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 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: 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 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 the set of benchmark maps provided in the project repository (see the `maps/` folder). The benchmark collection contains several ASCII grid maps (e.g. `map-s/map1.txt` through `maps/map5.txt`) used throughout the experiments in this work. Each map encodes free space, obstacles, and the task endpoints in a compact, human-readable format.

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.

Evaluation protocol and metrics:

- **Repetitions:** Each algorithm/configuration is executed multiple times per map to capture stochastic variability. In the experiments reported in this paper we used `RUNS=30` (see the benchmarking script `aco_enhancement/benchmark_aco.py`).
- **Recorded metrics:** For each run we record whether a feasible path from `S` to `F` was found, the total path length (computed as the sum of Euclidean edge lengths along the returned route), and the runtime until termination or success. From the repeated runs we report the success rate, and 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 time or iteration budget used in the experiments. Beyond feasibility, we evaluate solution quality using a composite objective that balances three factors: (i) path optimality (shorter total path length), (ii) result stability (low variability across repeated runs, measured by the standard deviation of path length), and (iii) convergence speed (lower average runtime or fewer iterations to convergence). For algorithm comparisons we normalize each metric and report a weighted composite score (default weights used in the experiments: length = 0.5, stability = 0.3, speed = 0.2); alternative weightings are examined in ablation studies to illustrate trade-offs between shortest-path performance, repeatability, and speed.
- **Reproducibility:** All experiments are executed with explicit random seeds when possible; the map files and the code used to parse them are included in the repository so results can be reproduced exactly.

Experiment configuration used in benchmarking

- **Benchmark script:** Parameters used by `aco_enhancement/benchmark_aco.py` are representative of the paper experiments.
- **Key parameters:** `RUNS=30`, `NO_ANTS=50`, `EVAPORATION=0.15`, `ITERATIONS=100`, `INIT_PHER=1e-4`. These values are set in the benchmarking

script and may be adjusted for ablation studies.

- **Indexing / plotting note:** The map and path coordinates in the code use `(row, column)` indexing. When plotting or converting to continuous coordinates the code maps `row` to the y axis and `column` to the x axis (see smoothing/plotting in the benchmarking script).

Notes and usage:

- The provided maps serve as compact benchmarks for algorithm comparison rather than large-scale real-world environments. If additional quantitative map statistics are required (map dimensions, obstacle density, shortest-manual-path baseline), we can compute and include those in the report and/or a supplementary table.
- Figures of each ASCII map (visualized occupancy grids with start/goal markers) can be generated from the parsing script and embedded in the paper to aid reproducibility and clarity.

This section documents the input maps and the evaluation protocol used to benchmark the ant-colony algorithms presented in this work.

3.2 Cone Pheromone (Ân)

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_co`
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 (Như)

3.3.1 Idea

Dynamically adapt α and β during iterations to encourage exploration early and exploitation later.

3.3.2 Implementation

Use a smooth schedule (e.g. quadratic/integral schedule):

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

Tune ξ to control adaptation strength.

3.3.3 Benefits

Reduces premature convergence and balances exploration/exploitation.

3.3.4 Experimental results

Include comparison tables/plots showing effect of adaptive scheduling on path quality and variance.

3.4 Division of Labor (Thương)

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.

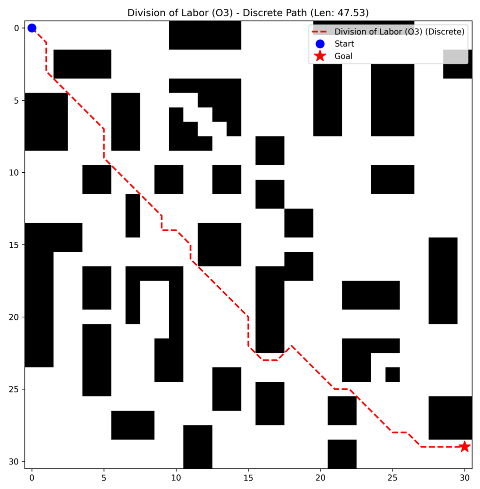
3.4.4 Experimental results

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

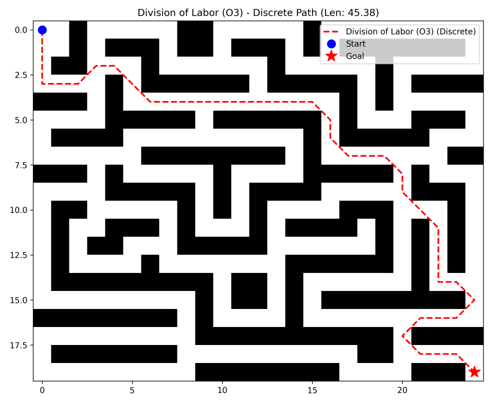
Bảng 3.1: Performance of Division of Labor strategy across different environments

Environment	**Min Length (m)**	**Mean Length (m)**	**Time (s)**
Map 1 (Simple)	5.24	5.24 ± 0.00	0.337
Map 3 (Complex)	51.00	51.00 ± 1.66	4.859
Map 7 (Medium)	47.46	47.46 ± 1.37	3.086
Map 8 (Medium)	22.97	22.97 ± 0.00	1.939

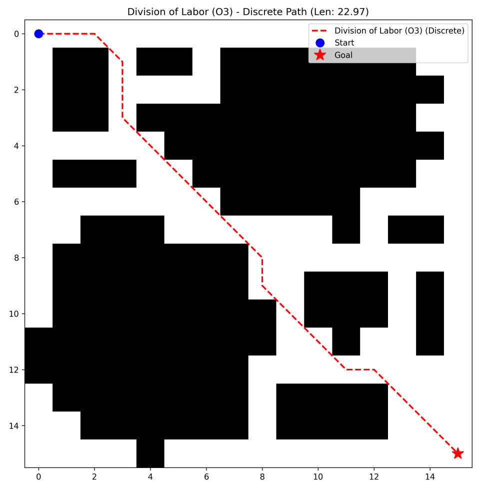
The results demonstrate the algorithm's capability to find feasible paths across varying levels of complexity. Fig. **??** visualizes the best paths generated by the Division of Labor strategy in each environment.



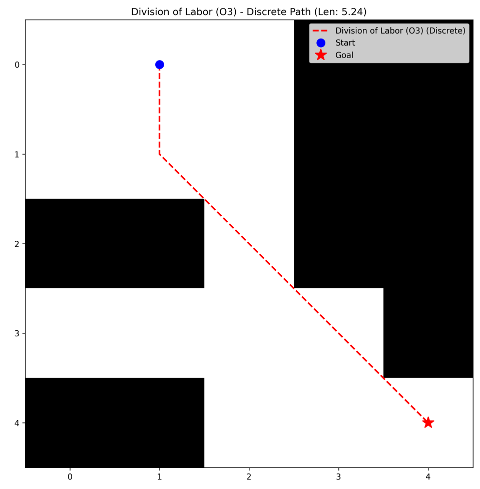
(a) Map 3



(b) Map 7



(c) Map 8



(d) Map 1

Hình 3.1: Discrete paths generated by Division of Labor strategy across different environments.

3.5 Backtracking (Phước)

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 **??** presents the comparative performance of backtracking against the baseline ACO.

Bảng 3.2: 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: 3.5× increase, Map 7: 5.9× 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 ± 1.94m—still elevated compared to baseline (47.77 ± 1.08m) but substantially better than standalone backtracking (52.59 ± 3.73m). 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) - Thương

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.

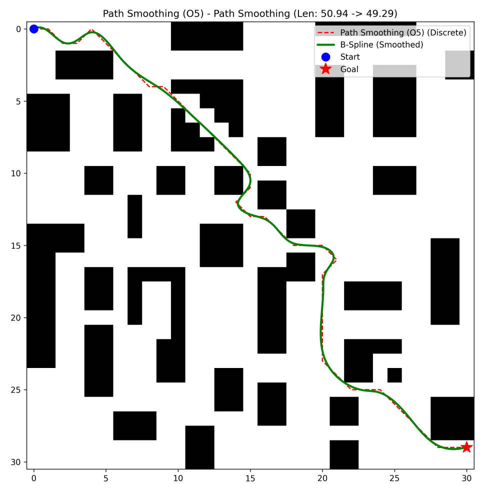
3.6.4 Experimental results

The smoothing algorithm was applied to the discrete paths generated in four different map environments. As summarized in Table **??**, 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.

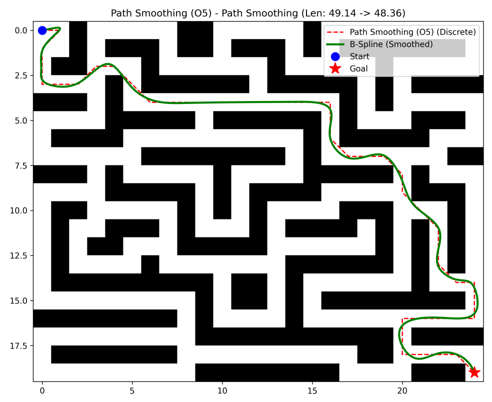
Bảng 3.3: Impact of B-Spline smoothing on path length across different maps

Environment	**Discrete (m)**	**Smoothed (m)**	**Reduction (m)**	**Improv. (%)**
Map 1 (Simple)	5.24	5.22	0.02	0.38%
Map 3 (Complex)	50.94	49.29	1.65	3.24%
Map 7 (Medium)	49.14	48.36	0.78	1.59%
Map 8 (Medium)	22.97	22.03	0.94	4.09%

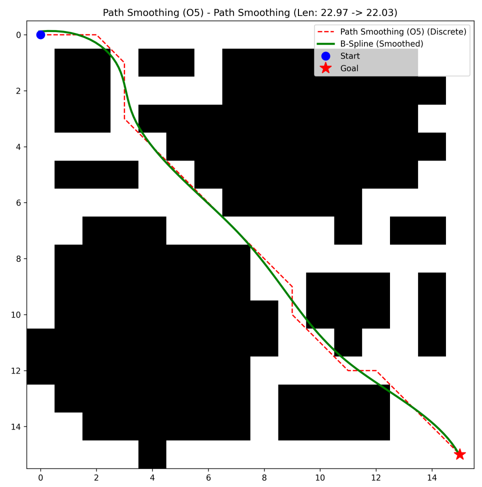
Visually, Fig. **??** illustrates the smoothing effect, where the red dashed lines represent the grid-constrained movement and the green solid lines depict the smoothed trajectories.



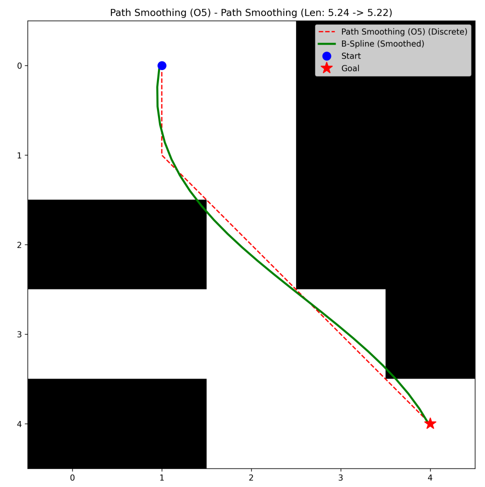
(a) Map 3



(b) Map 7



(c) Map 8



(d) Map 1

Hình 3.2: Comparison of discrete ACO paths (red) and smoothed B-Spline paths (green) in different environments.

3.7 Finetuning Parameters (Ân)

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

Tài liệu tham khảo

[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] P. E. Hart, N. J. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

[4] Thomas Stützle and Holger H. Hoos. “A Max–Min Ant System for Combinatorial Optimization”. In: *Future Generation Computer Systems* 16.8 (2000), pp. 889–914.