Robot Evacuation

Margarita Otochkina 100950500

COMP 4001,Dec 12, 2016

Outline

1. Introduction
2. Background
3. Implementation
4. Result
5. Conclusion

1 Introduction

Search algorithms have a long history in mathematic and computer science. The search problem discussed in this paper involves the evacuation of robots from a given environment. Robots are placed in a circle at certain position. The goal for robots is to reach an exit, located at the circumference of the circle. The project is broken into two parts, where the first part requires an implementation of a software visualizing the evacuation, and the second part evaluate the results. Section 2 gives a brief background for a problem, including a description of the environment. Implementation of the project is described in section 3. The implementation includes description of the program and algorithms solving the problem. Section 4 discusses the algorithms' execution time. Software calculates the execution time for the number of trials, and saves it to a text file. The last section draws the conclusion based on the results and describes possible future improvement to the algorithm and software.

2 Background

The project, stated in "Evacuating Two Robots from Multiple Unknown Exists in a Circle", examines the impact of the communication model used in solving the evacuation problem in distributed computing.

Let's consider the problem: Two robots, with the same max speed, are initially located at a certain location inside the disk with radius r. The goal is for both robots to evacuate though the exit, in the boundary of the disk, in the most optimal time. The exit location is unknown. By assumption, robots can communicate wirelessly with each other at any time and distance.

The project is splitted into two parts: The first part requires the implementation of a software visualizing the evacuation process. The second part analyses the evacuation time, where the experiment is repeated n number of times. For each trial, the project requires the average time and evaluation of the results.

The paper considers three scenarios:

1. Robots are initially located at the center of the disk.
2. One robot is placed at the center of the disk, while the other robot is placed at a random position inside the disk.
3. Both robots are placed at random positions in the interior of the disk.

The simulation for the first part vusializes all three scenarios. The second part calculates:

1. The worst-case evacuation time.
2. The expected (average) evacuation time.

for each algorithm, and provides the execution time for different number of trials.

The application, called "Robot Evacuation" is implemented using Python and Pygame. Python is a high-level programming language. Its syntax expresses concepts in fewer lines of code comparing to other languages, like C++. Pygame is a cross-platform set of Python modules designed for writing video games. The library includes computer graphic and sound libraries. Pygame allows to implement the first part of the project in Python with simple and clear user interface (UI). The implementation, including algorithm, is described in the next section.

3 Implementation

The section describes the implementation of first part of the project, including algorithms. First, the section provides algorithms for each situation, then, the section describes software implementation.

3.1 Algorithms

The project requires three different algorithms, that are based on both requirements and the environment described in section 2.

Algorithm 1 Evacuation Algorithm 1

```
1: procedure EVACUATION FROM ORIGIN O
2:     Both robots start at origin O
3:     r1 and r2 go to point A1 at the perimeter of the circle;
4:     while exit is not found do
5:         r1 goes in clockwise dir.
6:         r2 goes in counter clockwise dir.
7:     end while
8:     if r1 found the exit at B then
9:         r1 send location of B to r2
10:        r2 goes to B from C through the circle
11:    end if
12: end procedure
```

During the first situation, both robots are place at the center of the disk. The algorithm solving this case is given in Evacuation Algorithms lecture:

In algorithm 1, robots start at origin O. Both robots go to point A on the circumference of the disk. Once both robots reach A, one robot goes in clockwise direction around the disk, another robots goes in counter-clockwise direction. Both robots continue until the exit is found. When one of the robots reach the exit, located at point B, the robot sends a message with the exit location to the other robot. The other robot cuts though the disk reaching the exit. Once both robts find the exit, the algorithm terminates.

During the second situation, one robot is placed at the origin O of the disk, the second robot is placed at a random position inside the disk. The algorithm 2 calculates the location on the border both robots have to reach. The location is calculated using the angles, robots are facing at. If one robot reaches the location first, it waits for the second robot. When two robots reach the location A, they start looking for exit by walking along the border. One robot goes in clockwise direction; the other robot goes in a counter-clockwise direction. The robot reached the exit sends the location of the exit to the second robot. Finally, second robot cuts through the circle and reaches the exit.

The third algorithm solves the problem when two robots start at random locations inside the disk.

From their locations, both robots move to the point on disk's border calculated by algorithm. Once both robots reach the point, the algorithm proceeds in the same way, as algorithms 1 and 2.

The next subsection describes the implementation of listed algorithms using

Algorithm 2 Evacuation Algorithm 2

procedure EVACUATION ALGORITHM 2
    One robot start at origin O, second robot starts at random location
    r_1 and r_2 go to point A. A is located on disks border, at the angle that is a middle of r_1 and r_2 angles.
    while exit is not found **do**
        r_1 goes in clockwise dir.
        r_2 goes in counter clockwise dir.
    end while
    if r_1 found the exit at B **then**
        r_1 send location of B to r_2
        r_2 goes to B from C through the circle
    end if
end procedure

Algorithm 3 Evacuation Algorithm 3

procedure EVACUATION ALGORITHM 3
    Both robots start at random locations
    r_1 and r_2 go to point A. A is located on disks border, at the angle that is a middle of r_1 and r_2 angles.
    while exit is not found **do**
        r_1 goes in clockwise dir.
        r_2 goes in counter clockwise dir.
    end while
    if r_1 found the exit at B **then**
        r_1 send location of B to r_2
        r_2 goes to B from C through the circle
    end if
end procedure

Python and Pygame.

3.2 Implementation of Algorithms in Python and Pygame

User passes the origin coordinate and radius, 100 units, to algorithms. Algorithm 1 places robots at the origin. Algorithm 2 generates one robot at the origin, and one robot at random location. The algorithm 3 generates both robots at random locations. The exit is generated randomly using formulas:

$$x_{exit} = x_{origin} + radius \times cos(angle)$$
$$y_{exit} = y_{origin} - radius \times sin(angle)$$

where angle is an angle generated randomly using *random* function provided by Python. The random coordinates for robots are generated in the same way as exit, but instead of a radius, the algorithm generates a random value between 0 and radius.

For the algorithm 1 both robots move at degree 90 at the top point of the disk. The movement is achieved by removing one unit from y coordinate of each robot.

For algorithms 2 and 3, the algorithm generates a point on a disk border. The point is called the mid point, and is calculated using coordinates angles. The mid point has an angle equal to the angle between angle of robot 1 and angle of robot 2.

When both robots reach the mid point, they continue moving along the disk's border. For each point, algorithms generate a new coordinate by changing the angle. Depending on the direction, algorithms either remove certain value from original angle, or add it. For the speed 1, $thevalue = 0.57$. Coordinates are generated using the same formula as above.

When one of the robots reaches the exit, it sends the coordinate to the other robot. The other robot memorizes his current coordinate, and calculates the add value using formula:

$$1/\sqrt{(x_{exit} - x_{robot})^2 + (x_{exit} - y_{robot})^2}$$

The add value is the value added to a counter. The counter starts with 0 and ends with 1, when the exit is reached. During every step, the robot calculates its coordinates using:

$$x_{robot} = x_{robot_temp} + counter \times (x_{exit} - x_{robot_temp})$$
$$y_{robot} = y_{robot_temp} + counter \times (y_{exit} - y_{robot_temp})$$

3.3 Implementation of the Application

The application is called Robot Evacuation. It is implemented using files:

1. evacuation.py - execution file located in the main folder
2. alg1.py - code for the algorithm 1, located in lib folder.
3. alg2.py - code for the algorithm 2, located in lib folder.
4. alg3.py - code for the algorithm 3, located in lib folder.
5. app.py - main app file, contains code for UI and calls for all algorithms.
6. eztext.py - Text input module for pygame, licated in lib folder.

Lib folder also contains logs folder. Logs folder consist of 3 folders: each for each algorithm. Logs folder stores log files created after each execution. Each algorithm creates log file with time and date, number of executions performed during test, and the average execution time.

The application starts with a user interface (UI). The interface draws a disk with an origin marked, and five blue buttons. Three buttons on the right represent algorithms, If one of the buttons is selected, it changes its color to grey. Two other buttons start the application. Start button launches an execution of the selected algorithm. Start test button launches a test with the number of trials entered into the text input. Text input doesn't have to be selected: any number entered on the keyboard will be displayed.

Figure 1: Robot Evacuation: UI

In total, the application executes in the following order:

1. Evacuation file calls app.py file
2. App.py file generates applications interface.
3. If user selects one of of the algorithms, the app colors it grey and remembers the matching value.

4. If user presses Start button, the program starts one execution of the selected algorithm.
5. If user presses Start Test button, the program starts n number of executions of the selected algorithm.
6. During the execution, the app animates the evacuations of robots. Robots are represented as blue and purple dots; the exit is represented as a red dot.
7. After the execution, app sends a request to algorithm app, and the algorithm create a log file with time and the number of trials. The algorithm button goes back to blue state.

4 Result

Each algorithm has its own worst case and average case. By assumption, robots in the algorithm wait for each other if one robot is late for reaching the disk border. Radius in the algorithm is 100 units.
Algorithm 1

Both robots walk the distance r to reach the border. Then, at most each robot walk $\pi \times r$ number of steps. Finally, second robot crosses the distance of a chord:

$$2r \times 2sin(\theta/2)$$

Where the biggest value is $2r$ (diameter) and the smallest is 0.
Thus the formula for algorithm 1 is:

$$r + 4r \sin(\theta \div 2) + \pi r((b - \theta) \div 180)$$

where b = 180 if robots haven't walked π distance on the disk's border, and b = 360 if it did.

The worst case scenario is when robots crossed $rpi(\theta\ 180)$ circumference of a disk, where θ ranges between 110 and 130 degrees. In this case, the time it takes to reach the exit is:

$$1 + 4 \sin(\theta \div 2) + \pi((360 - \theta) \div 180)\ 8.639$$

for radius 1. An average case scenario would be :

$1 + 4\sin(\theta \div 2) + \pi((360 - \theta) \div 180)$ 8.639/2 4.2

The smallest possible time is:

$1 + 4\sin(\theta \div 2) + \pi((180 - \theta) \div 180) = r = 1$

Algorithm 2 and 3

It takes at most $2r$ for two robots to reach the border. The rest is the same as for the algorithm 1. Thus the formula for the time is:

$r + 4\sin(\theta \div 2) + \pi((b - \theta) \div 180)$

where r varies between r and $2r$ for algorithm 2 and between 0 and $2r$ for algorithm 3.

The worst case scenario for both Algorithms 2 and 3:

$2r + 4\sin(\theta \div 2) + \pi((360 - \theta) \div 180)$ 9.64

where θ is between 130 and 110.

An average case for both algorithms 2 and 3:

$r + 4\sin(\theta \div 2) + \pi((360 - \theta) \div 180)$ 8.639/2 4.2

with $r = 1$.

4.1 Results

Each algorithm has been executed 10, 25, 50, and 100 times. For each trial, the application calculated the average time.

Algorithm 1

1. Trials - 10; Time: 7.47916140556s
2. Trials - 25; Time: 6.54008516312s
3. Trials - 50: Time: 6.24802094936s
4. Trials - 100: Time: 4.15029038668s

5. Total average: 6.075s

Algorithm 2

1. Trials - 10; Time: 5.61222643852s
2. Trials - 25; Time: 7.40427503586s
3. Trials - 50: Time: 6.51540182114s
4. Trials - 100: Time: 6.91943815231s
5. Total: 6.6s

Algorithm 3

1. Trials - 10; Time: 6.37249093056s
2. Trials - 25; Time: 6.94276947021s
3. Trials - 50: Time: 6.40472808361s
4. Trials - 100: Time: 6.28656085253s
5. Total: 6.475s

As the results show, the average time is mostly between 6 and 7 seconds. The maximum average time is achieved with algorithm 1 during 10 trials. The lowest average time is achieved with algorithm 1 with 100 trials. The smallest result looks rather like an outliner caused by a line of best-case scenarios. The worst-case scenario for the algorithm 1 took 8.18438601494 seconds. For the other two algorithms, it's hard to measure the time due to the randomness. However, the achieved time appears to be slightly better than the one predicted.

5 Conclusion

While the first algorithm cannot be improved, the second and third can be. The future work would improve the time it takes for robots to reach the border, thus reducing the execution time.