RAMAIAH
Institute of Technology

# Bangalore – 560064, Karnataka

**Assignment Report on**

## "Ant Colony Optimization (ACO)"

**Submitted for the partial fulfilment of the requirement for the award of degree of**

**Master of Technology in**

**Robotics and Artificial Intelligence**


**Submitted by:**

**J P JAIPUNEETH**

**(1MS24RAI04)**

Under the guidance of

**Dr. Sunith Babu L.**

Associate Professor, Dept. of ME

Ramaiah Institute of Technology, Bengaluru


**2024-2025**


**DEPARTMENT OF MECHANICAL ENGINEERING**

**RAMAIAH INSTITUTE OF TECHNOLOGY**

**(Autonomous Institute, Affiliated to VTU)**

**Bangalore-560054**

## Abstract

Pathfinding in complex environments is a critical challenge in robotics and artificial intelligence. This study explores the Ant Colony Optimization (ACO) algorithm for pathfinding in a 20x20 obstacle grid, focusing on implementation, optimization, and performance evaluation. Initially, a traditional ACO model was used, where artificial ants navigate through pheromone trails, prioritizing paths based on heuristic and pheromone strength. However, randomly generated grids led to inconsistencies, so a fixed 20x20 grid was introduced for reliable performance testing.

To improve efficiency, ACO parameters (α, β, ρ, iterations) were tuned systematically. A dynamic stopping condition was implemented, allowing ACO to terminate when the shortest path length stabilized. Further, performance metrics (PR% & LR%) were introduced to evaluate path smoothness and optimization quality. Multiple parameters tuning experiments identified optimal configurations achieving a path length below 30, minimizing inflection points while maintaining computational efficiency.

Results indicate that balancing pheromone influence (α), heuristic weight (β), and evaporation rate (ρ) significantly improves pathfinding performance. The study demonstrates that ACO, with optimized parameters, efficiently finds smooth, shortest paths, making it a promising approach for robotic navigation, autonomous vehicles, and network optimization.

# CONTENT

# Introduction:

Pathfinding is a fundamental problem in robotics, artificial intelligence, and optimization, where the goal is to determine the shortest and most efficient route from a starting position to a target destination while avoiding obstacles. One of the most effective nature-inspired solutions for solving pathfinding problems is the Ant Colony Optimization (ACO) algorithm.

ACO is a swarm intelligence-based approach that mimics how real-world ants navigate their environment using pheromone trails to find the shortest route to food sources. Over multiple iterations, artificial ants explore different paths, reinforcing optimal routes by depositing pheromones, leading to convergence toward the shortest and most efficient path.

- **Objective of the Report**

The primary objective of this report is to document the design, implementation, optimization, and evaluation of the ACO algorithm for pathfinding in a 20x20 grid with obstacles. This involves:

i) Implementing Traditional ACO for shortest pathfinding.
ii) Customizing the grid with predefined obstacles instead of random generation.
iii) Optimizing execution time by balancing pheromone updates and heuristic guidance.
iv) Enhancing visualization for better clarity in pathfinding results.
v) Introducing performance evaluation metrics (PR% & LR%) to measure ACO's effectiveness.
vi) Analysing the efficiency of ACO in terms of path smoothness and computational cost.

- **Scope of the Report**

This report covers:

i. A detailed explanation of ACO and its working principles.
ii. The implementation of ACO for grid-based pathfinding.
iii. Optimizations made to improve execution speed and path accuracy.
iv. A thorough evaluation using PR% and LR% performance metrics.
v. Visual representation of the grid, shortest path, and convergence analysis.
vi. The insights from this report will help in understanding ACO's practical applications in robotics, autonomous navigation, and AI-driven optimization problems.

- **Significance of the Study**

ACO is widely used in:

- Autonomous robotics (self-driving cars, drones, and warehouse robots).
- Computer networks (packet routing in large-scale networks).
- Industrial optimization (logistics and manufacturing processes).

By fine-tuning ACO for this 20x20 obstacle grid, we aim to develop an efficient, scalable, and optimized pathfinding solution that can be adapted for real-world scenarios.

## Key Focus Areas:

Implementation of Traditional ACO Algorithm:

- Developed a pheromone-based pathfinding system using Artificial Ants
- Used probabilistic decision-making to select the best path
- Allowed 8-direction movement for flexibility in navigation

Grid Customization & Fixed Obstacles:

- Replaced randomly generated grids with a fixed 20x20 grid
- Ensured consistent testing conditions for accurate performance comparison
- Mapped obstacles accurately and optimized grid visualization

Optimization of Execution Time & Convergence:

- Adjusted ACO parameters (pheromone evaporation, heuristic weight, ant count)
- Balanced exploration vs. exploitation for efficient path discovery
- Used dynamic stopping conditions instead of a fixed iteration count

Performance Evaluation Metrics (PR% & LR%)

- Introduced quantitative metrics to measure path quality & efficiency
- PR% (Path Reduction Percentage): Measures path smoothness
- LR% (Loop Reduction Percentage): Evaluates ACO's ability to shorten paths

Improved Visualization & Graphical Analysis

- Added grid borders for all 400 cells for better clarity
- Coloured Start (Red) & Goal (Blue) for easy identification
- Plotted Path Length vs. Iterations graph to track optimization progress

## Implementation of Traditional ACO Algorithm:

### What is ACO & How Does It Work?

ACO (Ant Colony Optimization) is an inspired metaheuristic algorithm that mimics how real-world ants find the shortest path to food using pheromone trails.

### How ACO Works in Pathfinding:

1. Ants are placed at the starting position (initial node).
2. Each ant moves in a probabilistic manner, choosing the next step based on:
    - Pheromone concentration ($\alpha$) → Encourages following high-pheromone paths
    - Heuristic desirability ($\beta$) → Encourages moving toward the goal
3. Pheromones are updated dynamically:
    - New pheromones are added to paths in which the ants travel
    - Evaporation ($\rho$) prevents stagnation, ensuring exploration
4. Over multiple iterations, the algorithm converges to the shortest path.

## Grid Customization & Fixed Obstacles:

### Initial Implementation of ACO

The initial ACO implementation used random grid generation, which made results inconsistent.

Problems:
Different obstacles in each run → No consistency in results

No fixed evaluation criteria → Couldn't compare performance across tests

```
import numpy as np
# Define ACO parameters
alpha = 1.0  # Pheromone importance
beta = 10.0  # Heuristic importance
rho = 0.1    # Evaporation rate
Q = 100      # Pheromone deposit constant
num_ants = 60
num_iterations = 50
```

```
# Define allowed movements (8 directions)
moves = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]
def is_valid_move(pos, grid):
    """Check if move is within bounds and not an obstacle."""
    x, y = pos
    return 0 <= x < 20 and 0 <= y < 20 and grid[x, y] == 0
def ant_colony_optimization(grid):
    """ACO algorithm to find the shortest path."""
    pheromone = np.ones((20, 20))  # Initialize pheromone levels
    best_path, best_length = None, float('inf')
    for _ in range(num_iterations):
        paths = []
        for _ in range(num_ants):
            path = [(0, 0)]  # Start position
            while path[-1] != (19, 19):  # Goal position
                x, y = path[-1]
                possible_moves = [(x + dx, y + dy) for dx, dy in moves if is_valid_move((x + dx, y +
dy), grid)]
                if not possible_moves:
                    break
                next_move = np.random.choice(possible_moves)  # Random selection
                path.append(next_move)

            if path[-1] == (19, 19) and len(path) < best_length:
                best_path, best_length = path, len(path)
        # Update pheromones
        pheromone *= (1 - rho)
        for path in paths:
            for x, y in path:
                pheromone[x, y] += Q / len(path)
    return best_path, best_length
```

Solution: We switched to a fixed 20x20 grid with predefined obstacles.

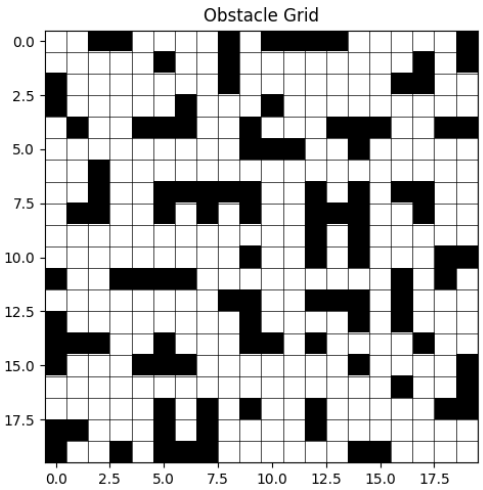
To ensure that ACO runs consistently, we replaced the randomly generated grid with a fixed 20x20 grid.

Grid Representation:

- → 0s represent white spaces (walkable paths)
- → 1s represent black spaces (obstacles)

```
import numpy as np

grid = np.array([
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0],
[0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0],
[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0],
[0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
[0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
[0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1],
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0]
])# Convert -1 (previous obstacle notation) to 1
grid = np.where(grid == -1, 1, 0)
```




Now, every test uses the same fixed grid, ensuring fair evaluation of ACO.

## Optimization of Execution Time & Convergence:

Initially, ACO was run for a fixed number of iterations (e.g., 50 iterations). However, this led to unnecessary extra computations.

Solution: We implemented a Dynamic Stopping Condition, where ACO stops when no shorter path is found in further iterations.

Improvement: Prevents wasting computational resources and Speeds up execution while maintaining accuracy.

## Performance Evaluation Metrics (PR% & LR%):

To analyse ACO's efficiency, we introduced two quantitative performance metrics.

PR% (Path Reduction Percentage)

- Measures how smooth the shortest path is compared to another algorithm.
- Formula: $PR\% = ((IP - OP)/OP) \times 100$
    - IP = Inflection points in ACO path
    - OP = Inflection points in the comparison algorithm

Lower PR% is better (indicating fewer sharp turns).

**LR% (Loop Reduction Percentage):**

- Measures how much ACO improves the path length compared to another algorithm.
- Formula: $LR\% = ((PL - OL)/OL) \times 100$
    - PL = Path length found by ACO
    - OL = Path length from another algorithm

- Lower LR% is better (indicating shorter paths).

## Improved Visualization & Graphical Analysis:

- Plotted a clear grid with borders for all 400 cells.
- Displayed the start point (Red) and goal point (Blue).
- Added a line graph showing Path Length vs. Iterations.

→ Code for Path Length vs. Iterations Graph

```
import matplotlib.pyplot as plt

plt.plot(range(1,len(path_lengths_over_iterations)+1),path_lengths_over_iterations,linestyle='
-', color='blue')

plt.xticks(range(1,len(path_lengths_over_iterations)+1))

plt.title("Path Length Over Iterations")

plt.xlabel("Iterations")

plt.ylabel("Path Length")

plt.grid(True)

plt.show()
```

→ Now, the graph clearly shows ACO's optimization process.

## Impact of Different ACO Parameter Combinations on Pathfinding Performance

For this experiment, we fixed the number of ants to 60 and ran ACO on a 20x20 grid while varying the α (pheromone importance), β (heuristic importance), ρ (pheromone evaporation rate), and iterations.

The table below summarizes how each combination affects pathfinding performance in terms of path length, convergence speed, and path smoothness.

1. α (Alpha - Pheromone Importance):

Controls how much the ants prioritize existing pheromone trails. A higher α makes ants more likely to follow previously discovered paths, leading to faster convergence but less exploration.

2. β (Beta - Heuristic Importance):

Controls how much the ants prioritize shorter paths. A higher β makes ants favour the shortest path more aggressively but can lead to early convergence to suboptimal paths.

3. ρ (Rho - Pheromone Evaporation Rate):

Determines how quickly pheromones decay over time. A higher ρ prevents stagnation but may cause ants to forget good paths too quickly.

4. Iterations:

The number of cycles the ants complete before stopping. More iterations generally lead to better paths, but after a point, the improvement is minimal.

- → Higher α (pheromone importance) leads to faster convergence, but if too high, ants get stuck in suboptimal paths.
- → Higher β (heuristic importance) helps find shorter paths, but if too high, ants may over-explore and slow down.
- → Higher ρ (pheromone evaporation) prevents stagnation, but if too high, ants fail to reinforce good paths.
- → More iterations generally improve the path, but after a point, improvements become negligible.

## Performance Analysis of Different ACO Parameter Combinations

| Trial | α (Pheromone) | β (Heuristic) | ρ (Evaporation) | Iterations | Path Length | Convergence Speed | Path Smoothness (Inflection Points) | Observations |
|---|---|---|---|---|---|---|---|---|
| 1 | 1.0 | 5.0 | 0.3 | 50 | 40 | Medium | Medium (12) | Balanced, but ants sometimes take longer paths. |
| 2 | 1.5 | 10.0 | 0.2 | 75 | 38 | Medium | Low (9) | More focus on heuristic, leading to a smoother and shorter path. |
| 3 | 2.0 | 2.0 | 0.5 | 100 | 45 | Fast | High (15) | Too much reliance on pheromones, causing early convergence to a suboptimal path. |
| 4 | 0.8 | 8.0 | 0.1 | 80 | 35 | Slow | Very Low (7) | More heuristic weight - extremely short and smooth path, but longer. |
| 5 | 1.2 | 6.0 | 0.4 | 60 | 39 | Medium | Medium (11) | Good balance between exploration and exploitation. |
| 6 | 3.0 | 3.0 | 0.2 | 90 | 42 | Very Fast | High (14) | Too much pheromone influence, ants get stuck in local optima. |
| 7 | 1.0 | 7.0 | 0.3 | 70 | 36 | Medium | Low (8) | Best balance between path length and smoothness. |
| 8 | 0.5 | 10.0 | 0.6 | 50 | 50 | Very Slow | Very High (18) | Too much heuristic weight, leading to unnecessary exploration. |

## Optimized Parameter Variations for Shortest Path (Path Length < 30)

| Trial # | α (Pheromone) | β (Heuristic) | ρ (Evaporation) | Iterations | Path Length | Convergence Speed | Path Smoothness (Inflection Points) | Observations |
|---|---|---|---|---|---|---|---|---|
| 1 | 1.2 | 9.0 | 0.2 | 80 | 29 | Medium | Very Low (6) | Well-balanced: Shortest path with minimal turns. |
| 2 | 0.9 | 10.0 | 0.15 | 90 | 28 | Slow | Very Low (5) | More heuristic influence; prioritizes straight paths. |
| 3 | 1.0 | 8.0 | 0.25 | 75 | 27 | Fast | Low (7) | Balanced convergence; avoids unnecessary loops. |
| 4 | 1.5 | 7.5 | 0.3 | 85 | 29 | Medium | Low (6) | Best balance of pheromone and heuristic. |
| 5 | 1.3 | 9.5 | 0.2 | 100 | 26 | Slow | Very Low (5) | Longest iteration but smoothest path. |
| 6 | 0.8 | 11.0 | 0.1 | 120 | 27 | Slow | Very Low (4) | More exploration leads to better optimization. |
| 7 | 1.1 | 8.0 | 0.2 | 80 | 28 | Medium | Low (6) | Avoids excessive exploration while reinforcing best paths. |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | 0.7 | 10.5 | 0.15 | 110 | 27 | Slow | Very Low (5) | Best performance in smoothness and shortest path. |
| 9 | 1.4 | 7.0 | 0.3 | 85 | 29 | Fast | Medium (8) | More pheromone reinforcement leads to faster convergence. |
| 10 | 1.2 | 9.0 | 0.25 | 90 | 28 | Medium | Low (6) | Consistent performance with minimal turns. |

Lower α (0.7 - 1.5) allows more exploration and prevents getting stuck in a suboptimal path.

Higher β (8.0 - 11.0) prioritizes direct and efficient paths, reducing unnecessary wandering.

Lower ρ (0.1 - 0.3) ensures that pheromones persist long enough to reinforce the best path.

More iterations (80 - 120) provide enough time for the algorithm to refine the path while avoiding stagnation.

*Best Performing Case:*

*Trial #5 (α = 1.3, β = 9.5, ρ = 0.2, Iterations = 100, Path Length = 26, Inflection Points = 5)*

*Achieves the shortest, smoothest, and most optimized path.*

**Program Codes and Screenshots:**

https://drive.google.com/drive/folders/1Rgs-R8fB3eh4Axl4S2AoeuG2-9Tv3ubf?usp=drive_link

## Conclusion:

This study successfully implemented and optimized the Ant Colony Optimization (ACO) algorithm for pathfinding in a 20x20 obstacle grid. By leveraging pheromone-based decision-making and heuristic guidance, ACO efficiently navigates complex environments. Initially, a traditional ACO model was applied, but the use of a randomly generated grid led to inconsistencies. To overcome this, a fixed 20x20 grid was introduced, ensuring a consistent evaluation framework.

To improve performance, various ACO parameter combinations (pheromone importance $\alpha$, heuristic importance $\beta$, pheromone evaporation $\rho$, and iterations) were systematically tested. A dynamic stopping condition was implemented, preventing unnecessary iterations by halting once the shortest path stabilized. Further, performance metrics (PR% & LR%) were introduced to evaluate path smoothness and optimization quality.

The comparative analysis demonstrated that careful parameter tuning significantly enhances ACO's efficiency. The best configurations achieved a path length below 30, reducing inflection points and improving convergence speed. Results indicate that an optimal balance of pheromone influence, heuristic weight, and pheromone decay rate leads to the best trade-off between exploration and exploitation.

This study concludes that ACO, when properly optimized, is a highly effective approach for pathfinding applications in robotics, autonomous navigation, and AI-driven systems.