Path Planning using Bacterial Foraging Algorithms

Major Project submitted in partial fulfilment of the requirements for the award of degree of

Master of Technology

in

Information Systems

Submitted By:

Hari Nair

(2K11/ISY/06)

Under the Guidance of:

Prof. O. P. Verma

(HOD, IT Department)

Department of Information Technology

Delhi Technological University

Bawana Road, Delhi – 110042 (2009-2011)

CERTIFICATE

This is to certify that Mr. Hari Nair (2K11/ISY/06) has carried out the major project titled “Path Planning using Bacterial Foraging Algorithms” as a partial requirement for the award of Master of Technology degree in Information Systems by Delhi Technological University.

The major project is a bona fide piece of work carried out and completed under my supervision and guidance during the academic session 2011-2013. The matter contained in this report has not been submitted elsewhere for the award of any other degree.

(Project Guide)

Prof. O. P. Verma

Head of Department

Department of Information Technology

Delhi Technological University

Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

I express my gratitude to my major project guide Prof. O. P. Verma, HOD, IT Dept., Delhi Technological University, for the valuable support and guidance he provided in making this major project. It is my pleasure to record my sincere thanks to my respected guide for his constructive criticism and insight without which the project would not have shaped as it has.

I humbly extend my words of gratitude to other faculty members of this department for providing their valuable help and time whenever it was required.

Hari Nair

Roll No. 2K11/ISY/06

M.Tech (Information Systems)

E-mail: harinair304@gmail.com

Abstract

Path planning has been the subject of research for many years in the field of robotics and navigation. This thesis presents a study of various approaches that utilizes the concept of evolutionary algorithm to solve the path planning problem. Bacterial foraging, a popular evolutionary algorithm has been used to compute paths. This is done by treating the entire working configuration space as a set of discrete points. Then from out of these free configuration points are extracted. These are those points that do not lie in any obstacle. These points are then inputted into the bacterial foraging algorithm. The bacterial foraging algorithm then outputs an optimal free configuration point. This serves as the next point in the path being constructed. Variations of this basic process have been described in this thesis. Results show that the method produces path in an acceptable time window.

Finally another approach has been proposed that also produces optimal paths, but with greater efficiency and accuracy. This method first creates a visibility graph of the given working scenario. Then the A* algorithm, which is a heuristic search algorithm, is applied to the above computed graph to obtain regions from where the optimal path would go. Now free configuration points are generated only around these regions. Then bacterial foraging is applied to each of these regions to work out the optimal free configuration points. The optimal path is then finally constructed by joining all these optimal free configuration points.

Table of Contents

List of Figures and Tables

CHAPTER 1: PATH PLANNING

1.1 Background

Path planning is a concept that finds use in a diverse set of fields ranging from molecular biology to artificial intelligence in computer gaming. The essence of the path planning problem is that given a configuration of obstacles, one has to generate a collision free path from the start to the goal. There is no fixed definition of optimality in path planning, because the criterion of optimality depends on the domain in which path planning is used. In some fields where distance is the major concern the shortest path would be considered the optimal one, while in other cases such as route planning on rough terrain safety would be paramount thus a smoother and safer path would be optimal.

1.2 Conventional Approaches

A variety of solutions have been proposed to the path planning algorithm. One of the most common ones is the grid based approach[1-4]. The grid based approach was first given by W.E. Howden. It superimposes a grid onto the configuration. Motion to another grid is possible if and only if there is no obstacle between the two grids. The grid size depends upon the active area of the mobile robot. Search algorithms can then be applied to this grid to obtain a path from start to goal. It is a fairly simplistic approach, but if it is handed inappropriately, the calculations involved become too complex and a suitable path may not be found. Thus many improvements have been made to this approach. Guo shuai, Dai Leyin and Ouyang YuPing[3] introduced an improved grid based path planning algorithm. They proposed that instead of having grids of equal size, the grid size should be a variable of the obstacle size, the grid size being the size of the smallest rectangle that would completely enclose the obstacle. This would result in a network of unequal size grids. The path calculated using this network grid will be better than the path outputted by the conventional grid as the distance between the grid corners and the obstacle contour is less. Work has also been done in the area of improving the quality of the path outputted by the grid map. Masatomo Kanehara, Satoshi Kagami, James.J Kuffner, Simon Thompson & Hiroshi Mizoguhi[1] gave an approach that modified the grid based A* results and pushed the path away from the obstacles. The technique employed a shortening algorithm and a smoothening algorithm based on clothoidal curvature that took into account the initial orientation of the robot

The potential field approach[5-8] parameterises each point in the configuration on the basis of the combined effect of the attraction from the goal and repulsion from obstacles. This method was first given by Charles H. Warren[7]. In this method each obstacle is mapped to a potential function which is somewhat similar to electrostatic potential. The free space is then characterised by topography in the form of minimum potential valleys through which a path was generated. The work was further extended by Yong K. Hwang, and Narendra Ahuja[5]. Here Path planning is performed at two levels. The global planner first computes the robot’s path from the minimum potential valleys and its orientation along the path that minimizes a heuristic function composed of path length and the chance of collision. The local planner then derives a collision free path by modifying the path and orientations. In case the local planner fails, the global planner selects a new path and orientations which are then passed along to the local planner. The process continues until a path is successfully derived or there are no paths left to examine. This method has low computation requirements but may get trapped in local minima.

The sampling based approach involves sampling the configuration space and creating a roadmap by selecting a set number of points that are not in any obstacle as milestones. A path between two milestones is possible if there is no obstacle between them. The algorithm finishes successfully if a path from start to goal can be obtained using these milestones. A very popular method that is often utilized is Probabilistic Road Mapping. This method was introduced be Karvaki and Latombe[9], and Overmars[21] who arrived upon it working independently. The basic PRM-method works by constructing a roadmap by iteratively sampling configurations randomly from the configuration space. Using a collision-checker, it determines whether a particular configuration belong to the free configuration space or the forbidden configuration space. If a configuration is collision-free, it gets incorporated into the roadmap in the form of a node. Gradually attempts are made to connect this node with the nodes that are already present in the roadmap. To save time only nodes that are close to each other (i.e. at a distance *d* or the *k* nearest neighbours) are considered as candidates for the connection process. The set of nodes to which a connection is attempted is called the *neighbour set* of the current node. Connections between nodes are tried using a *local planner*, which is a simple planner that is allowed to fail on all but the simplest queries. In the basic PRM implementation, the local planner simply tries to connect two configurations by a straight line through the configuration space. The local planner gives a successful output when the straight line thus generated is collision free, which it determine by recursively applying the same process to intermediate configurations, up to a predefine resolution. An edge is added to the roadmap in case such a connection succeeds between the two nodes. If a connection has been tried to all nodes of the neighbour set, a new node is sampled, and the process repeats. Once the entire roadmap has been fully created, a path is searched for in the roadmap using Dijkstra's algorithm or the A* algorithm.

1.3 The Evolutionary Approach

A number of evolutionary algorithms such as the particle swarm optimization (PSO) [10-12], genetic algorithms (GA) [13-14], bacterial foraging [16-17] have been applied to implement solutions to the path planning problem. Solutions generated by evolutionary algorithms have a number of advantages over the ones generated by more conventional techniques, as listed in [15] and below

- Randomized search can be very efficient and effective in escaping local minima.
- Evolutionary approaches look for a solution in parallel, where individual candidates interact through genetic operators to generate possibly better solutions
- The evolutionary approaches can also easily be implemented on a massively parallel machine to achieve superlinear speed up with the number of processors
- The shortest path optimization criterion is only meaningful for special cases and is usually not an ideal optimization goal. In general, an ideal optimization criterion can be very complex and subjective.
- Since it may be difficult to combine all the important factors into a single evaluation function, a few important criteria can be used in constructing the optimization criteria and generating alternative solutions to the problem

- Can generate alternative paths. Generating alternative solutions may also be important in dynamic environments, where one or more of the alternatives may become infeasible.

Chapter 2: Evolutionary Algorithms

2.1 Introduction

Evolutionary algorithms are a set of population based metaheuristic optimization algorithms that derive their functionality from natural biological evolution. They utilize the Darwinian principle of "Survival of the Fittest" in an iterative fashion to produce better solutions. Each generation produces a new set of approximations by selecting members with appropriate levels of fitness characterised by the problem domain and then breeding them together using operators derived from natural processes. Such a procedure gradually results in the evolution of a population set that is better suited to the problem domain than the original population set, just like in natural adaptation and evolution.

2.2 Working of an Evolutionary Algorithm

Most evolutionary algorithms have computation processes that are functionally equivalent to natural processes like selection, recombination, mutation, migration, locality and neighbourhood. The following figure depicts the working of a simple evolutionary algorithm.

In the beginning a number of individuals are randomly initialized. These form the original population set of the problem. The objective function, i.e. the problem that is to be optimized is evaluated for this population set and the first generation is produced.

If the optimization criterion is not met, the next generation is required to be evolved. This is done by first identifying individuals in the previous population set that possess the required value of fitness. Such individuals serve as parents. Parents are recombined to produce an offspring. To preserve diversity in the solution set there is also a need to mutate the offspring to a small extent. The fitness of the offspring is then computed. It then replaces its parents in the population set thereby creating a new generation. This cycle repeats itself until the optimization criterion is not met.

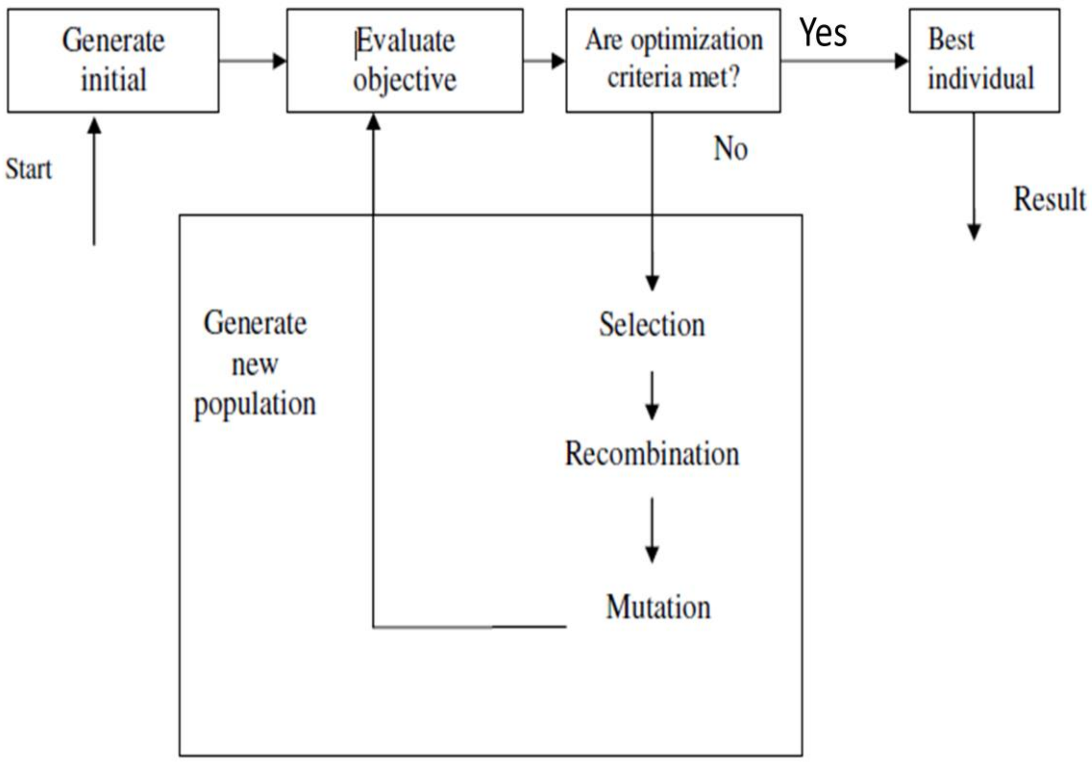


Fig 2.1: Structure of a single population evolutionary algorithm

Single population evolutionary algorithms are powerful and give satisfactory results for a variety of optimization problems. Nevertheless though, even better results can be obtained by introducing multiple sub populations instead of a single population. Each subpopulation evolves akin to the method explained above, but in isolation. After a few generations however, one or more individuals are exchanged between the subpopulation. The following figure depicts the working.

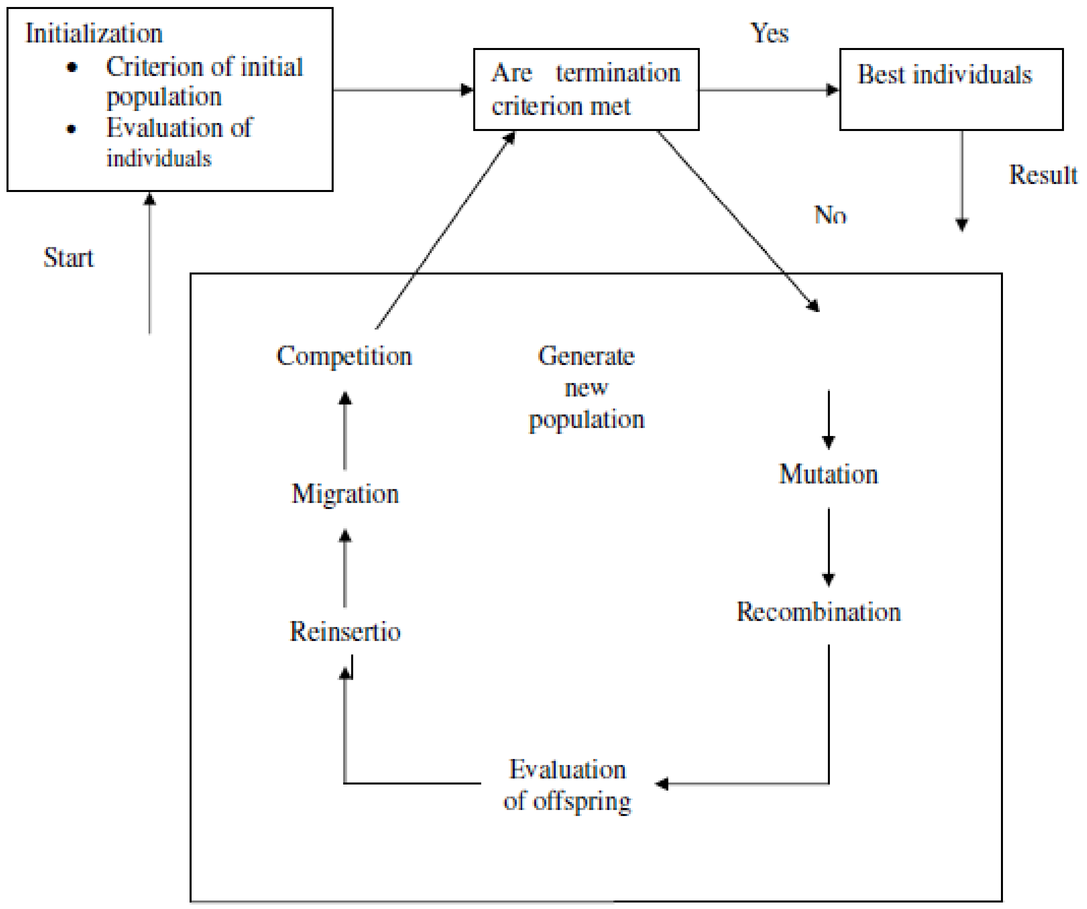


Fig 2.2: Structure of an extended multipopulation evolutionary algorithm

2.3 The Bacterial Foraging Optimization Algorithm

The bacterial foraging optimization algorithm is one of the more recent evolutionary algorithms in use today. It was proposed by Kevin M. Passino[18]. The foraging behaviour of a bacterial life form called as E. Coli, commonly found in the human gut was carefully observed and used as a reference while creating this algorithm. During its life time the swarm passes through 4 distinct stages each of which is programmed into the algorithm. These stages are chemotaxes, swarming, reproduction and elimination-dispersal.

- **Chemotaxes:** The locomotive behaviour of a bacteria in the presence or absence of nutrients is known as chemotaxes. It comprises of two modes of motion, namely tumble and swim. When the flagella rotate clockwise the bacteria engages in a tumble. In such a kind of motion, there is minimum displacement and after completion the bacteria is aligned along a random direction. On the other hand, counter-clockwise rotation allows the bacteria to swim, or to move forward in a particular direction. During its lifetime the bacteria alternates between these modes of motion. In locations with high nutrient concentration the bacteria swims more often than it tumbles and vice versa in areas with less favourable conditions. Thus by continuously swimming and tumbling the bacteria effectively conducts the foraging process. The tumbling process can be mathematically represented as[19]

$$\theta^i(j+1,k,l)=\theta^i(j,k,l)+C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$$

Where θ represents *i*-th bacterium at *j*th chemotactic, *k*-th reproductive and *l*-th elimination-dispersal step, *C*(*i*) is the size of the step taken in the random direction specified by the tumble (run length unit) and Δ indicates a vector in the random direction whose elements lie in [-1, 1].

- **Swarming:** It has been noticed that several bacteria species including E.Coli form stable spatio temporal ring shaped swarms in the presence of a semi solid nutrient medium. The E.Coli cells when stimulated by a high level of succinate release an attractive called aspertate that helps them aggregate in such groups. This cell-cell attraction can be mathematically represented as [19]

$$J_{cc}(\theta, P(j,k,l)) = \sum_{i=1}^{S} J_{cc}(\theta, \theta^{i}(j,k,l))$$
$$= \sum_{i=1}^{S}[-d_{\text{attractant}}\exp(-w_{\text{attractant}}\sum_{m=1}^{p}(\theta_m - \theta_m^i)^2)] + \sum_{i=1}^{S}[h_{\text{repellant}}\exp(-w_{\text{repellant}}\sum_{m=1}^{p}(\theta_m - \theta_m^i)^2)]$$

where p is the dimension of search space, $J_{cc}(\theta, P (i, j,k, l))$ is the cost function that is to be added to the original cost function, $d_{attract}$, $w_{attractt}$, $h_{repellant}$, $w_{repellant}$ are the coefficients which determine the depth and width of attractant and height and width of repellant, which are to be selected properly. S is the total number of bacteria, him is the mth component of ith bacterium position hi.

- **Reproduction:** In this step, the least healthy bacteria die out. The healthiest bacteria (based on the nutrient function) then asexually split into two (without mutation and crossover) at the locations replacing the dead bacteria. This keeps the swarm size constant.

- **Elimination and Dispersal:** Environmental factors such as running water or extreme temperatures may cause a group of bacteria to die out or get transferred to some other location. To incorporate this in the algorithm, on the basis of a very small probability some bacteria are killed off and their replacements are placed at random points in the solution space.

2.4 The Bacterial Foraging Algorithm

Parameter Initialization

p: the dimension of the search space

S: the number of bacteria in the population iterated by counter i

N_c: the number of chemotactic steps iterated by counter j

N_s: the number of swims after tumble iterated by the counter m

N_{re}: the number of reproductive steps iterated by counter k

N_{ed}: the number of elimination dispersal events iterated by the counter ell

p_{ed}: elimination dispersal probability

C(i,k): the size of step taken in a random direction specified by tumble

The Algorithm

[Step 1] Initialize parameters $p, S, N_c, N_s, N_{re}, N_{ed}, P_{ed}$, C(i)(i=1,2,....,S), θ^i

[Step 2] Elimination dispersal loop: l=l+1

[Step 3] Reproduction loop: k=k+1

[Step 4] Chemotaxis loop: j=j+1

[a] For i=1,2...,S take a chemotactic step for bacterium i as follows

[b] Compute fitness function J(i,j,k,l).

[c] Let J_{last} = J(i,j,k,l) to save this value since we may find a better cost via a run

[d] Tumble: generate a random vector Δ(i) with each element $\Delta_m(i)$, m=1,2,...,p a random number on [-1,1].

[e] Move: Let

$$\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$$

```
This results in step of size C(i) in the direction of the tumble for bacterium i
[f] Compute J(i,j+1,k,l)
[g] Swim
   i) Let m=0 (counter for swim length)
   ii) while $m<N_s$ (if have not climber down too long)
      - Let m=m+1
      - If $J(i,j+1,k,l)<J_{last}$ let $J_{last} = J(i,j+1,k,l)$ and let
          $\theta^i(j+1,k,l) = \theta^i(j,k,l) + C(i)\frac{\Delta(i)}{\sqrt{\Delta^T(i)\Delta(i)}}$
        And use this $\theta^i(j+1,k,l)$ to compute new J(i,j+1,k,l)
      - Else, let $m=N_s$. This is the end of the while statement
 [h] Go to the next bacterium (i+1), if i != S (i.e., go to [b] to process the next
 bacterium)
```

[Step 5] if $j<N_c$, go to step 4. In this case continue chemotaxis since the life of the bacteria is not over

[Step 6] Reproduction

[a] For the given *k* and *l*, and for each *i* = 1,2,..., *S* , let

$$J^i_{health} = \sum_{i=1}^{N_c+1} J(i,j,k,l)$$

be the health of the bacterium *i* (a measure of how many nutrients it got over its lifetime and how successful it was at avoiding noxious substances). Sort bacteria and chemotactic parameters *C*(*i*) in order of ascending cost J_{health} (higher cost means lower health).

[b] The S_r bacteria with the highest J_{health} values die and the remaining S_r bacteria with the best values split (this process is performed by the copies that are made are placed at the same location as their parent).

[Step 7] if $k<N_{re}$, go to step 3. In this case we have not reached the number of specified the reproduction steps, so we start the next generation of the chemotactic loop

[Step 8] Elimination-dispersal: For i=1,2,.,S with probability $P_{ed,}$ eliminate and disperse each bacterium (this keeps the number of bacteria in the population constant) To do this, if a bacterium is eliminated simply disperse another one to a random location on the optimization domain. If $l<N_{ed}$, then go to step 2 ; otherwise end.

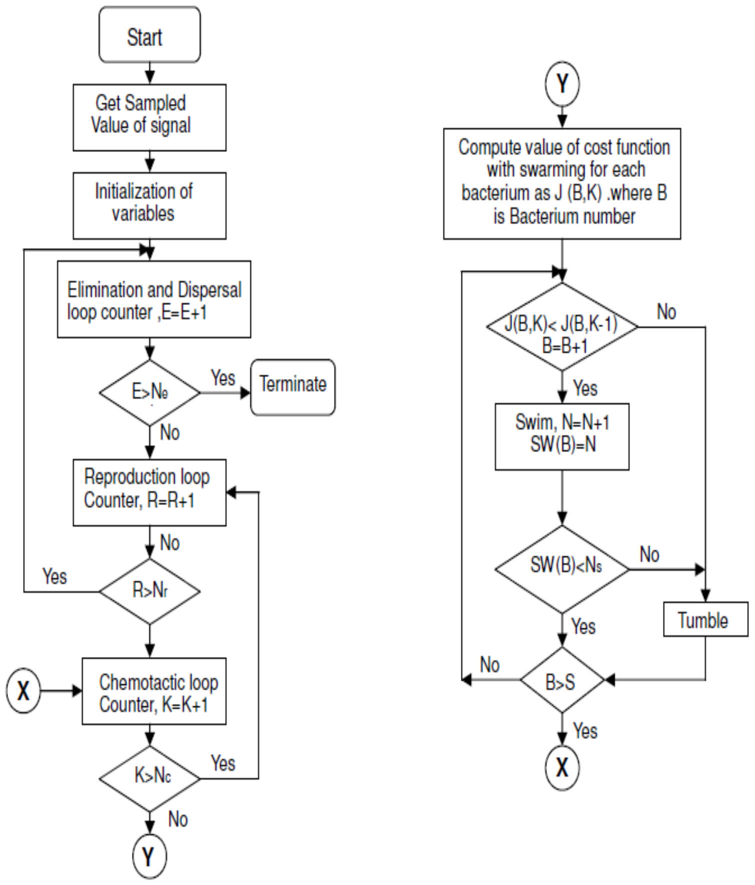

Start
Get Sampled Value of signal
Initialization of variables
Elimination and Dispersal loop counter ,E=E+1
E>Ne
Yes
Terminate
No
Reproduction loop Counter, R=R+1
No
R>Nr
Yes
X
Chemotactic loop Counter, K=K+1
K>Nc
Yes
No
Y
Y
Compute value of cost function with swarming for each bacterium as J (B,K) .where B is Bacterium number
J(B,K)< J(B,K-1) B=B+1
No
Yes
Swim, N=N+1 SW(B)=N
SW(B)<Ns
No
Yes
Tumble
No
B>S
Yes
X

Fig 2.3: Flowchart of the Bacterial Foraging algorithm

2.5 Guidelines for Algorithm Parameter Choices

Size of population 'S': Increasing the size of *S* can significantly increase the computational complexity of the algorithm. However, for larger values of *S*, it is more likely at least some bacteria near an optimum point should be started, and over time, it is then more likely that many bacterium will be in that region, due to either chemotaxis or reproduction.

Length of chemotactic step 'C(i)': If the C(i)values are too large, then if the optimum value lies in a valley with steep edges, the search will tend to jump out of the valley, or it may simply miss possible local minima by swimming through them without stopping. On the other hand, if the C(i) values are too small, convergence can be slow, but if the search finds a local minimum it will typically not deviate too far from it. $c(i)$ can be treated as a type of "step size" for the optimization algorithm.

Chemotactic step ' *Nc* **':** If the size of *Nc* is chosen to be too short, the algorithm will generally rely more on luck and reproduction, and in some cases, it could more easily get trapped in a local minimum (premature convergence). *Ns* creates a bias in the random walk (which would not occur if Ns = 0), with large values tending to bias the walk more in the direction of climbing down the hill.

Reproduction number ' *Nre* **':** If *Nre* is too small, the algorithm may converge prematurely; however, larger values of *Nre* clearly increase computational complexity.

Elimination and dispersal number ' ***Ned*** **':** A low value for Ned dictates that the algorithm will not rely on random elimination-dispersal events to try to find favourable regions. A high value increases computational complexity but allows the bacteria to look in more regions to find good nutrient concentrations. Clearly, if *ped* is large, the algorithm can degrade to random exhaustive search. If, however, it is chosen appropriately, it can help the algorithm jump out of local optima and into a global optimum.

The parameters that define the cell-to-cell attractant functions ' *Jcc* **:** If the attractant width is high and very deep, the cells will have a strong tendency to swarm (they may even avoid going after nutrients and favor swarming). On the other hand, if the attractant width is small and the depth shallow, there will be little tendency to swarm and each cell will search on its own. Social versus independent foraging is then dictated by the balance between the strengths of the cell-to-cell attractant signals and nutrient concentrations. For this work, cell-cell attraction has not been considered.

Chapter 3: The Proposed Approach

3.1 Evolution of the proposed approach

The approached proposed in this thesis is the final product of a process that has included building various versions of the path planning program that utilized different concepts and chains of thought. What follows is a brief description of the builds that preceded the final stable algorithm.

3.1.1 Version 1

In this version a concept of free and feasible configurations was used. Free configuration and feasible configurations basically mean the same thing, which is a set of points that don't lie in any obstacle, the only difference being that the feasible configuration set is a more restricted one, only those set of points that can be directly reachable from a given point; i.e. a straight line that doesn't intersect any obstacle can be drawn between the two. The feasible configuration around the current waypoint is inputted to the bacterial foraging algorithm to produce the next optimal waypoint.

The algorithm of the approach is as follows

Step1. First we create an obstacle map

Step2. Next we specify a start and goal point

Step3. Then movement towards goal is simulated

Step4. Once the sensor detects an obstacle it stops

Step5. Then a free configuration is generated

Step6. From the above a feasible configuration is selected

Step7. This feasible configuration set is used as the solution space for bacterial foraging

Step8. Bacterial foraging outputs an 'optimal' point

Step9. Check optimal point-goal feasibility

Step10. If yes then stop, else continue process from step 6.

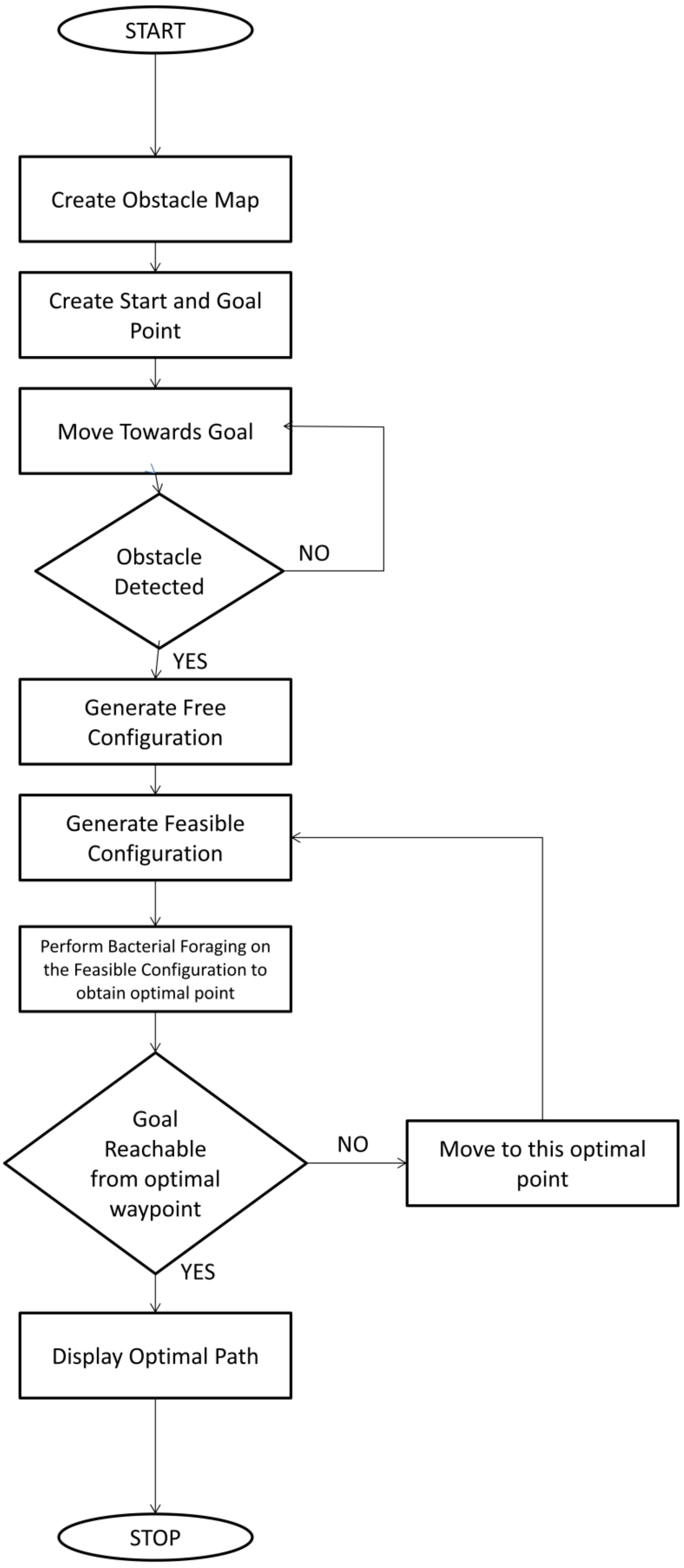

START
Create Obstacle Map
Create Start and Goal Point
Move Towards Goal
Obstacle Detected
NO
YES
Generate Free Configuration
Generate Feasible Configuration
Perform Bacterial Foraging on the Feasible Configuration to obtain optimal point
Goal Reachable from optimal waypoint
NO
Move to this optimal point
YES
Display Optimal Path
STOP

Figure 3.1: Flowchart of the Version 1 algorithm

But the above approach was not able to handle many other obstacle maps. There were some in which it produced non optimal paths and some in which it failed altogether. What follows are some of the unsuccessful trial runs of the algorithms

3.1.2 Version 2

Here the feasible set points are classified before being used as input to the bacterial foraging algorithm. This classification is based on the assumption that a point that is surrounded by a larger number of points is a better candidate for selection than a point with a fewer number of points in its surrounding. Thus using the above definition is point is associated with its relative 'Degree of Freedom'(dof). Then the points whose dof satisfy a certain threshold are input to the bacterial foraging algorithm. Following is a degree of freedom graph of a point that has a 100 point feasible configuration set around it. Now only those points that have an acceptable degree of freedom are inputted to the bacterial foraging algorithm.

A major disadvantage that Version 2 suffered from was that in some cases where an optimal path was possible the approach would compute a suboptimal path just because the waypoint that led to the optimal path did not satisfy the required degree of freedom level.

3.1.3 Version 3

Version 3 was an improved adaptation of Version 1. The basic algorithm was left unchanged. Instead of inputting the entire feasible configuration to the bacterial foraging algorithm, the 50 nearest neighbours to the current optimal waypoint were first calculated and these were sent as input to the bacterial foraging process. This produced noticeably better results than those produced by Version1.

3.1.4 Version 4

In this approach the feasible set of points are classified on the basis of the Euclidean distance from the goal. These points are then divided into seven bands namely VIBGYOR, violet being the points closest to the goal and red containing the points that are farthest. The problem then reduces to identifying the band in which the previous point lies and its two adjacent bands, choosing that band which is closer to the goal as a candidate for the bacterial foraging process. The main advantage of Version 4 was that the time required to compute the optimal path was less compared to its previous versions, as in most cases the workspace was divided into a discrete set of bands rather than an a almost continuous set of points in the previous versions. The results produced by Version 4 were comparable to the ones produced by version 3, though there were cases where it also failed to produce an optimal path

3.2 Proposed Approach

The method described in this section is the final stable algorithm that proposes a solution to the path planning problem. First a visibility graph is constructed that describes the current working scenario. This graph is inputted to the A* algorithm [20] that outputs the vertex regions through which the optimal path will pass. Once these have been identified, we apply the bacterial foraging algorithm to these regions to obtain the optimal path point This method outperforms all the above described versions in terms of both accuracy and efficiency. A comparison has also been made with a popular path planning technique called probabilistic road mapping (PRM) and results show that the proposed approach produces shorter, more optimal paths

Chapter 4: Results

4.1 Basic Information

This chapter illustrates the results of the various approaches described in the previous section. The approach has been implemented using Matlab 7.9.0 on an AMD Athlon 1.70 GHz processor.

Figure 4.1 depicts the simulation. The polygons in red serve as the obstacles around which the path needs to be computed. The dark blue dot represents the start position, while the light blue one the goal.

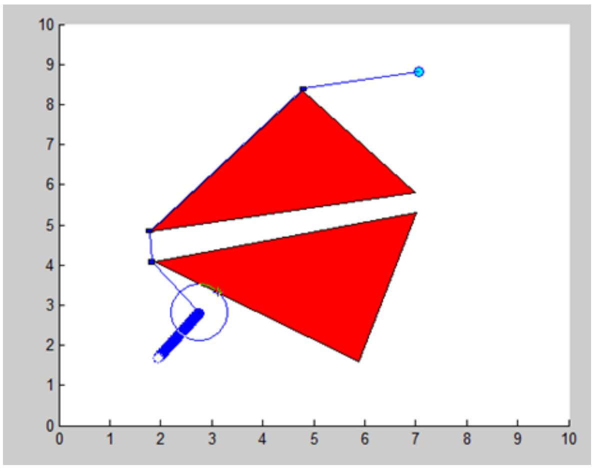


Fig 4.1 The working scenario

4.2 Results of Version 1 Algorithm

Following are the screenshots of some of the successful runs of the Version 1 algorithm. The gray rectangles act as the intermediate waypoints that the path follows. The small green rectangles denote the free configuration points populating the workspace

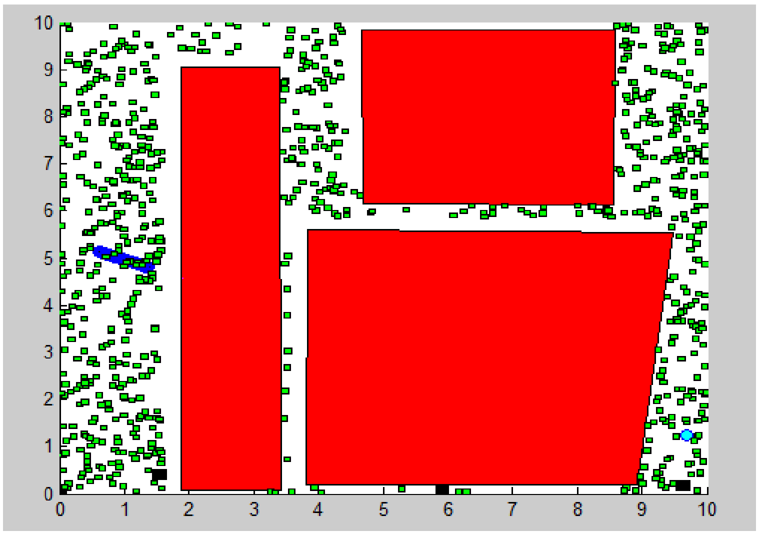


Fig 4.2: Successful run of Version 1 algorithm

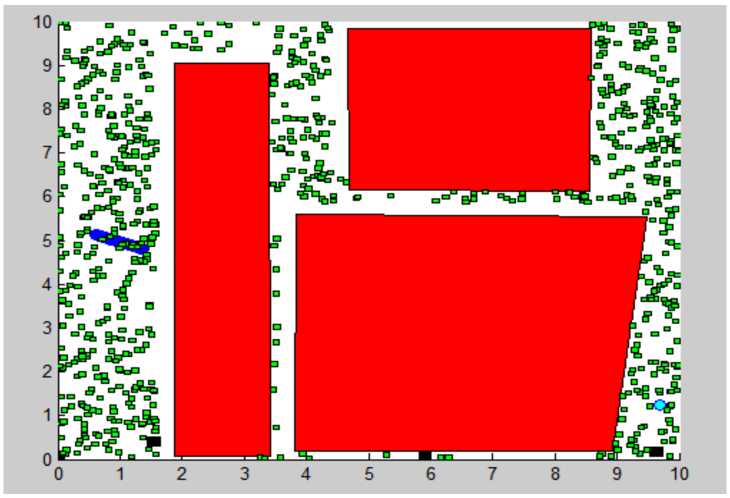


Fig 4.3: Successful run of Version 1 algorithm

But the above approach was not able to handle many other obstacle maps. There were some in which it produced non optimal paths and some in which it failed altogether. What follows is of the unsuccessful trial runs of the algorithm.

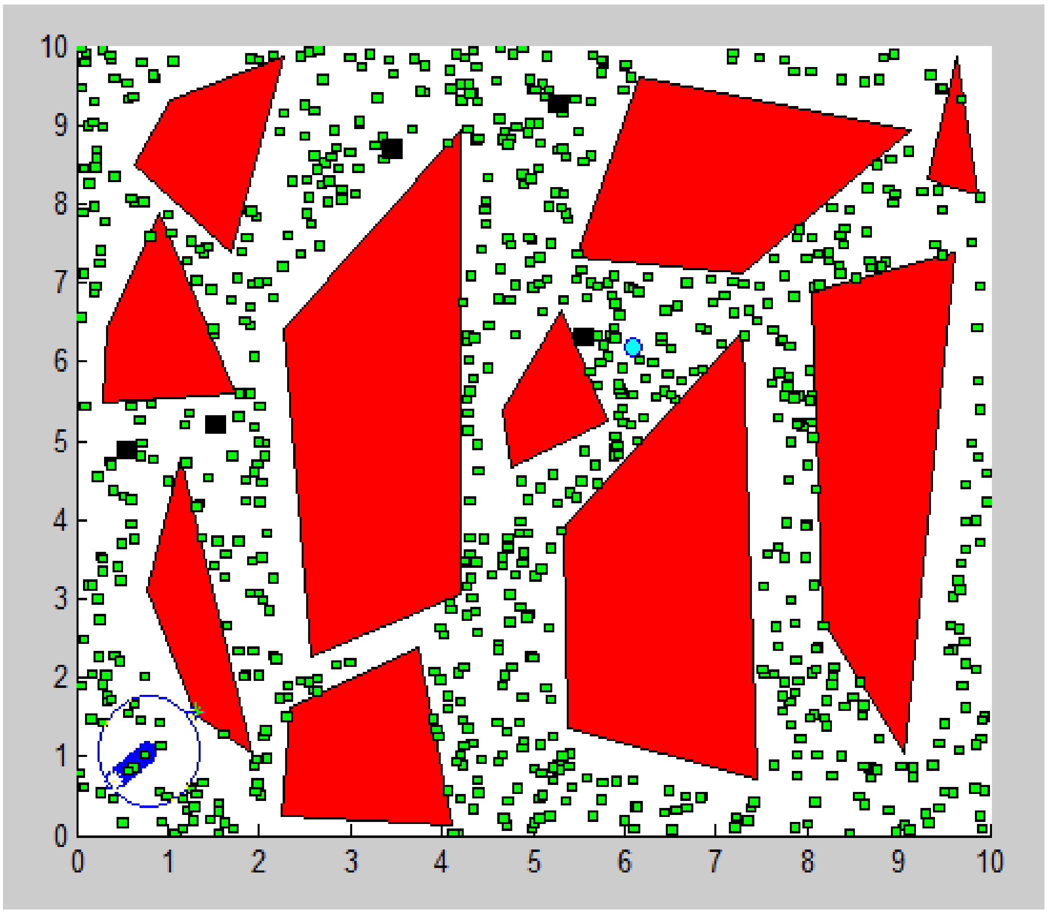


Fig 4.4: Unsuccessful run of Version 1 algorithm

As can be seen from the figure above, the path computed by version 1 is not an optimal one. The optimal path is the path that travels from the bottom of the obstacle closest to the start point and towards the goal.

4.3 Results of Version 2 Algorithm

Following is a degree of freedom graph of a point that has a 100 point feasible configuration set around it. Now only those points that have an acceptable degree of freedom are inputted to the bacterial foraging algorithm.

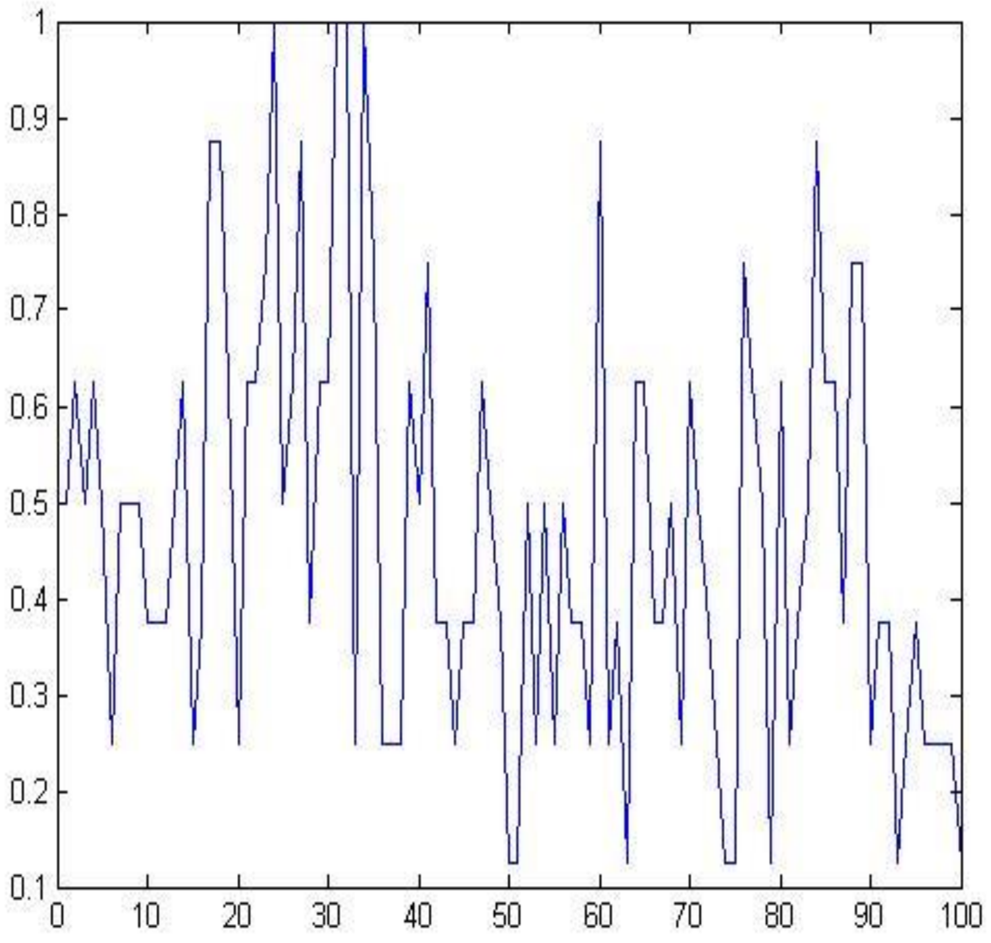


Fig 4.5: Degree of freedom graph for a given configuration point with 100 neighbours

A major disadvantage that Version 2 suffered from was that in some cases where an optimal path was possible the approach would compute a suboptimal path just because the waypoint that led to the optimal path did not satisfy the required degree of freedom level.

4.4 Results of Version 3 Algorithm

The following images show the results of the approach described in Version 3.

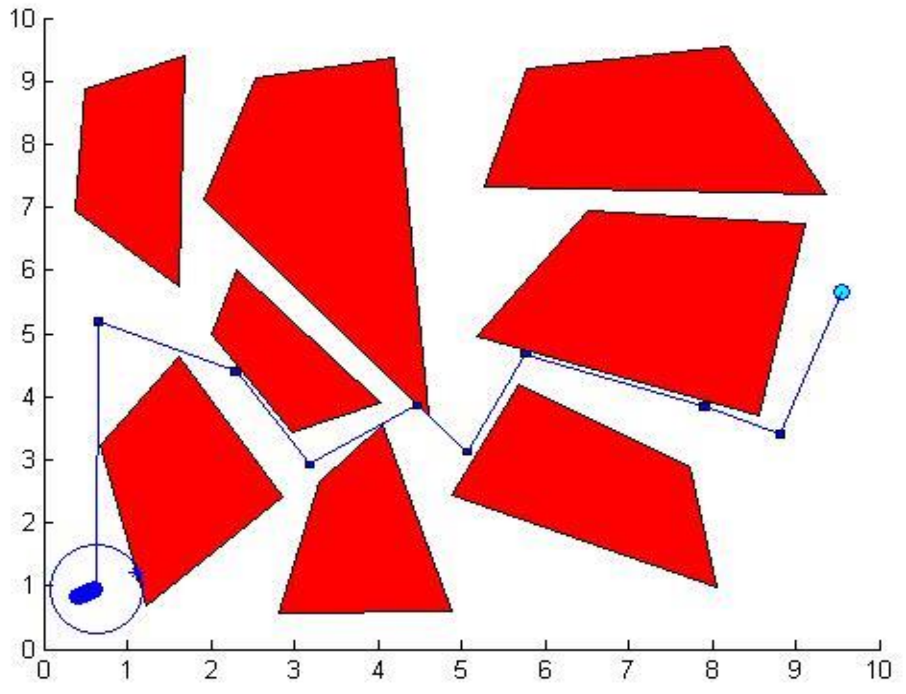


Fig 4.6: Successful run of Version 3 Algorithm

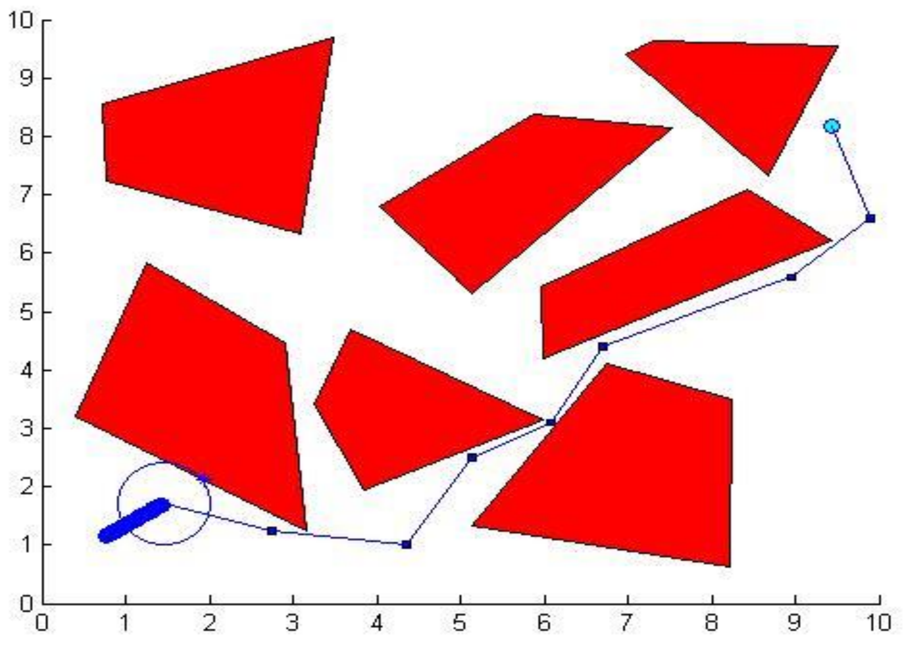


Fig 4.7: Successful run of Version 3 Algorithm

Also Version 3 performed where Version 2 failed, computing the optimal path in cramped situations like the one shown below. The path computed by Version 2 is shown in red and the one by Version 3, in blue.

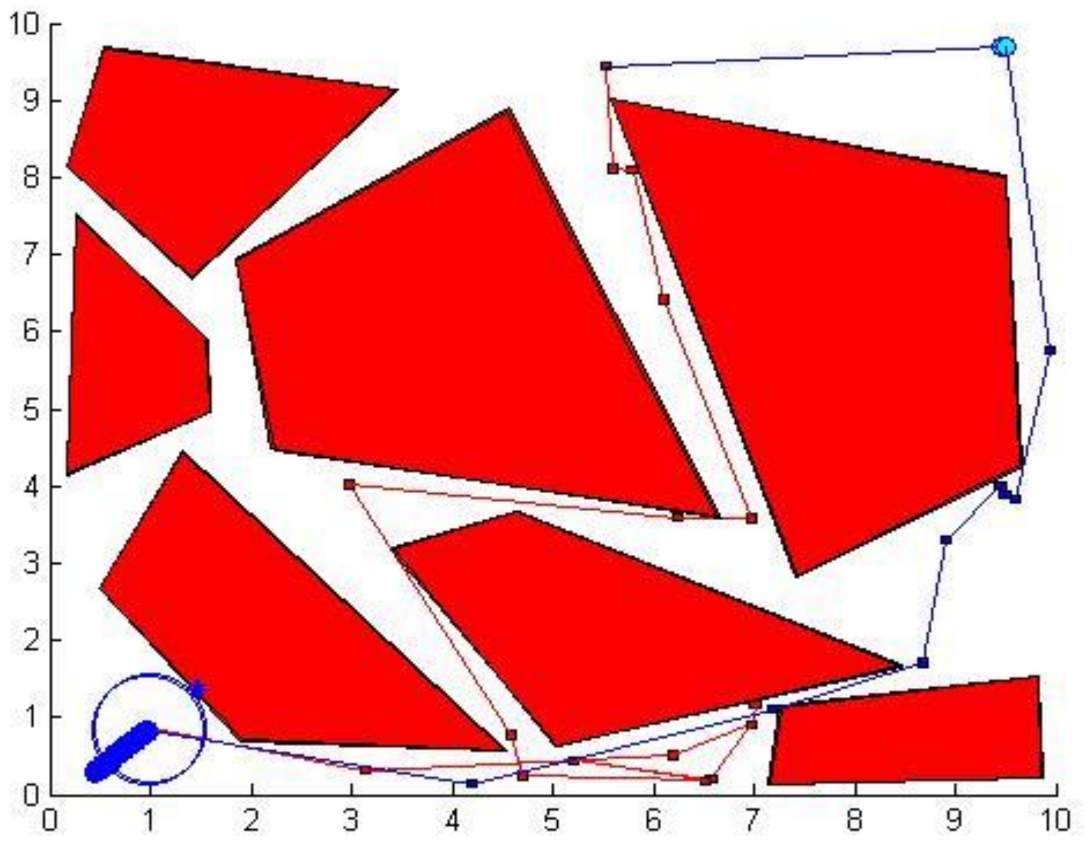


Figure 4.8: Comparison between the Version 2 and Version 3 algorithms

Even still though, there were some input configurations where the process seemed to hang or get stuck, one of which is being reproduced below.

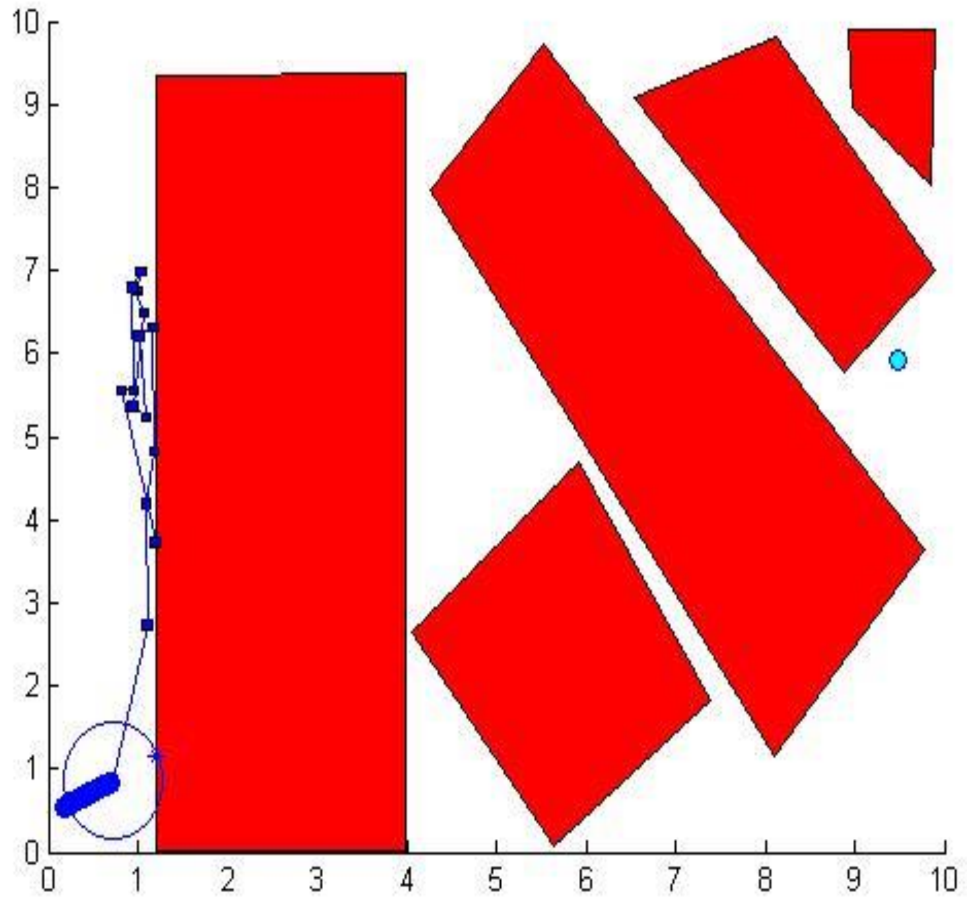


Figure 4.9: Unsuccessful run of Version 3 Algorithm

4.5 Results of Version 4 Algorithm

The image shows us the result of the spectral analysis performed on a sample working scenario. As explained before in the previous chapter, the entire set of free configuration points has been coloured coded into seven different bands, based on their Euclidean distance from the goal; violet being the nearest and red the farthest.

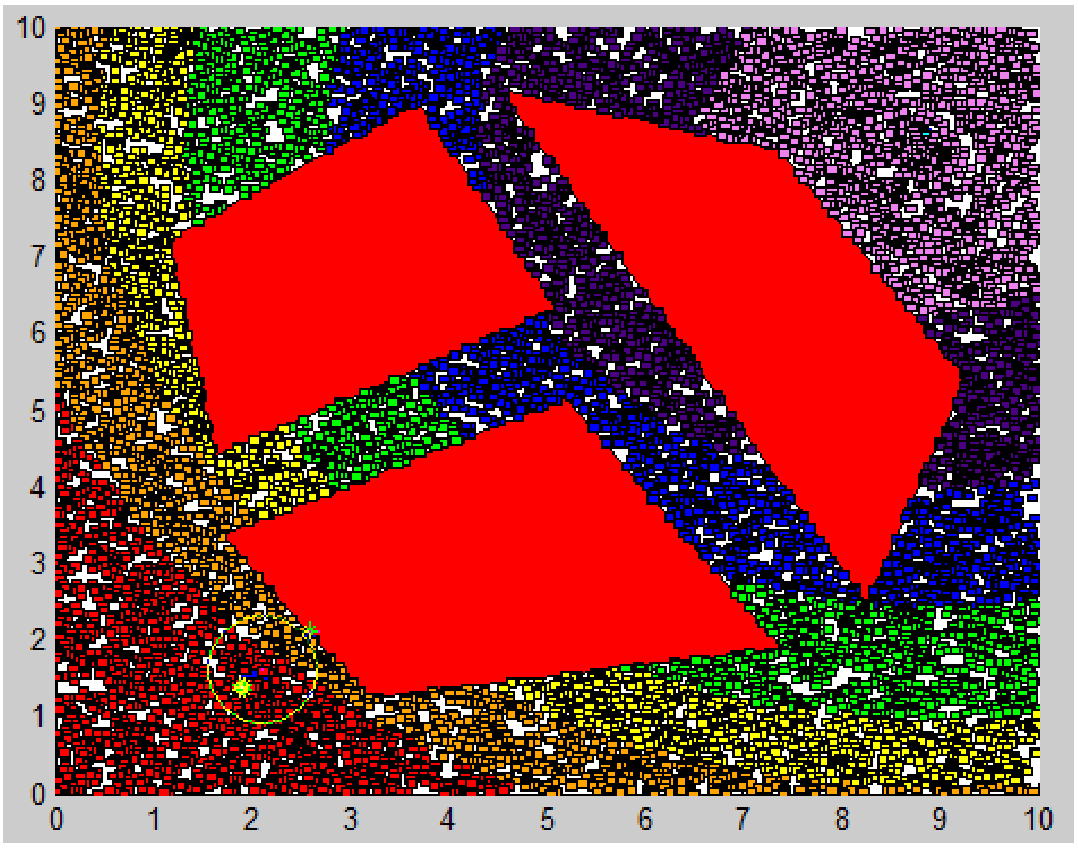


Figure 4.10: Unsuccessful run of Version 3 Algorithm

The results produced by Version 4 were comparable to the ones produced by version 3, though there were cases where it also failed to produce an optimal path

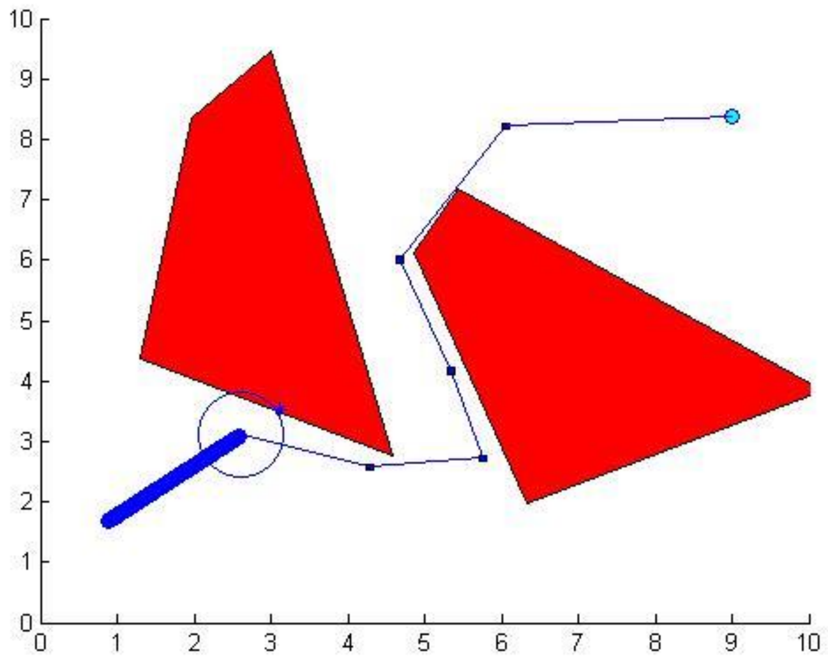


Figure 4.11: Successful run of Version 4 Algorithm

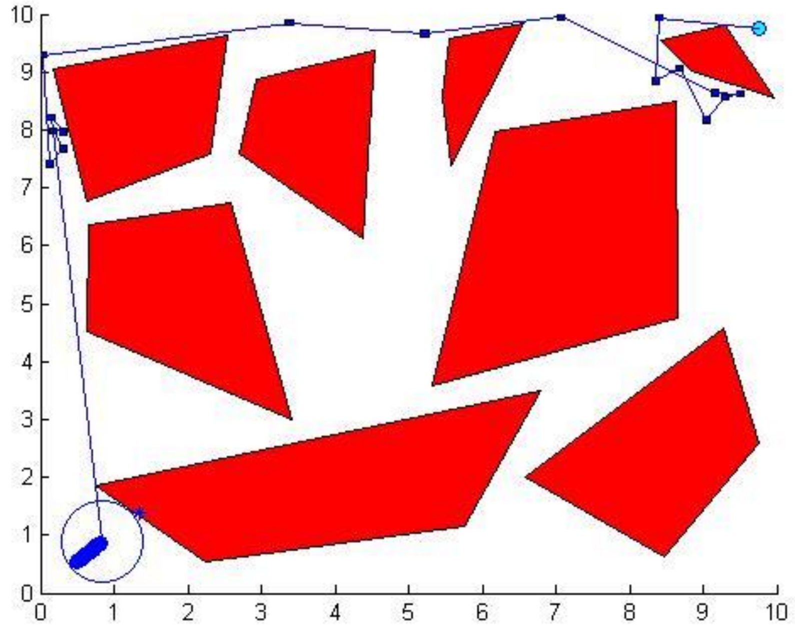
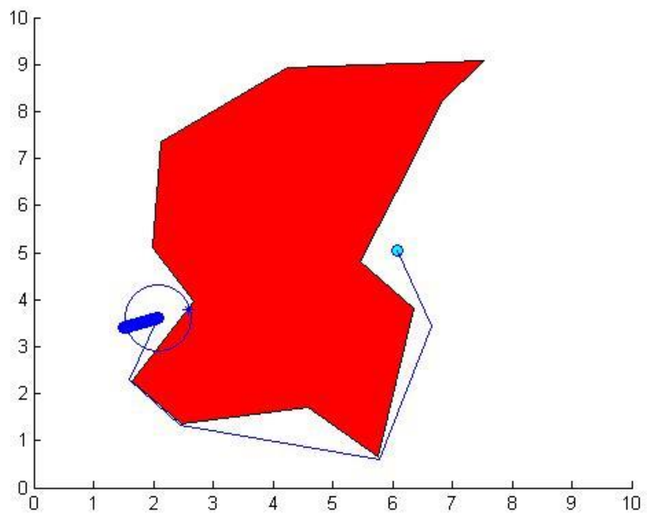


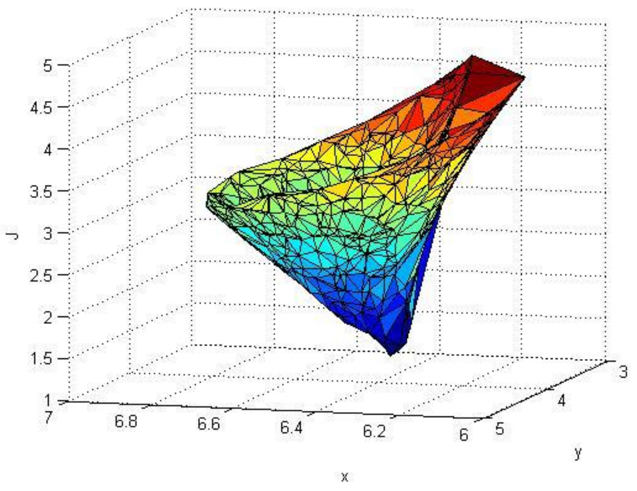
Figure 4.12: Unsuccessful run of Version 4 Algorithm

4.6 Results of the Proposed Approach

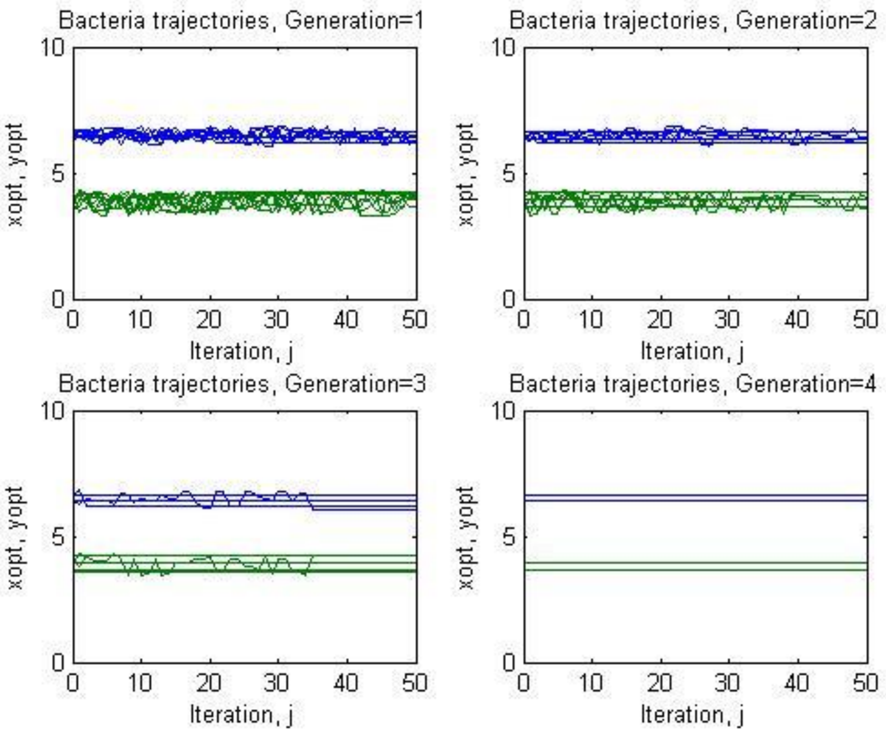
The following figures depict the results of the proposed approach described in the previous chapter. The first out of the four figures shows the computed optimal path. The second, a 3D representation of the objective function. The third and fourth figures show the trajectories travelled by the bacteria during the two generations of its lifetime.



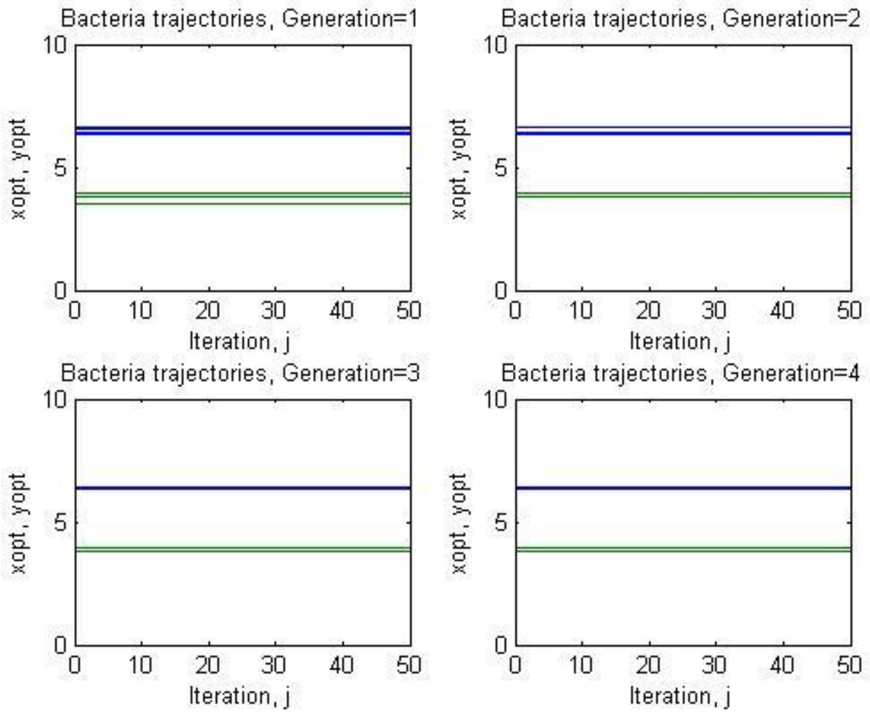
(a)



(b)

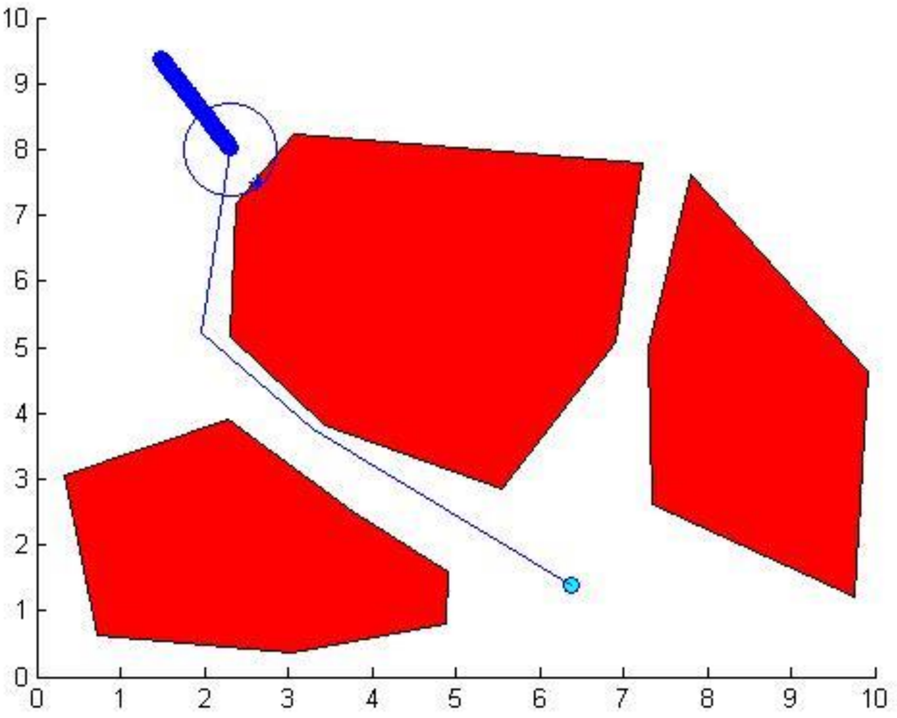


(c)

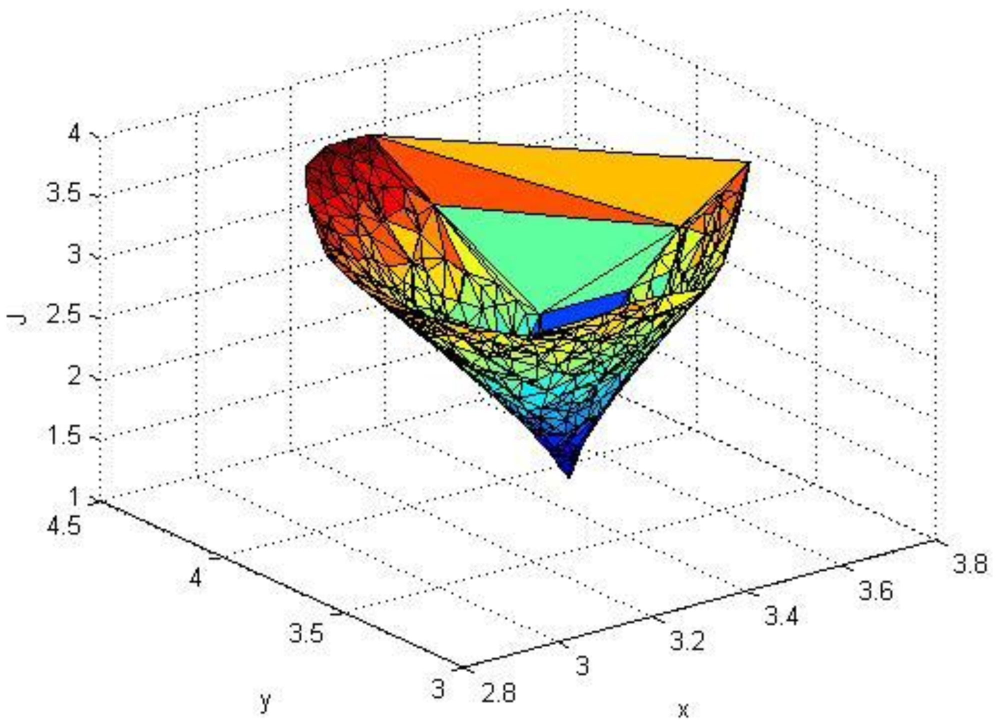


(d)

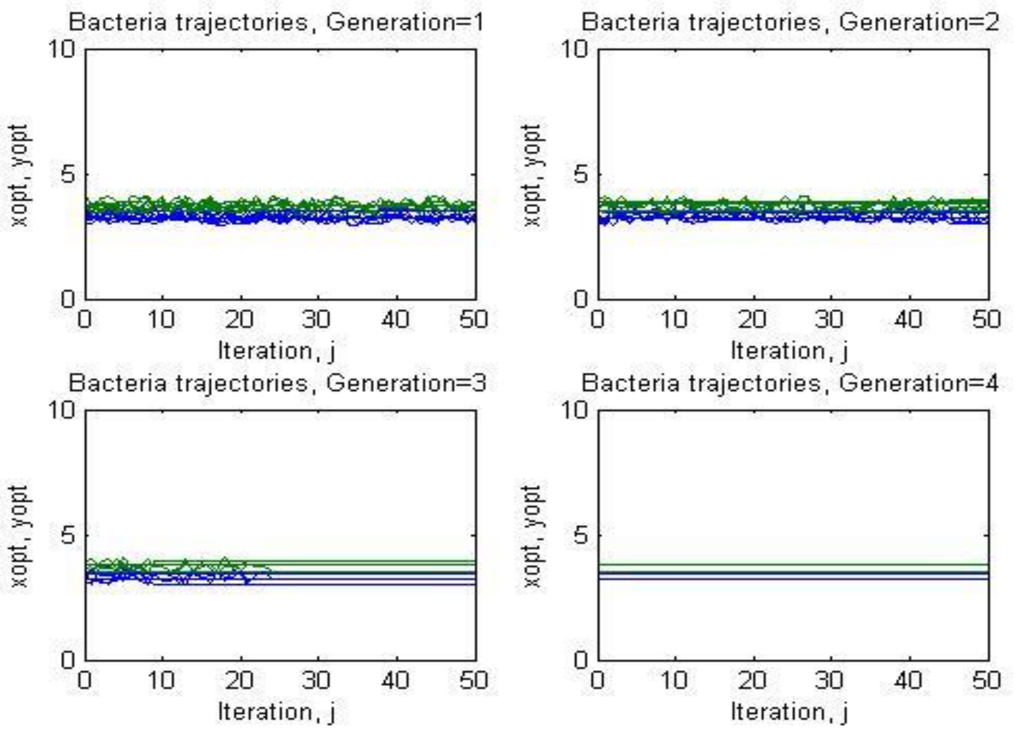
Fig 4.13(a) Optimal path computed, (b) 3D representation of the objective function, (c-d) Bacteria Trajectories



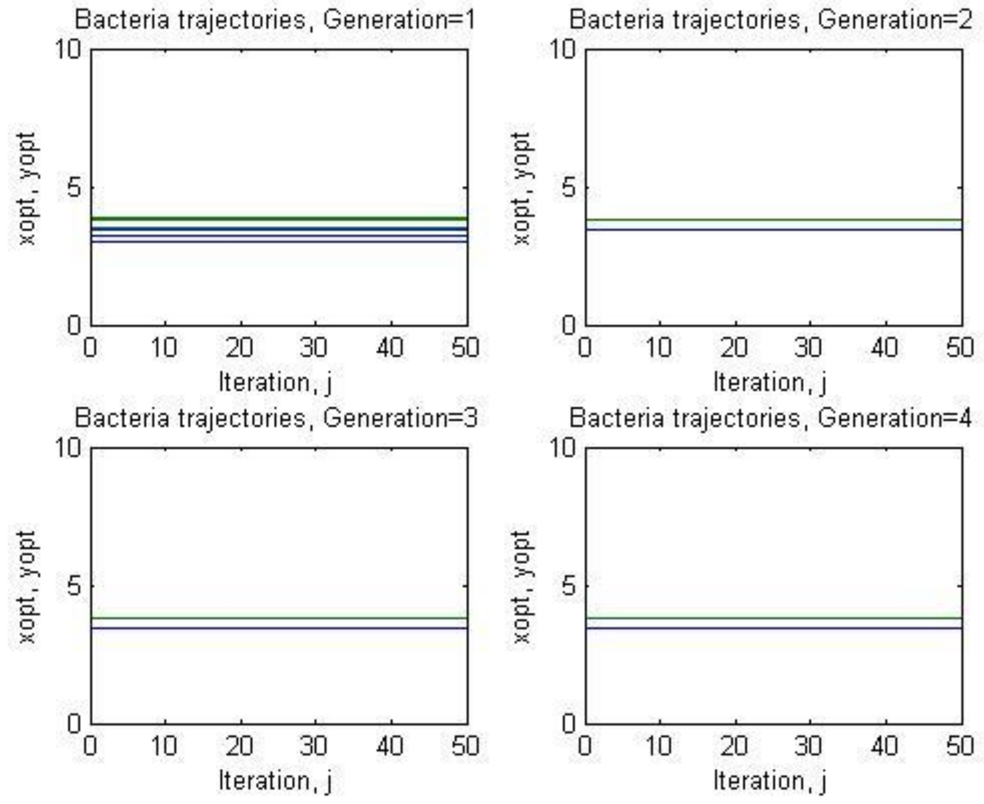
(a)



(b)

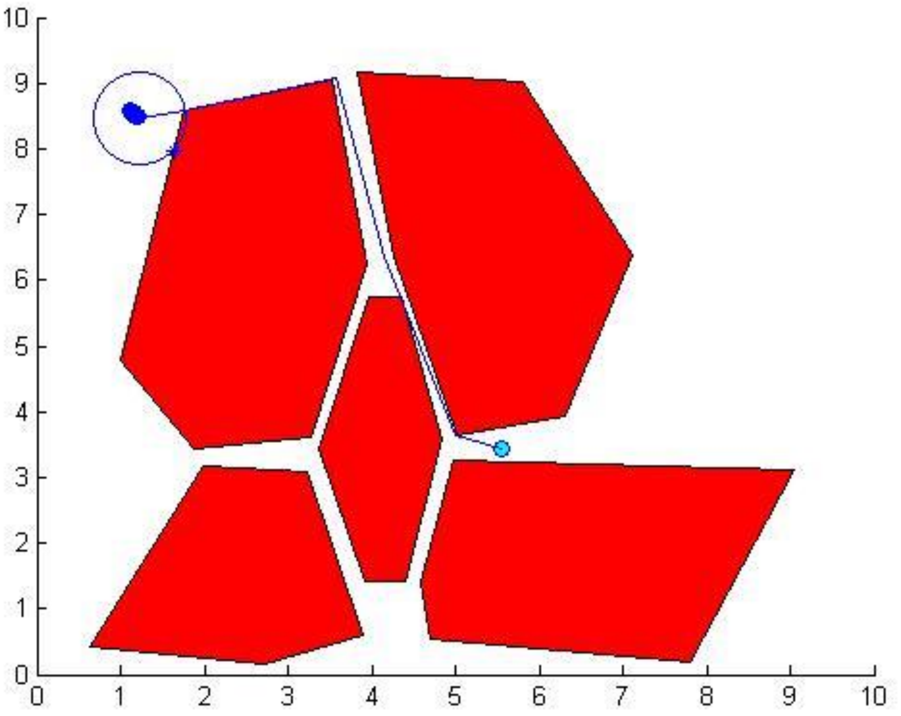


(c)



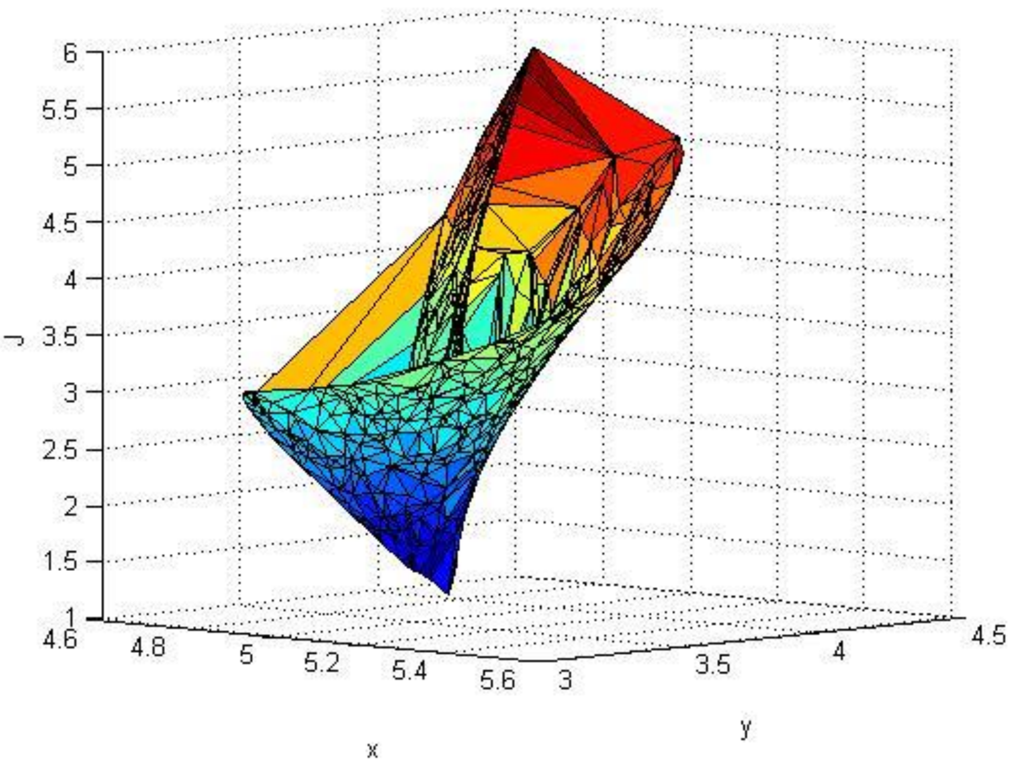
(d)

Fig 4.14(a) Optimal path computed, (b) 3D representation of the objective function, (c-d) Bacteria Trajectories



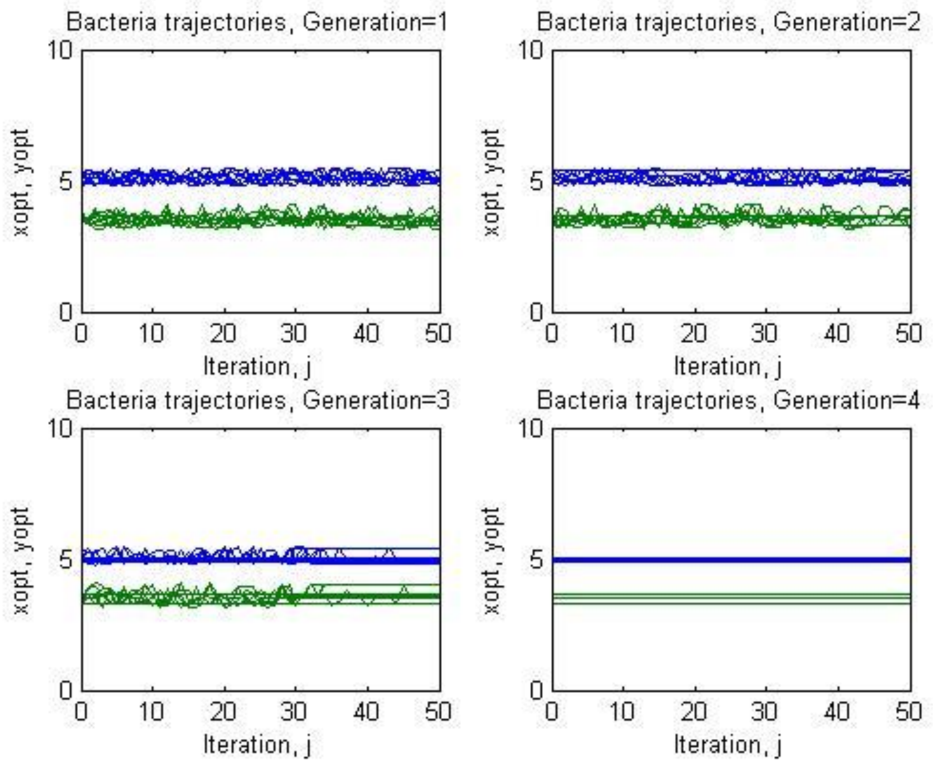
10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
10

(a)

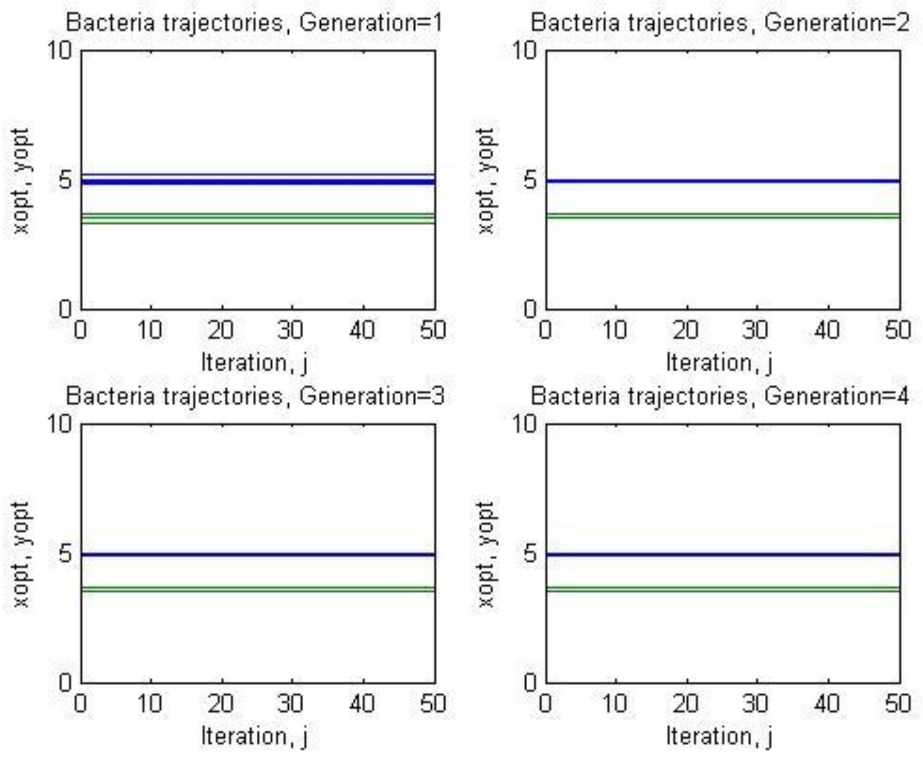


J
x
y
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1
4.6
4.8
5
5.2
5.4
5.6
3
3.5
4
4.5

(b)

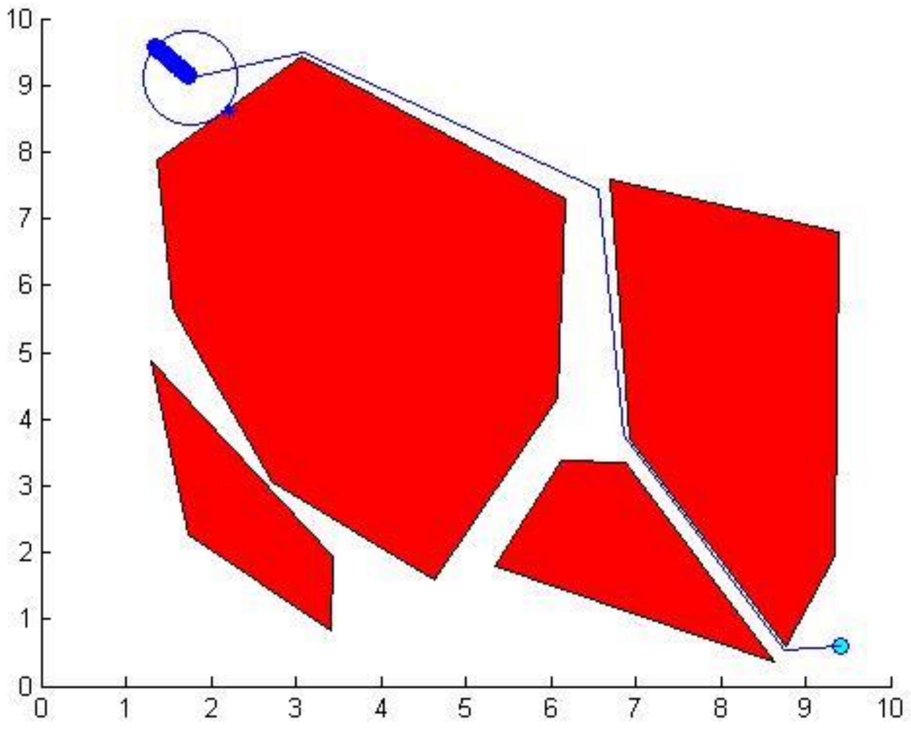


(c)

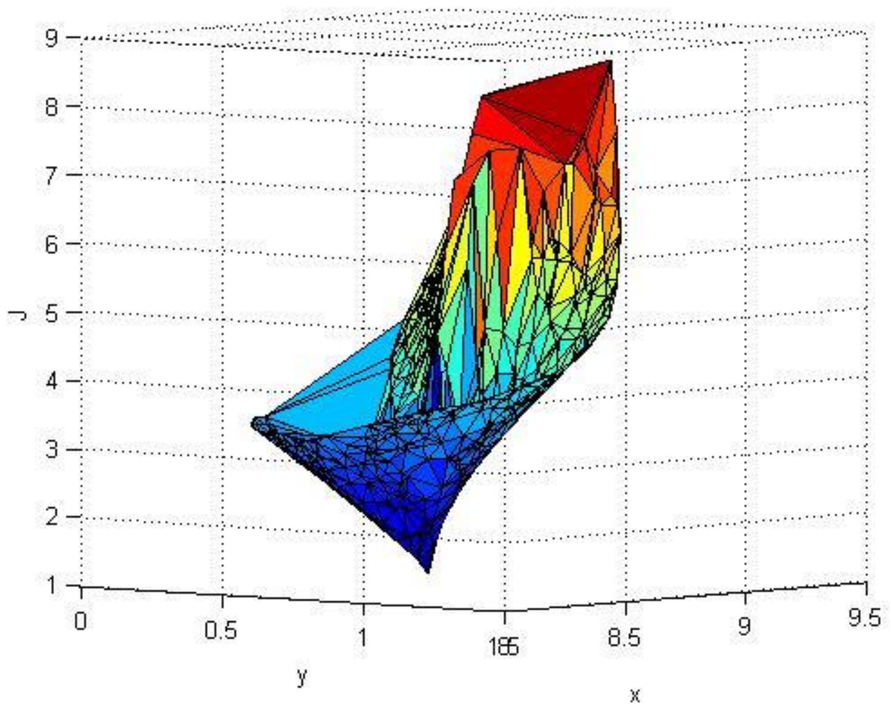


(d)

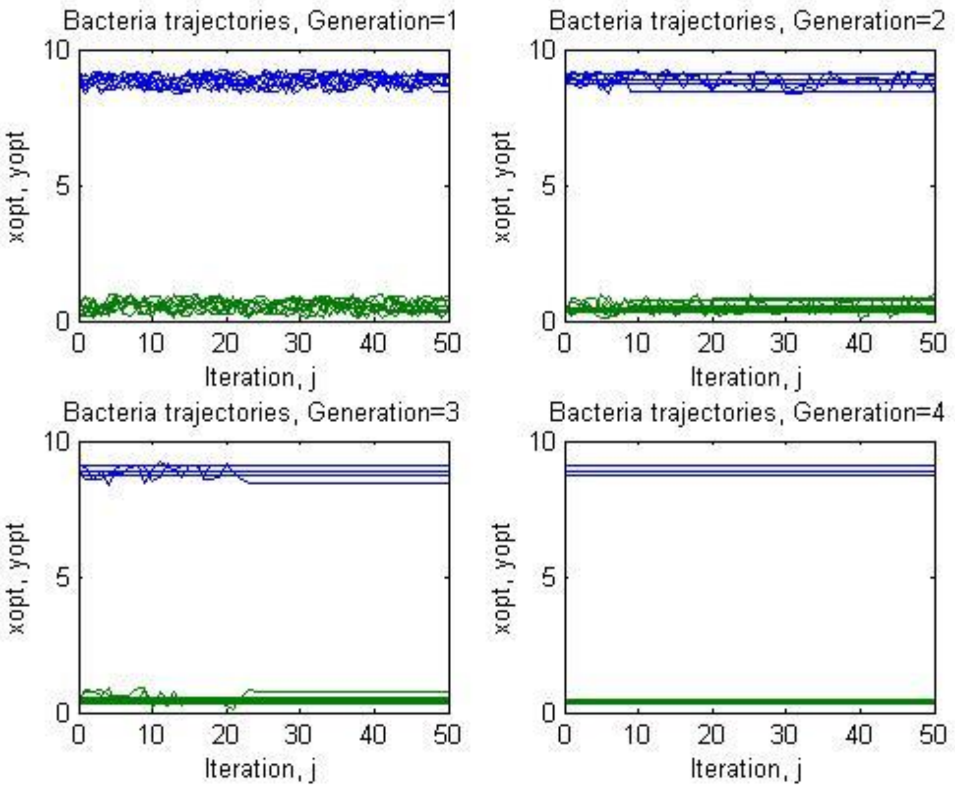
Fig 4.15(a) Optimal path computed, (b) 3D representation of the objective function, (c-d) Bacteria Trajectories



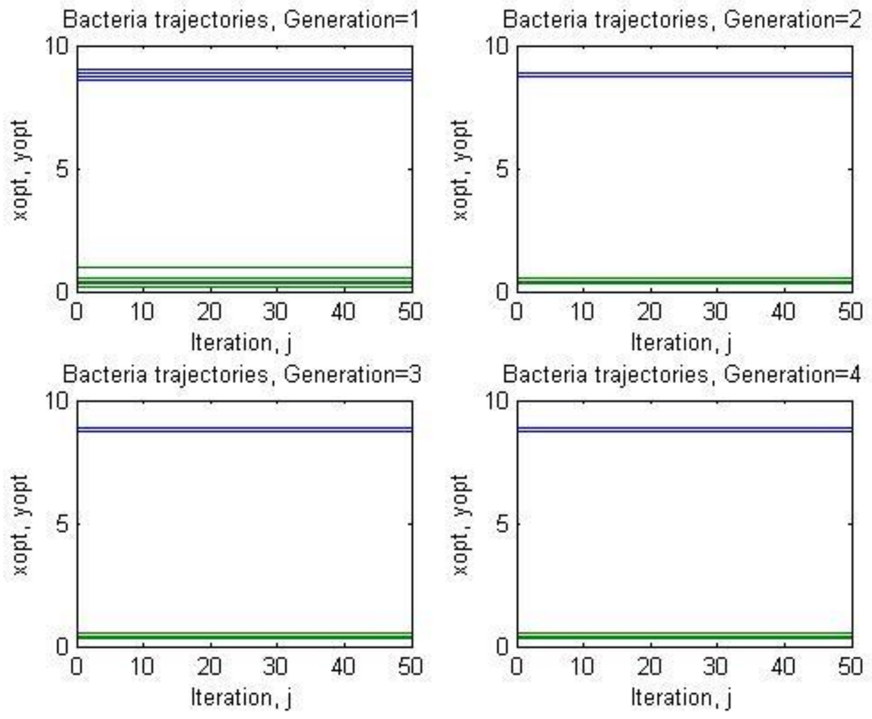
(a)



(b)

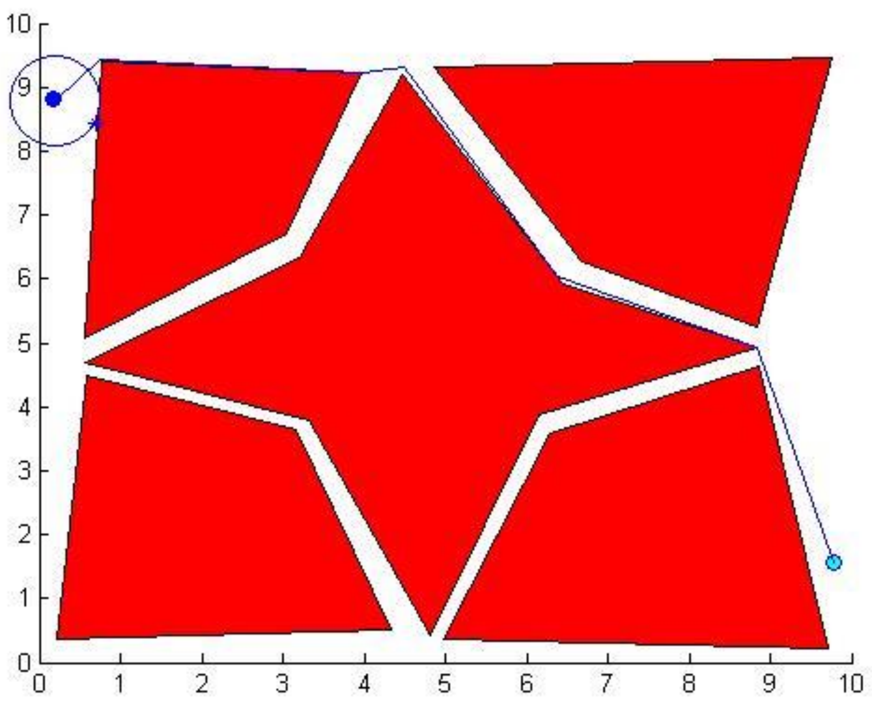


(c)

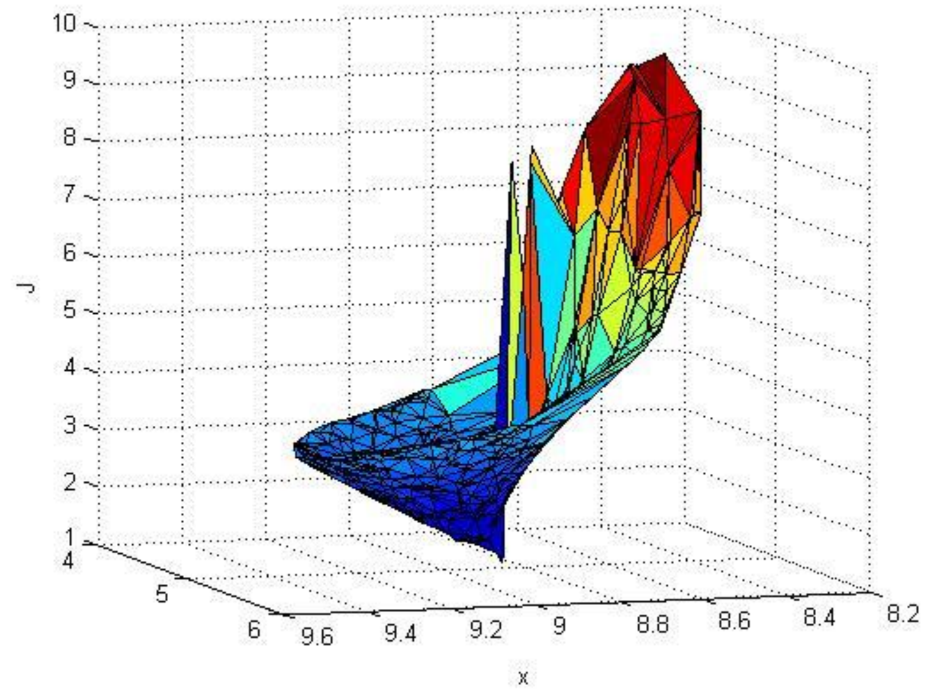


(d)

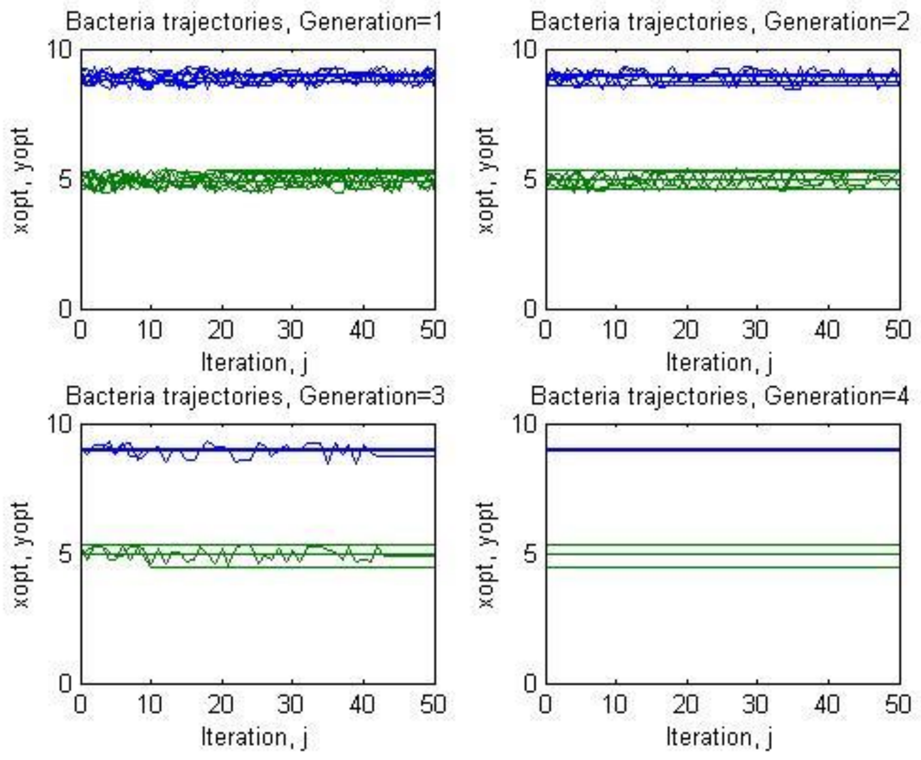
Fig 4.16(a) Optimal path computed, (b) 3D representation of the objective function, (c-d) Bacteria Trajectories



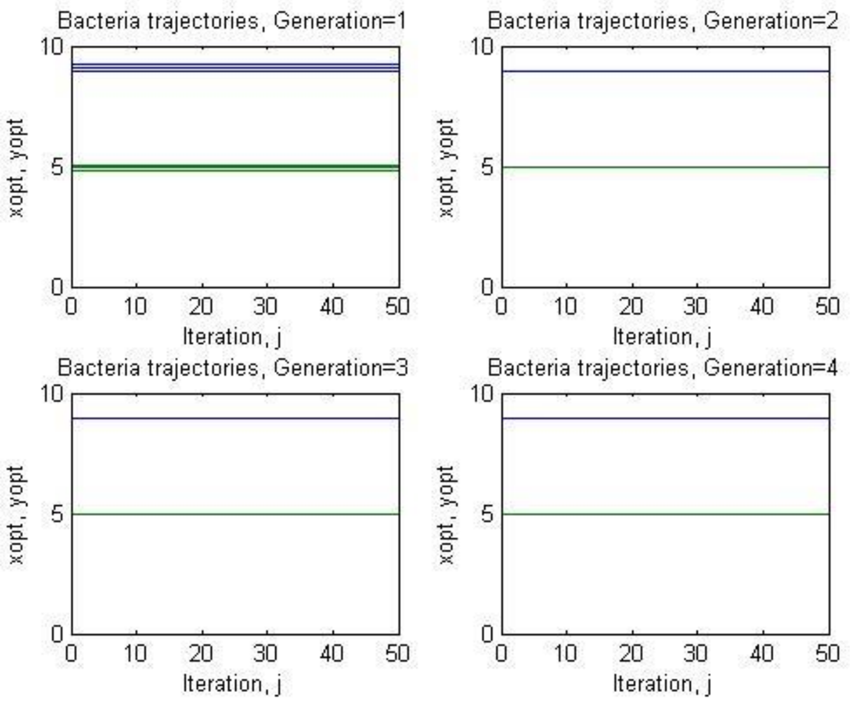
(a)



(b)



(c)



(d)

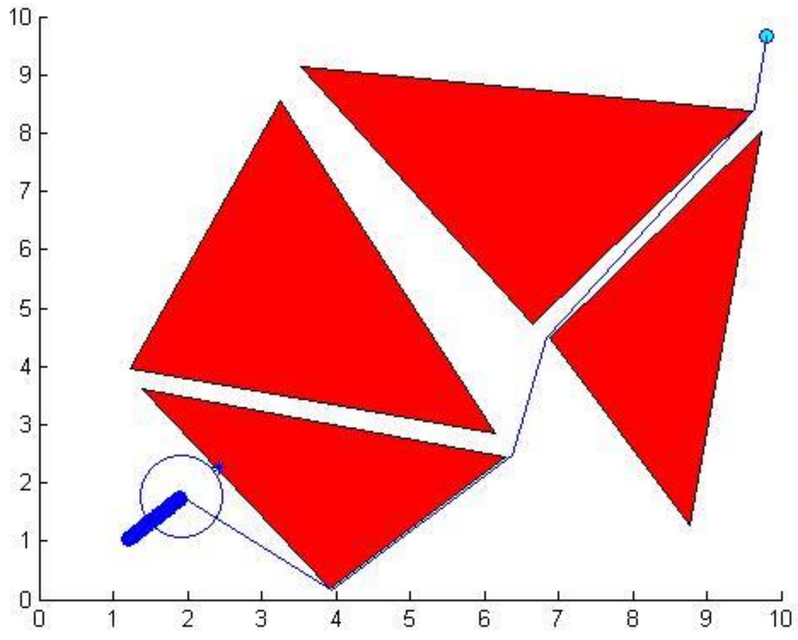
Fig 4.17(a) Optimal path computed, (b) 3D representation of the objective function, (c-d) Bacteria Trajectories

4.7 Comparison of Proposed Approach with Probabilistic Road Mapping

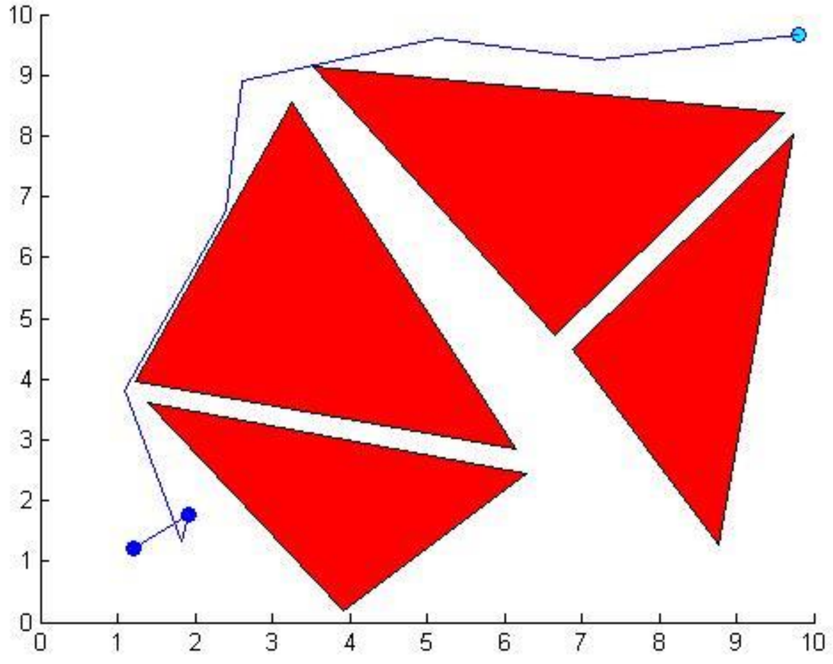
	Sim 1	Sim 2	Sim 3	Sim 4
Proposed Approach	15.1005	14.2366	12.2051	10.2347
PRM	16.6548	17.6717	19.3290	11.9023

Table 4.1: Comparison of the proposed approach with PRM on the basis of path length

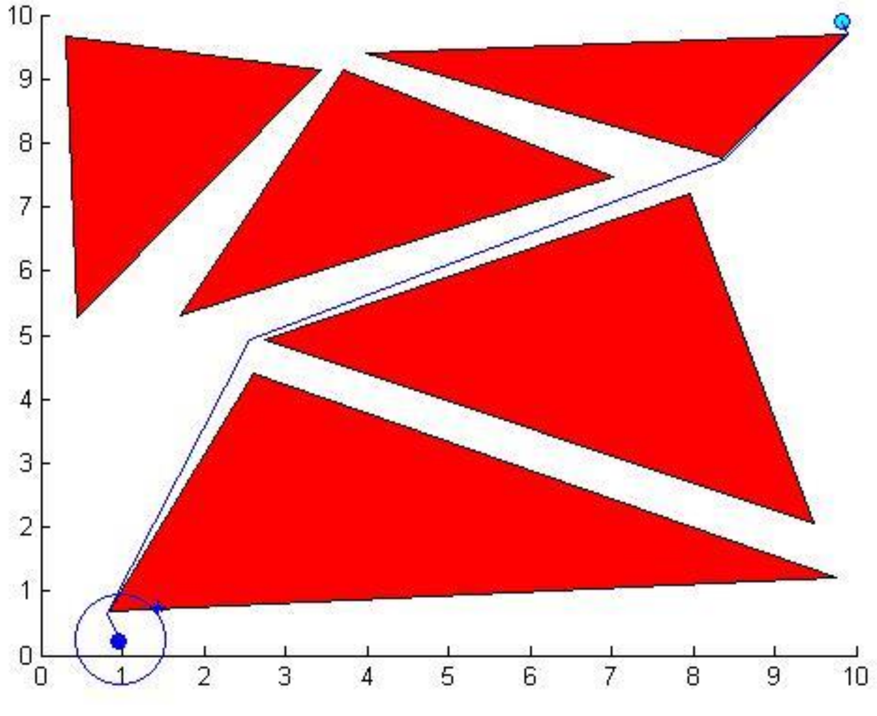
The proposed approach has been run and tested on several input configurations. Also a comparison has been made with the probabilistic road mapping planning algorithm. As depicted by table 1, the path lengths produced via the proposed approach are shorter in length than those produced by PRM. The actual working configurations listed in table 1 have been reproduced below. The first figure in every simulation depicts the result of the proposed approach, while the second the result of PRM. It is clear from the simulations below that the proposed approach indeed produces paths that are shorter in length than those produced by the PRM approach. Paths are also relatively straighter with no backtracking or zig zagging.



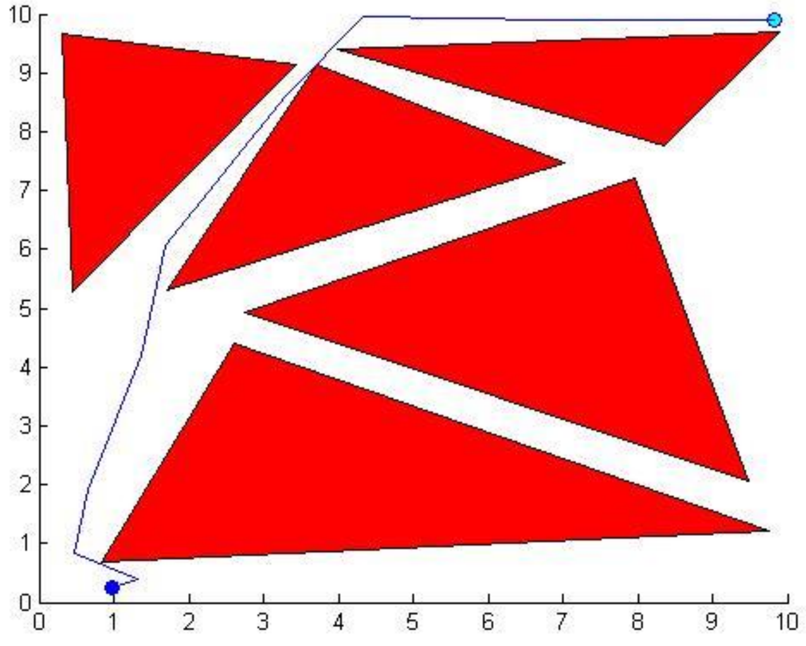
Simulation 1(a)



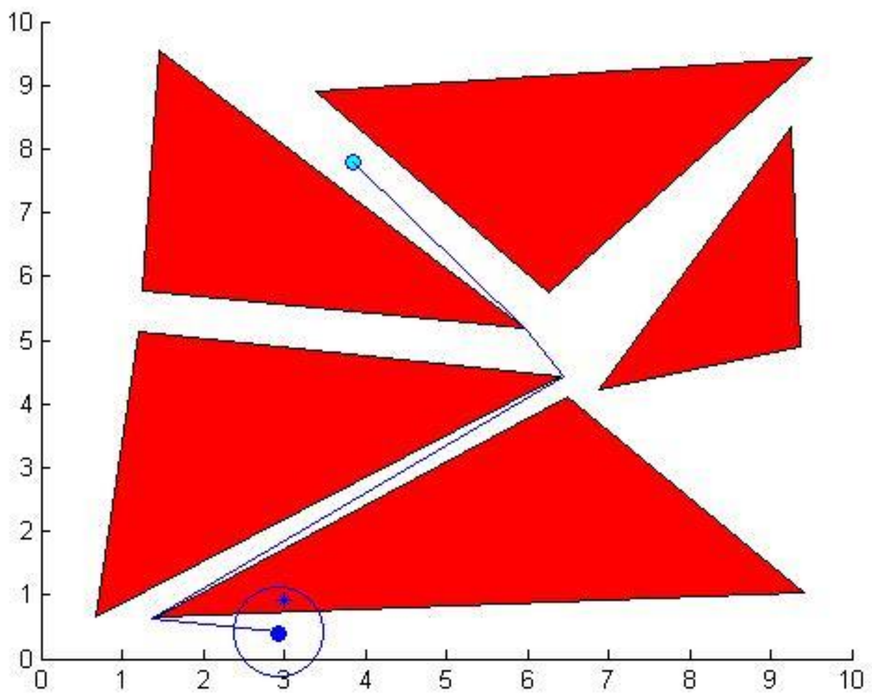
Simulation 1(b)



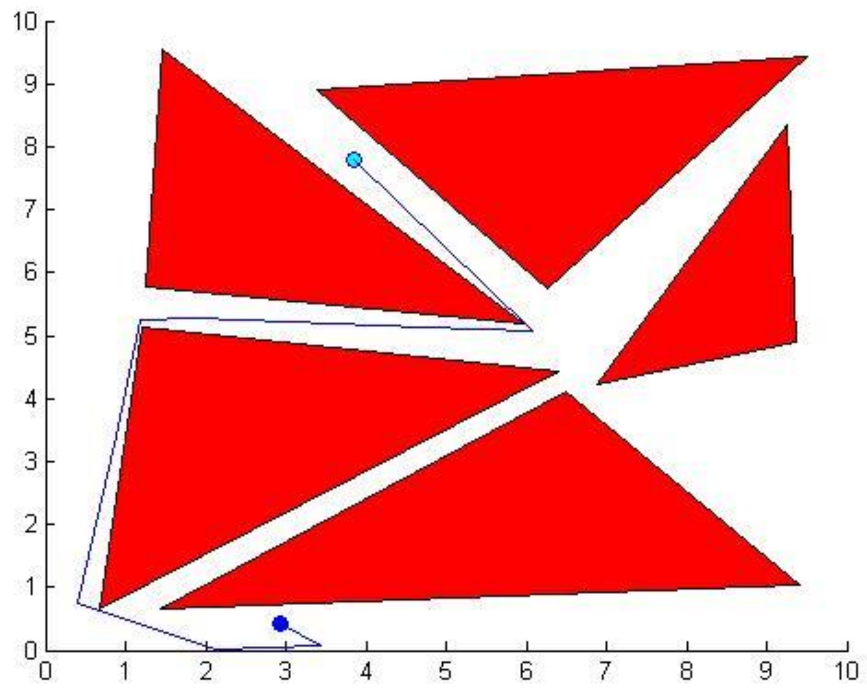
Simulation 2(a)



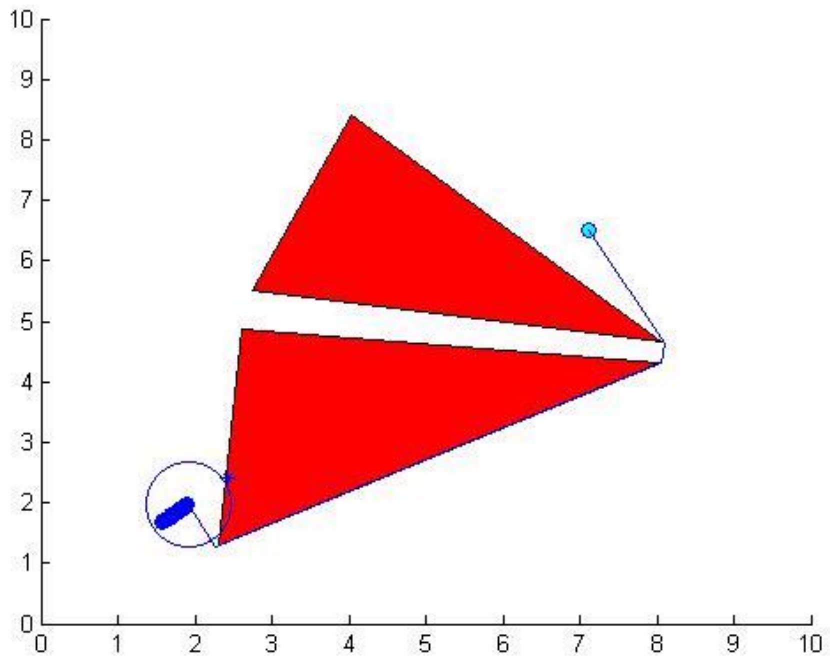
Simulation 2(b)



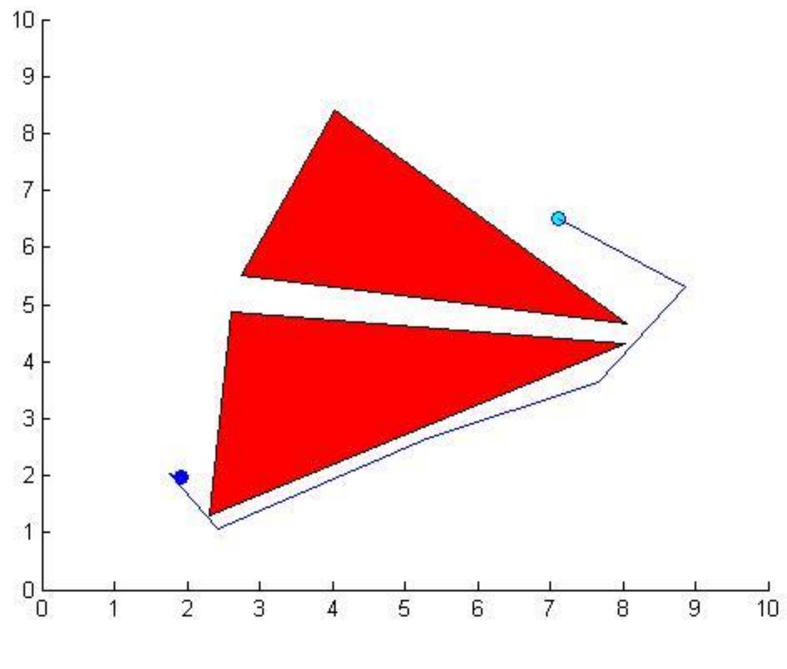
Simulation 3(a)



Simulation 3(b)



Simulation 4(a)



Simulation 4(b)

Figure 4.16(1(a-b)-4(a-b)): Results of the comparison done between the proposed approach, Simulations(a) and the PRM, Simulations(b) listed in Table 1

Chapter 5: Conclusion and Future Scope

Bacterial foraging, an evolutionary algorithms has be used to generate a solution to the path planning algorithm. The process entails first treating the entire working configuration as a set of discrete points. Then a set of free configuration points have been generated. These are those points that are outside any obstacle. These points are then forwarded to the bacterial foraging algorithm which then outputs the optimal configuration point from the current set of free configuration points. The second method solves the path planning problem by creating a visibility graph of the given obstacle network. The A* algorithm is then applied to this visibility graph to output regions from where the optimal path will traverse. Instead of populating the entire workspace with free configuration points, Free configuration fields are generated only at the regions returned by the A* algorithm. Bacterial Foraging then selects a point that serves as the path point in this optimal region. Results are fairly optimal and are generated within an acceptable time window. An advantage of using evolutionary algorithms is the flexibility and sophistication of its objective function. A future research direction would be the incorporation of various other parameters such as path feasibility and path safety to the objective function. Domain specific information could be also added to the objective function to make the objective function even more versatile, as different fields will have different optimal path criteria. Also other evolutionary algorithms like PSO, ACO, BBO etc.. could be explored for the purpose of parameter optimization

References

[1] Masatomo KANEHARA, Satoshi KAGAMI, James.J Kuffner, Simon Thompson, Hiroshi MIZOGUHI, “Path shortening and smoothing of grid-based path planning with consideration of obstacles”, ISIC, IEEE International Conference on Systems, Man and Cybernetics, 2007, pg 991-996.

[2] Han-dong Zhang Bao-hua Dong Yu-wan Cen Rui Zheng, “Path Planning Algorithm for Mobile Robot Based on Path Grids Encoding Novel Mechanism”, ICNC 2007, Third International Conference on Natural Computation, 2007, pg 351-356

[3] Guo shuai, Dai Leyin, Ouyang YuPing, “The Research and Simulation on the path planning based on the Improved Grid Model” 2009 International Conference on Artificial Intelligence and Computational Intelligence”, International Conference on Artificial Intelligence and Computational Intelligence, 2009. AICI '09, pg 318-321.

[4] M. Al Marzouqi, “Efficient Path Planning for Searching a 2-D Grid-Based Environment Map”, 2011 IEEE GCC Conference and Exhibition (GCC), pg 116-119

[5] Yong K. Hwang, and Narendra Ahuja, “A Potential Field Approach to Path Planning”, IEEE transactions on robotics and automation, vol. 8, no. 1, February 1992, pg 23-32

[6] Guanghui Li, Atsushi Yamashita and Hajime Asama Yusuke Tamura, "An Efficient Improved Artificial Potential Field Based Regression Search Method for Robot Path Planning", 2012 IEEE International Conference on Mechatronics and Automation, pg 1227-1232

[7] Charles W. Warren , "Global Path Planning Using Artificial Potential Fields", Robotics and Automation, 1989. Proceedings, 1989 IEEE International Conference, pg 316-321

[8] He Xiaoxi Chen Leiting, "Path Planning Based on Grid-Potential Fields" , 2008 International Conference on Computer Science and Software Engineering, pg 1114-1116

[9] L. Kavraki and J. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3020–3026, 1994.

[10] Li, Wei ; Wang, Gai-Yun, "Application of improved PSO in mobile robotic path planning" International Conference on Intelligent Computing and Integrated Systems (ICISS), 2010 , pg 45-48

[11] Masehian, Ellips ; Sedighizadeh, Davoud**,** "A multi-objective PSO-based algorithm for robot path planning", IEEE International Conference on Industrial Technology (ICIT), 2010 , pg 465-470

[12] Yong Tang, Qing Li, Lijun Wang, Chao Zhang, and Yixin Yin, “ An Improved PSO for Path Planning of Mobile Robots and Its Parameters Discussion”, International Conference on Intelligent Control and Information Processing, pg 34-38

[13] Yanrong Hu, Simon X. Yang1, Li-Zhong Xu and Max Q.-H. MengA , “Knowledge Based Genetic Algorithm for Path Planning in Unstructured Mobile Robot Environments”, International Conference on Robotics and Biomimetics 2004, pg 767-772

[14] Yanrong Hu and Simon X. Yang, “A Knowledge Based Genetic A1gorithm for Path Planning of a Mobile Robot”. International Conference on Robotics and Biomimetics 2004, pg 4350-4355

[15] Cem Hocao˘glu and Arthur C. Sanderson, “Planning Multiple Paths with Evolutionary Speciation”, IEEE transactions on evolutionary computation, vol. 5, no. 3, June 2001, pg 169-191

[16] Cezar Augusto Sierakowski and Leandro dos Santos Coelho, “Path Planning Optimization for Mobile Robots Based on Bacteria Colony Approach”, Applied Soft Computing Technologies: The Challenge of Complexity, Advances in Soft Computing Volume 34, 2006, pp 187-198

[17] Anupama Sharma, Miss Sampada Satav, “Path Navigation Using Computational Intelligence”, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 7, July 2012, pg 395-398

[18] Passino, K.M. (2002) Biomimicry of bacterial foraging for distributed optimization and control, *IEEE Control Systems* 22(3), pp. 52-67

[19] Swagatam Das, Arijit Biswas, Sambarta Dasgupta1, and Ajith Abraham, “Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications”

[20] Hart, P.E; Nilsson, N.J.; Raphael, B. (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. IEEE Transactions on System Science and Cybernetics SSC4 4(2): 100-107

[21] M. Overmars. A randomapproach to motion planning. Technical Report RUU-CS-92-32, Universiteit Utrecht, 1992.