Genetic algorithms in mazes

A comparative study of the performance for solving mazes between genetic algorithms, BFS and DFS.

ANTON JONASSON, SIMON WESTERLIND

SUPERVISOR: PAWEL HERMAN
Genetiska algoritmer i labyrinter

En jämförande studie av prestandan för lösande av labyrinter mellan genetiska algoritmer, BFS och DFS.

ANTON JONASSON, SIMON WESTERLIND

SUPERVISOR: PAWEL HERMAN
Abstract
In this report, genetic algorithms were compared to BFS and DFS for solving mazes with regards to computational time and solution path length. This was done by creating mazes with Wilson’s algorithm and running several tests in a controlled environment. The solution path length found by the genetic algorithm was in general better than the DFS and often almost as good as the one found by the BFS. The computational time needed by the genetic algorithm however was significantly worse than the time needed by both the DFS and the BFS.
Sammanfattning

I denna rapport har genetiska algoritmer jämförts med BFS och DFS för att lösa labyrinter med avseende på beräkningstid och lösningslängd. Detta gjordes genom att skapa labyrinter med Wilsons algoritm och köra en stor mängd tester i en kontrollerad miljö. Lösningslängderna som den genetiska algoritmen fann var i allmänhet bättre än de av DFS och ofta nästan lika bra som de som BFS fann. Den genetiska algoritmens beräkningstid var dock betydligt längre än den tid som krävdes för både DFS och BFS.
Contents

1 Introduction 1
  1.1 Scope of the report .............................. 3
  1.2 Outline of the report ........................ 3

2 Background 4
  2.1 Mazes ........................................... 4
  2.1.1 Generating mazes ............................ 4
  2.1.2 Solving mazes ............................... 5
  2.1.3 Genetic algorithms ......................... 7

3 State of the Art 9
  3.1 GAs and maze solving robots .................. 9
  3.2 Optimizing genetic algorithms for solving mazes 11
    3.2.1 Improved segment crossover ............... 11
    3.2.2 Change last operator, remove loop operator and out of dead end operator 12
    3.2.3 Fitness condition ........................ 12
  3.3 BFS and DFS in mazes .......................... 12

4 Method 13
  4.1 Maze generation .................................. 13
  4.2 BFS and DFS .................................... 13
  4.3 GA .............................................. 14
  4.4 Test program ................................... 14

5 Results 15
  5.1 Tests on mazes with diagonally opposed source and destination cells ......................... 15
  5.2 Tests on mazes with randomly selected source and destination cells 21

6 Discussion 28
  6.1 Summary of the results .......................... 28
  6.2 Discussion of the method ........................ 29
  6.3 Discussion of the reviewed studies ............ 29
Chapter 1

Introduction

With the increased utilization of big data and the boundaries of what is feasible expanding, we find more complex problems containing larger amounts of data than ever before. An important area of problems is that of optimization problems. Not every optimization problem has a solution that is easily reached through careful consideration. In some cases, it is only known what makes for a good solution, but not how to reach it. This is where genetic algorithms (GA) can step in to provide an alternative approach. Using GAs it is possible to reach a good solution through a process of evolutionary trial and error even if the path to that solution is unknown beforehand [BMB93] [Cha98]. This process has been used in real life applications such as designing antennas used by NASA on satellites. NASA knew what made for an efficient antenna, but did not know how to the design the antenna for it to have these important properties. They therefore used GAs to develop the antenna through evolutionary iterations and ended up finding an effective design [LHL05].

A large amount of research has been conducted on the use of GAs for solving optimization problems, for example, [CS96] developed a GA approach to solving the known NP-Hard problem redundancy allocation. GAs have also been used to solve path finding problems within mazes. In 2004 [GM04] showed that a GA approach can be used to find paths from any point in mazes of varying sizes and complexity to a set goal, however the shortest path was not always found.

Mazes are closely connected with modern graph theory, since a maze can easily be represented by a graph where each vertex represent a cell in the maze, and each edge represents a connection between cells as can be seen in figure 1.1 and 1.2. This allows us to use path finding algorithms for mazes to solve path problems in graph theory.
Figure 1.1: Simple maze.

Figure 1.2: Graph conceptualized from the maze.
The study of graphs is an exciting field with many possibilities. Graphs can be used to investigate problems of many different subjects [Bol13]. For example, graphs could be constructed to simulate a network of websites, a network of roads in a city or a network of transportation between countries. This allows us to solve a number of problems in real life by applying algorithms for graph problems.

In their report, [Sad+10] investigates the effectiveness of using graph theory to solve mazes. Their case is based upon the micro mouse competition held by IEEE, where contestants design algorithms for moving a robot along a maze. They propose that graph theory based algorithms dominate non graph theory based algorithms in completing this task. When using graph theory to solve a maze, they propose two methods, depth first search (DFS) and breadth first search (BFS).

GAs have been shown through extensive studies to be quite effective at solving mazes [PE] [GM04]. However, the effectiveness of the GAs has not to the best of our knowledge been compared to BFS, DFS or any other maze solving algorithms. This is where this report aspires to contribute, by making sound comparisons between GAs and BFS and DFS with regards to solving mazes. It is our aspiration that the GA will show to compliment the BFS and DFS by being faster than the BFS while still constructing a better solution than the DFS. Focus of the tests will be regarding computational time and the length of the solution, and since the GA is not guaranteed to find a solution the number of times it fails will also be regarded.

1.1 Scope of the report

This report will investigate how well GAs compare to DFS and BFS, for finding a path through a non-perfect, two dimensional, normal, orthogonal maze. The report will not go further in depth on the many other ways for finding solutions to a maze, but simply try and benchmark the effectiveness of GAs in relation to these standard ways of finding paths in mazes.

1.2 Outline of the report

Firstly this report will provide an introduction to the concept of mazes as a depiction of graph theory. Secondly the report will introduce an algorithm for creating mazes, as well as further explain the concepts of using BFS, DFS and GAs to solve mazes. This report then will then go on to present the method of testing that will be applied to these algorithms, in order to benchmark the GA in comparison to the BFS and DFS in maze solving, as well as present the test results. Lastly a discussion will be held with regards to these tests, and GAs role in maze solving.
Chapter 2

Background

In this chapter what constitutes a maze will be investigated, and how they can be effectively generated for testing. Further a clear picture of the DFS, the BFS and the GA will be given, and how these work when solving mazes.

2.1 Mazes

Mazes have been an object of interest for a variety of cultures since ancient times. Today mazes can be connected closely with graph theory, since mazes can be seen as a representation of a set of vertices and edges[BLW76]. The advancements in computer technology has allowed the use of computers to compute paths in complex graphs with great success and a variety of different algorithms exists for accomplishing this task. Some are designed to quickly find a path between two vertices in the graph, and some are designed to more slowly find the shortest path between said vertices.

2.1.1 Generating mazes

Mazes can be constructed in a number of ways but in this report the focus will be on two dimensional, orthogonal, normal, non-perfect mazes. This means the maze will be a two dimensional maze drawn on a flat rectangular surface with walls that are always aligned with either the x-axis or the y-axis. The maze also contains more than one way to reach the destination cell from the source cell. This type of maze was chosen since it is the type of maze that bears the closest resemblance to graphs. Since the generation of mazes is not the focus of this report the process will be kept simple. An algorithm will be implemented that generates perfect mazes and then walls will be removed at random in the generated maze to create a non-perfect maze. The algorithm chosen to implement is Wilson’s algorithm. The reason for this is that Wilson's algorithm is an adequate solution for generating perfect mazes in a reasonable amount of time.
Wilson’s algorithm

Wilson’s algorithm follows four steps in creating a uniform spanning tree, or as we know them, a perfect maze [Wil96].

1. Randomly choose any vertex and add it to the uniform spanning tree.
2. Randomly select a vertex that is not already in the uniform spanning tree and perform a random walk until you encounter a vertex that is in the uniform spanning tree.
3. Add the vertices and edges touched in the random walk to the uniform spanning tree.
4. Repeat step 2 and 3 until all vertices have been added to the UST.

Using this algorithm, a perfect maze is created. By then removing different percentages of walls in the maze at random, non-perfect mazes with different levels of what we will refer to as openness are created.

2.1.2 Solving mazes

There are numerous algorithms for solving mazes, that is, to get from one specific cell of the maze, to another specific cell of the maze. For this report, two ways to solve a maze will be investigated, which will then be compared to our specialized GA. As previously mentioned, the algorithms that will be tested are the DFS, the BFS and the GA. Both the BFS and DFS originates from the 19th century, as algorithms for solving mazes [Cor04]. The BFS generally requires less time to reach a solution to path finding problems in graphs than the DFS, but in turn also requires more memory [Den+10].

Depth first search

A DFS algorithm traverses down one path of a graph all the way down to the final vertex, it then backtracks and traverses down the next path until all paths have been tried [PH07]. For example, a left traversing DFS would traverse down all the left branches of a graph until it reaches the end, it would then backtrack until the second to last vertex and travel down the second most left path as shown in figure 2.1.

The DFS is a fast way to travel through a graph with few branches, but if the path contains many branches the DFS can quickly become time consuming [Sad+10]. In the worst case scenario in a maze where the algorithm can keep track of the cells it has visited, the algorithm has to visit every vertex V and every edge E once, meaning the time complexity is $O(V + E)$ [PH07].
Figure 2.1: Depth first search, the numbers represent in what orders nodes would be visited.

**Breadth first search**

The BFS is a simple way to reach the shortest solution path in a graph, the cost is that it requires a large amount of time and memory to utilize, which scales up proportionally to the size of the maze \([PH07]\). The method itself is rather simple, the algorithm creates a tree structure containing all possible paths, where each level of the tree represents an amount of steps taken, this way, as soon as a branch contains the goal, the shortest path is found by selecting the parents of that branch all the way up to the root of the tree as shown in figure 2.2.

If implemented in its simplest form, this can become a slow and ineffective method for finding the shortest path when the graph becomes to large \([Sad+10]\). In the worst case scenario the algorithm has to visit every vertex \(V\) and edge \(E\) once, meaning the time complexity is \(O(V + E)\) \([PH07]\).
2.1.3 Genetic algorithms

Genetic algorithms draw inspiration from the work of Charles Darwin and is basically a model of an evolutionary system. The genetic algorithms randomly generate solutions and then continues building on the solutions that are most successful. This means that a GA can find solutions to problems even when the programmer is unaware of how to reach a good solution but has knowledge about the properties of such a solution. GAs have been shown effective at solving a large amount of optimization problems. GAs can be designed to meet any optimization goal, and is often a way to find an approximation to an optimal solution, when the path to finding that solution is unclear [BMB93].

A GA can vary greatly in appearance, depending on the optimization problem the algorithm is aiming to solve, but the parts making up the algorithm are always the same [PE] [Cha98]. A GA always contains a combination of the following:

1. A population of potential solutions to the optimization problem, often referred to as genomes. These are generated at random and most are probably not solutions at all in the beginning.

2. A fitness condition, this condition measures how well a member of the population fits the needs of the optimization.

3. A crossover method, this includes how to select the successful members of the population, usually through some probability setup, and how the
members selected propagate their genes, usually through a mixture of 2 successful solutions.

4. Some form of mutation of genes, to create variety and make sure evolution doesn’t stop at a dead end.

The population then goes through a series of generations, gradually improving on the original solutions until a fixed goal is reached, usually the goal is a set amount of generations during which no better solution is reached.

Deciding on the details of the parameters of the GA is not a trivial task. The size of the starting population is of great importance to the success of the GA, it must be large enough so that genetic diversity can be maintained, but if the population is too large the computational costs will be too great. GAs which utilize both a crossover method and mutations have been proven more effective than algorithms that utilize only one of these. For complex problems, a crossover operator is of bigger importance than mutation, since mutation alone have been shown to fail under these conditions[PE][Cha98].

To keep the diversity of the population, and thus increase the chances of reaching an optimal solution to a problem, the population of the algorithm is sometimes divided using a method know as many islands algorithm. The algorithm divides the population into groups, and prohibits reproduction between groups, thus allowing different genetic strains to survive, when they might otherwise die out[PE][Cha98].

An effective crossover method when designing a GA for solving mazes is the improved segment crossover. To understand this method, we need to first have a quick look on the segment crossover method. The method divides the genome of the parents into three segments, and the children receives their three segments from the mother and father randomly. The improved segment crossover is built upon the segment crossover method, but the first segment size is the size of the best working genes sequence between the mother and father[PE].

Instead of letting the genes mutate randomly, a directed mutation can be used to increase the effectiveness of the genetic algorithm when solving mazes[Cha98]. The best result is achieved when using a combination of Change Last Operator, Remove Loop Operator and Out Of Dead End Operator[PE]. These details will be further explained in chapter 3.
Chapter 3

State of the Art

3.1 GAs and maze solving robots

In their report, [Kal+09] explored the use of neural networks, A* algorithm and GAs for moving a mobile robot through a dynamically changing environment. The object was to decide which of these methods could be used to move the robot through the environment while avoiding collisions. It was shown that all of these methods could effectively be used.

Before we move further into their research regarding GAs, we must first examine the robot and the environment used in their study. The robot was of a simple design, a box with two wheels, the two wheels meant that the robot could move forward and backwards, and rotate around itself clockwise or counter clockwise. The direction of the robot was then numbered for algorithm purposes. The environment was a finite space containing a grid for positioning.

When implementing the GA, they constructed the solution path as a series of positions on the grid, as well as the robots facing on every position. The solution could then be represented as a series of tri-tuples like this $(x, y, d)$ where $x$ and $y$ represent the coordinates of the robot, and $d$ represents the facing of the robot. The steps were then represented as a graph, where a path between two steps was made if the move was a legal one. [Kal+09] imagined three possible scenarios, named full, left and right. In the full scenario the solution was complete and connected the source to the destination, in the left scenario the robot started at the source but was unable to reach the destination, and in the right scenario the robot did not start at the source but reached the destination.

The reason for the three scenarios was that the path was divided into time frames, each frame making up a gene in the genome of the individual of the population. So if the robot started at the source, and did reach the destination in the allotted time, the steps were added as a full solution, if not they were added as a left solution. In the next time frame another set of steps was randomized, if they did reach the destination, they were added as a right solution, otherwise they were added as a left solution.
The fitness condition was then calculated by calculating a separate fitness score for each time frame using a specific formula for each of the three scenarios. The fitness formulas were designed as follows.

1. **Full**: Number of steps needed to reach the destination

2. **Left**: Number of steps needed to traverse from source to the last point traversed + (Square of the distance of this point to the final destination + R(n))

3. **Right**: (Square of the distance of source point to the first point traversed + R(n)) + Number of steps needed to traverse from this point to the final solution

A number of initial solutions was first generated by following four steps.

1. Find a number of straight path solutions between the source and the destination. Do this by taking a number of random length steps.

2. Find a number of random left solutions by starting at the source and making random moves.

3. Find a number of random right solutions by starting at the destination and making random moves backwards.

4. Iterate over the right and left solutions and try to find pairs that meet each other at the ends, combine these to create more full solutions.

The crossover was made by selecting members of the three different sets of solutions, and combining them at points in the graph they have in common. The possible combinations were

1. Full Solution and Full Solution.

2. Left Solution and Full Solution.

3. Full Solution and Right Solution.

4. Left Solution and Right solution.

Finally, mutation was added to solutions, the chance of mutation was set to 0.002%. Mutation was done by picking two points in the solution, and finding a full path between these, thus creating a new solution. In conclusion of the paper, the GA turned out to be rather slow, but in the end was able to generate good solutions for even the most chaotic environments.
3.2 Optimizing genetic algorithms for solving mazes

In their report [PE] explores how to best optimize a GA for solving mazes, and propose an optimal setting for the parameters of the GA when solving mazes. They explain the importance of each of the parameters and how they affect the results when chosen poorly. In their experiments the source cell is located in top left corner, and the destination cell located in the bottom right corner.

They start of by explaining how the size of the starting population affects the results of the GA. Choosing a too small population will be detrimental for the time needed to reach a solution. Choosing a too large population will be detrimental for the time needed to optimize a solution. It is therefore important to choose a fitting population size.

[PE] goes on explaining how crossover and mutation methods affect the results. They suggest combining crossover and mutation, since a combined method has proven superior to methods using only crossover or only mutation. They do however go on to state that in complex problems such as a maze, crossover plays the more important role.

Diversity plays an important role in the success of a GA according to [PE], who define diversity as a number, calculated as the number of genes that differs between two individuals, divided by the total number of genes. The closer the diversity is to zero, the closer the GA is to a solution, albeit not always a very optimal solution. Diversity is one of the most important factors of the GA, and when the individuals of the gene pool are close to identical, the process is said to converge, and crossovers will stop. If this happens before an adequate optimum has been found, the process is said to have converged prematurely. To avoid this, diversity must be maintained as long as possible[PE].

In their report [PE] investigates how to keep the GAs diverse long enough to achieve good results when solving a maze. They suggest using the many islands method, a method of dividing the population of the GA into groups. Propagation is only allowed within a group, until the diversity within each group falls below a certain limit, then the groups are added together. This ensures that diversity is maintained longer than if the population was just divided into one group. [PE] further propose that a population of 50 individuals gives the best results when solving the maze.

[PE] then investigates the best crossover method and mutation to utilize in GA when solving mazes. They conclude that the best results are achieved when using the improved segment crossover, change last operator, remove loop operator and out of dead end operator. These will now be explained in a little more detail.

3.2.1 Improved segment crossover

This crossover method allows propagation between two individuals of the population by dividing the genome into three segments, the length of the first segment is the length of the most successful segment part of the parents, and the other two segments are of random length. The three segments of the child...
is then constructed by choosing three segments at random from the parents. This allows for maintained functionality and allows good genome constructions to thrive.

3.2.2 Change last operator, remove loop operator and out of dead end operator

These three operators works as improvements and mutations on the genomes in the population, it is a way to ensure that the population does not reach a dead end in its evolution. The change last operator switches the first faulty gene in a genome, genes that lead into a wall in the maze, for a gene that leads into a valid path. The remove loop operator, tries to make sure a genome does not loop back on itself in the maze. The out of dead end operator tries to make sure that an individual does not reach a dead end, without the ability to get back out.

3.2.3 Fitness condition

The fitness of the individuals was calculated by combining two factors. The first factor is the distance between the source and the destination when drawn as a straight line, the second the difference in path length between the solution and the distance mentioned above. A value called alpha is set to decide the weights of the two factors when calculating the fitness condition, [PE] suggested using the value 0.6, which would favour the first factor.

3.3 BFS and DFS in mazes

In their report [Ter+11] explores the use of several path finding algorithms for simulating path planning of a robot in a maze. Among the algorithms explored are the BFS and the DFS. The mazes used were constructed using a third party software named Mapper3 and the source and destination cells could be placed as required. The path was then calculated using one or more of the implemented algorithms. This report gives some insight into the uses of path finding algorithms and graph theory in modern day science as a tool for development of robotic technology.
Chapter 4

Method

4.1 Maze generation

To begin with, a testing environment was set up starting with a maze generation algorithm, Wilson’s algorithm[Wil96]. Thereafter these mazes had walls removed at random, with the only condition that removing the room would not create open areas. The different levels of walls removed is referred to as openness as was mentioned earlier.

The mazes generated were in the sizes 10x10, 25x25, 50x50 and 100x100 since mazes smaller than this make for trivial examples, and mazes bigger than this put too much strain on the available hardware. All of these mazes were constructed with four different levels of openness, which were 0%, 5%, 15% and 25%. These levels of openness were chosen since trial and error showed higher levels than this created mazes that no longer resembled mazes. The source and destination cells were selected in two different ways, in order to make our comparison as universally applicable as possible. In the first case the source and destination cells were placed diagonally opposite of each other, meaning most of the maze had to be traversed to reach the destination. In the second case the source and destination were randomly placed, allowing for some variation in difficulty. For both cases one thousand mazes of the smallest size were created, but due to hardware restraints only 100 mazes of sizes 25x25 and 50x50 were created, and 10 mazes of size 100x100.

4.2 BFS and DFS

The DFS is a standard recursive implementation using linked lists and the BFS is a standard implementation as well, utilizing a queue. The rightmost path is explored first for both the BFS and DFS.
4.3 GA

When running the tests, a GA developed by [PE] was utilized. Small changes were made to the original program to allow for source and destination cells to be placed randomly. The full program is complete with testing and graphical presentation of results, but in this report only the genetic algorithm itself was used. Since the GA developed by [PE] was optimized for small mazes where the source and destination cells were diagonally opposed, some trial and error was necessary to ensure that the GA had optimal parameters. Therefore, the settings suggested by [PE] was used to start with, and then what settings produced the best results for the GA in the cases of this report was investigated. Since great interest lay in the performance with regards to speed of the algorithms, it was decided that no more than 2 minutes was allotted each algorithm to solve a maze. This was because both the DFS and BFS seemed to perform considerably better than this also because of the limited time available to conduct tests. This meant that the GA was not always allowed to finish finding a solution, something that is reflected in our results. Otherwise the GA would terminate as soon as a generation had found a solution. While GAs can be used after a solution has been found to optimize it, this was not used in this report. The lengths of the solution pathways rarely got improved significantly, but the computational time increased, and therefore this option was deemed to be not worth using.

Our implementation uses the many islands algorithms, the improved segment crossover, the change last operator, the remove loop operator and the out of dead end operator[PE]. The starting population was set to 30 individuals and the number of islands set to 10 for all mazes, except for the 100x100 where the starting population was 20 individuals and the number of islands set to 5. The alpha value was set to 0.6, which was the value suggested by [PE] and also the value which proved optimal, according to our trial and error testing.

4.4 Test program

The algorithms mentioned above were all implemented in the Java programming language; Java was used both because it is a pre-compiled imperative language, and also because the program created by [PE] was written in Java. The test runs begin by generating a maze using the aforementioned maze generator and then allowing the algorithms to attempt to solve it individually, one at a time. The algorithms’ computational time and length of the solution pathway was measured. This process was repeated for the predetermined number of times. When all the tests were finished the data from the test runs where the GA managed to solve the maze were collected and used to calculate mean and box plots for both computational time and length of the solution pathway. The number of times the GA failed to find a solution were also recorded.
Chapter 5

Results

In this section the results from the tests will be presented. First the tests run on mazes where the source and destination cells were placed diagonally opposed to each other will be presented and afterwards those with randomly placed source and destination cells. The results are presented with plots displaying the relation between size and mean computational time or between the size and mean length of the solution path. The tests run on mazes with different levels of openness will have separate plots. A few graphs will have no data for the 50x50 and the 100x100 and that is due to the fact the GA failed all of its attempts of finding solutions to the mazes of those sizes. However, it was decided that the same format for the graphs would be kept for the sake of regularity and to elucidate the aforementioned fact.

5.1 Tests on mazes with diagonally opposed source and destination cells

Figures 5.1-4 show how the three algorithms performed on mazes with different levels of openness in terms of computational time and figures 5.5-8, how the three algorithms did in terms of solution path length. The four levels of openness are 0%, 5%, 15% and finally 25%.
Figure 5.1: A diagram showing computational time of the algorithms on mazes with 0% openness and diagonally opposed source and destination cells.

Figure 5.2: A diagram showing computational time of the algorithms on mazes with 5% openness and diagonally opposed source and destination cells.
Figure 5.3: A diagram showing computational time of the algorithms on mazes with 15% openness and diagonally opposed source and destination cells.

Figure 5.4: A diagram showing computational time of the algorithms on mazes with 25% openness and diagonally opposed source and destination cells.
Figure 5.5: A diagram showing length of the solutions by the algorithms on mazes with 0% openness and diagonally opposed source and destination cells.

Figure 5.6: A diagram showing length of the solutions by the algorithms on mazes with 5% openness and diagonally opposed source and destination cells.
Figure 5.7: A diagram showing length of the solutions by the algorithms on mazes with 15% openness and diagonally opposed source and destination cells.

Figure 5.8: A diagram showing length of the solutions by the algorithms on mazes with 25% openness and diagonally opposed source and destination cells.
Inspection of Figures 5.1-4 reveal that the computational time of the GA is significantly worse than the computational time of the BFS and the DFS. Already at the 10x10 mazes the GA was two orders of magnitude slower than the BFS which in turn was slower than the DFS. As the mazes get larger, the performance of the GA drops further in comparison to the other two algorithms and at 100x100 ends up being approximately five orders of magnitude slower. The DFS was slightly quicker than the BFS for all levels of openness. The boxplots in Figures 5.1-4 reveal the fact that even the quickest run for the GA was slower than the slowest of runs for the other two algorithms for any given size of maze.

Lengthwise the GA performed as hypothesised. In figures 5.6-8, the solution pathways found by the GA was not as short as the optimal solution pathways of the BFS, but also not as long as those found by the DFS. Besides having the longest solutions, the DFS also had a wider distribution of its solution pathway lengths, as evident from the tall upper whiskers on the DFS’s boxplots. In figure 5.5 all of the algorithms have the same solution pathway lengths, since that graph depicts tests which were ran on a perfect maze and perfect mazes by its very definitions has only one pathway between any two points.

At times the GA does not find a solution path within the allotted time. When that happens its execution was terminated. The portion of test runs where the GA did not manage to find a solution is displayed in the table below.

Table 5.1: A table showing the percentages of GA runs which did not find a solution path for different levels of maze sizes and openness.

<table>
<thead>
<tr>
<th>Openness</th>
<th>10x10</th>
<th>25x25</th>
<th>50x50</th>
<th>100x100</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>1%</td>
<td>71%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>5%</td>
<td>0%</td>
<td>19%</td>
<td>74%</td>
<td>100%</td>
</tr>
<tr>
<td>15%</td>
<td>0%</td>
<td>3%</td>
<td>9%</td>
<td>60%</td>
</tr>
<tr>
<td>25%</td>
<td>0%</td>
<td>5%</td>
<td>2%</td>
<td>20%</td>
</tr>
</tbody>
</table>

By inspecting table 5.1, it is evident that the GA seldom failed to find a solution when the maze was only of size 10x10. The GA also performed better when the openness was high, allowing it more possible pathways to progress through. The general trend which can be observed in table 5.1 is that, the larger the maze and the lower the openness, the more frequently the GA failed at finding a solution.
5.2 Tests on mazes with randomly selected source and destination cells

Figures 5.9-12 show how the three algorithms performed on mazes with different levels of openness in terms of computational time and figures 5.13-16, how the three algorithms did in terms of solution path length. The four levels of openness are 0%, 5%, 15% and finally 25%.
Figure 5.9: A diagram showing computational time of the algorithms on mazes with 0% openness and randomly selected source and destination cells.

Figure 5.10: A diagram showing computational time of the algorithms on mazes with 5% openness and randomly selected source and destination cells.
Figure 5.11: A diagram showing computational time of the algorithms on mazes with 15% openness and randomly selected source and destination cells.

Figure 5.12: A diagram showing computational time of the algorithms on mazes with 25% openness and randomly selected source and destination cells.
Figure 5.13: A diagram showing length of the solutions by the algorithms on mazes with 0% openness and randomly selected source and destination cells.

Figure 5.14: A diagram showing length of the solutions by the algorithms on mazes with 5% openness and randomly selected source and destination cells.
Figure 5.15: A diagram showing length of the solutions by the algorithms on mazes with 15% openness and randomly selected source and destination cells.

Figure 5.16: A diagram showing length of the solutions by the algorithms on mazes with 25% openness and randomly selected source and destination cells.
Figures 5.9-12 reveal that the computational time of the GA is significantly worse than the computational time of the BFS and the DFS, similar to the tests run on diagonal mazes. For the mazes of size 10x10, the GA is between two and four orders of magnitudes slower than the other two algorithms and for the mazes of size 100x100, somewhere between four and six orders of magnitudes slower. The mean of the DFS was lower than the mean of the BFS for virtually every size of maze and all levels of openness, with the possible exception of the 100x100 mazes in figure 5.10, where the two were quite even.

Lengthwise the GA performed as following. In figures 5.13-16, the solution pathways found by the GA was almost as short as the optimal solution pathways of the BFS. The DFS however performed worse than the GA and BFS in regards to solution pathway length. Besides having the longest solutions, the DFS also had a wider distribution of its solution pathway lengths, as evident from the tall upper whiskers on the DFS’s boxplots. In figure 5.13 all of the algorithms have the same solution pathway lengths, since that graph depicts tests which were ran on a perfect maze and perfect mazes by its very definitions has only one pathway between any two points.

At times the GA does not find a solution path within the allotted timespan. When that happens its execution was terminated. The portion of test runs where the GA did not manage to find a solution is displayed in the table below.
Table 5.2: A table showing the percentages of GA runs which did not find a solution path for different levels of maze sizes and openness.

<table>
<thead>
<tr>
<th>Openness</th>
<th>10x10</th>
<th>25x25</th>
<th>50x50</th>
<th>100x100</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0%</td>
<td>55%</td>
<td>92%</td>
<td>100%</td>
</tr>
<tr>
<td>5%</td>
<td>0%</td>
<td>11%</td>
<td>60%</td>
<td>70%</td>
</tr>
<tr>
<td>15%</td>
<td>0%</td>
<td>10%</td>
<td>44%</td>
<td>80%</td>
</tr>
<tr>
<td>25%</td>
<td>0%</td>
<td>7%</td>
<td>33%</td>
<td>50%</td>
</tr>
</tbody>
</table>

By inspecting table 5.2, it is evident that the GA seldom failed to find a solution when the maze was only of size 10x10. The GA also performed better when the openness was high, allowing it more possible pathways to progress through. The general trend which can be observed in table 5.2 is that, the larger the maze and the lower the openness, the more frequently the GA failed at finding a solution. The only table entries that does not adhere to this trend are those in the 100x100 column and the second and third rows. This discrepancy is most likely due to the inherent randomness in these tests, since the source and destination cells are placed randomly.
Chapter 6

Discussion

The data we have gathered from our experiments has left us rather disappointed, we were hoping the GA would provide better results. This does not affect the import of our research however, since it still provides a benchmark of the GA’s ability to solve mazes. These results could be considered a strong indication that GA is not an appropriate algorithm for solving mazes, since the evolutionary steps take too long to perform. Even in the best situation it does not appear sound to utilize GA for the purpose of solving mazes, in regards to computational time and solution path length when a BFS or DFS could be implemented instead.

6.1 Summary of the results

It is clear from the results that the GA generally performs worse than both the DFS and the BFS in terms of computational time for reaching a solution. We had hoped to be able to show that the GA could perform as well as the DFS or BFS in certain scenarios, the results have not shown this to be true with regards to computational time.

Lengthwise however, the results do reflect our expectations. The solution path length of the GA in general is far shorter than those of the DFS and almost as short as the solution path length of the BFS in many cases. The results of the GA are consistent with the results produced by [Kal+09]. It is important to highlight the fact that the GA is far less reliable than the BFS and the DFS, as it is not always able to reach the destination cell within the predefined time frame. In the smaller mazes, this is not a considerable factor, but when the tests were run on the larger mazes with lower percentages of openness it became a deciding factor. In the mazes of size 100x100 with 0% or 5% openness, the GA rarely managed to solve the maze. The most probable cause for this is due to the fact that bigger mazes with lower openness naturally entails more dead ends which causes GAs to converge prematurely towards false optimums. Regardless of what reason is causing this phenomenon, it should be considered
further evidence against GA’s ability to solve mazes quickly. Furthermore, this also means that in the cases of the larger mazes, the data presented here may be less reliable as well, as there was less data for which to measure the GA’s performance against the BFS and DFS. Despite this it would seem quite clear that the internal mechanism of the GA is simply to slow and cumbersome in order to get as short computational times as the BFS and the DFS.

6.2 Discussion of the method

The GA developed by [PE] was optimized for solving small mazes, in sizes smaller than 50x50 where the source and the destination cells were placed diagonally opposed each other. Since their GA was utilized outside of its intended environment this could possibly have affected the results. It might be possible to optimize the GA further for each maze and thus reach better end results. This would however indicate that the GA while diverse in its use, is not as reliable as the BFS and DFS when it comes to solving mazes. The GA’s parameters were chosen based on the suggestions given by [PE] in conjunction with our internal testing. It is possible that the parameters could be calibrated further and thereby increase the GA’s potential for each type of maze configuration.

The number of tests conducted on each size and percentage could also be a source of errors. We ran 1000 tests on the mazes 10x10, which was deemed adequate. On the 25x25 and 50x50 however, we ran 100 and on the 100x100 only 10, which entails a significant margin of error. Note again that [PE] did not scale their mazes up at all, meaning their results only truly reflect the conditions they created. It is therefore difficult to assess how much the GA could be improved upon, but it is not likely the performance could be enhanced to the levels of the BFS and DFS.

The restrictions placed upon this research by the hardware available, must also be taken into account. Running test on mazes larger than 100x100 was not an option, since the time and memory consumed was too straining on the hardware. Our aspiration was to be able to show some form of convergence between the BFS and the GA in terms of computational time when the mazes grew in size, but due to the restraints on hardware we were unable to confirm this hypothesis.

No proper statistical model was used for analysing the data as we deemed what the data expose to be too plain. Any errors or inconsistencies were presumed to be far lesser than the computational time differences that were observed.

6.3 Discussion of the reviewed studies

Though the field of GA’s is widely studied, and the area of using GAs to solve path traversing problems well explored, it would seem that studies regarding the performance of GAs as compared to more traditional methods for graph
traversing is severely lacking. This limited us somewhat in our studies, as we were unable to investigate a large enough volume of studies conducted in the field. This is the main reason as to why we chose to utilize the studies performed by [PE].

The paper written by [PE] has yet to be published and receive auditing from the scientific community. This could indicate that the genetic algorithm may yet be improved upon further to receive better results, however we find it unlikely that the improvements would be significant unless the whole structure of the GA provided by [PE] would be reconstructed.
Chapter 7

Conclusion

We have found that the GA simply does not measure up to the BFS and the DFS for solving mazes. The GA is nowhere as efficient as the DFS or the BFS in terms of computational time, although it does often provide good solution lengths. It might be possible to increase the performance of the GA to achieve better results although the improvements would have to be significant for the GA to be able to measure up to the DFS and BFS. The GA’s performance can be summarized as follows.

1. The GA is overall slower than the BFS and DFS.
2. The GA does often provide good solution path lengths.
3. The GA does not always find a solution, this problem increases in the bigger more complex mazes.
4. The more open pathways in the maze, the better the GA performs when compared to the BFS and DFS.

In short there may be a place for GAs when solving mazes, but it is not when a fast and efficient solution is required. Our results could be an indication that GAs considerable internal processes means that the GAs will have difficulties performing as well as efficient specialized algorithms when solving optimization problems.
Bibliography


[Cor04] Derek G Corneil. "Lexicographic breadth first search—a survey”. In: *Graph-theoretic concepts in computer science*. Springer. 2004, pp. 1–19.


