Enhancing the Efficiency of Ant Colony Optimization: Comparative Analysis of Optimizations and Multi-Pheromone Strategies

Tommy M. Bennett Conrad O. Voigt

December 7, 2024

Abstract

Ant Colony Optimization (ACO) is a widely used metaheuristic algorithm for solving complex combinatorial optimization problems, such as the Traveling Salesman Problem (TSP) and network routing. While effective, standard ACO implementations often face challenges like premature convergence, inefficiency, and sensitivity to parameter settings. This paper investigates several enhancements to ACO, including adaptive multi-pheromone systems, dynamic parameter tuning, and hybrid strategies. Five configurations—ranging from a baseline implementation to an optimized hybrid approach—are evaluated on a network routing problem with 50 nodes.

The results highlight the trade-offs between exploration and exploitation across different configurations. Adaptive mechanisms and hybrid strategies significantly improve performance, with the hybrid implementation achieving the best results, including superior solution quality, fastest convergence (0.50 iterations on average), and lowest runtime (34.00 seconds). These findings demonstrate the potential of advanced ACO configurations for addressing complex optimization challenges, offering valuable insights for future research and real-world applications in fields such as logistics, telecommunications, and traffic management.

1 Introduction

Optimization problems are central to many fields, including logistics, telecommunications, and artificial intelligence. These problems often involve vast, complex solution spaces that challenge traditional methods like gradient-based optimization, which can struggle with issues such as local optima and computational inefficiency. Metaheuristic algorithms have emerged as powerful tools for navigating such challenges, offering flexibility and robustness in diverse scenarios.

Ant Colony Optimization (ACO), inspired by the foraging behavior of ants, has been particularly successful in solving combinatorial optimization problems like the Traveling Salesman Problem (TSP) and Vehicle Routing Problem (VRP). By utilizing probabilistic solution construction and pheromone-guided learning, ACO effectively balances exploration and exploitation, making it adaptable to a wide range of applications. However, ACO's performance can be hindered by issues such as premature convergence and sensitivity to parameter settings.

This paper explores advancements in ACO, focusing on adaptive parameter tuning, multi-pheromone systems, and hybrid strategies to address its limitations. Five configurations—ranging from a baseline implementation to a hybrid model combining multiple enhancements—are evaluated on a network routing problem using metrics such as solution quality, convergence speed, and computational efficiency.

2 Literature Review

2.1 Overview of Metaheuristic Algorithms

Metaheuristic algorithms have emerged as powerful tools for solving complex optimization problems across various domains. These algorithms, including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and Simulated Annealing (SA), are designed to explore and exploit the solution space effectively. The adaptability and robustness of metaheuristic algorithms make them suitable for a wide range of applications, from engineering to artificial intelligence [5, 8]. The fundamental principle behind these algorithms is to mimic natural processes or social behaviors, allowing them to escape local optima and converge towards global solutions [4].

The variety of metaheuristic algorithms stems from their ability to handle different types of optimization problems, including continuous, discrete, and combinatorial optimization [7]. Each algorithm has its unique mechanisms for searching the solution space, which can include population-based strategies, local search techniques, and hybrid approaches that combine multiple methods [3]. The effectiveness of these algorithms often hinges on their parameter settings, which significantly influence their performance [6]. Consequently, researchers have focused on developing adaptive mechanisms that allow these parameters to adjust dynamically based on the problem characteristics, thereby enhancing the algorithms' efficiency and robustness [2].

2.2 Ant Colony Optimization in Network and Routing Problems

Ant Colony Optimization (ACO) is a prominent metaheuristic inspired by the foraging behavior of ants, particularly their ability to find the shortest paths to food sources. This algorithm has been successfully applied to various network and routing problems, including the Traveling Salesman Problem (TSP) and vehicle routing problems [1]. Foundational works by Dorigo et al. have established ACO's utility in finding near-optimal solutions, demonstrating its effectiveness in scenarios characterized by dynamic and complex environments [8].

The strength of ACO lies in its use of pheromone trails, which guide the search process by reinforcing paths that yield better solutions. This collective behavior mimics the natural process of pheromone deposition and evaporation, allowing the algorithm to balance exploration and exploitation effectively [1]. ACO has been shown to outperform traditional optimization methods in various applications, particularly in telecommunications and logistics, where routing efficiency is critical [1]. The adaptability of ACO to different problem domains makes it a versatile tool in the optimization landscape [7].

2.3 Optimizations in ACO

2.3.1 Pheromone Update Techniques

One of the critical areas of research in ACO is the development of pheromone update techniques, which are essential for enhancing the algorithm's performance. Techniques such as adaptive evaporation rates have been proposed to improve the balance between exploration and exploitation [1]. By dynamically adjusting the pheromone levels based on the quality of solutions found, these methods can prevent premature convergence and encourage the exploration of new paths [1].

2.3.2 Parameter Adaptation in Metaheuristics

Parameter adaptation is another significant aspect of optimizing ACO. The influence of parameters such as alpha (which controls the pheromone importance) and beta (which controls the heuristic information importance) is crucial in determining the convergence rate of the algorithm [3]. Adaptive mechanisms that adjust these parameters in real-time based on the search progress have shown promising results in improving the overall efficiency of ACO [8]. Such adaptations allow the algorithm to respond to the changing landscape of the optimization problem, thereby enhancing its robustness and effectiveness [4].

2.4 Multi-Pheromone Approaches

The introduction of multi-pheromone systems in ACO has further advanced its capabilities in solving complex optimization problems. By employing multiple pheromone types, these approaches can enhance the diversity of solutions generated by the algorithm [1]. Multi-pheromone strategies allow for the simultaneous exploration of different solution paths, which can lead to improved solution quality and convergence speed [2].

Research has demonstrated that multi-pheromone systems can significantly outperform traditional single-pheromone ACO in various applications, particularly in scenarios where the solution space is highly complex and multi-modal [2]. The ability to maintain multiple pheromone trails enables the algorithm to explore various regions of the solution space concurrently, thus increasing the likelihood of finding optimal or near-optimal solutions [1]. This approach has been particularly beneficial in routing problems, where diverse solution paths can lead to more efficient network configurations [8].

2.5 Hybrid Optimization Strategies

Hybrid optimization strategies that combine ACO with other algorithms or enhancements have shown significant improvements in complex routing scenarios. By integrating ACO with techniques such as Genetic Algorithms or Particle Swarm Optimization, researchers have developed hybrid models that leverage the strengths of each method [3]. These hybrid approaches can enhance the exploration capabilities of ACO while maintaining its effective exploitation of promising solutions [5].

For instance, combining ACO with PSO can lead to a more robust search process, where the swarm intelligence of PSO complements the pheromone-based guidance of ACO [5]. Such hybrid models have been successfully applied to various optimization problems, including job scheduling, resource allocation, and network design, demonstrating their versatility and effectiveness [7]. The adaptability of these hybrid strategies allows them to tackle a wide range of optimization challenges, making them a valuable addition to the metaheuristic toolkit [6].

In conclusion, the field of metaheuristic algorithms, particularly ACO, has seen significant advancements in recent years. The continuous exploration of optimization techniques, parameter adaptations, and hybrid strategies has contributed to the development of more efficient and robust algorithms capable of addressing complex real-world problems. As research progresses, the integration of adaptive mechanisms and multi-pheromone approaches will likely play a crucial role in enhancing the performance of ACO and other metaheuristic algorithms.

3 Methodology

3.1 Ant Colony Optimization (ACO)

Ant Colony Optimization (ACO) is a probabilistic metaheuristic algorithm inspired by the foraging behavior of real ants. This technique is particularly effective for solving combinatorial optimization problems such as the Traveling Salesman Problem (TSP), Vehicle Routing Problem (VRP), and others. The algorithm mimics how ants deposit pheromones on paths while searching for food, which helps them collectively discover and reinforce the shortest paths between their nest and food sources.

In ACO, artificial ants are used to iteratively build candidate solutions to a given problem by moving through a graph representation of the problem. Each ant probabilistically chooses the next node to visit based on a combination of pheromone levels and heuristic information associated with the edges of the graph. The probability of an ant transitioning from node i to node j is determined by the following equation:

$$P_{ij} = \frac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{k \in N_i} (\tau_{ik})^\alpha (\eta_{ik})^\beta}$$

where:

- P_{ij} is the probability of an ant moving from node i to node j.
- τ_{ij} is the pheromone level on the edge (i, j). This represents the collective memory of previous solutions that have traversed this edge.
- η_{ij} is the experienced information for the edge (i, j), often defined as $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance or cost associated with traveling between nodes i and j.
- α is a parameter controlling the influence of the pheromone trail. Higher α values give more weight to pheromone levels in determining path probabilities.
- β is a parameter controlling the influence of heuristic information. Higher β values emphasize the heuristic desirability of edges.
- N_i is the set of all feasible nodes that an ant can move to from node i, ensuring that constraints like visiting each node only once are respected.

The Ant Colony Optimization (ACO) algorithm operates through the following main steps: initialization, solution construction, pheromone update, solution refinement, and the application of stopping criteria. These steps, inspired by the foraging behavior of real ants, form the foundation of a robust method for solving complex optimization problems.

In the **initialization** phase, the algorithm begins by setting the pheromone levels τ_{ij} on all edges to a small positive value, typically denoted as τ_0. This initialization ensures that all edges are initially considered feasible, promoting an unbiased exploration of the solution space and preventing premature convergence to suboptimal paths. Additionally, critical algorithm parameters are set during this phase, including the number of ants (m), the maximum number of iterations, the relative influence of pheromones (α) and heuristic information (β), the pheromone evaporation rate (ρ), and the pheromone deposit factor (Q). These parameters play a pivotal role in balancing the exploration of new solutions and the exploitation of known high-quality solutions, directly impacting the algorithm's efficiency and effectiveness.

The **solution construction** phase involves each ant independently building a candidate solution by traversing the problem's graph. Each ant begins its journey at a randomly selected node or a predefined starting point. At each step, the ant chooses the next node to visit based on a probabilistic transition rule, where the probability of selecting a specific edge is influenced by the pheromone level on that edge and the associated heuristic information. This probabilistic approach introduces diversity in the solutions generated by different ants while guiding the search process toward promising regions of the solution space.

Once all ants have constructed their solutions, the algorithm proceeds to the pheromone update phase. This phase consists of two key processes: pheromone evaporation and pheromone deposition. During **pheromone evaporation**, a fraction of the pheromone on each edge evaporates, reducing its influence over time. This process, controlled by the evaporation rate ρ, is described by the equation $\tau_{ij} \leftarrow (1-\rho)\tau_{ij}$, where $0 \leq \rho \leq 1$. Evaporation serves as a critical mechanism for avoiding premature convergence by preventing the overemphasis of heavily traversed edges, thereby encouraging exploration of alternative paths.

In the **pheromone deposition** process, ants reinforce the edges they have traversed by depositing pheromone in amounts proportional to the quality of their solutions. For an ant k with a solution cost L_k, the amount of pheromone deposited on an edge (i, j) is given by:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k}, & \text{if ant } k \text{ used edge } (i,j), \\ 0, & \text{otherwise.} \end{cases}$$

The total pheromone level on each edge is updated by summing the contributions from all ants, as expressed by

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k.$$

This positive feedback mechanism amplifies the pheromone levels on edges associated with high-quality solutions, effectively guiding subsequent ants toward these promising paths.

As the algorithm iterates, the concentration of pheromone trails on certain edges leads to a convergence of solutions. However, to prevent stagnation, where all ants follow the same path prematurely, additional exploration mechanisms may be employed. These mechanisms include introducing random noise or biases to encourage ants to explore less-traveled paths, thereby maintaining diversity in the solution space.

The ACO algorithm terminates when one of the predefined **stopping criteria** is met. These criteria may include reaching a maximum number of iterations, achieving a solution of satisfactory quality, or observing stabilization in the quality of solutions over several iterations. By balancing the processes of exploration and exploitation through its iterative design, the ACO algorithm efficiently identifies high-quality solutions to complex optimization problems, demonstrating its adaptability and robustness in various applications.

Overall, the ACO has three key features. Firstly, positive pheromone reinforcement ensures that better solutions receive more emphasis over time. It also distributes the computation through the use of multiple ants, which reduces the likelihood of getting stuck in local optima. The combination of heuristic desirability and stochastic exploration through pheromone trails balances exploration and exploitation of the solution space. Ant Colony Optimization has been successfully applied to a wide range of optimization problems and can be extended with techniques like elitist strategies, hybrid algorithms, and dynamic parameter tuning for improved performance.

3.2 Optimizations Implemented

3.2.1 Standard ACO without Optimizations

The baseline ACO algorithm was implemented as a reference to evaluate the effectiveness of the proposed optimizations. In this standard implementation, pheromone trails are updated using a fixed evaporation rate and constant pheromone deposition.

3.2.2 ACO with Optimizations

To improve the convergence speed and solution quality, we introduced two optimizations.

First, the pheromone update rules were modified to emphasize shorter paths by increasing the pheromone deposition inversely proportional to the path length.

Secondly, the pheromone evaporation rate was adapted dynamically based on the pheromone entropy. Lower entropy led to increased evaporation to encourage exploration, while higher entropy led to decreased evaporation to exploit promising regions.

3.2.3 Adaptive Multi-Pheromone System

In the adaptive multi-pheromone system, multiple pheromone trails were used to maintain solution diversity.

Specifically we used primary and secondary pheromone levels so that each edge in the graph had two pheromone levels. The primary pheromone focused on the standard ACO process, while the secondary pheromone introduced additional diversity, promoting alternative paths.

The influence of the secondary pheromone trail was regulated by pheromone weighting to ensure a balance between exploration and exploitation.

3.2.4 Dynamic Alpha-Beta Parameters

For the fourth variation, the parameters α and β were adjusted dynamically during the simulation. Initially, higher exploration was encouraged with a higher β value. As iterations progressed, the values of α and β were adjusted based on two criteria.

If solution or path quality improved, α was increased to emphasize pheromone influence, while β was decreased to reduce the reliance on heuristic information.

Additionally, the entropy of pheromone levels was monitored. Higher entropy led to higher α to focus on exploiting known good paths, while lower entropy encouraged more exploration by adjusting β.

3.2.5 Hybrid Implementation

The hybrid approach combined multiple of the above optimization strategies to further improve the performance of ACO. First, both adaptive multi-pheromone and dynamic alpha-beta parameter strategies were combined to enhance solution diversity and convergence.

Additionally, a local search heuristic was applied to refine the solutions found by ants. This involved reversing segments of the path to potentially reduce the total path length, especially in later iterations where promising solutions were identified.

4 Experimental Setup

4.1 Dataset and Simulation Environment

The problem considered in this study involves a network routing scenario with 50 nodes, represented as a graph where nodes correspond to network devices, and edges represent communication links. The graph structure was designed to reflect realistic routing challenges, including varying edge weights to simulate different communication costs (e.g., latency, bandwidth, or distance).

To ensure the simulation environment closely resembles practical scenarios, we generated a network graph using a random geometric graph model, ensuring nodes are spatially distributed, and edges are defined based on proximity and weighted by communication cost.

The edge weights were assigned based on a uniform distribution to mimic diverse routing costs. For the ACO, the initial pheromone levels, heuristic weight (α), pheromone weight (β), evaporation rate (ρ), and total number of ants were carefully tuned for optimal performance.

Each experiment was run with 10 ants, 1000 iterations, and on 10 different graph arrangements, to account for the stochastic nature of the algorithm and ensure statistically significant results.

The experiment was implemented in Python, using a simulation framework optimized for graph-based problems. All computations were performed on an HP Envy Laptop with AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx @ 2.10 GHz, and 8.00 GB of RAM.

4.2 Evaluation Metrics

To evaluate the performance of the different ACO algorithm variants, best path length, convergence iterations, average path length over iterations pheromone entropy and runtime in seconds were used to compare the implementations. Each metric was averaged over 50 independent runs to mitigate randomness and provide a reliable assessment.

The **best path length** is the shortest path discovered by the algorithm in each run, representing the optimal solution for the given graph instance. It measures the quality of the final solution produced by ACO. A smaller path length indicates better performance.

The **convergence iterations** is the number of iterations required for the algorithm to converge to the best solution, where no significant improvement is observed in subsequent iterations. It evaluates the algorithm's efficiency in finding an optimal solution. Faster convergence indicates better exploitation of the search space.

The **average path length over iterations** is the mean path length discovered by all ants across all iterations of the algorithm. It provides insight into the algorithm's exploration-exploitation trade-off. Lower average path lengths indicate that most ants are converging towards optimal or near-optimal solutions.

Pheromone entropy is measure of the diversity of pheromone levels on the edges, defined as:

$$H = -\sum_{(i,j)} p_{ij} \log p_{ij}$$

where p_{ij} is the normalized pheromone level on edge (i, j). High entropy indicates greater exploration, while low entropy suggests convergence. It tracks the balance between exploration and exploitation during the algorithm's execution.

The **runtime** is the total time taken by the algorithm to complete all iterations in each run. it assesses the computational efficiency of the algorithm, which is critical for real-time or large-scale applications.

5 Results

The results of the experiments are summarized in Table 1. The performance of the five configurations of the ACO algorithm was evaluated across five key metrics, as discussed above. Each configuration was tested over 50 runs, and the averages of these runs are reported in the table. The analysis of each configuration and the corresponding observations are detailed below.

Table 1: Average Metrics for ACO Configurations

Configuration	**Best Path**	**Convergence**	**Avg. Path Length**	**Pheromone Entropy**	**Runtime (s)**
ACO without Optimizations	3.70	1.90	13.08	2.21	46.17
ACO with Optimizations	3.50	2.20	12.09	2.10	38.93
ACO with Adaptive MPS	5.60	1.60	17.58	2.69	68.32
ACO with Adaptive MPS and Dynamic Alpha-Beta	3.70	1.60	9.30	2.05	36.41
ACO with Hybrid Implementation	3.20	0.50	9.95	1.55	34.00

5.1 Baseline: ACO Without Optimizations

The baseline implementation of ACO served as the control configuration, achieving an average best path length of 3.70 and requiring 1.90 iterations to converge. The average path length over iterations was 13.08, with a pheromone entropy of 2.21. However, the runtime was relatively high, averaging 46.17 seconds. These results indicate that while the baseline configuration provided reasonable path quality, it was less efficient compared to other configurations, particularly in terms of runtime. The routing graph of this implementation can be seen in Figure 1.

5.2 ACO with Optimizations

Incorporating optimizations into the ACO algorithm led to noticeable improvements across most metrics. The best path length improved slightly to 3.50, indicating marginally better solution quality. The average path length over iterations reduced to 12.09, reflecting enhanced exploration-exploitation dynamics. Despite requiring more convergence iterations (2.20 compared to 1.90 in the baseline), the overall runtime decreased significantly to 38.93 seconds. This suggests that the optimizations improved computational efficiency while maintaining solution quality, as seen in the reduction of used edges in Figure 2.

5.3 ACO with Adaptive Multi-Pheromone System (MPS)

The adaptive multi-pheromone system configuration exhibited a notable increase in the best path length to 5.60 and an average path length of 17.58. This indicates that the configuration struggled to maintain solution quality compared to other implementations. Additionally, the runtime increased significantly to 68.32 seconds, the highest among all configurations. The high pheromone entropy of 2.69, however, reflects greater exploration during the search process. While the adaptive MPS improved diversity in pheromone distribution, it came at the cost of both efficiency and solution quality, compare Figure 3 as well.

5.4 ACO with Adaptive MPS and Dynamic Alpha-Beta

Adding dynamic adjustment of the α and β parameters to the adaptive MPS configuration resulted in substantial improvements. The best path length returned to 3.70, matching the baseline. The average path length was reduced significantly to 9.30, and the pheromone entropy decreased to 2.05, indicating more focused convergence. Additionally, the runtime decreased to 36.41 seconds, suggesting that the dynamic parameter adjustment improved the efficiency of the adaptive system, see Figure 4.

5.5 ACO with Hybrid Implementation

The hybrid implementation achieved the best overall results across all metrics. The best path length improved to 3.20, the lowest (best) among all configurations, indicating superior solution quality. The average path length was reduced to 9.95, reflecting consistent optimization throughout iterations. This configuration also converged in just 0.50 iterations on average, the fastest among all configurations, indicating highly efficient exploitation. Moreover, the runtime was the lowest, at 34.00 seconds, and the pheromone entropy of 1.55 reflects a balanced approach to exploration and exploitation. These results highlight the hybrid implementation as the most effective and efficient configuration, also compare the routing graph in Figure 5.

5.6 Comparison of Configurations

Among all configurations above, the hybrid implementation consistently outperformed all other methods across most categories, achieving the best solution quality with a best path length of 3.20, the fastest convergence at just 0.50 iterations on average, and the lowest runtime of 34.00 seconds. These results highlight its ability to efficiently exploit high-quality solutions while minimizing computational overhead.

The optimized ACO configuration demonstrated a clear improvement over the baseline, offering reduced runtime and slightly better path quality. In contrast, the adaptive multi-pheromone system (MPS) increased exploration, as reflected by the higher pheromone entropy, but this came at the cost of decreased solution quality and efficiency. Introducing dynamic alpha-beta parameters to the adaptive MPS addressed many of these issues by improving the balance between exploration and exploitation, leading to reduced average path lengths, faster convergence, and lower runtime compared to the standalone adaptive MPS.

These findings demonstrate the trade-offs inherent in different configurations. While baseline ACO and simpler optimizations may be sufficient for smaller problems or less dynamic environments, advanced methods such as hybrid implementations are better suited for achieving both efficiency and high-quality solutions in complex scenarios.

6 Discussion

The inclusion of multi-pheromone systems and dynamic adjustment of alpha-beta parameters had significant implications for the performance of the algorithm. Multi-pheromone systems allowed for greater exploration of the solution space by maintaining diversity in pheromone levels, which helped mitigate premature convergence to suboptimal solutions. However, without proper balancing, this increased diversity often led to inefficiency and poorer solution quality, as seen in the standalone adaptive MPS configuration.

Dynamic alpha-beta parameters, on the other hand, enabled the algorithm to adapt its search strategy over time, transitioning smoothly between exploration in the early stages and exploitation in later stages. This adaptation was particularly effective when combined with the multi-pheromone approach, as it allowed the algorithm to use the benefits of diversity while maintaining focus on high-quality solutions. The hybrid implementation leveraged both of these strategies, achieving the best balance and, consequently, the highest performance across all metrics.

Despite the promising results, the current implementation has several limitations. First, computational constraints restricted the experiments to graphs with a maximum of 50 nodes, leaving the scalability of the proposed methods untested on larger and more complex networks. Additionally, the stochastic nature of the algorithm introduces variability in performance, which may require fine-tuning of parameters for different problem instances. Finally, while the proposed enhancements improved performance significantly, their real-world applicability would benefit from further validation in dynamic and uncertain environments, such as traffic systems or telecommunications networks.

7 Conclusion

This paper presented a comprehensive evaluation of several enhancements to the classical Ant Colony Optimization (ACO) algorithm, including adaptive multi-pheromone systems, dynamic alpha-beta parameter adjustment, and a hybrid approach that combined these strategies. The baseline ACO provided a reasonable performance benchmark but was outperformed in terms of both runtime and solution quality by configurations with optimizations. The standalone multi-pheromone system demonstrated increased exploration but suffered from inefficiency and reduced solution quality. Dynamic alpha-beta parameters significantly improved the balance between exploration and exploitation, particularly when integrated with the multi-pheromone system.

The hybrid implementation emerged as the most effective configuration, achieving superior solution quality with the best path length of 3.20, fastest convergence at 0.50 iterations, and lowest runtime of 34.00 seconds. These results highlight the potential of hybrid strategies to address the inherent trade-offs in ACO by leveraging the strengths of multiple approaches.

Future research could focus on further hybridizing ACO with other optimization algorithms, such as genetic algorithms or particle swarm optimization, to explore synergies and enhance performance further. Additionally, extending these methods to larger and more complex graphs would provide valuable insights into their scalability and robustness. Another promising direction is the application of these techniques to dynamic and real-time problems, such as adaptive traffic routing, wireless network optimization, and logistics systems under uncertain conditions. Developing more efficient implementations to reduce computational overhead in high-dimensional scenarios would also be a critical area for exploration.

The proposed optimizations to ACO have significant practical implications. By improving both solution quality and computational efficiency, these methods are well-suited for real-world applications in areas such as traffic routing, where minimizing travel time and congestion is critical; telecommunications, where efficient network routing can enhance bandwidth utilization and reduce latency; and logistics, where optimizing delivery routes can lead to substantial cost savings. Moreover, the ability to adaptively balance exploration and exploitation makes these methods particularly relevant for dynamic environments, ensuring their applicability across a wide range of industries.

References

[1] A. Afia, S. Bouzbita, and R. Faizi, "The effect of updating the local pheromone on ACS performance using fuzzy logic," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 7, no. 4, p. 2161, 2017. `https://doi.org/10.11591/ijece.v7i4.pp2161-2168`

[2] E. Bernal, O. Castillo, J. Soria, and F. Valdez, "Fuzzy galactic swarm optimization with dynamic adjustment of parameters based on fuzzy logic," *SN Computer Science*, vol. 1, no. 1, 2020. `https://doi.org/10.1007/s42979-020-0062-4`

[3] S. Chen, A. Bolufé-Röhler, J. Montgomery, and T. Hendtlass, "An analysis on the effect of selection on exploration in particle swarm optimization and differential evolution," in *Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3037–3044, 2019. `https://doi.org/10.1109/cec.2019.8790200`

[4] A. Eiben and S. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011. `https://doi.org/10.1016/j.swevo.2011.02.001`

[5] S. Mirjalili, S. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014. `https://doi.org/10.1016/j.advengsoft.2013.12.007`

[6] M. Tessari and G. Iacca, "Reinforcement learning based adaptive metaheuristics," in *Proceedings of the 2022 Genetic and Evolutionary Computation Conference Companion*, pp. 1854–1861, 2022. `https://doi.org/10.1145/3520304.3533983`

[7] M. Ulukök, "Bi-attempted based optimization algorithm for numerical optimization problems," *European Journal of Science and Technology*, 2021. `https://doi.org/10.31590/ejosat.953349`

[8] J. Xu and L. Xu, "Optimal stochastic process optimizer: a new metaheuristic algorithm with adaptive exploration-exploitation property," *IEEE Access*, vol. 9, pp. 108640–108664, 2021. `https://doi.org/10.1109/access.2021.3101939`

Appendices

Figures

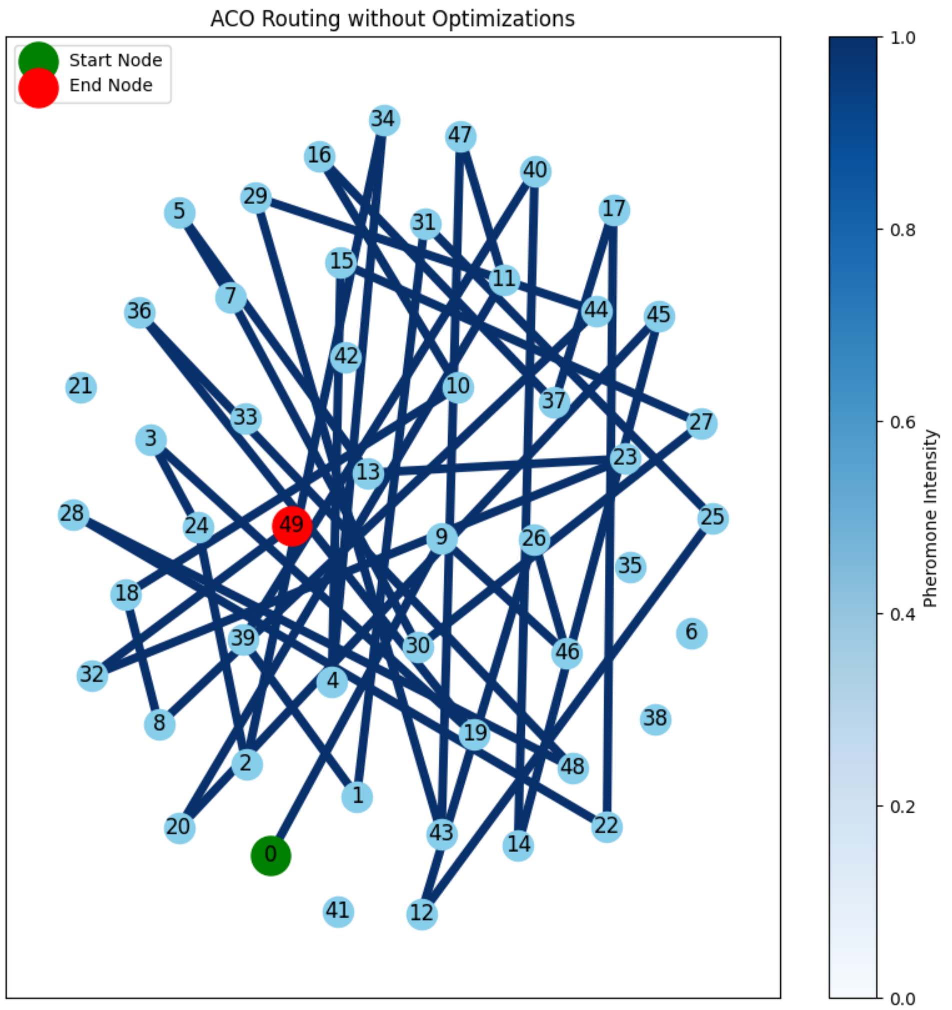

Figure 1: ACO Routing without Optimizations

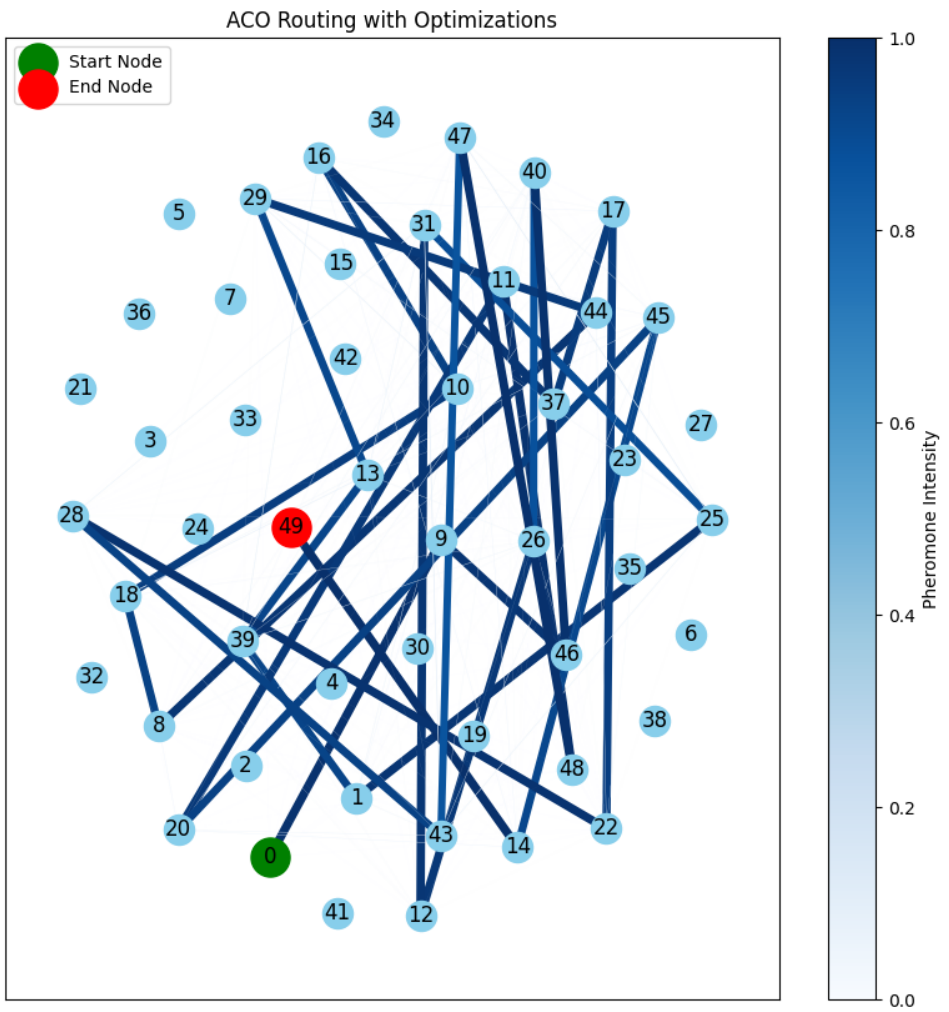

Figure 2: ACO Routing with Optimizations

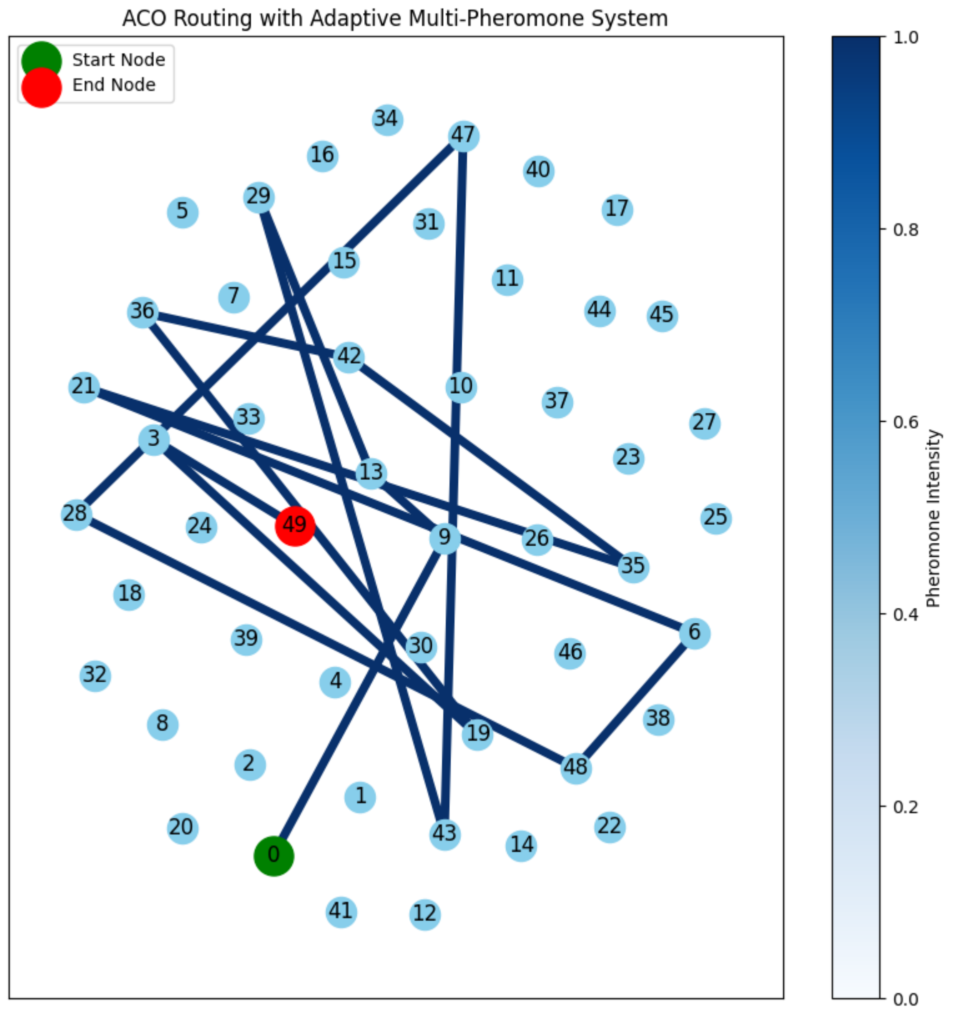

Figure 3: ACO Routing with Adaptive Multi-Pheromone System

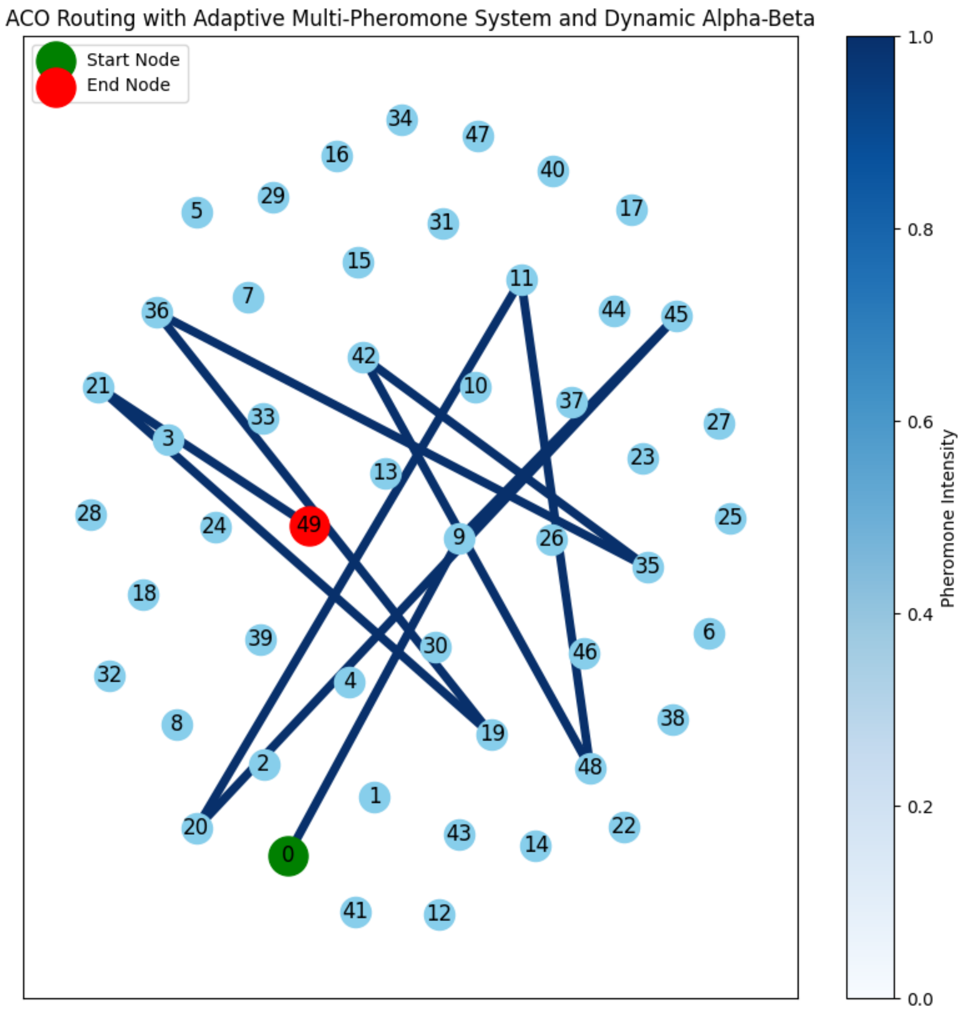

Figure 4: ACO Routing with Adaptive Multi-Pheromone System and Dynamic Alpha-Beta

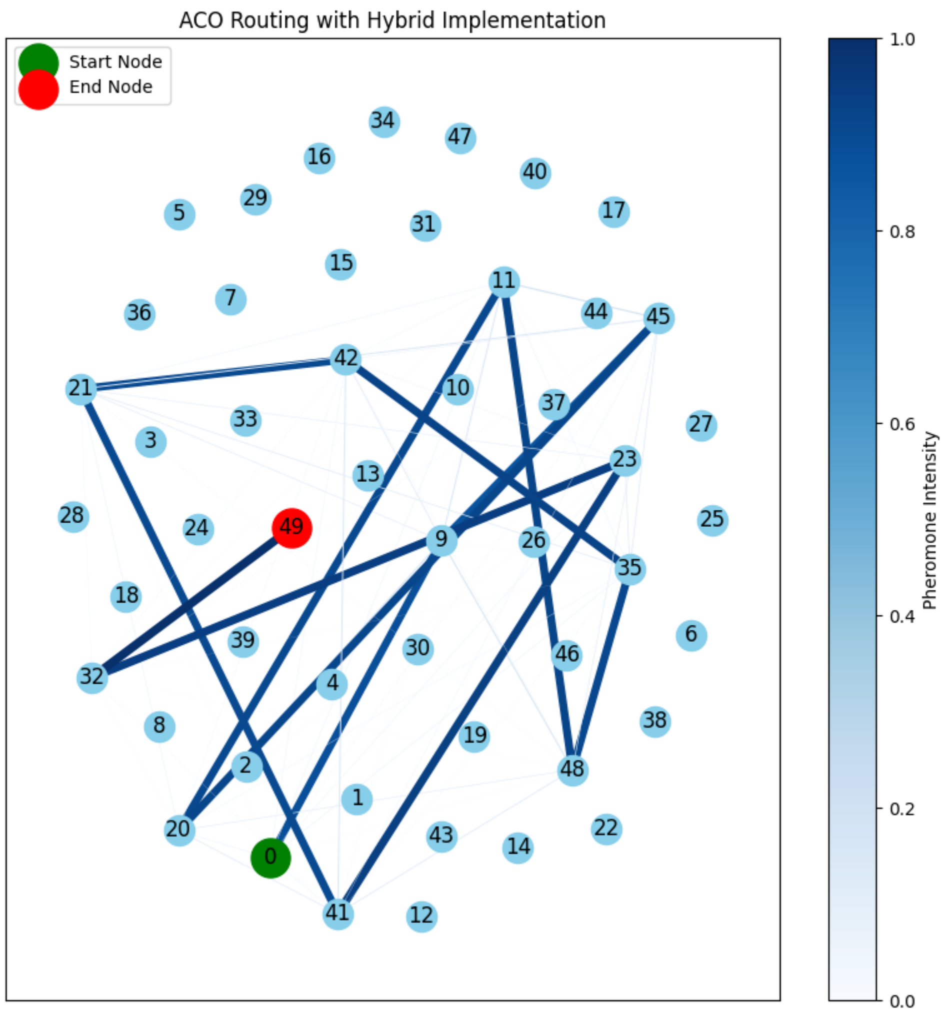

Figure 5: ACO Routing with Hybrid Implementation

Code

The following Python code contains the implementation of the Ant Colony Optimization (ACO) simulation used for the experiments described in this work.

```python
import networkx as nx
import numpy as np
import random
import matplotlib.pyplot as plt
from copy import deepcopy
import math
import time

def run_aco_simulation(G, pheromone_levels, num_ants, iterations, source, sink, adaptive_evaporation=False, apply_local_search=False, multi_pheromone=False, secondary_pheromone_levels=None):
    # Parameters for ACO
    # Initial values for alpha and beta
    alpha = 1.0  # Pheromone influence
    beta = 2.0   # Distance influence
    evaporation_rate = 0.5  # Initial evaporation rate

    def choose_next_node(current_node, visited):
        neighbors = list(G.neighbors(current_node))
        probabilities = []
        for neighbor in neighbors:
            if neighbor not in visited:
                edge = (min(current_node, neighbor), max(current_node, neighbor))
                pheromone = pheromone_levels[edge]
                distance = np.linalg.norm(np.array(G.nodes[current_node]['pos']) - np.array(G.nodes[neighbor]['pos']))
                prob = (pheromone ** alpha) * ((1.0 / distance) ** beta)
                if multi_pheromone and secondary_pheromone_levels:
                    secondary_pheromone = secondary_pheromone_levels[edge]
                    prob *= (secondary_pheromone ** 0.5)  # Weight secondary pheromone
                probabilities.append(prob)
            else:
                probabilities.append(0.0)

        total_prob = sum(probabilities)
        if total_prob == 0:
            return random.choice(neighbors)  # Random fallback
        probabilities = [p / total_prob for p in probabilities]
        return random.choices(neighbors, probabilities)[0]

    def local_search(path):
        # Local search strategy: reverse a segment of the path to see if it improves quality
        if len(path) > 3:
            new_path = deepcopy(path)
            i, j = sorted(random.sample(range(1, len(path) - 1), 2))
            new_path[i:j] = reversed(new_path[i:j])
            return new_path
        return path

    best_path = None
    best_path_length = float('inf')
    convergence_iteration = None
    all_path_lengths = []

    start_time = time.time()

    for i in range(iterations):
        # Calculate pheromone entropy before adjusting parameters
        pheromone_values = np.array(list(pheromone_levels.values()))
        pheromone_probabilities = pheromone_values / pheromone_values.sum()
        pheromone_entropy = -np.sum(pheromone_probabilities * np.log(pheromone_probabilities + 1e-9))  # Small epsilon to avoid log(0)
        # Adjust alpha and beta dynamically based on previous success and pheromone entropy
        if i > 0 and len(all_path_lengths) > 1:
            if all_path_lengths[-1] < all_path_lengths[-2]:
                # If the average path length improved, increase exploitation
                alpha = min(3.0, alpha + 0.05)  # Cap alpha to avoid over-exploitation
                beta = max(1.0, beta - 0.05)    # Decrease beta to reduce exploration
            else:
                # If the average path length did not improve, increase exploration
                alpha = max(1.0, alpha - 0.05)
                beta = min(3.0, beta + 0.05)  # Cap beta to avoid over-exploration
            # Adjust parameters based on pheromone entropy
            if pheromone_entropy < 1.5:
                # Low entropy: increase exploration
                alpha = max(1.0, alpha - 0.05)
                beta = min(3.0, beta + 0.05)
            elif pheromone_entropy > 2.5:
                # High entropy: increase exploitation
                alpha = min(3.0, alpha + 0.05)
                beta = max(1.0, beta - 0.05)

        all_paths = []
        total_path_length = 0
        for ant in range(num_ants):
            current_node = source
            visited = [current_node]
            while current_node != sink:
                next_node = choose_next_node(current_node, visited)
                visited.append(next_node)
                current_node = next_node
            if apply_local_search:
                visited = local_search(visited)  # Apply local search if enabled
            all_paths.append(visited)
            total_path_length += len(visited)

            # Update best path
            if len(visited) < best_path_length:
                best_path = visited
                best_path_length = len(visited)
                convergence_iteration = i

        all_path_lengths.append(total_path_length / num_ants)  # Average path length for this iteration

        # Adjust evaporation rate dynamically
        if adaptive_evaporation:
            if pheromone_entropy > 2.5:
                evaporation_rate = max(0.3, evaporation_rate - 0.01)  # Decrease evaporation to retain pheromone longer
            elif pheromone_entropy < 1.5:
                evaporation_rate = min(0.7, evaporation_rate + 0.01)  # Increase evaporation to promote exploration  # Gradually increase evaporation

        # Evaporate pheromone
        for edge in pheromone_levels:
            pheromone_levels[edge] *= (1 - evaporation_rate)
            if multi_pheromone and secondary_pheromone_levels:
                secondary_pheromone_levels[edge] *= (1 - evaporation_rate)

        # Update pheromones based on path quality
        for path in all_paths:
            for j in range(len(path) - 1):
                edge = (min(path[j], path[j + 1]), max(path[j + 1], path[j]))
                pheromone_levels[edge] += 1.0 / len(path)  # Shorter paths get more pheromone
                if multi_pheromone and secondary_pheromone_levels:
                    secondary_pheromone_levels[edge] += 0.5 / len(path)  # Update secondary pheromone

    end_time = time.time()
    runtime = end_time - start_time

      # Small epsilon to avoid log(0)

    # Output metrics
    return {
        "best_path_length": best_path_length,
        "convergence_iteration": convergence_iteration if convergence_iteration is not None else "Did not converge",
        "average_path_length": np.mean(all_path_lengths),
        "pheromone_entropy": pheromone_entropy,
        "runtime": runtime
    }

# Step 1: Run the simulations on multiple graphs
num_graphs = 10
num_nodes = 50
radius = 15
results_without_optimizations = []
results_with_optimizations = []
results_multi_pheromone = []

for i in range(num_graphs):
    print(f"\nRunning simulations on graph {i + 1}/{num_graphs}...")
    G = nx.random_geometric_graph(num_nodes, radius, dim=2)
    pos = nx.spring_layout(G, seed=42)

    # Run ACO without optimizations
    print("Running ACO without optimizations...")
    pheromone_levels_initial = {edge: 1 for edge in G.edges}  # Initialize
     pheromone levels
    result = run_aco_simulation(G, pheromone_levels_initial, num_ants=10,
   iterations=1000, source=0, sink=num_nodes - 1)
    results_without_optimizations.append(result)

    # Run ACO with optimizations
    print("Running ACO with optimizations...")
    pheromone_levels_optimized = {edge: 1 for edge in G.edges}  #
   Reinitialize pheromone levels
    result = run_aco_simulation(G, pheromone_levels_optimized, num_ants
   =10, iterations=1000, source=0, sink=num_nodes - 1,
   adaptive_evaporation=True, apply_local_search=True)
    results_with_optimizations.append(result)

    # Run ACO with adaptive multi-pheromone system
    print("Running ACO with adaptive multi-pheromone system...")
    primary_pheromone_levels = {edge: 1 for edge in G.edges}  # Primary
   pheromone levels
    secondary_pheromone_levels = {edge: 1 for edge in G.edges}  #
   Secondary pheromone levels
    result = run_aco_simulation(G, primary_pheromone_levels, num_ants=10,
   iterations=1000, source=0, sink=num_nodes - 1, multi_pheromone=True,
   secondary_pheromone_levels=secondary_pheromone_levels)
    results_multi_pheromone.append(result)

    # Run ACO with adaptive multi-pheromone system and dynamic alpha-beta
    print("Running ACO with adaptive multi-pheromone system and dynamic
   alpha-beta...")
    primary_pheromone_levels_dynamic = {edge: 1 for edge in G.edges}  #
   Primary pheromone levels
    secondary_pheromone_levels_dynamic = {edge: 1 for edge in G.edges}  #
   Secondary pheromone levels
    result = run_aco_simulation(G, primary_pheromone_levels_dynamic,
   num_ants=10, iterations=1000, source=0, sink=num_nodes - 1,
   multi_pheromone=True, secondary_pheromone_levels=
   secondary_pheromone_levels_dynamic)
```

```python
    results_multi_pheromone.append(result)

    # Run ACO with hybrid implementation
    print("Running ACO with hybrid implementation...")
    hybrid_pheromone_levels = {edge: 1 for edge in G.edges}  # Hybrid pheromone levels
    hybrid_secondary_pheromone_levels = {edge: 1 for edge in G.edges}  # Secondary pheromone levels
    result = run_aco_simulation(G, hybrid_pheromone_levels, num_ants=10, iterations=1000, source=0, sink=num_nodes - 1, adaptive_evaporation=True, apply_local_search=True, multi_pheromone=True, secondary_pheromone_levels=hybrid_secondary_pheromone_levels)
    results_multi_pheromone.append(result)

# Step 2: Calculate average metrics for each implementation
def calculate_average_metrics(results):
    avg_best_path_length = np.mean([r["best_path_length"] for r in results])
    avg_convergence_iteration = np.mean([r["convergence_iteration"] if isinstance(r["convergence_iteration"], int) else iterations for r in results])
    avg_average_path_length = np.mean([r["average_path_length"] for r in results])
    avg_pheromone_entropy = np.mean([r["pheromone_entropy"] for r in results])
    avg_runtime = np.mean([r["runtime"] for r in results])
    return {
        "avg_best_path_length": avg_best_path_length,
        "avg_convergence_iteration": avg_convergence_iteration,
        "avg_average_path_length": avg_average_path_length,
        "avg_pheromone_entropy": avg_pheromone_entropy,
        "avg_runtime": avg_runtime
    }

average_metrics_without_optimizations = calculate_average_metrics(results_without_optimizations)
average_metrics_with_optimizations = calculate_average_metrics(results_with_optimizations)
average_metrics_multi_pheromone = calculate_average_metrics(results_multi_pheromone[:num_graphs])
average_metrics_multi_pheromone_dynamic = calculate_average_metrics(results_multi_pheromone[num_graphs:num_graphs*2])
average_metrics_hybrid = calculate_average_metrics(results_multi_pheromone[num_graphs*2:])

# Step 3: Print average metrics
print("\nAverage Metrics for ACO without Optimizations:")
print(f"  - Average Best Path Length: {average_metrics_without_optimizations['avg_best_path_length']:.2f}")
print(f"  - Average Convergence Iteration: {average_metrics_without_optimizations['avg_convergence_iteration']:.2f}")
print(f"  - Average Path Length Over Iterations: {average_metrics_without_optimizations['avg_average_path_length']:.2f}")
print(f"  - Average Pheromone Entropy: {average_metrics_without_optimizations['avg_pheromone_entropy']:.2f}")
print(f"  - Average Runtime (seconds): {average_metrics_without_optimizations['avg_runtime']:.2f}")

print("\nAverage Metrics for ACO with Optimizations:")
print(f"  - Average Best Path Length: {average_metrics_with_optimizations['avg_best_path_length']:.2f}")
print(f"  - Average Convergence Iteration: {average_metrics_with_optimizations['avg_convergence_iteration']:.2f}")
print(f"  - Average Path Length Over Iterations: {average_metrics_with_optimizations['avg_average_path_length']:.2f}")
print(f"  - Average Pheromone Entropy: {average_metrics_with_optimizations['avg_pheromone_entropy']:.2f}")
print(f"  - Average Runtime (seconds): {average_metrics_with_optimizations['avg_runtime']:.2f}")

print("\nAverage Metrics for ACO with Adaptive Multi-Pheromone System:")
print(f"  - Average Best Path Length: {average_metrics_multi_pheromone['avg_best_path_length']:.2f}")
print(f"  - Average Convergence Iteration: {average_metrics_multi_pheromone['avg_convergence_iteration']:.2f}")
print(f"  - Average Path Length Over Iterations: {average_metrics_multi_pheromone['avg_average_path_length']:.2f}")
print(f"  - Average Pheromone Entropy: {average_metrics_multi_pheromone['avg_pheromone_entropy']:.2f}")
print(f"  - Average Runtime (seconds): {average_metrics_multi_pheromone['avg_runtime']:.2f}")

print("\nAverage Metrics for ACO with Adaptive Multi-Pheromone System and Dynamic Alpha-Beta:")
print(f"  - Average Best Path Length: {average_metrics_multi_pheromone_dynamic['avg_best_path_length']:.2f}")
print(f"  - Average Convergence Iteration: {average_metrics_multi_pheromone_dynamic['avg_convergence_iteration']:.2f}")
print(f"  - Average Path Length Over Iterations: {average_metrics_multi_pheromone_dynamic['avg_average_path_length']:.2f}")
print(f"  - Average Pheromone Entropy: {average_metrics_multi_pheromone_dynamic['avg_pheromone_entropy']:.2f}")
print(f"  - Average Runtime (seconds): {average_metrics_multi_pheromone_dynamic['avg_runtime']:.2f}")

print("\nAverage Metrics for ACO with Hybrid Implementation:")
print(f"  - Average Best Path Length: {average_metrics_hybrid['avg_best_path_length']:.2f}")
print(f"  - Average Convergence Iteration: {average_metrics_hybrid['avg_convergence_iteration']:.2f}")
print(f"  - Average Path Length Over Iterations: {average_metrics_hybrid['avg_average_path_length']:.2f}")
print(f"  - Average Pheromone Entropy: {average_metrics_hybrid['avg_pheromone_entropy']:.2f}")
print(f"  - Average Runtime (seconds): {average_metrics_hybrid['
    avg_runtime']:.2f}")
```

Listing 1: ACO Simulation Implementation