MCT-242 Computer Programming-I

Complex Engineering Problem

Autonomous Robot Navigation

Submitted by:

Name: Abiam Asif Khalid Registration No: 2021-MC-40

Submitted to:

Sir Shujat Ali

Department of Mechatronics & Control Engineering

University of Engineering & Technology, Lahore.

December-2022

Table of Contents

Introduction:

Robotics are becoming a vital part of our lives and at gaining popularity and usage at an exponential rate. Through this project we aim to make a program in C language that finds the most suitable path and displays the course robot will take. It has multiple practical uses such as in case of emergencies like earthquakes, fires and other natural or man-made disasters.

For the purpose of this report will be discussing a grid of 16x16, but the project has been built with a square grid of nxn size in mind.

The robot calculates its path in the navigation area(grid), optimizes its path based on the infeasible steps, turns it take and number of steps it follows. This allows the program to efficiently find the solution and allows for greater resources to be used in performing the actual task (such as rescue mission). The program optimizes the path by allowing the robot to move in direction until an obstacle is detected. Once the obstacle has been avoided, the robot moves forward in a straight line toward the end point until either (1) another obstacle is encountered or (2) the target location is reached.

Genetic Algorithm:

The entire is based on the search heuristic called the genetic algorithm. This algorithm is in fact based of Charles Darwin's "Theory of Evolution by Natural Selection". Genetic algorithms frequently employ biologically inspired operators including mutation, crossover, and selection to produce high-quality solutions to optimization and search issues. [1] Solving sudoku puzzles,[2] hyperparameter optimization, and decision tree performance improvement are a few examples of GA applications.

The genetic algorithm generates a random population, evaluates the fitness of each sample on the bases of properties and abilities. Then selects the most fit ones, following the suite of "survival of the fittest", then using crossover produces children from the most fit parents, in an attempt to increase the fitness of the next generation. This process also includes random mutation of the generations. This process continues over multiple generations until the most fit sample is found.

The project has a total of 12 functions including the main(), uses 3 standard libraries [1]stdio.h [2]stdlib.h [3]time.h

The program comprises of 550 lines of code.

Path Planning:

The most important aspect of this project is the path planning, for this case, four (4) different options of path are used. The four option allows the program to explore different paths, and optimize the path of robot. This efficiently allows the robot find the optimum path. **The paths mainly depend on the representation of each chromosome. Chromosomes are represented as a 2-D array that stores the location of important landmarks in the path the robot follows on the grid.** We use this representation to define the various types movement, each equal in importance as there are many ways the obstacles can be placed and hence many ways to traverse around them.

The four options are:

1- Column Wise Column First
2- Colum Wise Row First
3- Row Wise Column First
4- Row Wise Row First

Column Wise Column First:

For column wise column first, we use chromosome as an array and the index of the array defines the column and value at each index of the chromosome represent the row on the grid.

Such that the chromosome 0 0 2 5 8 8 8 8 9 4 6 6 6 11 15 15 can be represented as X, and the intermediate steps are represented as O.

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X														
1		O														
2		O	X													
3			O													
4			O						O	X						
5			O	X					O	O						
6				O					O	O	X	X	X			
7				O					O				O			
8				O	X	X	X	X	O				O			
9								O	X				O			
10													O			
11													O	X		
12														O		
13														O		
14														O		
15														O	X	X

Column Wise Row First:

For column wise column first, we use chromosome as an array and the index of the array defines the column and value at each index of the chromosome represent the row on the grid.

Such that the chromosome 0 0 2 5 8 8 8 8 9 4 6 6 6 11 15 15 can be represented as X, and the intermediate steps are represented as O.

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	O													
1			O													
2			X	O												
3				O												
4				O						X	O					
5				X	O					O	O					
6					O					O	X	X	X	O		
7					O					O				O		
8					X	X	X	X	O	O				O		
9									X	O				O		
10														O		
11														X	O	
12															O	
13															O	
14															O	
15															X	X

Row Wise Column First:
For column wise column first, we use chromosome as an array and the index of the array defines the row and value at each index of the chromosome represent the column on the grid.

Such that the chromosome 0 0 2 5 8 8 8 8 9 4 6 6 6 11 15 15 can be represented as X, and the intermediate steps are represented as O.

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X															
1	X															
2	O	O	X													
3			O	O	O	X										
4						O	O	O	X							
5									X							
6									X							
7									X							
8									O	X						
9					X	O	O	O	O	O						
10					O	O	X									
11							X									
12							X									
13							O	O	O	O	O	X				
14												O	O	O	O	X
15																X

Row Wise Column First:
For column wise column first, we use chromosome as an array and the index of the array defines the row and value at each index of the chromosome represent the column on the grid.

Such that the chromosome 0 0 2 5 8 8 8 8 9 4 6 6 6 11 15 15 can be represented as X, and the intermediate steps are represented as O.

+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X	X	O													
1			O													
2			X	O												
3				O												
4				O						X	O					
5				X	O					O	O					
6					O					O	X	X	X	O		
7					O					O				O		
8					X	X	X	X	O	O				O		
9									X	O				O		
10														O		
11														X	O	
12															O	
13															O	
14															O	
15															X	X

Functions:

generate_population ():
This function generates the random population for the grid using random numbers generator rand() function and srand() from stdlib.h header file. The function uses only one parameter that is the 2D array storing all the chromosomes. Each chromosome starts with 0 and ends with 15, or for nxn grid starts with 0 and ends with n-1. The chromosome has random values ranging from 0 to n-1 or 15 for our case. The size of each chromosome is actually two less, so for our case it would be an array 14 elements. The return type of function is void.

generate_direction_orientation ():
This function works similar to generate population but fills two global arrays both 1D and return type void. These arrays have directional value and orientation value. The orientation decides where the index of the chromosome represents row or columns and directional value represents whether the value stored in array represents rows or columns.

copymaker ():
Since we created all chromosomes excluding first and last genes. These genes are added using copymaker () function it creates a copy array with all 16 genes to be used in path planning functions. This function also has return type void.

pathfunc ():
The pathfunc () takes the copy array containing all the genes made from copymaker (), along with direction and orientation bit then using these values returns a 2D array with all the coordinates that are mapped from the chromosome.

pathevaluator ():
The path evaluator function evaluates the path that was produced by pathfunc () and fills the three arrays containing data about steps in paths, turns in path, and path length, for each chromosome.

fitnessfn ():
The fitnessfn () takes the arrays containing data about steps in paths, turns in path, and path length, for each chromosome. Find minimum and maximum values for each parameter, normalizes the values of all three parameters so that they range from 0 to 1. Finally, it calculates the fitness of all the chromosomes and stores it an array.

sortfitness ():
This function basically works as the parent selector for the next generation of chromosomes using the principles of bubble sort on fitness value and arranges the half of all most fit chromosomes.

crossoverfn ():

This function uses the half of the fittest chromosomes are produces the next generation by breaking 2 chromosomes in 2 halves each having 7 chromosomes for our case, producing 2 daughter chromosomes. Daughter chromosome 1 is comprises of 1st half of parent chromosome 1 and 2nd half of parent chromosome 2 and likewise daughter chromosome 2 is comprises of 1st half of parent chromosome 2 and 2nd half of parent chromosome 1. Producing new generation ready to be mutated.

mutation ():
This function randomly changes the value of random index for **each** chromosome in the population. This random mutation allows for better and more efficient results.

solution ():

solution () function checks all the chromosomes for the best and most efficient one. The solution has check for infeasible steps should be zero and fitness should be greater than 250. If the chromosome satisfies these conditions, it returns the index of population array that stores the chromosomes.

display_soln ():

This function is mainly used to display the best path we found using our genetic algorithm on grid 16x16 and displays all the steps for the robot would follow to achieve the goal.

Graphs:

Grid:

The 16x16 grid was provided by the instructor and shown below:

Population size for this grid was **2000** and iterations **10000.**

```
int grid[GRIDSIZE][GRIDSIZE] =      // 20-by-20
              {{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                {0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0},
                {0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0},
                {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
                {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
                {0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0},
                {0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0},
                {0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0},
                {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
                {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
                {0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0},
                {0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0},
                {0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0},
                {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
                {0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1},
                {0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1},
```

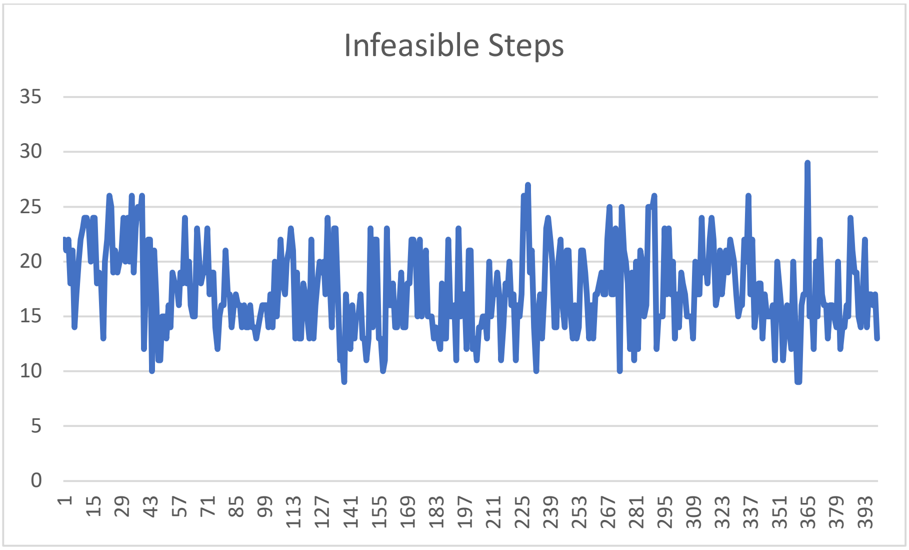
{0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

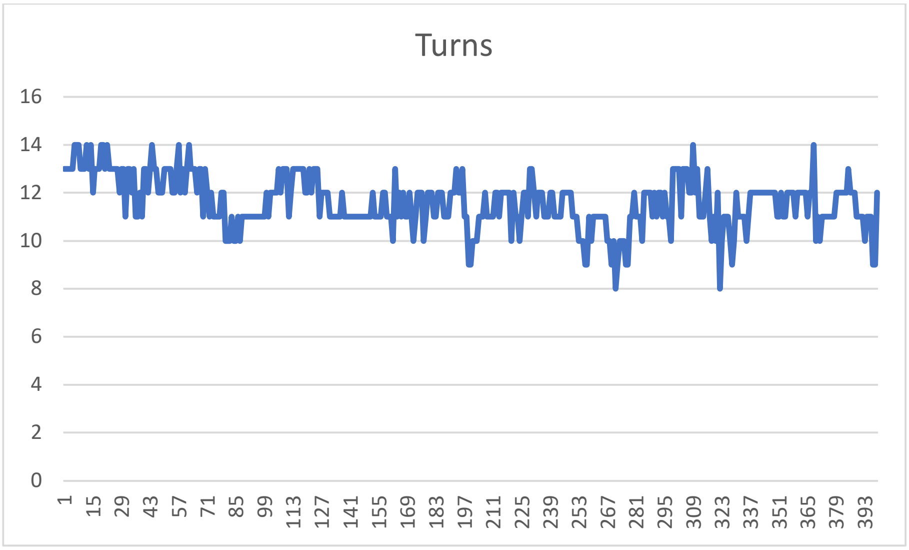
{0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0},

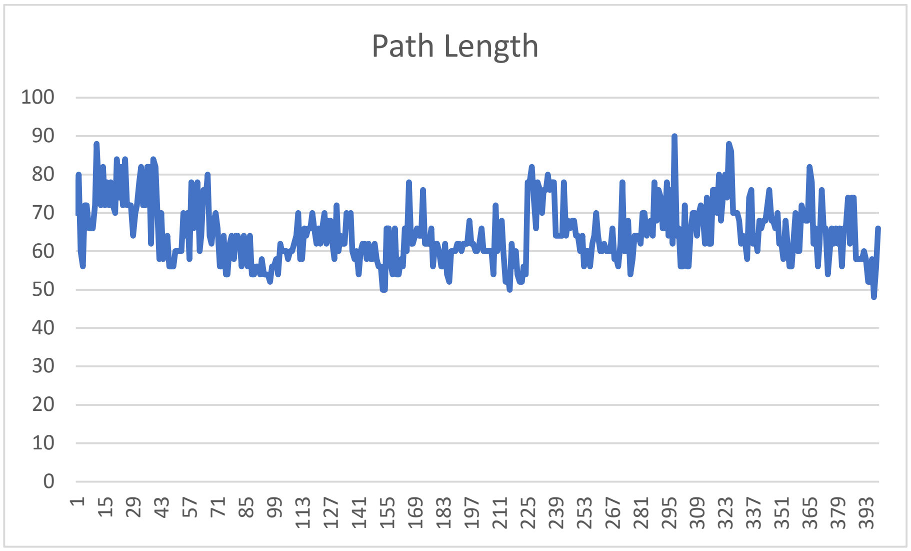
{0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0},

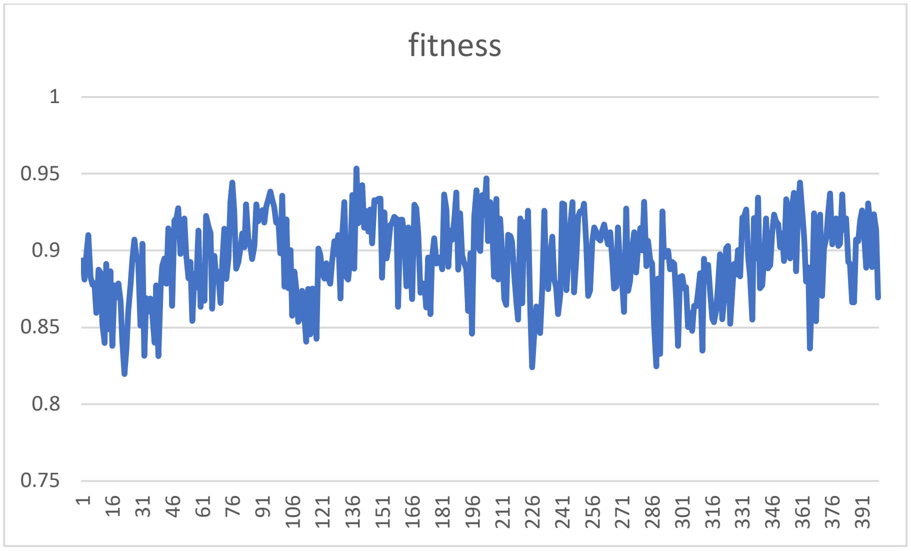
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}};

Run 1:

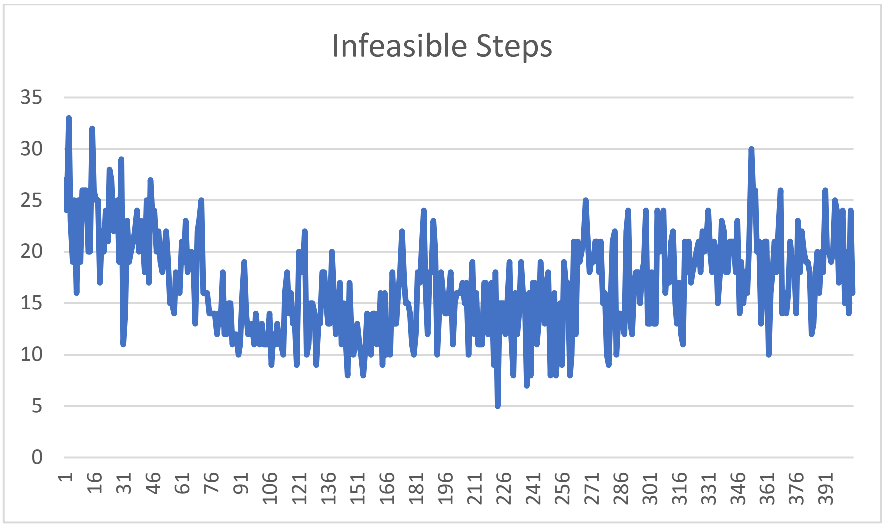


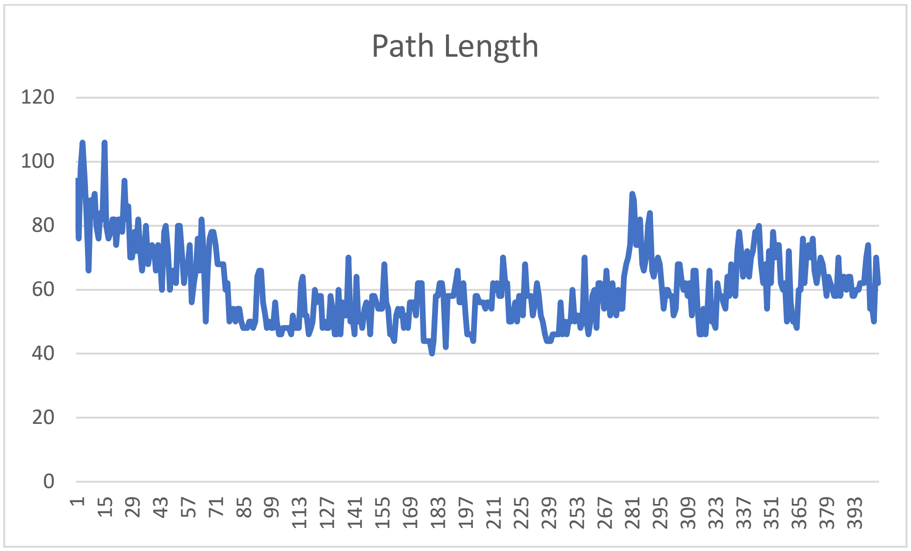



Path Length
100
90
80
70
60
50
40
30
20
10
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393

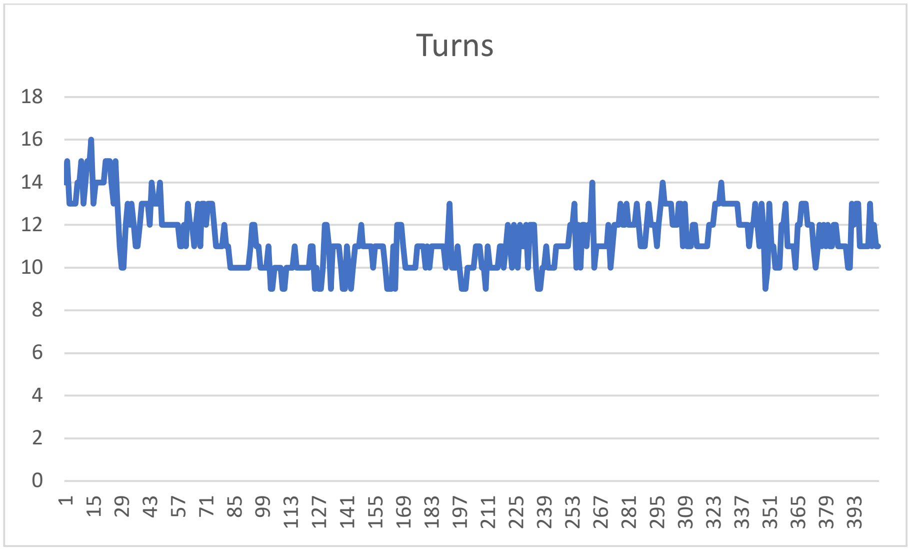

fitness
1
0.95
0.9
0.85
0.8
0.75
1
16
31
46
61
76
91
106
121
136
151
166
181
196
211
226
241
256
271
286
301
316
331
346
361
376
391

Run 2:

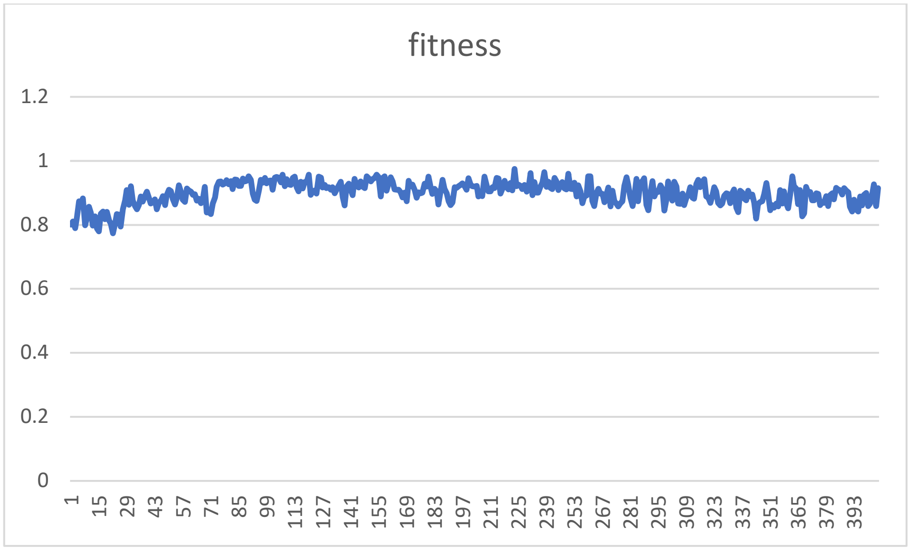

Infeasible Steps
35
30
25
20
15
10
5
0
1
16
31
46
61
76
91
106
121
136
151
166
181
196
211
226
241
256
271
286
301
316
331
346
361
376
391



Path Length
120
100
80
60
40
20
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393

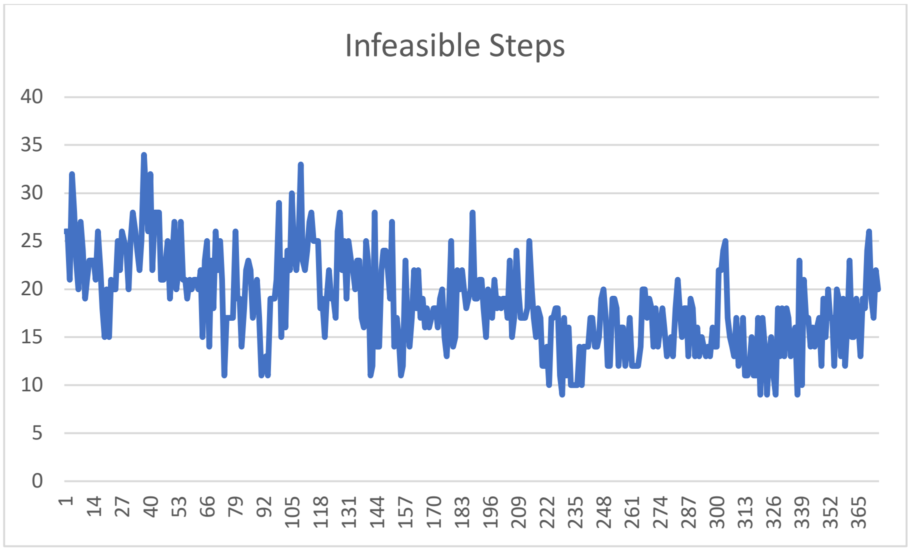


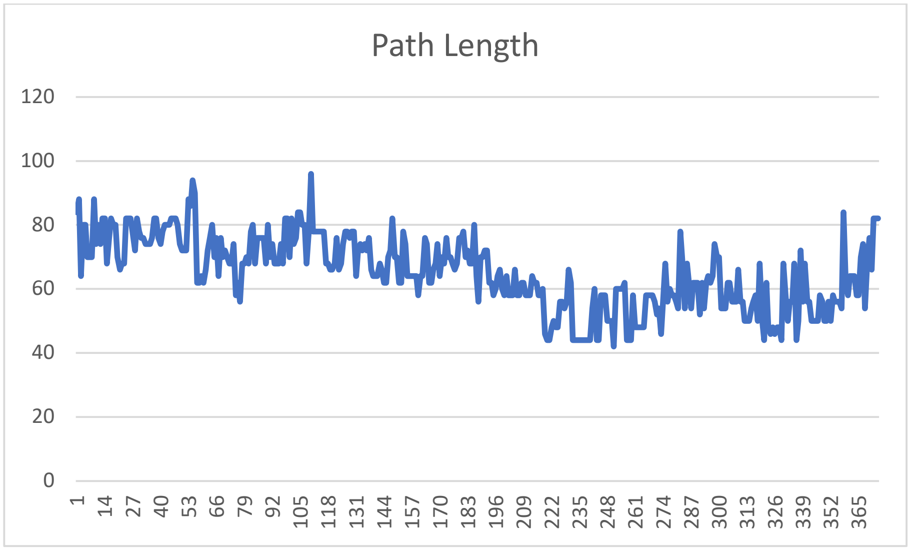
Turns
18
16
14
12
10
8
6
4
2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393

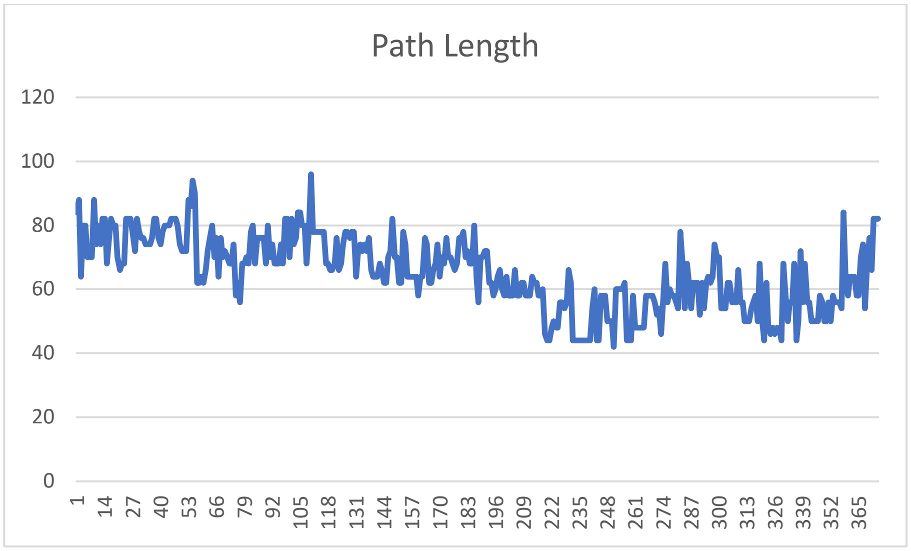


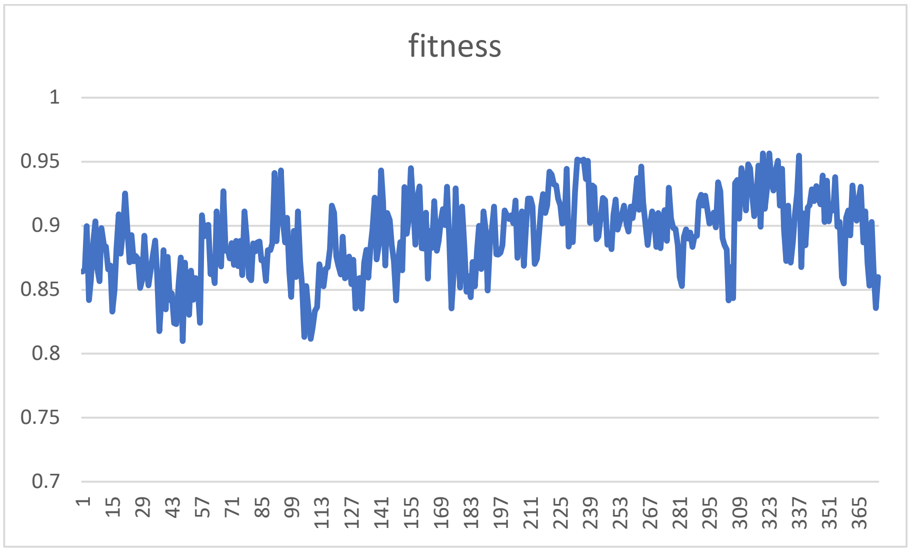
fitness
1.2
1
0.8
0.6
0.4
0.2
0
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365
379
393

Run 3:

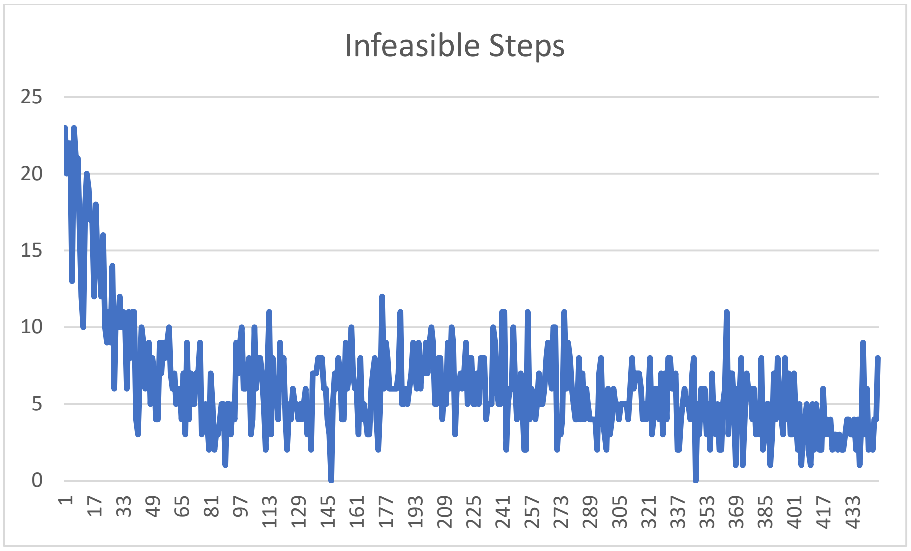

Infeasible Steps
40
35
30
25
20
15
10
5
0
1
14
27
40
53
66
79
92
105
118
131
144
157
170
183
196
209
222
235
248
261
274
287
300
313
326
339
352
365

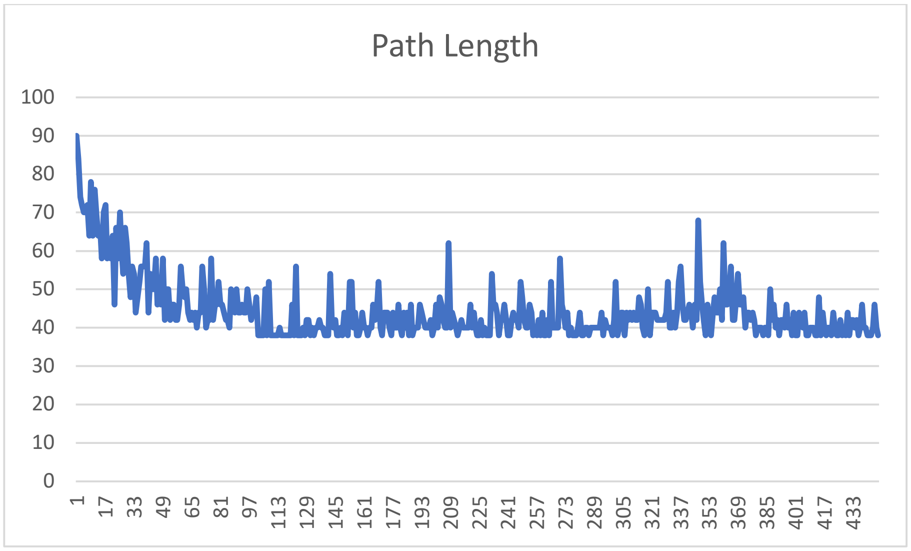

Path Length
120
100
80
60
40
20
0
1
14
27
40
53
66
79
92
105
118
131
144
157
170
183
196
209
222
235
248
261
274
287
300
313
326
339
352
365

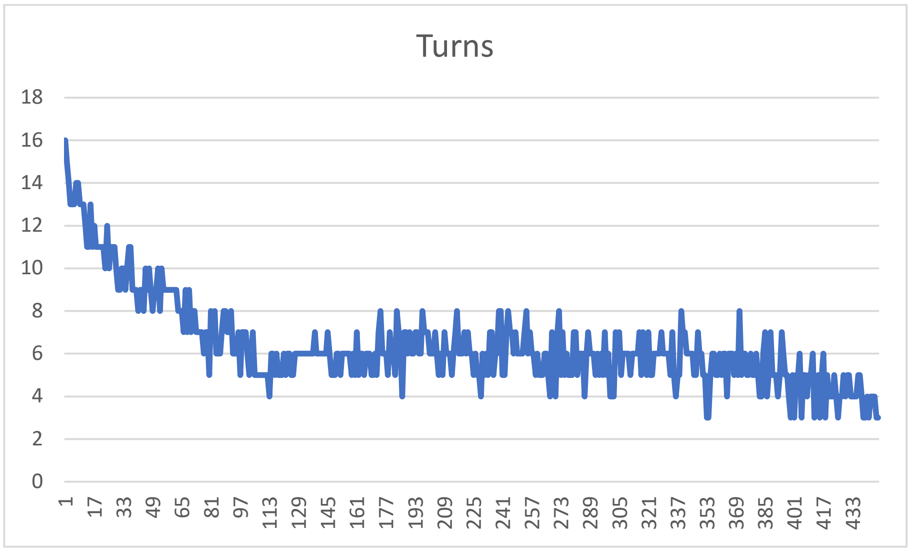

Path Length
120
100
80
60
40
20
0
1
14
27
40
53
66
79
92
105
118
131
144
157
170
183
196
209
222
235
248
261
274
287
300
313
326
339
352
365

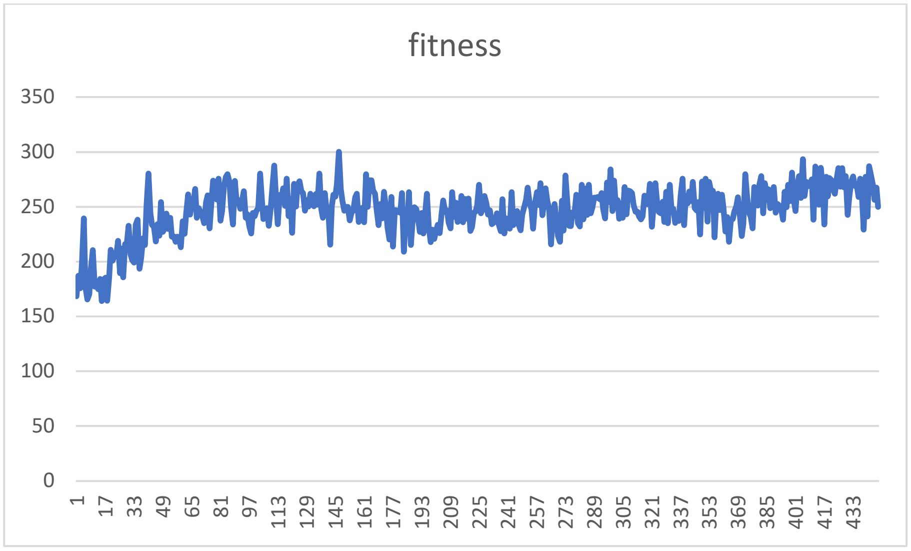

fitness
1
0.95
0.9
0.85
0.8
0.75
0.7
1
15
29
43
57
71
85
99
113
127
141
155
169
183
197
211
225
239
253
267
281
295
309
323
337
351
365

Run 4:


Infeasible Steps
25
20
15
10
5
0
1
17
33
49
65
81
97
113
129
145
161
177
193
209
225
241
257
273
289
305
321
337
353
369
385
401
417
433

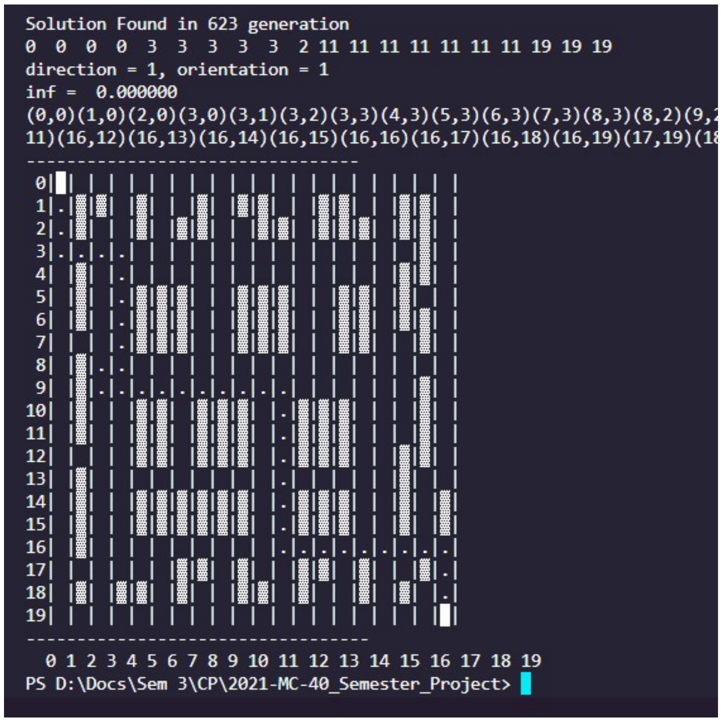

Path Length
100
90
80
70
60
50
40
30
20
10
0
1
17
33
49
65
81
97
113
129
145
161
177
193
209
225
241
257
273
289
305
321
337
353
369
385
401
417
433


Turns
18
16
14
12
10
8
6
4
2
0
1
17
33
49
65
81
97
113
129
145
161
177
193
209
225
241
257
273
289
305
321
337
353
369
385
401
417
433


fitness
350
300
250
200
150
100
50
0
1
17
33
49
65
81
97
113
129
145
161
177
193
209
225
241
257
273
289
305
321
337
353
369
385
401
417
433

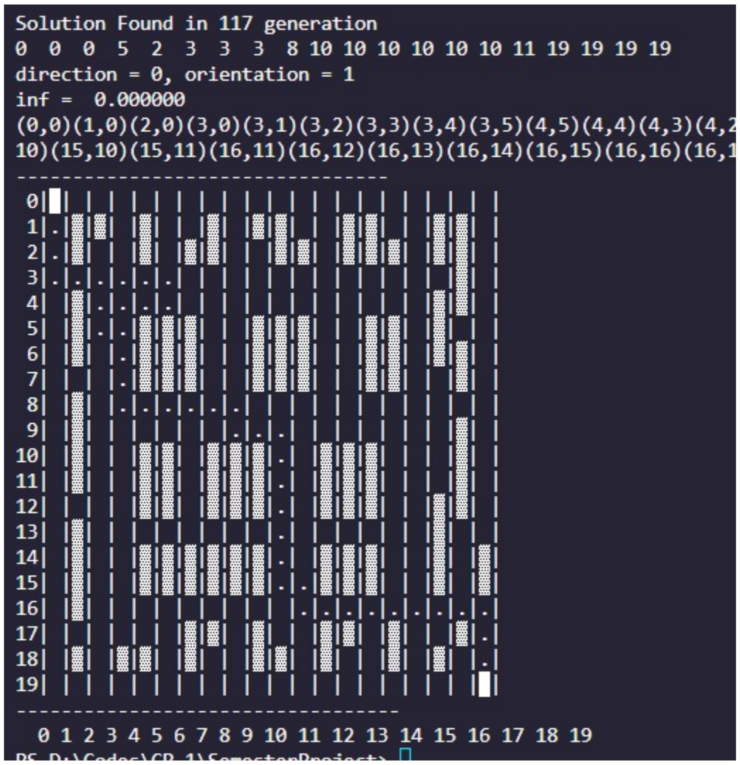
Possible Paths:

Population: **400** Iterations: **6000**



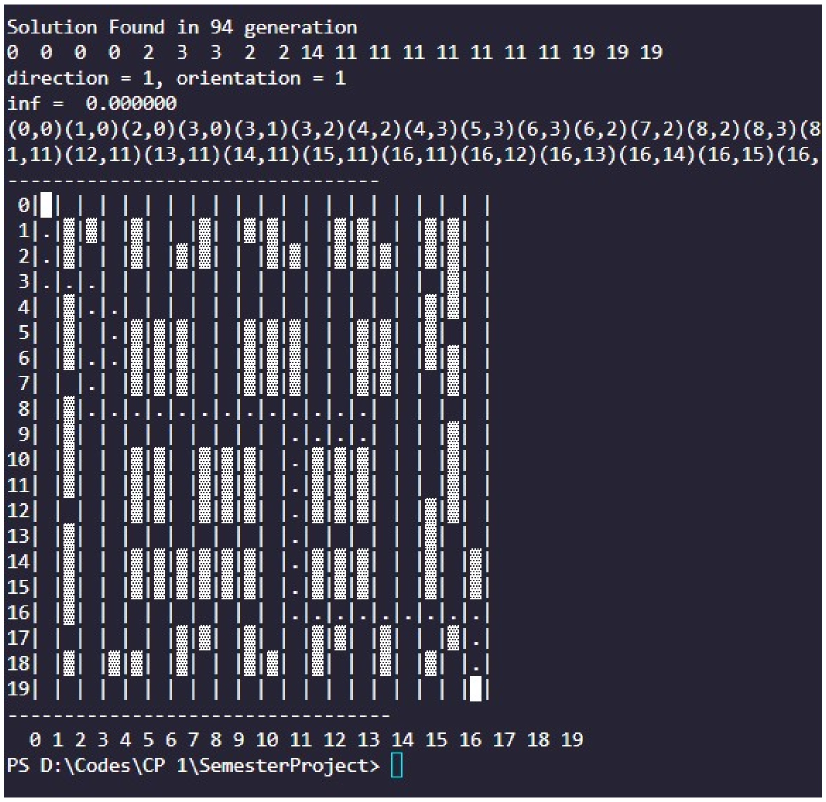
```
Solution Found in 623 generation
0  0  0  0  3  3  3  3  3  2 11 11 11 11 11 11 11 19 19 19
direction = 1, orientation = 1
inf =  0.000000
(0,0)(1,0)(2,0)(3,0)(3,1)(3,2)(3,3)(4,3)(5,3)(6,3)(7,3)(8,3)(8,2)(9,
11)(16,12)(16,13)(16,14)(16,15)(16,16)(16,17)(16,18)(16,19)(17,19)(1
---------------------------------
 0|█| | | | | | | | | | | | | | | | | | | |
 1|.|▒|▒| |▒| | |▒| |▒|▒| | |▒|▒| | |▒|▒| |
 2|.|▒| | |▒| |▒|▒| | |▒|▒| |▒|▒|▒| |▒|▒| |
 3|.|.|.|.| | | | | | | | | | | | | | |▒| |
 4| |▒| |.| | | | | | | | | | | | | |▒|▒| |
 5| |▒| |.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| |▒| | |
 6| |▒| |.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| |▒|▒| |
 7| | | |.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| | |▒| |
 8| |▒|.|.| | | | | | | | | | | | | | | | |
 9| |▒|.|.|.|.|.|.|.|.|.|.| | | | | | |▒| |
10| |▒| | |▒|▒| |▒|▒|▒| |.|▒|▒|▒| | | |▒| |
11| |▒| | |▒|▒| |▒|▒|▒| |.|▒|▒|▒| | | |▒| |
12| | | | |▒|▒| |▒|▒|▒| |.|▒|▒|▒| | |▒|▒| |
13| |▒| | | | | | | | | |.| | | | | |▒| | |
14| |▒| | |▒|▒|▒|▒|▒|▒| |.|▒|▒|▒| | |▒| |▒|
15| |▒| | |▒|▒|▒|▒|▒|▒| |.|▒|▒|▒| | |▒| |▒|
16| |▒| | | | | | | | | |.|.|.|.|.|.|.|.|.|
17| | | | | | |▒|▒| |▒| | |▒|▒| |▒| | |▒|.|
18| |▒| |▒|▒| |▒| | |▒|▒| |▒| | |▒| |▒| |.|
19| | | | | | | | | | | | | | | | | | | |█|
----------------------------------
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
PS D:\Docs\Sem 3\CP\2021-MC-40_Semester_Project>
```

Population: **2000** Iterations: **4000**



```
Solution Found in 117 generation
0  0  0  5  2  3  3  3  8 10 10 10 10 10 10 11 19 19 19 19
direction = 0, orientation = 1
inf =  0.000000
(0,0)(1,0)(2,0)(3,0)(3,1)(3,2)(3,3)(3,4)(3,5)(4,5)(4,4)(4,3)(4,
10)(15,10)(15,11)(16,11)(16,12)(16,13)(16,14)(16,15)(16,16)(16,
---------------------------------
 0|█| | | | | | | | | | | | | | | | | | | |
 1|.|▒|▒| |▒| | |▒| |▒|▒| | |▒|▒| | |▒|▒| |
 2|.|▒| | |▒| |▒|▒| | |▒|▒| |▒|▒|▒| |▒|▒| |
 3|.|.|.|.|.|.| | | | | | | | | | | | |▒| |
 4| |▒|.|.|.|.| | | | | | | | | | | |▒|▒| |
 5| |▒|.|.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| |▒| | |
 6| |▒| |.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| |▒|▒| |
 7| | | |.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| | |▒| |
 8| |▒| |.|.|.|.|.|.| | | | | | | | | | | |
 9| |▒| | | | | | |.|.|.| | | | | | | |▒| |
10| |▒| | |▒|▒| |▒|▒|▒|.| |▒|▒|▒| | | |▒| |
11| |▒| | |▒|▒| |▒|▒|▒|.| |▒|▒|▒| | | |▒| |
12| | | | |▒|▒| |▒|▒|▒|.| |▒|▒|▒| | |▒|▒| |
13| |▒| | | | | | | | |.| | | | | | |▒| | |
14| |▒| | |▒|▒|▒|▒|▒|▒|.| |▒|▒|▒| | |▒| |▒|
15| |▒| | |▒|▒|▒|▒|▒|▒|.|.|▒|▒|▒| | |▒| |▒|
16| |▒| | | | | | | | | |.|.|.|.|.|.|.|.|.|
17| | | | | | |▒|▒| |▒| | |▒|▒| |▒| | |▒|.|
18| |▒| |▒|▒| |▒| | |▒|▒| |▒| | |▒| |▒| |.|
19| | | | | | | | | | | | | | | | | | | |█|
----------------------------------
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
PS D:\Codes\CP 1\SemesterProject>
```

Population: **2000** Iterations: **4000**



```
Solution Found in 94 generation
0  0  0  0  2  3  3  2  2 14 11 11 11 11 11 11 11 19 19 19
direction = 1, orientation = 1
inf =  0.000000
(0,0)(1,0)(2,0)(3,0)(3,1)(3,2)(4,2)(4,3)(5,3)(6,3)(6,2)(7,2)(8,2)(8,3)(8
1,11)(12,11)(13,11)(14,11)(15,11)(16,11)(16,12)(16,13)(16,14)(16,15)(16,
---------------------------------
 0|█| | | | | | | | | | | | | | | | | | | |
 1|.|▒|▒| |▒| | |▒| |▒|▒| | |▒|▒| | |▒|▒| |
 2|.|▒| | |▒| |▒|▒| | |▒|▒| |▒|▒|▒| |▒|▒| |
 3|.|.|.| | | | | | | | | | | | | | | |▒| |
 4| |▒|.|.| | | | | | | | | | | | | |▒|▒| |
 5| |▒| |.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| |▒| | |
 6| |▒|.|.|▒|▒|▒| | |▒|▒|▒| | |▒|▒| |▒|▒| |
 7| | |.| |▒|▒|▒| | |▒|▒|▒| | |▒|▒| | |▒| |
 8| |▒|.|.|.|.|.|.|.|.|.|.|.|.|.| | | | | |
 9| |▒| | | | | | | | | |.|.|.|.| | | |▒| |
10| |▒| | |▒|▒| |▒|▒|▒| |.|▒|▒|▒| | | |▒| |
11| |▒| | |▒|▒| |▒|▒|▒| |.|▒|▒|▒| | | |▒| |
12| | | | |▒|▒| |▒|▒|▒| |.|▒|▒|▒| | |▒|▒| |
13| |▒| | | | | | | | | |.| | | | | |▒| | |
14| |▒| | |▒|▒|▒|▒|▒|▒| |.|▒|▒|▒| | |▒| |▒|
15| |▒| | |▒|▒|▒|▒|▒|▒| |.|▒|▒|▒| | |▒| |▒|
16| |▒| | | | | | | | | |.|.|.|.|.|.|.|.|.|
17| | | | | | |▒|▒| |▒| | |▒|▒| |▒| | |▒|.|
18| |▒| |▒|▒| |▒| | |▒|▒| |▒| | |▒| |▒| |.|
19| | | | | | | | | | | | | | | | | | | |█|
----------------------------------
  0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
PS D:\Codes\CP 1\SemesterProject> 
```