Determining the Optimal Search Algorithms to Solve Problems Using a Genetic Algorithm-Based Framework

George Le

Undergraduate Student (CS)

University of Massachusetts Lowell

Lowell, Massachusetts

Email: george\_le@student.uml.edu

*Abstract*—Given a set of search problems to solve, it is unlikely that a single algorithm will be efficient at solving all search problems and even more unlikely to find all of the optimal solution. For processes such as path planning, current techniques are attempting to find a single search algorithm such as D\* that is optimal to tolerably sub-optimal at solving all problems [1][2][3]. However, as every algorithm to date has a sub-optimal worst-case scenario, it would be better to have a collection of multiple search algorithms that an intelligent agent can select depending on the search problem. To optimize the search of different environments, a framework was designed to generate an optimal algorithm selector for a dynamic number of problems, utilizing a genetic algorithm as its learning component. A study on the proof of concept of the framework involved three sample mazes (see Appendix A for the mazes) and a genetic algorithm which as stated above serves as the framework’s supervised learning component.

# Introduction

The genetic algorithm is an evolutionary computation, where an “evolutionary computation refers to computer-based problem-solving systems that uses computational models of evolutionary processes, such as natural selection, survival of the fittest and reproduction, as the fundamental components of such computational systems” [4]. The genetic algorithm used in this study uses a natural selection method; it generates a randomly sized population of random parent solutions, determines the viability of the solutions using a fitness function, selects the fittest parent solutions of the population, and produces a new generation of possible child solutions until a child (the new parents) solution is determined to be the fittest possible solution to the problem or until an *n* number of generations has passed. Variations in child solutions occur with genetic operators: crossover and mutation, controlled by a mutation rate. The natural selection model described is the approach used in this study.

# Framework Architecture

The framework consists of four parts: a maze solver, an algorithm solver, a genetic algorithm, and an algorithm selector (full diagram detailed in Appendix D). The maze solver is the first module of the framework; any problem solver can be substituted with the maze solver as long as scores can be assigned to the potential methods that solve the problem, according to their viability. Again, in the case of the study, the maze solver will pass the optimal path as the optimal solution to the problem and the optimal algorithm as the optimal method to generate the optimal solution. The maze solver’s output is the results of applying searches to the mazes. In the study, the results were tuples of four values for each algorithm. The tuples were comprised of two Boolean values that tracked whether the goal was found and if the path found by the algorithm was optimal, a counter value that tracked the number of operations done by each algorithm, and the time in milliseconds each algorithm took to solve the problems. This data passed to the next module, the scorer, which grades the optimal method on its viability which is judged based on user-defined metrics. The metrics are completely arbitrary and adjustments to the weights of each metrics can change which algorithms are selected. Users can define any metrics to judge the algorithms, however the metrics should be consistent, be attainable by the algorithm, and be represented as data passed from the maze solver. The scorer’s algorithm scores are used in the genetic algorithm’s fitness function. The genetic algorithm is run and repeated a user-specified *n* numbers of time; the data produced in each repetition is stored in a data table and when the repetitions of the genetic algorithms are complete, the table is passed to the algorithm selector. The algorithm selector takes the table and determines what the optimal algorithm is for each of the problems the user provided. The selector determines this by finding the percentages each algorithm was generated by the genetic algorithm to be the optimal search algorithm.

# Methodology of Genetic Algorithm Parameters and design

The parameters of the genetic algorithm are: 100 generations, a population size of 50 individuals, a 1% mutation rate, 50% survival rate, bit string mutation, and two-point crossover. An individual of the genetic algorithm is made up of a Boolean array, where the size of the array is equal to the number of traits these individuals will have and each Boolean of the array tracks whether the search algorithm that the Boolean is mapped to is used (True for used, False for not used). Scores were given to each of the search algorithm for their viability in being the optimal algorithm that best solves the test problems given to the genetic algorithm.

These test problems are made up of a set of three different 10x10 square mazes. Each maze comprises of a perimeter wall, an interior start and node, and interior walls. The problem solver’s search algorithms used as traits in the genetic algorithm are: A\*, best-first, breadth-first, and Dijkstra (uniform-cost search). The metrics that the algorithms were judged on are: completeness (the algorithm found the goal), optimality (the algorithm found the shortest path compared to other algorithms), efficiency (the algorithm used the least amount of operations compared to other algorithms), and time (the amount of time used by the algorithm). Points were given to the algorithms for being complete, for having generated the optimal solution, and for utilizing the least amount of operations and time.

The genetic algorithm outputs the search algorithms mapped to the active traits of the fittest individual of the last generation.

This study also utilized an online maze solver to find the paths, length of paths, number of operations, and time taken by each algorithm generated by the four search algorithm and data [6].

# Results of genetic Algorithm Based Framework

For all of the mazes, the actual ideal search algorithm to be used is the best-first algorithm. The algorithm selector, after the conclusion of each trial of the framework, selected best-first search (see Appendix B for full results of each repetition of the genetic algorithms) (see Appendix C for the percentages of how often each search algorithm was selected). This is primarily due to best-first search being generated as the optimal algorithm by the genetic algorithm 80% of the time for problem 1, 95% of the time for problem 2, and 55% for the last one; best-first beat out all other algorithms in terms of being selected by the genetic algorithm. In all three search problems, the algorithm selector selected the correct optimal algorithm. Future work on this framework can include adding additional algorithms to see if the highest percentage selection method used by the algorithm selector is viable and adding problems that have ties in search algorithm scores to see if the genetic algorithm and selector can still accurately select one of the optimal algorithms.

In analyzing the results, I was surprise that A\* wasn’t selected very often, however I do understand that since the cost to traverse the maze was 1 for each edge, that A\* only has the Manhattan heuristic to determine the path costs. A\* was second best compared to best-first in the third problem, but this appears to be an outlier as UCS in the third problem was selected more often compared to the other problems. A future study can be conducted to determine if it would be viable to use a fast sub-optimal algorithm to determine if a maze is solvable then use a slower optimal algorithm to generate an optimal path for the maze.

# Conclusion

Searching mazes can be very tedious, but this framework of an automated maze solver, algorithm scorer, and a genetic algorithm can be used to optimize the search of many of the same mazes and rooms. Otherwise, one would have to use the same algorithm every time for all search problems (risking high performance costs or inefficiencies in searching).

The practical use of the framework would involve generating all of the possible mazes for a given dimension (x, y), applying multiple search algorithms once on all of the mazes, assign scores for each of the algorithms based on a series of user-defined metrics, providing the scores to the genetic algorithm which will generate populations of individuals with random combinations of traits and then after a user-specified amount of generations should output which algorithm should be used to solve a given maze. This output can be recorded into a lookup table that can be referenced in the future and be applied to solve the maze again.

Mazes were used as test problems due to their similarity to closed environments that a potential robot might encounter and have to traverse. They are also scalable and the maze’s walls can be used to model obstacles. Further studies can take a series of the 2D mazes stacked vertically to form 3D maps, which can be solved by finding an optimal path that solves all the 2D mazes on the stack. Also, modifications to the parameter of the genetic algorithm can be done; studies can be conducted to see how these adjustments to population size, number of generations, etc., affect the accuracy of the genetic algorithm.

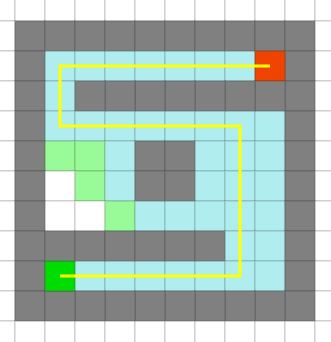
# acknowledgment

The author would like to thank Dr. Jonathan Mwaura for assistance in developing the project proposal, developing the framework, and for guidance on writing this paper.

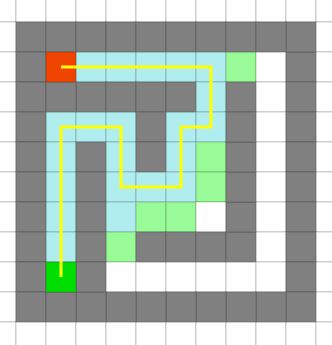
References

1. Dolgov, D., Thrun, S., Montemerlo, M. and Diebel, J. (2017). https://ai.stanford.edu/~ddolgov/papers/dolgov\_gpp\_stair08.pdf. [online] Ai.stanford.edu. Available at: https://ai.stanford.edu/~ddolgov/papers/dolgov\_gpp\_stair08.pdf [Accessed 21 Dec. 2017].
2. Bell, S. (2017). An Overview of Optimal Graph Search Algorithms for Robot Path Planning in Dynamic or Uncertain Environments. [online] stanford.edu. Available at: https://stanford.edu/~sebell/oc\_projects/ieeepaper19Mar10\_stevenbell.pdf [Accessed 21 Dec. 2017].
3. Lavin, A. (2017). A Pareto Optimal D\* Search Algorithm for Multiobjective Path Planning. [online] Arxiv.org. Available at: https://arxiv.org/ftp/arxiv/papers/1511/1511.00787.pdf [Accessed 21 Dec. 2017].
4. Engelbrecht, Andries. “Computational Intelligence: An Introduction.” England: John Wiley & Sons Ltd., 2007. Print.
5. Jiao, Licheng (2000, September) “A Novel Genetic Algorithm Based On Immunity.” IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans. Print. 30.5, 552-561.
6. Qiao.github.io. (2017). PathFinding.js. [online] Available at: https://qiao.github.io/PathFinding.js/visual/ [Accessed 19 Dec. 2017].

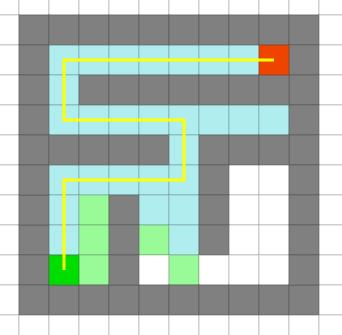
**Appendix A: Test Grids with Best-First Path**



Grid 1



Grid 2



Grid 3

**Appendix B:**

Problem 1 Results

|  |  |
| --- | --- |
| Trial Number | Algorithms Selected |
| Trial 1 | Best-first |
| Trial 2 | Best-first, Breadth-first, UCS |
| Trial 3 | Best-first, Breadth-first |
| Trial 4 | Best-first, Breadth-first |
| Trial 5 | Best-first |
| Trial 6 | Best-first, Breadth-first, UCS |
| Trial 7 | Best-first, Breadth-first, UCS |
| Trial 8 | Breadth-first |
| Trial 9 | A\*, Best-first |
| Trial 10 | Best-first, Breadth-first |
| Trial 11 | Best-first |
| Trial 12 | Best-first, Breadth-first |
| Trial 13 | Best-first |
| Trial 14 | A\*, Best-first |
| Trial 15 | Best-first |
| Trial 16 | Breadth-first, UCS |
| Trial 17 | Best-first |
| Trial 18 | Breadth-first |
| Trial 19 | Best-first, Breadth-first |
| Trial 20 | Best-first, UCS |

Problem 2 Results

|  |  |
| --- | --- |
| Trial Number | Algorithms Selected |
| Trial 1 | Best-first |
| Trial 2 | Best-first |
| Trial 3 | Best-first |
| Trial 4 | Best-first, Breadth-first |
| Trial 5 | Best-first |
| Trial 6 | A\*, Best-first |
| Trial 7 | Best-first |
| Trial 8 | Best-first |
| Trial 9 | Best-first |
| Trial 10 | Best-first, Breadth-first |
| Trial 11 | Best-first |
| Trial 12 | Best-first |
| Trial 13 | Best-first |
| Trial 14 | Best-first |
| Trial 15 | A\*, Best-first |
| Trial 16 | Best-first |
| Trial 17 | Best-first |
| Trial 18 | Best-first |
| Trial 19 | Best-first, UCS |
| Trial 20 | Best-first, UCS |

Problem 3 Results

|  |  |
| --- | --- |
| Trial Number | Algorithms Selected |
| Trial 1 | A\*, Best-first |
| Trial 2 | Best-first, Breadth-first |
| Trial 3 | A\*, Best-first |
| Trial 4 | A\*, Best-first, UCS |
| Trial 5 | A\* |
| Trial 6 | A\*, Best-first, Breadth-first |
| Trial 7 | Breadth-first, UCS |
| Trial 8 | Best-first |
| Trial 9 | Breadth-first, UCS |
| Trial 10 | UCS |
| Trial 11 | A\*, Best-first |
| Trial 12 | A\*, Best-first |
| Trial 13 | Best-first, Breadth-fist |
| Trial 14 | UCS |
| Trial 15 | A\*, UCS |
| Trial 16 | Best-first |
| Trial 17 | Best-first, UCS |
| Trial 18 | A\*, Best-first |
| Trial 19 | Best-first |
| Trial 20 | Best-first |

Scores for Search Algorithms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A\* | Best | Breadth | UCS |
| Problem 1 | 65 | 90 | 75 | 65 |
| Problem 2 | 65 | 100 | 65 | 65 |
| Problem 3 | 65 | 90 | 65 | 75 |

Scores for the Metrics

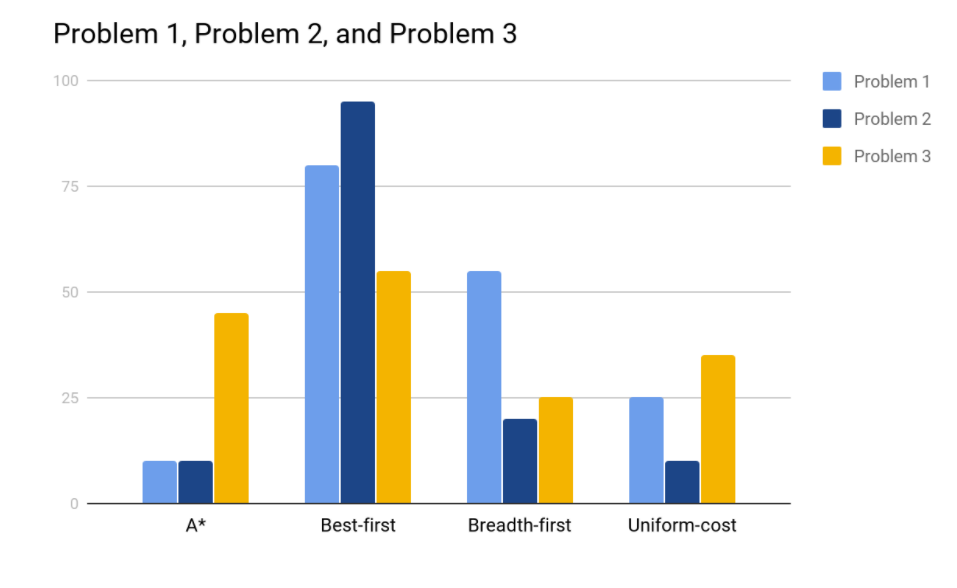
|  |  |
| --- | --- |
|  | Scores |
| Completeness | 40 pts |
| Optimality | 25 pts |
| Efficiency | 25 pts |
| Time | 10 pts |

**Appendix C:**

Percentage of Search Algorithms Selected by Genetic Algorithm

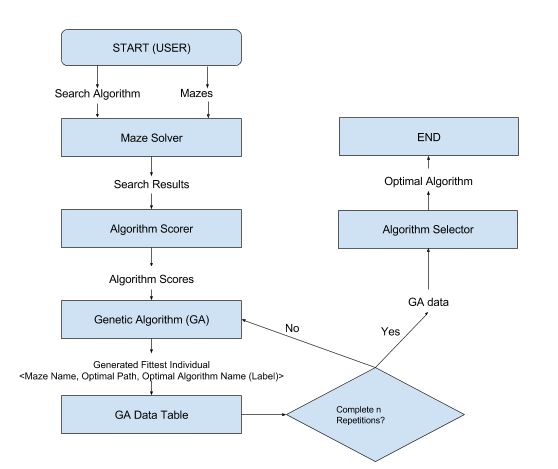
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Best First | Breadth-First | A\* | Uniform-cost |
| Problem 1 | 80% | 55% | 10% | 25% |
| Problem 2 | 95% | 20% | 10% | 10% |
| Problem 3 | 55% | 25% | 45% | 35% |

Graphical Representation of the Above Chart



**Appendix D:**

**Framework Architecture (Note: Algorithms also include score metrics)**



In a general example, the maze solver would be a problem solver, the algorithm scorer would be a solution and problem-solving method scorer, the data from the fittest individual would include problem name, optimal solution, and optimal solving method, and the algorithm selector would be the solving method selector.