A Genetic Algorithm Approach to Path Planning in Cluttered 2D Environments

Krishna Kalavadia
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Canada
kkalavad@uwaterloo.ca

***Abstract*—Autonomous navigation through a complex environment is a difficult problem with significant challenges. Genetic algorithms have emerged as a valuable tool in path planning because of their effectiveness in generating high-quality solutions. A common implementation of the genetic algorithm for path planning applications involves discretization of the environment, which can limit the precision of the planner and flexibility of the solution. In this paper, I investigate and propose a continuous-space genetic algorithm that leverages variable-length chromosomes and lightweight genetic operators to find a collision-free path through a cluttered environment. Through various simulations of the proposed algorithm in a range of different environment complexities, I demonstrate the validity of the proposed GA as an approach to navigating cluttered continuous space environments.**

***Index Terms*—Path planning, genetic algorithms, autonomous systems**

I. Introduction

Autonomous mobile robotics are intelligent agents that can move and operate independently without the involvement of humans [1]. This allows mobile robots to be applied in various domains, such as transportation, medicine, agriculture, manufacturing, and military [2].

A core part of robotics is the task of path planning, which involves finding a route for the robot between some initial point and a goal [3]. This is a highly challenging problem as robots often have to navigate complex environments, choose between many possible paths, and adhere to constraints placed on the robot by its environment.

Path planning is a difficult problem to solve, and it requires the development of efficient and advanced planning algorithms. Over the years, significant research has been conducted into developing these algorithms, which can be traditionally split into search-based and sampling-based algorithms [4]. Search-based algorithms first discretize the environment and employ search algorithms such as A* [5] to find an optimal path. Search-based planners have the advantage of being optimal and complete [4], but they are computationally expensive in high dimensions and complex environments [6] [4]. Furthermore, because these methods first discretize the environment, this inherently comes with a loss of precision in representing the environment, which may not be desirable in highly cluttered and complex environments with narrow passages [7]. Sampling-based algorithms operate by randomly sampling points in the environment and connecting those points to form a graph; this approximates the environment's connectivity [4]. While this allows the planner to generate a fast solution, sampling-based planners generally provide sub-optimal solutions that are probabilistically complete, which means that a solution will be found given enough runtime and samples [4].

Over the years, more novel algorithms have been applied to the path planning problem, but they do not fit directly into the traditional categories described above. One such method that has gained significant interest is the genetic algorithm (GA) due to its ability to find high-quality, near-optimal solutions. The GA is an optimization algorithm inspired by the biological process of natural selection and survival of the fittest [8]. The general concept behind a GA is to evolve a given set of potential solutions toward optimal through genetic operators. First, the algorithm begins with some initial population P, which consists of a set of chromosomes. Each chromosome represents a candidate solution to the optimization problem at hand. We then evaluate our population according to a fitness function, which scores how good each chromosome is. Next, the algorithm selects two chromosomes from P, called "parents" and produces "child" chromosomes by crossover. The crossover operation, illustrated in Fig. 1, is the exchange of genetic material between two parents.

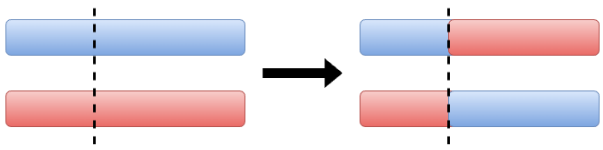

Fig. 1. An illustration of the single point crossover, the crossover point is indicated with the dashed line

The resulting child chromosomes then undergo mutation, where parts of their solutions are modified and altered. The child chromosomes are then placed in the next-generation population, and the process is repeated. The algorithm terminates based on some stopping criteria.

The effectiveness of a GA depends highly on how solutions are encoded as chromosomes, the fitness function and the implementation of genetic operators, namely selection, crossover and mutation.

A. Contributions

In this paper, I first explore existing implementations of GAs within the context of path planning for autonomous mobile robotics. Second, I propose an implementation of a GA that operates strictly in continuous space through tailored genetic operators. The proposed GA seeks to avoid discretization of the provided environment, which can result in a loss of precision. Third, I demonstrate the proposed GA is able to find high-quality solutions in cluttered 2-dimensional (2D) environments of varying complexities. Finally, I will discuss the results and outline limitations and future work.

B. Related Work

GAs have been extensively studied and applied to autonomous mobile robotics and path planning. While they have shown a promising ability to find high-quality feasible paths, most literature applies GAs to discrete environments [9] [10] [11] [12] [13] [14] [15]. In [9], the authors represent the environment in an occupancy grid, where each position (x, y) has the value of 0 for free cells and 1 for cells that are not or are occupied by obstacles. They then plan a path minimizing multiple objectives and leverage an improved crossover operator for better child chromosomes. In [10] and [11], instead of coordinates, each cell in the grid is denoted by a unique number. They then apply more specific genetic operators based on domain heuristic knowledge and leverage variable length and real-valued chromosome representation. In [10], the GA is adapted to handle static and dynamic environments, with successful results in both. In [12] and [13], the authors seek to improve grid-based planning by implementing a more comprehensive fitness function considering path smoothness and length. The authors of [13] also apply the algorithm on a real-world robot to demonstrate its practical application. In the fitness function proposed in [12], the smoothness contribution is adaptive and based on the angle of the line segment. In [14], the environment is first gridded and then internally represented as a weighted graph where each node is the centre point of each cell. The authors of [14] also propose an improved rank-based selection operator where the selection pressure is adjusted during runtime. This adaptive selection pressure is implemented to prevent premature convergence of the GA [14]. In [15], the environment is represented as a set of points and connections between points that the agent can move between. Unlike some of the other GAs explored, [15] leveraged a fixed-length, binary-encoded chromosome; this simplifies the genetic operators but limits the variability of solutions.

The authors of [16] and [17] improve upon the shortcomings of discretizing an environment and develop GAs that operate in continuous space. In [16], a hybrid algorithm is leveraged, first the environment is gridded and a deterministic algorithm finds feasible paths in discrete space. The result of this serves as an initial population for a GA. The GA then improves the initial paths to find an optimal path in continuous space by connecting points with a piecewise linear or a cubic spline [16]. The novel algorithm in [16] can solve multi-objective planning problems and is extended to handle multi-agent systems. In [17], the authors develop a GA that operates in continuous space that leverages domain-specific genetic operators and a unique obstacle detection mechanism. Instead of binary obstacle detection, [17] leverages a real-valued measure of the degree of the collision. The resulting GA is highly effective in finding paths in continuous space but with the trade-off of longer run times.

This paper seeks to address the same problem space as the authors of [16] and [17], in which I attempt to develop a GA that operates in continuous space. The proposed GA focuses on simplicity to create a lightweight GA capable of finding high-quality solutions in continuous space.

II. Problem Statement

In this section, I formulate my path planning problem. Consider an autonomous agent moving in a 2D map defined by $E \subseteq \mathbb{R}^2$. Within our environment, there is set of circular obstacles $O = \{o_1, \ldots, o_n\}$, a starting point s and an endpoint e. Any path that collides with any of the obstacles in O is infeasible. Fig 2. shows a visualization of the environment and its components. The objective is to plan a collision-free path from s to e while minimizing path length.

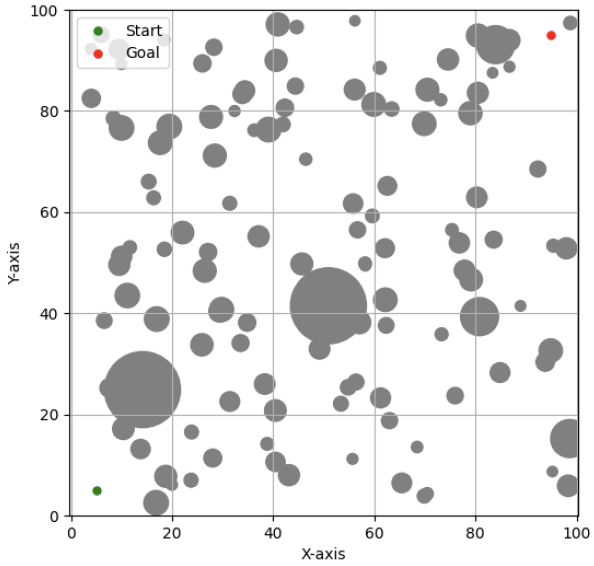

Fig. 2. A sample environment is shown with the start position located at $(5, 5)$ and the end position at $(95, 95)$

III. Approach

In this section, I present the proposed GA that operates strictly in continuous space to find a feasible path from the start point to the endpoint. I describe the implementation of each component of the GA, and the final algorithm integrating all components is presented at the end.

A. Genetic Representation

Genetic representation refers to how a potential solution is encoded in each chromosome. The encoding scheme heavily depends on the problem space and the nature of the information that represents a solution [8]. The most common encoding used is binary encoding, where the chromosome is represented as a string of 1 or 0, but other representations, such as valued-based and tree-based representations, can also be used [8]. Another attribute of genetic representation is whether chromosomes have a fixed length or can be variable.

The proposed algorithm leverages a real-valued, variable-length encoding scheme. Each chromosome is represented as a sequence of (x, y) coordinates with a specified minimum and maximum length.

B. Initial Population

In order to generate the initial population, the algorithm randomly selects a number n between the minimum and maximum chromosome length. Then, a path is created between $n-2$ randomly sampled waypoints, w_i, in free space with s and e attached at the start and end points, respectively. The resulting chromosome takes the structure $\{s, w_1, \ldots, w_{n-2}, e\}$. This process is repeated to create an initial population size of a desired size. It's important to note that it is not required for the population of initial paths to be feasible. Fig 3. below shows an initial path population of five paths plotted over a sample environment. In the proposed algorithm, the population size remains constant across generations. With a sufficiently large population size, higher genetic diversity is maintained, which increases the chance of the GA to discover high-quality solutions. The trade-off with a larger population is that it increases the computational overhead. In the proposed GA, multiple population sizes were experimented with, but a final population size of 500 was selected to balance genetic diversity and computational overhead.

C. Fitness Function

The fitness function of a GA is used to evaluate how well the chromosomes are performing according to the objective of the problem. This is highly specific to the problem domain. The objective of the proposed problem domain is to find a collision-free path from s to e. Therefore, a given chromosome's fitness is evaluated c as follows:

$$F(c) = \sum_{i=1}^{n} dist(w_i, w_{i+1}) + (\rho \cdot (\text{collision count})) \quad (1)$$

Where $dist(w_i, w_{i+1})$ is the distance between two waypoints in our chromosome, and ρ is the collision penalty. Any path with a fitness greater or equal to ρ is determined to be infeasible as the GA is looking to solve for collision-free paths.

1) Collision Detection: A methodology for counting collisions is required. Collision detection in continuous space is more computationally intensive than in discrete space, specifically gridded environments. In a gridded environment, an obstacle-filled cell can simply be marked with a '1', allowing for easier collision checking. In the proposed algorithm, recall that a chromosome is represented by a sequence of waypoints. By definition, all waypoints are sampled in free space, but each of the segments that connect waypoints need to be checked for collisions.

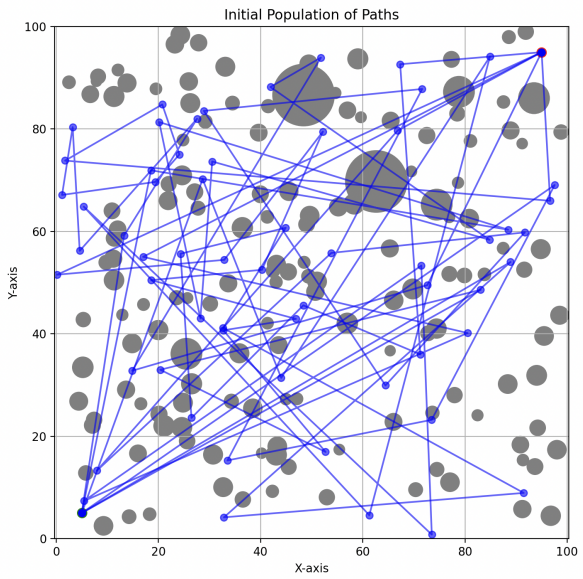

Fig. 3. An initial population of five paths plotted over a sample environment

a) Collision Detection Methods: To check if a given segment contains a collision, vector projections are leveraged. An illustration of the implemented collision detection can be found in Fig 4. Consider a line segment defined by the points A and B and a nearby circular obstacle with the center point C and radius r. To determine if the line segment between A and B intersects the circle, the vector between points A and C is projected onto the vector between points A and B. This projection enables the construction of a point D, representing the point on the line segment closest to the obstacle. A collision is detected if the distance between D and C is less than or equal to r.

b) Spatial Decomposition: Although the approach described above effectively identifies collisions, running collision checks against every environmental obstacle is computationally inefficient. For instance, a line segment located in one corner of the environment does not require collision checks with obstacles in the opposite corner. To address this inefficiency, spatial decomposition techniques are leveraged, which have been widely applied in collision detection [18]. In spatial decomposition, the environment is broken into smaller units of space called cells, and obstacles are mapped to these cells [18]. When collision tests need to be conducted on some object of interest, only the obstacles in the same cell or nearby cells must be checked instead of the entire environment [18]. The proposed algorithm breaks the environment down uniformly into a grid with a specified cell size. Then, a hash map is created where each cell is mapped to a list of obstacles that occupy or overlap with the cell. Therefore, when a collision

check is run on a line segment, based on the location of the line segment, only the relevant obstacles are checked.

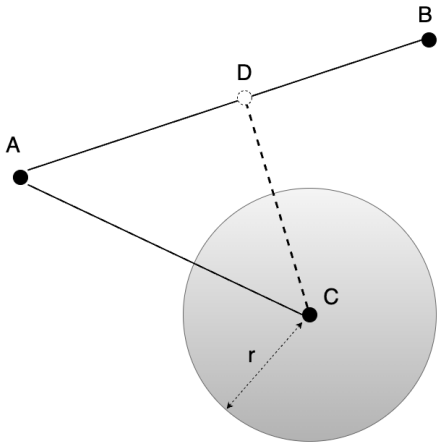

Fig. 4. Collision detection for line segment AB with an obstacle defined by centre C and radius r

D. Selection

The Selection operation of a GA determines which chromosomes of our current population will be a part of our reproductive step to create a new population [8]. The main types of selection techniques include roulette wheel, rank, tournament, Boltzmann, and stochastic universal sampling, each with its own set of advantages and disadvantages [8]. In the proposed algorithm, rank-based selection is leveraged as it is relatively simple, reduces the chances of premature convergence and preserves genetic diversity [8]. The trade-off with rank-based selection is that it requires sorting of the population, which can be computationally intensive [8].

In rank-based selection, the population is first sorted based on their fitness values and each chromosome is given a rank based on this ordering [8]. Chromosomes with lower fitness values correspond to better performance and are given higher ranks. Based on this ranking, a probability is assigned to each chromosome proportional to its rank [8]. The selection probability in the proposed algorithm is calculated according to (1):

$$P(r) = \frac{r}{\sum_{i=1}^{N} i} = \frac{r}{\frac{N(N+1)}{2}},$$

Where r is the rank of the chromosome in the population, N is the population size, and $\sum_{i=1}^{N} i$ is the sum of all ranks. Each chromosome is selected for reproduction based on this probability.

To enhance the selection process, rank-based selection was also combined with elitism. As a GA evolves, there is a risk that the best-performing chromosomes in the population are lost as they undergo cross-over and mutation. [8] This behaviour can be corrected by implementing elitism, which directly adds the best-performing individuals to the next population without modification, preserving their genetic material [8]. In the proposed algorithm, the chromosomes are sorted based on their fitness values before the population undergoes selection. The top performing chromosomes, referred to as "elites," are then taken from the population and directly appended to the next generation to preserve their paths.

E. Crossover

The crossover is an operator that will combine the genetic information of two parent chromosomes to produce children [8]. The main types of crossover operators, summarized in [8], include single-point, two-point, k-point, uniform, partially matched, etc. The simplest form is single-point crossover, illustrated in Fig 1. In single-point crossover, a cross-over point is randomly selected for the two parents. The genetic information of the two parents beyond that crossover point will then be swapped to produce two child chromosomes. While there are more sophisticated crossover operators that generally perform better, single-point cross-over is the simplest [8] and more computationally efficient. As I'm seeking to develop a lightweight GA, I have selected single-point crossover, which has also been used by [12], and [14]. To implement single-point crossover with the variable chromosome structure of the proposed GA, the crossover point is selected based on the length of the shorter parent.

F. Mutation

Mutation operators seek to produce genetic diversity in the population, allowing the GA to effectively explore the solution space [8]. After offspring are generated from crossover, they undergo a mutation process. The proposed algorithm applies three main types of mutation operators tailored for the path planning problem in continuous space: perturb waypoints, add waypoints and delete waypoints. A given chromosome goes through each mutation operator with a specified mutation rate. The mutation constant is a number between 0 and 1. The proposed algorithm randomly samples a number between 0 and 1 to determine if a mutation is to occur. If the sample is less than the mutation rate, a mutation is carried out on the chromosome. These mutations enable the GA to create new solutions that may result in a more optimal path from start to end.

1) Perturb Waypoint: The first mutation operator applied is the perturbation of waypoints within chromosomes. For a given chromosome, the GA iterates through every waypoint within the chromosome and mutates each per the mutation rate mechanism above. To perturb a waypoint, a new location within some specified radius around the original waypoint is randomly selected. If the new candidate location does not fall within any obstacle and is within the defined environment bounds, then it is accepted. Otherwise, the process is retried. To prevent the GA from spending excessive time in the retry operation, the number of tries is specified with an upper bound, after which the mutation is skipped. This mutation aims to move waypoints that may be forcing a path through some obstacle, as illustrated in Fig 5.

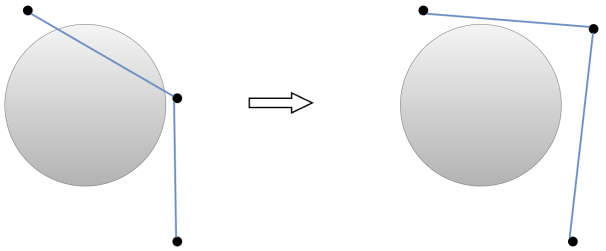

Fig. 5. An example showing the intended effects of perturbing waypoints

2) Add Waypoint: The next operator adds a waypoint to a chromosome. First, an index i within the chromosome is selected. Next, a new waypoint is created initially as the midpoint between index i and index $i+1$. This new waypoint is then perturbed similarly to the perturbation operator and checked for validity, i.e., it does not lie in an obstacle or outside the environment's bounds. This also uses the retry mechanism described in the perturbation operator. If the waypoint is valid, it is inserted in the chromosome between the points at index i and index $i+1$. This mutation aims to bend paths around obstacles, as illustrated in Fig 6.

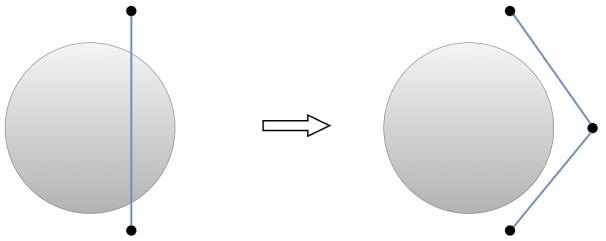

Fig. 6. An example showing the intended effects of adding waypoints

3) Delete Waypoint: The final operator deletes a waypoint in a chromosome. An index i within the chromosome is selected and removed. This deletion has multiple purposes. It can remove unnecessary waypoints from paths that are making the path erratic and jagged; therefore, their removal will result in a more direct path. The other purpose is to remove waypoints forcing paths to cross over obstacles, resulting in a collision. An illustration of the latter case is shown in Fig 7.

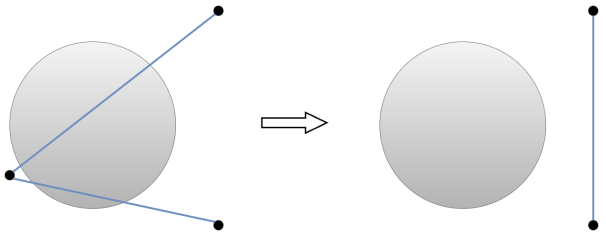

Fig. 7. An example showing the intended effects of removing waypoints

G. Proposed Algorithm

In this section, I integrate the components described above and present the proposed GA algorithm. The proposed GA takes in a start location, end location, an obstacle spatial grid, and environment bounds. The output is the best-performing path from start to end. First, an initial population is generated, which is a set of random paths from start to end. While all waypoints are guaranteed to be located in free space, the feasibility of the path is not, and paths can contain collisions. The algorithm then repeats the following steps. First, the population is sorted by fitness values, and the elites are excluded. Next, a set of parents is selected using rank-based selection, and cross-over is performed to produce child chromosomes. The child chromosomes then undergo mutation, diversifying the paths to explore the search space. After mutation, the child chromosomes are combined with the elites to form the next generation. Finally, the finesses of the next generation are evaluated, and the process is repeated until convergence.

The convergence criteria of a GA determines when the algorithm should terminate. The proposed algorithm relies on two convergence conditions. The first is a specified upper bound on the number of iterations. The second criterion is a threshold for the improvement of solutions over successive generations. The best-performing chromosome at each generation is tracked. If the best-performing fitness values are not sufficiently improving over some specified number of generations, known as the convergence window, the algorithm is terminated. While the convergence is upper bounded by some specified number of generations, in practice, it is the second criterion on fitness improvements, which terminates the algorithm.

Algorithm 1 Proposed GA

Input: Start position s, end position e, obstacle grid O, environment bounds B
Output: Path from s to e

Initialize population P
Compute initial population fitness $F \leftarrow \text{fitness}(P)$
for $i = 1$ to `max generations` **do**
    Select elites E from P
    Select parents $P_{parents}$ from P
    Initialize offspring $P_{offspring}$
    while size($P_{offspring}$) $<$ size(P) - size(E) **do**
        Select two parents (p_1, p_2) from $P_{parents}$
        Perform crossover: $(c_1, c_2) \leftarrow \text{crossover}(p_1, p_2)$
        Perform mutation: $(c_1, c_2) \leftarrow \text{mutate}(c_1, c_2)$
        Add c_1 and c_2 to $P_{offspring}$
    end while
    Update population $P \leftarrow E + P_{offspring}$
    Evaluate fitness $F \leftarrow \text{fitness}(P)$
    if Convergence met (finesses not improving) **then**
        Break
    end if
end for
Return best chromosome from P

IV. Results

In this section, I present the results of the proposed algorithm when navigating various environments to demonstrate its viability and performance. The test environments are composed of circular obstacles of random locations and sizes. The proposed GA is configured with the parameters specified in Table 1.

TABLE I
GA Parameters

Main GA Parameters	Value
Population Size	500 Chromosomes
Minimum Chromosome Size	10 Waypoints
Maximum Chromosome Size	100 Waypoints
Maximum Number of Generations	500
Collision Penalty	10,000
Mutation Rate	0.4
Convergence Window	25 Generations
Convergence Tolerance	1.0
Waypoint Perturbation Radius	2.5
Number of Parents	50% of Population Size
Number of Elites	1% of Population Size

To demonstrate the GA's performance, four environments of size 100 x 100 are constructed and filled with 50, 100, 150, and 200 obstacles, respectively. The GA navigates each environment 10 times, and the runtime, path cost, and convergence generation are recorded. The averaged results of the four environments are summarized in Table II. Fig 8, Fig 9, Fig 10, and Fig 11 plot examples of paths found by the GA within each respective environment.

TABLE II
Proposed GA results on various environments

Environment	Runtime	Path Cost	Convergence Generation
50 Obstacles	8.7s	128.8	71.7
100 Obstacles	11.8s	137.7	75.2
150 Obstacles	16.2s	142.3	82.4
200 Obstacles	13.9s	1154	85.1

The results that are summarized in Table II and shown in Fig 8 - 11. illustrate that the proposed GA can find high-quality solutions in cluttered environments. In sparse environments such as in Fig 8. the proposed GA quickly finds an optimal path consistently in less than 9 seconds. The runtime and convergence generation increase as the obstacle field becomes more dense, but high-quality solutions are still consistently found. In a cluttered obstacle field, the solution path tightly corners obstacles to keep the path as direct as possible and finds small gaps in between obstacles that could be lost by discretization. This is a significant result and validates the application of GAs in continuous space, specifically in complex environments that cannot be effectively discretized.

These results also verify the success of the implemented mutation operators as they enable the GA to evolve paths around complex obstacle fields. In areas of high obstacle density, the GA adds more waypoints to bend around various

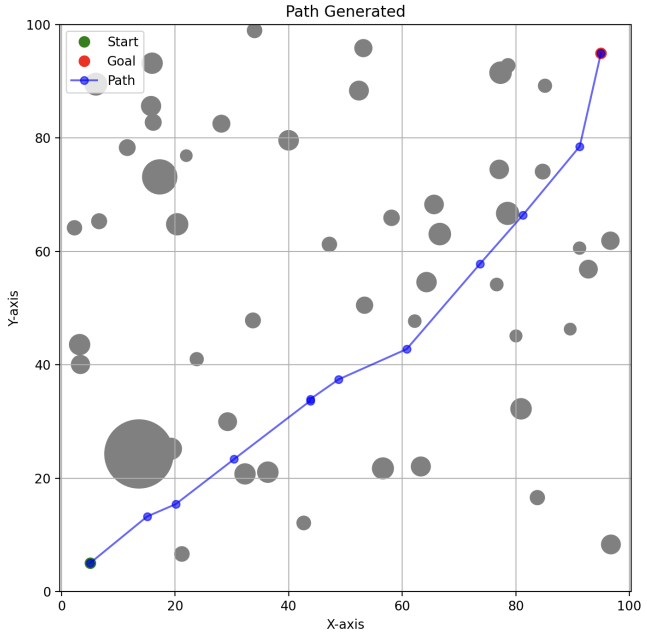

Fig. 8. Proposed GA result on an environment with 50 obstacles

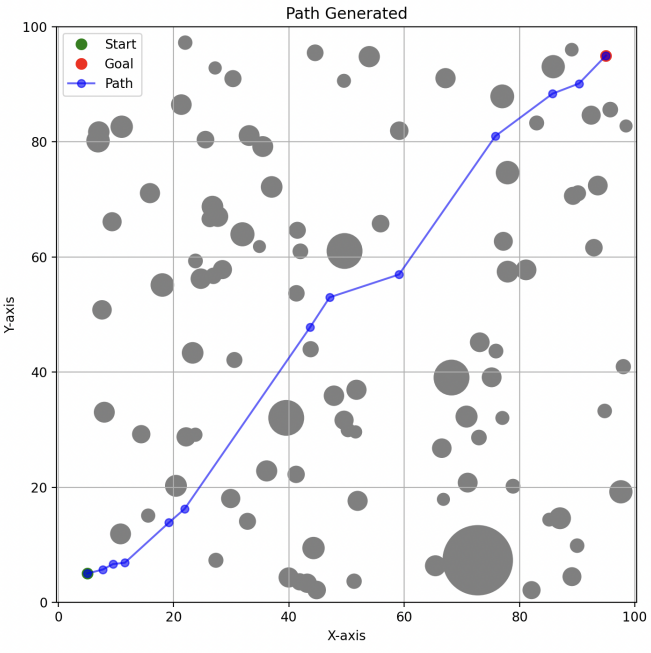

Fig. 9. Proposed GA result on an environment with 100 obstacles

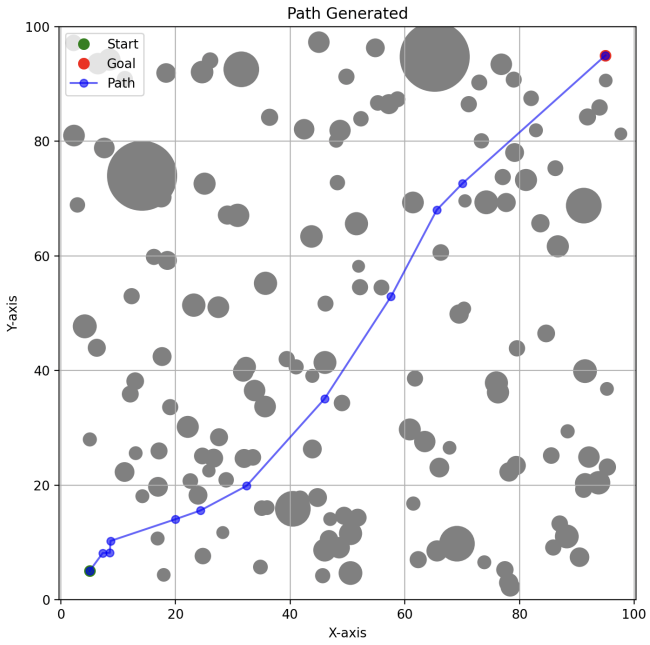

Fig. 10. Proposed GA result on an environment with obstacles

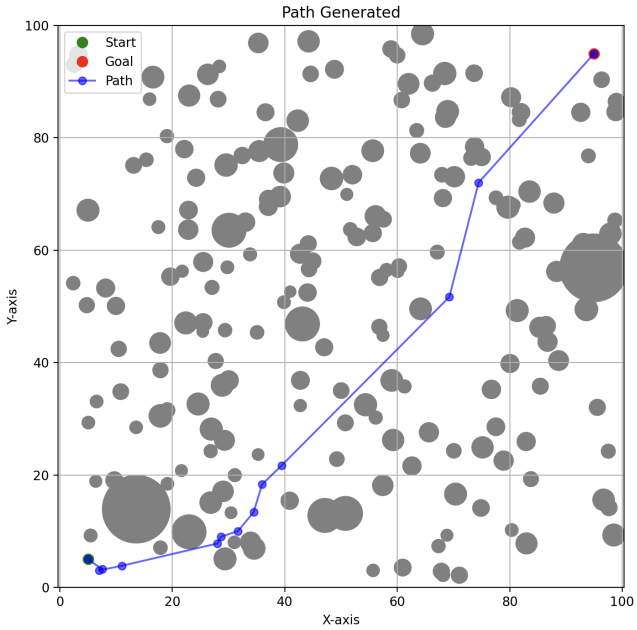

Fig. 11. Proposed GA result on an environment with 200 obstacles

obstacles, while in sparse areas, waypoints are removed, which results in a direct path without unnecessary turns. This behaviour is clearly illustrated in Fig 11. where the majority of the waypoints are located in the bottom left quadrant, where the GA has to navigate a particularly obstacle-dense area. It is important to note that while waypoint placement generally follows this trend, the stochastic nature of the GA can still output path segments with unnecessary waypoints, which is seen at the start of the solutions in Fig 9 and Fig 10.

Although the proposed GA successfully navigates cluttered environments, Table II shows that it sometimes cannot find a feasible solution. In environments with 200 obstacles, the average path length is larger than the map dimensions, which can be attributed to a path containing a collision and being penalized accordingly.

V. Discussion

The results outlined above illustrate that the proposed GA can successfully navigate environments of varying complexities. This demonstrates that a GA is a valid approach to solving continuous space planning problems, specifically those in complex environments requiring precise navigation. Below, I address the algorithm's main advantages and limitations compared to related works in the literature surrounding GA-based path planning.

1) Advantages: The main advantage of the proposed GA is that it operates strictly in continuous space. Discretizing an environment comes with a loss of precision, which becomes particularly important in highly complex and cluttered environments where narrow passages exist [7]. A continuous-space GA can find these narrow passages, which may result in more optimal paths than a discrete-space GA, which is more common in literature. The second advantage of the proposed algorithm is its dynamic waypoint spacing. Certain planning algorithms, particularly those that operate on uniform grids, utilize equally space waypoints. In contrast, the proposed GA allows for dynamic waypoint spacing, which allows the path's complexity to be adaptive to the environment and its demands for precise vs course navigation.

2) Disadvantages: One evident limitation of the algorithm is the simplicity of the mutation and crossover operators. Waypoints are mutated randomly without consideration of any information or heuristics of the environment. While this has little impact in simple environments, in more complex cases such as the 200-obstacle environment, the proposed GA can fail to find a feasible path. Another limitation is the algorithm's computational overhead, resulting in runtimes exceeding 10 seconds in more complex environments. Being a continuous-space planner largely guided by randomization, it takes a significant number of iterations and computational power to converge onto a quality solution. Depending on the specific application of the GA, this can be undesirable and hinder the algorithm's success.

A. Future Work

Future work should consider improving the genetic operators to be targeted/guided rather than relying on randomiza-

tion. Leveraging knowledge-based operators proposed by other works in literature, such as [10] and [11], can improve the evolution process to search for optimal paths more efficiently. Instead of relying largely on randomization to explore the search space, adding heuristics and more domain knowledge could improve the proposed GA significantly. Another avenue of work can focus on parallelization, as the GA is an inherently parallel algorithm that operates on a population of solutions. For instance, the proposed GA selects two parents, performs crossover, and mutates the resulting children. This cycle is repeated until a sufficient number of offspring are generated; parallelizing these repetitive operations can significantly enhance runtime efficiency. Lastly, more comprehensive parameter tuning should be carried out to determine the best set of parameters. The GA's performance can be significantly impacted by specific parameters such as mutation rates, population sizes, etc, and therefore, finding the optimal parameter set is critical.

REFERENCES

[1] L. Liu, X. Wang, X. Yang, H. Liu, J. Li, and P. Wang, "Path planning techniques for mobile robots: Review and prospect," Expert Systems with Applications, vol. 227, Oct. 2023, doi: 10.1016/j.eswa.2023.120254.

[2] B. K. Patle, G. Babu L, A. Pandey, D. R. K. Parhi, and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot," Defence Technology, vol. 15, no. 4, pp. 582–606, Aug. 2019, doi: 10.1016/j.dt.2019.04.011.

[3] J. R. Sánchez-Ibáñez, C. J. Pérez-del-Pulgar, and A. García-Cerezo, "Path Planning for Autonomous Mobile Robots: A Review," Sensors, vol. 21, no. 23, Nov. 2021, doi: 10.3390/s21237898.

[4] G. Sotirchos and Z. Ajanovic, "Search-based versus Sampling-based Robot Motion Planning: A Comparative Study," Jun. 17, 2024, arXiv: arXiv:2406.09623. doi: 10.48550/arXiv.2406.09623.

[5] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.

[6] M. Elbanhawi and M. Simic, "Sampling-Based Robot Motion Planning: A Review," IEEE Access, vol. 2, pp. 56–77, 2014, doi: 10.1109/ACCESS.2014.2302442.

[7] W. Du, F. Islam, and M. Likhachev, "Multi-Resolution A*," SOCS, vol. 11, no. 1, pp. 29–37, Sep. 2021, doi: 10.1609/socs.v11i1.18532.

[8] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," Multimedia Tools and Applications, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.

[9] 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, doi: 10.1016/j.procs.2018.01.113.

[10] Y. Hu and S. X. Yang, "A Novel Knowledge-Based Genetic Algorithm for Robot Path Planning in Complex Environments," Sep. 03, 2022, arXiv: arXiv:2209.01482. doi: 10.48550/arXiv.2209.01482.

[11] S. C. Yun, V. Ganapathy, and L. O. Chong, "Improved genetic algorithms based optimum path planning for mobile robot," in 2010 11th International Conference on Control Automation Robotics & Vision, Dec. 2010, pp. 1565–1570. doi: 10.1109/ICARCV.2010.5707781.

[12] Y. Li, Z. Huang, and Y. Xie, "Path planning of mobile robot based on improved genetic algorithm," in 2020 3rd International Conference on Electron Device and Mechanical Engineering (ICEDME), Suzhou, China: IEEE, May 2020, pp. 691–695. doi: 10.1109/ICEDME50972.2020.00163.

[13] J. Ni, K. Wang, H. Huang, L. Wu, and C. Luo, "Robot path planning based on an improved genetic algorithm with variable length chromosome," in 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Aug. 2016, pp. 145–149. doi: 10.1109/FSKD.2016.7603165

[14] A. H. Karami and M. Hasanzadeh, "An adaptive genetic algorithm for robot motion planning in 2D complex environments," Computers & Electrical Engineering, vol. 43, pp. 317–329, Apr. 2015, doi: 10.1016/j.compeleceng.2014.12.014.

[15] G. Nagib and W. Gharieb, "Path planning for a mobile robot using genetic algorithms," in International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04., Cairo, Egypt: IEEE, 2004, pp. 185–189. doi: 10.1109/ICEEC.2004.1374415.

[16] M. Nazarahari, E. Khanmirza, and S. Doostie, "Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm," Expert Systems with Applications, vol. 115, pp. 106–120, Jan. 2019, doi: 10.1016/j.eswa.2018.08.008.

[17] V. R. Ragusa, H. D. Mathias, V. A. Kazakova, and A. S. Wu, "Enhanced genetic path planning for autonomous flight," in Proceedings of the Genetic and Evolutionary Computation Conference, in GECCO '17. New York, NY, USA: Association for Computing Machinery, Jul. 2017, pp. 1208–1215. doi: 10.1145/3071178.3071293.

[18] W. Zhao and H. Qu, "Collision Detection Technique Based on the Spatial Decomposition Method," 2023 International Seminar on Computer Science and Engineering Technology (SCSET), New York, NY, USA, 2023, pp. 385-388, doi: 10.1109/SCSET58950.2023.00090.