Kathmandu University Journal of Science, Engineering and Technology, Vol. 19, No. 1, March 2025

Kathmandu University
Journal of Science, Engineering and Technology

Study and implementation of modified ant colony optimization for travelling salesman problem

Prashant Timalsina* and Aayush Shrestha

Department of Mathematics, Kathmandu University, Dhulikhel, Kavre, Nepal.

Abstract

The paper explores and visualizes a modified Ant Colony Optimization (ACO) algorithm to solve the Travelling Salesman Problem (TSP), implemented in Python. The TSP, a classic optimization problem, requires finding a Hamiltonian path that visits each city exactly once and returns to the starting point while minimizing the total distance travelled. Our approach introduces dynamic random thresholds for node selection and fixed pheromone update, which adapts based on the algorithm's performance. This probabilistic component enhances exploration and reduces the likelihood of premature convergence, diverging from traditional ACO methods. The implementation features an interactive, grid-based environment where users can select nodes representing cities. The modified ACO algorithm iteratively identifies a local optimal Hamiltonian path by simulating multiple generations of ant colonies, with customizable parameters such as the number of ants and generations. Key features include real-time visualization of the best path found and dynamic pheromone updates. This paper provides a basis for further research into adaptive evolutionary intelligence algorithms for optimization problems. It offers insights into applying ACO to find Hamiltonian paths in complex graphs.

Keywords: Ant colony optimization; Travelling salesman problem; Hamiltonian path; Evolutionary intelligence; Pheromone update.

1. Introduction

The Travelling Salesman Problem, introduced in the 1930s, grew from a math puzzle to a key challenge in computer science [1, 2]. It has spurred advances in algorithms and computation, evolving to tackle larger problems while remaining a benchmark for optimization methods [3].

The Travelling Salesman Problem (TSP) involves finding the shortest route for a salesman to visit each city exactly once and return to the starting point. Despite its simple formulation, the TSP is computationally challenging due to its NP-hard nature, making it a benchmark problem for evaluating optimization algorithms. Over the years, TSP has driven the development of exact and heuristic methods, with applications in logistics, manufacturing, and theoretical computer science [4].

Ant Colony Optimization (ACO), proposed by Marco Dorigo in the early 1990s [5], is a heuristic algorithm inspired by the foraging behaviour of ants. ACO uses artificial "ants" to construct solutions, guided by simulated pheromone trails that represent promising paths. These pheromones influence the construction of future solutions, enabling iterative improvement. ACO has proven to be an effective method for solving combinatorial optimization problems, including the TSP, and has found applications in areas such as multi-objective and dynamic optimization [6].

An ant colony optimization algorithm is a problem-solving method inspired by how ants find food, using computer-generated "ants" to build possible solutions and leave virtual trails that help guide future attempts to find the best answer to complex problems [6].

The application of ACO to the TSP is well-suited due to their complementary characteristics. In this context, artificial ants probabilistically select cities to visit based on heuristic information and pheromone levels. Pheromone updates reinforce efficient routes, allowing the algorithm to converge on optimal or near-optimal solutions. This combination has facilitated progress in solving large-scale TSP instances, demonstrating the utility of bio-inspired methods in addressing complex optimization problems [7].

*Corresponding author. Email: timalsinaprashant4@gmail.com

2. Methods

The Ant Colony Optimization (ACO) algorithm is a metaheuristic optimization method inspired by the foraging behaviour of ants. It has been widely used to solve combinatorial optimization problems, such as the Travelling Salesman Problem (TSP). This section presents the implementation details of the ACO algorithm and discusses its modifications.

2.1. Ant colony optimization (ACO) algorithm

The Ant Colony Optimization (ACO) algorithm models ant behaviour to find optimal paths between nodes. It initializes pheromone and distance matrices to represent trail intensity and pairwise distances. Nodes are placed based on user input, and distances are calculated. In each generation, ants explore paths probabilistically, influenced by pheromone levels and distances. Pheromone updates emphasize shorter paths while applying a decay factor of 0.9 to balance exploration and exploitation. This process iterates until the shortest path is identified and visualized. Users can dynamically place nodes, with real-time updates to the grid and visualization.

The original ACO algorithm for the Travelling Salesman Problem (TSP) is provided in Algorithm 1, as outlined in [8]. Algorithm 2 incorporates the modifications detailed in Section 2.2.

2.2. Modifications

The modifications are as follows:

1. Dynamic threshold for node selection: During tour construction, a dynamic threshold increases if no valid next city is selected, encouraging exploration and avoiding premature convergence on suboptimal paths.

2. Random shuffling of node list: At each iteration, the available city list is randomly shuffled for each ant, promoting diverse exploration and reducing identical solutions.

3. Disabling revisits using the pheromone matrix: After a city is visited, its pheromone values in both directions are set to zero, preventing revisits and ensuring valid tours.

4. Best path selection and update: The shortest tour in each iteration is identified. If it surpasses the global best solution, it becomes the new global best, ensuring consistent improvement.

These modifications aim to balance exploration and exploitation by diversifying the search process and improving computational efficiency.

Algorithm 1 Original ant colony optimization for the travelling salesman problem.

Require: Graph $G = (V, E)$, distances d_{ij}, parameters m, ρ, α, β, and termination criteria.
Ensure: Best tour T_{best} and its length L_{best}.
Initialize pheromone trails $\tau_{ij}(t)$ for all arcs (i, j) with a small positive value.
Place m ants randomly on cities in the graph.
Set iteration counter $t = 0$.
while termination condition is not met **do**
  for each ant **do**
    Start at a randomly assigned city.
    Initialize memory (tabu list) to track visited cities.
    while not all cities are visited **do**
      Compute transition probabilities:

$$P_{ij} = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k \in \text{Allowed}} [\tau_{ik}(t)]^{\alpha} \cdot [\eta_{ik}]^{\beta}}$$

      Move to city j based on P_{ij} and update tabu list.
    end while
    Complete the tour by returning to the starting city.
  end for
  for each ant **do**
    Apply local search (e.g., 2-opt or 3-opt) to improve the constructed tour.
  end for
  Update pheromone trails:
  Evaporate pheromones:

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t)$$

  Deposit pheromones based on ant tours:

$$\tau_{ij}(t+1) = \tau_{ij}(t+1) + \sum_{\text{ants}} \Delta\tau_{ij}$$

  Update best solution T_{best} and L_{best} if a better tour is found.
  Increment iteration counter: $t = t + 1$.
end while
return T_{best} and L_{best}.

Algorithm 2 Ant movement for ACO.

Initialize the parr matrix with zeros to store ant paths and distances.
Create a list a containing nodes from 1 to POINT.
for each ant v in 1 to ANT **do** ▷ Simulate the movement of each ant.
  Copy the pref matrix to p_temp to track pheromone preferences.
  Set the current node $i = 0$ and initialize path length $n = 0$.
  while $n \neq$ `POINT` $- 1$ **do** ▷ Visit all nodes except the last return to start.
    Randomly shuffle the list of nodes a.
    Track if a new node was selected using n1, initialize a threshold $x = 0$.
    for each node j in a **do**
      Generate a random number rand in range $[5, x]$.
      if p_temp[i][j] > 0 and p_temp[i][j] $\leq$ rand **then**
        Set parr[n+1][v] = j, the next node for ant v.
        if $n \neq 0$ **then**
          Set p_temp[:,i] and p_temp[i,:] to 0 to disable revisiting.
        end if
        Increment n by 1 and update the path cost:
        parr[POINT+1][v] += dist[i][j].
        Move to the next node: $i = j$.
        Break the loop for node j.
      end if
    end for
    Increase the threshold x by 10 to encourage exploration if no node was selected.
  end while
  Close the loop for ant v by returning to the start node:
  parr[n+1][v] = 0 and update the total path cost:
  parr[POINT+1][v] += dist[i][0].
end for
Identify the ant with the shortest path:
l = parr[POINT+1][0], set m = 0.
for each ant k in 1 to ANT **do**
  if parr[POINT+1][k] $< l$ **then**
    Update $m = k$ and $l = parr[POINT + 1][k]$.
  end if
end for
Update parr[: ,ANT] with the path of the best-performing ant.

2.3. Setup

The above algorithm is implemented using Python, with the pygame library for graphical visualization and numpy for numerical computations. As a proof of work for the algorithm, the following parameters and variables were initialized as an example, to model the Ant Colony Optimization (ACO) algorithm for solving the Travelling Salesman Problem (TSP):

- Window dimensions: The application runs on a 700×700 pixel display.
- Grid dimensions: A 50×50 grid is used to facilitate spatial representation of nodes.
- Constants:
  - POINT: Number of points (nodes) to visit, initialized to 8.
  - ANT: Number of ants in the simulation, set to 12.
  - GENER: Number of generations, set to 9.
- Arrays:
  - dist: A 8×8 matrix storing pairwise Euclidean distances between nodes.

- pref: A 8×8 pheromone preference matrix, initialized with default values.
- parr: A 10×13 matrix to store paths and weights for each ant.

- Grid and nodes:
  - grid: A 50×50 grid of Box objects, each representing a cell in the window.
  - NOD: A list storing user-defined node locations.

2.4. User interface (UI)

The graphical interface was developed using pygame, allowing users to interactively add nodes and visualize results. Key design elements include:

- Color scheme:
  - Grid cells are white by default.
  - Nodes are displayed in red.
  - Optimal paths are drawn in black lines.
- Controls:
  - Left-click to add nodes.
  - Press Spacebar to start the algorithm.
  - Press C to clear the window.
  - Press Backspace to reset the simulation.
- Text instructions: Instructions for user interaction are displayed on the top-left corner of the screen.

3. Results and discussion

This section of the paper presents the resulting execution of the Algorithm 2 given the setup discussed in section 2.3.

3.1. User input interface

Fig. 1 demonstrates the user interface for node selection. The interface provides an intuitive method for users to interact with the Ant Colony Optimization (ACO) algorithm.

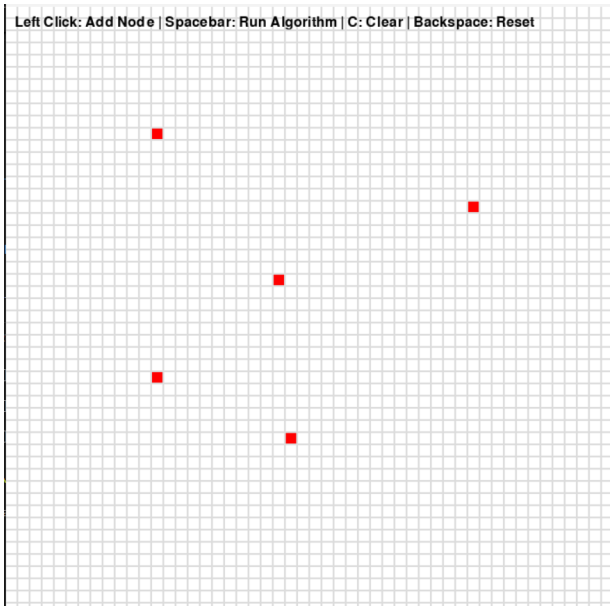

Figure 1: User interface for node selection.

3.2. Algorithmic visualization

The sections below prove the visualizations to provide insight into how the algorithm processes spatial and pheromone information to find optimal solutions.

3.2.1. Distance matrix

In one of the algorithm runs, we generated a sample distance matrix, as illustrated in Fig. 2. This matrix represents the spatial relationships between nodes.

```
[[ 0.         15.26433752 30.47950131 13.         10.63014581 28.46049894
  23.         14.31782106]
 [15.26433752  0.         15.62049935 13.03840481 20.         19.41648784
  12.80624847 17.08800749]
 [30.47950131 15.62049935  0.         27.31300057 32.31098884 15.65247584
  20.         26.68332813]
 [13.         13.03840481 27.31300057  0.         23.02172887 32.14031736
  12.08304597 24.20743687]
 [10.63014581 20.         32.31098884 23.02172887  0.         24.8394847
  31.04834939  7.21110255]
 [28.46049894 19.41648784 15.65247584 32.14031736 24.8394847   0.
  30.41381265 17.69180601]
 [23.         12.80624847 20.         12.08304597 31.04834939 30.41381265
   0.         29.52964612]
 [14.31782106 17.08800749 26.68332813 24.20743687  7.21110255 17.69180601
  29.52964612  0.        ]]
```

Figure 2: Distance matrix for ACO algorithm.

3.2.2. Pheromone matrix

Fig. 3 presents the pheromone matrix corresponding to the same run, showcasing the dynamic information exchange mechanism in the ACO algorithm.

```
[[ 0. 15. 30. 13. 10. 28. 23. 14.]
 [15.  0. 15. 13. 20. 19. 12. 17.]
 [30. 15.  0. 27. 32. 15. 20. 26.]
 [13. 13. 27.  0. 23. 32. 12. 24.]
 [10. 20. 32. 23.  0. 24. 31.  7.]
 [28. 19. 15. 32. 24.  0. 30. 17.]
 [23. 12. 20. 12. 31. 30.  0. 29.]
 [14. 17. 26. 24.  7. 17. 29.  0.]]
```

Figure 3: Pheromone matrix representation.

3.2.3. Path array

The final path array matrix for the run is depicted in Fig. 4, illustrating the solution trajectory of the algorithm.

```
[ 0.         3.         1.         6.         2.         5.
  7.         4.         0.        99.40003769]
[ 0.         5.         2.         7.         4.
   1.         3.         6.         0.       126.80867467]
[ 0.         4.         7.         5.         2.         1.
  6.         3.         0.        91.69532402]
[ 0.         3.         4.         1.         7.
   5.         2.         6.         0.       126.49242302]
[ 0.         6.         1.         5.         7.
   4.         3.         2.         0.       119.54342029]
[ 0.         4.         7.         1.         3.
   2.         5.         6.         0.       121.34694973]
[ 0.         4.         7.         6.         3.
   1.         2.         5.         0.       103.76532046]
[ 0.         4.         7.         3.         6.         1.
  2.         5.         0.        98.21095488]
[ 0.         7.         4.         5.         2.
   1.         6.         3.         0.       102.53067795]
```

Figure 4: Path array matrix.

3.2.4. Algorithm output

Fig. 5 and Fig. 6 showcase the algorithm's output for different random re-runs, demonstrating its adaptability and solution diversity.

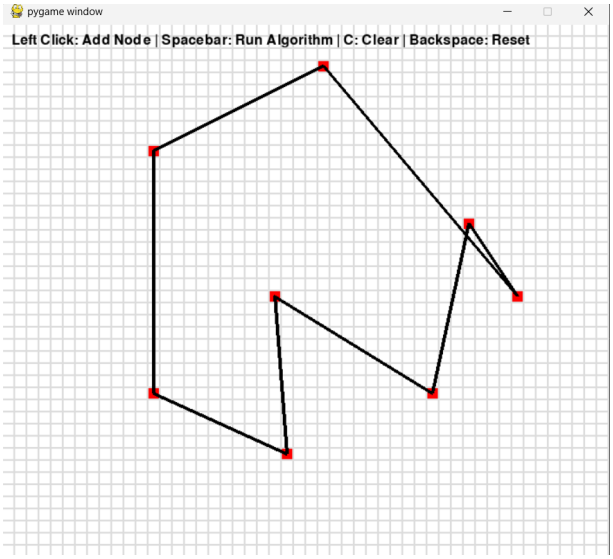

Figure 5: ACO solution for random re-run.

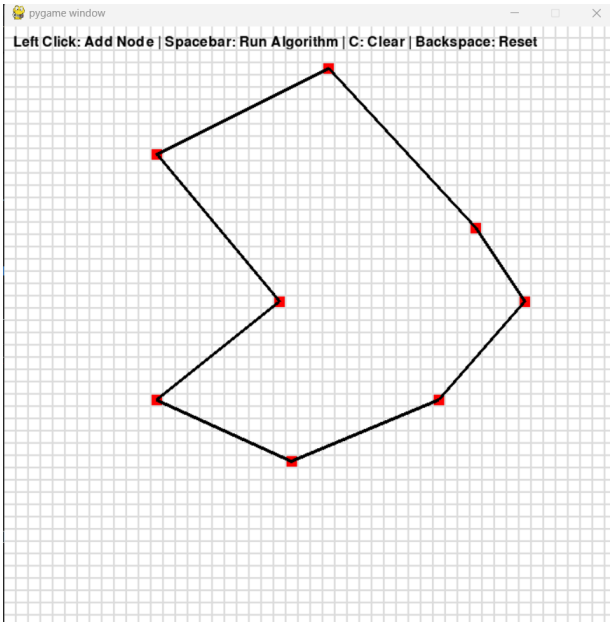

Figure 6: ACO solution for third re-run.

4. Conclusion

Our research yielded several significant findings:

- Successfully adapted the ACO algorithm to a grid-based representation, enabling intuitive visualization and user interaction.
- Developed a simplified computational model that preserves the core principles of ACO while reducing algorithmic complexity.
- Demonstrated the algorithm's effectiveness in finding optimal or near-optimal solutions for user-defined Travelling Salesman Problem (TSP) instances.
- Implemented real-time visualization of the algorithm's progress, enhancing understanding of ACO's behaviour and decision-making process.

Our implementation incorporates randomization and exploration, using random shuffling of nodes to prevent deterministic path selection. Also, it includes a basic pheromone decay mechanism (multiplying by 0.9) which simulates the natural ant behaviour of reinforcing promising paths. For visualization and interactivity, we used a Pygame-based interactive interface which allows manual node placement and provides real-time visualization of path optimization.

The implementation also has some limitations. The main ones are algorithmic constraints like a fixed number of generations (GENER = 9), limited population (ANT = 12), and simplistic pheromone update rule. Also, the algorithm lacks sophisticated distance metrics and instead uses Euclidean distance squared.

The potential future enhancements can be to select more complex path selection probabilities, incorporate local and global pheromone update strategies, and develop more sophisticated pheromone update rules.

Acknowledgements

We would like to express our gratitude to Dr. Prakash Poudyal, for his consistent support and feedback throughout this paper. His feedback and suggestions helped direct the paper to its completion. Lastly, we would like to thank Kathmandu University, Kathmandu University Mathematics Students Club, the Department of Mathematics, the Department of Computer Science and Engineering, and everyone else for their direct and indirect role in supporting us in this paper.

References

[1] Katiyar S, Nasiruddin I & Ansari A Q. Ant colony optimization: A tutorial review. In: *National conference on advances in power and control* (2015), pp. 99–110. URL https://www.researchgate.net/profile/Sapna-Katiyar/publication/281432201_Ant_Colony_Optimization_A_Tutorial_Review/links/5613da7308aed47facede845/Ant-Colony-Optimization-A-Tutorial-Review.pdf.

[2] Sangwan S, Literature review on travelling salesman problem, *International Journal of Research*, 5(16) (2018) 1152. URL https://www.researchgate.net/publication/341371861_Literature_Review_on_Travelling_Salesman_Problem.

[3] Dorigo M & Gambardella L M, Ant colonies for the travelling salesman problem, *Biosystems*, 43(2) (1997) 73–81. https://doi.org/10.1016/S0303-2647(97)01708-5.

[4] Laporte G, A concise guide to the traveling salesman problem, *The Journal of the Operational Research Society*, 61(1) (2010) 35–40. ISSN 01605682, 14769360. URL http://www.jstor.org/stable/40540226.

[5] Dorigo M, Birattari M & Stutzle T, Ant colony optimization, *IEEE Computational Intelligence Magazine*, 1(4) (2006) 28–39. https://doi.org/10.1109/MCI.2006.329691.

[6] Blum C, Ant colony optimization: Introduction and recent trends, *Physics of Life Reviews*, 2(4) (2005) 353–373. https://doi.org/10.1016/j.plrev.2005.10.001.

[7] Akhtar A, Evolution of ant colony optimization algorithm – a brief literature review, *arXiv preprint arXiv:1908.08007*. URL https://arxiv.org/abs/1908.08007.

[8] Stützle T & Dorigo M. ACO algorithms for the traveling salesman problem. In: *Evolutionary Algorithms in Engineering and Computer Science: Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications* (1999). URL https://faculty.washington.edu/paymana/swarm/stutzle99-eaecs.pdf.