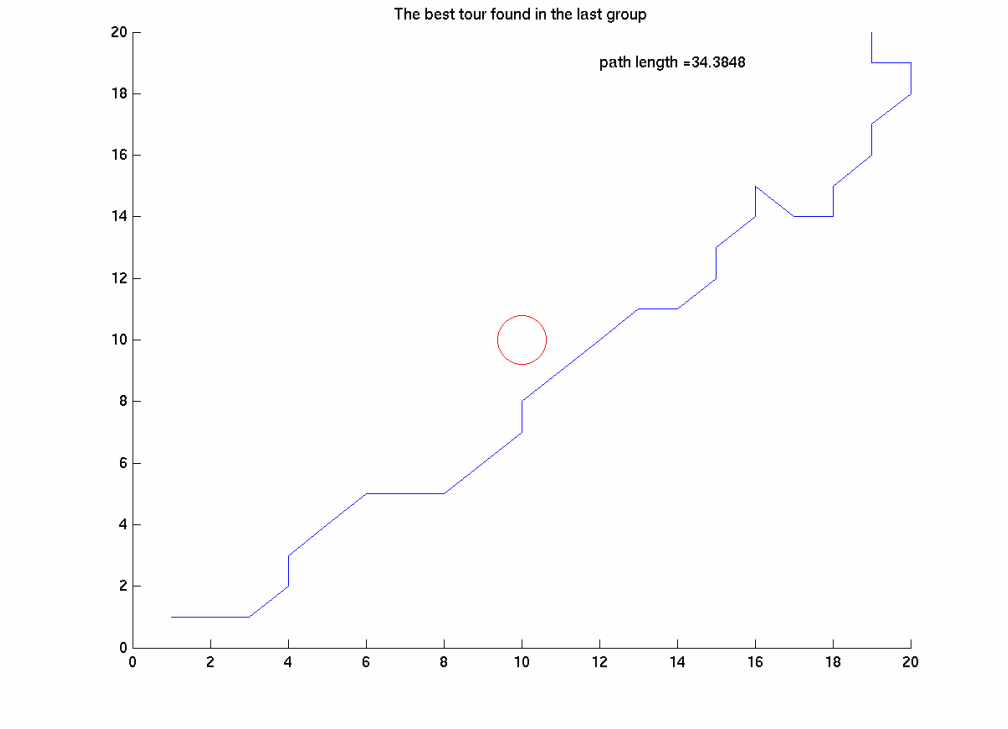


Fig 5.2.17: A path example with the number of ant = 500

Discussion of the experiment 6:

Comparing Fig 5.2.15 and Fig 5.2.16 leads us to the conclusion that a larger size group is better than a smaller size group. If the group size is too small, not only does the number of ants that can reach the destination get smaller but also the amount of pheromone accumulation decreases. However, it is not always true that large number of ants is always the best. As it is shown in table 5.2.1, if the group size is greater than some minimum need, there is not much improvement in performance. Indeed, a large difference of performance exists between 10 ant and 100 ants while not much improvement presents between 100 ants and 500 ants

6.0. Conclusion

In this thesis, the results of detailed investigation of the GA and ACO algorithms being applied to a path optimization problem have been presented. It has been demonstrated that both approaches have a great potential of being a solution to the proposed problem. In addition, it has been discovered that employing elitism led the algorithms to find a solution more easily. However, in a majority of cases the GA found a better solution than did the ACO algorithm in terms of the performance. Such a result is due to the fact that in the ACO algorithm each ant only proceeds 1 grid point at a time while this is unnecessary in the GA case. Despite the good performance for both algorithms, some limitations were found. Overcoming these limitations represent a challenge for future research. The three subsections describe some of the limitations of the proposed algorithms along with possible future work

6.1 Limitation in Genetic Algorithm

In order to guarantee obstacle avoidance, it is necessary to impose another constraint on the algorithm. That is, looking for any intersection point created by the path line and the obstacles can create a significant amount of computation. If only a few obstacles block the optimal path, this computation issue can be ignored. However, when a large number of obstacles exist in the map, this algorithm will require a significant amount of computation time. Although it turned out that GA is better than ACO algorithm in terms of finding an optimal path, GA algorithm may pay for that advantage in terms of computation time; this limitation needs to be addressed.

6.2 Limitation in Ant Colony Optimization.

As it is mentioned previously, the ACO algorithm has an advantage over the Genetic algorithm in terms of the algorithm execution time. No matter how many obstacles are present, this algorithm does not devote an in ordinate amount of time in iteration process. However, as it was discussed in the results section, this method is inferior to GA method in the sense that ACO approach takes some unnecessary steps, so that the algorithm does not return the best solution. Furthermore, a global attraction term had to be added to lead ant to reach the goal point. Eliminating this term may cause not only the ant wander around in the map, but also the ant may become stuck at a point.

6.3 Scope for future work

Perhaps, a more effective and reliable solution can be found if one can adapt only positive phases of the two algorithms. In other words, the best approach in the path finding problem would be an algorithm that converges rapidly toward a meaningful solution. Such a topic can be considered as a challenging problem for future work.

Moreover, applying this algorithm to real robot can be a challenging topic. Because, in a real environment, the algorithm has to be able to deal with nonlinear factors such as noise. In addition, the 3D path finding problem is also challenging topic. Such an application may be found more easily than some 2D application; as for example, aircraft, underwater vehicles, and so forth.

Appendix A

MATLAB code

```
clc
clear all

for    aa=1:3
%%%%%%%%%%%%%%%%%%%% Define globally used variables
%%%%%%%%%%%%%%
tic;
pop=100    ;                  %%% size of the population
Lg=[1 2 4 ];                          %%% Define the length of the gene ( the number of via
points)
ra=5;                         %%%% Define the radius of the obstacle
map_size=20;

Sp= [0; 0]                    %%%% Define the starting point
Op=[ 10, 10; 10 , 10]      ;
%Op=[ 3,3; 10,10; 14,10; ; 14 , 4    ; 8, 14; 6, 18]'%; 4,10; 16,14; 10, 16]'; %%%%
Define the position of the obstacle
Ep=[map_size; map_size]                    %%%% Define the destination point

%%%%%%%%%%%% create the possible locations in the map
%%%%%%%%%%%%%
pos=[];
for k=1:map_size+1
       for l=1:map_size+1
              pos_t=[l-1; k-1];
              pos=[pos pos_t];
       end
end
%
% figure(1)
%    plot( Op(1,:), Op(2,:), 'ro', 'markersize', ra*28 )
%    hold on
%    ezplot( '(x-10)^2 + (y-10)^2 - 2.25', [0 20 ])
%    axis( [0 20 0 20] )


%%%%%%%% Make sure that the via points is not the staring point and goal
%%%%%%%% point. Furthermore, it should not be lying on the obstacles.
no_via_ind1= find(    ( pos(1,:) ==Sp(1) & pos(2,:)== Sp(2) )    |      ( pos(1,:)== Ep(1) &
pos(2,:)==Ep(2)    )    );
no_via_ind2=[];
```

```
for i=1: size (Op,2)
      no_via_ind2=[no_via_ind2    find( (pos(1,:) - Op(1,i) ).^2 + ( pos(2,:) -Op(2,i) ).^2
<= ra^2 ) ];    %% find points which are inside of the obstacle circle
end

no_via = union ( no_via_ind1, no_via_ind2 );
pos( :, no_via )=[];

% figure(2)
% plot( pos(1,:), pos(2,:), 'ro' );
% title( 'possible via points' );

[ Xch, Ych      ]= gen_ch ( pop , Lg(aa), pos, Sp, Op, Ep, ra);    %%% generate possible
solutions to start the algorithm

      ch_dist=[];
      for i=2:Lg(aa)+2
            ch_dist(:,i-1)= sqrt(    ( Xch(:,i-1)- Xch(:,i) ).^2 + ( Ych(:, i-1)-    Ych(:, i ) ).^2
);
      end

      ch_fitness = sum( ch_dist')';
      best_ch_ind=    find( ch_fitness == min( ch_fitness) );

      ch_order = [ [1:length(ch_fitness)']'    ch_fitness] ;
      ch_order = sortrows( ch_order, 2)          %%%% reoder the chromosomes in terms
of their fitness

%%%%%%%%%%% iteration begions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for generation = 1:500
      generation


      %%%%%%%%%%%%%%% divide the chromosomes into 10 groups
%%%%%%%%%%%%%%%%%%5

      %%%%%%%%%%%%%%% Do crossover and mutation for the 10 groups
%%%%%%%%%%%%
      %%%%%%%%%%%%%% Variate Pc and Pm by the fitness of the group
      %%%%%%%%%%%%%% Good fitness group <== low Pc and Pm
      %%%%%%%%%%%%% Bad fitness group <== high Pc and Pm
      Pc=.2; %[ .05,    .1, .2, .2, .3, .3, .35, .35, .4, .5 ];
      Pm=.2; %[ .001, .01, .05, .05, .1, .15, .2, .2, .2, .5];

      %for i=1:10
```

```
      elite= ch_order(1,:);
     Xe=Xch( elite(:,1), : );
     Ye=Ych( elite(:,1), :);

      order=ch_order(2:end,:);

      %order= ch_order( 1 + 10*(i-1) : 10*i, :) ;
      bit=round( rand*(Lg(aa)-1) );

      [ Xchild1, Ychild1 ] = crossover ( order, Xch, Ych,    Pc , bit, Op, ra, Lg(aa));
%%% Perform the X over
      chn_size=0;
      chn_size= pop- size(Xchild1,1);
      [ Xchdn2, Ychdn2      ]= gen_ch ( chn_size, Lg(aa), pos, Sp, Op, Ep, ra ) ;      %%%
Make up the current population up to the number of the staring population
      Xchild1=[ Xchild1; Xchdn2];
      Ychild1=[ Ychild1; Ychdn2];
      [ Xchild2, Ychild2 ] = mutation( Xchild1, Ychild1, pos, Pm, Op, ra );

      %end

      %%%%%%%%%%%%%%%%% Update all chromosomes
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      Xchn=    Xchild2; %Xch_new;
      Ychn=    Ychild2; %Ych_new;

      %%%%%%%%%%%%% In case that the size of chromosome is short... make up
      %%%%%%%%%%%%% chromosome..
      chn_size=0;
      chn_size= pop- size(Xchn,1);
      [ Xchn2, Ychn2      ]= gen_ch ( chn_size, Lg(aa), pos, Sp, Op, Ep, ra) ;

      Xch_t= [Xchn ; Xchn2];, Xch(2:end,:)= Xch_t(1:pop-1,:);, Xch(1,:)=Xe
      Ych_t= [Ychn ; Ychn2];, Ych(2:end,:)= Ych_t(1:pop-1,:);    Ych(1,:)=Ye
      %%%%%%%%%%%%%%%%%%%% chromosome re evaluation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      ch_dist=[];
      for i=2:Lg(aa)+2
            ch_dist(:,i-1)= sqrt(    ( Xch(:,i-1)- Xch(:,i) ).^2 + ( Ych(:, i-1)-    Ych(:, i ) ).^2
);
      end

      ch_fitness = sum( ch_dist')';
      best_ch_ind=    find( ch_fitness == min( ch_fitness) );
```

```
    ch_order = [ [1:length(ch_fitness)']'    ch_fitness] ;
    ch_order = sortrows( ch_order, 2)          %%%% reoder the chromosomes in terms
of their fitness



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%

    if generation==10
        Xch10=Xch;
        Ych10=Ych;
        order10=order;
        best10= best_ch_ind;

        figure(1)
        plot( Xch10(best10(1),:), Ych10(best10(1),:), 'g*' )
        hold on
        line( Xch10(best10(1),:), Ych10(best10(1),:) )
        plot( Op(1,:), Op(2,:), 'ro', 'markersize', ra*27 )
        title('the best chromosome found at the 10th generation')
        text(12,19, ['path length =', num2str( order10(1,2) )] )
        hold off

       print ('-dtiff',[ 'ob3_10_',num2str(aa)])
    end

    if generation==50
        Xch50=Xch;
        Ych50=Ych;
        order50=ch_order;
        best50=best_ch_ind;

%           figure(2)
%           %plot( Xch50(best50(1),:), Ych50(best50(1),:), 'g*' )
%           hold on
%           line( Xch(best50(1),:), Ych(best50(1),:) )
%           plot( Op(1,:), Op(2,:), 'ro', 'markersize', ra*28 )
%           hold off
%           title('the best chromosome found at the 50th generation')
% text(12,19, ['path length =', num2str( ch_order50(1,2) )] )
%           print -dtiff fig_50
    end

    if generation==100
        Xch100=Xch;
```

```
            Ych100=Ych;
            order100=ch_order;
            best100=best_ch_ind;

%               figure(2)
%               plot( Xch100(best100(1),:), Ych100(best100(1),:), 'g*' )
%               hold on
%               line( Xch100(best100(1),:), Ych100(best100(1),:) )
%               plot( Op(1,:), Op(2,:), 'ro', 'markersize', ra*28 )
%               hold off
%               title('the best chromosome found at the 50th generation')
%
% text(12,19, ['path length =', num2str( order10(1,2) )] )
%               print -dtiff fig_100h4
      end


end

toc;

figure(4)
plot( Xch(best_ch_ind(1),:), Ych(best_ch_ind(1),:), 'g*' )
hold on
line( Xch(best_ch_ind(1),:), Ych(best_ch_ind(1),:) )
plot( Op(1,:), Op(2,:), 'ro', 'markersize', ra*27)
title(' the best chromomosome found at the 100th generation' )
text(12,19, ['path length =', num2str( ch_order(1,2) )] )
hold off
%
   print ('-dtiff',[ 'ob3_100_',num2str(aa)])

end %of aa



function [ Xch, Ych      ]= gen_ch    ( N , Lg, pos, Sp, Op, Ep, ra)

i=1;
Xch=zeros(N, Lg+2);
Ych=zeros(N, Lg+2);
   while i < N+1

      chs_ind=ceil( rand(1,Lg)*length(pos) );    %%% pop size of    possible combinations
      ch_t=[ Sp pos(:,chs_ind) Ep ] ;
```

```
        [ test_result ]=test_chromosome( ch_t(1,:), ch_t(2,:) , Op, ra )    ;

        if test_result == 1
              Xch (i,:)= ch_t(1,:);
              Ych (i,:)= ch_t(2,:);
              i=i+1;
        end
  end

end




function [ test_result ]=test_chromosome( Xc, Yc, Op, ra )

clear test_result;

v_fit=zeros( size(Xc,2), length(Op),size(Xc,1)    );

for p=1: size(Xc,1)
      for q=2: size( Xc,2 )
            for r=1: length( Op )
                  x2= Xc(p,q);, y2=Yc(p,q);, x1=Xc(p,q-1);, y1=Yc(p,q-1);
                  x0= Op(1,r);, y0=Op(2,r);

                  %%%%%%%%%% if x1=x2 then find Ysol first %%%%%
                  if x1==x2
                        pol1=1;
                        pol2= -2*y0;
                        pol3= y0^2+x1^2-2*x0*x1+x0^2-ra^2;

                        Ysol= roots( [ pol1 pol2 pol3] ) ;
                        Xsol= [x1; x2];

                        %%%%% else represent in polynoimal form.. i.e) descending order
of X
                  else
                        m=(y2-y1)/(x2-x1);
                        c=y1-m*x1;
                        pol1= m^2+1;
                        pol2= (2*m*c-2*y0*m-2*x0);
                        pol3= c^2-2*y0*c+y0^2+x0^2-ra^2;

                        Xsol= roots( [ pol1 pol2 pol3] ) ;
```

```
                Ysol= m*Xsol + c;
            end
            sol= [ Xsol, Ysol ] ;
            sol_d = dist( [x1, y1], [Xsol' ; Ysol'] );
            obs_d = dist( [x1, y1], [x0 ; y0 ] );


            if isreal(sol) & ( obs_d > min( sol_d) & obs_d < max( sol_d ) )
                v_fit(q,r,p ) = 1;
            end
        end

        if sum(sum( v_fit(:,:,p) ) ) <1
            test_result(p)=1;
        else
            test_result(p)=0;
        end


    end



end

end
```

```
%%%%%%%%%% This function performs crossover for the give chromosome
%%%%%%%%%%%% Performing 1 point crossover at the 1st bit position %%%%
function [ Xchild, Ychild ] = crossover ( order, Xch, Ych,    Pc , bit, Op, ra, Lg)
%bit=1;
if bit < 0 | bit > 5
    error( 'choose diff bit number' )
    return %break
end

% ch_fitness= order(:,2) - min( order(:,2) )+1;
% fit_sum=sum(ch_fitness )    ;
% p_t= sum( ch_fitness )./ch_fitness ;
% p= p_t/sum(p_t);

ch_fitness_t = order(:,2)
```

```
ch_fitness= 1./ch_fitness_t
ch_fitness/ sum(ch_fitness)

% fit_sum=sum(ch_fitness_t);
% ch_fitness= fit_sum./ ( ch_fitness_t - sqrt( 20^2+20^2 ) ) ;

p= ch_fitness/(sum(ch_fitness) ) ;

Xp= Xch( order(:,1), :);
Yp= Ych( order(:,1), :);

for i=1:length(p)
      q(i)=sum( p(1:i) )    ;
end

%%%%%%%%%%%%%    chromosome clone %%%%%%%%%%%%%%%%%
roulet=rand(1,length(p) );

for i=1:length(p)
qk_t= find( q > roulet(i) ) ;, qk(i)=qk_t(1);
%qk_1_t=find( q < roulet(1) );, qk=qk_1_t(end)
end

Xp=Xp(qk,: );
Yp=Yp(qk,: );
%%%%%%%%%%%%% chromosome crossover %%%%%%%%%%%%%%%%%
roulet=rand(1, length(p) );
vc=find( roulet < Pc );

Xparent = Xp ;, X_head= Xparent(vc, 1:Lg-bit );,    X_tail=Xparent(vc,Lg-bit+1:end) ;
Yparent = Yp ;, Y_head= Yparent(vc, 1:Lg-bit );,    Y_tail=Yparent(vc,Lg-bit+1:end);
Xchild=Xparent;
Ychild=Yparent;

X_tail= flipud(X_tail);
Y_tail= flipud(Y_tail);

Xchild(vc,:)= [ X_head X_tail ];
Ychild(vc,:)= [ Y_head Y_tail ];

[ test_result ]=test_chromosome( Xchild, Ychild, Op, ra )    ;
Xchild( find( test_result == 0 ), : )=[];
Ychild( find( test_result == 0 ), : )=[];

end
```

```
%%%%%%%%%%%%%%%%%% end of the
crossover %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%% This function performs mutation for the given chromosome

function [ Xchild, Ychild ] = mutation( Xp, Yp, pos, Pm , Op, ra)

X_head= Xp(:,1);, X_body= Xp(:, 2:end-1 );, X_tail= Xp(:,end) ;
Y_head= Yp(:,1);, Y_body= Yp(:, 2:end-1 );, Y_tail= Yp(:,end) ;

%%%% The mutation will be performed on the body of the chomosome %%%%

Ppm=rand( size( X_body,1),    size( X_body,2) );
m_ind=find( Ppm < Pm);
child_ind=ceil( rand( size( m_ind) )*length(pos)    );

X_body( m_ind )= pos( 1, child_ind) ;
Y_body( m_ind) = pos( 2, child_ind);

Xchild= [ X_head X_body X_tail];
Ychild= [ Y_head Y_body Y_tail];

[ test_result ]=test_chromosome( Xchild, Ychild, Op, ra );
Xchild( find( test_result == 0 ), : )=[];
Ychild( find( test_result == 0 ), : )=[];
End


/// ACO code here //

clc
clear all
close all


%%%%%%%%%%%%%%%%%% Define globally used
variables %%%%%%%%%%%%%%
tic;

ra=1;                          %%%% Define the radius of the obstacle
map_size=20;

Sp= [1 1]                      %%%% Define the starting point
Op=[10; 10]      ;
%Op=[ 4, 6; 14, 12; 13, 11; 8,10; 10,10; 14,10; ; 14 , 4    ; 8, 14; 13, 14; 4,10; 16,14; 10,
16; 17,10; 19, 18]';
```

```
Ep=[map_size map_size]                    %%%% Define the destination point

%%%%%%%%%%% create the possible locations in the
map %%%%%%%%%%%%%
pos=[];
for k=1:map_size
      for l=1:map_size
            pos_t=[l ; k ];
            pos=[pos pos_t];
      end
end

%%%%%%%% point. Furthermore, it should not be lying on the obstacles.
no_pos_ind1= find(    ( pos(1,:) ==Sp(1) & pos(2,:)== Sp(2) )      );
no_pos_ind2=[];

for i=1: size (Op,2)
      no_pos_ind2=[no_pos_ind2    find( (pos(1,:) - Op(1,i) ).^2 + ( pos(2,:) -Op(2,i) ).^2
<= ra^2 ) ];    %% find points which are inside of the obstacle circle
end

no_pos = union ( no_pos_ind1, no_pos_ind2 );
pos( :, no_pos )=[];

% figure(1)
%    plot( Op(1,:), Op(2,:), 'ro', 'markersize', ra*28 )
%    hold on
%    ezplot( '(x-10)^2 + (y-10)^2 - 2.25', [0 20 ])
%    axis( [0 20 0 20] )
% hold off
a=.1;, Q=5;                  %%%%%% pheromone decay factor
ph_l=zeros(20,20);    %%%%%% Initialize local pheromone map
%tij=(1-p)*tij+dtij      %%%%% Update the pheromone
max_step=50
ant_number=500;

for iteration=1:50

      acc=1 ;
      pp_hist=[];
      path_length=[];, non_emp_ind=[];, dT=[];

      for k=1:ant_number
            k
            cp=Sp;                  %%%%%% cp is the current ant position
            pp=[];                      %%%%%% pp is history of the ant positions
```

```
%ph_g=Ep;                    %%%%%% Global pheromone

lm=0;
dm=0;

for step=1:max_step%100%%%%%% repeat following process until the kth ant reach the goal position

    xp=cp(1);
    yp=cp(2);
    pp=[pp ; cp ] ;                    %%%% pp is the past path points
    nps=[ xp+1 yp; xp+1 yp+1;xp yp+1; xp-1 yp+1;xp-1 yp; xp-1 yp-1; xp yp-1; xp+1 yp-1 ]; %%% np neighbor points with respect to the current point
    nps=intersect(nps, pos' , 'rows' );
    %mem_ind=ismember( intersect(nps, pos' , 'rows' ), nps, 'rows' ) ;
    %nps( find( ~mem_ind ) ) = [];      %%%% The ant's next location must be the one of the possible postions in the map

    %%%%%%%%%%%%%% Determine the ant's next position by probabilty based on the pheromone %%%%%%%%%%%%

    clear t;
    for i=1:size(nps,1)
        t(i)=ph_l( nps(i,1), nps(i,2) );
    end

    zindt= ismember( nps, intersect( nps, pp, 'rows' ), 'rows') ;
    zind= find( zindt);

    head_vector= nps - repmat( cp, [size(nps,1) , 1] );
    ph_g=dot( head_vector, repmat( (Ep/norm(Ep) ), [size(nps,1),1 ] ), 2 )    ;
    ph_g( find(ph_g <=0 ) )=0 ;
   % ph_g=    ph_g+abs( min( ph_g ) )    ;%/    (dist( cp, Ep' ) + 1 )    ;
    %ph_g=0.01;

    Ps= t+ ph_g' ;
    Ps(zind) = 0.00001;                    %%%% The ant's next movement should not be one of the past positions unless all neighbor positions belong to the past position
    P= Ps/sum(Ps) ;                        %%%% Probabilities of the ant's next movement is dependent on the amount of pheromone in the next positions

    q=[];, q_t=0;
    for i=1:length(P)
        q_t=q_t+P(i);
```

```
                q=[q q_t];
            end
            roullete=rand ;
            qk=find(q > roullete)    ;              %%% wheel of the fortune... :)
            np_ind=qk(1) ;

            np=nps(np_ind,:);                       %%%% np is the finally chosen next
position

            %%%%%%%%%%%%%%%% If the ant moved to its diagonal direction
then
            %%%%%%%%%%%%%%%% increment dm. Otherwise increment lm;
            if dist(cp, np') > 1
                dm= dm +1;
            else
                lm= lm +1;
            end

            cp=np;

            if cp == [20 20];      %%% the ant reached at the destination
                pp=[pp ; cp];
                break
            end

        end    %%% end of the max number steps
        %

        Lk= dm*sqrt(2) + lm;
        %         figure(1)
        %         line( pp(:,1), pp(:,2) )
        if step < max_step
            path_length( k ) = Lk;
            dT(k)= Q/(Lk-20*sqrt(2)) ;
            pp_hist{k,1}=pp ;
            pp_hist{k,2}=dT(k) ;
            non_emp_ind(acc)=k ;
            acc=acc+1;
        else
            path_length( k ) = Inf;
            dT(k)= 0;
        end


    end %%% end of a group travel
    elite_ind=find( path_length == min( path_length) );
```

```
        best_Lk(iteration)=path_length( elite_ind(1));

        %%%%%%% Update local pheromone
map %%%%%%%%%%%%%%%%%%%%%
        for n=1:length( non_emp_ind )
                c_pp= cell2mat( pp_hist( non_emp_ind(n) ) );
                c_dT= dT( non_emp_ind(n));
                for m=1:size(c_pp,1)
                        indi=sub2ind(size(ph_l), c_pp(m,1), c_pp(m,2) ) ;
                        ph_l(indi)= ph_l(indi)* (1- a) + c_dT;
                end
        end

end %%% end of the iteration
          best_path= pp_hist{elite_ind(1),1} ;
          Lk= dm*sqrt(2) + lm;
          figure(2)
          line( best_path(:,1), best_path(:,2)    )
            text(12,19, ['path length =', num2str( best_Lk(iteration)    )] )
                  hold on
            %ezplot( '(x-10)^2 + (y-10)^2 - 25', [0 20 ])
            plot( Op(1,:), Op(2,:), 'ro', 'markersize', ra*28 )
          title('The best tour found in the last group')
          figure(3)
          plot( 1:iteration, best_Lk(1:iteration))
          title('Best tour at the each iteration')
          xlabel('iteration')
          ylabel('Best tour found by an agent')
          figure(4)

            bar3(ph_l)
            title( ' Map pheromone amount ' )

            figure(5)
            contour(ph_l)
            title('Pheromone amount contour map' )
```

References:

[1] Xin Yao, “Evolutionary computation theory and applications,” World Scientific 1999, pg 2~3.

[2] Laura F. Landweber and Erik Winfree, “Evolution as Computation,” Springer 1998, pg 95.

[3] Amit Konar, “Computational Intelligence principles, techniques and applications,” Springer 05 pg 339.

[4] D. Huh, J, Park, U. Huh, H. Kim, “Path Planning and Navigation for Autonomus Mobile Robot,” IECON 02 IEEE annual conference.

[5] S. Lee and G. Kararas, “Collision-Free Path Planning with Neural Networks,” 1997 IEEE Interational Conference on Robotics and Automation.

[6] N.G. Bourbakis, D. Goladman, R.Fematt, I. Vlachavas, L.H. Tsoukalas. “Path Planning in a 2-D Known Space using Neural Networks and Skeletonization,” Man and Cybernetics IEEE, 1997.

[7] N. Sadati and J. Taheri. “Genetic Algorithm in Robot Path Planning Problem in Crisp and Fuzzified Environments,” IEEE ICIT02, Bangkok, Thailand

[8] Marco Dorigo and Thomas Stuzle “The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances,” in *New Ideas in Optimization*, D. Corne *et al.*, Eds. McGraw Hill, London, UK, 1999, pp. 11–32.

[9] Marco Dorigo, Mauro Birattari, and Thomas Stutzle “Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique,” IEEE Computational Intelligence Magazine, 2006.

[10] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni “The Ant System: Optimization by a colony of cooperating agents,” IEEE Transaction on Systems, Man, and Cybernetics-Part B, Vol, No.1, 1996, pp.1-13.

[11] Luca M. Gambardella and Marco Dorigo “ Ant-Q: A Reinforcement Learning approach to the traveling salesman problem,” In Proceedings of the Eleventh International Conference on Machine Learning, (Morgan Kaufmann, San Francisco, USA, 1996) 622-627.

[12] Watkins C.J.C.H. “Learning with delayed rewards,” Ph. D. dissertation, Psychology Department, University of Cambridge, England.

[13] Thomas Stuzle and Holger H. Hoos, “MAX-MIN Ant System,” Future Gen. Comput. Syst. 2000, vol. 16, no. 8 pp. 889-914

[14] Marco Dorigo, Mauro Birattari, and Thomas Stutzle, “Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique,” IRIDIA- TECHICAL REPORT SERIES: TR/IRIDIA/2006-023.

[15] Christian Blum, and Marco Dorigo, “The Hyper-Cube Framework for Ant Colony Optimization,” IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B: CYBERNETICS, VOL. 34, NO.2, APRIL 2004

[16] Amit Konar, “Comutational Intelligence Principles, Techniques and Applications,” Springer, Springer Berlin Heidelberg NewYork, 2005, pg 18.

[17] M. Youself Ibrahim and Allwyn Fernandes, “Study on Mobile Robot Navigation Techniques,” IEEE ICIT, Tunisia, December 8-10, 2004.

[18] James Kennedy and Russell Eberhart, “Particle Swarm Optimization,” Neural Networks, 1995 Proceedings, IEEE International Conference.