[1]

Mobile Robot Path Planning Algorithm

Michael Aguadze, Adric J. Crawford, and Vincent M. Grady,

***Abstract*—This project investigates the use of a Genetic Algorithm (GA) for path planning in 2D static environments, with the objective of identifying the shortest, collision-free route for a mobile robot. A custom GA was developed in MATLAB to evolve candidate paths encoded as binary chromosomes, and heuristic enhancements such as global elitism and diversity-based resets were implemented to reduce premature convergence. The algorithm was evaluated on both small-scale (15-node) and large-scale (100-node) maps, including pseudo random environments. While the GA performed reliably in smaller test cases, it struggled to generate feasible paths efficiently in larger maps due to excessive chromosome length, slow execution, and an exponential increase in solution space complexity. The findings underscore both the promise and limitations of GAs in real-time autonomous navigation, and suggest future improvements in encoding, platform efficiency, and selection mechanisms are needed to support large-scale deployment.**

***Index Terms*— Genetic Algorithm, Autonomous Vehicle, Path Planning**

I. Introduction

Mobile robots are autonomous systems capable of perceiving their environment, processing sensory data, and planning feasible paths toward predefined goals. In structured and semi-structured environments, path planning plays a crucial role in ensuring safety, efficiency, and optimal energy consumption. Applications of mobile robots span domains such as healthcare, agriculture, military reconnaissance, and industrial automation. Effective navigation algorithms must enable robots to traverse from a start point to a destination while avoiding obstacles and minimizing distance or cost.

Classical path planning approaches—like A* search or roadmap methods—guarantee results under certain conditions but suffer from scalability and flexibility issues in complex or dynamic settings. In contrast, heuristic-based methods such as Genetic Algorithms (GA)s offer a promising alternative by framing path planning as an optimization problem. This work presents a GA-based path planner designed to generate shortest collision-free trajectories within a 2D static map.

II. Problem Statement

The goal of this project is to design and implement a Genetic Algorithm that computes the shortest feasible path for a mobile robot navigating from a known start point to a fixed endpoint in a 2D grid-based environment with static obstacles. Each candidate solution is a sequence of intermediate via points encoded in binary, and the GA iteratively evolves these paths to maximize a fitness function that favors feasibility and minimal path length. To reduce premature convergence and improve result quality, heuristic mechanisms were incorporated into the solution space such as global elitism. We would like to evaluate the effectiveness of this algorithm style on a large map to more accurately demonstrate capabilities.

III. Attempted Solution

The problem was formulated as a combinatorial optimization task in which each path is treated as a chromosome composed of a series of via points. The fitness of each path is evaluated based on its total Euclidean length and whether it avoids collisions with obstacles. Feasible paths are scored inversely proportional to their length, while infeasible paths receive heavy penalties to discourage their selection.

A static workspace is initially generated using a known global map of the environment. During operation, local maps are rapidly generated from sensor data collected by the ego vehicle, allowing the algorithm to respond to environmental changes while still referencing the global structure. While the map remains unchanged during the algorithm's execution cycle, this hybrid approach enables more reactive and realistic navigation planning in dynamic settings.

The robot must travel from a defined start location (e.g., point 1) to a fixed destination (e.g., point 15) through intermediate waypoints. The GA evolves a population of such candidate paths through successive generations using standard operators like selection, crossover, and mutation.

[1]Manuscript received May 6, 2025.

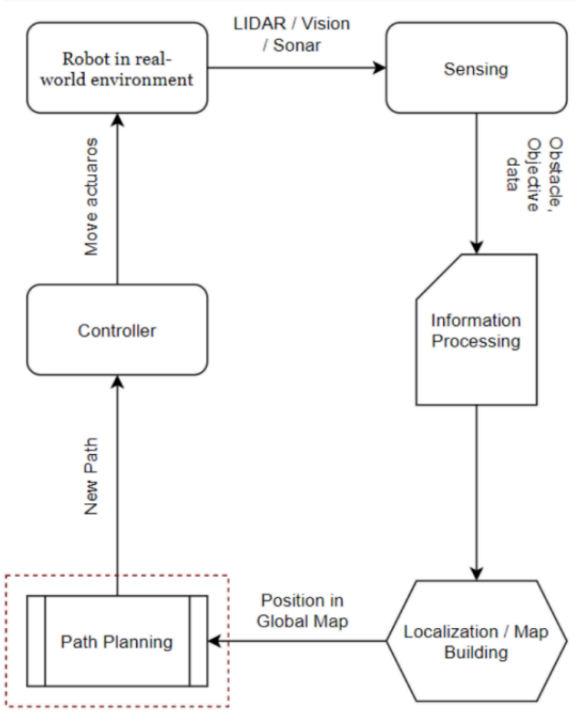
Michael Aguadze with Electrical Engineering Department, North Carolina Agricultural and Technical State University, (e-mail: maguadze@aggies.ncat.edu).

A. J. Crawford was with the Electrical Engineering Department, North Carolina Agricultural and Technical State University, (e-mail: ajcrawford@aggies.ncat.edu).

V. M. Grady is with the Electrical Engineering Department, North Carolina Agricultural and Technical State University, (e-mail: vmgrady@aggies.ncat.edu).

2

Fig 1. Operational flow chart



IV. Genetic Algorithm Implementation

A. High-Level Overview

The Genetic Algorithm used in this project is a population-based search method designed to optimize path quality under defined constraints. Individuals represent possible paths, and the population is evolved over iterations with the goal of improving overall fitness. The GA was custom implemented in MATLAB for complete control over encoding, genetic operations, and visualization.

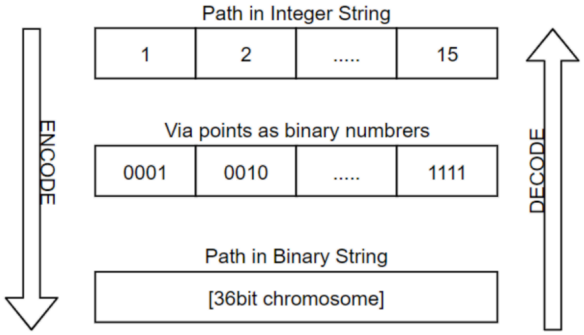
B. Encoding Scheme

The adopted encoding approach is binary, where each via point in a path is represented by a binary string. 4-bit strings were used for the initial 15 node tests. This encoding was chosen for its compatibility with classical GA operators and its widespread use in literature. For the initial phase of testing—where the workspace consisted of 15 predefined via points—each integer (from 1 to 15) was converted into a unique 4-bit binary value. This allowed the representation of all via points within a single chromosome of fixed length.

A complete path is constructed by concatenating the binary encodings of each via point in the sequence. For example, a path like [1, 2, 5, 15] would be encoded as 0001 0010 0101 1111. This representation supports the application of crossover and mutation operators at the bit level and simplifies integration with other components of the GA.

The total chromosome length depends on the number of via points used. In early experiments, where path lengths were fixed, the chromosome length followed the rule $(m+2)\times 4$, where m is the number of obstacles and 4 is the number of bits per via point.

Fig 2. Encoding Diagram



C. Parameters

The parameters used in this genetic algorithm were chosen based on the requirements of the initial testing environment, which consisted of a workspace with 15 via points. In that context, each via point was represented using a 4-bit binary encoding, as 4 bits are sufficient to uniquely represent integers from 1 to 15. For other scenarios, this bit length may be adjusted accordingly to accommodate larger or more complex maps.

The algorithm maintained a fixed population size of 10 candidate paths per generation. The number of generations typically ranged from 50 to 100, depending on the complexity of the map and the desired level of solution quality. A high crossover probability of 90% was selected to maximize genetic mixing between individuals, while the mutation probability was kept low—between 0.001 and 0.006—to prevent premature convergence without introducing excessive randomness.

The start and end locations of each path were fixed during the genetic operations to preserve boundary conditions, with typical values being point 1 for the starting location and point 15 for the destination. Additionally, to improve early performance, a small number of known feasible paths were inserted into the initial population. This seeding strategy helped avoid situations where random initialization would fail to produce any valid solutions, especially in early generations where genetic diversity is still forming.

D. Reproduction

The reproduction process in this genetic algorithm consists of selection, crossover, and mutation operations, all of which are performed on binary-encoded paths. For the initial 15-node testing environment, each via point was represented using a 4-bit binary code, but this encoding length is subject to change depending on the number of nodes in future applications.

Selection is carried out using the roulette wheel method, which probabilistically favors individuals with higher fitness values. While this approach helps maintain diversity and allows for occasional selection of less fit individuals, it can also lead to premature convergence or stagnation, particularly in small populations. Crossover is implemented using a two-point strategy, where two random positions within the chromosome are selected and the corresponding segments between two parent strings are exchanged. Mutation is performed by randomly flipping bits within a defined region of the chromosome, excluding the segments reserved for the fixed start and end points.

In future iterations of the algorithm, alternative selection strategies such as tournament selection will be explored. These approaches offer improved selection pressure and may reduce the algorithm's dependence on global elitism to preserve high-performing solutions. By using a more controlled and competitive selection mechanism, tournament selection can enhance diversity while still guiding the population toward convergence.

E. Heuristic Mechanisms

In addition to standard genetic operators, the implementation incorporates two custom mechanisms designed to improve convergence behavior and overall solution quality: global elitism and diversity enforcement through population resets. These strategies are particularly important when working with limited population sizes or low mutation rates, where the risk of premature convergence is high.

Global elitism ensures that the most fit individual found across all generations is preserved throughout the evolutionary process. At the end of each generation, the algorithm compares the best solution from the current population with the best-known solution from all previous generations. If the current solution outperforms the stored global best, it replaces it; otherwise, the global best is reinserted into the population to guarantee it is not lost due to stochastic effects.

To complement this, the algorithm monitors population diversity by checking for minima lock, a condition where a significant portion of the population becomes identical or nearly identical in terms of fitness. If more than 80% of individuals have the same fitness score, indicating convergence to a local minimum, the population is reset. However, this reset is performed carefully: while most of the population is regenerated, the globally best solution is re-injected into the new population. This preserves valuable genetic information while restoring diversity and search capability.

Together, these two mechanisms maintain a balance between exploitation of good solutions and exploration of new areas in the solution space, leading to more reliable convergence behavior in repeated trials.

F. Design Variables

The design variables in this problem are defined by the structure of the workspace and the representation of viable paths through it. The environment is modeled as a 2D grid populated by a fixed number of via points, each assigned a unique identifier and corresponding (x, y) coordinate. These points form the basis of all candidate paths. In the initial testing configuration, the workspace consisted of 15 such nodes, each located at predefined positions on the map. A coordinate table is used internally by the algorithm to compute the Euclidean distances between points, which are essential for evaluating path fitness.

The start and end points are also defined in advance and remain fixed throughout the evolutionary process. For example, in the 15-node test case, the robot is required to begin at node 1 and reach node 15. These positions are enforced by constraining the first and last segments of each chromosome, preventing crossover and mutation from altering them. This structure ensures that all candidate solutions adhere to the navigation objective while allowing the algorithm to explore different intermediate paths.

G. Fitness Function

The fitness function is defined as the inverse of the path's total Euclidean distance, provided the path is feasible:

Eq 1. Fitness Function

$$f(x) = \begin{cases} \dfrac{1}{\sum_{i=1}^{m+1} d(P_i, P_{i+1})} : Feasible\ path \\ 0.001 : Infeasible\ Path \end{cases}$$

$$d(P_i, P_{i+1}) = \sqrt{(X_{i+1} - X_1)^2 + (Y_{i+1} - Y_1)^2}$$

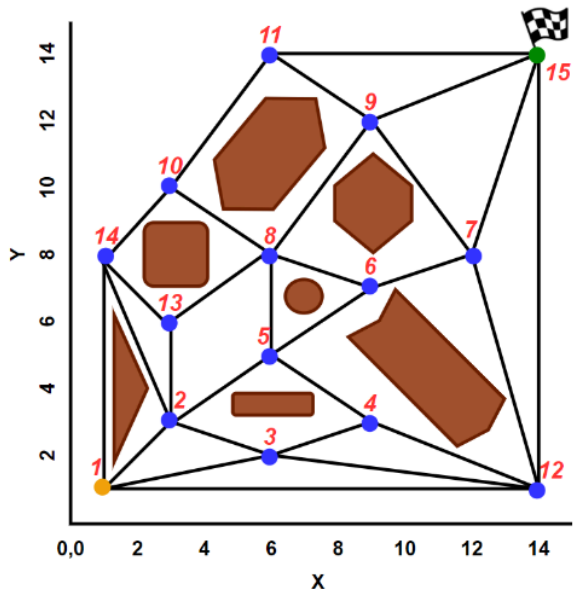
This is a maximization problem because the path with the fitness value is maximized for individuals with the shortest path. So, the path with a higher fitness value is more likely to be the shortest path.

V. How the Code Works

A. Program Flow

The code initializes a static map and generates an initial population with a mix of random and known valid paths. It then enters a loop that performs selection, crossover, mutation, and evaluation of each generation.

Fig 3. Map Generation



Each candidate's feasibility is checked by evaluating total distance travelled from the start position to the end position.

B. Operator implementation

In the MATLAB-based implementation of the genetic algorithm, each path is encoded as a binary string, where individual via points are represented by fixed-length binary segments. In the initial test environment using 15 nodes, each via point is represented using 4 bits, enabling binary encodings from 0001 to 1111. Full chromosomes are constructed by concatenating these 4-bit segments in sequence, forming a single binary string that encodes the entire path. This representation enables direct manipulation at the bit level for genetic operations.

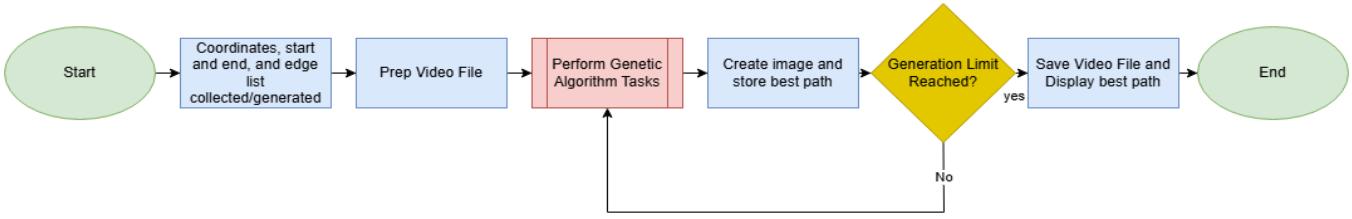
Selection is implemented using the roulette wheel method, where each individual in the population is assigned a selection probability proportional to its fitness score. A cumulative distribution is computed across the population, and a random number is drawn to determine which individual is selected. Since this method can disproportionately favor a few high-fitness candidates and lead to premature convergence, tournament selection is being explored for future versions of the algorithm. Tournament selection would allow greater control over selection pressure and reduce the algorithm's dependence on global elitism.

Crossover is performed using a two-point approach that swaps portions of two parent chromosomes. To protect the fixed start and end positions, the crossover points are restricted to the range between the second and second-to-last via point encodings. For example, in a 64-bit chromosome where the first and last 4-bit segments encode the fixed start and end nodes, crossover points are randomly selected within bit positions 5 to 60. This ensures that offspring maintain valid start and end locations while still allowing recombination of the intermediate path.

Mutation is implemented as a bit-flip operator, where a single bit in the mutable region of the chromosome is randomly chosen and inverted—changing a 0 to a 1 or vice versa. This operation is applied probabilistically based on a low mutation rate, typically between 0.001 and 0.006, to preserve diversity without introducing excessive randomness. The mutation only affects bits outside the start and end boundaries, maintaining path integrity while encouraging exploration of new solutions.

These operator implementations are written using low-level binary operations in MATLAB, enabling efficient chromosome manipulation and strict adherence to constraint logic. Together, selection, crossover, and mutation form the core evolutionary cycle that drives the genetic algorithm's progression toward optimal or near-optimal path solutions.

Fig 4. Program flow chart



VI. Large Map Experiment

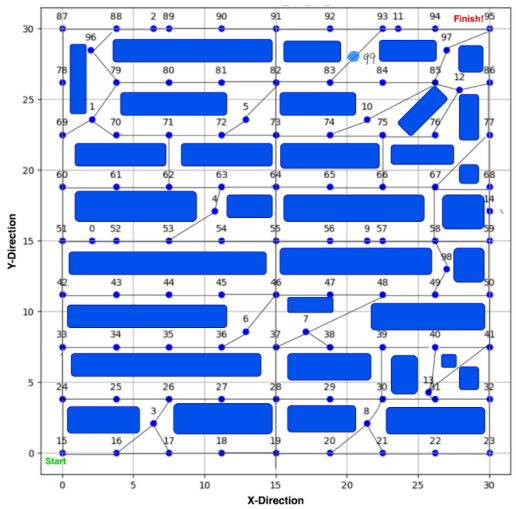
To further explore the algorithm's performance on large-scale environments, an experiment was conducted using a map with pseudorandomly generated nodes and obstacles. The layout of the map is shown in the accompanying figure. This test case was designed to simulate a more complex and realistic navigation scenario, significantly increasing the number of possible path combinations. However, the increased scale introduced severe computational bottlenecks.

Due to the large number of nodes and the inherent randomness of obstacle placement, the algorithm struggled to generate feasible paths, even when executed with a modest population size. The high density of the search space required an impractically large number of generations to yield any meaningful results. In one test, the algorithm was run for 25 generations, but the execution time was extremely slow—demonstrating that the current MATLAB implementation is not well-suited for handling such complexity.

Each generation became a computational burden, with the algorithm taking considerable time to process due to the exponential increase in chromosome length. In this setup, via points were encoded using 7-bit binary strings to accommodate a larger index space. This resulted in extremely long chromosomes and a vast search space with a correspondingly high number of infeasible combinations. Whenever the algorithm failed to identify a valid path, the fitness function returned a default low value (typically 1), indicating poor solution quality and minimal progress.

Overall, this experiment highlighted the practical limitations of scaling the genetic algorithm under the current implementation. The combination of long binary encodings, slow execution in MATLAB, and the algorithm's sensitivity to infeasible paths made it difficult to draw useful conclusions without excessive computational resources. This underscores the need for more efficient encodings, hardware acceleration, and optimized software platforms in future iterations of this work.

Fig 5. 100 node map



VII. Usage Potential for Large Scale Maps

To evaluate the scalability of the genetic algorithm, an experiment was conducted using a significantly larger test case involving a map with 100 via points. A visual representation of this environment is provided in the accompanying figure. Unlike the smaller 15-node scenarios, this expanded map revealed substantial limitations in both performance and feasibility assessment.

When executed in MATLAB on standard personal computers, the algorithm struggled to identify even a moderately feasible path within a small number of generations. A trial using 100 generations failed to produce a clearly optimal or even practically acceptable path. The resulting trajectories were not only suboptimal but often infeasible or inefficient, highlighting the algorithm's diminished effectiveness as problem complexity increases.

Moreover, the computational cost associated with running the algorithm in MATLAB was considerable. The simulation exhibited extremely slow performance, underscoring that MATLAB, while convenient for prototyping, is not a viable platform for real-time implementation or onboard vehicle deployment. It is plausible that running the algorithm for several thousand generations could yield a high-quality path; however, this would demand a level of computational power and time that far exceeds the resources available in this study.

As such, we are currently unable to demonstrate that the genetic algorithm approach, in its present form and environment, outperforms conventional or alternative methods. Further work involving optimized code, alternative programming environments (e.g., C++ or Python with GPU support), and more efficient selection strategies would be required to make a conclusive comparison.

VIII. Conclusion

The implementation of a Genetic Algorithm for mobile robot path planning successfully demonstrated its viability in small, structured environments with clearly defined nodes and obstacles. Through binary encoding, custom fitness evaluation, and heuristic mechanisms such as global elitism and population resets, the algorithm was able to generate collision-free paths while avoiding common pitfalls like premature convergence. However, the algorithm's performance declined significantly when applied to larger-scale maps, where high-dimensional chromosome

representations and resource-intensive MATLAB execution introduced severe computational challenges.

Tests with 25 to 100 generations in 100-node and pseudorandom environments failed to produce consistently feasible or efficient paths, and highlighted the impracticality of using MATLAB for real-time or onboard applications. The experiment revealed that while GAs are promising for offline or constrained environments, they require significant optimization—both in software implementation and algorithmic structure—to be competitive with or superior to traditional path planning techniques in large or dynamic scenarios. Future work should focus on refining encoding strategies, accelerating computation through optimized programming languages or hardware, and incorporating adaptive selection methods like tournament selection for improved convergence behavior.

References

[1] M. A. Azmyin, "A Path Planning Framework for Autonomous Surface Vehicles in Unknown Environments Using Genetic Algorithms," *ENGR 635 Final Report*, North Carolina A&T State University, Spring 2024.

[2] G. Nagib and W. Gharieb, "Path planning for a mobile robot using genetic algorithms," in *IEEE Proceedings of Robotics*, 2004, pp. 185–189.

[3] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, pp. 180–189, 2018.

[4] A. Bakdi, M. H. Rahmani, A. Chibani, and A. Mostefaoui, "Optimal path planning and execution for mobile robots using genetic algorithm and adaptive fuzzy-logic control," *Robotics and Autonomous Systems*, vol. 89, pp. 95–109, 2017.

[5] B. K. Patel, S. N. Sharma, and R. K. Patel, "A review: On path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, no. 4, pp. 582–606, 2019.