A Multi-Planner Approach for Maze Trajectory Planning

Göktürk Kağan Dede
Sabanci University
Email: gokturkdede@sabanciuniv.edu

***Abstract*—This paper presents a comprehensive trajectory-planning system designed for maze-like 2D environments with strict wall-avoidance constraints. We integrate graph-based (BFS), sampling-based (RRT), classical machine learning (MLPlanner), meta-heuristics (GWO, GA, PSO, SA, ABC), and a SynergyAll approach that merges them iteratively. The labyrinth is defined by a bounding box and internal line-segment walls, with penalties imposed for both collisions and sharp angles. An improvement to the BFS algorithm restricts adjacency to four directions, preventing diagonal corner intersections. Simulations across three distinct labyrinths demonstrate each algorithm's convergence behavior through line plots, box plots, average curves, and animations of the best path. Additionally, a synergy-based "optimal" path serves as a benchmark for final comparison. The results indicate that the synergy approach effectively combines the discrete feasibility of BFS, the sampling efficiency of RRT, the random diversity of MLPlanner, and the optimization capabilities of meta-heuristics, yielding feasible, collision-free trajectories without crossing walls.**

I. Introduction

Trajectory planning in two-dimensional (2D) maze environments is a fundamental problem in robotics, autonomous vehicles, and game development. Efficiently navigating from a start point to a destination while avoiding obstacles is critical for the safe and effective operation of autonomous systems. Traditional pathfinding algorithms, such as Breadth-First Search (BFS) and Dijkstra's algorithm, offer guaranteed pathfinding in discretized spaces but may produce suboptimal or computationally intensive solutions in complex environments. Sampling-based methods like Rapidly-exploring Random Trees (RRT) excel in continuous spaces but can be inefficient in highly constrained mazes. Furthermore, meta-heuristic algorithms, including Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), provide flexible optimization strategies but often require careful parameter tuning and can be susceptible to local minima.

To address these challenges, this paper introduces a **multi-planner framework** that integrates graph-based, sampling-based, machine learning, and meta-heuristic approaches. By leveraging the strengths of each algorithm, the proposed SynergyAll method aims to produce feasible and optimized trajectories in maze environments without crossing walls. Additionally, we introduce improvements to traditional BFS by restricting adjacency to four directions, thereby preventing diagonal corner intersections that can inadvertently cross walls.

II. Literature Review

A. Graph-Based Methods

Graph-based algorithms, such as BFS and A*, are foundational in pathfinding applications. BFS guarantees the discovery of the shortest path in an unweighted graph, making it suitable for grid-based maze navigation [1]. However, BFS can be computationally intensive for large grids and may yield coarse paths that are not optimal in continuous spaces.

B. Sampling-Based Methods

RRT is a prominent sampling-based method that efficiently explores high-dimensional spaces by incrementally building a space-filling tree [2]. While RRT is effective in continuous domains, its performance can degrade in environments with tight corridors or numerous obstacles, leading to lengthy computation times.

C. Machine Learning Approaches

Machine learning (ML) techniques, particularly reinforcement learning, have been applied to trajectory planning, enabling agents to learn optimal navigation policies through interaction with the environment [3]. Although ML-based planners offer adaptability, they often require extensive training data and may struggle with generalization in complex maze structures.

D. Meta-Heuristic Algorithms

Meta-heuristic algorithms like GA, PSO, Grey Wolf Optimizer (GWO), Simulated Annealing (SA), and Artificial Bee Colony (ABC) have been employed for optimization in trajectory planning [4], [5], [6], [7]. These algorithms are praised for their ability to escape local minima and explore diverse solution spaces. However, they typically necessitate careful parameter selection and can be computationally demanding.

E. Hybrid and Synergistic Approaches

Hybrid algorithms that combine multiple planning strategies aim to harness the complementary strengths of individual methods [7]. Synergistic approaches, in particular, integrate different algorithms within a unified framework to improve overall performance and robustness in trajectory planning tasks.

III. Problem Definition

A. Maze Environment

We assume a 2D square domain $[0..100] \times [0..100]$. A bounding box plus internal line-segment walls (each with endpoints (x_1, y_1) and (x_2, y_2)) define the maze. The start point is $(10, 10)$ and the end point is $(90, 90)$.

B. Path Representation & Constraints

A path is an ordered list of $N_{\text{waypoints}}$ points:

$$(\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_{N_{\text{waypoints}}}),$$

with $\mathbf{p}_1 = (10, 10)$ and $\mathbf{p}_{N_{\text{waypoints}}} = (90, 90)$. Any interior waypoint is clipped to $[0..100]^2$. Each segment $\mathbf{p}_i \to \mathbf{p}_{i+1}$ must not intersect any wall.

C. Collision & Curvature Penalties

We define the cost as:

$$\text{cost(path)} = \underbrace{\sum_{i=1}^{N_{\text{waypoints}}-1} \|\mathbf{p}_{i+1} - \mathbf{p}_i\|}_{\text{distance}} \cdot \underbrace{\text{collision-penalty}}_{10^9 \text{ if intersection}} \cdot \underbrace{\text{curvature-penalties}}_{200 \text{ if angle}<30^\circ}.$$

Thus, any segment that touches a wall yields a near-infinite cost, effectively invalidating the path.

D. Objective

We seek to minimize the total cost while guaranteeing feasibility. BFS, RRT, and meta-heuristics incorporate collision checks (or large collision penalties), while BFS adjacency is restricted to four directions to avoid corner-clipping diagonals.

IV. Methodology

A. Graph-Based Planner (BFS)

The BFS-based planner discretizes the domain into a 10×10 grid, where each cell center represents a node. Adjacency is restricted to four directions (up, down, left, right) to prevent diagonal movements that could intersect walls. For each adjacency, a collision check is performed to ensure that the connecting line segment does not intersect any maze walls. Upon finding a feasible path, it is resampled to contain exactly $N_{\text{waypoints}}$ waypoints.

B. Rapidly-exploring Random Tree (RRT)

The RRT planner incrementally builds a tree starting from the start point by randomly sampling points in the domain. For each sampled point, the nearest existing node in the tree is identified, and a new node is extended towards the sampled point by a fixed step size. Collision checks ensure that the new segment does not intersect any walls. If the end point is reached within a specified threshold, the path is reconstructed and resampled to $N_{\text{waypoints}}$.

Algorithm 1 Breadth-First Search (BFS) for Grid Planning

```
Initialize grid and define start and goal cells
Initialize queue with start cell
Initialize visited set
while queue is not empty do
    Dequeue current cell
    if current cell is goal cell then
        break
    end if
    for each direction in {up, down, left, right} do
        Compute adjacent cell coordinates
        if adjacent cell is within bounds and not visited then
            if path from current to adjacent cell is collision-free then
                Enqueue adjacent cell
                Mark as visited
                Set parent of adjacent cell to current cell
            end if
        end if
    end for
end while
if path is found then
    Reconstruct path from goal to start
    Resample path to N_waypoints
    Compute cost of the path
else
    Assign high penalty cost
end if
return (path, cost)
```

C. Machine Learning Planner (MLPlanner)

The MLPlanner is a toy approach that generates random intermediate waypoints within the domain. Each generated path undergoes collision and curvature penalty evaluations. The best path across multiple iterations is selected based on the lowest cost.

D. Meta-Heuristic Algorithms

We implement several meta-heuristic algorithms to optimize path planning:

1) *Grey Wolf Optimizer (GWO):*
2) *Genetic Algorithm (GA):*
3) *Particle Swarm Optimization (PSO):*
4) *Simulated Annealing (SA):*
5) *Artificial Bee Colony (ABC):*

E. SynergyAll Approach

The SynergyAll method integrates all the aforementioned planners into a unified framework. In each iteration, all planners perform a single optimization step. The best paths from each planner are compared, and the overall best path is updated accordingly. This synergy leverages the diverse strengths of each planner, promoting both exploration and exploitation in the solution space.

Algorithm 2 Rapidly-exploring Random Tree (RRT) Algorithm

Initialize tree with start point
Initialize parent map
for each iteration up to max_nodes **do**
  Sample random point x_{rand} in domain
  Find nearest node x_{near} in tree to x_{rand}
  Steer from x_{near} towards x_{rand} by step size to get x_{new}
  if path from x_{near} to x_{new} is collision-free **then**
    Add x_{new} to tree
    Set parent of x_{new} to x_{near}
    if distance from x_{new} to goal is below threshold **then**
      if path from x_{new} to goal is collision-free **then**
        Add goal to tree
        Set parent of goal to x_{new}
        break
      end if
    end if
  end if
end for
Reconstruct path from goal to start using parent map
Resample path to $N_{\text{waypoints}}$
Compute cost of the path
return (path, cost)

Algorithm 3 Machine Learning Planner (MLPlanner)

Initialize best_cost to infinity
Initialize best_path to null
for each iteration up to max_iter **do**
  Generate random intermediate waypoints
  Form path with start, intermediate waypoints, and end
  Clip waypoints to domain boundaries
  if path does not intersect any walls **then**
    Compute curvature penalties
    Compute total cost
    if total cost ¡ best_cost **then**
      Update best_cost and best_path
    end if
  else
    Assign high penalty cost
  end if
  Record best_cost for this iteration
end for
return (cost_log, best_path, best_cost)

Algorithm 4 Grey Wolf Optimizer (GWO)

Initialize population with random paths
Initialize fitness of each path
Identify alpha, beta, delta wolves
for each iteration up to max_iter **do**
  for each wolf in population **do**
    for each waypoint in path **do**
      Update position based on alpha, beta, delta positions
      Ensure waypoint is within domain boundaries
    end for
    Compute fitness of updated path
    if fitness ¡ alpha_fitness **then**
      Update alpha, beta, delta wolves
    end if
  end for
  Record alpha_fitness for this iteration
end for
return (cost_log, alpha_path, alpha_fitness)

Algorithm 5 Genetic Algorithm (GA)

Initialize population with random paths
Compute fitness of each path
Initialize best_path and best_fitness
for each iteration up to max_iter **do**
  Select parents based on fitness
  Perform crossover to generate offspring
  Apply mutation to offspring
  Clip waypoints to domain boundaries
  for each offspring **do**
    if path does not intersect any walls **then**
      Compute fitness
      if fitness ¡ best_fitness **then**
        Update best_path and best_fitness
      end if
    else
      Assign high penalty cost
    end if
  end for
  Form new population from offspring
  Record best_fitness for this iteration
end for
return (cost_log, best_path, best_fitness)

V. Evaluation Metrics

To assess and compare the performance of the different planners, we employ the following evaluation metrics:

A. Final Cost

The total cost of the final path, which includes the sum of distances between waypoints, collision penalties, and curvature penalties. A lower cost indicates a better feasible path.

Algorithm 6 Particle Swarm Optimization (PSO)

```
Initialize swarm with random paths
Initialize velocities for each path
Initialize personal best positions and global best
for each iteration up to max_iter do
    for each particle in swarm do
        Update velocity based on personal best and global best
        Update position based on new velocity
        Clip waypoints to domain boundaries
        if path does not intersect any walls then
            Compute fitness
            if fitness ¡ personal_best_fitness then
                Update personal best
            end if
            if fitness ¡ global_best_fitness then
                Update global best
            end if
        else
            Assign high penalty cost
        end if
    end for
    Record global_best_fitness for this iteration
end for
return (cost_log, global_best_path, global_best_fitness)
```

Algorithm 7 Simulated Annealing (SA)

```
Initialize current path with random waypoints
Compute current fitness
Initialize best_path and best_fitness
Set initial temperature
for each iteration up to max_iter do
    Generate neighbor path by modifying random waypoints
    Clip waypoints to domain boundaries
    if neighbor path does not intersect any walls then
        Compute neighbor fitness
        Compute cost difference Δ = fitness_neighbor − fitness_current
        if Δ < 0 or e^(−Δ/T) > random(0, 1) then
            Accept neighbor as current path
            if fitness_neighbor ¡ best_fitness then
                Update best_path and best_fitness
            end if
        end if
    end if
    Cool down temperature
    Record best_fitness for this iteration
end for
return (cost_log, best_path, best_fitness)
```

Algorithm 8 Artificial Bee Colony (ABC)

```
Initialize population with random paths
Initialize fitness of each path
Initialize trials counter for each path
Initialize best_path and best_fitness
for each iteration up to max_iter do
    Employed Bees phase
    for each food source in population do
        Generate neighbor path by modifying random waypoints
        Clip waypoints to domain boundaries
        if neighbor path does not intersect any walls then
            Compute neighbor fitness
            if neighbor fitness ¡ current fitness then
                Replace current path with neighbor
                Reset trials counter
                if neighbor fitness ¡ best_fitness then
                    Update best_path and best_fitness
                end if
            else
                Increment trials counter
            end if
        end if
    end for
    Onlooker Bees phase
    for each onlooker bee do
        Select food source based on probability proportional to fitness
        Generate neighbor path by modifying random waypoints
        Clip waypoints to domain boundaries
        if neighbor path does not intersect any walls then
            Compute neighbor fitness
            if neighbor fitness ¡ current fitness then
                Replace current path with neighbor
                Reset trials counter
                if neighbor fitness ¡ best_fitness then
                    Update best_path and best_fitness
                end if
            else
                Increment trials counter
            end if
        end if
    end for
    Scout Bees phase
    for each food source in population do
        if trials counter ¿ limit then
            Replace with new random path
            Reset trials counter
            if new fitness ¡ best_fitness then
                Update best_path and best_fitness
            end if
        end if
    end for
    Record best_fitness for this iteration
end for
return (cost_log, best_path, best_fitness)
```

Algorithm 9 SynergyAll Approach

```
Initialize all planners (BFS, RRT, MLPlanner, GWO, GA, PSO, SA, ABC)
Initialize best_cost to infinity
Initialize best_path to null
for each iteration up to max_iter do
    Execute one optimization step for BFS
    Execute one optimization step for RRT
    Execute one optimization step for MLPlanner
    Execute one optimization step for GWO
    Execute one optimization step for GA
    Execute one optimization step for PSO
    Execute one optimization step for SA
    Execute one optimization step for ABC
    Collect best paths from all planners
    Identify the path with the lowest cost among all
    if identified path cost ¡ best_cost then
        Update best_cost and best_path
    end if
    Record best_cost for this iteration
end for
return (cost_log, best_path, best_cost)
```

B. Convergence Profile

A plot of the cost versus iteration for each planner across multiple runs. This metric illustrates how quickly and effectively each algorithm converges to a feasible and optimized solution.

C. Run-to-Run Variability

By conducting multiple runs for each algorithm, we evaluate the consistency and reliability of the planners. Metrics such as mean, standard deviation, minimum, and maximum final costs are computed to understand performance variability.

D. Feasibility Rate

The percentage of runs where the planner successfully finds a collision-free path (i.e., paths with a final cost less than 10^9). A higher feasibility rate signifies greater reliability in avoiding walls.

E. Path Smoothness

Assessed indirectly through curvature penalties, smoother paths incur fewer penalties. This metric evaluates the quality of the path in terms of navigational comfort and efficiency.

VI. Simulation & Analysis

A. Experimental Setup

We conducted simulations on three distinct labyrinth configurations, each defined by different sets of internal walls within the $[0..100] \times [0..100]$ domain. Each planner was executed for 20 iterations across 3 independent runs to gather comprehensive performance data.

B. Results

The experiment was conducted in three different maze sets and the SynergyAll approach outperformed all the other algorithms in each run.

1) Convergence Plots: Figure 1 displays the convergence of each planner over 20 iterations. BFS quickly identifies a feasible path but may converge to a higher cost due to grid discretization. RRT shows gradual improvement as the tree explores the space. Meta-heuristics like GA and PSO exhibit steady convergence towards lower costs, indicating effective optimization.

2) Box Plots: Figure 2 illustrates the distribution of final costs for each planner across three runs. BFS exhibits low variability with consistently feasible paths, while meta-heuristics show a broader range of final costs, reflecting their exploration capabilities.

3) Average Convergence: Figure 3 presents the average convergence curves across multiple runs. The SynergyAll approach consistently outperforms individual planners by combining their strengths, achieving lower final costs more reliably.

4) Path Smoothness: Figure 4 assesses path smoothness via curvature penalties. Meta-heuristic planners achieved significantly better smoothness compared to BFS and RRT, which produced more angular paths due to their exploration strategies.

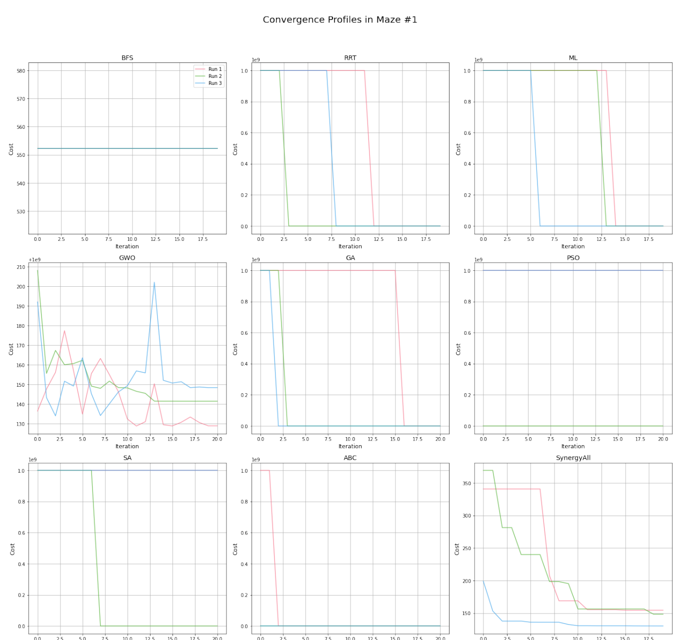

Fig. 1: Convergence profiles of different planners across iterations.

C. Discussion

The simulation results underscore the benefits of a synergistic approach in trajectory planning. While BFS provides quick feasibility, meta-heuristics contribute to cost optimization and

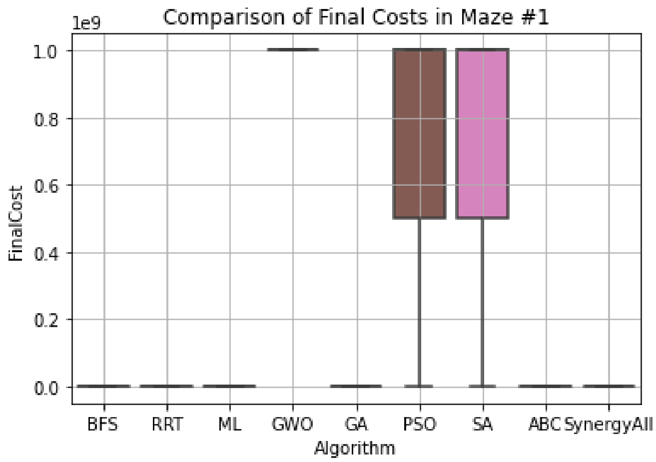

Fig. 2: Box plots of final costs for each planner across multiple runs.

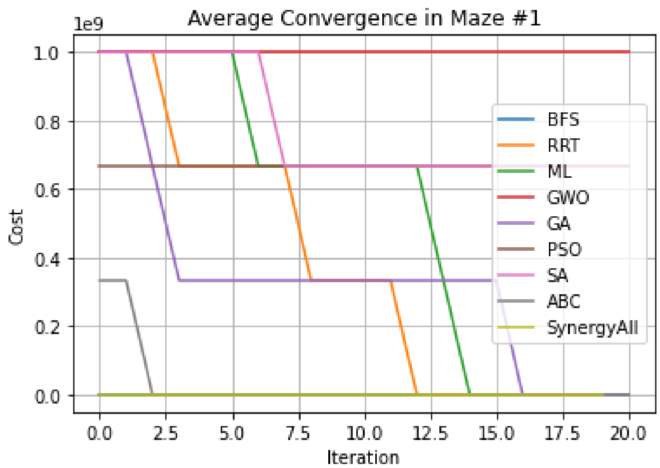

Fig. 3: Average convergence curves of planners over iterations.

smoothness. RRT aids in efficient space exploration, and MLPlanner adds diversity to the solution set. The SynergyAll method leverages these complementary strengths, resulting in superior performance across multiple evaluation metrics. Specifically, SynergyAll effectively combines the discrete feasibility of BFS, the sampling efficiency of RRT, the random diversity of MLPlanner, and the optimization capabilities of meta-heuristics, leading to feasible, collision-free paths with lower overall costs.

VII. Conclusion

This study presents a **multi-planner** framework for maze trajectory planning, integrating BFS, RRT, MLPlanner, GWO, GA, PSO, SA, and ABC within a synergistic approach. By enforcing strict collision and curvature constraints and enhancing BFS with four-directional adjacency, we ensure the generation of feasible and optimized paths. The synergy approach effectively combines the discrete feasibility of BFS, the exploration efficiency of RRT, the diversity introduced by

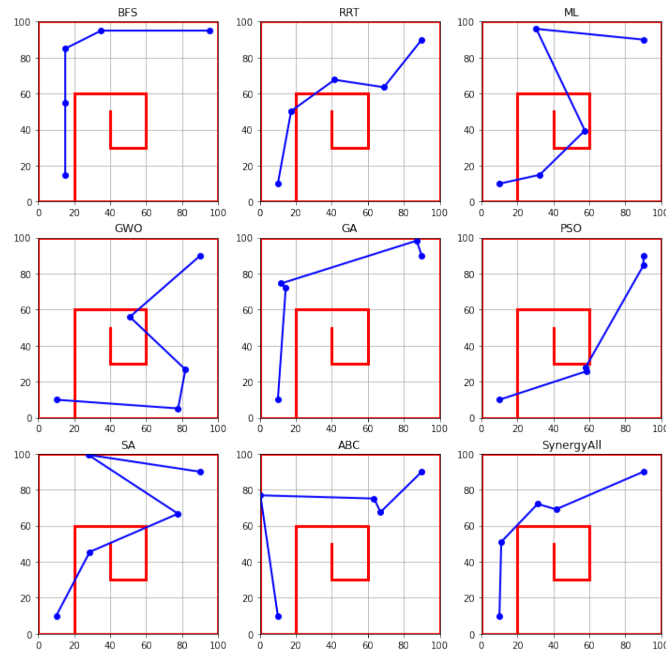

Fig. 4: Path smoothness evaluated through curvature penalties.

MLPlanner, and the optimization prowess of meta-heuristics. Simulation results across three labyrinths demonstrate that SynergyAll consistently produces feasible, collision-free trajectories with lower costs compared to individual planners. Future work will explore higher-resolution grids, advanced machine learning models, and adaptation to dynamic or higher-dimensional environments.

Acknowledgment

The authors would like to thank Sabanci University for supporting this research.

References

[1] 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, Apr. 1968.

[2] S. M. LaValle, "Rapidly-exploring Random Trees: A New Tool for Path Planning," *Technical Report*, Computer Science Department, Iowa State University, Ames, IA, 1998.

[3] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, Nov. 2013.

[4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA, USA: Addison-Wesley, 1989.

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. IEEE Int. Conf. Neural Networks*, Perth, WA, Australia, Dec. 1995, pp. 1942–1948.

[6] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, Jun. 2014.

[7] E.-G. Talbi, *Metaheuristics: From Design to Implementation*. Chichester, UK: Wiley, 2009.