Maze:
Genetic algorithm

Maciej Kieżel

Contents

1 Introduction

1.1 What is the project about?

Project 'Maze: Genetic algorithm' is intended to use the genetic athorythm to complete mazes set as a two dimensial binary arrays. I will check diffrences in results when we change fitness function or parameters of algorithm. I will verify also my own genetic algorithm and algorithm A*. (`https://en.wikipedia.org/wiki/A*_search_algorithm`)

Needed software:

- Python 3 default packages
  - os
  - time
  - math
  - random
- additional packages
  - numpy
  - pyeasyga
  - matplotlib

NOTE

Time tests of small mazes was made on notebook:

- i5 4210h
- 8gb ram ddr3 - dual channel

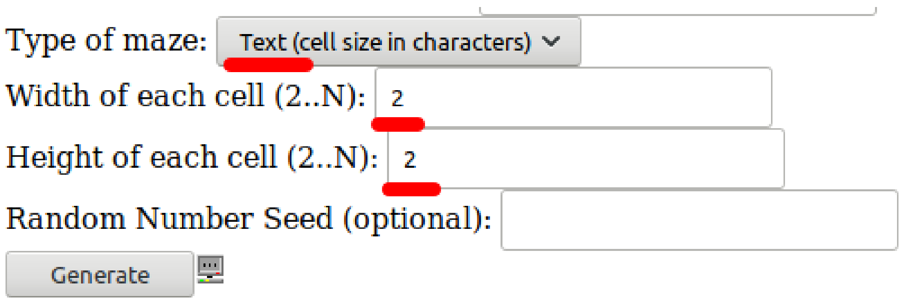
Test of megamaze was made on PC:

- ryzen 5 2600
- 16gb ram ddr4 - dual channel

Both computers were set up on Linux Mint 19.2 MATE, but differences in hardware makes this tests incomparable.

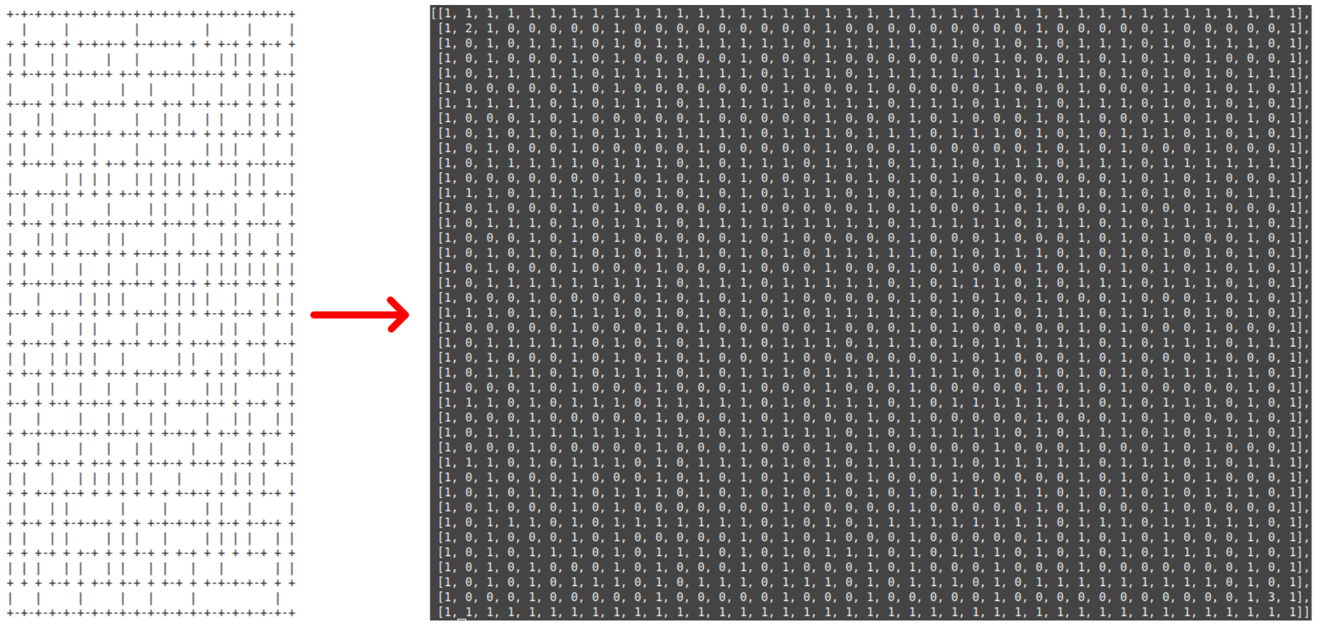
1.2 Download and conversion of mazes

I generated mazes with `http://www.delorie.com/game-room/mazes/genmaze.cgi`. The settings are presented on the picture below.



Type of maze: Text (cell size in characters)
Width of each cell (2..N): 2
Height of each cell (2..N): 2
Random Number Seed (optional):

To converse from text version to binary array i created script convert.py. It took output of maxe generator and returned it as python array.



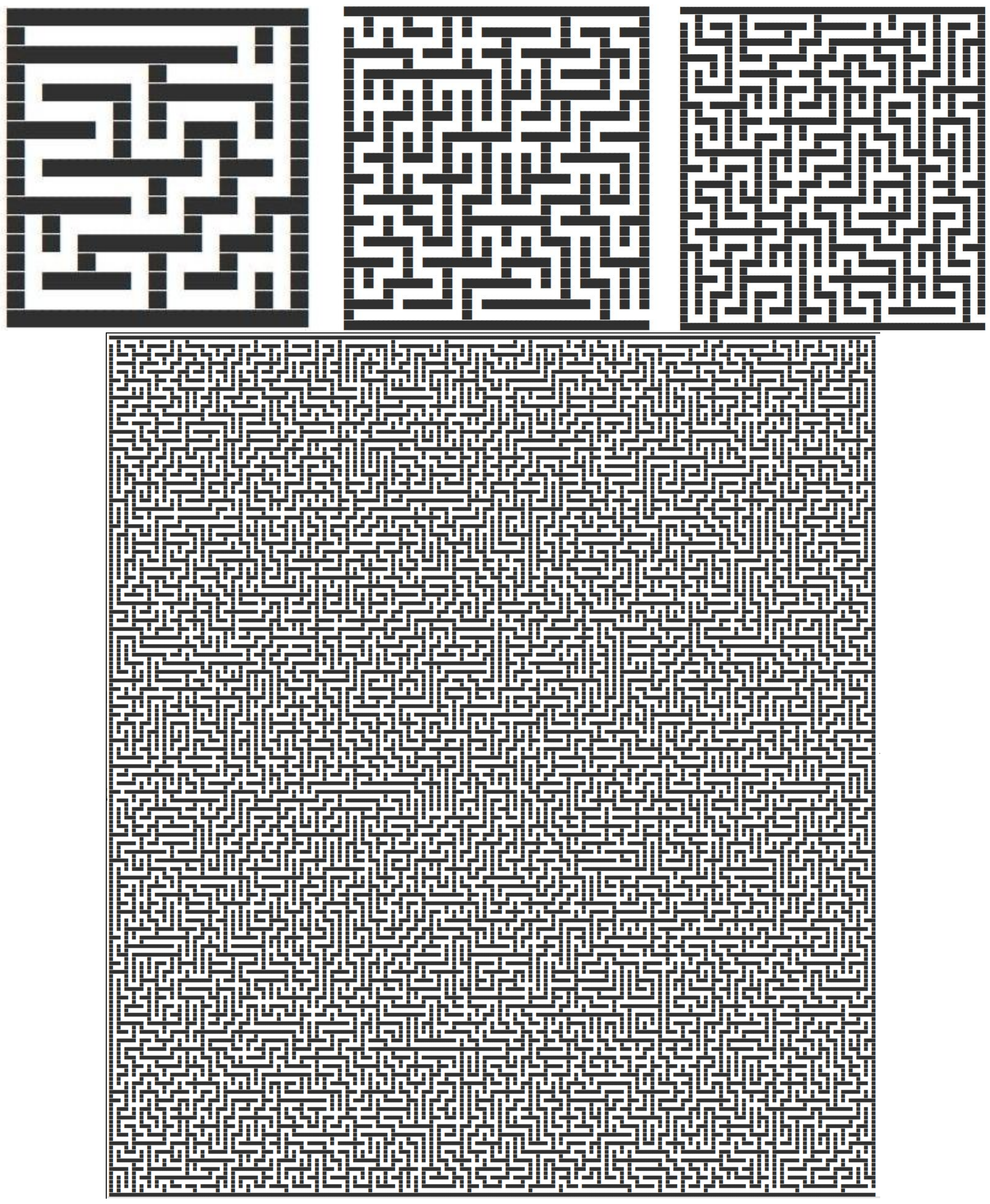
```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  |     |         |         |     |     |
+ + +-+ + +-+-+-+ +-+-+-+ + + +-+ + +-+ +
| |   | |     |   |       |   | | | |   |
+ +-+-+ +-+-+-+ +-+ +-+-+-+-+-+ + + + +-+
|     | |       |   |     |   |   | | | |
+-+-+ + +-+ +-+-+ +-+ +-+ +-+ +-+ + + + +
|   | |     |     |   | |   | |   | | | |
+ + + + +-+-+-+ +-+ +-+ +-+ + + +-+ + + +
| |   |     |     |   |     | | |   |   |
+ +-+-+ +-+ + +-+ +-+ +-+ +-+ +-+ +-+-+-+
|       | | | |   | | | | |     | | |   |
+-+ +-+-+ + + + +-+ + + + + +-+ + + + +-+
| |   | |     |     | |   | |   |   |   |
+ +-+ + +-+ +-+-+-+-+ +-+-+ +-+ + +-+-+ +
|   | | |     | |     |   |   | | |   | |
+ + + + + +-+ + + +-+-+ + +-+ + + + + + +
| |   |   |   |   |   | |   | | | | | | |
+ +-+-+-+-+ +-+ +-+-+ + +-+ + +-+ +-+ + +
|   |     | | | |     | | | |   |   | | |
+-+ + +-+ + + + + +-+-+ + + +-+ +-+ + + +
|     |   | |     |   | |     | |   |   |
+ +-+-+ + + +-+ +-+ +-+ + +-+-+ + +-+ +-+
| |   | | | |   |       | |   | |   |   |
+ +-+ + +-+ + +-+ +-+-+-+ + + + + +-+-+ +
|   | |   |   |   |   |     | | |     | |
+-+ + +-+ +-+-+ + +-+ + +-+-+-+ + +-+ + +
|   |     |   | |   | |     |   | |   | |
+ +-+-+-+-+-+ +-+-+ + +-+-+ + +-+ + +-+ +
|   |     |   |   | |     |   |   | |   |
+-+ + +-+ + +-+ + + +-+-+ +-+-+ +-+ + +-+
| |   |   | | | | | |   |     | | | |   |
+ + +-+ +-+ + + + + + + +-+-+ + + +-+ + +
| |   | |       |     |     | |   |     |
+ +-+ + +-+-+-+ + + +-+-+-+-+ +-+ +-+-+ +
| |   | |     | | |   |     | | | |   | |
+ + +-+ + +-+ + + +-+ + +-+ + + + +-+ + +
| | |   | |   | |   | |   |   |       | |
+ + + +-+ + +-+ +-+ + +-+ + +-+-+-+-+ + +
|   |     |     |   |     |           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
[[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 2, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1],
 [1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1],
 [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1],
 [1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1],
 [1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1],
 [1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1],
 [1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1],
 [1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1],
 [1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
 [1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1],
 [1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1],
 [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
```

1.3 Mazes

In tests I used mazes similar to these:

- 15x15
- 20x20
- 40x40
- 200x200



Small mazes and megamaze 200x200.

2 Used algorithms

2.1 Genetic algorithms

Fitness skip

In this version of algorithm fitness function skip moves, which consequence is running into a wall. This kind of fitness function causes longer chromosomes, to avoid exception, when skipped moves causes ending of the program too soon.

Fitness crash

The fitness crash algorithm is based on finishing the action, every time you enter the wall of the maze. This function returns the distance to the exit after last step before crash. It is relatively fast, because the vast majority of passes do not reach even half the length of the chromosome.

My algoritm

I've noticed that the worst for genetic algorithms are labyrinths that look like a 'squeezed sinusoidal'. To try to minimize this problem I decided to write an algorithm that goes through the labyrinth with a genetic algorithm with fitness skip function twice. First from the start, writing down all the points he visited. Then the algorithm starts from the end of the maze retreating and creating a second board with the points reached from the finish. At the end, we compare the received lists and at least one common coordinate on the lists indicates the passage of the labyrinth.

2.2 Algorithm A*

The A* algorithm is a complete algorithm, which means it will pass the maze every time it is possible. It is also a heuristic algorithm, so it is trying to make your future work easier by choosing a 'better' path. Considering our labyrinths, the algorithm has always chosen the path in order:

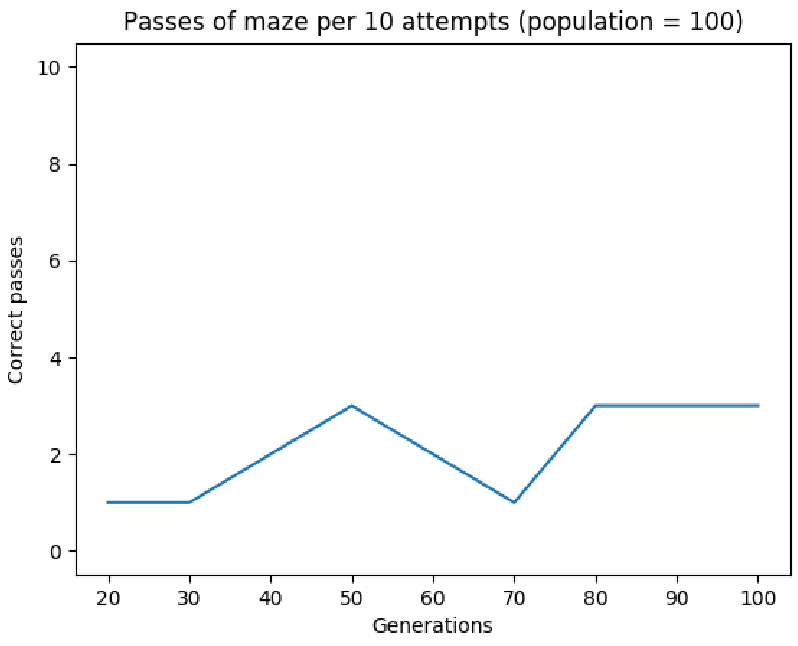
- DOWN
- RIGHT
- LEFT
- UP

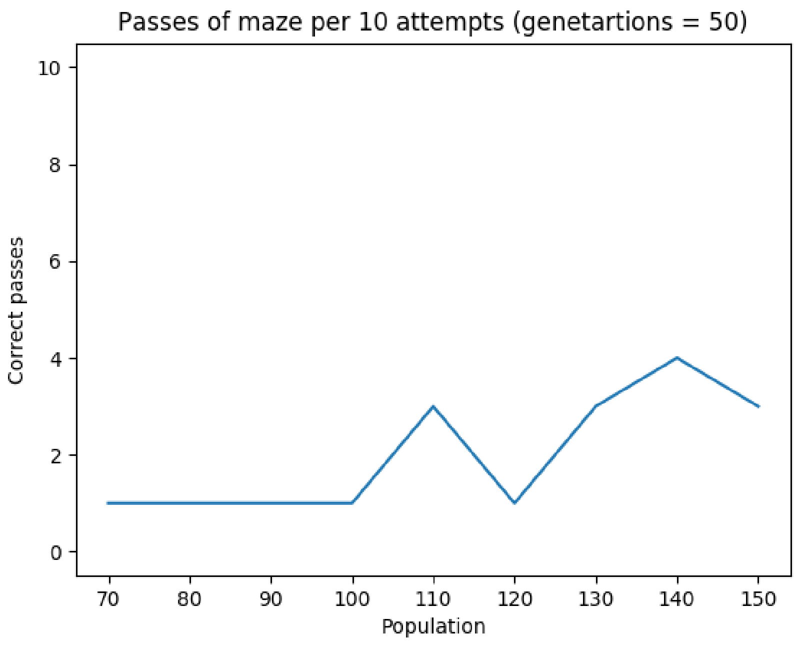
Note that all the labyrinths from our generator have a start in the upper left corner and an end in the lower right. Choosing this order allows us to skip the step where we have to compare the quality of the available steps. As we go through the labyrinth we change the places where we stand behind the visited, and as we retreat we 'build' fields behind us. This algorithm allows you to pass the labyrinth very quickly.

3 Experiments

3.1 Optimal parameters

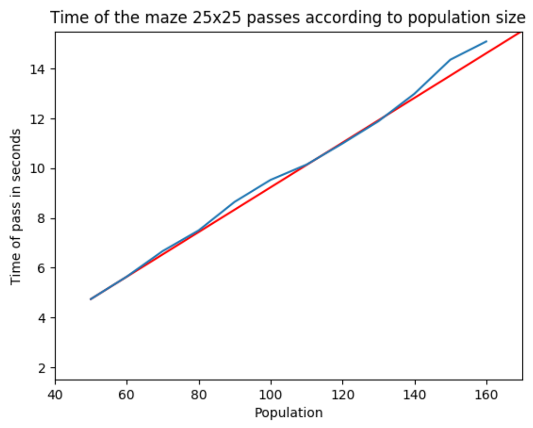
The first experiment was to investigate the influence of genetic algorithm parameters on labyrinth passages. Based on the results, it is easy to conclude that genetic algorithms are very random, even with a larger population and with more generations, the most depends on the initial population drawn. In the case of a bad draw, we can only count on mutations.

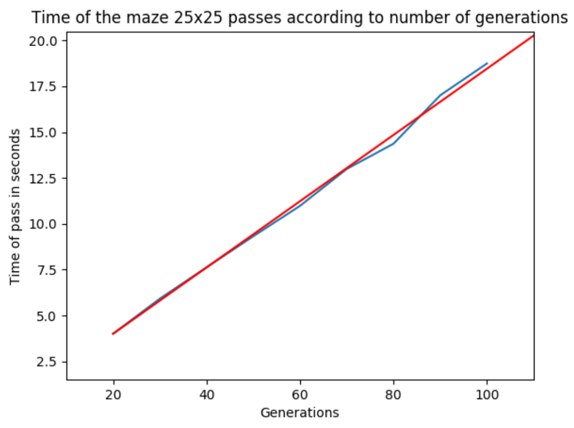




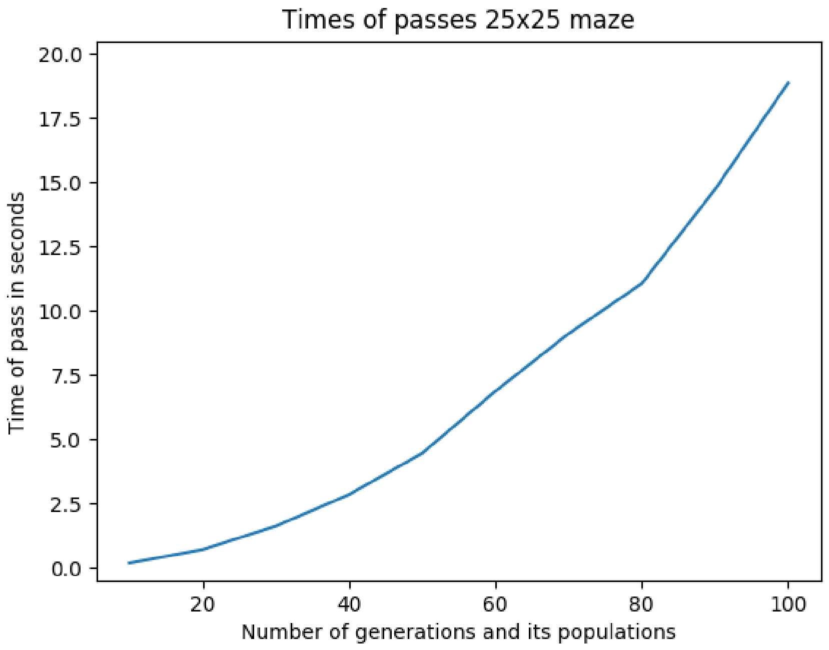
3.2 Impact of parameters on operating time

Now let's check the impact of the parameters on algorithm runtime. On the graphs I additionally inserted a red line connecting the first measurement with an additional measurement with a much higher value of the parameter being checked.



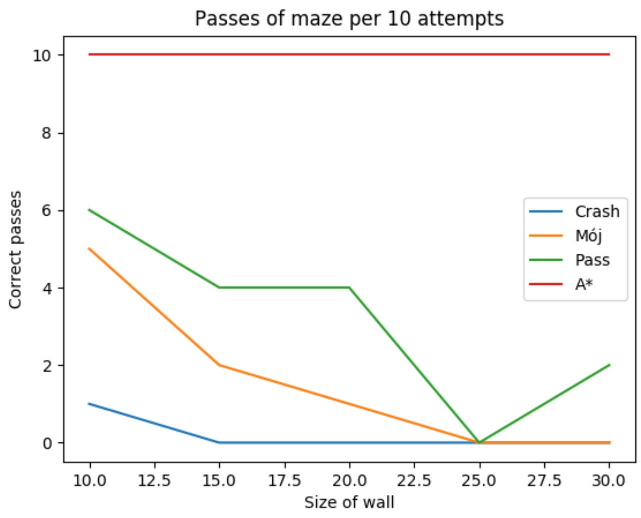


So we see that in both cases the time needed to complete the labyrinth grows linearly and will maintain this trend in the future. Now let's see if we can simultaneously increase the size of the population and the number of generations and get a plot similar to the parabola.



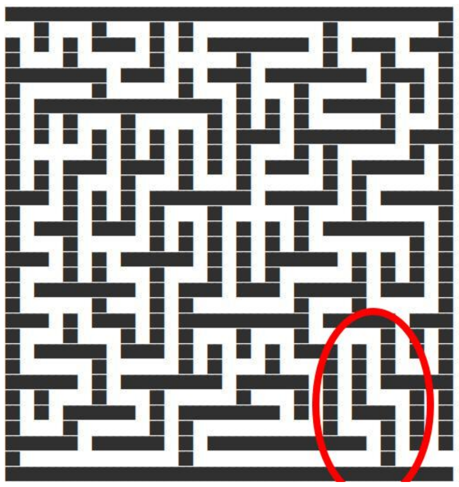
A time chart that looks like we expected.

3.3 Comparison of algorithm performance



Results of tests.

I also made a test by generating 5 labyrinths from dimensions 10x10 to 30x30. You can immediately see that with the crash algorithm the effectiveness is almost zero. So I'll skip it in later tests. However, as the graph shows, the passage of the labyrinth mainly depends on its structure, and genetic algorithms have a problem with going back to change the path.

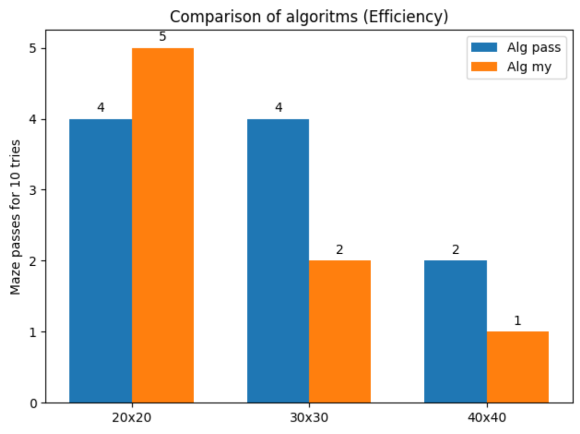


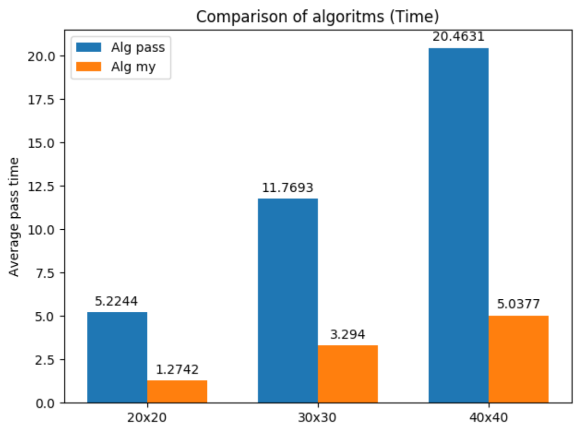
An example of a place in a maze that makes it almost impassable for genetic algorithms.

3.4 Algirthm pass vs my

From the previous experiment, the only competing algorithms are pass and mine. Let's compare them more closely. For pass we will use a population of 100 and 50 generations, in case of my algorithm the population will be the same, but on each side there will be only 25 generations.

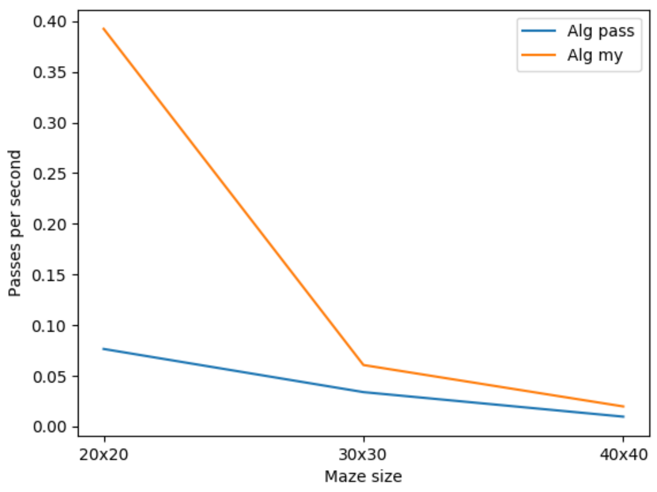
3.4.1 Results and time





3.4.2 Ratio results to time

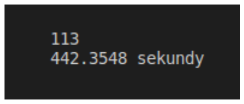
Looking at the graphs below I decided to chart the correct transitions per second to see the effectiveness of the algorithms over time.



Considering the time, in each of the labyrinths, my algorithm scored better than a standard genetic algorithm.

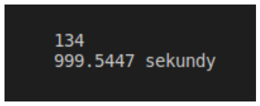
3.5 Test of megamaze

Finally, I decided to test the algorithms in megalabyrinth. The results show the disproportion between the A* algorithm and all genetic algorithms I have tried.



```
113
442.3548 sekundy
```

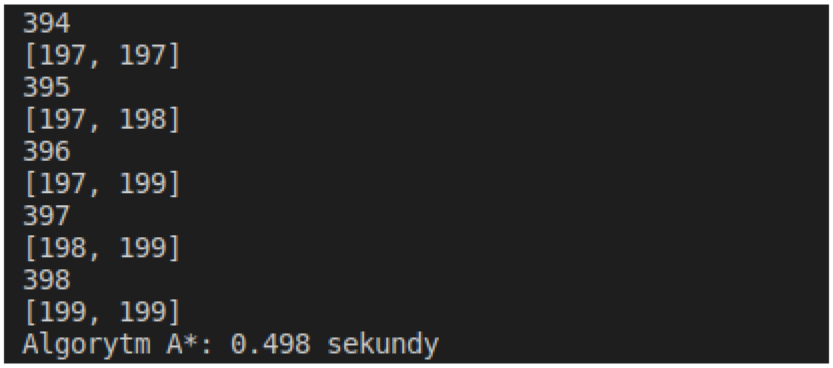
population: 100
population: 50



```
134
999.5447 sekundy
```

generations: 150
generations: 75

The above mentioned times only allowed the labyrinth to pass to more than half, and they are over 7 and 15 minutes. Add to this the fact that we are not sure if we are going down the right path, and it can be considered that genetic algorithms are not the best solution in this case.



```
394
[197, 197]
395
[197, 198]
396
[197, 199]
397
[198, 199]
398
[199, 199]
Algorytm A*: 0.498 sekundy
```

Note that the A* algorithm was able to complete the labyrinth correctly in less than half a second. That's far less than one percent of the time for a faster version of genetic algorithm.

4 Summary

The A* algorithm outclasses genetic algorithms in every way. You can spend time researching, checking and improving genetic algorithms to improve them, as I have shown in this work. However, I don't think they ever come close to the level of the A* algorithm.