Cracking Mazes: The Power of Genetic Algorithms in Maze Solving

Natural Copmuting
Mini Project

Sneha Naidu[1]

Abstract.
In the realm of computational problem-solving, finding efficient paths is key, especially when it comes to navigating through mazes. This project focuses on improving maze-solving efficiency by combining Genetic Algorithms (GA) with the A* pathfinding algorithm. We begin by demonstrating the A* algorithm's ability to immediately pinpoint the ideal path using a heuristic-based method. To build upon this, we integrate GA, capitalizing on its iterative selection, crossover, and mutation processes to further optimize our solutions. In addition, the performance of GA as an independently maze-solving method is evaluated.

The results show while the A* algorithm solves mazes quickly, combining it with GA produces even more optimal solutions in a manageable timescale, in contrast to the significantly slower performance of GA when used alone. This project investigates the possibility of improving problem-solving approaches by combining traditional rule-based algorithms with evolutionary algorithms, with the goal of developing more sophisticated and effective solutions.

1 Introduction

Embarking on this project, my aim was to address a fundamental question in the domain of computational problem-solving: how can we improve maze-solving strategies in increasingly complex environments? The A* method, known for its heuristic efficiency and generally used for structured issues like maze navigation, was important to our investigation. However, during my research, I discovered that the static aspect of A*'s heuristic approach could be restrictive, especially when confronted with mazes with dynamic changes or irregular patterns. For instance, consider a maze that changes its paths or introduces new obstacles; the A* algorithm, relying simply on its pre-defined heuristic, may struggle to adapt to these changes efficiently. Such instances highlight the need for more adaptable, flexible solutions. This realization directed my research toward Genetic Algorithms (GA), which are renowned for their evolutionary principles and capacity to optimize solutions in constantly changing problem areas.
I aimed to explore whether the deterministic precision of A* with the evolutionary flexibility of GA could overcome the constraints of each method when used separately. The goal of this study was to create an integration between two independent algorithmic approaches that could perhaps lead to more robust answers in complex maze-solving problems. Furthermore, by studying GA's performance in isolation, I hoped to present a thorough analysis that would not only evaluate the efficiency of each method but also reveal GA's unique potential in handling mazes on its own.
Next, the Literature Review will dive into the research and studies about the A* algorithm, Genetic Algorithms (GA), and how they might work together. This section is key for showing how my project fits into the wider world of algorithms, drawing on what's already known and applying it in a new way.

[1] School of Computing and Mathematical Sciences, University of Greenwich, London SE10 9LS, UK, email: sn1653j@gre.ac.uk

2 Literature Review

The journey of this project was significantly influenced by a collection of insightful academic papers, each contributing to a deeper understanding of the problem at hand.
The research into the A* method was heavily influenced by the paper "A-based Pathfinding in Modern Computer Games" from ResearchGate [2]. The insights provided in this study into A*'s application in gaming environments showed its versatility and efficiency, which influenced my use of A* in the project. In addition, "A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games" [6]from MDPI's Applied Sciences provided a broader picture of A* in dynamic circumstances, validating its choice for my maze-solving investigation.
The comprehensive analysis published in Springer's Multimedia Tools and Applications, titled "A review on genetic algorithm: past, present, and future," [5] had a big impact on how I understood and used Genetic Algorithms (GA). This paper gave important historical background and theoretical support for GAs. An arXiv [1] study taught me even more, especially about GA's selection, crossover, and mutation processes, which helped me figure out how to use them in my project.

The idea of merging A* with GA was solidified by the theoretical insights from "Algorithms for path optimizations: a short survey" in Springer's Computing. Although not exclusively about A* and GA, this paper supported the idea that combining different algorithmic approaches could result enhanced problem-solving.

Additionally, I was motivated to experiment with GA as a standalone method for maze solving, drawing from the discussions in both the Springer and arXiv papers. These works helped me a lot with planning my experiment, which let me to test and learn about the pros and cons of GA on its own in the context of maze navigation.

Finally, the cumulative insights provided by these studies were critical in the creation and direction of my project. They not only inspired a blend of classic and novel ways in handling the maze-solving difficulty, but they also grounded my approach in existing research.

3 Experiments and results

In this section, we delve into the core of our research, exploring the outcomes of our experiments with the

A. A* Algorithm: The Foundation
B. Hybrid Approach: A* Enhanced by GA
C. Genetic Algorithm: Independent Exploration

We will analyze the results obtained from each approach, evaluating their effectiveness in maze solving and the impact of different algorithmic strategies on the final solutions. This comprehensive study aims to provide a clear understanding of how each method performs individually and in combination, highlighting the advancements and challenges encountered during the process.

A. A* Approach : The Foundation

The initial phase of my study focuses on the A* pathfinding algorithm. To build a performance baseline, it was critical to understand A*'s independent performance in maze-solving scenarios. This exploration on A*'s independent capabilities was important for identifying its strengths and limitations. This fundamental understanding of A* served as the foundation for developing and testing the hybrid approach with GA.

1. Explanation

Initial Pathfinding with A*: The A* algorithm makes use of the Manhattan Distance heuristic, which is particularly well-suited for applications that are grid-based. By calculating the sum of the absolute differences in the coordinates of the nodes, this heuristic makes the process of estimating the cost between the nodes more straightforward. To broaden our understanding and explore alternative heuristics, we also implemented the A* algorithm using Euclidean Distance in certain scenarios. Unlike Manhattan Distance, Euclidean Distance measures the straight-line distance between two points, which can be more direct but less reflective of actual path costs in a grid maze.

To demonstrate the difference between Manhattan and Euclidean heuristics in the A* algorithm's pathfinding, a comparative graph is presented.

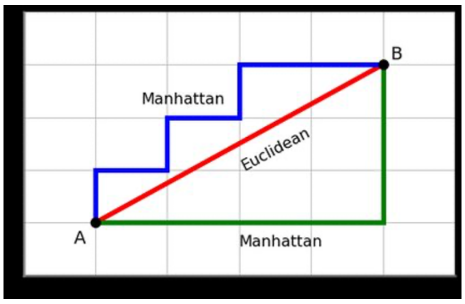


Figure 1.

In both approaches, the A* algorithm employs a priority queue to effectively manage node selection based on a composite cost function. This function combines the actual distance traversed (g-cost) with the estimated distance to the goal (h-cost), ensuring that the search is consistently oriented towards the most promising paths. This methodology involves a systematic process of analyzing, recalculating, and updating routes based on the specific costs associated with each path.

Path Generation Process: The goal is not the only thing that matters; it's also how to reach that goal as quickly as possible. The algorithm finds a happy medium between the amount of ground that has been traversed and the expected path that lies ahead. After the goal has been accomplished, the A* algorithm will retrace its steps along the path until it reaches the starting point, thereby reconstructing the route that was travelled. It is crucial to perform this retrace since it represents the solution to the algorithm.

2. OutPut :

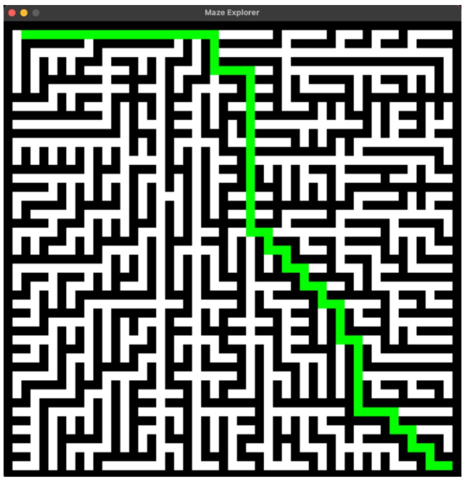
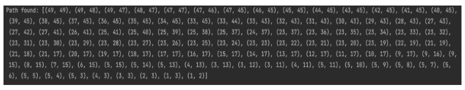


Figure 2.

The output from the A* algorithm provided a direct, unambiguous path through the maze, as opposed to the iterative approach seen with Genetic Algorithms (GA). The result was a clear sequence of coordinates, illustrating the step-by-step journey from the maze's entrance to the exit.

The path found was: This sequence represents a series of moves,



```
Path found: [(49, 49), (49, 48), (49, 47), (48, 47), (47, 47), (47, 46), (47, 45), (46, 45), (45, 45), (44, 45), (43, 45), (42, 45), (41, 45), (40, 45),
 (39, 45), (38, 45), (37, 45), (36, 45), (35, 45), (34, 45), (33, 45), (33, 44), (33, 43), (32, 43), (31, 43), (30, 43), (29, 43), (28, 43), (27, 43),
 (27, 42), (27, 41), (26, 41), (25, 41), (25, 40), (25, 39), (25, 38), (25, 37), (24, 37), (23, 37), (23, 36), (23, 35), (23, 34), (23, 33), (23, 32),
 (23, 31), (23, 30), (23, 29), (23, 28), (23, 27), (23, 26), (23, 25), (23, 24), (23, 23), (23, 22), (23, 21), (23, 20), (23, 19), (22, 19), (21, 19),
 (21, 18), (21, 17), (20, 17), (19, 17), (18, 17), (17, 17), (16, 17), (15, 17), (14, 17), (13, 17), (12, 17), (11, 17), (10, 17), (9, 17), (9, 16), (9,
 15), (8, 15), (7, 15), (6, 15), (5, 15), (5, 14), (5, 13), (4, 13), (3, 13), (3, 12), (3, 11), (4, 11), (5, 11), (5, 10), (5, 9), (5, 8), (5, 7), (5,
 6), (5, 5), (5, 4), (5, 3), (4, 3), (3, 3), (2, 3), (1, 3), (1, 2)]
```

Figure 3.

where each pair of numbers corresponds to a specific point in the maze grid. The first number in each pair denotes the row, and the second number denotes the column.

3. Code Implementation and Structure

In this project, the implementation of the A* algorithm for maze solving is organized across several key Python files: AStar.py, gui.py, maze.py, player.py, and utils.py.

AStar.py:

Role: This file is central to the implementation. It defines the MazeSolver class, which encapsulates the A* algorithm.

Functionality: Responsible for determining the path from the start to the end within the maze. It employs utility functions from utils.py for heuristic calculations and pathfinding.

gui.py:

Role: Manages the graphical user interface of the application.

Functionality: This file is responsible for initializing the display window, visualizing the maze, and updating the screen to show the maze-solving process in action. It uses pygame for graphical rendering and interaction.

bmaze.py:old

Role: Essential for creating and managing the maze structure.

Functionality: It handles the maze's data, such as walls and paths, and provides the necessary information to the A* algorithm in AStar.py for navigating the maze.

player.py:

Role: Represents an entity within the maze, potentially used for simulating movement or additional logic.

Functionality: Could include logic for evaluating paths, moving through the maze, or other functionalities depending on the project's scope.

utils.py:

Role: A utility module providing support functions.

Functionality: Contains crucial functions like the Manhattan distance for heuristic calculations and pathfinding algorithms. It supports the A* algorithm with critical functions for calculating distances and path reconstruction.

Now that we have established a solid understanding of the A* algorithm's capabilities, let's explore how integrating Genetic Algorithms (GA) can enhance the solution.

B. Hybrid Approach: A* Enhanced by GA

The hybrid technique merges the heuristic accuracy of A* with the evolutionary problem-solving abilities of GA. The main objective is to utilize the efficient pathfinding capabilities of the A* algorithm as a foundation, which is subsequently improved and optimized by the Genetic Algorithm (GA) over multiple generations.

1. Explanation

Initial Pathfinding with A*: In this hybrid approach, the initial phase still relies on the A* algorithm to kickstart the maze-solving process. However, the focus shifts slightly as we lay the groundwork for subsequent optimization by GA. While A* uses the same Manhattan Distance heuristic to estimate the cost from any node to the goal and employs a priority queue for node evaluation, the paths it generates are now viewed as possible solutions in the future. Even though these first paths work, they can be made even better through GA's evolutionary processes.

Path Generation Process: A* algorithm generates a path by following the most cost-effective route from the starting point to the destination. It takes into account both the distance previously travelled and the predicted distance that still needs to be covered. After achieving the objective, A* algorithm retraces its steps from the goal to the starting point, rebuilding the path it previously took. This particular route represents the initial successful solution to the maze and acts as a reference point for Genetic Algorithm .

Incorporating A* Path into GA: The Genetic Algorithm (GA) then takes over, using the path found by A* as one of its initial potential solutions. This inclusion ensures that the GA has a solid, viable starting point, influencing its evolutionary process greatly. In our GA setup, a population of these path solutions are created . To add genetic diversity, the population with different versions of the A* path are chosen.

Evolutionary Process of GA: The key component of the genetic algorithm used in our code is the progressive development of these pathways across successive generations. It achieves this through the processes of selection, crossover, and mutation.

Selection Fitness Evaluation:

The core of the selection process lies in evaluating the fitness of each path within the GA's population. Fitness is determined by two major factors: the length of the path and its ability to navigate the maze without colliding with obstacles. The fitness function in the Player class, in particular, analyses the effectiveness of a path by taking into account the entire distance walked (pathLength) and penalizing any interactions with obstacles (obstaclePenalty), using the formula:

$$Fitness = [(pathLength + obstaclePenalty)] \tag{1}$$

2. Parent Selection: The GA then chooses the parent pathways that will be used to produce the subsequent generation. Our code makes use of a method called (_pickParents), which sorts the population according to fitness and chooses the paths that perform the best and most efficiently. In this way, it is ensured that the next generation will be generated from the most proficient parents of the existing population.

Crossover Combining Parent Paths: GA's ability to create new solutions is dependent on the crossover operation. Segments from two parent pathways are merged to form child paths in this process. This method allows the offspring to inherit the abilities of both parents, leading to unique and more efficient path solution.

The crossover point is calculated as 80% of the length of the fitter

$$\text{parent's path}\ \ index = [(80 * len(maxParent.path)//100)] \tag{2}$$

This means the child inherits the first 80% of its path from the fitter parent.

Diversity in Offspring: By producing offspring that blend characteristics of parent paths, the crossover process introduces diversity to the population. This variety is essential for exploring various parts of the maze and determining optimal routes.

Mutation The mutate method introduces random changes to the player's path, enhancing diversity and exploration: Random Path Truncation:

Int(mapFromTo(random.random(),0,1,1,len(self.path))

This reduces the player's route at a random point.

This operation effectively shortens the path to a random length, after which the player may need to find a new way to the goal during subsequent evaluations.

Enhancing Diversity:

By randomly altering the paths of players, mutation introduces new characteristics and variations in the population. This is crucial for exploring new potential solutions and avoiding getting stuck in less optimal paths.

Iterative Improvement and Finding the Best Way: The GA improves the routes over and over again as new generations come along. The algorithm learns and changes with each generation, which could lead to more efficient methods. At the end of this process, the best way is found, which can be seen by its high fitness score. This path is the GA's best way to get through the maze, showing that it is better than the first path that A* gave.

The hybrid approach effectively combines the strengths of both the A* algorithm and the Genetic Algorithm (GA). Through this integration, we have achieved a powerful synergy where the structured pathfinding of A* is dynamically enhanced by the evolutionary capabilities of GA. The iterative process of selection, crossover, and mutation within GA not only refines the initial paths provided by A* but also introduces innovative solutions, leading to the discovery of the most efficient route through the maze.

Diversity in Offspring:2. Output Analysis :

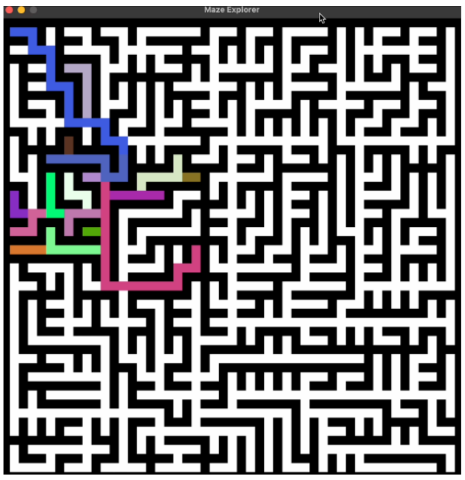


Figure 4.

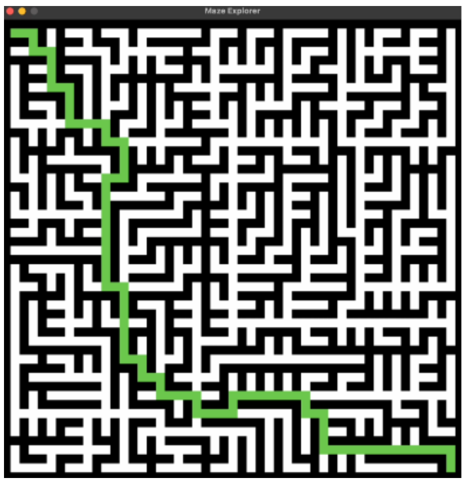


Figure 5.

Progress Across Generations: The output shows a series reflecting the algorithm's progress across multiple generations, beginning with "Current generation: 0" and ending with "Current generation: 4." This shows the Genetic Algorithm's evolutionary process, in which each generation represents a new set of solutions evolved from the preceding one.

Optimal Path Found: The optimum path is found and displayed in a set of coordinates beginning at (1, 1) and progressing to (49, 49). The Genetic Algorithm determined the most cost-effective route after multiple rounds, effectively navigating the maze from start to finish.

Fitness : The fitness notation "Fitness: 1.0" indicates that the best path found has attained the maximum fitness score. This indicates that the path not only achieves the goal but also does it in the most efficient way. Explanation of Individual Files

The first image is the GA attempting to examine all feasible pathways. The second image depicts a maze being solved.

Progress Across Generations: The output shows a series reflecting the algorithm's progress across multiple generations, beginning with "Current generation: 0" and ending with "Current generation: 4." This shows the Genetic Algorithm's evolutionary process, in which each generation represents a new set of solutions evolved from the preceding one.

Optimal Path Found: The optimum path is found and displayed in a set of coordinates beginning at (1, 1) and progressing to (49, 49). The Genetic Algorithm determined the most cost-effective route after multiple rounds, effectively navigating the maze from start to finish.

Fitness : The fitness notation "Fitness: 1.0" indicates that the best path found has attained the maximum fitness score. This indicates that the path not only achieves the goal but also does it in the most efficient way. Explanation of Individual Files

2. Code Implementation and Structure To use this hybrid technique and solve the maze, several important components are required. The project's codebase is divided into four main files: gui.py, ga.py, player.py, and utils.py.

gui.py: Manages the maze's user interface and visual representation. The display window is initialized, the maze is created, and the screen is updated to show the maze-solving process in action.

ga.py: Contains the core logic of the Genetic Algorithm. Manages the GA population, implements selection, crossover, and mutation functions. Determines the fitness of paths and selects the best solutions for future generations.

Player.py: Represents an individual in the GA population. Contains logic for path evaluation, movement through the maze, and fitness calculation. Crucial for understanding how each potential solution (path) behaves and evolves over time.

utils.py: Provides utility functions like manhattan for heuristic calculation in A* and pathfinding for initial path generation. Supports the main algorithms with essential calculations and path reconstructions.

C. Genetic Algorithm: Independent Exploration

After diving into the technique of integrating the A* algorithm with Genetic Algorithms (GA) and studying A*'s standalone capabilities in maze solution, it's time to turn our attention entirely to the Genetic Algorithm. This independent examination of GA will not only highlight its strengths and limitations but also provide a comprehensive comparison with the previously discussed A* algorithm and A* and GA Hybrid approach.

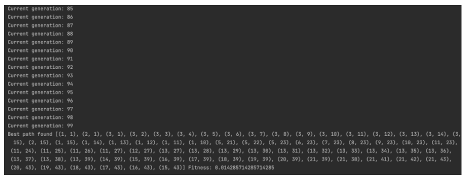
Initializing the GA Process: The process begins by creating the initial population for the GA. Each member of this population represents a potential maze-solving path, characterized by unique attributes and decision-making capabilities. Unlike the hybrid approach, where the GA's population starts with solutions influenced by A*'s pathfinding, the GA-only method introduces a wider spectrum of initial conditions. This range can have random or planned modifications to the starting paths, which makes sure that the genetic pool is diverse. This kind of variety is very important because it gives the algorithm a lot of room to try out and improve different maze-navigating tactics.

The diversity of the original population is a key difference between this method and the hybrid method. Because of A*'s input, the pathways in the hybrid model are partially optimized from the start. In the pure GA technique, however, the paths can be completely random, even impractical, reflecting a greater range of possibilities and obstacles for the algorithm to conquer.

Evolutionary Mechanics of GA: Once the GA is initialized, it starts its evolutionary journey. This process is based on the basic ideas of mutation, natural selection, and genetic crossover: **Selection:** The fitness of each road is judged by how well it gets through the maze. The fittest paths are chosen to pass their genes to the next generation. e.i Part of the decision process is figuring out whether each road is possible by seeing how well it gets through the maze. The most suitable ways are chosen so that their genes can be passed on to the next generation. **Crossover:** This is the process of mixing segments from two different routes to create new offspring who may acquire successful features from both parents. **Mutation:** Random changes are made to some pathways during the mutation process to protect genetic diversity and look into previously unknown routing methods.

Iterative: Improvement and Solution Discovery: The A* and GA combined approach gives GA a starting solution that is already optimized. The GA-only method, on the other hand, may need a lot more time and generations to find efficient paths. This is because it starts with a much larger and more random set of starting solutions. In this case, the GA faces the challenge of not only refining but often fundamentally reshaping these initial paths. Because of this, the algorithm might have a harder time finding the best answer, which would mean that it would need to go through a longer iterative evolution process. In this iterative cycle, each generation slowly moves the population closer to finding a better way through the maze. However, this progress can be slower and less reliable than A*'s help with the start. This difference shows the balancing act between the exploratory depth of a pure GA method and the speed of a hybrid method that uses A*'s initial pathfinding abilities.

Diversity in Offspring: Output



```
Current generation: 85
Current generation: 86
Current generation: 87
Current generation: 88
Current generation: 89
Current generation: 90
Current generation: 91
Current generation: 92
Current generation: 93
Current generation: 94
Current generation: 95
Current generation: 96
Current generation: 97
Current generation: 98
Current generation: 99
Best path found [(1, 1), (2, 1), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (3, 7), (3, 8), (3, 9), (3, 10), (3, 11), (3, 12), (3, 13), (3, 14), (3,
  15), (2, 15), (1, 15), (1, 14), (1, 13), (1, 12), (1, 11), (1, 10), (5, 21), (5, 22), (5, 23), (6, 23), (7, 23), (8, 23), (9, 23), (10, 23), (11, 23),
 (11, 24), (11, 25), (11, 26), (11, 27), (12, 27), (13, 27), (13, 28), (13, 29), (13, 30), (13, 31), (13, 32), (13, 33), (13, 34), (13, 35), (13, 36),
 (13, 37), (13, 38), (13, 39), (14, 39), (15, 39), (16, 39), (17, 39), (18, 39), (19, 39), (20, 39), (21, 39), (21, 38), (21, 41), (21, 42), (21, 43),
 (20, 43), (19, 43), (18, 43), (17, 43), (16, 43), (15, 43)] Fitness: 0.014285714285714285
```

Figure 6.

Analyzing the results of the Genetic Algorithm (GA) in the maze-solving task, it's evident that the GA progressed through 99 generations, indicating a thorough search effort. The best path found, as detailed in the output, along with its fitness score (0.014285714285714285), suggests that while the GA managed to evolve a solution, this solution was not highly efficient. The low fitness score implies that the path is either longer or more complex than an ideal solution would be. This outcome is typical of GAs in complex problem spaces like maze-solving, where finding the most efficient path can be challenging due to the randomness and exploratory nature of genetic algorithms. The results show that while the GA is working towards a solution, further iterations, parameter tuning, or a more guided approach (like a hybrid GA-A* method) might be needed to achieve a more optimal path.

3. Code

Here's an overview of what happens in the code: **1. Initialization of GA Process:** • The GA class initializes the genetic algorithm process. • It starts by creating an initial population of potential solutions (Player objects) for the maze. • Each Player in the population represents a unique maze-solving path with distinct attributes and decision-making strategies. • The initial population is diverse, varying from logical to potentially impractical paths. This diversity is crucial for exploring a wide range of maze-solving strategies.

2. Population Initialization: • In the initPopulation method, the population is populated with Player objects. • Each Player is created with starting (init) and ending (end) points of the maze, and the maze itself. • This step is critical for establishing a varied genetic pool, setting the stage for the evolution of maze-solving strategies.

3. Fitness Evaluation: • The _fitness method is used to evaluate the fitness of each Player. • Fitness evaluation is essential as it determines how well each Player solves the maze, guiding the selection process.

4. Selection Process: • The _selection method simulates the natural selection process. • Pairs of Player objects are randomly chosen to create offspring through crossover. • This process gradually leads to the evolution of more effective maze-solving strategies in the population.

5. Mutation Process: • The _mutation method introduces random changes to the Player objects. • Mutations ensure genetic diversity and can lead to new maze-solving abilities in the population.

6. Generation Advancement: • The nextGen method progresses the algorithm to the next generation. • It updates the current generation, performs selection, mutation, and re-evaluates the fitness of each new Player. • The best Player in terms of fitness is identified. If this Player achieves perfect fitness (1), the victory flag is set, indicating that an optimal solution to the maze has been found.

7. Diverse Approach: • Unlike the hybrid GA-A* method where paths are partially optimized from the start due to A*'s input, this pure GA method starts with a broad range of possible paths. • The diversity in the initial population allows the algorithm to explore various maze-solving tactics, making the process dynamic and potentially more robust against a variety of maze configurations.

4 Discussion

Contextual Implications: This study expands the understanding of algorithmic problem-solving in mazes. The combination of heuristic-based and evolutionary algorithms, shown by the A* and GA hybrid, demonstrates a growing trend in computational problem-solving that prioritizes flexibility and efficiency. refference.[3] This study contributes to the current debate on the potential combination of several algorithmic approaches to improve performance in dynamic contexts.

Effective problem solving often requires solutions that can handle a wide variety of problems, from simple to complex. By integrating heuristic-based pathfinding algorithms with genetic algorithms, we can make that happen.

Unforeseen Results and Analysis: One interesting finding in the study was that the A* and GA hybrid method did better in a moderately complex maze, revealing its potential for broader application in problem-solving. The A* algorithm, on the other hand, tends to do best in situations that are more predictable or straightforward.

Real-Life Examples of Hybrid Model Applicability: referencing for all

Autonomous Vehicle Navigation: In the real world, autonomous cars must navigate through dynamic surroundings filled with unexpected obstacles. While A* can create efficient routes under predictable conditions, the hybrid model may be able to better respond to sudden adjustments, such as road closures or unexpected traffic, by recalculating routes on the fly. [7]

Dynamic Game Level Design: The hybrid approach could provide more adaptable AI pathfinding in video game creation, particularly in games with dynamic level design, where the environment changes based on player activities. In contrast to A*, which may struggle with the level's dynamic nature [9]

Supply Chain Management: In supply chain management, particularly in crisis scenarios such as natural disasters, the hybrid model could be used for dynamic logistics routing where it needs to rapidly adapt to rapidly changing situations such as blocked routes or supply shortages. [4]

The hybrid A* and GA model's ability to solve and adapt to a wide range of problems shows that it could work well in unpredictable and quickly changing real-life situations. This ability to change, which might not be possible with just the A* algorithm, is a big plus. Additionally, relying only on GA has its problems, mostly because it takes longer to find answers. This processing delay makes the GA-only method less useful in situations where time is of the essence, showing once again how useful and efficient the hybrid model is.

5 Conclusion and future work

Conclusion

In this paper, we investigated maze-solving algorithms in depth, focusing on the unique characteristics and performance of the A* algorithm, the Genetic Algorithm (GA), and their hybrid integration. Our trials demonstrated the advantages and disadvantages of each strategy. The A* algorithm, known for its heuristic-based precision, performed exceptionally well at navigating predicted mazes. However, its reliance on pre-defined heuristics indicates that it may have drawbacks in more dynamic situations where adaptation is essential. On the other hand, despite its slower rate of discovering answers, GA excelled in its ability to evolve and adapt, proving to be a solid choice for contexts where change is constant.

The hybrid A*/GA model came up as a particularly effective technique, combining A*'s structured pathfinding with GA's evolutionary flexibility. This combination provided a better balanced strategy capable of managing the maze problems. It demonstrated the ability to be highly effective in a variety of settings by balancing predictability and adaptability. The hybrid model's performance in our controlled studies suggests that it has broader relevance, possibly extending to real-world scenarios that require both precision and the ability to adapt to changing conditions.

Future Work

The findings from my study, while comprehensive, naturally lead to further questions and areas for exploration. Work in the Future While my study's findings are extensive, they naturally raise more issues and possibilities for investigation. Recognizing the limited scope of my current research, I propose the following future research directions to improve the knowledge and applicability of maze-solving algorithms:

Moving Dynamic Maze: [8]

• **Creating a Dynamic Maze:** Extending my research to include dynamic mazes would provide a more complex test environment, allowing for a more in-depth evaluation of the algorithms' real-time adaptation.

• **Simulation Studies:** Running simulations that mimic real-world unpredictability would improve the algorithms' practical use and give a more robust test of their capabilities.

• **Behavioural Analysis:** A more detailed analysis of how each algorithm adapts to changing conditions would complement my current findings and offer a broader perspective on their flexibility.

Advanced GA Methods [10]

• **Integrating GA with Other Algorithms:** Combining GA with other evolutionary or machine learning algorithms could result in more advanced problem-solving approaches.

• **Parameter optimization:** Fine-tuning the parameters of GA could lead to improved efficiency, building on the foundation built in my study.

• **Real-World Applications:** [9] Applying the insights acquired from my research to different domains, such as robotics or urban planning, will test the algorithms' flexibility in a variety of practical circumstances. • Exploration of how these algorithms work in user-driven environments would provide vital insights into their usability and real-world effectiveness.

In conclusion, while my research has shed light on the capabilities of various maze-solving algorithms, there is still a large landscape of opportunities for furthering this research. The planned future effort intends to expand present understanding, investigate new applications, and further knowledge growth in the field of algorithmic problem-solving.

ACKNOWLEDGEMENTS

I express profound gratitude to Dr. Mohammad Majid al-Rifaie for his invaluable guidance and constant support over the course of this project. The insightful feedback and patient mentorship provided by him have played a vital role in shaping this study. I sincerely appreciate his dedication and the significant role he has played in my academic journey.

REFERENCES

[1] Tanweer Alam, Shamimul Qamar, Amit Dixit, and Mohamed Benaida, 'Genetic algorithm: Reviews, implementations, and applications', *arXiv preprint arXiv:2007.12673*, (2020).

[2] Xiao Cui and Hao Shi, 'A*-based pathfinding in modern computer games', *International Journal of Computer Science and Network Security*, **11**(1), 125–130, (2011).

[3] SCMS De Sirisuriya, TGI Fernando, and MKA Ariyaratne, 'Algorithms for path optimizations: a short survey', *Computing*, **105**(2), 293–319, (2023).

[4] Sunil Kumar Jauhar and Millie Pant, 'Genetic algorithms in supply chain management: A critical analysis of the literature', *Sādhanā*, **41**, 993–1017, (2016).

[5] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar, 'A review on genetic algorithm: past, present, and future', *Multimedia tools and applications*, **80**, 8091–8126, (2021).

[6] Sharmad Rajnish Lawande, Graceline Jasmine, Jani Anbarasi, and Lila Iznita Izhar, 'A systematic review and analysis of intelligence-based pathfinding algorithms in the field of video games', *Applied Sciences*, **12**(11), 5499, (2022).

[7] Jatin Luthra, Abhishek Sharma, and Shubham Kaushik, 'Implementation of genetic algorithm for path estimation in self driving car', *SN Computer Science*, **3**(2), 147, (2022).

[8] Alex Fernandes da V Machado, Ulysses O Santos, Higo Vale, Rubens Gonçalvez, Tiago Neves, Luiz Satoru Ochi, and Esteban W Gonzalez Clua, 'Real time pathfinding with genetic algorithm', in *2011 Brazilian Symposium on Games and Digital Entertainment*, pp. 215–221. IEEE, (2011).

[9] Evan Kusuma Susanto, Rifqi Fachruddin, Muhammad Ihsan Diputra, Darlis Herumurti, and Andhik Ampuh Yunanto, 'Maze generation based on difficulty using genetic algorithm with gene pool', in *2020 International Seminar on Application for Technology of Information and Communication (iSemantic)*, pp. 554–559. IEEE, (2020).

[10] Mingyuan Zhao, Chong Fu, Luping Ji, Ke Tang, and Mingtian Zhou, 'Feature selection and parameter optimization for support vector machines: A new approach based on genetic algorithm with feature chromosomes', *Expert Systems with Applications*, **38**(5), 5197–5204, (2011).