MetaPaths: Utilizing meta-heuristic approaches for optimal path planning

Saliq Gowhar-211AI043
Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
saliqgowhar.211ee250@nitk.edu.in

Bhavya-211AI011
Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
shrmabhav.211ai011@nitk.edu.in

Ashutosh Kumar - 211AI008
Information Technology
National Institute of Technology Karnataka
Surathkal, India 575025
ashutosh.211ai008@nitk.edu.in

***Abstract*—Mobile robots have gained significant traction in a wide range of environments, including automation and remote monitoring applications. To ensure seamless task execution, the ability to navigate between points while efficiently avoiding obstacles within the environment is of paramount importance. In this study, we aim to delve into the exploration of various meta-heuristic algorithms, such as Genetic Algorithms, Particle Swarm Optimization, A* algorithm, and Conflict-Based Search, to facilitate effective, optimal, and collision-free path planning in both static and dynamic obstacle-rich environments. Our investigation encompasses both single-agent and multiple-agent path planning scenarios. By analyzing the workings and advantages of these diverse approaches, we have identified Genetic Algorithms, Particle Swarm Optimization, and A* algorithm as suitable choices for single agent path planning. For multi-agent path planning, we have leveraged the Conflict-Based Search (CBS) algorithm to address coordination and conflict resolution challenges. Through comprehensive experimentation and evaluation, we assess the performance of the selected algorithms in terms of efficiency, optimality, adaptability to real-time constraints, and successful obstacle avoidance. The outcomes of this study provide valuable insights into the efficacy of these meta-heuristic algorithms for path planning in obstacle-rich environments.**

Keywords: Genetic algorithms, Particle Swarm Optimisations (PSO), A-star algorithm, Conflict-based search (CBS), Multi-robot path planning, Single agent path planning, meta-heuristic algorithms

I. Introduction

The optimal path planning problem, which entails determining the most suitable route for a robot or autonomous agent to traverse from an initial location to a destination while avoiding obstacles in the surrounding environment, is a critical challenge with far-reaching implications. This problem holds substantial relevance in diverse domains and practical contexts, spanning robotics, autonomous systems, safety-critical applications, efficiency optimization, real-time adaptability, a collaboration between humans and robots, as well as urban planning and infrastructure design [5]. The outcomes of the research addressing the optimal path planning problem have the potential to drive technological advancements in multiple fields. They can enhance safety measures, improve resource utilization, enable adaptive decision-making in dynamic environments, facilitate harmonious interactions between humans and robots, and optimize urban functionality for enhanced livability.

Path planning in real-life scenarios is challenging because it involves navigating through complex and dynamic environments, managing uncertainty and incomplete information, meeting real-time constraints, ensuring safety while optimizing efficiency, and addressing scalability issues [1]. To tackle these challenges, it is necessary to develop advanced algorithms, employ environment perception techniques, and establish efficient decision-making processes. Existing solutions for path planning in obstacle-rich environments include grid-based methods, potential field methods, sampling-based methods, evolutionary algorithms, and machine learning-based methods. Grid-based methods divide the environment into a grid and use algorithms like A* or Dijkstra's algorithm. Potential field methods guide the robot using artificial potential fields. Sampling-based methods randomly sample the configuration space and create a graph. Evolutionary algorithms employ genetic algorithms or particle swarm optimization [7]. Machine learning-based methods use techniques like deep reinforcement learning or imitation learning. The choice of method depends on the specific environment and the robot's capabilities.

In this project, our primary objective is to implement multiple meta-heuristic approaches, namely Genetic Algorithms, Particle Swarm Optimization (PSO) [4], and the A* (A-star) algorithm [2], to address the challenges of both single-point and multiple agent path planning in obstacle-rich environments. Our aim is to analyze the workings and advantages of these meta-heuristic algorithms in the context of our specific problem. By evaluating their performance and suitability, we intend to determine the most effective approach for our desired path-planning problem. Furthermore, for multi-agent path planning, we plan to implement Conflict-Based Search (CBS)

[9], a specialized algorithm that focuses on resolving conflicts and achieving efficient coordination among multiple agents in dynamic environments. This addition will allow us to explore the feasibility and effectiveness of CBS in solving the multi-agent path planning problem. Through our implementation and analysis of these various meta-heuristic approaches, we aim to gain insights into their strengths, limitations, and potential applications for solving path-planning problems in obstacle-rich environments.

The rest of the article is structured as follows: The ”Literature Survey” presents information regarding existing research and the methodologies used, The ”Problem Statement” clearly explains our motives and objectives for the project, The ”Methodology” section which explains our methodology and implementation in detail, The ”Experimental results and analysis” which contain our insights gained and results obtained along with complexity analysis, The ”Conclusion and Acknowledgment” and finally the ”Individual Contributions”.

II. Literature Survey

Daniel Foead et al.[1] provides a comprehensive analysis of the A-star algorithm in a wide range of discrete domains, problem scenarios, and application areas. It evaluates the overall performance and efficacy of A-star variations in terms of their ability to find optimal paths across diverse environments. The research aims to shed light on the strengths and weaknesses of A-star algorithms, offering insights into their suitability for different real-world applications requiring efficient path finding solutions. Notably, A-star algorithms exhibit high efficiency, surpassing other methods in terms of their intelligent search strategy. They strike a favorable balance between finding optimal paths and computational efficiency, making them widely preferred in various path-finding applications. However, A-star algorithms may not always guarantee the absolute optimal path or may experience slower convergence in certain scenarios. Moreover, they may require significant memory resources when dealing with graphs containing numerous nodes or multiple concurrent path explorations, potentially impacting performance and scalability.

Lisang Lui et al.[2] propose an optimized A-star algorithm that incorporates rule modifications aimed at enhancing its efficiency in path planning. When encountering obstacles during the search process, the algorithm adjusts the cost function of obstacle nodes from an open value to infinity. This adjustment effectively reduces data redundancy and computational requirements, resulting in a substantial reduction in path planning time. By swiftly identifying and disregarding obstacles in the search process, the optimized A-star algorithm improves computational efficiency and expedites the overall pathfinding operation. The merits of this optimization approach include a reduction in data redundancy as the cost function of obstacle nodes is set to infinity, thereby streamlining the search process and improving computational efficiency. Additionally, the optimized A-star algorithm significantly reduces path planning time by efficiently handling obstacles and focusing on finding the most optimal path. This speed enhancement proves beneficial in time-sensitive applications or dynamic environments. However, there are certain demerits to consider. Assigning an infinite cost to obstacle nodes may result in the algorithm overlooking potentially suboptimal but feasible paths. This trade-off between optimality and speed may cause the algorithm to choose longer or less efficient routes simply to avoid obstacles. Furthermore, the modification assumes a static environment, and the algorithm's performance may suffer if obstacles change dynamically or are inaccurately represented. Outdated or missing obstacle information can lead to inefficient or invalid paths, highlighting the algorithm's lack of adaptability in such scenarios.

David et al.[3] assess the effectiveness of the Particle Swarm Optimization (PSO) algorithm in finding solutions and pathfinding in environments obstructed by obstacles. Successful implementation of the PSO algorithm in such scenarios can have implications for maze games and robot motion planning. The study focuses on investigating the influence of PSO parameters, including the number of particles, weight constant, particle constant, and global constant, on the algorithm's performance in generating solution paths. Adjusting these parameters can lead to faster movement of the swarm towards the target point, but it may also result in longer convergence time due to excessive random movements. Conversely, reducing the PSO parameters slows down the swarm's movement but promotes quicker convergence by minimizing random movements. Through extensive simulations with various parameter settings, the study demonstrates that the PSO algorithm has the capability to generate viable path solutions in obstacle-rich environments. The methodology combines global and local search methods, utilizing the particles' positions and fitness evaluation to guide their movement toward optimal solutions. The results indicate that selecting an appropriate particle population size is crucial for balancing convergence speed and obstacle complexity. The advantages of the PSO algorithm include its global and local search capabilities, allowing it to converge to near-optimal solutions efficiently. However, challenges such as premature convergence and limited robustness to objective function noise are also acknowledged.

Nasrollahy et al.[4] introduce a novel approach for path planning in dynamic environments with both moving and static obstacles using Particle Swarm Optimization (PSO). The objective is to minimize the total path planning time while avoiding local optima. The proposed approach considers a scenario where the goal position is also moving over time, and several assumptions and principles are established. The PSO algorithm is employed, with particles representing potential solutions for the robot's next position. A penalty function is incorporated to account for obstacles' presence, size, and position, aiming to find the shortest collision-free path to the goal. The newly proposed penalty function effectively handles dynamic environments, preventing the algorithm from getting stuck in local optima traps. However, the paper lacks extensive experimentation and comparative analysis to validate the approach's effectiveness and efficiency. Furthermore, the limitations and challenges of implementing the proposed approach in real-world robotic systems are not discussed. As a result, further evaluation and analysis are necessary to establish the practical applicability of the approach in real-life scenarios.

Z. Chen et al.[5] focus on addressing the issue of premature convergence in the basic genetic algorithm for robot path planning. To overcome this limitation, the authors propose several improvements and optimizations. First, they utilize different population initialization methods to enhance population diversity. This helps to explore a wider range of solutions. Next, they enhance the adaptive and elite strategies of crossover and mutation operators to improve the convergence speed of the algorithm. Additionally, they introduce the concept of path tortuosity into the fitness function to encourage smoother planned paths. Obstacle avoidance is incorporated by adding constraints to the model. Furthermore, the authors transform the coding paradigm of the improved genetic algorithm, enabling it to run efficiently on a distributed cluster using Flink technology. This allows for faster solution speeds, catering to the efficiency requirements of path planning in large-scale robot cluster systems. The optimized algorithm is evaluated against the basic genetic algorithm through simulations, demonstrating its effectiveness in robot path planning. The disadvantage of the paper is that it lacks a comprehensive analysis and comparison with other existing algorithms. While the improvements and optimizations to the basic genetic algorithm are described, there is limited discussion on how the proposed approach performs in comparison to other methods in terms of convergence speed, solution quality, and computational efficiency.

S. Choueiry et al.[6] provides an overview of the genetic algorithm and its applications in solving optimization problems and modeling systems involving randomness. It focuses on the use of genetic algorithms for robot path planning in a static environment. The algorithm aims to determine the fastest route within a specified number of steps while avoiding obstacles in the environment. To enhance performance, the algorithm automatically disregards any suggested paths that intersect with the boundaries of the environment. Genetic algorithms are widely employed as a computational search tool for optimization and search problems, enabling the discovery of precise or approximate solutions. The algorithm's effectiveness is evaluated by testing it with different numbers of steps in various environments. The results demonstrate how the number of steps influences the selection of the optimal path to be followed. Overall, the paper highlights the utility of genetic algorithms in path planning optimization and presents findings from experiments conducted in different scenarios with the major disadvantage being that it is restricted to static environments and not yet generalized to real-world applications.

Guni Sharon et al.[9] propose the Conflict Based Search (CBS) algorithm for solving the multi-agent pathfinding problem (MAPF). Unlike previous methods, CBS adopts a two-level approach without converting the problem into a joint agent model. In the high-level phase, a Conflict Tree (CT) is constructed based on conflicts between individual agents, representing constraints on agent motion. The low-level phase utilizes fast single-agent searches to satisfy CT node constraints. CBS minimizes the number of examined states compared to A* while ensuring optimality. Additionally, the Meta-Agent CBS (MA-CBS) algorithm extends CBS by allowing agents to form joint agent groups, addressing limitations and improving performance. MA-CBS enhances existing MAPF solvers and demonstrates significant speedup, outperforming previous approaches by up to ten times. CBS guarantees optimality by considering conflicts and efficiently handles agent motion constraints. However, it introduces additional complexity and may face scalability challenges in scenarios with numerous agents, where alternative algorithms may be preferred for timely and efficient pathfinding solutions. Overall, CBS offers an efficient optimality trade-off for optimal multi-agent pathfinding.

III. Problem Statement

To design and develop optimal and time-efficient algorithms for path planning in obstacle-rich environments using meta-heuristic algorithms including Genetic, PSO, A-star and CBS algorithms.

A. Objectives

- To implement various meta-heuristic algorithms for finding optimal, collision-free paths in obstacle-rich environments.
- To analyze and experiment with our methodologies over single and multiple-agent path planning environments and finalize algorithms based accordingly.
- To simulate a real-life application based on single and multiple agent path planning for robot-controlled facilities.

IV. Methodology

A. Particle Swarm Optimisation (PSO)

Particle Swarm Optimization (PSO) is an optimization algorithm inspired by the collective behavior of bird flocking or fish schooling. PSO aims to find the optimal solution in a search space by simulating the movement and interaction of particles in a swarm. Each particle represents a potential solution, and they adjust their positions and velocities based on their own experience and the collective knowledge of the swarm. Particles are guided towards better solutions by continuously updating and improving their positions. PSO has been widely applied to various optimization problems due to its simplicity, efficiency in exploring the search space, and ability to handle both single-objective and multi-objective optimization tasks. The methodology utilized and the flowchart is given below.
Algorithm:

1) The PSO function is defined, which takes the problem to be solved as input along with optional parameters for customization. The required parameters such as the

TABLE I
SUMMARY OF LITERATURE SURVEY

Authors	Methodology	Merits	Limitations
Daniel Foead et al.[1]	Evaluate A-star algorithms across discrete domains	Favorable balance between optimal paths and computational efficiency	Slower convergence rate and the requirement for significant memory resources
Lisang Lui et al. [2]	A-star algorithm with a modification with obstacle nodes having infinite cost.	Improves computational efficiency and expedites path planning	Algorithm chooses less efficient routes simply to avoid obstacles.
David et al. [3]	Combined local and global search methods to guide movements	Efficient convergence to near-optimal solutions through global and local search	Premature convergence and limited robustness to objective function
Nasrollahy et al. [4]	Utilises PSO with a modified penalty cost function	Effectively handles dynamic environments and prevents local optima traps.	Lacks to validate the approach's effectiveness and efficiency
Z. Chen et al. [5]	Improved genetic algorithm in terms of crossover and mutation	Improved convergence speed, smoother planned paths, and obstacle avoidance	Lacks analysis of convergence speed, solution quality, and computational efficiency.
S. Choueiry et al. [6]	Use genetic operators such as selection to evolve and improve the paths.	Effectiveness in optimization and searching problems, hence enabling precise solutions.	Lack of generalization to real-world based applications.

maximum number of iterations, population size, coefficients, damping factor, callback function, and resetting frequency are extracted from the provided keyword arguments.
2) An empty particle template is defined, which represents an individual in the population. It consists of position, velocity, cost, and details attributes, along with a 'best' dictionary to store the best position and cost found by the particle so far and the information about the problem, including the cost function, variable bounds, and the number of variables is extracted. The global best is initialized to a high-cost value.
3) The initial population is created by generating random particles within the variable bounds. Each particle's cost and details are evaluated using the provided cost function.
4) The PSO loop iterates for a specified number of iterations. If resetting is enabled, particles' positions and velocities are randomly reset. For each particle, the velocity is updated using the PSO equations. The particle's position is updated by adding the velocity and clipping it within the variable bounds and the particle's cost and details are evaluated based on the updated position. If the particle's cost is better than its best cost, the best position, cost, and details are updated. If the particle's best cost is better than the global best cost, the global best is updated.
5) The inertia weight (w) is multiplied by the damping factor to gradually reduce the influence of previous velocities.
6) Once the PSO loop completes, the final gbest and the population are returned as the output.

B. A-Star Algorithm (A*)

A* is a popular path finding algorithm that efficiently finds the shortest path in a graph or grid. It combines Dijkstra's algorithm and heuristics to guide the search. In obstacle-rich environments, A* considers actual and estimated costs while avoiding obstacles. It explores neighboring nodes, discarding blocked ones, until reaching the goal or exhausting options. A*'s effectiveness lies in its admissible heuristic function, accurately estimating remaining costs. By balancing actual

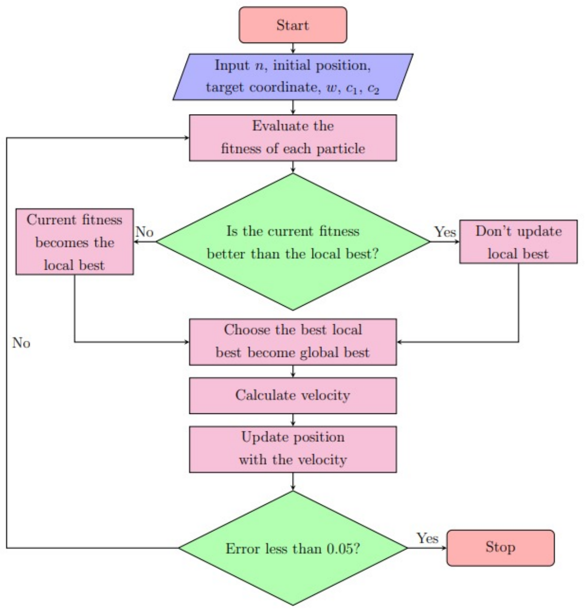
Start
Input n, initial position, target coordinate, w, c_1, c_2
Evaluate the fitness of each particle
Is the current fitness better than the local best?
No
Current fitness becomes the local best
Yes
Don't update local best
Choose the best local best become global best
Calculate velocity
Update position with the velocity
Error less than 0.05?
No
Yes
Stop

Fig. 1. Flowchart for PSO [3]

and estimated costs, A* intelligently navigates the search space, minimizing unnecessary exploration. With its ability to handle obstacles and find optimal paths, A* is widely used in various applications requiring efficient path finding in complex environments. The algorithm utilized and flowchart is given below.

Algorithm:

1) Create an instance of the `AStar` class with `s_start`, `s_goal`, and `heuristic_type`.
2) Initialize the `OPEN` list as an empty priority queue and the `CLOSED` list as an empty visited order list.
3) Set the parent of `s_start` as `s_start` and the cost to come `g` of `s_start` as 0.
4) Set the cost to come `g` of `s_goal` as infinity.
5) Push a tuple `(f_value(s_start), s_start)` into the `OPEN` priority queue.
6) While the `OPEN` priority queue is not empty, do the

following:

   a) Pop the state `s` with the minimum priority (lowest `f` value) from the `OPEN` priority queue.
   b) Add `s` to the `CLOSED` list.
   c) If `s` is the goal state `s_goal`, exit the loop.
   d) For each neighbor `s_n` of `s`:
      i) Calculate the new cost `new_cost` as the cost to come `g` of `s` plus the cost of the motion from `s` to `s_n`.
      ii) If `s_n` is not in the cost to come `g` dictionary, set its value to infinity.
      iii) If `new_cost` is less than the current cost to come `g` of `s_n`, update `g[s_n]` with `new_cost`, set the parent of `s_n` as `s`, and push a tuple `(f_value(s_n), s_n)` into the `OPEN` priority queue.

7) Extract the path by starting from the goal state `s_goal` and following the recorded parents until reaching the start state `s_start`.
8) Return the extracted path and the visited order `CLOSED`.

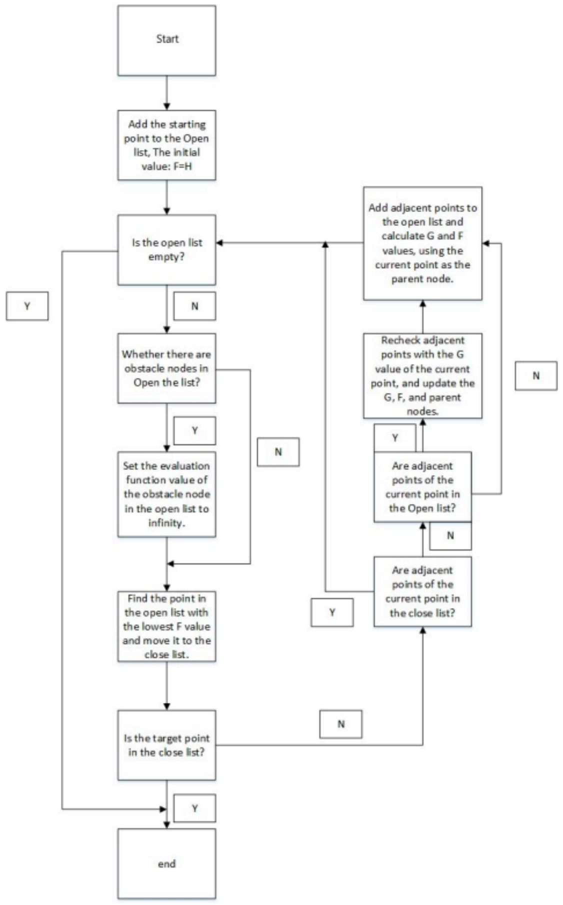
Start
Add the starting point to the Open list, The initial value: F=H
Is the open list empty?
Y
N
Whether there are obstacle nodes in Open the list?
Y
N
Set the evaluation function value of the obstacle node in the open list to infinity.
Find the point in the open list with the lowest F value and move it to the close list.
Is the target point in the close list?
N
Y
end
Add adjacent points to the open list and calculate G and F values, using the current point as the parent node.
Recheck adjacent points with the G value of the current point, and update the G, F, and parent nodes.
N
Y
Are adjacent points of the current point in the Open list?
N
Are adjacent points of the current point in the close list?
Y

Fig. 2. Flowchart for A* algorithm [8]

C. Genetic Algorithm

Genetic Algorithms (GAs) are optimization techniques inspired by the principles of natural selection and genetics. They are widely used to solve complex optimization problems by simulating the process of evolution. In a GA, a population of candidate solutions is evolved over multiple generations through the application of genetic operators such as selection, crossover, and mutation. The process starts with an initial population, where each solution is encoded as a chromosome. Selection mechanisms favor fit individuals based on their fitness value, representing their ability to solve the problem. Crossover combines genetic information from selected parents to create new offspring, while mutation introduces small random changes. These genetic operators drive the search for better solutions, iteratively improving the population until a satisfactory solution is found. [10] GAs offer a powerful and versatile approach for solving optimization problems in various domains. The algorithm utilized and the flowchart is given below.

Algorithm:

1) Define the `mutation` function:
   - Select a random index within the chromosome.
   - Flip the bit at the selected index.
   - Return the mutated chromosome.
2) Define the `fitness` function:
   - Calculate the path length of the chromosome.
   - Calculate the fitness as the reciprocal of the path length and return fitness.
3) Define the `sortByFitness` function which returns the list of chromosomes sorted in descending order of their fitness values.
4) Define the `chooseRandomParent` function which returns the selected random parent's chromosome.
5) Define the `crossover` function which performs crossover by combining parts of the parent chromosomes based on the split size.
6) Define the `chromosomeValid` function:
   - Iterate over the chromosome genes and their corresponding path points.
   - If a path overlaps with an obstacle, return False.
   - If all paths are valid, return True.
7) Define the `pathOverlapsObstacle` function which returns true or false if the path intersects an obstacle or not respectively.
8) Define the `generatePopulation` function:
   - Create an empty population list.
   - Generate the initial population by iterating for the specified population size:
     – Generate a random chromosome. If the generated chromosome is valid, append it to the population list.
   - Return the generated population.
9) Define the `generateChromosome` function:

- Initialize the chromosome with '1' to represent the source point.
- Iterate for each path point starting from the second point:
  - Check the validity of the path. If the path is valid, generate a gene randomly and update the previous path point.
  - Append the gene to the chromosome.
- Return the valid chromosome.

10) Define the `start` function:
    - Generate an initial population using the generatePopulation function.
    - Calculate the path lengths for each chromosome in the population.
    - Iterate for each generation:
      - Create a new population and clear the path lengths.
      - Sort the population based on fitness.
      - Iterate for each chromosome in the population:
        - Choose two parents.
        - Perform crossover and mutation on the parents to generate a child chromosome.
        - Check the validity of the child's chromosome.
        - Calculate the path length of the child chromosome.
        - Append the child chromosome and its path length to the new population.
      - Update the population with the new population.
      - Plot the current population.

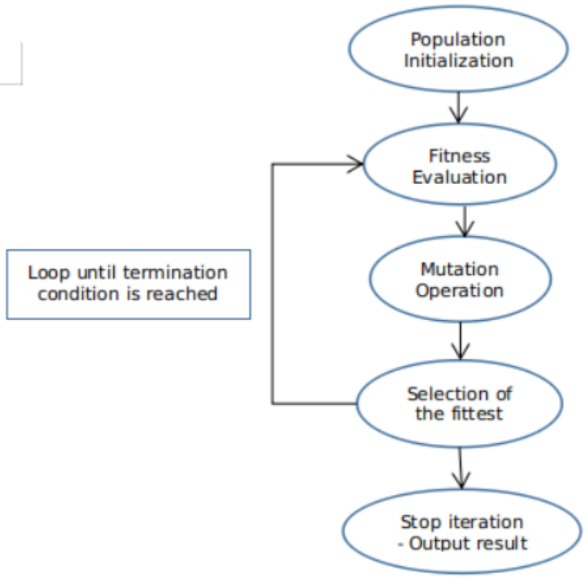
Population Initialization
Fitness Evaluation
Loop until termination condition is reached
Mutation Operation
Selection of the fittest
Stop iteration - Output result

Fig. 3. Flowchart for Genetic algorithm [11]

D. Conflict based search (CBS)

CBS (Conflict Based Search) is an optimal multi-agent pathfinding algorithm that operates at two levels. It avoids converting the problem into a single 'joint agent' model [13] . At the high level, CBS searches for conflicts between individual agents and builds a Conflict Tree (CT). Each node in the CT represents a set of motion constraints for the agents. At the low level, fast single-agent searches are performed to satisfy the constraints imposed by the high-level CT nodes. CBS has the advantage of examining fewer states than A* while maintaining optimality. It is a flexible and efficient approach for solving multi-agent pathfinding problems. The algorithm utilized is given below.
Algorithm:

1) Define an `AStar` class that implements the A* search algorithm used for low-level search within CBS. It includes methods for reconstructing the path from the goal, searching for a solution, and evaluating heuristics.
2) Define a `CBS` class that implements the Conflict-Based Search algorithm. It takes an `Environment` object as input.
3) Define a `getFirstConflict` method that iterates through each time step of a given solution to identify the first conflict that occurs. It checks for vertex conflicts by comparing the states of pairs of agents at each time step and identifies edge conflicts by comparing the states of agents between consecutive time steps. If a conflict is found, it returns a Conflict object containing the conflict details. If no conflicts are found, it returns False.
4) Define a `search` method in the `CBS` class performs the CBS algorithm as:
   - It initializes the start node, which includes an empty constraint dictionary and a computed initial solution, and the start node is added to the open set.
   - While the open set is not empty, the algorithm proceeds to the next node `P`.
   - The environment's constraint dictionary is updated with `P`'s constraint dictionary.
   - The first conflict in the current solution is obtained using the `getFirstConflict` method.
   - If there are no conflicts, a solution is found and if a conflict is found, new nodes are created by adding the conflict constraints to `P`'s constraint dictionary for each involved agent.
   - For each new node, the environment's constraint dictionary is updated, a new solution is computed, and the cost is calculated.
   - If a new node is not already in the closed set, it is added to the open set.
   - The process continues until a solution is found or the open set is empty.

V. Experimental Results and Analysis

A. Genetic Algorithm

1) Environmental setup:
   We have created a simulation for a 2D environment consisting of 25 path points, and 20 obstacles, with each obstacle having maximum width and height of 9 each,

and minimum width and height of 4 each. Among the 20 obstacles, 7 are hard-coded whereas the remaining are random. The current generation, chromosome, and length of the path are displayed for each iteration. The maximum number of generations taken is 10 and the population size considered is 100.

2) Analysis:
The results obtained from the five experiments conducted are given below in the table where IPL stands for initial path length, FPL stands for final path length, TG stands for total number of generations, and TT stands for total time taken to find optimal path in seconds.

Genetic Algorithm			
TT	IPL	FPL	TG
20 sec	410	151	20
23 sec	384	162	18
16 sec	423	155	16
27 sec	475	172	21
13 sec	365	132	13

3) Conclusions:
   - For an average initial path length of 411.4, the algorithm is capable of reducing the path length to an optimal average length of 154.4 resulting in an average 62.4% reduction in path length.
   - The average time elapsed is around 19.8 seconds but it must be taken into consideration that time taken entirely depends on the complexity of the environment and the number of obstacles.
   - The average number of generations used to reach an optimal solution is 17 and it also depends on the complexity of the environment and the number of obstacles.

4) Output:

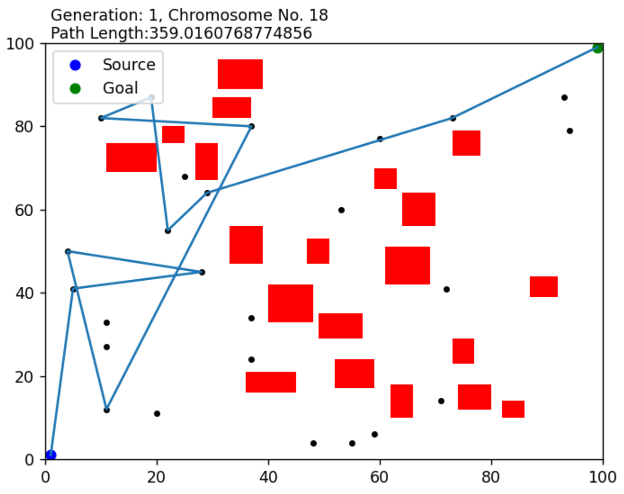

Fig. 4. Generation 1

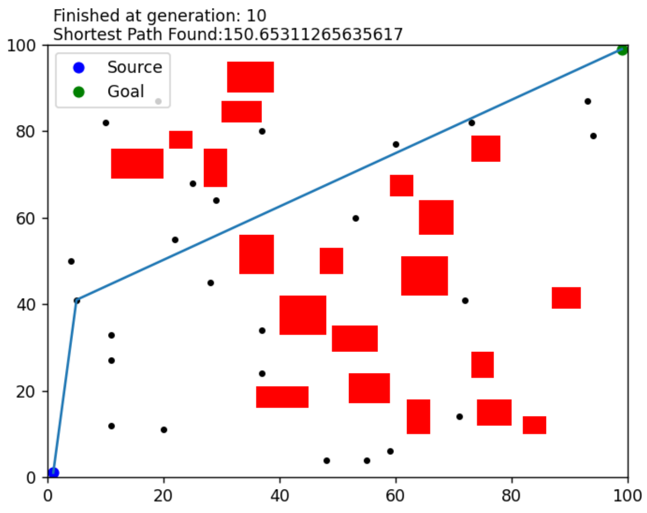

Fig. 5. Generation 10

5) Time and space complexity: The time and space complexities respectively for genetic operators defined are as under:
   - Mutation: O(1) and O(chromosomeSize)
   - Fitness: Both O(pathPointLength)
   - Crossover: Both O(parentChromosomeLength)
   - Population generation: Both O(populationSize * pathPointLength)

   Overall Time Complexity: O(generations * populationSize * (pathPointsLength + obstaclesSize))
   Overall Space Complexity: O(populationSize * pathPointsLength + obstaclesSize)

B. Particle swarm (PSO)

1) Environmental setup:
We have created a simulation for a 2D environment consisting of 7 obstacles, with each obstacle being circular and having manually coded radii. The current iteration and length of the path are displayed for each iteration. The maximum number of iterations taken is 100, maximum population size considered is 100, personal influence is 2, and social influence factor taken is 1.

2) Analysis:
The results obtained from the five experiments conducted are given below in the table where IPL stands for initial path length, FPL stands for final path length, TI stands for the total number of iterations, and TT stands for total time taken to find an optimal path in seconds.

PSO Algorithm			
TT	IPL	FPL	TI
8	458	131	32
12 sec	341	158	40
5 sec	133	129	6
18 sec	503	155	87
22 sec	252	147	24

3) Conclusions:
   - For an average initial path length of 508.2, the algorithm is capable of reducing the path length to an optimal average length of 142.4 resulting in an average 72% reduction in path length.
   - The average time elapsed is around 13 seconds but it must be taken into consideration that time taken entirely depends on the complexity of the environment and the number of obstacles.
   - The average number of generations used to reach an optimal solution is 37 and it also depends on the complexity of the environment and the number of obstacles.

4) Output:

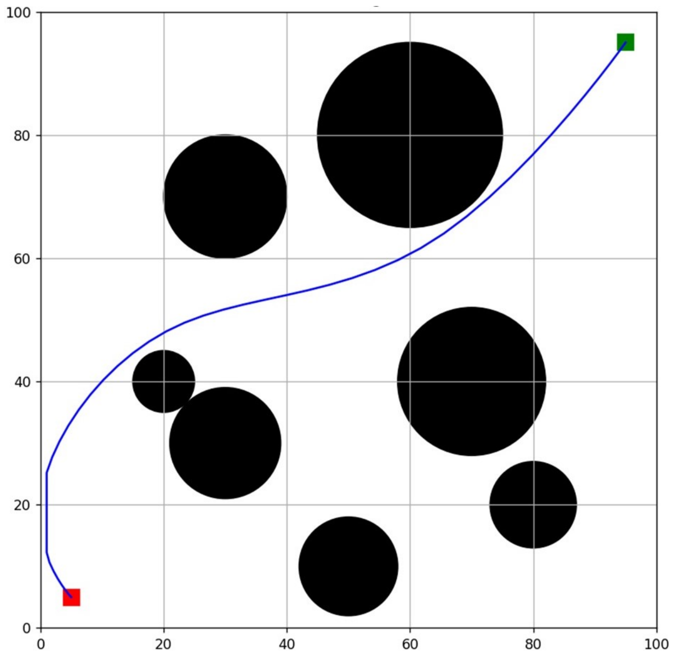

Fig. 6. PSO Iteration 1

5) Time complexity analysis:
The time complexity of the Particle Swarm Optimization (PSO) algorithm depends on factors such as problem size and convergence rate. In the PSO algorithm, each iteration involves initializing particles, evaluating fitness, updating personal and global bests, and updating positions and velocities. These steps have a time complexity of O(N), where N is the number of particles. The algorithm continues until a termination condition is met. Therefore, the overall time complexity can be approximated as O(N * Iterations), where Iterations is the number of iterations needed for convergence.

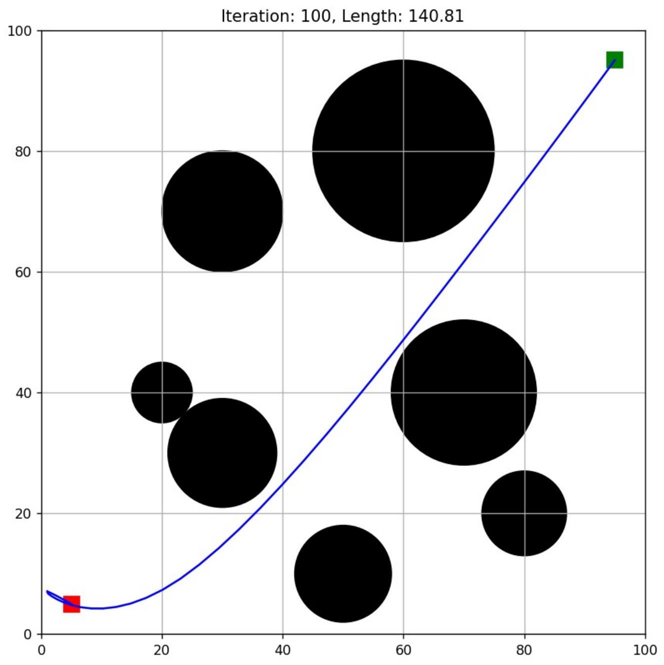

Fig. 7. PSO Iteration 100

6) Space complexity analysis:
The space complexity of the Particle Swarm Optimization (PSO) algorithm primarily depends on the dimensionality of the problem and the number of particles in the population. The space required to store a single particle representation is O(D), where D is the problem's dimensionality. The population size determines the space complexity for storing all particles, resulting in O(N * D), where N is the number of particles. Additionally, there is space needed for the global best position, which is also O(D). Other variables and data structures used for calculations may contribute to the space complexity, but their impact varies depending on the problem and implementation details. Overall, the approximate space complexity of the PSO algorithm can be expressed as O(N * D) + O(D) + Additional space requirements.

C. *A-star algorithm*

1) Environmental setup:
We have created a simulation for a 2D environment with a background size of 51x31. The environment consists of obstacles represented by coordinates on the map. The obstacles are initialized in a specific pattern, including the boundaries of the map and additional obstacles within the map. The obstacles create a blocked path in certain areas, restricting movement in those

regions. The environment also defines a set of possible motions that can be made, allowing movement in eight directions (horizontal, vertical, and diagonal). Overall, the environment provides the structure and constraints for agents or objects to interact and navigate within the 2D space

2) Output:

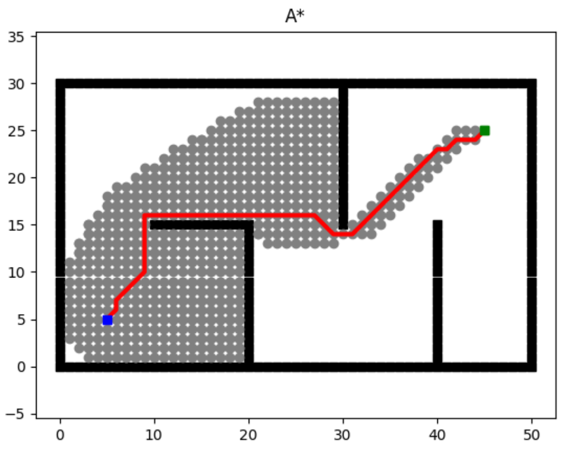

Fig. 8. A* Search

3) Time complexity analysis: The time complexity of the A* algorithm depends on the characteristics of the problem and the efficiency of the heuristic function. In the worst case, where the heuristic is not informative and all nodes need to be expanded, the time complexity can be exponential, specifically O(b^d), where b is the branching factor and d is the depth of the optimal solution. However, if the heuristic is admissible and consistent, A* is guaranteed to find an optimal solution with polynomial time complexity, specifically O(b^d), but with a much smaller constant factor compared to uninformed search algorithms like breadth-first search or depth-first search.
4) Space complexity analysis: The space complexity of the A* algorithm is determined by the data structures used to store the open set, closed set, parent pointers, and other variables. In the worst case, where all nodes need to be stored, the space complexity can be exponential, specifically O(b^d). The actual space complexity depends on the size of the problem, the branching factor, and the memory requirements of the data structures used.

D. Conflict-based search (CBS)

1) Environmental setup:
We have created a simulation of a 2D environment as N by N grid where N = 4,8,16,32, with multiple static obstacles acting like walls along with other agents acting as dynamic obstacles. The number of agents can range from 0 to 20 and grid size can go up to any size. Each agent is assigned a starting point and a finishing point as coordinates, and every agent tries to reduce the path length as much as possible and also finds a collision-free path. The velocity of each agent is fixed and the time taken for each simulation is no more than 10 seconds for an 8 by 8 grid.
   - Average collisions is 0
   - Average time: 10 seconds for 8 by 8 grid and 30 seconds for 32 by 32 grid. Note that the time can be drastically improved by increasing the velocities of the agents.
2) Output :

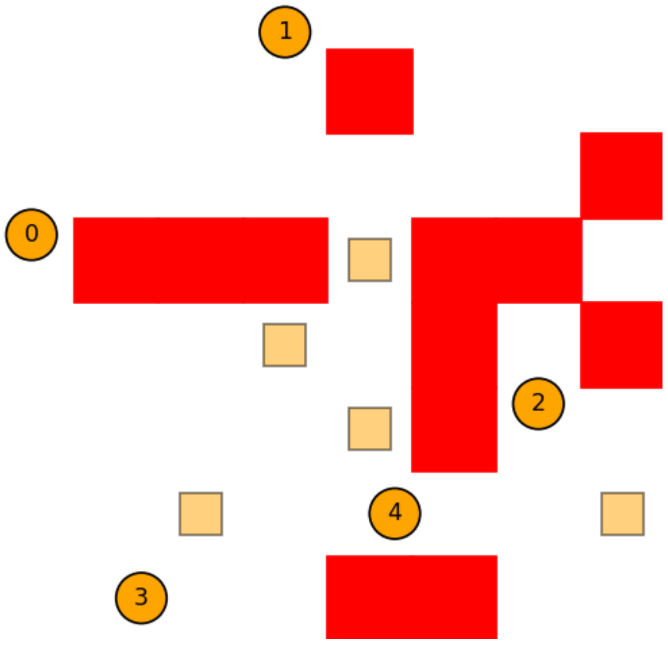

Fig. 9. CBS Start state

VI. Conclusion

In path-finding problems involving environments with numerous obstacles, solving them through brute force measures becomes impractical due to the exponential time complexity. Therefore, alternative approaches, known as meta-heuristic algorithms, are employed to achieve solutions in polynomial time. Our research indicates that algorithms like genetic algorithms, Particle Swarm Optimization (PSO), and A-star is effective in static obstacle-based environments. However, for scenarios with dynamic obstacles and multi-agent path planning, the Conflict-based Search algorithm has demonstrated superior performance. As part of our future work, we intend to focus on enhancing the efficiency of genetic algorithms. This will involve improvements in chromosome generation techniques and the development of pruning methods to reduce the number of generations required to reach the optimal state. By reducing the computational overhead, we aim to significantly improve the runtimes of genetic algorithms. Furthermore, we

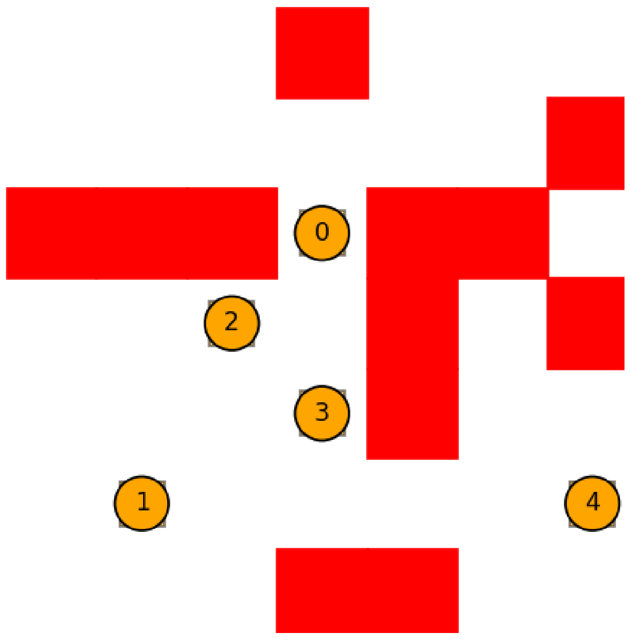

Fig. 10. CBS Goal state

are also dedicated to exploring alternative optimal methods for multi-agent path planning problems in three-dimensional (3D) environments that involve velocity alterations. By considering the complexities introduced by 3D spaces and the dynamics of agent movement, we aim to devise innovative approaches that can effectively handle such scenarios.

Individual contribution

1) Saliq Gowhar Khan - Path planning via genetic algorithms and CBS.
2) Ashutosh Kumar Gupta - Path planning via A-star algorithm and CBS.
3) Bhavya - Path planning via PSO algorithm and CBS.

References

[1] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, Eric Gunawan,A Systematic Literature Review of A* Pathfinding,Procedia ComputerScience,Volume 179,2021,Pages 507-514,ISSN 1877-0509,

[2] Liu, L.; Wang, B.; Xu, H. Research on Path-Planning Algorithm Integrating Optimization A-Star Algorithm and Artificial Potential Field Method. Electronics 2022, 11, 3660. https://doi.org/10.3390/electronics11223660

[3] David,. (2021). Using Particle Swarm Optimization as Pathfinding Strategy in a Space with Obstacles. 10.31219/osf.io/zs9p5.

[4] A. Z. Nasrollahy and H. H. S. Javadi, "Using Particle Swarm Optimization for Robot Path Planning in Dynamic Environments with Moving Obstacles and Target," 2009 Third UKSim European Symposium on Computer Modeling and Simulation, Athens, Greece, 2009, pp. 60-65, doi: 10.1109/EMS.2009.67.

[5] Z. Chen, G. Xiong, S. Liu, Z. Shen and Y. Li, "Path Planning of Mobile Robot Based on an Improved Genetic Algorithm," 2022 IEEE 2nd International Conference on Digital Twins and Parallel Intelligence (DTPI), Boston, MA, USA, 2022, pp. 1-6, doi: 10.1109/DTPI55838.2022.9998894.

[6] S. Choueiry, M. Owayjan, H. Diab and R. Achkar, "Mobile Robot Path Planning Using Genetic Algorithm in a Static Environment," 2019 Fourth International Conference on Advances in Computational Tools for Engineering Applications (ACTEA), Beirut, Lebanon, 2019, pp. 1-6, doi: 10.1109/ACTEA.2019.8851100.

[7] Gad, A.G. Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review. Arch Computat Methods Eng 29, 2531–2561 (2022). https://doi.org/10.1007/s11831-021-09694-4

[8] Liu, Lisang Wang, Bin Xu, Hui. (2022). Research on Path-Planning Algorithm Integrating Optimization A-Star Algorithm and Artificial Potential Field Method. Electronics. 11. 3660. 10.3390/electronics11223660.

[9] Guni Sharon, Roni Stern, Ariel Felner, Nathan R. Sturtevant, Conflict-based search for optimal multi-agent pathfinding, Artificial Intelligence, Volume 219, 2015,Pages 40-66,ISSN 0004-3702,

[10] Katoch, S., Chauhan, S.S. Kumar, V. A review on genetic algorithm: past, present, and future. Multimed Tools Appl 80, 8091–8126 (2021). https://doi.org/10.1007/s11042-020-10139-6

[11] S. D. Immanuel and U. K. Chakraborty, "Genetic Algorithm: An Approach on Optimization," 2019 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2019, pp. 701-708, doi: 10.1109/ICCES45898.2019.9002372.

[12] Y. Liang and M. Liu, "Multi-AGV Simulation System Based on Bipartite Graph, Space-Time A*, and Conflict Search," 2022 China Automation Congress (CAC), Xiamen, China, 2022, pp. 3767-3772, doi: 10.1109/CAC57257.2022.10056052.

[13] Z. Ren, S. Rathinam and H. Choset, "A Conflict-Based Search Framework for Multiobjective Multiagent Path Finding," in IEEE Transactions on Automation Science and Engineering, vol. 20, no. 2, pp. 1262-1274, April 2023, doi: 10.1109/TASE.2022.3183183.