Maze Benchmark for Testing Evolutionary Algorithms*

Camilo Alaguna
Universidad Nacional de Colombia
Bogotá
caalagunac@unal.edu.co

Jonatan Gomez
Universidad Nacional de Colombia
Bogotá
jgomezpe@unal.edu.co

ABSTRACT

In this paper, a new benchmark, for testing the capability of Evolutionary Algorithms to generate maze solving Exploration Strategies is introduced. This benchmark solves three problems commonly found in other benchmarks: small maze set, all mazes belonging to the same kind of maze, and biased methods for choosing starting locations. In this benchmark, the Connectivity Based Maze Generation Algorithm is proposed for building maze sets. Mazes generated using this algorithm exhibit a property called connectivity that indicates how much the walls are connected among them. Connectivity can be used to control the diversity of the sets. Additionally, a new method for choosing starting locations, that takes into account the maze goal positions, is presented. Using the Connectivity Based Maze Generation Algorithm and the new method for choosing starting locations, two problems that test the capability of an Evolutionary Algorithm for solving mazes with similar and different connectivity are built. Example datasets are generated and experiments are performed to validate the proposed benchmark.

CCS CONCEPTS

• **General and reference → Cross-computing tools and techniques**; • **Computing methodologies → Artificial intelligence**;

KEYWORDS

Benchmark, Maze Solving, Evolutionary Algorithms, Maze Generation Algorithms, Connectivity, Starting Locations

ACM Reference Format:
Camilo Alaguna and Jonatan Gomez. 2018. Maze Benchmark for Testing Evolutionary Algorithms. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208285

1 INTRODUCTION

Planning a route, from one point to another, is one of the most common problems in daily life. We face it when we are moving around a city, when delivering goods, while cleaning a house and in many other cases. Many technologies have been developed to make solving this problem easier [1, 9, 15]. Nevertheless, planning a route becomes a really hard problem when the place being explored is unknown, the goal place is not completely defined, or there are obstacles and closed paths in the way. For example, when a building collapses, rescuers spend hours and even days trying to find survivors. The search process can be accelerated by sending autonomous devices to look for survivors. However, these devices might fail as a result of new conditions that were not taken into account when they were developed. In this case, the designer must develop new devices and Exploration Strategies (ES), spending time and resources on it. One way of finding better ES is to abstract the exploration problem as a maze solving problem, because mazes present several challenges commonly found in real navigation problems, like the presence of loops [3] and dead ends [11].

In a maze solving problem an agent is placed inside a maze, and then it performs an ES to find a way out. Mazes are interesting, because in them the walls block direct paths to the exit, hindering the exploration task. The maze problem, originally proposed by Koza in [10], has been used extensively when finding ES by using Evolutionary Algorithms (EA), in which testing in real scenarios is costly and it is unknown whether the algorithm can generate coherent ES.

Lehman and Stanley, as Koza proposes, use EA and only one maze to obtain and test ES [11, 12]. This has the problem that it is not possible to verify if the EA generates a general ES for solving mazes, or is just solving one instance of the problem. In actual scenarios it is important for the ES to solve more than one instance of the problem so it can be robust against unknown circumstances. By increasing the number of mazes, it is possible to test the capability of an obtained ES of solving multiple mazes. Gordon and Matley use five mazes, with different sizes [6]. They evolve, however, an specific ES for each maze, and the specific ES are not tested in the other mazes. Georgiou et al use two simply-connected mazes to generate ES [4, 5], but the mazes are used both for the Evolutionary Process (EP) and for the Testing Process (TP). Keane does something similar with fifty non-simply-connected mazes [8], but again all of them are used in both the EP and the TP. To avoid that, Shorten and Nitschke generate a set of one thousand mazes, and divide them into two sets, one for evolution and the other for testing [16, 17]. Nevertheless, these mazes are designed with the same features (to obtain the Wall-Follower algorithm), and the obtained ES are not verified against mazes with other kinds of features.

One approach to avoid testing on the same kind of maze, is to move the exit, and increasing the number of instances. Velez and Clune do this by using fourteen different locations (chosen manually) on a single maze [19] starting from the center. However, all of the exit locations were used both for the EP and for the TP, making it difficult to determine whether the obtained ES are able to find an exit placed on locations different from the original fourteen. Another approach, is to change the location where the agent starts.

*Produces the permission block, and copyright information

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208285

Urbano, Trujillo and Naredo generate randomly a hundred different starting locations [13, 18]. They divide the starting locations set in two, one for the EP, and the other for the TP, but they do not test whether the obtained ES solve mazes different from the used ones.

In summary, proposed benchmarks are presenting three problems: the maze set is small, all mazes belong to the same kind of maze, and there is not a clear method for choosing starting locations. By increasing the amount of mazes it is possible to solve the first problem, however, doing this in a wrong way leads to the second problem. For that reason, in the maze set creation, it is necessary to use an algorithm able to generate different kind of mazes, and also to have a metric that indicates whether mazes belong to the same kind or not. Finally, the selection of starting locations have been done either manually (introduces researcher bias) or randomly (ignores the topological characteristics of mazes), therefore, it is necessary to propose a new method for choosing starting locations.

In this paper, a new benchmark, for testing the capability of an EA to generate maze solving ES, that takes into account the problems mentioned above, is proposed. Section 2 describes how a diverse maze set is generated, by using a metric called connectivity and a modified version of Wilson's Algorithm. Section 3 explains how the starting locations are chosen. Section 4 presents the generated benchmark dataset. Section 5 shows how EA are evaluated by using the dataset. Finally, Section 6 draws some conclusions.

2 MAZE GENERATION

To generate the maze set, we propose a new algorithm that we name "Connectivity Based Maze Generation Algorithm" (CoMGA, see Section 2.2), based on the Wilson's Algorithm for simple connected mazes generation (See Section 2.1). This algorithm can generate different kinds of mazes by varying only one parameter, because the generated mazes present property called connectivity (See Section 2.3). and in each maze we place a goal cell avoiding dividing the mazes (See section 2.4).

2.1 Simply Connected Mazes Generation

Simply connected mazes are defined as mazes that do not have loops (paths which start and end on the same place) [21], for that reason, this kind of mazes can always be solved by the Wall-Follower Algorithm. Wilson's Algorithm has been used to generate simply connected mazes [20], without using structures that can introduce bias in the maze generation. The idea is to surround all the maze's cells with walls, and then, remove these walls iteratively in order to build the maze structure (See Algorithm 1).

The algorithm starts by surrounding all the maze's cells with walls (line 5), then one randomly chosen cell is marked as a goal cell (lines 6 and 7) (See Figure 1a). While non-marked cells still exist (line 8), one non-marked cell is randomly chosen (line 9), as shown in Figure 1b. After that, a path is randomly generated, starting from the last chosen cell, until it reaches another marked cell (line 10) (See Figure 1c). Next, the walls that obstruct the path (Figure 1c) are removed (line 11), as in Figure 1d). Finally, the cells that the path crosses are marked as goal cells (line 12) as in Figure 1d.

Algorithm 1 Wilson's Algorithm

```
1:  function generate()
2:      structure = new MazeStructure()
3:      nonMarked = structure.cells
4:      marked = []
5:      fill(structure)
6:      cell = chooseOne(nonMarked)
7:      marked.add(cell)
8:      while nonMarked.size > 0 do
9:          cell = chooseOne(nonMarked)
10:         path = buildPath(cell, marked)
11:         removeWalls(structure, path)
12:         marked.add(nonMarked, path.cells)
13:     end while
14:     return structure
15: end function
```

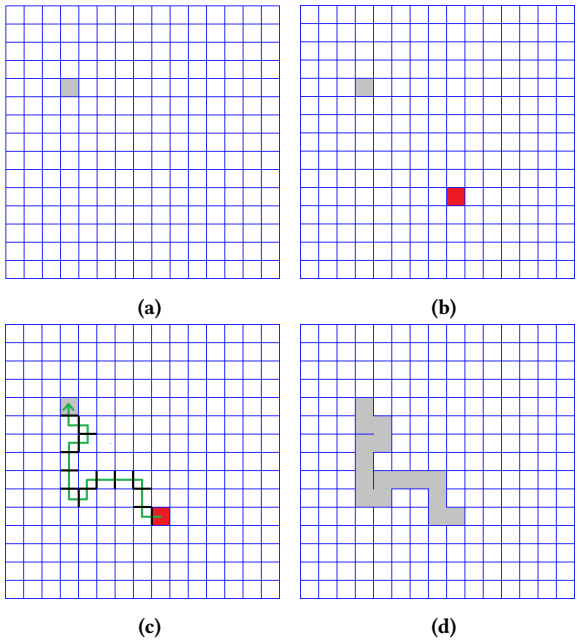


Figure 1: Wilson's Algorithm steps: (a) surround the maze cells with walls and mark one cell randomly, (b) choose a non-marked cell randomly, (c) generate a random path between the chosen cell and another marked cell, (d) mark cells in the path and remove walls.

2.2 Connectivity Based Maze Generation Algorithm

Wilson's Algorithm only produces simple-connected mazes, because it is not possible to choose already marked cells to start a path at the beginning of each iteration (see Algorithm 1 line 9). For this reason, we modify this statement to make possible to choose already marked cells. The idea is to choose either a marked cell or

non-marked cell depending on a given probability. We call this Algorithm **Co**nnectivity Based **M**aze **G**eneration **A**lgorithm (CoMGA), that can be seen in Algorithm 2.

Algorithm 2 Connectivity Based **M**aze **G**eneration **A**lgorithm

```
1:  function generate()
        same as Algorithm 1...
9:          cell = chooseEither(nonMarked, marked)
        same as Algorithm 1...
15: end function

16: function chooseEither(nonMarked, marked)
17:     option = random(0, 1)
18:     if option < nonMarkedProbability then
19:         return chooseOne(nonMarked)
20:     else
21:         return chooseOne(marked)
22:     end if
23: end function
```

As in Wilson's Algorithm, CoMGA surrounds all maze cells with walls, marks cells as goal cells, iterates until all cells are marked and removes walls to build the maze structure. The main difference is the function *chooseEither*, called in line 9, which chooses either a non-marked cell or a marked cell to start a path depending on a probability (see Algorithm 2). This probability is known a priori and indicates the likelihood of choosing a non-marked cell. To select a cell, function *chooseEither* generates a random number between zero and one (line 17), if this number is less than the given probability, one non-marked cell is chosen randomly (line 19), otherwise, one marked cell is chosen randomly (line 21).

Figure 2 shows what happens when a marked cell is chosen. We continue from where the iteration in Figure 1 ends (See 2a). Function *chooseEither* chooses one marked cell (Figure 2b). Then, starting from the last chosen cell, a path is randomly generated until one marked cell is reached, as in Figure 2c. Finally, the walls that block the path are removed and the cells where the path crosses are marked as goal cells (See Figure 2d). Notice that, when a marked cell is chosen, the algorithm creates a loop in the maze. The emergence of these loops is what lets the algorithm generate non-simply connected mazes, as a result of disconnecting a group of connected walls from another.

2.3 Connectivity

As mentioned above, the emergence of loops enables the proposed algorithm to build non-simply connected mazes. Figure 3 shows four mazes generated by setting the probability of choosing a non-marked cell as 0 (a), 0.3 (b), 0.6 (c) and 1 (d). Notice that, the amount of walls and how much these walls are connected among them, increases when this probability tends to 1. That is to say, mazes become more connected when the probability tends to 1. So we call this property **connectivity**.

Two mazes, generated by setting a similar non-marked probability in CoMGA, have a similar connectivity. On the contrary, when this probability is different, mazes have different connectivity. Additionally, as mentioned above, connectivity increases when this

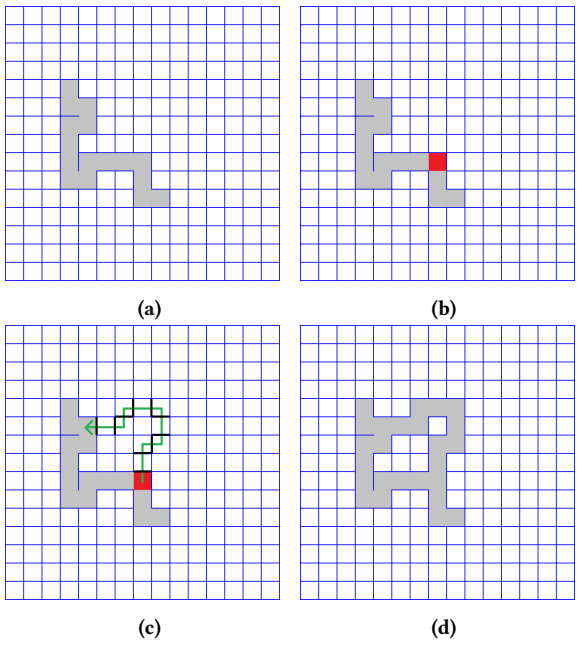


Figure 2: Choosing a marked cell: (a) starting after the iteration in Figure 1, (b) one already marked cell is chosen, (c) a random path between the last chosen cell and one marked cell is generated, (d) cells in the path are marked and the walls are removed.

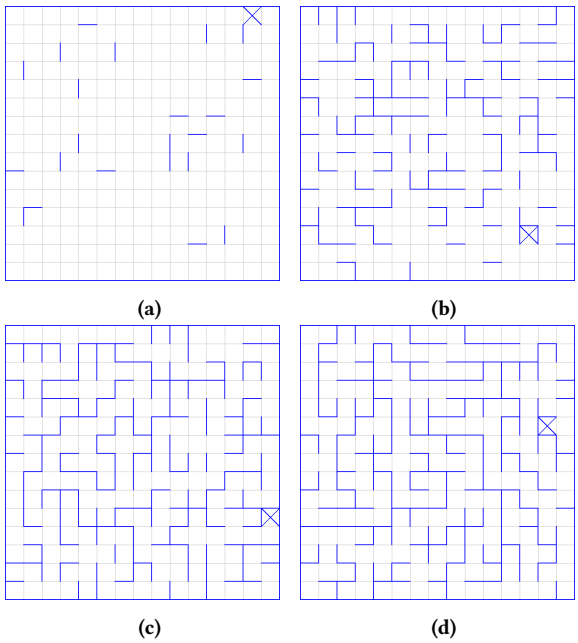


Figure 3: Mazes generated by setting a probability of choosing non-marked cells of: (a) 0, (b) 0.3, (c) 0.6, (d) 1.

probability tends to 1. This property allows us to conclude the probability given to the CoMGA is a measure of connectivity, because

what inputs the algorithm indicates how connected a maze is going to be. We express this connectivity as a percentage (multiply the given probability by one hundred).

2.4 Selecting a Main Goal

We should be careful when we are choosing the position in which the Main Goal cell of a maze is going to be, because this cell might divide the maze in two. Figure 4 shows what happens when this goal is wrongly chosen. Observe that, in both mazes it is not possible to built a path to visit all cells avoiding crossing the Main Goal. In this situation, we have divided the maze problem in two smaller problems.

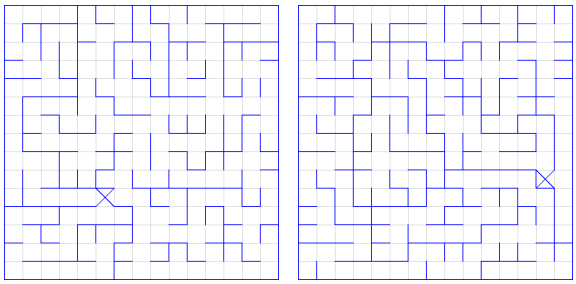


Figure 4: Samples of placing the Main Goal wrongly.

As mentioned above, placing the goal on a wrong cell can divide a maze in two, for this reason it is important to correctly choose where to place the main goal of the maze. In order to do that, we initially look for cells surrounded by three walls (this kind of cells never divide the maze) as in Figure 5a. If there is at least one cell of this kind, one of them is chosen randomly and marked as the main goal cell (See Figure 5b), otherwise any cell is chosen randomly an marked as the main goal cell.

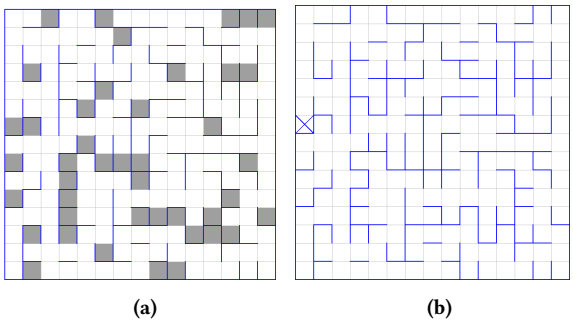


Figure 5: Choosing the main goal among cells with three walls: (a) selectable locations, (b) choose one cell as the main goal.

3 SELECTING STARTING LOCATIONS

To perform experiments, it is necessary to choose starting locations in a maze set, either to evolve Maze ES or to test them. However, the selection of starting locations is often performed in two ways: manually and randomly. The first one introduces a researcher bias, and the other ignores the topological characteristics of mazes. In this section we propose a new method for choosing starting locations that does not have these two disadvantages.

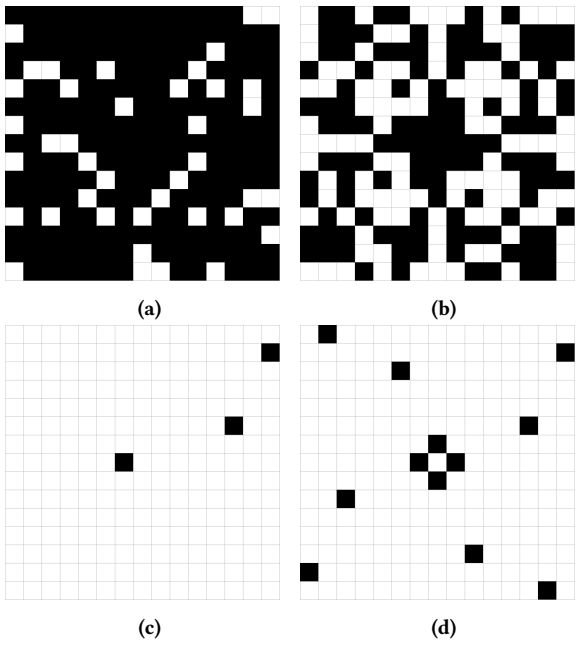


Figure 6: Location filtering process: (a) remove the maze set main goals cells, (b) rotate the main goals cells positions and continue removing the matching cells, (c) for each of the remaining cells measure the average distance from these to the main goals original locations, choose the cells which are at the nearest, median, and furthest distance from the main goal cells, (d) rotate the obtained cells

First of all, it is necessary to generate a maze set (See Section 2). In addition, it is not desirable to choose starting locations that coincide with goals, because in those locations the maze problem is already solved. To avoid this, all the cells are marked as possible starting locations, and then the cells which coincide with the main goals of all mazes in the set are removed (See Figure 6a). However, when we rotate the mazes by 90, 180 and 270 degrees, there are other cells that match the main goals, these cells are also removed (See Figure 6b).

It is expensive (in time) to use all locations that do not match main goals, because there can be a huge number of possibilities. To reduce the selectable starting points, for each location in the maze we calculate the average Manhattan distance between the location itself and all main goals. Then, we keep only the three which have the lowest, the median and the greatest average distance from all main goals (See Figure 6c). They cover uniformly a maze set respect to the main goals when these locations are rotated by 90, 180 and 270 degrees (See Figure 6d). Finally, the locations resulted for this process are used for both evolution and testing.

4 DATASET

This section describes our publicly available benchmark dataset (See mazebenchmark.github.io). Section 4.1 shows the software used to generate the dataset. Section 4.2 proposes two benchmark problems for testing ES. After this, Section 4.3 provides a data representation for the maze set and the starting locations.

4.1 Software

A Maze Generation Tool (See Figure 7) has been developed for building datasets. This software generates mazes by using CoMGA accepting connectivity as input. In addition, this tool performs the selection of starting locations automatically, it also produces images of stored mazes and starting locations for ease of use.

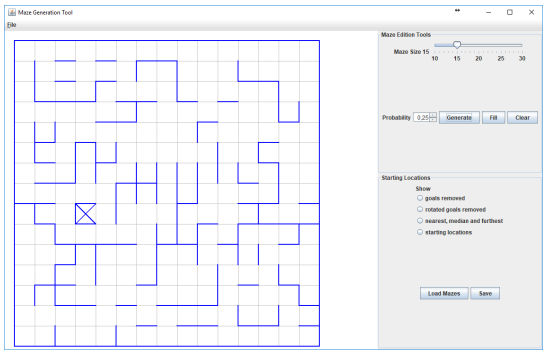


Figure 7: Screenshot of the software used for generating mazes and selecting starting locations.

4.2 Benchmark Problems

Two benchmark problems are proposed to test the capability of an ES for solving both similarly connected mazes and differently connected mazes. The first problem is called the "Similarly Connected Maze Problem" (SCMP), which, as the name implies, is designed to test if an ES can solve mazes with similar connectivity. The other is the "Differently Connected Maze Problem" (DCMP), that is designed to test if an ES is able to solve several mazes with different connectivities.

In the SCMP, a set of ten mazes with the same connectivity is generated. Then, the maze set is divided in two: four randomly chosen mazes for the EP and the other six for testing. The starting locations are chosen as described in Section 3. We generate four instances of the problem, in order to check if an EA performs well when finding ES for specific kinds of mazes. The first instance has mazes with connectivity of 0% (SCMP1), the second 30% (SCMP2), the third 60% (SCMP3) and the last 100% (SCMP4). We generate 15ÃŮ15 cell mazes, because this maze size is enough to allow a maze to present different structures for testing maze ES and is often used in Micromouse competitions.

The DCMP is designed to test if an EA is able to produce an ES, that solves kinds of mazes which are not used in the EP (generalization). To generate a diverse testing set with different kinds of mazes, we use probabilities of 0, 0.25, 0.5, 0.75 and 1, that are uniformly distributed between zero and one. Then, for each probability four mazes are generated. As in the SCMP dataset, we generate four datasets where the first training set has 15×15 mazes with connectivity of 0%, 15% and 30%. The second 40%, 60% and 80%. The third 70 %, 85% and 100%. The last one 0% and 100%. The idea is to evolve ES with less amount of kinds of mazes than the testing set have. By doing this, it is possible to check if obtained ES solve other kinds of mazes. Finally, starting locations are generated as Section 3 explains.

4.3 Data Representation

Our dataset includes three kinds of files: .mz for mazes, .loc for starting locations and .png for images. The .mz files store the data structure of a maze using a set of integer numbers. The first two numbers represents the width and the height of the maze. The following number indicates the maze's connectivity. Then there are $width \times height$ integer numbers which represent the maze structure, where each number indicates which walls surround a cell.

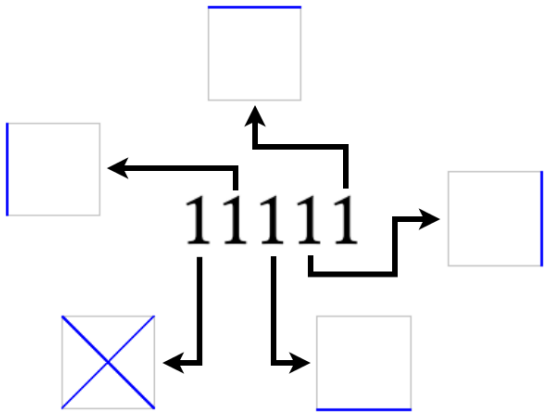


Figure 8: Cell binary representation.

To determine which walls actually surround each cell, we must parse the number from decimal base to binary base. As shown in Figure 8, five bits are used for cell representation, the rightmost one indicates whether a wall should be placed in the upper side of the cell, the second to the left indicates the same for the right side wall, the third one controls the lower side wall, and the fourth the left side wall. Finally, the leftmost bit indicates whether the cell is a Main Goal cell. Table 1 lists all possible cell values, shows their binary representation, and also shows the final graphical representation of the cell.

To store the starting locations, we use the .loc files. These files store only integer numbers. The first two numbers represent the maximum width and the maximum height of all mazes in the maze set. The next number is used to indicate how many starting locations there are. The following two numbers represent the column and the row in which a starting cell is located. Finally, .png files are used to store a graphical representation of the maze and indicate the starting locations.

Value	Bits	Cell	Value	Bits	Cell
0	00000		16	10000	
1	00001		17	10001	
2	00010		18	10010	
3	00011		19	10011	
4	00100		20	10100	
5	00101		21	10101	
6	00110		22	10110	
7	00111		23	10111	
8	01000		24	11000	
9	01001		25	11001	
10	01010		26	11010	
11	01011		27	11011	
12	01100		28	11100	
13	01101		29	11101	
14	01110		30	11110	
15	01111		31	11111	

Table 1: Cell Encoding Values with their binary representation and graphical output

5 EVALUATING EVOLUTIONARY ALGORITHMS

This section describes how to analyze the results of EA that find maze ES using the proposed benchmark. To illustrate this we test two EA, described in Section 5.1. Thirty experiments are performed (See Section 5.2). After that, we evaluate the performance of the obtained ES on the datasets, based on the metric described in Section 5.3. Finally, we discuss results of ES in both SCMP and DCMP (See Section 5.4 and 5.5).

5.1 Benchmarked Methods

One of the earliest Evolutionary Approaches proposed for finding maze ES directly defines the path between a starting location and the goal as a solution. Baba and Handa evolve binary strings where every bit represents two possible directions: down and right [2]. However, CoMGA generated mazes are usually not solvable by using only two directions to define a path, as the goal can be in one of the undefined directions. For this reason, we use two bits (instead of one) to represent four possible movement directions: up, right, down and left. Obtained paths can still collide with walls, so running agents ignore colliding movements.This feature allows agents to solve more than one kind of mazes. Finally, the agent execution stops when a goal cell has been found, or there are not more movements to be executed.

Two versions of the path definition approach are evaluated on the dataset (See Section 4). The first one, consist in evolving paths that can increase their size without limit. The other, limits the size to a maximum of 500 movements. This upper bound is enough for any given path to explore the maze up to two times. To identify each approach, we name the first one as the "Evolved Path Method" (EPM), and the second as the "500 Limited Evolved Path Method" (500LEPM).

5.2 Experimental Settings

Initial population is composed of 100 randomly generated binary strings of 20 bits. This population is evolved for 100 generations, by using the Generational Genetic Algorithm (GGA) [7]. The mutation probability is defined as one divided by the size of the genome ($1/|genome|$), and the crossover probability is 0.6. The two cross point crossover operator (each parent is divided by a different cross point) [14] is used instead of the typical crossover, because this operator lets genomes (binary string) increase in size. The fitness function is the average Manhattan Distance between the goal position and the last position of each agent after executing their path on different instances of the SCMP and DCMP datasets.

5.3 Performance Metric

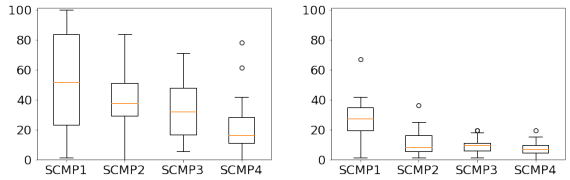
As mentioned in Section 4.2, the SCMP and the DCMP maze set is divided in two subsets, one for evolving ES and the order for evaluating the obtained ES. In both processes agents execute ES on each maze starting from the previously defined starting locations (See Section 3). When an agent finds a goal the ES is considered successful for that instance of the problem, otherwise it fails. We can consider the number of these hit events as a metric of performance, however, this metric depends on the number of mazes and the number of starting locations, something that can hinder the comparison among ES and Algorithms. For this reason, we choose the percentage of hit events as the performance metric of ES.

5.4 Analysis of Results for SCMP

The performance of obtained ES, when solving similar connected mazes, is evaluated on the four SCMP datasets (See Section 4.2). Figure 9 shows the distribution of hit percentage obtained with both EPM and 500LEPM, after having performed 30 repetitions. Observe that, in both methods the maximum median hit percentage and spread over the first and third quartiles are achieved on SCMP1, and then decrease until the minimum is reached on SCMP4. This allows us to conclude that the capacity of both algorithms for solving mazes with the same connectivity decreases when the connectivity increases (Remember that SCMP1 has mazes with connectivity 0%, SCMP2 30%, SCMP3 60% and SCMP4 100%). This analysis allows us to identify whether an EA presents difficulties while finding maze ES for solving certain kinds of mazes. In addition, this analysis can be used to compare the performance of different EA when finding maze ES for solving the SCMP. In this case we can observe that the hit percentage across datasets of 500LEPM tends to be lower than the hit percentage of EPM. For this reason, we can conclude that EPM is better that 500LEPM when solving the SCPM, because limiting the path size by 500 movements decreases the performance of obtained ES.

5.5 Analysis of Results for DCMP

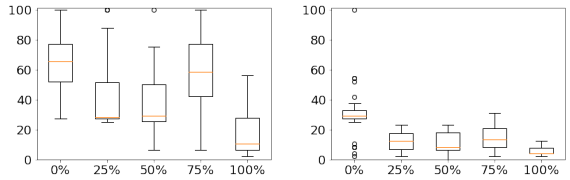
The capability of an EA for finding ES that solve different kinds of mazes not used during the Evolution (generalization), is evaluated on the four DCMP datasets (See Section 4.2). Figure 10 shows the hit percentage obtained with both EPM and 500LEPM, after having



(a) (b)

Figure 9: Hit percentage when evaluating in SCMP (30 repetitions): (a) boxplot of EPM, (b) boxplot of 500LEPM

performed 30 repetitions on the DCMP1 dataset. In this dataset mazes with connectivity of 0%, 15% and 30% are used in the EP. Notice that, with DCMP1 EPM can generalize maze ES for mazes with connectivity 50% and 75% that behaves almost as well as mazes with connectivity 0% and 25%, despite these kinds of mazes being not used in the EP. Therefore, we can conclude that EPM can find ES for solving some kinds of mazes different from the ones used in the EP. Moreover, Figure 10 also shows that mazes with connectivity 100% are the hardest to solve, and 500LEPM behaves clearly worse than EPM when finding general ES by using DCMP1.



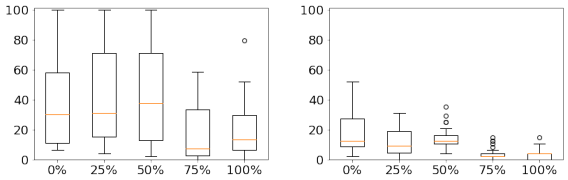
(a) (b)

Figure 10: Hit percentage vs connectivity when evaluating in DCMP1 (30 repetitions): (a) boxplot of EPM, (b) boxplot of 500LEPM

Figure 11 shows the hit percentage obtained with both EPM and 500LEPM, after having performed 30 repetitions on the DCMP2 dataset. Notice that, the median for all connectivities is lower and the spread is greater than in Figure 10. This result suggests that, it is harder for EPM and 500LEPM to find ES for solving other kinds of mazes when using mazes with connectivity of 40%, 60% and 80%. Additionally, EPM can generalize ES that in some cases solve mazes with connectivity 0%, 25% and 50%. Nevertheless, both EAs obtain better ES for mazes with connectivity of 75% when DCMP1 is used instead of DCMP2. This suggests that the training set affects the generalization capability of an EA, and this analysis can be used to identify which sets increase this capability.

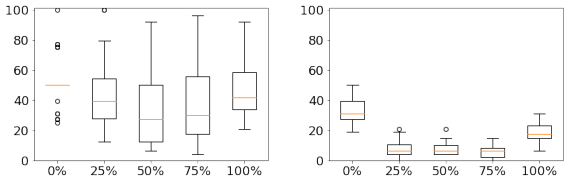
Figure 12 shows the hit percentage obtained with both EPM and 500LEPM, after having performed 30 repetitions on the DCMP3 dataset. Contrary to what is shown in Figure 10 and Figure 11, EPM and 500LEPM can generate better ES for solving mazes with connectivity of 100%, when mazes with connectivities of 70%, 85% and 100% are used in the EP.

Figure 13 shows the hit percentage obtained with both EPM and 500LEPM, after having performed 30 repetitions on the DCMP4 dataset. Notice that, EPM and 500LEPM cannot obtain results that



(a) (b)

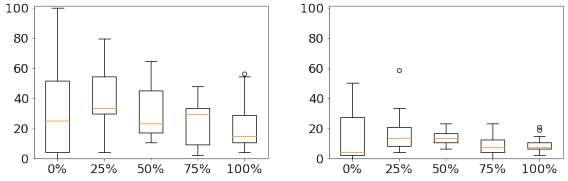
Figure 11: Hit percentage vs connectivity when evaluating in DCMP2 (30 repetitions): (a) boxplot of EPM, (b) boxplot of 500LEPM



(a) (b)

Figure 12: Hit percentage vs connectivity when evaluating in DCMP3 (30 repetitions): (a) boxplot of EPM, (b) boxplot of 500LEPM

performs well for mazes with connectivity of 0% and 100%, despite these kinds of mazes being used in the EP. This suggests that, to use mazes with characteristics similar to the ones tested, not always generates ES that behaves well with the testing set.



(a) (b)

Figure 13: Hit percentage vs connectivity when evaluating in DCMP4 (30 repetitions): (a) boxplot of EPM, (b) boxplot of 500LEPM

Finally, EPM obtains quite more general maze solving ES than 500LEPM for all kinds of mazes in the testing set. This suggests that EPM is better than 500LEPM when finding ES for the DCMP. This allows us to compare the generalization capability of EAs.

6 CONCLUSIONS

In this paper, a new benchmark, for testing the capability of EAs to generate maze solving ES is introduced. This benchmark solves three problems commonly found in other benchmarks: small maze set, all mazes belonging to the same kind of maze, and biased methods for choosing starting locations. The first problem can be solved by increasing the amount of mazes, however, doing this in a wrong way leads to the second problem. For this reason the CoMGA is

proposed, mazes generated using CoMGA exhibit a property called connectivity which can be used to control the diversity of the sets. For solving the third problem, this paper proposes a new method that takes into account the positions of maze goals for choosing starting locations.

The benchmark proposes two problems to test EA: SCMP and DCMP. The SCMP is designed to test how capable is an EA for producing ES that solve mazes with the same connectivity. The other problem (DCMP) is designed to test if obtained ES can solve different kinds of mazes not used in the EP (generalization). A Dataset for each problem is built, and then two EA are tested on this dataset. In addition, a metric is presented to assess the performance of EAs, allowing the comparison among them. Results show that the proposed problems (SCMP and DCMP) are able to test the capability of an EA for finding ES that solve both similar and diverse kinds of mazes. Moreover, both can be used to identify in which kind of mazes an EA presents difficulties, and if these difficulties increase or decrease when the connectivity changes. Additionally, results indicates that testing on DCMP datasets allow us to determine what kinds of mazes increases the performance of an EA.

ACKNOWLEDGMENTS

The authors would like to thank Rodrigo Moreno for all his help in writing this document.

REFERENCES

[1] 2017. Hydrodynamic assessment of planing hulls using overset grids. *Applied Ocean Research* 65 (2017), 35–46. https://doi.org/10.1016/j.apor.2017.03.015

[2] Norio Baba and Hisashi Handa. 1994. Genetic Algorithm Applied to Maze Passing Problem of Mobile Robot - A Comparison with the Learning Performance of the Hierarchical Structure Stochastic Automata. 12, IEEE World Congress on Computational Intelligence (1994), 2690–2695. https://doi.org/10.1109/ICNN.1994.374647

[3] Johannes Feldmaier and Klaus Diepold. 2014. Path-finding using reinforcement learning and affective states. *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication* 2014-October, October (2014), 543–548. https://doi.org/10.1109/ROMAN.2014.6926309

[4] Loukas Georgiou and William J Teahan. 2009. Constituent Grammatical Evolution. Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence Constituent (2009), 1261–1268. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-214

[5] Loukas Georgiou and William J Teahan. 2012. Constituent Grammatical Evolution, Ph. D. thesis. (2012).

[6] V Scott Gordon and Zach Matley. 2004. Evolving Sparse Direction Maps for Maze Pathfinding. 1, Congress on Evolutionary Computation (2004), 835–838. https://doi.org/10.1109/CEC.2004.1330947

[7] John H. Holand. 1984. Genetic Algorithms and Adaptation. In: Selfridge O.G., Rissland E.L., Arbib M.A. (eds) Adaptive Control of Ill-Defined Systems. 16 (1984), 317–333. https://doi.org/10.1007/978-1-4684-8941-5_21

[8] Andy Keane. 2015. Genetic Programming , Logic Design and Case-Based Reasoning for Obstacle Avoidance. 9th International Conference on Learning and Intelligent Optimization (2015), 104–118. https://doi.org/10.1007/978-3-319-19084-6

[9] G. Klančar, S. Blažič, and A. Zdešar. 2017. C2-continuous path planning by combining bernstein-bézier curves. *ICINCO 2017 - Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics* 2, Icinco (2017), 254–261. https://doi.org/10.5220/0006406602540261

[10] John R. Koza. 1992. *Genetic Programming On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge.

[11] Joel Lehman and Kenneth O Stanley. 2010. Efficiently Evolving Programs through the Search for Novelty. Proceedings of the Genetic and Evolutionary Computation Conference (2010). https://doi.org/10.1145/1830483.1830638

[12] Joel Lehman and Kenneth O Stanley. 2011. Improving Evolvability through Novelty Search and Self-Adaptation. 19, IEEE International Conference on Evolutionary Computation (2011), 2693–2700. https://doi.org/10.1109/CEC.2011.5949955

[13] Enrique Naredo, Paulo Urbano, and Leonardo Trujillo. 2016. The training set and generalization in grammatical evolution for autonomous agent navigation. *Soft Computing* (2016), 1–8. https://doi.org/10.1007/s00500-016-2072-7

[14] Vinicius Paulo L Oliveira, Eduardo F D Souza, and Claire Le Goues. 2016. Improved Crossover Operators for Genetic Programming for Program Repair Vinicius. 9962 (2016), 112–127. https://doi.org/10.1007/978-3-319-47106-8

[15] Chengshan Qian, Xinfeng Shen, Yonghong Zhang, Qing Yang, Jifeng Shen, and Haiwei Zhu. 2017. Building and Climbing based Visual Navigation Framework for Self-Driving Cars. Mobile Networks and Applications (2017), 1–15. https://doi.org/10.1007/s11036-017-0976-9Building

[16] David Shorten and Geoff Nitschke. 2014. How Evolvable is Novelty Search ? 1, IEEE International Conference on Evolvable Systems (2014), 125–132. https://doi.org/10.1109/ICES.2014.7008731

[17] David Shorten and Geoff Nitschke. 2015. Evolving Generalised Maze Solvers. EvoApplications (2015), 783–794. https://doi.org/10.1007/978-3-319-16549-3

[18] Paulo Urbano, Enrique Naredo, and Leonardo Trujillo. 2014. Generalization in Maze Navigation Using Grammatical Evolution and Novelty Search. 3rd International Conference on the Theory and Practice of Natural Computing (2014), 35–46. https://doi.org/10.1007/978-3-319-13749-0_4

[19] Roby Velez and Jeff Clune. 2014. Novelty Search Creates Robots with General Skills for Exploration. Proceedings of the Genetic and Evolutionary Computation Conference (2014), 737–744. https://doi.org/10.1145/2576768.2598225

[20] David Bruce Wilson. 1996. Generating Random Spanning Trees More Quickly than the Cover Time. Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (1996), 296–303. https://doi.org/10.1145/237814.237880

[21] Jie Xu and Craig S. kaplan. 2007. Image-guided maze construction. 26 (2007). https://doi.org/10.1145/1275808.1276414