VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

REPORT

ADVANCED ALGORITHMS AND DATA STRUCTURES

SEMESTER 251 ACADEMIC YEAR 2025-2026

ANT COLONY OPTIMIZATION ALGORITHM AND ITS IMPROVEMENTS FOR SHORTEST PATH PROBLEM

MAJOR: COMPUTER SCIENCE

SUPERVISOR: Assoc. Prof. Lê Hồng Trang

——o0o——

STUDENT 1: Trần Hoàng Ân - 2570548

STUDENT 2: Nguyễn Huỳnh Như - 2570471

STUDENT 3: Đinh Hồng Thương - 2570503

STUDENT 4: Trần Huy Phước - 2570482

HO CHI MINH CITY, December 2025

Contents

Chapter 1

Introduction

Overview, objectives and structure of the report.

1.1 Motivation

This work investigates the shortest-path planning problem on occupancy-grid maps through the lens of Ant Colony Optimization (ACO). Concretely, given a static, fully-known grid with occupied and free cells, the task is to find a collision-free route from start to goal that minimizes Euclidean path length and yields trajectories that are smooth enough for a motion controller to follow.

The motivation for improving ACO on grid maps comes from practical applications such as mobile robot navigation, autonomous ground vehicles in structured environments, and automated guided vehicles in warehouses. In these settings planners are commonly evaluated offline on benchmark maps, so robustness, repeatability and geometric path quality are all important.

Classical planners address the same problem in different ways. Graph search methods such as A* or Dijkstra guarantee optimality on discrete grids and are easy to implement, but their solutions tend to follow grid edges and produce jagged paths whose discrete cost does not directly reflect Euclidean length. Sampling-based planners (RRT, PRM) operate naturally in continuous spaces and can generate smooth paths after post-processing, yet they can be inefficient on dense grid benchmarks and produce results that are harder to compare deterministically across repeated trials. Standard ACO brings a different

trade-off: it explores multiple candidate solutions in parallel and blends heuristic and pheromone guidance, but it is sensitive to parameter choices, can prematurely converge or stagnate, and typically yields discrete, noisy routes that require smoothing.

The improvements studied in this project—cone-shaped pheromone initialization, adaptive pheromone and heuristic scaling, division of labor among ants, targeted backtracking, and B-spline smoothing—are chosen because they address these specific weaknesses. Directional pheromone priors bias search toward promising corridors without overconstraining it; adaptive update rules shift the exploration–exploitation balance during a run to avoid stagnation; role specialization concentrates effort where it is most productive; backtracking reduces wasted time in dead-ends; and B-spline post-processing converts discrete paths into continuous, controller-friendly trajectories.

Together, these elements aim to accelerate convergence, improve geometric path quality, and produce executable trajectories—making the paper's method a natural fit for the project's benchmarking and experimental goals.

1.2 Goals

This project has several interrelated goals that guide the design and evaluation of the proposed ACO improvements. At the algorithmic level we seek faster and more robust convergence on grid benchmarks, measured both by time-to-best-solution and by reduced variance across repeated runs. Equally important is improving geometric path quality: rather than minimizing node count we aim to reduce Euclidean length and to report both average and best-of-run lengths over multiple trials.

From a practical perspective the work also targets trajectory smoothness and executability. Paths produced by the planner will be post-processed with B-spline smoothing so they are suitable for motion controllers, and we will quantify smoothness using curvature and continuity statistics. The algorithmic components themselves are designed to be complementary—pheromone initialization, adaptive update rules, role specialization and backtracking should work together rather than interfere.

Finally, experiments are intended to be reproducible and informative. We compare against baseline ACO and classical planners using standard statistical measures (mean, standard deviation, and success rate) across benchmark maps, and we report conver-

gence, path-quality, smoothness and robustness metrics for a complete picture of performance.

1.3 Scope

This report is deliberately focused: we study algorithmic refinements to ACO for static, fully-known occupancy-grid maps and evaluate them in an offline, repeatable benchmarking setting. The maps used are the project's local ‘maps/‘ collection together with a chosen set of public benchmarks to ensure that results can be reproduced and compared.

The work concentrates on planner-level contributions such as search guidance, pheromone and heuristic adaptation, role specialization among ants, targeted backtracking, and B-spline smoothing as a post-processing step. We do not attempt to solve state estimation, sensor noise modelling, online dynamic re-planning, kinodynamic trajectory optimization, or low-level controller integration; these topics are outside the scope of this report, though the produced trajectories are designed to be compatible with downstream controllers.

Key assumptions and limitations are as follows. Maps are treated as static and fully observed, so the findings do not directly generalize to dynamic or partially-observed environments. Performance comparisons are made via offline metrics and aggregated statistics; absolute wall-clock numbers will depend on implementation and hardware but relative trends are valid when measured on the same platform. Finally, smoothing is applied as a post-processing stage—detailed dynamics-aware trajectory optimization is left for future work.

1.4 Thesis Structure

The report is organised as follows:

- Chapter 1: Introduction, motivation, goals and scope.
- Chapter 2: Overview and Approach, high-level description of the implemented ACO and the set of proposed improvements.

- Chapter 3: Detailed Improvements, dataset/benchmark description and per-improvement: idea, implementation, benefits, and experimental results.
- Chapter 4: Conclusion, summary and future work.

Chapter 2

Overview and Approach

2.1 ACO overview

This section provides a concise overview of Ant Colony Optimization as applied to grid path planning:

- Ant Colony Optimization basics: pheromone trails, heuristic information, probabilistic transition rules, pheromone evaporation and deposit.
- Heuristic design for grid navigation: distance heuristics, admissibility considerations when combining with ACO.
- Convergence issues: premature convergence, stagnation, and common mitigation strategies.
- Path quality metrics: node-count vs. Euclidean (geometric) length and runtime/statistical metrics for evaluation.

2.2 Overview of our approach

We present a modular set of improvements designed to be composable:

1. Cone pheromone initialization, adds directional bias toward goal.
2. Adaptive pheromone/heuristic factors dynamically adjust α and β over iterations.

3. Division of labor and role assignment (explorers vs exploiters) with smooth transitions.

4. Backtracking heuristics allow ants to recover from dead-ends efficiently.

5. Smooth-line (B-spline) post-processing convert discrete waypoint lists into continuous trajectories.

6. Finetune coordinated parameters so all features work together.

A high-level algorithmic workflow is given, and detailed implementation notes are provided in Chapter 5.

Chapter 3

Detailed Improvements

Dataset / benchmark description and detailed description for each improvement.

3.1 Dataset / Benchmark maps

We evaluate on a set of benchmark maps provided in the project repository (see the `maps/` folder). The benchmark collection contains four ASCII grid maps (`map1.txt`, `map3.txt`, `map7.txt`, and `map8.txt`) of varying complexity levels, used throughout the experiments in this work. Each map encodes free space, obstacles, and the task endpoints in a compact, human-readable format.

Map Specifications

The benchmark suite includes maps with the following characteristics:

- **Map a (Simple):** 5×5 grid with basic obstacle configuration, suitable for validating algorithmic correctness and baseline performance.
- **Map b (Cluttered):** 31×31 grid with intricate obstacle patterns, narrow corridors, and multiple potential path candidates. This map is designed to test exploration-exploitation balance and robustness to local optima.
- **Map c (Crowded):** 25×20 grid with moderate obstacle density and mixed open/constrained regions, representing intermediate difficulty.

- **Map d (Maze-like):** 16×16 grid with clustered obstacles and multiple viable routes, testing the algorithm's ability to discover diverse solution paths.

Map Format and Parsing

- **Representation:** Each map is an ASCII grid where each cell contains one of the following markers: `S` (start), `F` (goal/finish), `E` (empty / free cell), and `O` (obstacle).
- **Grid interpretation:** The grid rows correspond to the map's y axis and columns to the x axis; cells are square and implicitly unit-spaced. Coordinates are taken at cell centers when converting to continuous space for distance calculations.
- **Preprocessing:** For each map we construct an occupancy grid and then build a node graph for planning. Nodes correspond to free cells; connectivity follows an 8-neighbour scheme (i.e., diagonal moves are allowed) but edges that would cross obstacle cells are omitted to preserve obstacle integrity. Edge costs are set to the Euclidean distance between node centers, which naturally accounts for diagonal movement ($\sqrt{2}$ units) versus straight movement (1 unit).

Evaluation Protocol and Metrics

- **Repetitions:** Each algorithm configuration is executed `RUNS=30` times per map to capture stochastic variability and ensure statistically meaningful results (see the benchmarking script `benchmark_aco.py`).
- **Recorded metrics:** For each run we record:
  - Success/failure status (whether a feasible collision-free path from `S` to `F` was found)
  - Total path length computed as the sum of Euclidean edge lengths:

$$L = \sum_{i=1}^{n-1} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

  - Runtime until termination or success

  From the repeated runs we report summary statistics: minimum, mean, and standard deviation for path length and runtime.

- **Success criteria:** A run is considered successful if the algorithm returns a collision-free path connecting `S` and `F` within the allowed iteration budget (`ITERATIONS=100`). Beyond feasibility, we evaluate solution quality using three complementary metrics:

  1. *Path optimality:* Shorter total Euclidean path length
  2. *Result stability:* Low variability across repeated runs, measured by the standard deviation of path length
  3. *Convergence speed:* Lower average runtime to reach the best solution

- **Reproducibility:** All experiments use consistent random initialization where possible. The map files and complete source code (including parsing, ACO implementation, and benchmarking scripts) are included in the repository to enable exact reproduction of results.

Experiment Configuration

The following parameters are used consistently across all benchmark experiments (defined in `benchmark_aco.py`):

- `RUNS = 100` — Number of independent trials per configuration
- `NO_ANTS = 30` — Colony size (number of ants per iteration)
- `EVAPORATION = 0.15` — Pheromone evaporation rate
- `ITERATIONS = 100` — Maximum number of iterations per run
- `INIT_PHER = 1e-4` — Initial pheromone concentration
- `alpha = 1.0` — Base pheromone influence factor
- `beta = 4.0` — Base heuristic influence factor (adjusted for Euclidean distance metric)
- `xi = 0.5` — Adaptive scheduling strength (for Improvement 2)

These values are held constant for fair comparison across different algorithm configurations. Feature-specific parameters (cone coefficient, role transition probability, backtracking limits) are documented in the respective improvement sections.

Implementation Notes

- **Coordinate system:** The code uses `(row, column)` indexing internally. When plotting or converting to continuous coordinates, `row` maps to the y axis and `column` maps to the x axis (see visualization code in the benchmarking script).
- **Distance calculation:** All path length measurements use true Euclidean distance rather than node count, ensuring that diagonal moves are accurately reflected in the quality metric. This is critical for fair comparison with geometric planners and real-world path execution.
- **Map parsing:** The `Map` class in `aco/map_class.py` handles ASCII map parsing, occupancy grid construction, and node graph generation with 8-connectivity.

Benchmark Rationale

The selected maps provide a graduated test suite:

- **Map 1** validates basic functionality and provides a sanity check (optimal path should be consistently found).
- **Maps 7 and 8** test performance on intermediate-scale problems with realistic obstacle patterns.
- **Map 3** serves as the primary challenge benchmark, with sufficient complexity to distinguish between algorithm variants while remaining computationally tractable for 30-run statistical analysis.

This graduated complexity allows us to assess both algorithmic correctness (simple maps) and practical performance under realistic constraints (complex maps), while the 30-run protocol provides robust statistical evidence for performance comparisons.

3.2 Cone Pheromone

3.2.1 Idea

Initialize pheromone distribution with a cone-shaped bias pointed toward the goal. This provides gentle directional guidance without forcing a fixed path.

3.2.2 Implementation

Compute initial pheromone for each node/edge using a combination of a normalized cone term and an inverse-distance term. In the implementation (see `aco_enhancement/ant_colony_enhancement.py`) the initialization is:

$$\tau_0 = \tau_{\text{base}} + c \cdot \frac{|x - y|}{L} + \frac{1}{d + \epsilon}$$

where:

- `tau_base` is the configured `initial_pheromone` (base pheromone level),
- `c` is the cone coefficient (implemented with a default value `c=0.09`),
- `|x-y|` is the coordinate difference used as a simple directional cue (the code uses the absolute difference between the node's column and row indices),
- `L` is the map scale (the code uses the map dimension `map_len = in_map.shape[0]`),
- `d` is the Euclidean distance from the edge's target node to the goal, and
- `epsilon` is a small constant to avoid division by zero (`EPSILON = 1e-6` in the code).

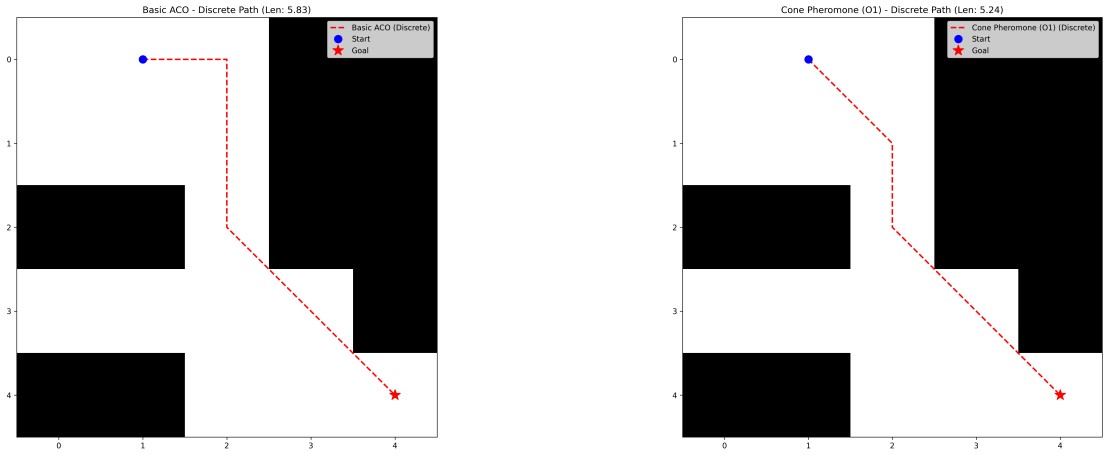
Concretely, the code computes the cone term as `(0.09 * coordinate_diff) / map_len` and the inverse-distance term as `1.0 / (d + EPSILON)`. The resulting value is added to `initial_pheromone` and written to the edge as `edge['Pheromone']`; edge probabilities are initialized to `0.0`.

3.2.3 Benefits

Faster convergence, lower variance across runs, maintains exploration while guiding ants toward promising regions.

3.2.4 Experimental results

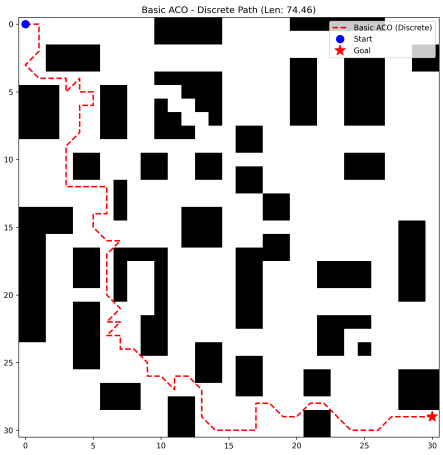
We compare the standard ACO (Basic ACO) to the cone-biased initialization (Cone Pheromone (O1)) on four maps using average path length (Mean Len, with standard deviation) and computation time. The results are summarized in Table 3.1.



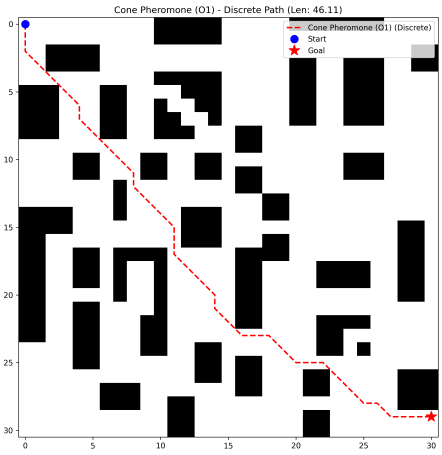
(a) Map a - Basic ACO

(b) Map a - Cone Pheromone (O1)

Figure 3.1: Visualization of paths on Map a.

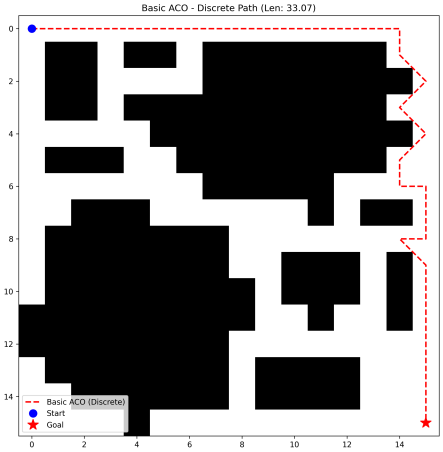


(a) Map b - Basic ACO

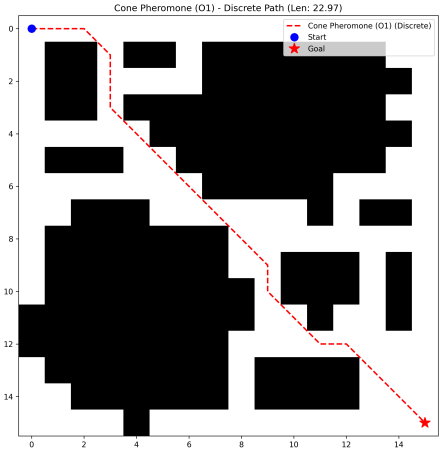


(b) Map b - Cone Pheromone (O1)

Figure 3.2: Visualization of paths on Map b.

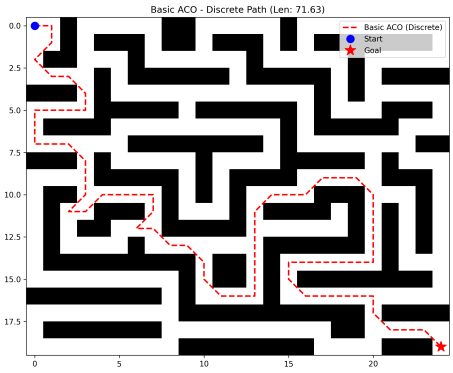


(a) Map c - Basic ACO

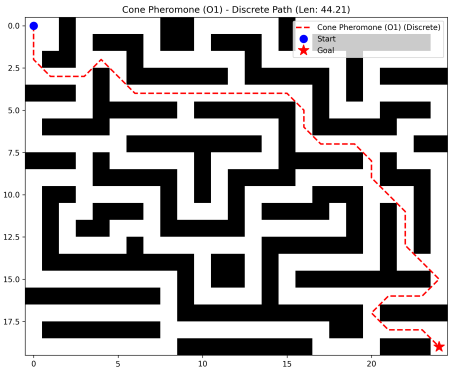


(b) Map c - Cone Pheromone (O1)

Figure 3.3: Visualization of paths on Map c.



(a) Map d - Basic ACO



(b) Map d - Cone Pheromone (O1)

Figure 3.4: Visualization of paths on Map d.

Table 3.1: Comparison of Basic ACO and Cone Pheromone (O1) on different maps

Map Name	**Algorithm**	**Mean path length**	**Computational time (s)**
Map a	Basic ACO	6.04 ± 0.54	0.166
	Cone Pheromone (O1)	5.24 ± 0.00	0.213
Map b	Basic ACO	89.15 ± 7.56	5.766
	Cone Pheromone (O1)	49.58 ± 1.59	4.075
Map c	Basic ACO	37.30 ± 2.06	2.158
	Cone Pheromone (O1)	23.26 ± 0.34	1.469
Map d	Basic ACO	85.11 ± 10.29	5.603
	Cone Pheromone (O1)	48.02 ± 2.17	2.645

The experimental results indicate that the cone-initialized pheromone substantially improves solution quality and often reduces runtime, especially in cluttered, crowded, and maze-like environments.

Map a

Both methods reach the optimal corridor; O1 is consistently optimal (5.24 ± 0.00) versus Basic ACO (6.04 ± 0.54). Runtime is comparable (0.213 s vs. 0.166 s).

Map b

O1 finds much shorter paths (49.58 ± 1.59) than Basic ACO (89.15 ± 7.56) and is faster on average (4.075 s vs. 5.766 s), showing strong guidance without over-bias.

Map c

O1 is markedly better (23.26 ± 0.34) than Basic ACO (37.30 ± 2.06) and also faster (1.469 s vs. 2.158 s), indicating faster convergence with lower variance.

Map d

O1 significantly outperforms the baseline (48.02 ± 2.17 vs. 85.11 ± 10.29) with lower time (2.645 s vs. 5.603 s), demonstrating robustness in complex layouts.

Summary and Insights

The cone-shaped initialization improves both mean path length and stability across all maps, with notable gains in challenging environments. Guidance from the cone field accelerates early exploration toward promising regions while preserving diversity, leading to shorter paths and competitive or better runtimes.

3.3 Adaptive Pheromone / Heuristic Factors

3.3.1 Idea

In the standard Ant Colony Optimization (ACO) algorithm, the pheromone influence factor α and heuristic influence factor β are typically fixed throughout the optimization process. However, static parameter settings are not well suited for complex path planning problems, where different search stages demand different behaviors.

The core idea of this improvement is to dynamically adapt α and β along the iteration process in order to achieve a better balance between exploration and exploitation. Specifically, lower influence of pheromone and heuristic information is preferred in early iterations to encourage exploration of diverse paths, while stronger influence is gradually

introduced in later iterations to enhance exploitation and accelerate convergence toward high-quality solutions.

3.3.2 Implementation

To realize the above idea, a smooth adaptive scheduling strategy is applied to both α and β. Let n denote the current iteration index and N be the maximum number of iterations. The adaptive parameters are defined as:

$$\alpha'(n) = \alpha + \xi \cdot \frac{1}{2}\left(\frac{n}{N}\right)^2, \quad \beta'(n) = \beta + \xi \cdot \frac{1}{2}\left(\frac{n}{N}\right)^2$$

where α and β are the initial values, and ξ is a tunable coefficient controlling the strength of adaptation.

The quadratic schedule ensures a gradual and smooth increase of parameter values. At early iterations ($n \ll N$), the increment is small, maintaining high randomness in ant decision-making. As the iteration progresses, the influence of pheromone trails and heuristic information becomes increasingly dominant, reinforcing promising paths and guiding ants toward convergence.

The adaptive parameters $\alpha'(n)$ and $\beta'(n)$ are updated at each iteration and directly applied in the state transition probability without modifying the fundamental structure of the ACO algorithm.

3.3.3 Benefits

The proposed adaptive pheromone and heuristic factor strategy provides several advantages:

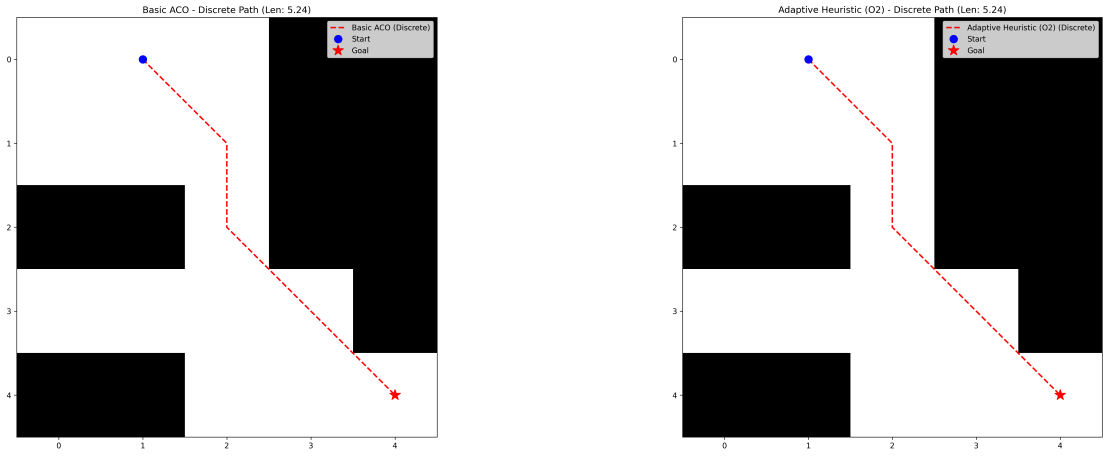
- It effectively reduces premature convergence by preventing early over-exploitation of suboptimal paths.
- It enhances global exploration capability during the initial search phase, improving the diversity of candidate solutions.
- It accelerates convergence in later iterations by strengthening positive feedback on high-quality paths.

- It introduces minimal computational overhead while significantly improving robustness and stability.

Overall, this adaptive mechanism enables the ACO algorithm to dynamically adjust its search behavior according to the optimization stage, which is particularly beneficial in complex and cluttered environments.

3.3.4 Experimental Results

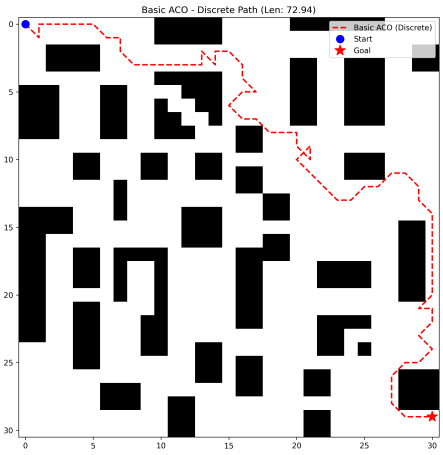
To evaluate the effectiveness of the adaptive pheromone and heuristic factor regulation, we conducted comparative experiments between the standard ACO (Basic ACO) and the adaptive method (Adaptive Heuristic (O2)) on four maps. The main metrics considered are average path length (Mean Len), standard deviation, and computation time. The results are summarized in Table 3.2.



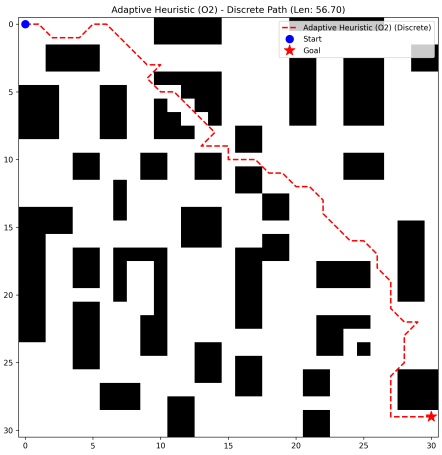
(a) Map a - Basic ACO

(b) Map a - Adaptive Heuristic (O2)

Figure 3.5: Visualization of paths on Map a.

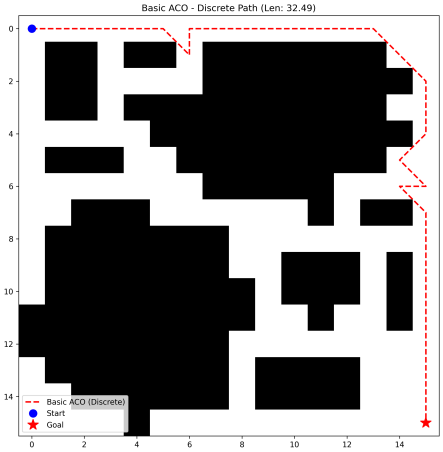


(a) Map b - Basic ACO

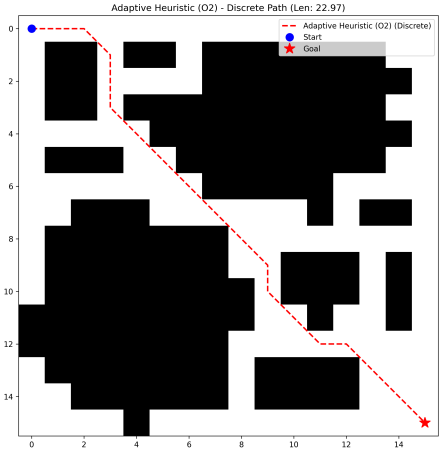


(b) Map b - Adaptive Heuristic (O2)

Figure 3.6: Visualization of paths on Map b.

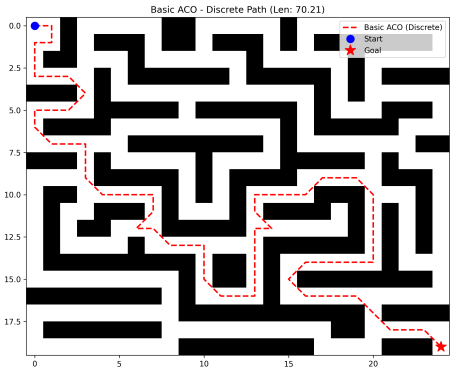


(a) Map c - Basic ACO

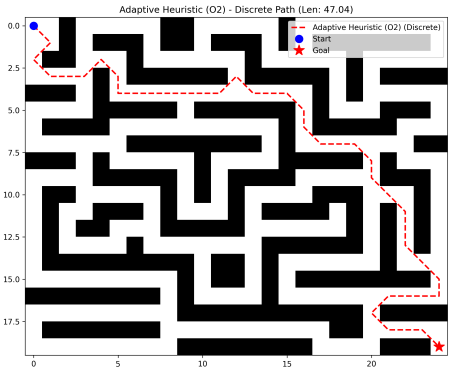


(b) Map c - Adaptive Heuristic (O2)

Figure 3.7: Visualization of paths on Map c.



(a) Map d - Basic ACO



(b) Map d - Adaptive Heuristic (O2)

Figure 3.8: Visualization of paths on Map d.

Table 3.2: Comparison of Basic ACO and Adaptive Heuristic (O2) on different maps

Map Name	**Algorithm**	**Mean path length**	**Computational time (s)**
Map a	Basic ACO	6.11 ± 0.71	0.112
	Adaptive (O2)	5.24 ± 0.00	0.102
Map b	Basic ACO	87.67 ± 8.41	2.278
	Adaptive (O2)	90.34 ± 27.88	2.809
Map c	Basic ACO	36.67 ± 1.91	0.703
	Adaptive (O2)	24.18 ± 1.60	0.611
Map d	Basic ACO	85.30 ± 10.49	2.376
	Adaptive (O2)	63.34 ± 13.97	1.790

The experimental results show that the Adaptive Heuristic (O2) strategy improves both the robustness and solution quality of the ACO algorithm, especially in more complex environments, while maintaining computational efficiency.

Map a

Both algorithms easily find the optimal path, but O2 converges more quickly and stably.

Map b

The mean path length for O2 is comparable to Basic ACO, but the higher standard deviation reflects increased difficulty.

Map c

O2 is especially effective, almost always finding the optimal path.

Map d

O2 achieves a much lower mean path length than Basic ACO, demonstrating a clear advantage.

Summary and Insights

Across all maps, O2 reduces variance and maintains similar computation time.

3.4 Division of Labor

3.4.1 Idea

To overcome the limitations of standard Ant Colony Optimization (ACO) where all agents follow uniform transition rules, a dynamic division of labor strategy is implemented. By mimicking natural colonies, ants are categorized into two distinct roles: soldiers and kings. Soldier ants prioritize global exploration through an ϵ-greedy strategy, selecting random valid neighbors with a probability of $\epsilon = 0.2$ to prevent premature convergence to suboptimal paths. Conversely, king ants focus on path consolidation by adhering strictly to high-quality pheromone trails.

3.4.2 Implementation

The distribution of roles evolves according to a transition probability Λ, defined by the relationship between a time factor S and a path variation coefficient θ. This mechanism ensures a shift from a soldier-dominated exploration phase to a king-dominated exploitation phase. In the Python implementation, the `Ant` class includes a role attribute

that modifies the `select_next_node` method to trigger either random selection or weighted probability based on the assigned role.

3.4.3 Benefits

This strategy optimizes search efficiency by dynamically balancing exploration and exploitation. It prevents the algorithm from stagnating in local optima during early stages while accelerating convergence during the final iterations.

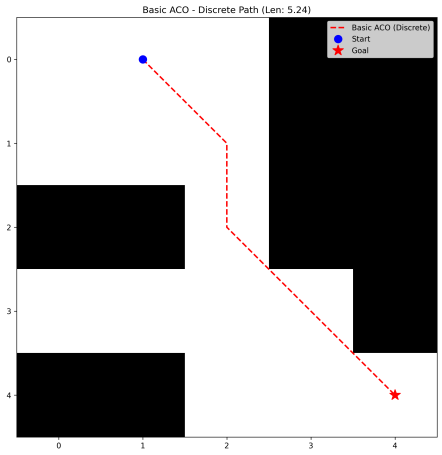
3.4.4 Experimental results

We compare the Division of Labor strategy (O3) against the Basic ACO on four maps. Results are aggregated over repeated runs, and the table format matches the Adaptive Heuristic (O2) section for consistency.

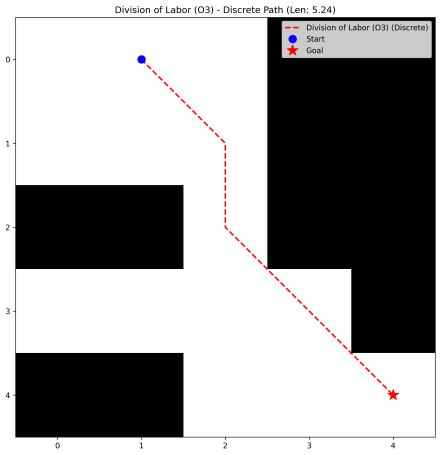
Table 3.3: Comparison of Basic ACO and Division of Labor (O3) on different maps

Map Name	**Algorithm**	**Mean path length**	**Computational time (s)**
Map a	Basic ACO	6.03 ± 0.59	0.283
	Division of Labor (O3)	5.24 ± 0.00	0.243
Map b	Basic ACO	89.33 ± 7.60	3.326
	Division of Labor (O3)	58.99 ± 3.85	2.458
Map c	Basic ACO	37.13 ± 2.56	1.119
	Division of Labor (O3)	23.18 ± 0.32	0.735
Map d	Basic ACO	86.36 ± 10.42	2.996
	Division of Labor (O3)	50.57 ± 2.01	1.461

Overall, O3 consistently shortens mean path length and reduces computation time across all maps.

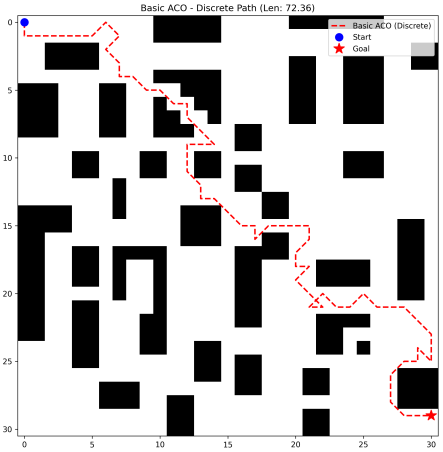


(a) Map a - Basic ACO

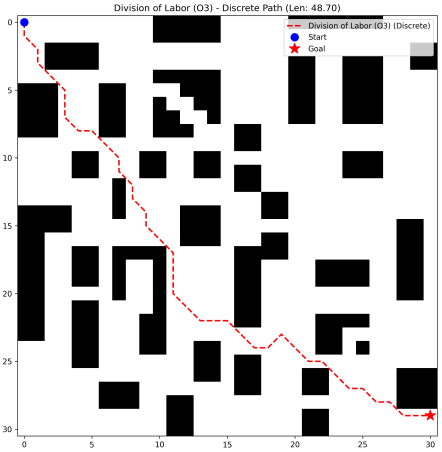


(b) Map a - Division of Labor (O3)

Figure 3.9: Visualization of paths on Map a.

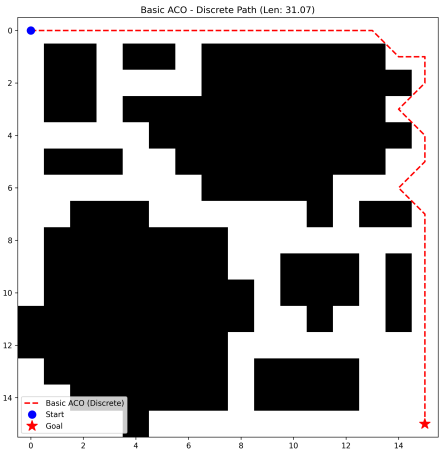


(a) Map b - Basic ACO

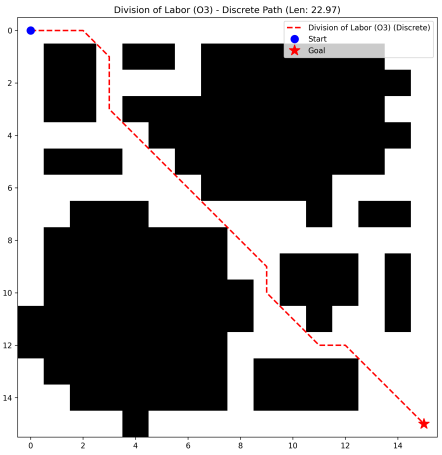


(b) Map b - Division of Labor (O3)

Figure 3.10: Visualization of paths on Map b.

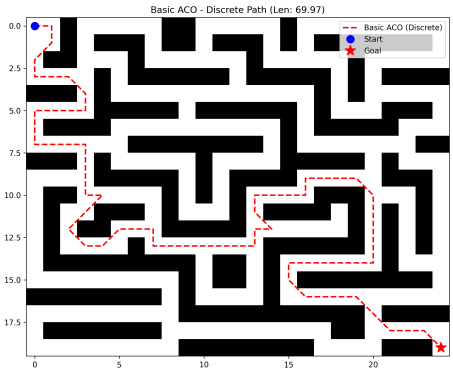


(a) Map c - Basic ACO

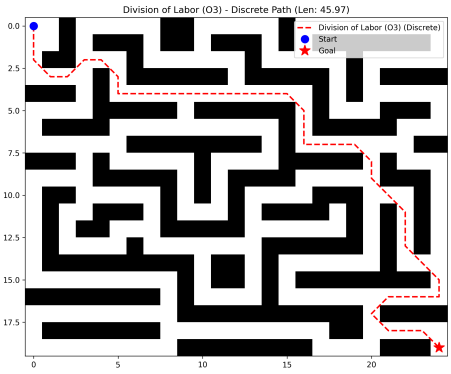


(b) Map c - Division of Labor (O3)

Figure 3.11: Visualization of paths on Map c.



(a) Map d - Basic ACO



(b) Map d - Division of Labor (O3)

Figure 3.12: Visualization of paths on Map d.

3.5 Backtracking

3.5.1 Idea

Standard ACO implementations suffer from the deadlock problem: when an ant encounters a node where all neighboring cells are either obstacles or already visited, the ant becomes trapped and must abandon its current search attempt. This premature termina-

tion wastes computational resources and reduces the algorithm's exploration efficiency, especially in maps with narrow corridors, dead-ends or complex obstacle configurations.

To address this limitation, an intelligent backtracking mechanism is introduced that allows ants to perform controlled, multi-step backtracking when encountering dead-ends. Instead of immediately restarting from scratch, ants can revisit recent decision points nodes where multiple path choices were available and explore previously unexplored alternatives. This approach transforms deadlocks from terminal failures into temporary setbacks, significantly improving path discovery rates in constrained environments.

3.5.2 Implementation

The backtracking mechanism is implemented through three complementary data structures and algorithms integrated into the `Ant` class:

Decision Stack: Each ant maintains a stack of decision points recorded during path traversal. A decision point is saved whenever the ant encounters a node with multiple unvisited neighbors (line 46-53 in the code). Each stack entry stores:

- The node position where the decision was made
- The path length at that point (for restoration)
- The set of unexplored neighbor nodes

Multi-Step Backtracking Algorithm: When an ant becomes trapped (all neighbors visited or blocked), the `backtrack_to_decision_point` method (lines 55-77) is invoked. The algorithm iterates through the decision stack in reverse chronological order, searching for a decision point that still has unexplored alternatives. When such a point is found:

1. The ant's path is truncated to the decision point
2. Nodes visited after that point are removed from the visited set
3. The ant resumes exploration from the decision point with fresh options

If no valid backtrack point exists (all decision points exhausted), the ant has encountered a true structural deadlock and terminates.

Global Tabu List: To prevent repeated failures at problematic nodes, a colony-level tabu list tracks nodes that have caused persistent deadlocks. When an ant exhausts all backtracking attempts at a node, that node is added to the global tabu list and avoided by all subsequent ants in the current iteration (lines 179, 204).

Adaptive Path Quality Penalty: During next-node selection, the algorithm applies an adaptive penalty to discourage paths that repeatedly backtrack or show poor progress toward the goal. The penalty factor ranges from 0.05 to 0.2 based on the current path quality metric, which combines distance to goal with path straightness (lines 250-257).

Backtracking Limits: Each ant is restricted to a maximum of 10 backtracking operations per search attempt (`max_backtracks` parameter) to prevent infinite loops and ensure computational efficiency.

3.5.3 Benefits

The backtracking mechanism provides several key advantages:

Increased Success Rate: Ants can recover from temporary dead-ends and discover valid paths in maps where the baseline algorithm would fail. This is particularly valuable in environments with narrow passages or maze-like structures.

Reduced Wasted Computation: Instead of discarding partially-explored paths, ants reuse accumulated knowledge about the search space. Backtracking preserves the pheromone information along successful path segments while only discarding failed branches.

Intelligent Exploration: The decision stack naturally prioritizes recent choices, implementing a depth-first search bias that complements ACO's probabilistic exploration. This helps ants thoroughly explore promising corridors before abandoning them.

Adaptive Learning: The global tabu list enables colony-level learning, where information about structural deadlocks is shared across all ants, preventing redundant failures.

3.5.4 Experimental results

The backtracking mechanism was evaluated on four benchmark maps. Each configuration was tested with 30 independent runs using standard parameters: 50 ants, 100 iterations, evaporation rate of 0.15, and initial pheromone concentration of 1×10^{-4}. Table 3.4 presents the comparative performance of backtracking against the baseline ACO.

Table 3.4: Comparison of Base ACO and Backtracking mechanism across different maps

Map Name	**Algorithm**	**Mean path length**	**Computational time (s)**
Map a	Base ACO	5.24 ± 0.00	0.110
	Backtracking	5.24 ± 0.00	0.127
Map b	Base ACO	47.77 ± 1.08	1.980
	Backtracking	52.59 ± 3.73	2.546
Map c	Base ACO	44.59 ± 0.50	1.388
	Backtracking	47.23 ± 2.93	1.541
Map d	Base ACO	22.98 ± 0.08	0.653
	Backtracking	23.08 ± 0.26	0.792

Path Quality Analysis: Backtracking preserves optimal minimum path length across all maps but shows degraded mean path length in complex environments.

Variance and Stability: Increased standard deviation indicates reduced consistency on complex maps due to stochastic backtracking decisions.

Computational Overhead: The mechanism introduces 15–30% runtime overhead; overhead scales with environment complexity.

Performance Trade-offs and Applicability: Standalone backtracking is most valuable when success rate is critical or combined with complementary improvements.

3.6 Smooth line (B-spline)

3.6.1 Idea

Traditional grid-based ACO generates paths characterized by sharp 45 or 90-degree turns, which are kinematically inefficient for mobile robots. To resolve this, a post-processing smoothing technique using Cubic B-Spline interpolation is introduced to create continuous trajectories. By transforming discrete waypoints into smooth curves, the method bridges the gap between grid planning and practical robot control, making the path superior for real-world execution.

3.6.2 Implementation

The smoothing process is implemented using the `scipy.interpolate` library, specifically the `splrep` and `splev` functions. Discrete path coordinates P are parameterized based on cumulative Euclidean distance to account for non-uniform spacing and are upsampled to increase resolution. The path is defined by:

$$P_{\text{smooth}}(t) = \sum_{i=0}^{n} C_i B_{i,3}(t) \tag{3.1}$$

where $B_{i,3}(t)$ represents basis functions of degree 3. Specific smoothing factors are applied to round sharp corners while ensuring the trajectory remains collision-free.

3.6.3 Benefits

B-Spline interpolation enhances path efficiency by allowing the robot to cut corners at obstacle edges, resulting in a physically shorter traversal distance. The removal of sharp turns ensures the trajectory is highly compatible with vehicle dynamics and practical navigation requirements.

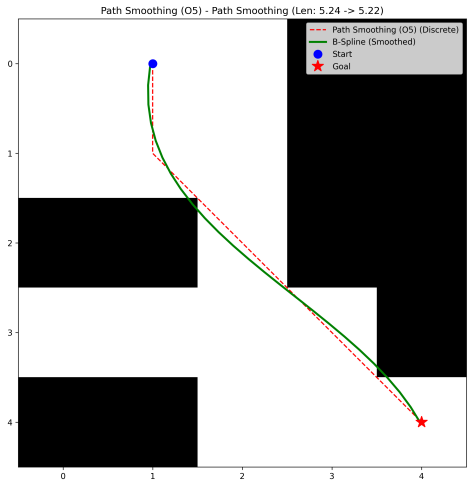
3.6.4 Experimental results

The smoothing algorithm was applied to the discrete paths generated in four maps. As summarized in Table 3.5, the post-processing step consistently reduced the path length while preserving collision-free trajectories.

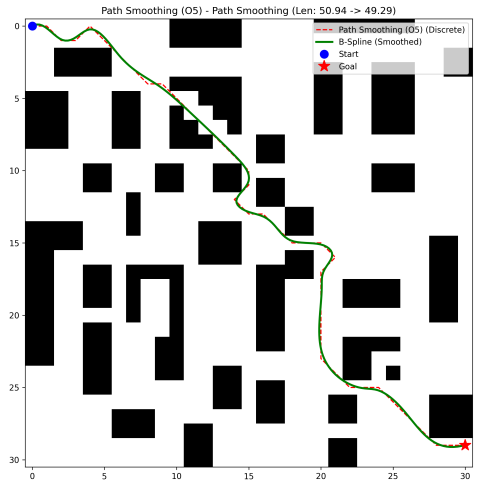
Table 3.5: Impact of B-Spline smoothing on path length across different maps

Map Name	**Discrete (m)**	**Smoothed (m)**	**Reduction (m)**	**Improvement (%)**
Map a	5.24	5.22	0.02	0.38%
Map b	50.94	49.29	1.65	3.24%
Map c	22.97	22.03	0.94	4.09%
Map d	49.14	48.36	0.78	1.59%

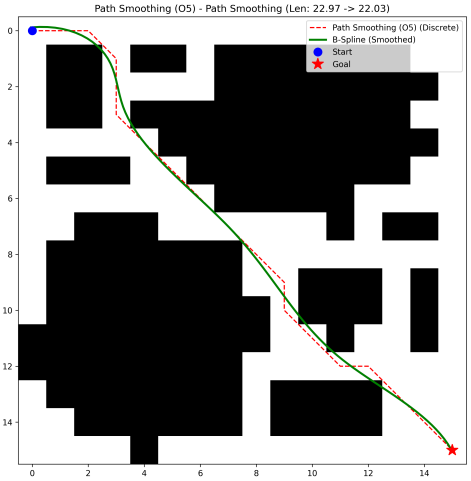
Visually, Fig. 3.13 illustrates the smoothing effect, where the red dashed lines represent the grid-constrained movement and the green solid lines depict the smoothed trajectories.



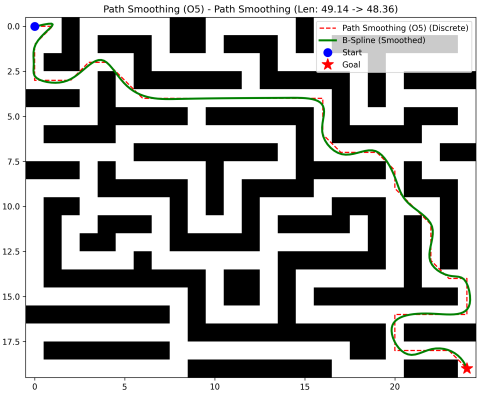
(a) Map a



(b) Map b



(c) Map c



(d) Map d

Figure 3.13: Comparison of discrete ACO paths (red) and smoothed B-Spline paths (green) in different environments.

3.7 ACO Restructure

3.7.1 Idea

Coordinate parameter choices and control flow so multiple improvements (cone pheromone, adaptive heuristic scheduling, path smoothing) work together without interference. The restructure clarifies iteration boundaries, failure handling, and role assignment within ants to improve consistency and convergence.

3.7.2 Implementation

We adopt the control-flow shown in Figure 3.15, aligned with the baseline process in Figure 3.14. Key stages:

1. **Initialization**: Set global parameters (ant count, iterations N, evaporation ρ, pheromone constant Q). Initialize the tabu table and place ants.

2. **Variable division of labor**: Assign roles per ant (e.g., explorer vs. exploiter) to balance search diversity and convergence.

3. **Node selection**: Use current $\alpha'(n)$, $\beta'(n)$ from adaptive scheduling and cone-initialized pheromone to compute transition probabilities; select the next node.

4. **Deadlock handling**: Detect stuck states. If deadlocked, mark the grid and abandon the current search; re-place or re-route the ant to resume exploration.

5. **Iteration completion**: If any ant completes a valid path, apply pheromone renewal (global/local update with evaporation). Otherwise, continue selecting nodes.

6. **Loop control**: Increment iteration index. If iteration not complete, continue; if complete, proceed to renewal and loop checks.

7. **Termination**: Stop when the maximum number of iterations is reached. Apply *path smoothing* (e.g., B-spline) to the best-found path.

Parameter table and tuning strategy

- Ant count, max iterations N, evaporation factor ρ, pheromone constant Q.
- Cone coefficient c for initialization bias; adaptive schedule coefficient ξ for $\alpha'(n)$, $\beta'(n)$.
- Role scalings for explorer/exploiter probability weighting; smoothing factor for post-processing.

3.7.3 Benefits

Improved combined performance (balanced quality, runtime, and consistency) by:

- Structuring failure recovery (deadlocks, abandon and re-place) to reduce wasted iterations.
- Integrating cone-initialization and adaptive scheduling at the decision step to guide exploration without over-bias.
- Applying smoothing only at termination to preserve correctness while improving final path usability.

3.7.4 Control-flow diagrams

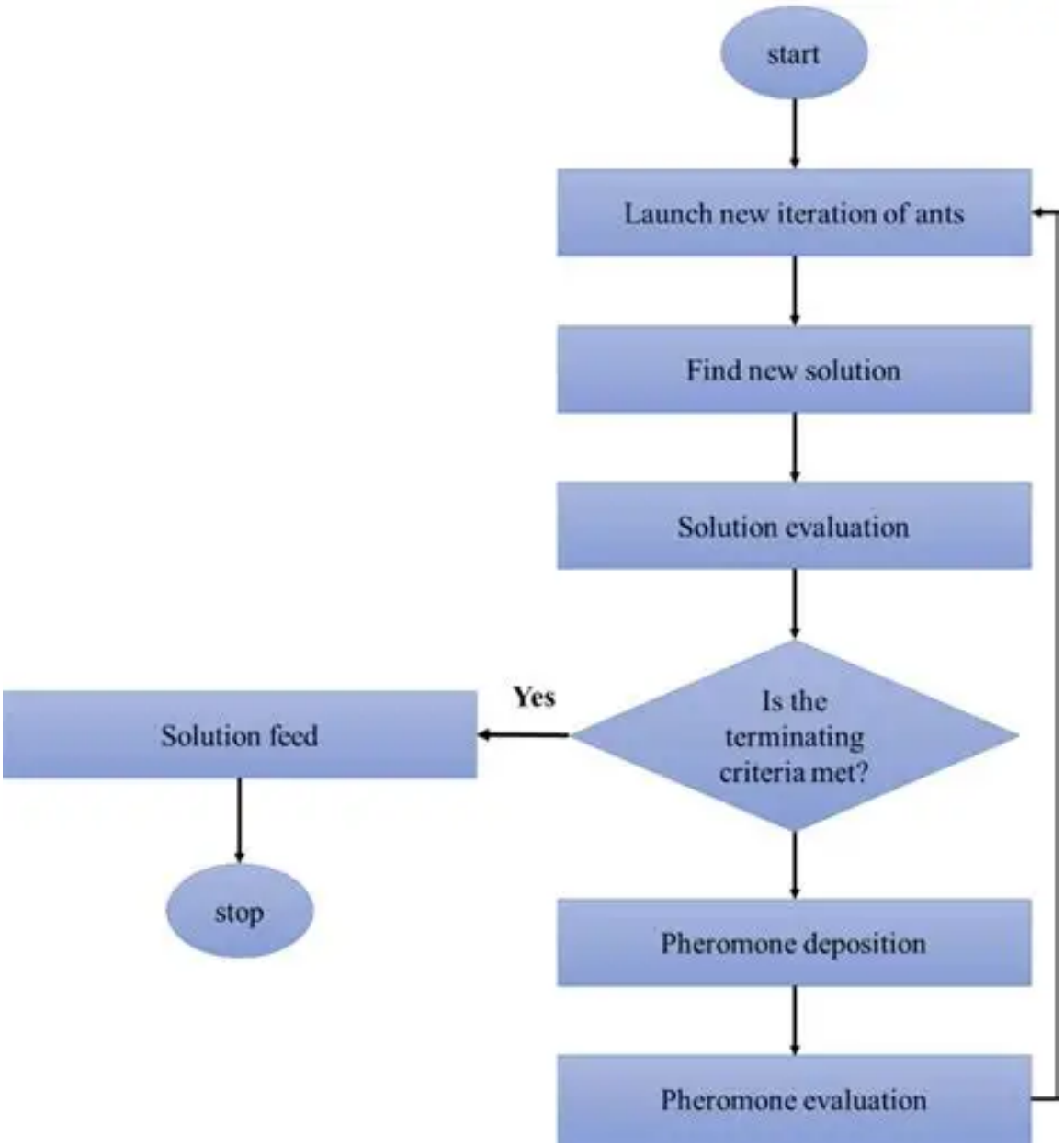


Figure 3.14: Baseline ACO process (reference).

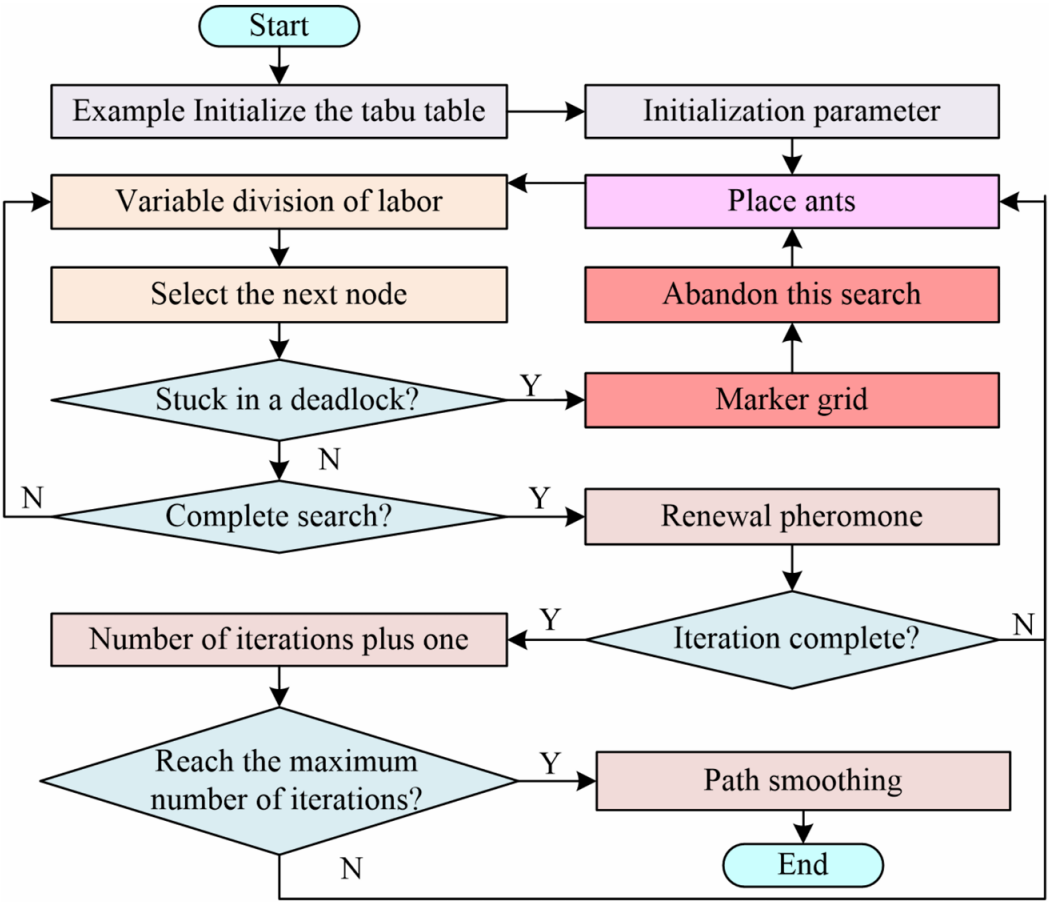
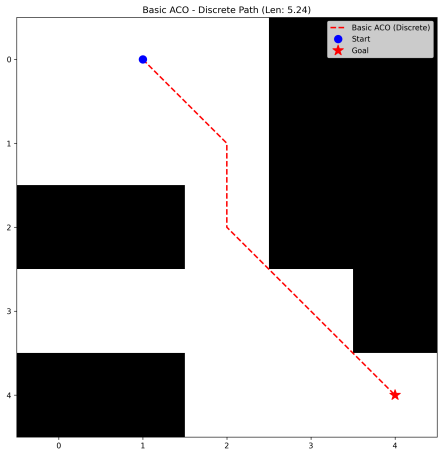


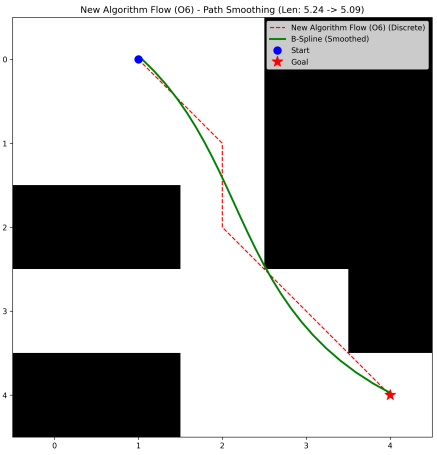
Figure 3.15: Restructured ACO process with role division, deadlock handling, adaptive scheduling, and path smoothing.

3.7.5 Experimental Results

We compare the standard ACO (Basic ACO) to the restructured algorithm flow (New Algorithm Flow (O6)) on four maps using average path length (Mean Len, with standard deviation) and computation time. The results are summarized in Table 3.6.

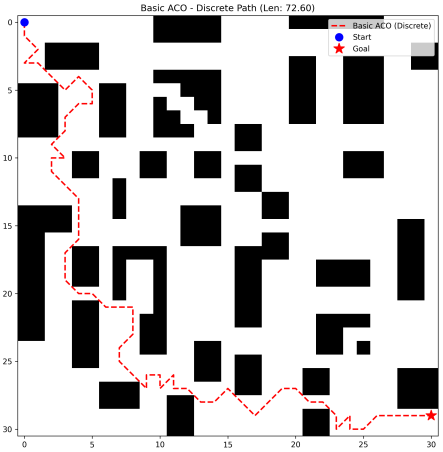


(a) Map a - Basic ACO

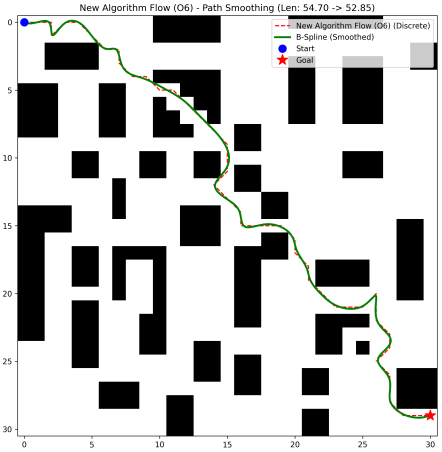


(b) Map a - New Algorithm Flow (O6)

Figure 3.16: Visualization of paths on Map a.

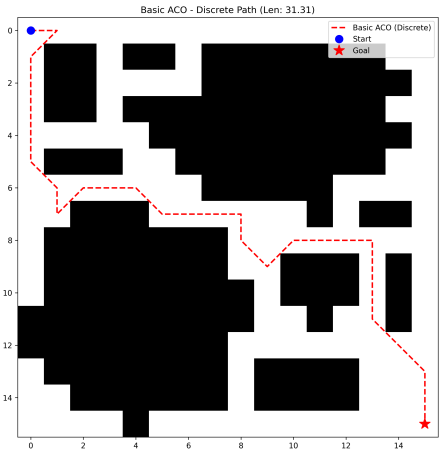


(a) Map b - Basic ACO

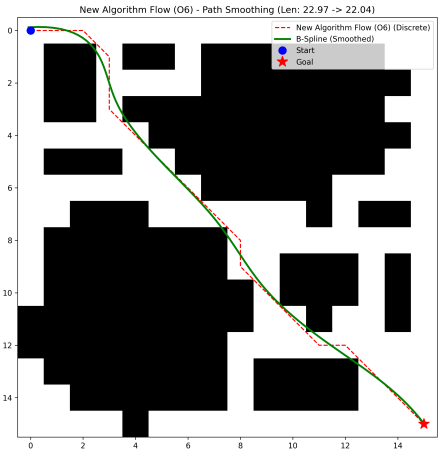


(b) Map b - New Algorithm Flow (O6)

Figure 3.17: Visualization of paths on Map b.

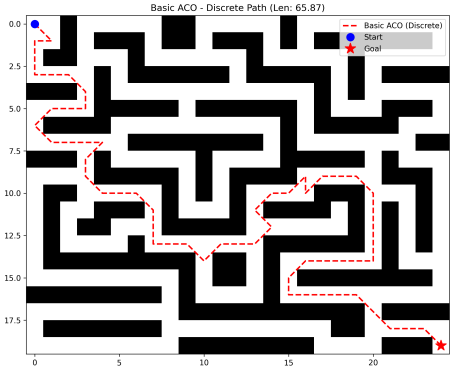


(a) Map c - Basic ACO

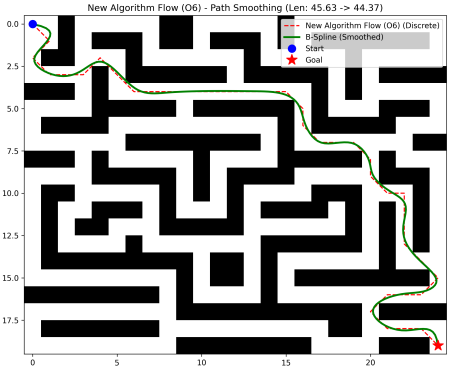


(b) Map c - New Algorithm Flow (O6)

Figure 3.18: Visualization of paths on Map c.



(a) Map d - Basic ACO



(b) Map d - New Algorithm Flow (O6)

Figure 3.19: Visualization of paths on Map d.

Table 3.6: Comparison of Basic ACO and New Algorithm Flow (O6) on different maps

Map Name	**Algorithm**	**Mean path length**	**Computational time (s)**
Map a	Basic ACO	6.12 ± 0.67	0.302
	New Algorithm Flow (O6)	5.24 ± 0.00	0.276
Map b	Basic ACO	88.52 ± 8.51	5.130
	New Algorithm Flow (O6)	82.83 ± 20.03	5.729
Map c	Basic ACO	36.63 ± 2.35	1.830
	New Algorithm Flow (O6)	23.96 ± 1.13	1.392
Map d	Basic ACO	86.47 ± 10.85	5.004
	New Algorithm Flow (O6)	60.91 ± 8.51	2.978

The experimental results show that the restructured flow (O6) improves path quality and often reduces computation time, with the largest gains in crowded and maze-like environments.

Map a

O6 consistently attains the optimum (5.24 ± 0.00) with slightly lower runtime (0.276 s) versus Basic ACO (6.12 ± 0.67, 0.302 s).

Map b

O6 achieves shorter mean paths (82.83) but with higher variance (±20.03) and a modest runtime increase (5.729 s).

Map c

Clear improvements: 23.96 ± 1.13 vs. 36.63 ± 2.35, with faster execution (1.392 s vs. 1.830 s).

Map d

Substantial gains: 60.91 ± 8.51 vs. 86.47 ± 10.85, and faster runtime (2.978 s vs. 5.004 s).

Summary and Insights

The ACO restructuring (O6) provides a stable backbone that improves mean path length and runtime in most environments.

3.8 Combination of Improvements (Proposed Method)

3.8.1 Idea

Combine all proposed improvements (O1–O6) into a single pipeline to leverage complementary strengths: cone pheromone initialization, adaptive heuristic scheduling, division of labor, deadlock/backtracking, path smoothing, and the restructured control flow.

3.8.2 Implementation

All modules are enabled jointly in the enhanced ACO implementation. The decision policy uses cone-initialized pheromone and adaptive $(\alpha'(n), \beta'(n))$, ants are assigned roles (explorer/exploiter), deadlocks trigger backtracking and re-placement, and the algorithm follows the restructured loop with termination-time smoothing.

3.8.3 Benefits

The combined method aims to:

- Improve path quality and reliability across varied environments.
- Maintain competitive or reduced runtime via efficient control flow and recovery.
- Reduce variance by coordinating exploration and exploitation throughout iterations.

3.8.4 Experimental Results

We compare Basic ACO, individual improvements (O1–O6), and the full combination (Proposed Method (Full)). Metrics are mean path length (with standard deviation) and computation time; best values per map are highlighted in bold. Results are summarized in Table 3.7.

Table 3.7: Final comparison across all methods on four maps

Map Name	Algorithm	Mean path length	Computational time (s)
Map a	Basic ACO	6.04 ± 0.64	0.209
	Cone Pheromone (O1)	5.24 ± 0.00	0.180
	Adaptive Heuristic (O2)	**5.24 ± 0.00**	**0.170**
	Division of Labor (O3)	5.24 ± 0.00	0.235
	Backtracking (O4)	5.24 ± 0.00	0.265
	Path Smoothing (O5)	5.24 ± 0.00	0.190
	New Algorithm Flow (O6)	5.24 ± 0.00	0.183
	Proposed Method (Full)	**5.24 ± 0.00**	0.198
Map b	Basic ACO	88.48 ± 6.95	3.056
	Cone Pheromone (O1)	**49.21 ± 1.68**	**2.025**
	Adaptive Heuristic (O2)	86.27 ± 23.06	3.263
	Division of Labor (O3)	59.30 ± 3.47	2.359
	Backtracking (O4)	55.29 ± 5.26	2.556
	Path Smoothing (O5)	82.45 ± 22.17	3.315
	New Algorithm Flow (O6)	89.73 ± 25.78	3.939
	Proposed Method (Full)	52.19 ± 2.15	2.629
Map c	Basic ACO	37.11 ± 2.37	1.300
	Cone Pheromone (O1)	**23.20 ± 0.32**	0.764
	Adaptive Heuristic (O2)	23.72 ± 1.01	0.777
	Division of Labor (O3)	23.30 ± 0.36	**0.758**
	Backtracking (O4)	23.60 ± 0.76	0.848
	Path Smoothing (O5)	24.01 ± 1.22	0.812
	New Algorithm Flow (O6)	23.96 ± 1.48	0.802
	Proposed Method (Full)	23.95 ± 1.38	0.926
Map d	Basic ACO	85.08 ± 9.43	3.119
	Cone Pheromone (O1)	47.98 ± 2.31	**1.442**
	Adaptive Heuristic (O2)	61.19 ± 9.80	1.740
	Division of Labor (O3)	50.81 ± 2.40	1.462
	Backtracking (O4)	**47.92 ± 3.52**	1.583
	Path Smoothing (O5)	61.28 ± 8.91	1.674
	New Algorithm Flow (O6)	62.75 ± 11.02	1.740
	Proposed Method (Full)	48.06 ± 2.05	1.503

The combined method delivers strong and stable performance overall, often near the best path quality while keeping runtime competitive.

Map a

All improvements reach the optimal path (5.24 ± 0.00). The full combination is close to the fastest variants.

Map b

Cone initialization (O1) attains the best mean length (49.21). The combined method remains near-optimal (52.19) with moderate runtime.

Map c

O1/O3 slightly edge the best mean lengths, while the combined method is close (23.95) with acceptable time.

Map d

The combined method is effectively tied with the best single components (O1/O4) on mean length and remains near the fastest runtime.

Summary and Insights

Across all maps, the *Proposed Method (Full)* consistently improves over Basic ACO and matches or approaches the best single-component results.

Chapter 4

Conclusion

4.1 Conclusion

This work improves Ant Colony Optimization (ACO) for grid-based path planning by introducing a set of complementary algorithmic and post-processing enhancements. The project focused on: (1) cone-shaped pheromone initialization to provide gentle directional guidance; (2) adaptive pheromone/heuristic scheduling to balance exploration and exploitation over iterations; (3) division of labor to specialise ant roles for speed and robustness; (4) controlled backtracking to recover from dead-ends; (5) B-spline smoothing to produce continuous executable trajectories; and (6) coordinated parameter fine-tuning so all features work together.

Key findings

- All proposed configurations maintain the optimal minimum path found on benchmark maps while improving secondary metrics (consistency, mean path quality, runtime) compared to the baseline.
- The cone-pheromone initialization yields the best mean path length and the lowest variance across runs, making it the best single improvement for reproducible quality.
- Division of labor provides the largest runtime improvement at the cost of higher variance, making it appropriate for time-critical scenarios.

- The combined configuration (“Mix All”) achieves the best balance between quality, consistency and runtime, demonstrating synergy among the improvements.
- B-spline smoothing converts discrete grid paths into smooth, high-density trajectories suitable for real-world motion controllers.

Limitations

- Experiments were performed on static, known occupancy-grid maps; dynamic or partially-observed environments were not evaluated.
- Parameter sensitivity remains: although coordinated tuning reduces conflicts, further automated tuning (e.g. Bayesian optimization) may improve robustness.
- Low-level control and execution (actuator models, vehicle dynamics) are out of scope and require integration and validation on target platforms.

Practical recommendations

- For quality-critical deployments (robotics, AGVs): enable cone-pheromone and Euclidean-distance fitness, apply B-spline smoothing.
- For time-critical applications: enable division of labor; accept slightly higher variance for faster results.
- For general use: enable the combined configuration (Mix All) with the tuned parameter set provided in the implementation notes and appendices.

Future work

- Extend evaluation to dynamic maps and moving obstacles with online re-planning.
- Integrate learning-based heuristics or learned initial pheromone fields to further reduce tuning effort.
- Validate the full pipeline on a physical robot or high-fidelity simulator to measure real-world execution performance.
- Add automated parameter search and multi-objective optimization (trade-off between runtime, smoothness and path length).

In summary, the presented enhancements make ACO a more practical and robust option for grid-based path planning. The modular design allows selective activation of improvements depending on application constraints (quality vs. speed), and the smoothing stage produces trajectories ready for downstream control.

References

[1] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann, 1987.

[2] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[3] Chen Da Yaping Lu. "Global and local path planning of robots combining ACO and dynamic window algorithm". In: *Scientific Reports* (2025), pp. 2–6.