ME312 Project Report

PA 6: Path Planning and Obstacle Avoidance for Fruit Harvesting Robot Using State-of-the-Art Metaheuristic Algorithms

Instructor: Dr. Jyotindra Narayan

Team Members:

- Anirudh Singh (2201ME11)
- Asmit Singh (2201ME20)
- Aryan Kaushal (2201ME16)
- Ashutosh Tiwari (2201ME18)
- Harsh Kumar (2201ME30)
- Harsh Mittal (2201ME31)
- Pushpendra Singh (2201ME49)
- Yuvansh Sharma (2201ME77)

Introduction:

This project explores the development of an intelligent path planning and obstacle avoidance system for a fruit-harvesting robot, leveraging both classical and modern optimization techniques. The overarching aim was to enable the robot to collect fruits while avoiding obstacles in an orchard environment, using algorithms that balance optimal pathfinding with the need for efficiency and adaptability.

Approach:

The simulation environment modeled the orchard as a 10×10 or 100×100 grid, where fruits and obstacles were randomly distributed. The robot always started from a fixed corner and had to make navigation decisions dynamically based on the selected algorithm. A comparative study was carried out to assess algorithm performance in terms of path length, number of fruits collected, number of iterations, and computational efficiency.

Path Planning Algorithms:

- A path planning algorithm is a computational method used to determine an optimal or feasible path for a robot to move from a starting position to a target destination while avoiding obstacles.
- The goal is to ensure efficiency in terms of distance, time, energy consumption, and safety. These algorithms are widely used in autonomous robots, self-driving cars, UAVs, and industrial automation
- There's three major sets of path-planning algorithms:
    - Classical Algorithms
    - Meta-heuristic Algorithms
    - Machine Learning Algorithms

One of each type of algorithms was to be studied.

A comprehensive evaluation of the A* search algorithm, Ant Colony Optimization (ACO), and Reinforcement Learning (RL) was undertaken, each offering unique strengths suited to different aspects of robotic path planning.

Classical Algorithms: A* Algorithm

The A* algorithm, a classical approach in the domain of graph traversal and search, was the foundational technique implemented. A* is considered both complete and optimal when using an admissible heuristic. It combines the merits of Dijkstra's algorithm and Greedy Best-First Search. Its cost function, $f(n) = g(n) + h(n)$, evaluates nodes based on the actual cost from the start node ($g(n)$) and an estimated cost to the goal node ($h(n)$). A* has found widespread application in robotics, games, and navigation systems due to its effectiveness and efficiency. In this project, multiple heuristic functions were tested, such as

Manhattan: $h(n)=|x_1-x_2|+|y_1-y_2|$

Euclidean: $h(n)=((x_1-x_2)^2+(y_1-y_2)^2)^{1/2}$

Chebyshev: $h(n)=\max(|x_1-x_2|,|y_1-y_2|)$

Octile: $h(n)=D_1\cdot(dx+dy)+(D_2-2D_1)\cdot\min(dx,dy)$

where,

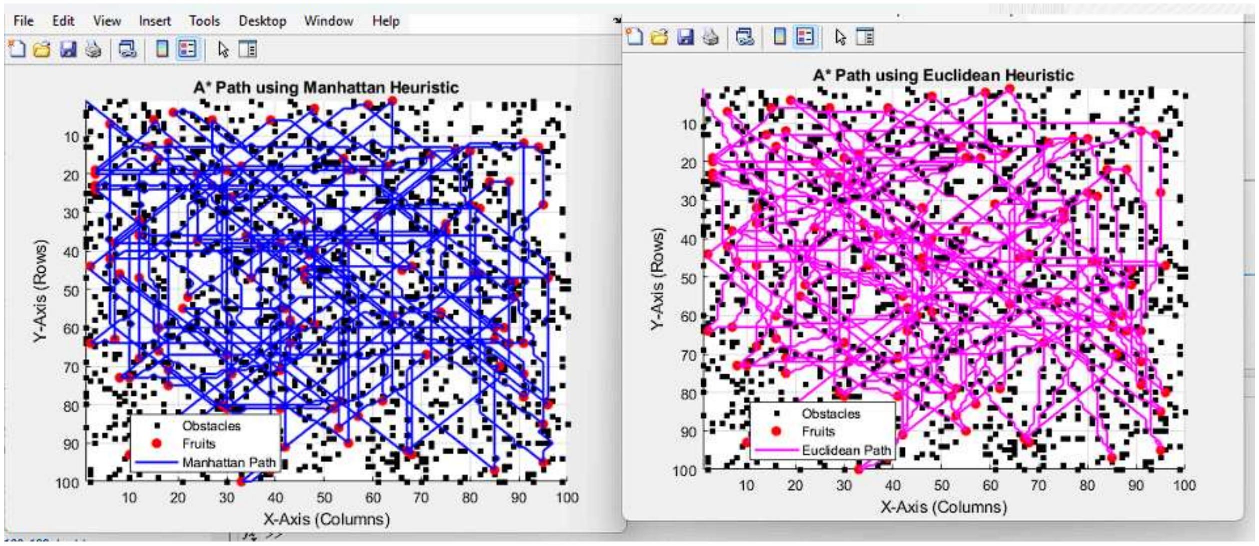
- $dx=|x_1-x_2|$, $dy=|y_1-y_2|$
- D_1 and D_2 are constants (usually $D_1=1$, $D_2=(2)^{1/2}$)

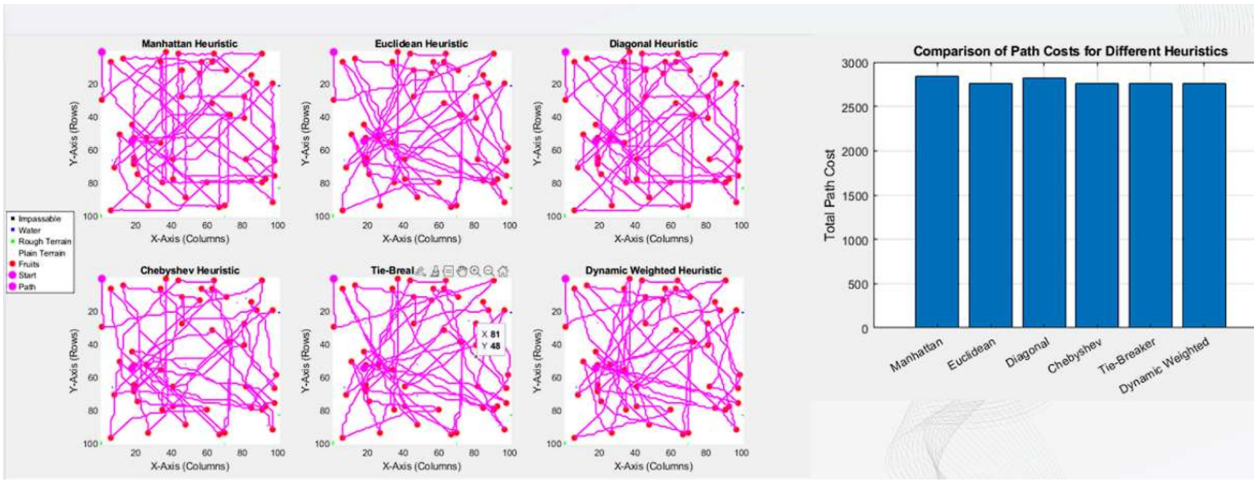
Weighted: $f(n)=g(n)+(1+\epsilon-d(n)N\epsilon)\cdot h(n)$

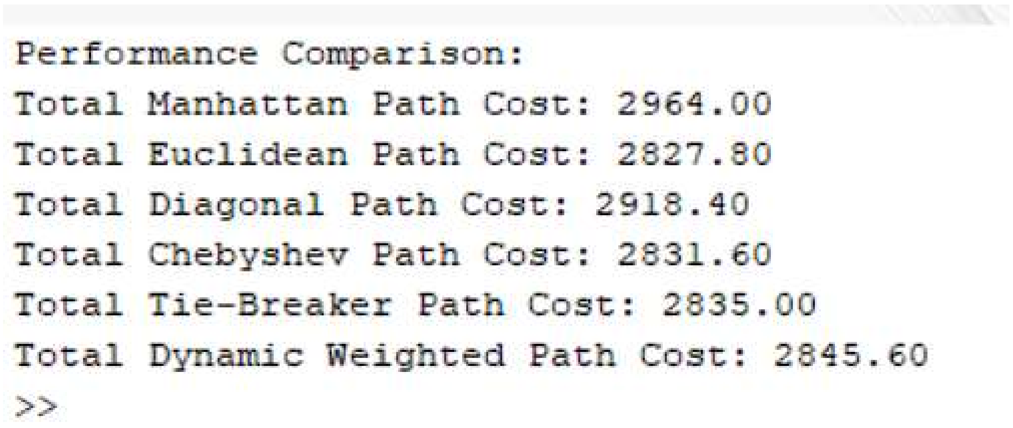
where,

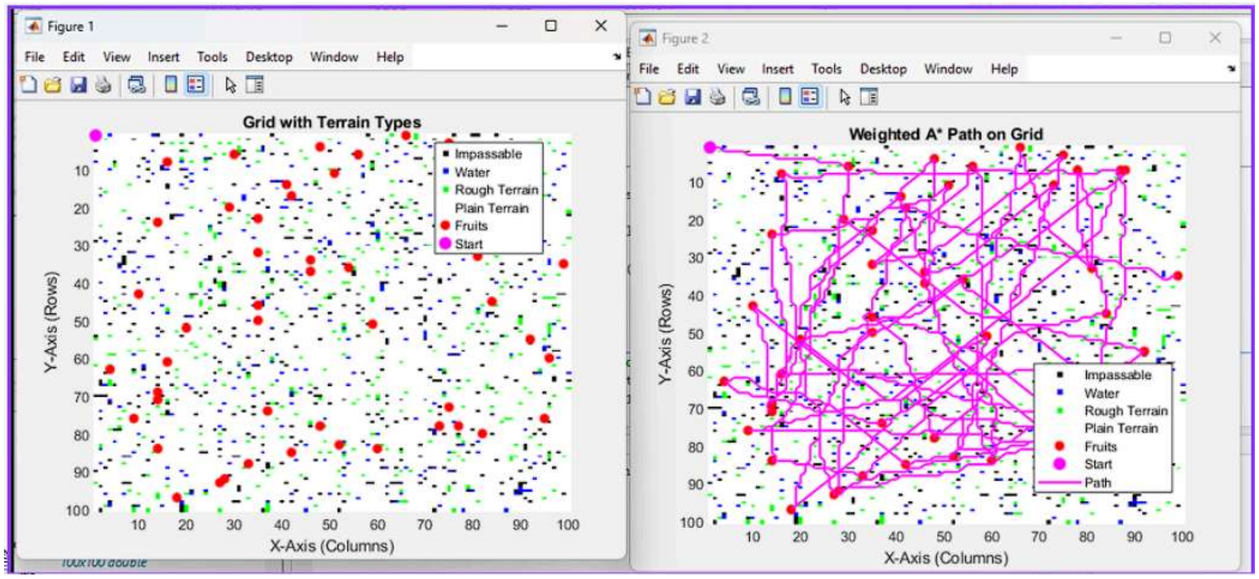
- ϵ: a small weight factor
- N: number of nodes
- $d(n)$: depth of the node in the search tree

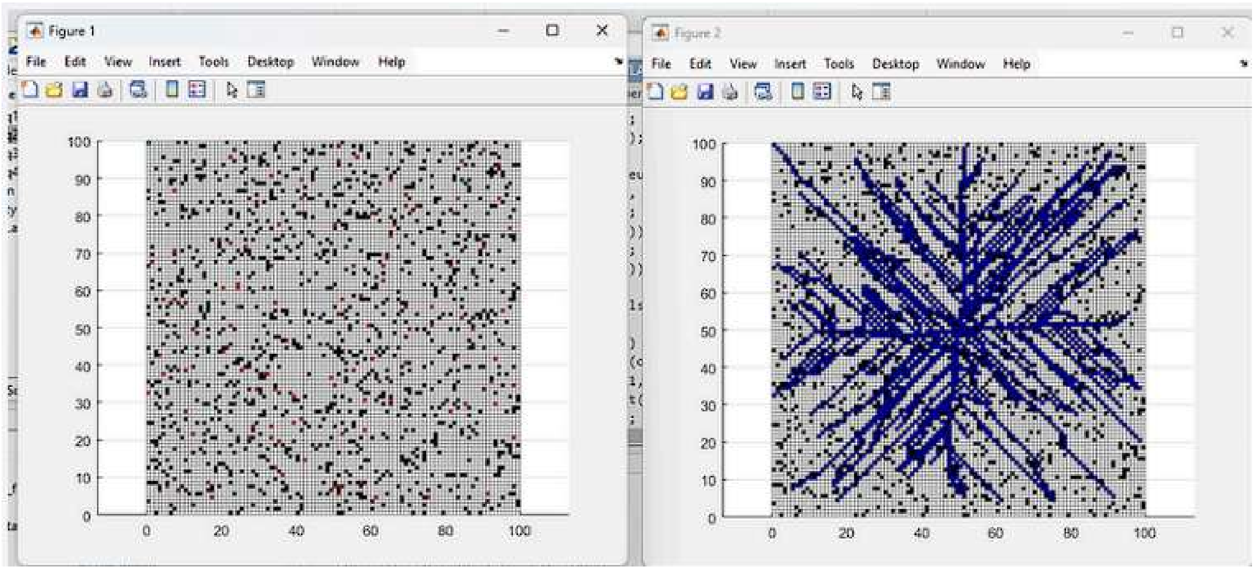
Each heuristic influenced how aggressively or conservatively the search progressed. For example, Euclidean distance often results in more direct paths, whereas Chebyshev is more suitable for environments where diagonal movement is permitted. Despite A*'s precision in calculating the shortest path, it falls short in multi-objective scenarios like fruit collection, where path optimality must be balanced with task performance

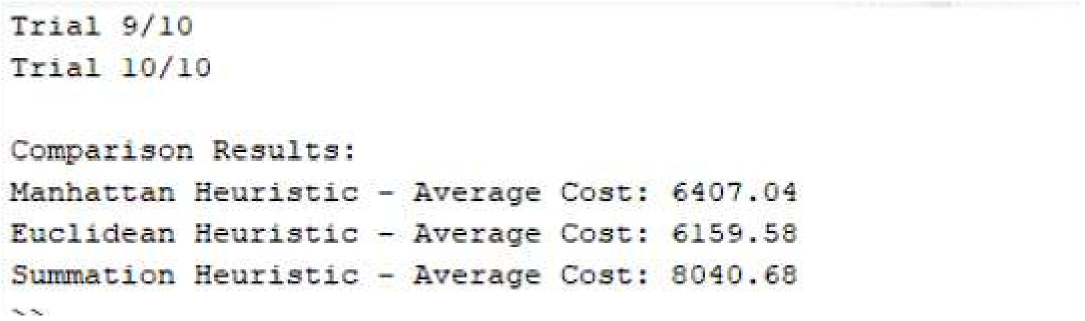

File Edit View Insert Tools Desktop Window Help
A* Path using Manhattan Heuristic
Y-Axis (Rows)
X-Axis (Columns)
Obstacles
Fruits
Manhattan Path
A* Path using Euclidean Heuristic
Euclidean Path



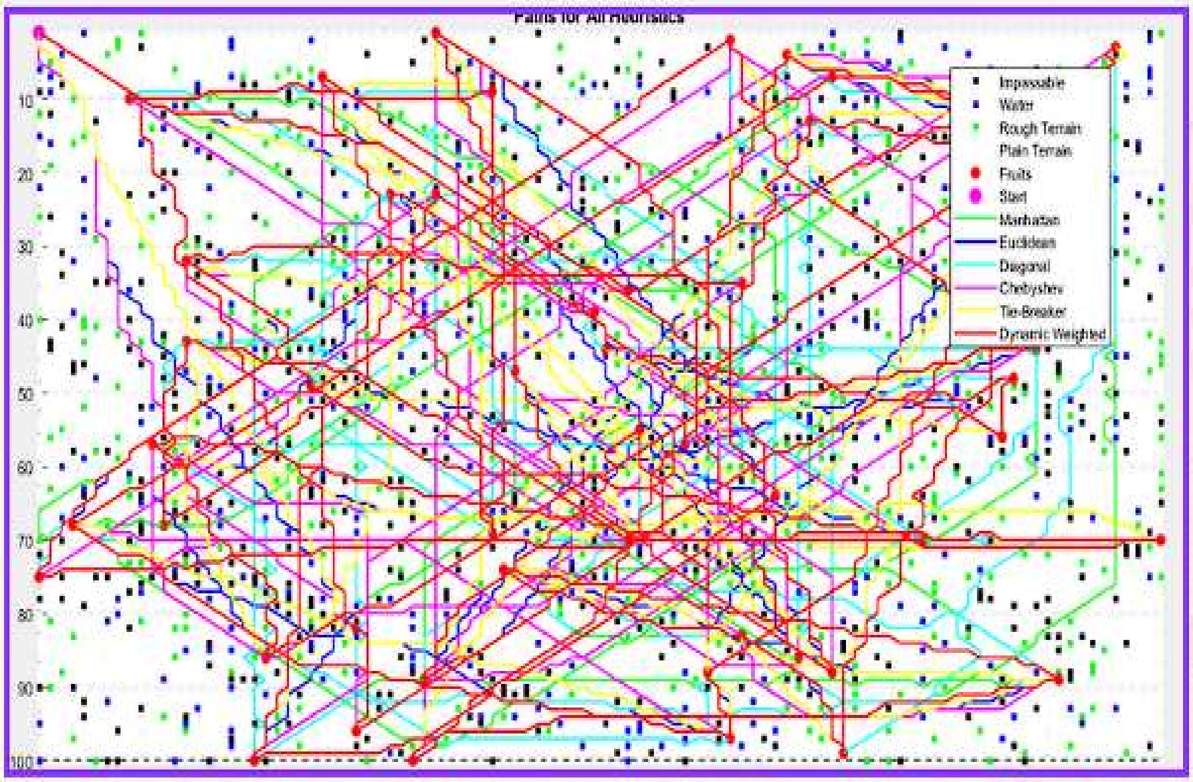

```
Performance Comparison:
Total Manhattan Path Cost: 2964.00
Total Euclidean Path Cost: 2827.80
Total Diagonal Path Cost: 2918.40
Total Chebyshev Path Cost: 2831.60
Total Tie-Breaker Path Cost: 2835.00
Total Dynamic Weighted Path Cost: 2845.60
>>
```

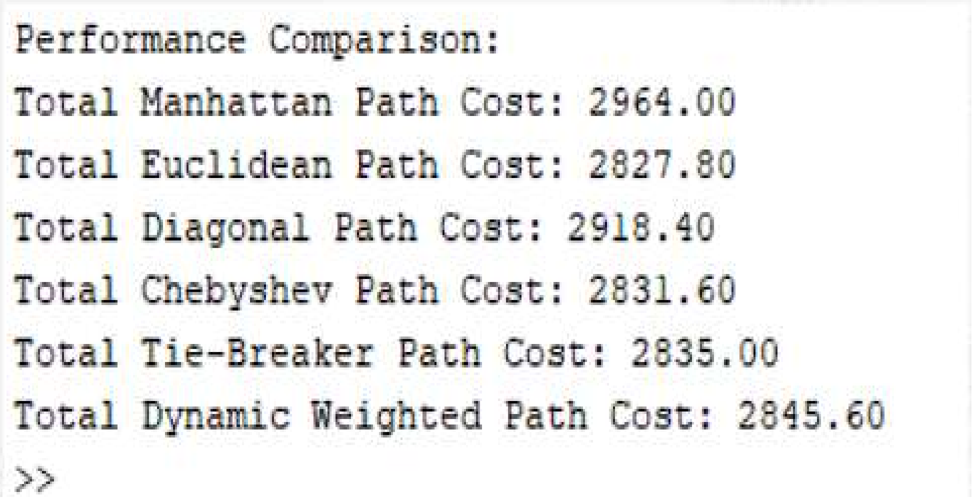






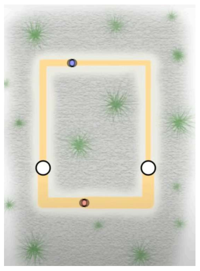
```
Trial 9/10
Trial 10/10

Comparison Results:
Manhattan Heuristic - Average Cost: 6407.04
Euclidean Heuristic - Average Cost: 6159.58
Summation Heuristic - Average Cost: 8040.68
>>
```




```
Performance Comparison:
Total Manhattan Path Cost: 2964.00
Total Euclidean Path Cost: 2827.80
Total Diagonal Path Cost: 2918.40
Total Chebyshev Path Cost: 2831.60
Total Tie-Breaker Path Cost: 2835.00
Total Dynamic Weighted Path Cost: 2845.60
>>
```

Meta-heuristic Algorithms: Ant Colony Algorithm

To overcome this limitation, Ant Colony Optimization (ACO) was explored. ACO is a probabilistic technique for solving computational problems that can be reduced to finding good paths through graphs. Inspired by the foraging behavior of real ants, ACO uses a swarm of agents (ants) that communicate indirectly through pheromone trails.

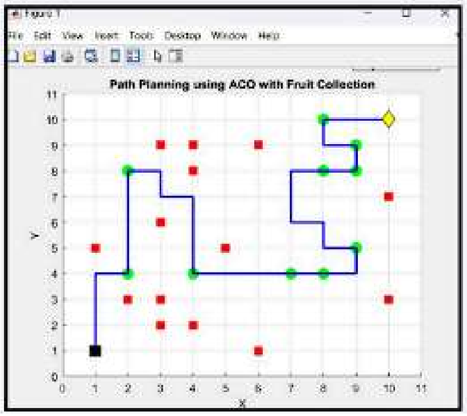


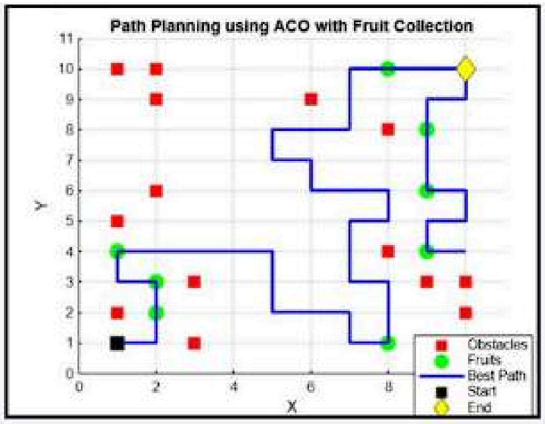
The essence of the algorithm is very simple. Ants begin to move along all the routes, delivering food to the ant hill. Along the way they leave an odorous trail of pheromones. On a shorter path, ants will go back and forth more times over the same time period. And as a result they will leave more pheromones per unit length of the path. This path will smell stronger and attract more ants. Over time, all the ants will switch to stronger smelling shortcuts and the long oath will be abandoned as pheromones evaporate.

Variants of the ACO algorithm have introduced mechanisms such as elitism, rank-based pheromone deposition, and pheromone evaporation near obstacles to prevent premature convergence and improve exploration. ACO is particularly effective in complex, dynamic environments like orchards, where the landscape is irregular and the target (fruits) is dispersed. In this project, enhancements such as pheromone boosting for efficient paths and decay near obstacles were implemented to improve both convergence speed and path quality.

$$h(n) = \sqrt{[(x_1 - x_2)^2 + (y_1 - y_2)^2]}$$

Eucledian





Manhattan

$$h(n) = (|x_1 - x_2| + |y_1 - y_2|)$$

1. Euclidean Distance (h(n))

- **$h(n) = \sqrt{[(x_1 - x_2)^2 + (y_1 - y_2)^2]}$**
- **Characteristics:**
    1. Measures straight-line ("as-the-crow-flies") distance
    2. Most accurate for unobstructed environments
    3. Admissible heuristic for A* algorithm

- **Best for:**
    1. Open terrain navigation
    2. Drone/aircraft path planning
    3. Environments without movement constraints

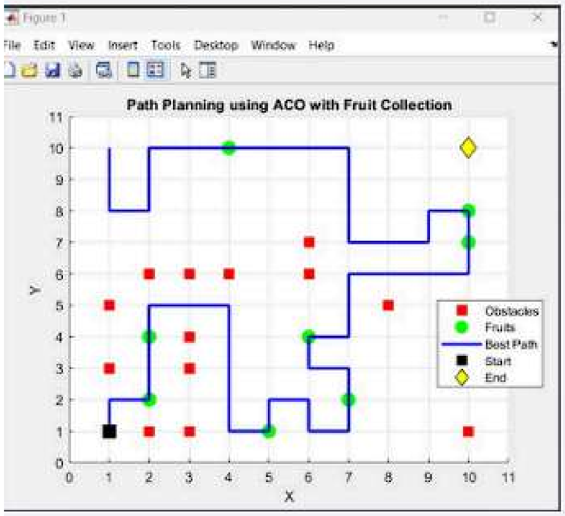
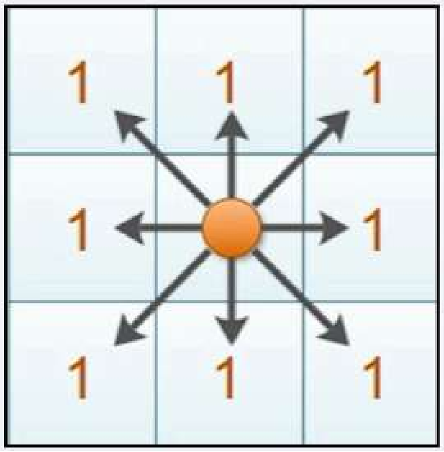


Figure 1
File Edit View Insert Tools Desktop Window Help
Path Planning using ACO with Fruit Collection
Y
X
■ Obstacles
● Fruits
Best Path
■ Start
◇ End

Chebyshev

$$h(n) = \max(|x_1 - x_2|, |y_1 - y_2|)$$



2. Manhattan Distance (h(n))

$$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

Characteristics:

1. Measures grid-based movement distance
2. Matches movement constraints in urban environments
3. Computationally simpler than Euclidean

Best for:

1. City navigation (hence "Manhattan" metric)
2. Grid-based movement systems
3. Robotic vacuum cleaners/warehouse robots

3. Chebyshev Distance (h(n))

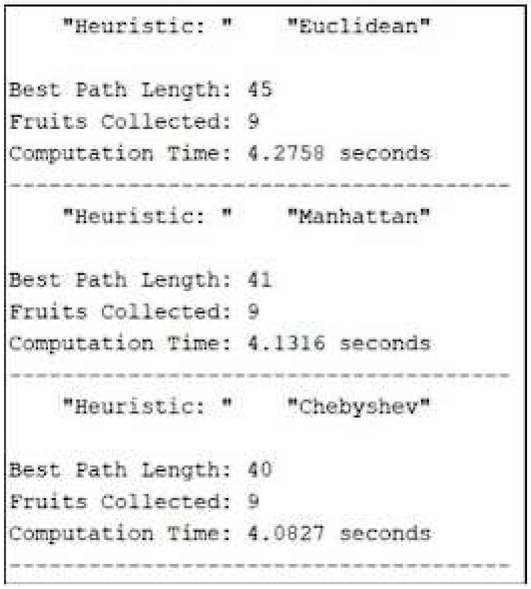
$h(n) = \max(|x_1 - x_2|, |y_1 - y_2|)$

Characteristics:

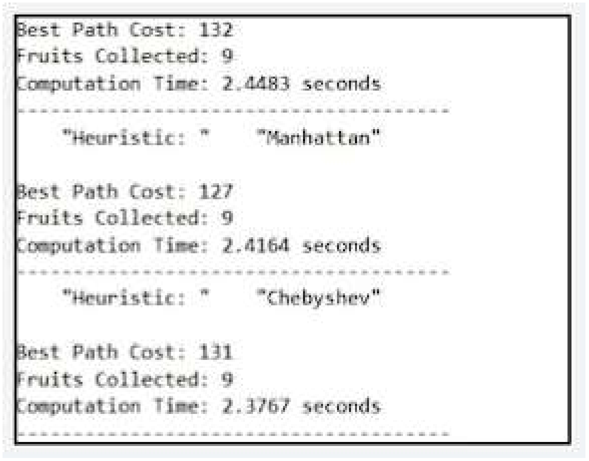
1. Measures distance for 8-directional movement
2. Useful for diagonal movement at no extra cost
3. Often used in chess (king's movement)

Best for:

1. Games with diagonal movement
2. Omnidirectional robots
3. Some drone navigation scenarios



```
    "Heuristic: "    "Euclidean"

Best Path Length: 45
Fruits Collected: 9
Computation Time: 4.2758 seconds
----------------------------------------
    "Heuristic: "    "Manhattan"

Best Path Length: 41
Fruits Collected: 9
Computation Time: 4.1316 seconds
----------------------------------------
    "Heuristic: "    "Chebyshev"

Best Path Length: 40
Fruits Collected: 9
Computation Time: 4.0827 seconds
----------------------------------------
```



```
Best Path Cost: 132
Fruits Collected: 9
Computation Time: 2.4483 seconds
----------------------------------------
    "Heuristic: "    "Manhattan"

Best Path Cost: 127
Fruits Collected: 9
Computation Time: 2.4164 seconds
----------------------------------------
    "Heuristic: "    "Chebyshev"

Best Path Cost: 131
Fruits Collected: 9
Computation Time: 2.3767 seconds
----------------------------------------
```

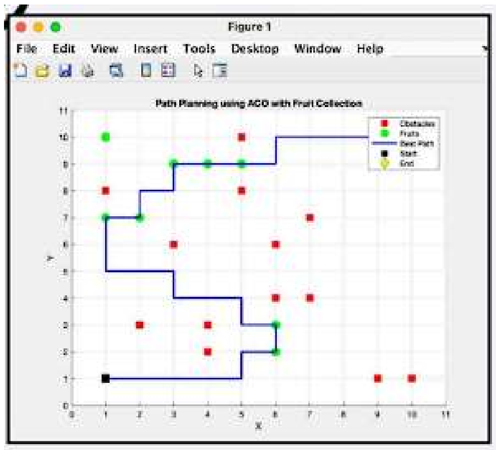


Figure 1
File Edit View Insert Tools Desktop Window Help
Path Planning using ACO with Fruit Collection
Obstacles
Fruits
Best Path
Start
End
X
Y

Pheromone decay near obstacles

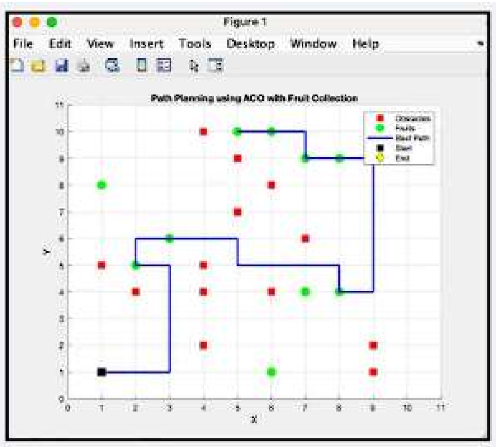


Figure 1
File Edit View Insert Tools Desktop Window Help
Path Planning using ACO with Fruit Collection
Obstacles
Fruits
Best Path
Start
End
X
Y

Pheromone booster for shorter paths

Pheromone decay:

- This approach simulates natural pheromone evaporation near obstacles.
- As ants explore paths, pheromones decay more rapidly around impassable areas (red squares), discouraging repeated selection of inefficient or blocked routes.
- This leads to safer and more feasible path formation, avoiding dense obstacle regions.

Pheromone booster:

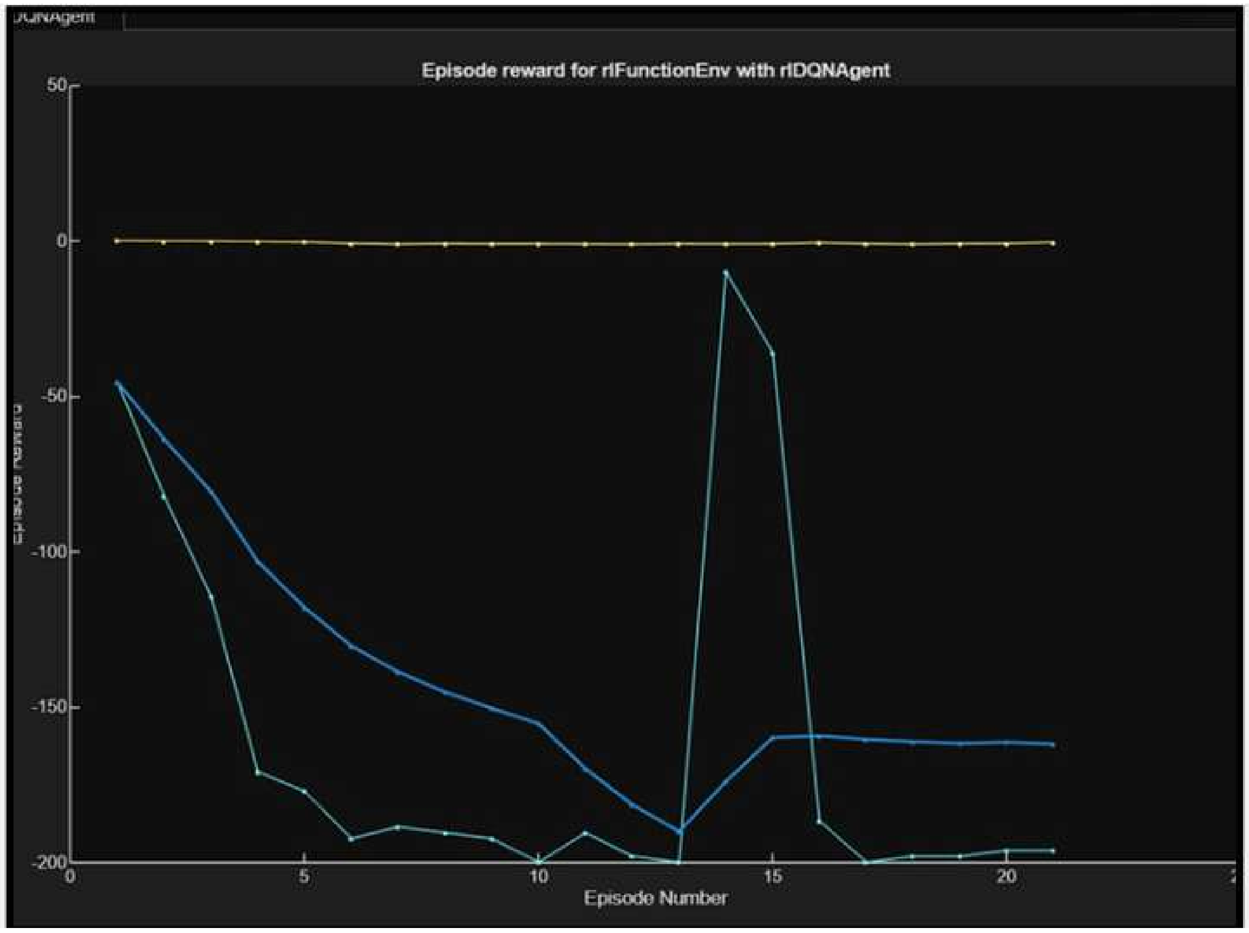
- This enhancement rewards shorter, efficient paths by reinforcing them with stronger pheromone trails.
- Ants are more likely to choose optimized routes that not only reach the destination but also collect key resources (green dots, representing fruits).
- Over time, the collective behavior converges to optimal or near-optimal paths.

Machine Learning Algorithms: Reinforcement Learning

Additionally, a learning-based approach using Reinforcement Learning (RL) was implemented to add a layer of intelligence to the robot's decision-making process. RL is a paradigm in machine learning where agents learn to take actions by interacting with an environment to maximize cumulative reward. The robot was trained using a Deep Q-Network (DQN), a neural network-based model that approximates the Q-value function used in Q-learning. The environment was modeled to include states such as the robot's position, fruit and obstacle locations, and possible actions like movement, turning, and picking. The reward structure was carefully crafted to encourage fruit collection and penalize collisions or inefficient paths. Training curves indicated that while the agent performed poorly in early episodes, significant learning occurred around episode 13, followed by gradual stabilization. Such trends are typical in RL environments, where exploration-exploitation trade-offs lead to initial instability but eventual policy convergence.

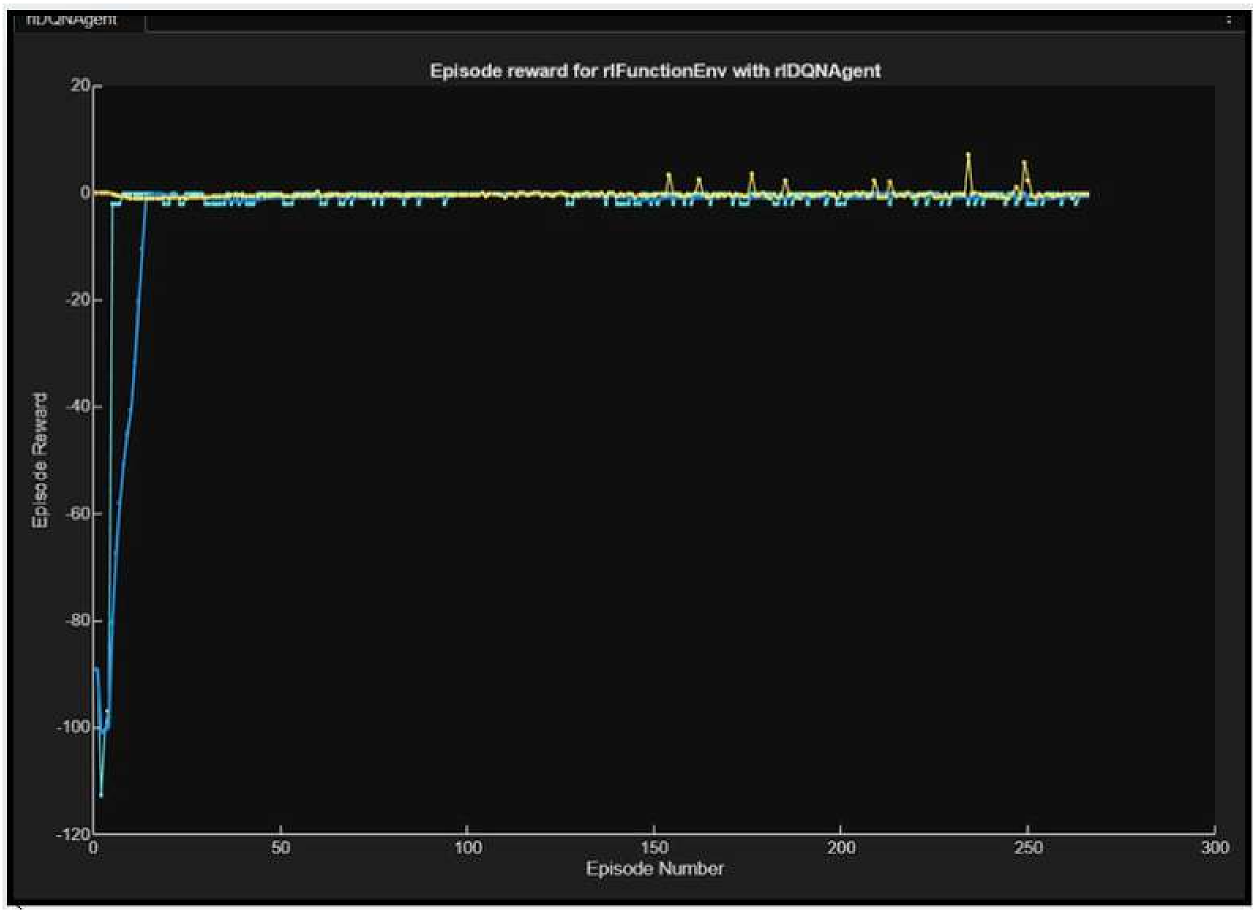
A* excelled in minimizing path length but lacked adaptability. ACO provided robust results under varying conditions, showing superior performance in multi-objective scenarios. RL, while computationally demanding and slower to converge, demonstrated potential for generalization and adaptability in evolving environments.

This study underscores the trade-offs inherent in different path planning strategies. Classical methods like A* are reliable and deterministic but limited to single-objective optimization. Metaheuristic algorithms such as ACO offer flexibility and robustness at the cost of higher computational complexity. Reinforcement Learning introduces an adaptive, self-learning framework that is best suited for dynamic and partially observable environments but demands significant training effort and data.



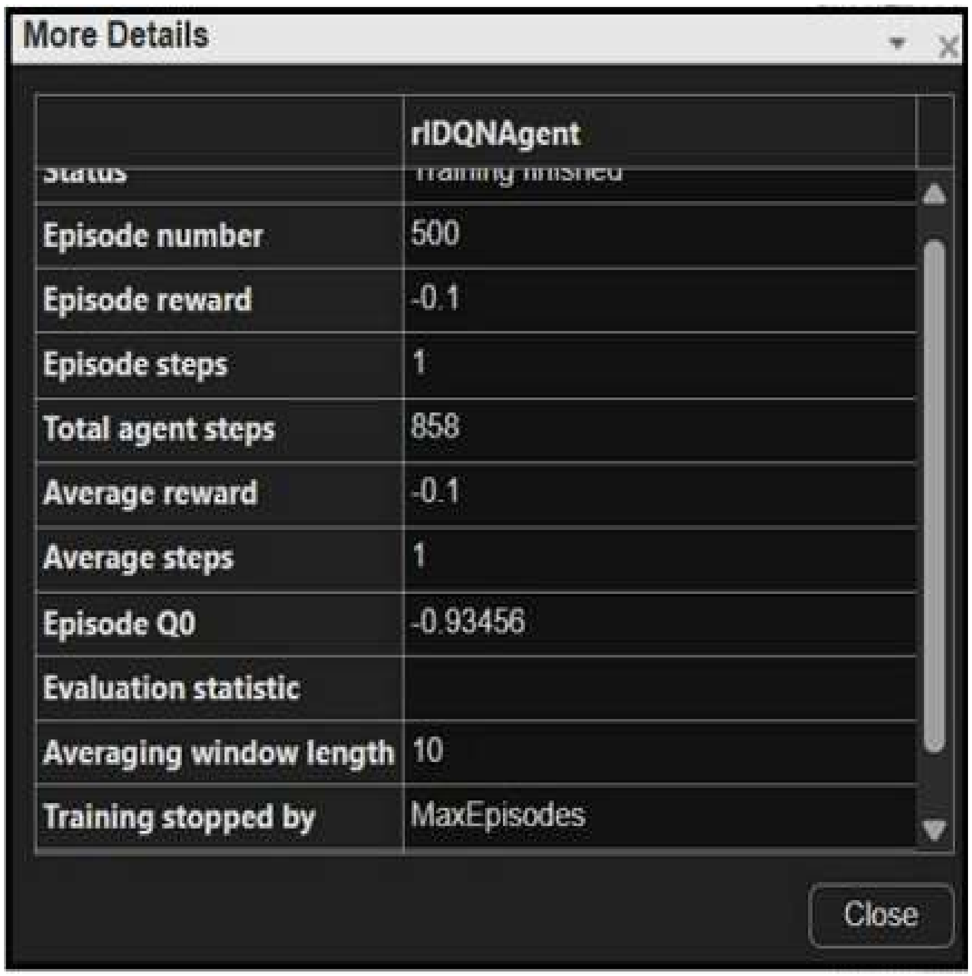
The core elements of an RL system are:

- **Agent:** The learning entity (in this case, the fruit harvesting robot).
- **Environment:** The world with which the agent interacts (the orchard).
- **State:** A representation of the agent's current situation in the environment (e.g., its position, sensor readings).
- **Action:** A choice made by



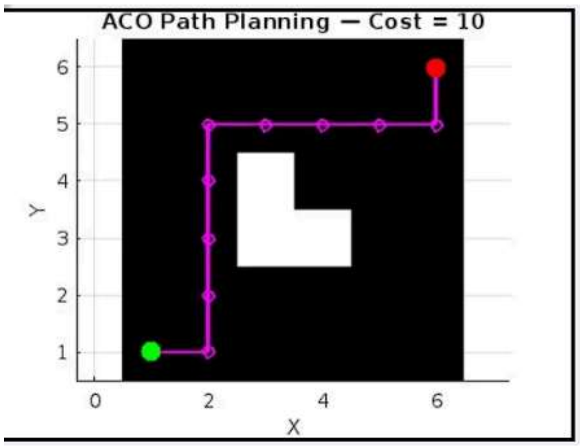
- This section evaluates the performance of the Reinforcement Learning (RL) agent (rlDQNAgent) in the rlFunctionEnv environment by analyzing the episode rewards throughout training. The goal is to assess learning progress and identify potential areas for improvement.

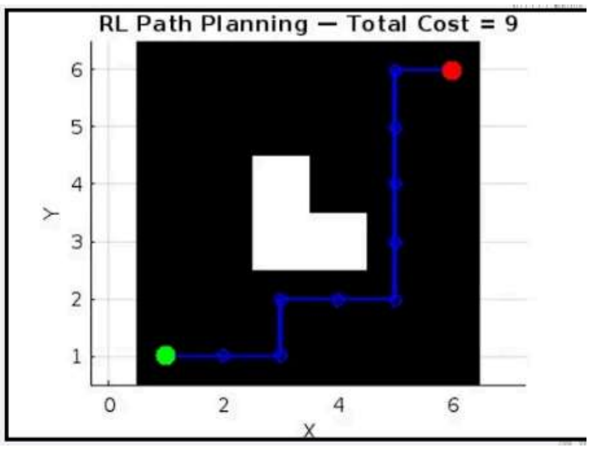
- Eventually, the episode reward converges to the baseline for a large value of episode number.



	rlDQNAgent
Status	Training finished
Episode number	500
Episode reward	-0.1
Episode steps	1
Total agent steps	858
Average reward	-0.1
Average steps	1
Episode Q0	-0.93456
Evaluation statistic	
Averaging window length	10
Training stopped by	MaxEpisodes

- This is a small negative reward applied every step to discourage wandering aimlessly = -0.2
- Hitting a wall or stepping into an obstacle results in a larger penalty: -2.
- If the agent steps on a fruit, it gets a +10 reward, and that fruit disappears.

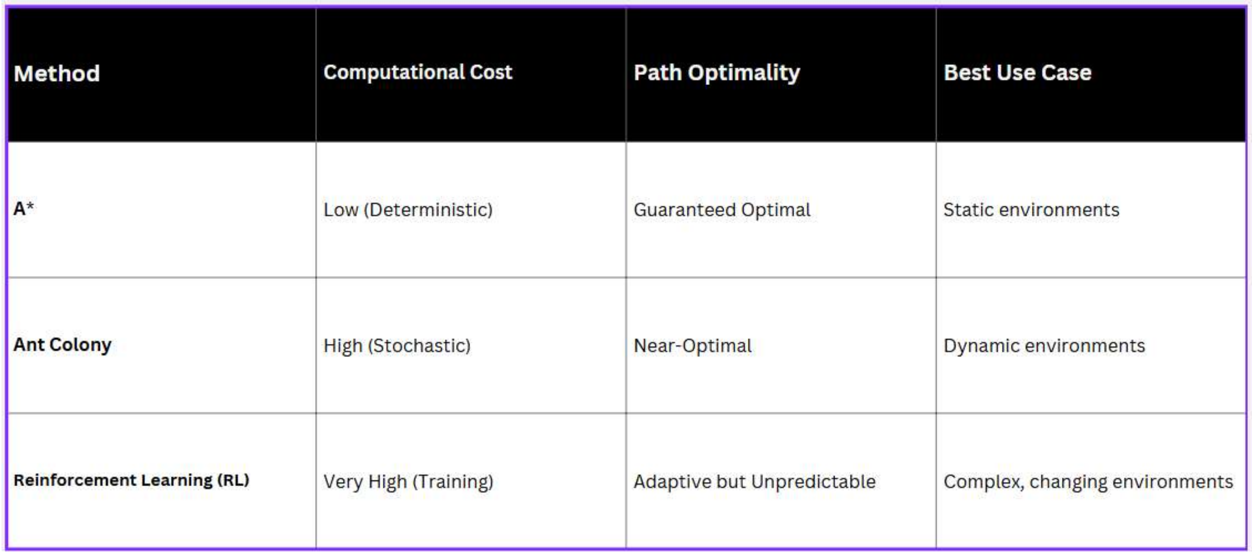




- The **Reinforcement Learnin**g approach achieved a 10% lower path cost (9 vs. 10) compared to Ant Colony Optimization, indicating:

1. More efficient route selection
2. Better optimization of movement constraints
3. Improved decision-making in the given environment

- This comparison highlights RL's potential as a superior path planning solution in the tested environment, while confirming ACO remains a viable alternative. The 1-unit cost difference, though seemingly small, could represent significant efficiency gains in large-scale or real-time applications.



Method	Computational Cost	Path Optimality	Best Use Case
A*	Low (Deterministic)	Guaranteed Optimal	Static environments
Ant Colony	High (Stochastic)	Near-Optimal	Dynamic environments
Reinforcement Learning (RL)	Very High (Training)	Adaptive but Unpredictable	Complex, changing environments

- **Key Takeaways:**

1. A* is deterministic and optimal for pathfinding when the heuristic is admissible, but struggles in dynamic environments.
2. ACO is nature-inspired and robust for combinatorial problems (like TSP) but can be slow and memory-intensive.
3. RI Learning (RL) is adaptive and model-free, making it suitable for unknown environments, but it requires careful tuning and exploration strategies.

- **When to Use Which?**

1. Use A* if you need optimal paths in static environments (e.g., game AI, GPS navigation).
2. Use ACO for combinatorial optimization (e.g., vehicle routing, network optimization).
3. Use RI Learning (RL) for adaptive decision-making in uncertain environments (e.g., robotics, real-time strategy games).

References:

https://www.mdpi.com/2079-9292/11/22/3660

https://youtu.be/ySN5Wnu88nE?si=go76jlJFYaDh7TUV

https://youtu.be/QR3U1dgc5RE?si=9Vxl4NBhxLIhlQBT

https://www.sciencedirect.com/science/article/pii/S0377221722004945

https://youtu.be/783ZtAF4j5g?si=e65D7kHtGwwrNmcX

https://youtu.be/u7bQomllcJw?si=ejzzFbmXqY1ziC7B

https://youtu.be/DcYLT37ImBY?si=_i9YkfFHGeEdnkZP

https://www.sciencedirect.com/science/article/abs/pii/S016816992100367