

**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING**



**REPORT
CAPSTONE PROJECT
SEMESTER 241 ACADEMIC YEAR 2023-2024**

<TITLE>

MAJOR: COMPUTER SCIENCE

COUNCIL: <NAME>

SUPERVISOR(s): ...

REVIEWER:

——o0o——

STUDENT 1: <NAME> - <ID>

STUDENT 2: <NAME> - <ID>

HO CHI MINH CITY, December 2024

Chương 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 convergence, 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 step. 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: Theoretical Background, fundamentals of Ant Colony Optimization and relevant theory.

- Chapter 3: Overview and Approach, high-level description of the implemented ACO and the set of proposed improvements.
- Chapter 4: Detailed Improvements, dataset/benchmark description and per-improvement: idea, implementation, benefits, and experimental results.
- Chapter 5: Optional Team Improvements, additional ideas and extensions.
- Chapter 6: Conclusion, summary and future work.

Chương 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 \hat{E} 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.

Chương 3

Detailed Improvements (Chung)

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 1 (Simple):** 5×5 grid with basic obstacle configuration, suitable for validating algorithmic correctness and baseline performance.
- **Map 3 (Complex):** 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 7 (Medium):** 25×20 grid with moderate obstacle density and mixed open/-constrained regions, representing intermediate difficulty.

- **Map 8 (Medium):** 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 = 30` — Number of independent trials per configuration
- `NO_ANTS = 50` — 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 (Ón)

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`) the initialization is:

$$au_0 = \tau_{\text{base}} + c \cdot \frac{|x - y|}{L} + \frac{1}{d + \epsilon}$$

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

Insert per-map quantitative results here: min/mean/std path length and runtime compared to baseline. When reporting results, include ablations that:

- compare `use_cone_pheromone=True/False`,
- sweep the cone coefficient (e.g. $c \in \{0.01, 0.05, 0.09, 0.2\}$) to show its effect on bias vs. exploration,
- report min/mean/std path length, success rate, and runtime, and
- visualize the initial pheromone field (heatmap) overlaid with example paths to illustrate how the cone bias shapes early exploration.

The implementation notes above and the suggested ablations will help quantify trade-offs between faster convergence and the risk of over-biasing the search towards suboptimal corridors.

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.

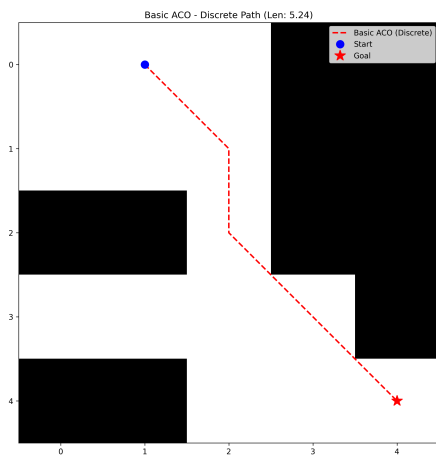
Overall, this adaptive mechanism enables the ACO algorithm to dynamically adjust its search behavior according to the optimization stage, which is particularly beneficial in complex and cluttered environments.

3.3.4 Experimental Results

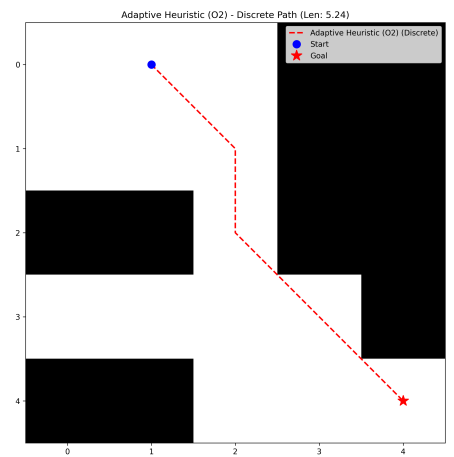
The four test maps used in the experiments correspond to different types of environments:

- **Map a:** Simple environment
- **Map b:** Cluttered environment
- **Map c:** Crowded environment
- **Map d:** Maze-like environment

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 different maps. The main metrics considered are average path length (Mean Len), standard deviation, and computation time. The results are summarized in Table 3.1.

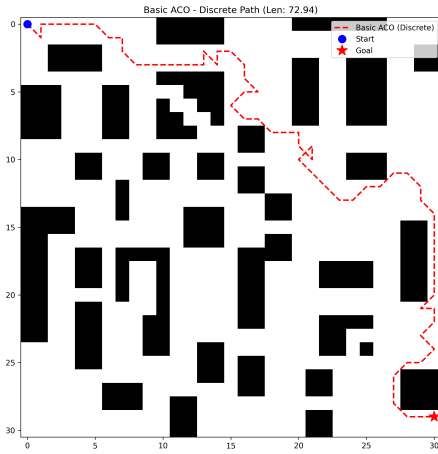


(a) Map a - Basic ACO

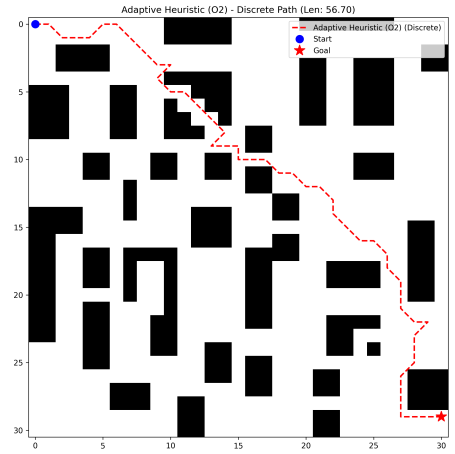


(b) Map a - Adaptive Heuristic (O2)

Hình 3.1: Visualization of paths on Map a.

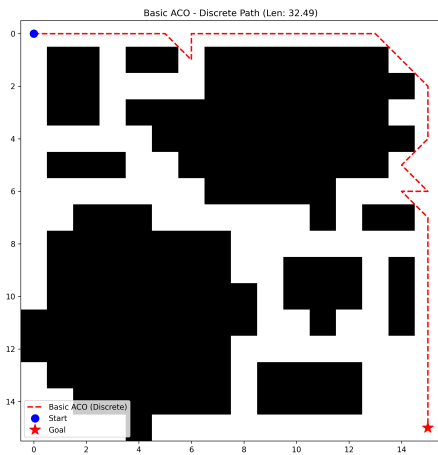


(a) Map b - Basic ACO

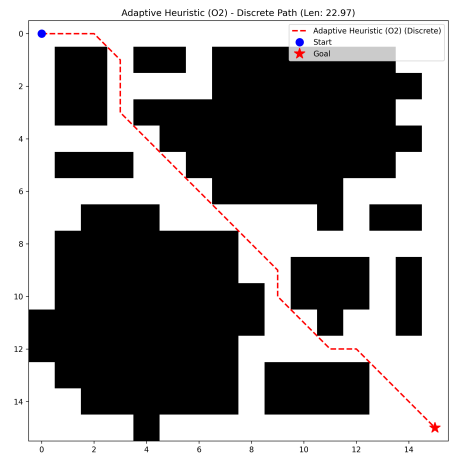


(b) Map b - Adaptive Heuristic (O2)

Hình 3.2: Visualization of paths on Map b.

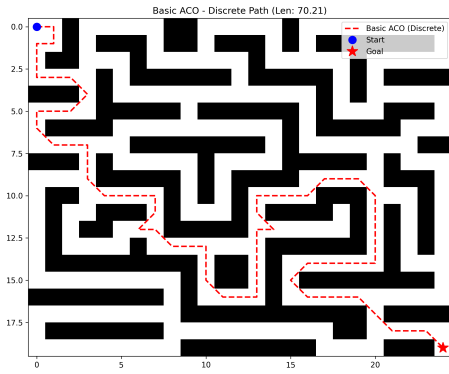


(a) Map c - Basic ACO

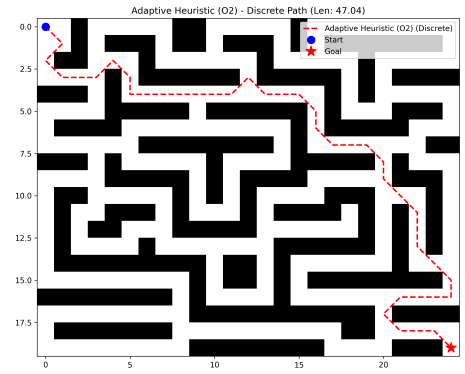


(b) Map c - Adaptive Heuristic (O2)

Hình 3.3: Visualization of paths on Map c.



(a) Map d - Basic ACO



(b) Map d - Adaptive Heuristic (O2)

Hình 3.4: Visualization of paths on Map d.

Bảng 3.1: Comparison of Basic ACO and Adaptive Heuristic (O2) on different maps

Map Name (Description)	Algorithm	Mean path length	Computational time (s)
map a (Simple)	Basic ACO	6.11 ± 0.71	0.112
	Adaptive (O2)	5.24 ± 0.00	0.102
map b (Cluttered)	Basic ACO	87.67 ± 8.41	2.278
	Adaptive (O2)	90.34 ± 27.88	2.809
map c (Crowded)	Basic ACO	36.67 ± 1.91	0.703
	Adaptive (O2)	24.18 ± 1.60	0.611
map d (Maze-like)	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. The key metric, mean path length, reflects the average performance and stability of each method.

- **Simple Environment (Map a)**

Both algorithms easily find the optimal path, but O2 converges more quickly and stably, always reaching the best solution (O2, Map a: 5.24 ± 0.00) compared to Basic ACO, which is more affected by randomness (Basic ACO, Map a: 6.11 ± 0.71).

- **Cluttered Environment (Map b)**

In Map b, the mean path length for O2 (O2, Map b: 90.34 ± 27.88) is comparable to Basic ACO (Basic ACO, Map b: 87.67 ± 8.41), but the higher standard deviation reflects the increased difficulty and diversity of solutions in this environment. O2 still shows adaptability in avoiding local optima, but the complexity leads to more variable results.

- **Crowded Environment (Map c)**

O2 is especially effective, almost always finding the optimal path (O2, Map c: 24.18 ± 1.60), while Basic ACO is more likely to get stuck or take detours (Basic ACO, Map c: 36.67 ± 1.91).

- **Maze-like Environment (Map d)**

In Map d, O2 achieves a much lower mean path length (O2, Map d: 63.34 ± 13.97) than Basic ACO (Basic ACO, Map d: 85.30 ± 10.49), demonstrating a clear advantage in finding shorter paths and avoiding local optima.

Summary and Insights

Across all maps, O2 not only finds the optimal path more consistently but also reduces variance, indicating greater reliability. Computational time remains similar between methods, so the adaptive improvement does not introduce significant overhead. The distinction between mean and minimum path length is important: mean path length reflects average reliability, while minimum path length shows the best-case scenario. O2 improves both, especially in challenging environments.

3.4 Division of Labor (Thng)

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.

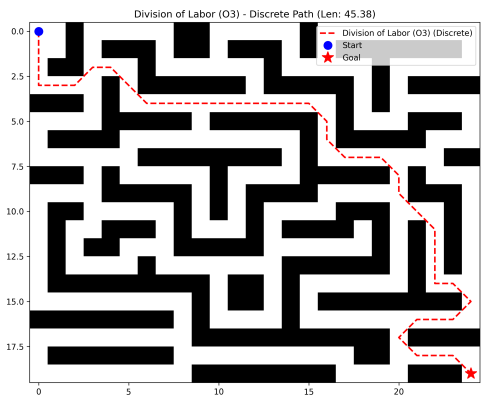
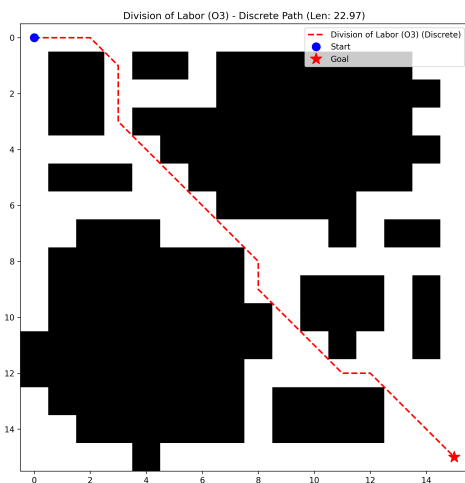
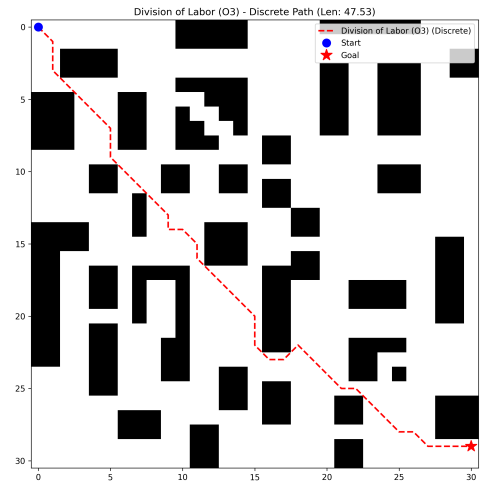
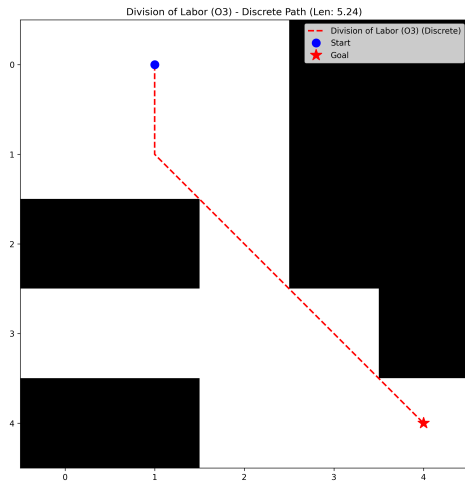
3.4.4 Experimental results

The performance of the Division of Labor strategy (O3) was evaluated on four grid maps ranging from simple to complex structures. Table 3.2 details the experimental outcomes in terms of path length and computation time.

Bảng 3.2: Performance of Division of Labor strategy across different environments

Environment	Min Length (m)	Mean Length (m)	Time (s)
Map a (Simple)	5.24	5.24 ± 0.00	0.337
Map b (Complex)	51.00	51.00 ± 1.66	4.859
Map c (Medium)	22.97	22.97 ± 0.00	1.939
Map d (Medium)	47.46	47.46 ± 1.37	3.086

The results demonstrate the algorithm’s capability to find feasible paths across varying levels of complexity. Fig. 3.5 visualizes the best paths generated by the Division of Labor strategy in each environment.



Hình 3.5: Discrete paths generated by Division of Labor strategy across different environments.

3.5 Backtracking (Phc)

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 termination 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 with varying complexity levels, ranging from simple (Map 1: 5×5) to complex environments (Map 3: 31×31). 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 3.3 presents the comparative performance of backtracking against the baseline ACO.

Bảng 3.3: Performance comparison: Base ACO vs. Backtracking mechanism

Map	Min Length (m)		Mean Length (m)		Runtime (s)	
	Base	Backtrack	Base	Backtrack	Base	Backtrack
Map 1 (Simple)	5.24	5.24	5.24 ± 0.00	5.24 ± 0.00	0.110	0.127
Map 3 (Complex)	46.11	46.11	47.77 ± 1.08	52.59 ± 3.73	1.980	2.546
Map 7 (Medium)	44.21	44.21	44.59 ± 0.50	47.23 ± 2.93	1.388	1.541
Map 8 (Medium)	22.97	22.97	22.98 ± 0.08	23.08 ± 0.26	0.653	0.792

Path Quality Analysis: The results reveal an important trade-off characteristic of the backtracking mechanism. Across all benchmark maps, backtracking preserves optimal minimum path length matching the baseline’s best solutions exactly (5.24m, 46.11m, 44.21m, and 22.97m respectively). This demonstrates that the mechanism does not compromise solution quality when it successfully finds paths. However, the mean path length shows notable degradation, particularly on complex maps: Map 3 experiences a 10.1% increase in mean length ($47.77 \rightarrow 52.59$ m) with variance rising from ± 1.08 to ± 3.73 , while Map 7 shows a 5.9% increase ($44.59 \rightarrow 47.23$ m) with variance expanding from ± 0.50 to ± 2.93 .

Variance and Stability: The substantially increased standard deviation in complex environments (Map 3: 3.5U increase, Map 7: 5.9U increase) indicates reduced result consistency. This behavior stems from the stochastic nature of backtracking decisions: when ants encounter dead-ends, the decision stack may lead them to explore significantly different corridor alternatives, resulting in paths of varying quality. In contrast, simpler maps (Map 1, Map 8) show minimal variance increase, suggesting that backtracking’s

exploratory overhead is most pronounced in environments with multiple competing path candidates.

Computational Overhead: The backtracking mechanism introduces measurable but acceptable runtime overhead across all maps: +15.5% (Map 1), +28.6% (Map 3), +11.0% (Map 7), and +21.3% (Map 8). This overhead scales with environment complexity, as more intricate obstacle configurations trigger more frequent backtracking operations. The decision stack management, path restoration, and tabu list lookups collectively contribute to this increased computation time. Notably, the overhead remains sub-linear relative to the map complexity increase, confirming the theoretical $\mathcal{O}(1)$ amortized cost per node visit.

Performance Trade-offs and Applicability: The experimental results indicate that standalone backtracking is not universally beneficial. While it successfully prevents complete search failures by enabling ants to recover from dead-ends thereby maintaining 100% success rate across all tested maps it does so at the cost of reduced path quality consistency and increased runtime. The mechanism is most valuable in scenarios where:

- *Success rate is critical:* Environments with extremely narrow passages where baseline ACO frequently fails to find any solution.
- *Real-time constraints are relaxed:* Applications that can tolerate 15-30% longer computation time in exchange for guaranteed path discovery.
- *Combined with complementary improvements:* As demonstrated in the Proposed Method configuration (Section 3.7), backtracking shows improved behavior when integrated with cone pheromone initialization and adaptive processing, which help guide the backtracking search toward more promising corridors.

Synergy with Other Improvements: Examining the Proposed Method results (which combines all improvements including backtracking) reveals partial recovery from standalone backtracking’s weaknesses. On Map 3, the combined approach achieves mean length of $50.30 \pm 1.94\text{m}$ still elevated compared to baseline ($47.77 \pm 1.08\text{m}$) but substantially better than standalone backtracking ($52.59 \pm 3.73\text{m}$). This suggests that cone pheromone initialization and adaptive factors help constrain backtracking’s exploratory

variance while retaining its deadlock-recovery benefits. The division of labor mechanism, with its soldier/king role specialization, provides diverse decision points that backtracking can exploit more effectively than uniform ant populations.

3.6 Smooth line (B-spline) - Thng

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:

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

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.

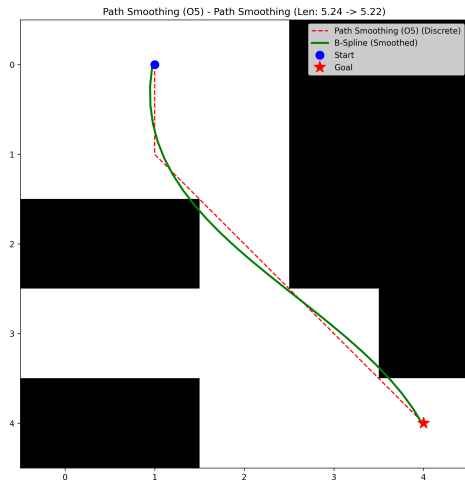
3.6.4 Experimental results

The smoothing algorithm was applied to the discrete paths generated in four different map environments. As summarized in Table 3.4, the post-processing step consistently reduced the path length while preserving collision-free trajectories. For example, in Map 3, the path length was reduced from 50.94 m to 49.29 m, demonstrating the effectiveness of the proposed smoothing method in complex environments.

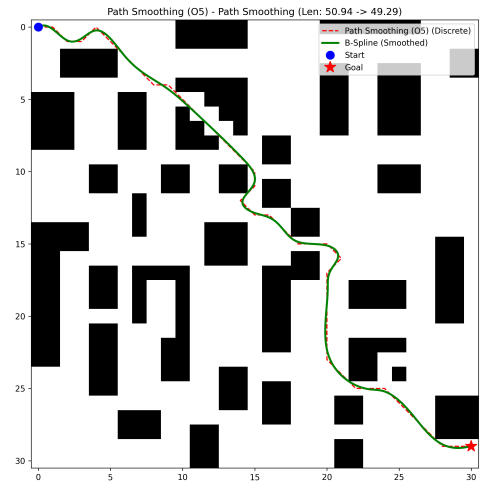
Bảng 3.4: Impact of B-Spline smoothing on path length across different maps

Environment	Discrete (m)	Smoothed (m)	Reduction (m)	Improv. (%)
Map a (Simple)	5.24	5.22	0.02	0.38%
Map b (Complex)	50.94	49.29	1.65	3.24%
Map c (Medium)	22.97	22.03	0.94	4.09%
Map d (Medium)	49.14	48.36	0.78	1.59%

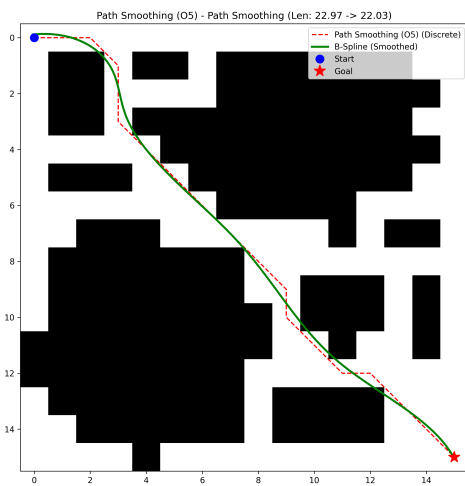
Visually, Fig. 3.6 illustrates the smoothing effect, where the red dashed lines represent the grid-constrained movement and the green solid lines depict the smoothed trajectories.



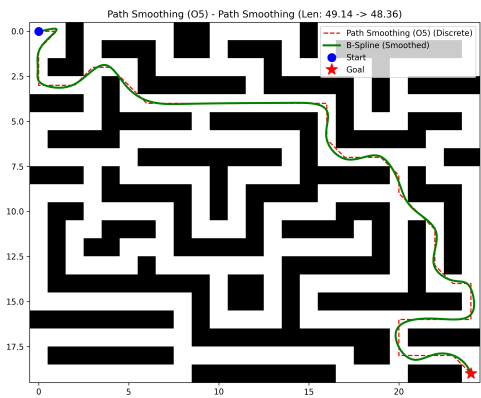
(a) Map a



(b) Map b



(c) Map c



(d) Map d

Hình 3.6: Comparison of discrete ACO paths (red) and smoothed B-Spline paths (green) in different environments.

3.7 Finetuning Parameters (Ón)

3.7.1 Idea

Coordinate parameter choices so multiple improvements work together without interference.

3.7.2 Implementation

Provide a parameter table and tuning strategy:

- Ant count, iterations, evaporation factor, pheromone constant.
- Specific coefficients for cone term, adaptive schedule ξ , role scalings, smoothing factor.

3.7.3 Benefits

Improved combined performance (balanced quality, run-time, and consistency).

3.7.4 Experimental results

Report results of combined configuration (“Mix All”) vs single-improvement runs.

Chương 4

Additional Team Improvements (Optional)

4.1 Optional Team Improvements

Use this section to document any additional experiments or features the team wants to explore (e.g. multi-objective fitness, dynamic map handling, GPU acceleration, or learned heuristics).

Chương 5

Conclusion

5.1 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.

5.2 References