RESEARCH ARTICLE

INTERNATIONAL RESEARCH JOURNAL OF MULTIDISCIPLINARY TECHNOVATION

Asian Research Association

DOI: 10.54392/irjmt2564

Evaluating and Improving Algorithms for Off-Road Routing

Jay Kansara [a,*], Tanmay Mistry [a], Vedant Bhawnani [a], Dwiti Choksi [a], Lakshmi Kurup [b], Pratik Kanani [a]

[a] Department of Artificial Intelligence and Data Science, Dwarkadas J. Sanghvi College of Engineering, Mumbai, India.
[b] School of AI, Amrita University, Delhi, India.
* Corresponding Author Email: jkansara03@gmail.com

DOI: https://doi.org/10.54392/irjmt2564
Received: 05-10-2024; Revised: 30-09-2025; Accepted: 17-10-2025; Published: 25-10-2025

Abstract: In this paper, we explore the multifaceted capabilities of route-finding algorithms and their role in delivering dynamic paths for diverse navigation scenarios. In off-road route finding, the shortest path is not always the best; the route's smoothness must also be considered. Present methodologies like A* have notable limitations, such as difficulty adapting to complex terrains and producing rugged routes. This limitation hampers performance in critical scenarios, such as in emergencies like landslides or earthquakes, where off-road exploration is crucial. This paper proposes 3 methodologies that have been fine-tuned with optimal hyperparameters for optimal results using gradient descent. Two of these methodologies, Simulated Annealing and Ant Colony Optimization, overcome some limitations of A* but not all. However, Q-Learning significantly overcomes the limitations and saves travel time by providing a route with 70% fewer undulations and a 16% reduction in route length compared to A*. Compared to other implementations, the Q-Learning implementation proposed in this research not only focuses on minimizing path length but also minimizes route undulations, providing a dual objective approach that is well suited to real-world scenarios. Unlike prior implementations, which focus on a single objective, such as path length or obstacle avoidance, this work leverages a reward function that penalizes elevation variance while rewarding shorter routes, resulting in smoother, more easily traversable paths. Thus, Q-Learning overcomes the cons of the present methodologies and can form synergistic combinations, enhancing the overall performance of off-road space searching systems and accommodating the various challenges and complexities inherent in the targeted applications. The dataset used includes features such as latitude, longitude, and elevation. The strategic application of heuristics enables the swift evaluation of multiple paths, facilitating the selection of optimal routes in real-time applications. By combining various heuristics, the development of off-road path identification systems capable of discerning optimal paths across varied terrains becomes feasible.

Keywords: The Simulated Annealing Algorithm, A* Method, Ant Colony Algorithm, Q-learning, Path finding, Off-road routing

1. Introduction

An Off-road routing algorithm is developed specifically to find routes for automobiles or travelers traveling on terrain not ideally suited to the current standard road infrastructure. Various path-planning algorithms have emerged today, including intelligent optimization algorithms such as the ant colony algorithm and the particle swarm algorithm, neural networks [1], and heuristic search algorithms such as the A*, DFS, and BFS algorithms. Optimizing and planning paths is crucial for identifying safe, efficient driving paths in off-road path-planning tasks [2]. When barriers are present, the goal of path planning is to determine the best route between the start and end points [3]. Autonomous vehicles [4], mobile robotics [5], interior path finding [6], and planetary and space missions [7] are among the many applications of this technology. Path planning can be viewed as identifying which grid cells comprise the desired path.

The A* algorithm, characterized by a relatively straightforward implementation, high search efficiency, and good operability due to its directed and heuristic aspects in the search process, has been used in the field of static global search. The field of integrating the benefits of static path planning with the A* algorithms with other techniques has seen many recent successes. When confronted with a large-scale problem with the shortest average path but a long running time, the ant colony method, with superior global optimization abilities, will face convergence speed and coding complexity issues [8]. An actual ant colony inspired this type of search algorithm. As social insects, ants are well-organized, have a skillful division of labor, and work together to accomplish difficult tasks, such as

determining the shortest route from their colony to a food source [9]. Because of its features of distributed computing, positive feedback, heuristic search, and ease of integration with other algorithms, the artificial ant colony method, which is based on the ant colony algorithm, is frequently applied to limited-optimization problems [10]. The ant colony algorithm is used to solve path planning problems based on the observation that ants leave a pheromone trail along the paths they travel during foraging. The shorter the path, the more pheromone is left behind, and the more likely the ants are to choose it, which increases the amount of pheromone on the optimal path over time, creating a positive feedback loop that encourages the entire group to find it [9]. Optimizing pathfinding algorithms for difficult-terrain navigation is a major challenge across many applications, including outdoor sports, robotics, and transportation. Sparse and uneven road networks usually pose a challenge for traditional algorithms in examining the most efficient routes. Techniques that have given confidence in addressing the problem include meta-heuristic optimization methods, such as Simulated Annealing (SA) [11]. Unlike greedy algorithms, which often get stuck in local optima, Simulated Annealing (SA) implements probabilistic acceptance criteria to escape local optima and explore more in the search space.

The algorithm accounts for elevation variations across the terrain to determine the best route between a start point and an endpoint. Given the temperature parameter that modifies the algorithm iteratively, the method explores different terrain areas to identify near-optimal path solutions, while also considering some poor solutions. The effectiveness of the Simulated Annealing algorithm is evaluated through experiments conducted on real-world terrain data. Performance is measured and evaluated based on scalability to large-scale terrains, convergence speed, and solution quality.

One of the ACO's main advantages is its efficiency in reducing elevation differences, which directly increases runtime efficiency and poses a significant challenge. These challenges are overcome using Q-Learning, a reinforcement Learning technique [12]. Reinforcement learning is a powerful machine learning paradigm in which an agent learns to make decisions by interacting with its environment. The agent relies on a reward-penalty mechanism: correct decisions lead to rewards, and incorrect decisions lead to penalties. Q-Learning works on the principle of an action-value function, which calculates the utility, or advantage, of performing a given task.

Q-learning can effectively converge to an optimal policy and guide the agent to choose the best actions, though it requires parameter fine-tuning. After the parameters were adjusted using gradient descent, Q-Learning outperformed ACO in a few crucial areas. Furthermore, we illustrate the benefits of Q-Learning in challenging terrain navigation scenarios by contrasting the outcomes with those of conventional pathfinding algorithms. Q-Learning has a multi-objective framework for terrain-aware routing that overcomes the single-objective limitations of prior work.

All things considered, this work advances pathfinding algorithms by presenting a flexible and strong optimization strategy appropriate for a range of terrain navigation applications.

2. Literature Review

Path planning is a fundamental problem in many real-world applications, such as autonomous vehicles, network communications, and robotics, as well as route optimization. Over the years, multiple algorithms have been proposed to find the shortest path between the source and destination, but they fail to capture the real-world dynamics and terrain complexities. This study aims to propose an approach using a reinforcement based Q-Learning algorithm with dynamic, policy-driven agents to address these limitations.

Q-learning, a model-free reinforcement learning model, has been widely used in pathfinding problems, particularly in grid-based environments. However, these grid-based models are 2D and fail to capture real-world terrain undulations and smoothness [12]. Real-world terrains are often rocky and have unknown undulations, whereas grid based models rely on discrete representations of the environment, leading to suboptimal paths. Furthermore, Q-learning has been applied to networking and channel access for communication, but these applications cannot be used for physical path finding, since they focus on communication networks [13].

The limitations of discrete grids and algorithms have forced researchers to explore other algorithms to solve the problem of pathfinding. Two such algorithms are Ant Colony Optimization (ACO) and Simulated Annealing (SA). These algorithms have been used for pathfinding in both dynamic and static models.

However, both these algorithms face limitations in terms of computational complexity and efficiency. Zhonghua *et al.* [14] proposed an improved A* algorithm to find the shortest path in an off-road environment; however, their algorithm lacked consistency, producing different paths in each run. Furthermore, the path provided inefficient, torturous routes in hilly areas due to its static threshold-based methodology. While ACO and SA can find paths in more complex environments, the computational costs and uncertainty about optimality make them less suitable for real-time applications or environments with large-scale, dynamic terrain. These challenges call for further exploration of reinforcement learning-based approaches, which offer adaptable strategies and agent-based solutions.

In parallel, several studies focus on urban path planning where dense road networks and real-time navigation pose additional challenges. Path finding in such situations often involves not only finding the short path but also considering factors such as established vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication [15]. Alhoula et al. [16] discuss the necessity of promoting public transportation to minimize road traffic congestion. It assesses two implementations of the k-shortest path in a time-dependent static network. The shortest pathways are determined by the study using Dijkstra's method. The order of pathways changes when time-dependent networks are used, according to the results. The study analyzes how long algorithms take to execute across networks of varying sizes.

Furthermore, reinforcement learning has been applied to shortest path problems in innovative approaches, using the method of hitchhiking in Q-learning [17]. The use of reinforcement learning and machine learning to discover the quickest path is discussed, along with its growing popularity due to its ability to find faster adaptively, optimized routes. Dhruv *et al.* [18] explore how reinforcement learning, particularly offline Q-learning, can enhance decision-making while addressing the time-complexity challenges that plague other algorithms. This demonstrates its effectiveness in complex, dynamic terrain. This showcases the clear growing relevance of reinforcement learning for pathfinding in uncertain environments, urban path planning, and terraneous expeditions, as these approaches can dynamically adapt to environmental changes, such as fluctuating traffic and sudden road closures due to landslides, which traditional algorithms may struggle to handle efficiently.

Building on this, reinforcement learning stands out for its ability to dynamically adjust and scale, even in dynamic environments. Unlike traditional algorithms, which rely on rule-based approaches or fixed conditions, reinforcement learning models can learn from past interactions and make dynamic decisions. The adaptive nature of reinforcement learning shows promise as a key contender for solving the pathfinding problem. Notable approaches such as Q-learning, with its model-free architecture, have demonstrated success in pathfinding applications.

However, while reinforcement learning and Q-learning have shown potential for optimizing pathfinding, existing solutions struggle to apply them effectively on large and complex terrains to adapt to real-world undulations. In this research, we propose an innovative solution by combining Q-learning with dynamic, policy-driven agents. Our approach incorporates terrain and environment feedback into the agent to enable real-time learning and path adaptation. We aim to build a more scalable, optimized model that is tested on 3-dimensional grid maps that better represent real-world terrain and its undulations, dynamically adjusting to terrain features, obstacles, and elevation differences between two points. This offers significant improvements by leveraging the reported ability of reinforcement based models to perform 100 times faster than A-star with similar planned paths [19].

With significant progress in path planning, there remain problems that need to be addressed to develop a conclusive approach that addresses real-world challenges and accounts for the complexities we encounter in our daily lives. Most of the research focuses on grid-based models, which do not capture the nuances of real-world applications, making them only theoretically sound, without any real-world applications [20]. Moreover, static parameters introduce staleness in the exploration-exploitation ratio, which may lead to suboptimal paths in terrains more complex than those on which they were trained.

Therefore, there is a clear need for a dynamic, adaptive solution to address these problems. The current implementation highlights computational complexity, impacting the efficiency of and scalability of these algorithms, but a Q-learning based dynamic agent system is not sufficiently addressed. The current solutions also do not address dynamic environmental changes in both off-road and urban environments.

This research aims to bridge these gaps by introducing dynamic Q-learning agents that can efficiently and adaptively tackle real-world path planning.

3. Methodology

A pathfinding optimization approach is outlined in Figure 1. The algorithm selection divides the approach into two paths: existing and proposed methodologies. The existing methodology uses a heuristic-based method, specifically the A* algorithm, whereas the proposed methodology splits into heuristic-based methods (Ant Colony Optimization and Simulated Annealing) and agent-based methods, including Q-learning.

3.1 About the Dataset

We have used synthetic data, including longitude and latitude, to explore geographical patterns. It is a 15x15 matrix of elevation values filled with Gaussian-distributed noise.

After structuring the data into a data frame and applying Gaussian filtering with a standard deviation of 5 to reduce noise, the data now closely resembles natural terrain undulations.

The best way to visualize the output of each algorithm is as a heatmap. The heatmap revealed clear spatial trends in elevation across the study area, aiding in the identification of significant geographical features and informing further analysis.

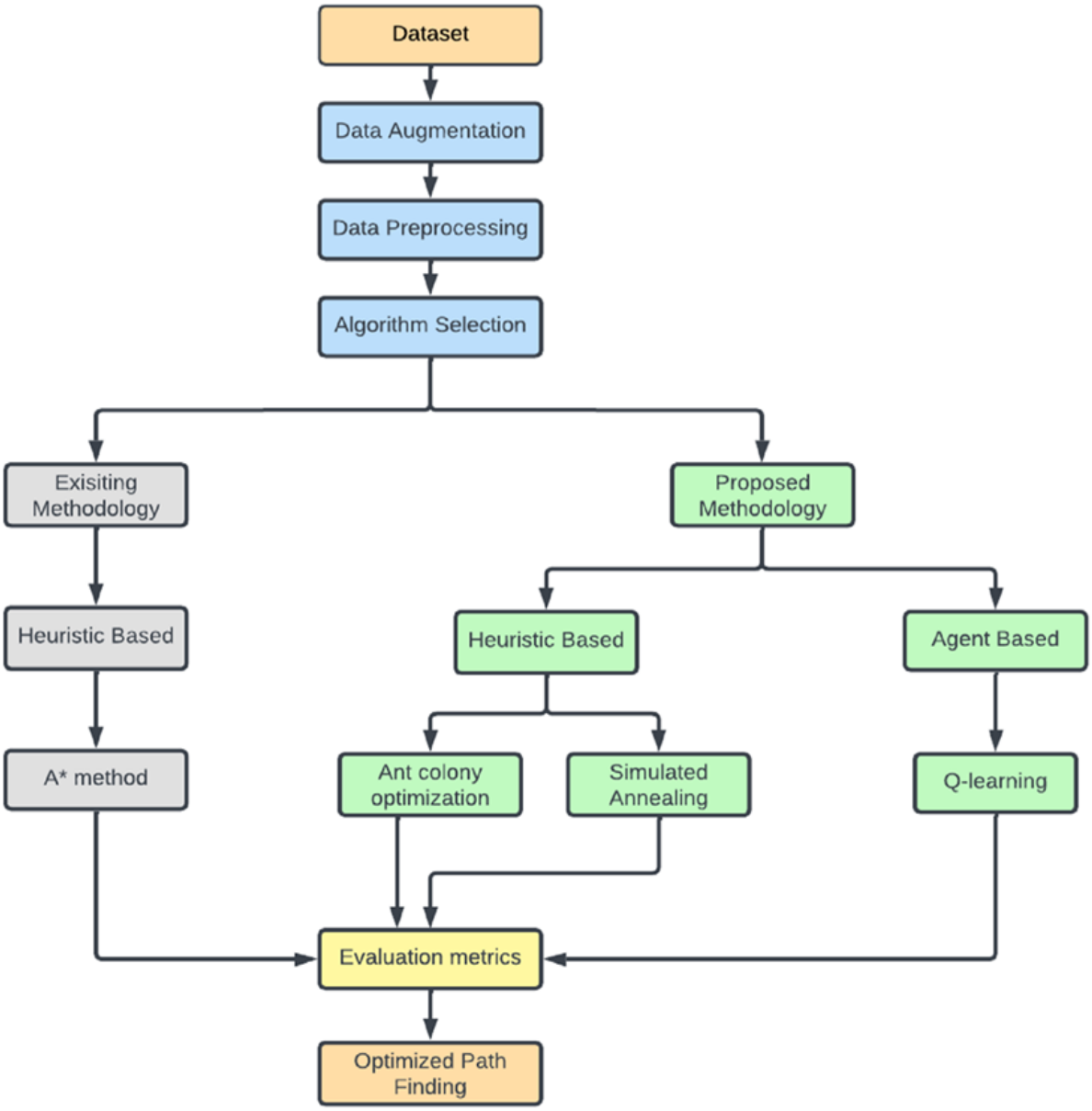

Figure 1. Proposed System Architecture

Choosing Q-learning, a reinforcement learning (RL) approach, for terrain recognition and pathfinding offers distinct advantages over heuristic-based methods. RL adapts dynamically to complex environments, crucial for terrains with varied landscapes [21]. Unlike heuristics, RL learns optimal strategies through experience, continuously improving over time. Q-learning's ability to handle uncertainties and noise, combined with its goal-oriented nature, makes it well-suited for efficient pathfinding in diverse terrains. This flexibility and adaptability justify its preference over heuristics in our research.

Once trained, reinforcement learning-based approaches can even prove to be faster than heuristic-based approaches.

3.2 Data Augmentation

In optimization methods, data augmentation is the process of generating new data from a dataset to enhance the model's robustness and performance [22]. Increasing the variety and amount of data artificially is a typical way to improve training in machine learning and optimization scenarios. We investigated geographic patterns using artificial longitude and latitude data. A 15x15 matrix of elevation values, filled with noise distributed according to a Gaussian distribution, makes up the data. The data now more closely resembles natural terrain undulations after being arranged into a data frame and subjected to 5-standard-deviation Gaussian filtering to reduce noise.

3.3 Data Preprocessing

The data preparation step in the design is essential because it gets raw data ready for the optimization phase. Typically, this block consists of operations such as data cleansing, normalization, and transformation to ensure that the data is noise- and outlier-free, consistently scaled, and consistent [23]. By improving the quality of the input data, these preprocessing techniques help the optimization algorithm perform better and converge more quickly. Preprocessing ensures the data is in an appropriate format, contributing to more precise and effective optimization outcomes.

3.4 Algorithmic Selection

Algorithm's selection is broadly classified into 2 types: heuristic and agent-based.

3.4.1 Heuristic-based

Among the heuristics employed in detection is the least resistance heuristic, which postulates that the optimal path is the one with the fewest energy requirements—that is, the path with the fewest undulations.

The best path is assumed to be the shortest path between two points, according to the shortest path heuristic. Off-road route recognition systems are able tcanthe optimal path through a acrossf diverse terrains by integrating several almultiples.

3.4.2 Agent-based

An agent-based method for route optimization uses autonomous agents, each representing a single vehicle or delivery unit, to choose the best routes. These autonomous agents communicate with their surroundings [21], with one another, and with pre-established rules to determine the best course of action based on local data. Efficient routing solutions emerge from the collective behaviour of these agents under the influence of several factors, including traffic conditions, delivery constraints, and dynamic environmental changes. This approach provides adaptability and scalability for challenging routes through complex terrain, making it ideal for expansive transportation networks.

3.5 Existing Methodology

The A* algorithm is used as a baseline against which the paths of other algorithms are tested and evaluated. A-star was chosen due to its simplicity and popularity in routing challenges.

3.5.1 A method:*

The A* algorithm is implemented on a grid map, where each location is represented by a cell, and the corresponding weight indicates the cost of traversing that cell. This is akin to real-world scenarios in which crossing each cell takes time. The proposed approach for maps should be seen in the same light.

1. Heuristic Cost: A calculation that approximates the remaining distance to the objective, usually based on the Manhattan distance.
2. G-Cost: The total cost incurred from the beginning location to the present cell.
3. The algorithm maintains separate and Enclosed Sets Two sets:
   - Unclosed Set Nodes that could be good candidates for exploration are kept in this group. A node's total cost—the sum of its G-cost and heuristic—is used to determine its priority.
   - Closed Configuration Nodes that have already been investigated and are not being revisited are included in this group.

The following steps are repeated by the A* search iteration:

1. Initialization: Initialize the Astar class with a matrix representing the grid terrain and convert it into a grid of Node objects.
2. Node Definition: Define a Node class representing a node in the grid, containing attributes for coordinates, weight, cost, and parent node.
3. Prepare Matrix: Convert the input matrix into a grid of Node objects.
4. Define the start and finish nodes for the pathfinding method, and initialize the open and closed lists. The operation iterates until the open list is empty. Pop the node with the lowest cost from the open list at the beginning of each iteration, and verify that it is still the goal node. The program will then recreate the ideal path, if it is. If not, create the current node's neighbors and adjust their costs appropriately. The current node is then transferred to the closed list, and these nearby nodes are added to the open list. Until the target node is located, this cycle is repeated.
5. Path Reconstruction: Reconstruct the optimal path from the start to the end node by backtracking through the parent nodes.
6. Calculate Parent's Weight: Sum the weights of nodes on the path from a given node to the start node.

3.5.1.1 Assumptions

1. ***Discrete Grid-Based Map:*** To make navigation easier, the code divides the world into uniform square cells using a grid-based representation.
2. ***Known Weights:*** The cost of visiting each grid cell is represented by a known, deterministic weight for each cell.
3. ***Static Environment:*** Throughout the pathfinding process, the algorithm assumes that there are no dynamic barriers or changes to the cell weights.
4. ***Perfect Knowledge:*** The algorithm assumes that all prior knowledge of the weight distribution and map layout has been obtained; exploration is unnecessary.
5. ***The Manhattan distance heuristic:*** It is effective for computation but might not always yield the most precise estimate in complicated surroundings. The algorithm uses it to estimate the remaining distance to the target.

3.6 Proposed Methodology

Proposing our own methods to trace paths in a map. For each algorithm, define specific formulas that help us determine the best path from source to destination. Thereafter, assumptions are mentioned.

3.6.1 Simulated Annealing

This algorithm, in the context of path finding, finds the optimal path from start to end point by navigating through all terrain with varying elevation differences on the heatmap. The cost involved in this represents the absolute sum of elevation differences between 2 adjacent nodes.

We have considered certain assumptions for efficient results, including that the elevation data be displayed as a smoothed-value heatmap. Figure 2 depicts the simulated annealing process. On an 8-neighborhood grid, movement is limited to neighboring spots. There are predetermined beginning and ending sites. The only basis for the cost function is the variation in elevation between places.

$$cost = mod(elevation\,[next] - elevation\,[previous]) \quad (1)$$

$$acceptance = exp[(currentcost - newcost)/ temperature] \quad (2)$$

The cost and acceptance of the model are determined by the formulas mentioned above. The cost formulas use the absolute value of the difference between the next and current elevations. Based on this cost, it determines the best neighboring cell. The acceptance rate is calculated as the exponential of the difference between the current and new costs, divided by the temperature at that time step. This acceptance rate determines whether the model will explore that path.

3.6.1.1. Assumptions

1) Discrete Grid-Based Map: To make navigation easier, the code divides the world into uniform square cells using a grid-based representation.
2) Static Environment: Throughout the pathfinding process, the algorithm assumes that there are no dynamic barriers or changes to the cell weights.
3) Temperature Schedule: SA assumes a predefined cooling schedule for decreasing the temperature parameter over iterations. This schedule influences the balance between exploration and exploitation during the optimization process.
4) Exploration of Solution Space: SA explores the solution space probabilistically, allowing for the acceptance of suboptimal solutions with a certain probability determined by the current temperature and the difference in cost between the current and new solutions.

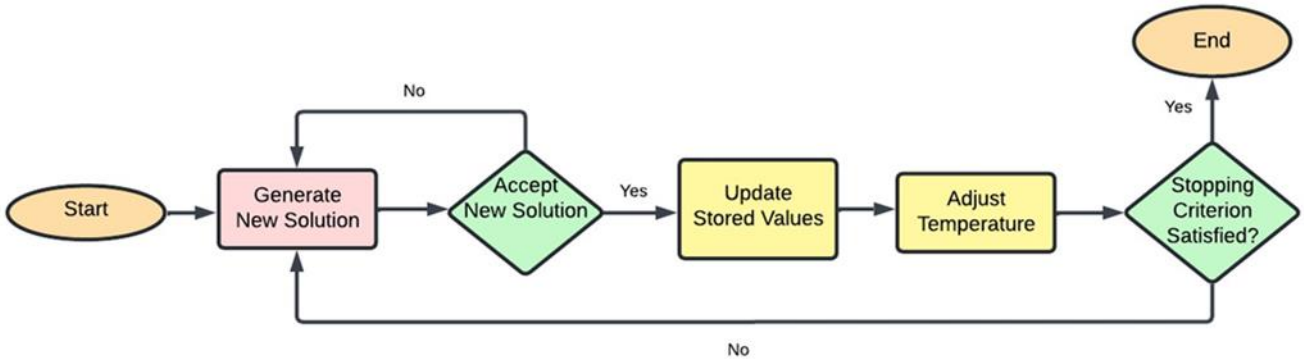

Figure 2. Simulated Annealing

3.6.2 Ant Colony Optimization

The Ant Colony Optimization (ACO) technique is applied in this study to solve the shortest path problem on a 3-dimensional grid-based map with elevation as the third dimension. The ACO algorithm is a bio-inspired method inspired by ants' foraging habits [24].

Ants are agents that traverse the terrain to find a path from the starting point to the destination. They keep track of their current position, the nodes they have visited, and whether they have reached the destination. Ants move through the terrain, construct paths, and reset their state after completing a path or iteration. Figure 3 explains the assumptions of Ant Colony Optimization. Trails of Pheromones is a matrix showing how desirable it is to navigate between the map's nodes. Every pheromone value is initially set to the same value. Pheromone levels on matching edges are increased to promote successful paths as the algorithm progresses.

The way the pheromone update rule and the heuristic function that direct ant movement are implemented can have a big impact on how well the algorithm works. These factors may be further tailored and researched to increase the effectiveness of the pathfinding process in particular issue areas.

$$\tau_{ij}(t+1) = (1 - \rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t) \quad (3)$$

Here:

$\tau_{ij}(t+1)$ ← updated pheromone value for the edge that connects the nodes i and j at time t+1.

$-\rho$ is the evaporation rate, which is a constant that lies in between 0 and 1.

$\tau_{ij}(t)$ ← existing pheromone value for the edge connecting nodes i and j at time t.

$\Delta\tau_{ij}(t)$ ← amount of pheromone deposited on the edge (i, j) by the ants that have completed their tours at time t.

3.6.2.1 Assumptions

1) ***Discrete Grid-Based Map:*** To make navigation easier, the code divides the world into uniform square cells using a grid-based representation.
2) ***Static Environment:*** Throughout the pathfinding process, the algorithm assumes that there are no dynamic barriers or changes to the cell weights.

3.6.3 Q-Learning

Q-learning is used for pathfinding in an environment where elevation data was examined in this work. Through trial-and-error interactions with its environment, an agent can learn an optimal policy through reinforcement learning techniques like Q-learning [21]. Representation of Environment: The environment is shown as a two-dimensional grid-world, with elevation values represented as a heatmap. Compared to conventional grid-based maps with uniform prices, the heatmap—which is considered to correlate to real-world elevation data—offers a more nuanced portrayal of the environment.

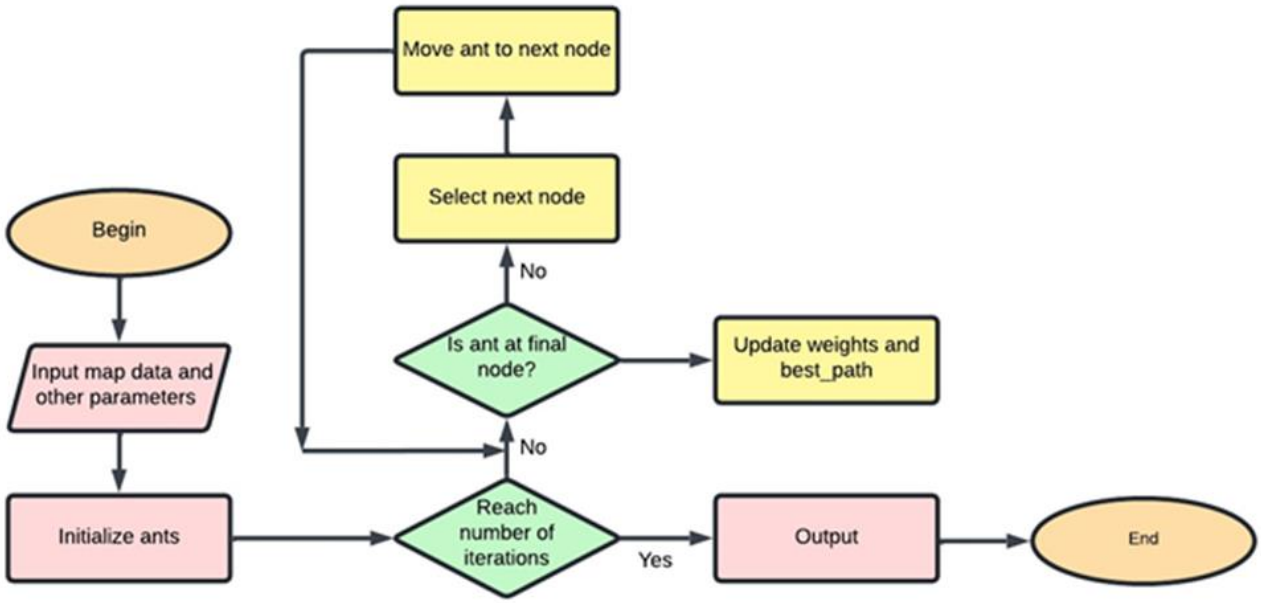

Figure 3. Ant Colony Optimization Assumptions

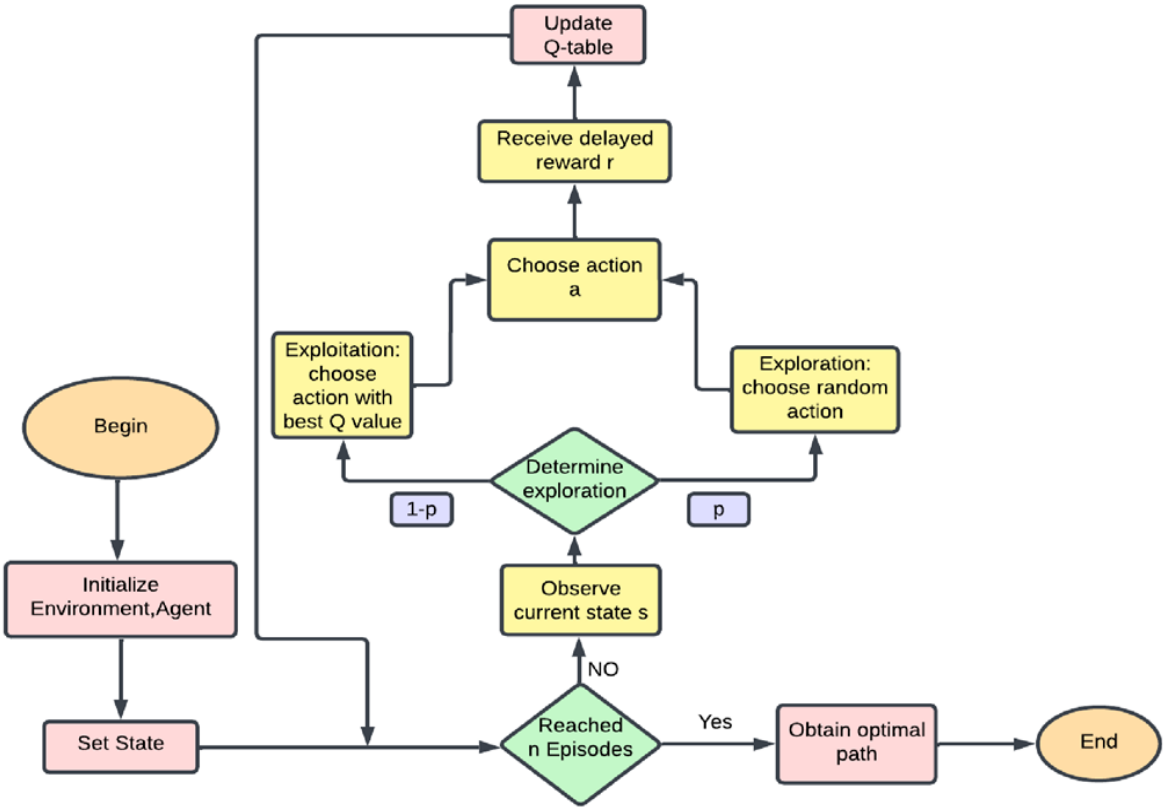

Figure 4. Q Learning

Figure 4 shows the Q-Learning process. The Q-Learning algorithm penalizes elevation variance mentioned in Equation 5 to minimize undulations.

$$Q(s,a) = (1 - \alpha) * Q(s,a) + \alpha * (r + \gamma * max(Q(s',a'))) \quad (4)$$

- Q(s, a) ← Q-value for the current state s and action a.
- α ← learning rate
- r ← immediate reward received after taking action in state s
- γ ← discount factor
- max(Q(s', a')) ← maximum Q-value for the next state s'

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(e_i - \bar{e})^2 \quad (5)$$

σ^2← elevation variance

n← number of segments or way-points

e_i← elevational at the ith point

$\bar{e}$ ← mean elevation of the currently selected path

3.6.3.1 Assumptions

1) ***Perception of Elevation Data:*** Q-learning assumes the agent has complete and accurate knowledge of the elevation data provided. In robotic process automation or autonomous vehicles, obtaining accurate environmental data is difficult and expensive. Due to this, there are discrepancies in the Q-value.
2) ***Epsilon-Greedy Exploration Strategy:*** The agent explores by taking random actions with probability epsilon, while all other actions are considered greedy. As this balances exploration and exploitation, the choice of or formulation of Epsilon and decay rate can heavily impact learning efficiency and performance.
3) ***Decaying Values for Discount Factor and Learning Rate:*** Discount factor and learning rate are often assigned Decaying values in Q-Learning to increase convergence rate and ensure stability. As a result, learners focus more on recent changes to improve performance in environments that tend to change rapidly.

3.7 Implementation

The table below shows the pseudocode for the SA, ACO, and Q-Learning algorithms. The hyperparameters listed below can be used to replicate the results in Table 1.

3.8 Hyper-parameters

The hyperparameters are chosen using a grid search method [25] for the SA, ACO and Q-Learning algorithms.

Table 1. Pseudo code for Simulated Annealing, Ant Colony Optimization, and Q-Learning algorithms

Name of Algo	Pseudo Code

Simulated Annealing	func simulated_annealing(heatmap, start, goal, T, cooling_rate):
	# 1. Initialization
	current = generate_initial_solution(heatmap, start, goal)
	best = current
	# 2. Find paths while temperature is greater than one
	while T > 1:
	#3. Find the best neighboring cell probabilistically
	neighbor = generate_neighbor(current, heatmap)
	current_cost = calculate_cost(current, heatmap)
	neighbor_cost = calculate_cost(neighbor, heatmap)
	if acceptance_probability(current_cost, neighbor_cost, T) > random(0, 1):
	current = neighbor
	# 4. Compare the current path cost and the original cost
	if neighbor_cost < calculate_cost(best, heatmap): best = neighbor
	# 5. Reduce the temperature after every iteration
	T *= cooling_rate return best
	func acceptance_probability(current_cost, new_cost, T):
	if new_cost < current_cost:
	return 1
	else:
	return exp((current_cost - new_cost) / T)

Ant Colony Optimization	func ant_colony_optimization(map, start, goal, num_ants, iterations, evaporation_rate, alpha, beta):
	# 1. Initialization
	pheromone_matrix = initialize_pheromone(map)
	best_path, best_length = None, infinity
	for _ in range (iterations):
	# 2. Find paths for each ant
	ant_paths = [construct_path(map, start, goal, pheromone_matrix, alpha, beta) for _ in range(num_ants)]
	# 3. Update best path if better path found
	for path in ant_paths:
	current_length = calculate_path_length(path) # Need to define this function if current_length < best_length: best_path, best_length = path, current_length
	# 4. Update pheromone levels
	update_pheromone(pheromone_matrix, ant_paths, evaporation_rate)
	return best_path
	func construct_path(map, start, goal, pheromone_matrix, alpha, beta):
	path = [start]
	# 5. Move ant until goal is reached
	while path[-1] != goal:
	neighbors = get_neighbors(path[-1], map)
	probabilities = calculate_transition_probabilities(path[-1], neighbors, pheromone_matrix, map, alpha, beta)
	next_node = choose_next_node(neighbors, probabilities)
	path.append(next_node)
	return remove_loops(path)

Q- Learning	func q_learning(heatmap, start, goal, α, γ, ε, ε_decay, episodes, λ):
	# 1. Initialization
	Q = zeros([rows, cols, num_actions])
	ε_current = ε
	# 2. Training episodes
	for episode in range(1, episodes+1):
	state = start
	#3. Episode rollout
	while state != goal:
	if random(0,1) < ε_current:
	action = random_action(state)
	else:
	action = max(Q[state, all_actions])
	next_state = move(state, action)
	# using both the cost functions results in a much more traversable path distance_cost = euclidean_distance(state, next_state)
	elevation_change_cost = abs(heatmap[next_state_elevation] - heatmap[state_elevation]) reward = - (distance_cost + λ * elevation_change_cost) if next_state == goal:
	reward += goal_reward
	next_best_q = max(Q[next_state, all_actions])
	Q[state, action] += α * (reward + γ * next_best_q - Q[state, action])
	state = next_state
	# 4. Decay exploration rate
	ε_current = max(ε_current * ε_decay, ε_min)
	# 5. Greedy path extraction
	path = [start]
	state = start while state != goal:
	action = max(Q[state, all_actions])
	state = move(state, action)
	path.append(state)
	return path

Table 2. Hyperparameters for SA, ACO, and Q-Learning

Model	Hyper parameters	Implemented Values
Simulated Annealing	Temperature	3
	cooling_rate	10^{-3}
	number of iterations	10^{4}
Ant Colony Optimization	Iterations	2010
	Ants	30
	ρ(evapouration_rate)	0.4
	Early_Stop	10
Q-Learning	learning_rate	0.001
	Discount_factor	0.9
	Epsilon	0.1

Using these hyperparameters (Table 2) in the models provides a high degree of flexibility to increase the model's efficiency. Parameters used are decided after many trials and errors.

4. Result and Discussions

The above mentioned metrics can be used to properly evaluate results from the algorithms. These metrics account for runtime of algorithm, length of the route, undulations in path and the smoothness of the path. By using these metrics, the performance of the algorithm can be properly evaluated. Explanation of the metrics is given below:

1. ***Runtime:*** The time, in seconds, needed to complete the execution of an algorithm, i.e., to get the route. A lower value signifies that the algorithm needs less time to run. Lower is better.
2. ***Elevation difference:*** It is the modulus of the difference between the elevations of each pixel from the next pixel. This signifies the smoothness of the route. A higher value means the route consists of more undulations, is rugged, and is difficult to traverse. A lower value means the route is smooth and safer to traverse. Lower is better.

$$\Delta E = \sum_{k=1}^{n-1} |E(i_k, j_k) - E(i_{k+1}, j_{k+1})| - |E(x_1, y_1) - E(x_1, y_1)| \quad (5)$$

3. ***Distance: Displacement ratio:*** It is the length of the route(displacement) divided by the Euclidean distance between the start and end points(displacement). This value is bound to be greater than or equal to 1. A distance-to-displacement ratio of 1 signifies that the route is a straight line from start to destination; any turns in the route will increase the value from 1. A higher value indicates a longer route. Lower is better.

$$\boldsymbol{R} = \frac{d(src,dest)}{\sum_{k=1}^{n-1} |d((i_k, j_k) - (i_{k+1}, j_{k+1})} \quad (6)$$

By using the above-mentioned metrics, we can analyze and compare the different algorithms, understanding their advantages, disadvantages, and practicality for each terrain. Using these metrics we can pinpoint the areas where each algorithm lacks and excels. This detailed comparison between each algorithm is given below.

Figure 5 shows us the routes given by each algorithm. In Figure 5(A), A* has given a route with many undulations as it passes through 2 high-altitude regions (0,5) and (9,12), resulting in a large elevation difference. Figure 5(B) shows that Simulated Annealing has a very high distance-to-displacement ratio and a high elevation difference, due to all the fluctuations in the route obtained by the SA algorithm. In Figure 5(C), ACO's route on this map is the best, as it results in the lowest distance-to-displacement ratio and the least elevation difference. Barring the fluctuation in route at (9,10) ACO gave an optimal path. In Figure 5(D), Q-Learning shows a moderate elevation difference, resulting from the bad route fluctuation at (11,7); it also has a good distance-to-displacement ratio, indicating a shorter route.

As shown in Figure 6, on comparing the mean values from 5 runs on the same data, we notice that Q-Learning provides us with the best performance in all our metrics, with very close performance to Ant Colony Optimization in terms of Distance to displacement ratio. The high difference in runtimes compensates for that small margin in distance/displacement ratio.

Elevation difference, runtime, and distance/displacement metrics have been recorded for 5 runs of each algorithm on the same dataset. Figure 7 shows the variability in the results of each algorithm. The graphs show that SA has the highest variance in elevation difference and distance/displacement (which determine the algorithm's accuracy), while ACO has the highest variability in algorithm runtimes. This is due to the high complexity of the algorithm (contributing to high time complexity) and to random ant initialization (contributing to high variability). A* has very low variability, which is a good factor, but as mentioned above, it performs worse on complex terrains. Q-Learning has very low variability and achieves high accuracy across all terrain types.

Figure 7(a) shows that simulated annealing has the highest elevation difference and also exhibits high variability. Thus, the route given by SA has the most undulations. On the contrary, Q-Learning has the lowest elevation difference and variance, thus giving a route with the fewest undulations.

Figure 7 (b) shows algorithm runtimes. From the graph, ACO has the highest runtime, whereas Q-Learning and A* have the lowest. SA's variability can be seen in its runtime graph too.

Figure 7(c) shows that SA's has the highest variability, indicating a longer route. ACO and Q-Learning's distance-to-displacement ratio is the lowest, indicating a short route. A* performs worse than ACO and Q-Learning in this metric.

Table 3 shows the mean metrics of each algorithm. This table suggests that while A* and simulated annealing perform moderately well on runtime and distance/displacement, they exhibit high elevation differences, meaning the routes they produce have many undulations. ACO instead gives a low elevation difference but has high runtime. Q-Learning, while having good runtime and a distance/displacement ratio, also has a low elevation difference, resulting in fewer undulations and a smoother route, like ACO.

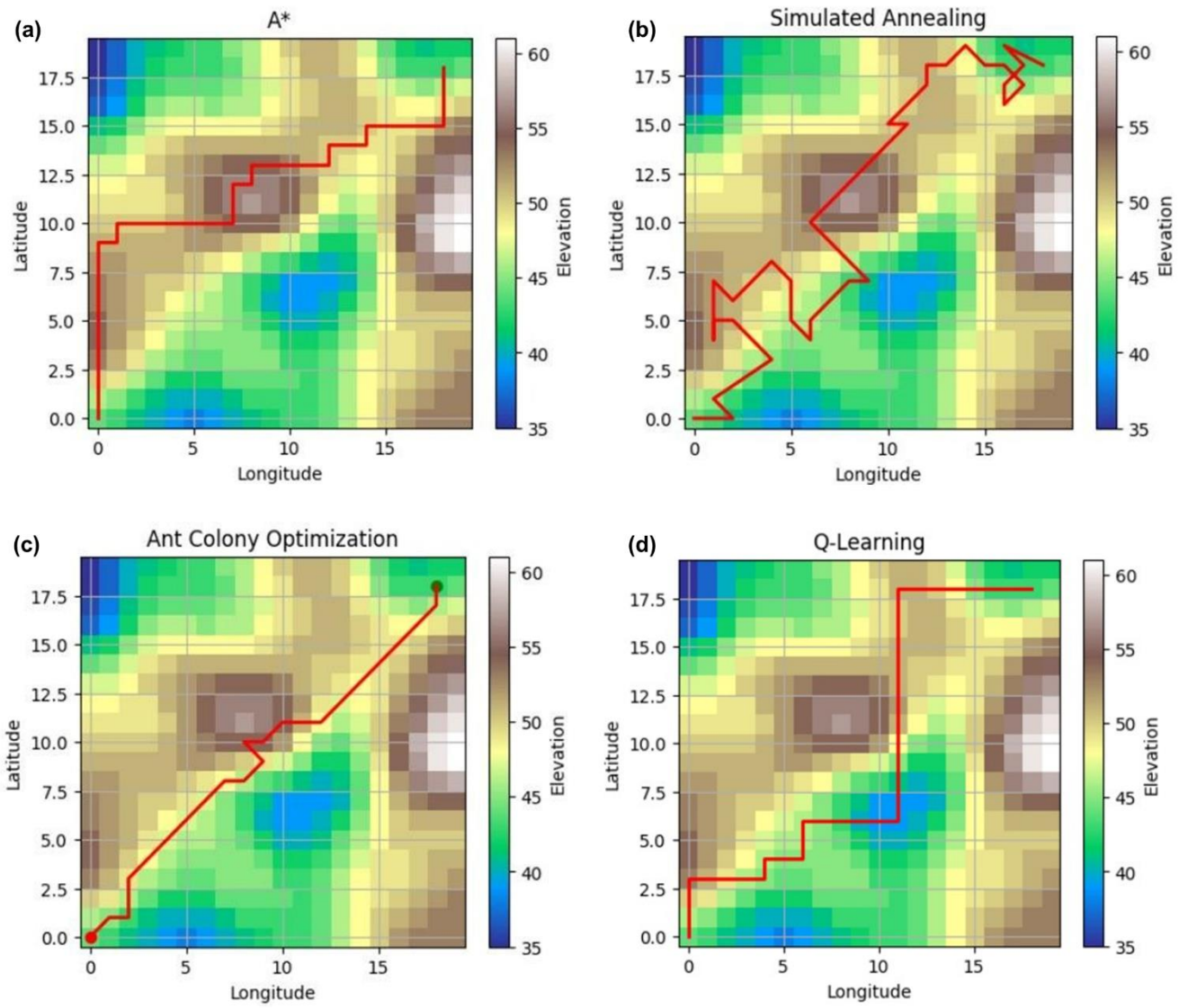

Figure 5. Routes given by A) A* B) Simulated Annealing C) ACO D) Q-Learning

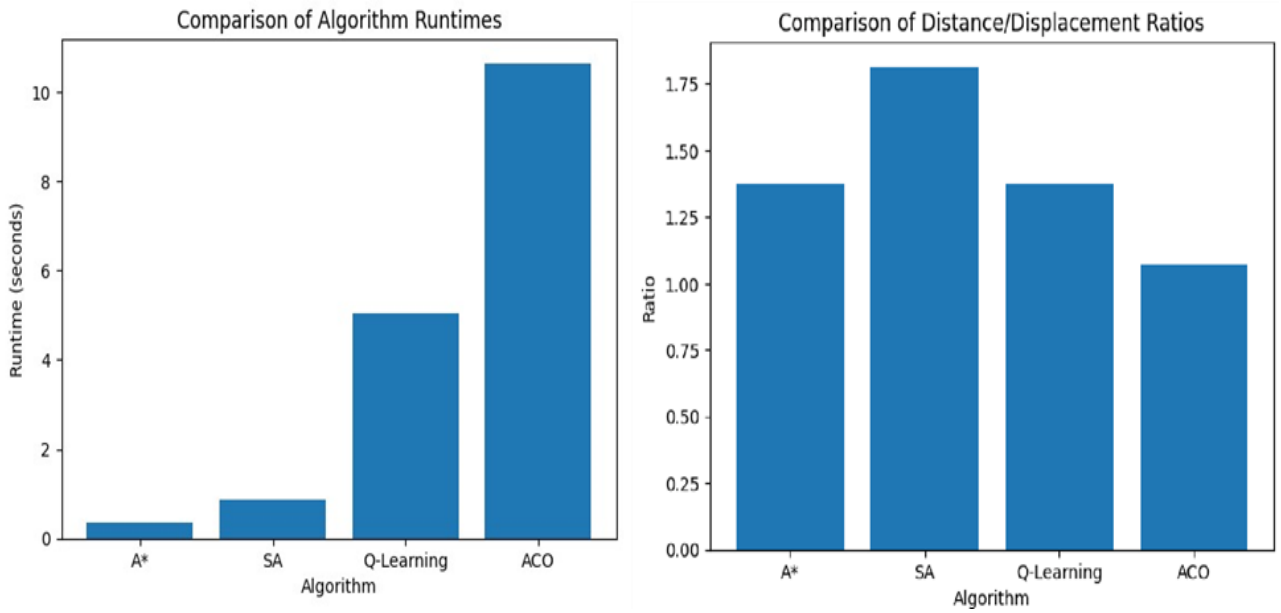

Figure 6. Bar Charts of Algorithm Runtime (left), Distance/Displacement (right)

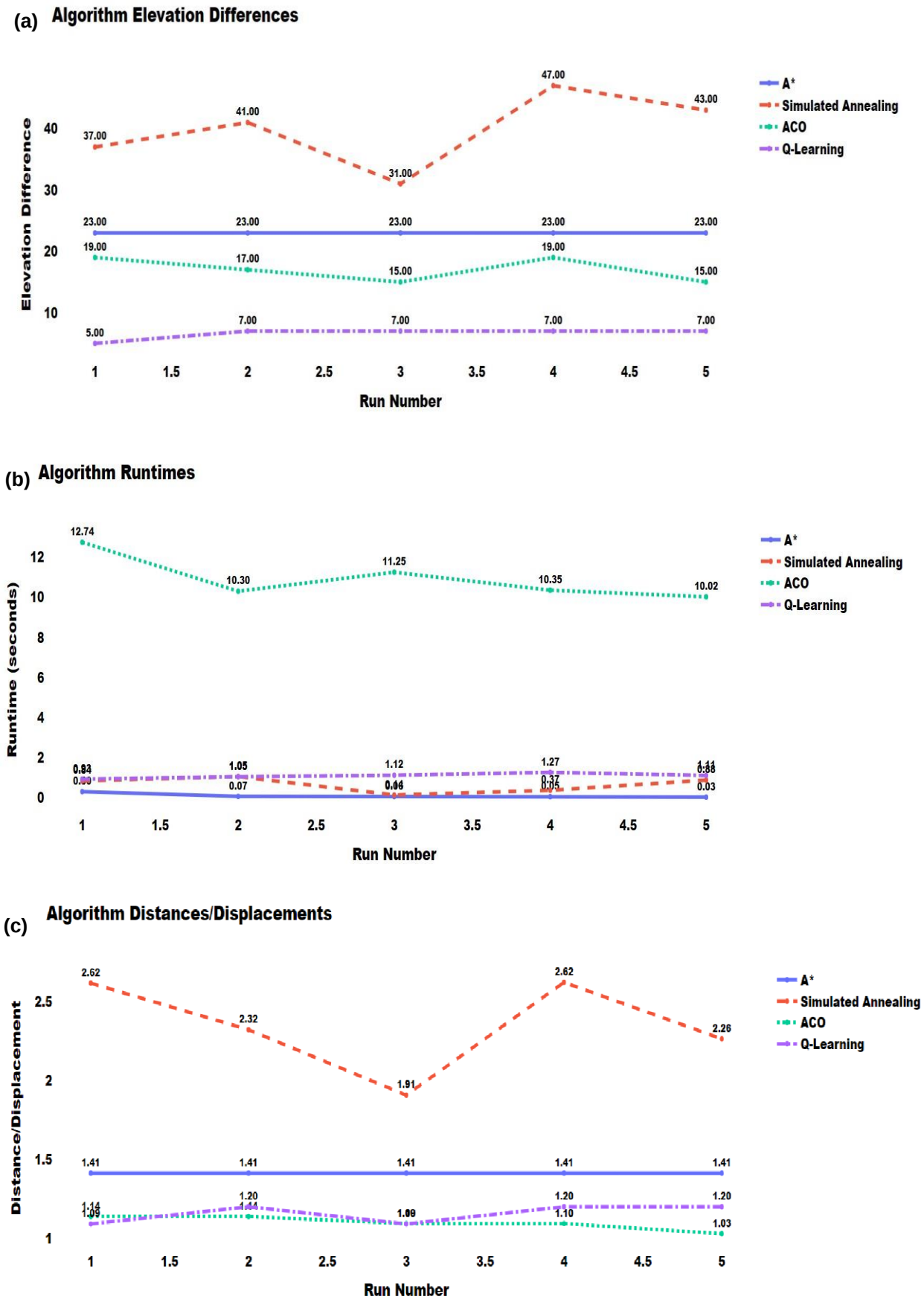

Figure 7(a) Line Charts of Algorithm Elevation Differences, **(b)** Line Charts of Algorithm Runtim, **(c)** Line Charts of Distance/Displacement

Table 3. Comparison of all Algorithms

Name of algorithm	A*	Simulated annealing	ACO	Q-learning
runtime	0.04	1.172	11.458	1.156
Distance/Displacement	1.41	2.878	1.09	1.18
Elevation Difference	23.0	44.20	17.40	6.80

6. Conclusion and Future Work

Realizing that real-world pathfinding requires more than just the shortest path, this study set out to evaluate the performance of 4 pathfinding algorithms when terrain smoothness is also taken into account. Through rigorous experimentation and testing, it was found that Q-learning is the most effective approach for off-road terrain routing.

Traditional methods like A* produced rugged, suboptimal routes, often getting lost when dealing with large, complex maps and terrain. Q-learning overcomes this limitation by producing paths with 70% fewer elevation undulations and 16% shorter path lengths. In contrast, ACO produces paths of comparable accuracy, and sometimes better paths, but it suffers from high runtime, making it unsuitable for real-time applications. Simulated annealing, while helpful for escaping local minima, produces inconsistent results and longer routes, often changing the path when used with the same start and end points on the same map. Overall, Q-learning's ability to learn and adapt to complex terrain and its fast execution, even on difficult, rough terrain, make it an excellent choice for off-road navigation systems.

References

[1] X. Liang, Y. Mu, B. Wu, Y. Jiang, A review of related algorithms for path planning. Value Engineering, 39(03), (2020) 295–299.

[2] J.C. Yang, S.X. Li, Z.Y. Cai, Research and development of path planning algorithm. Control Engineering, 24(07), (2017) 1473–1480.

[3] M. Korkmaz, A. Durdu, (2018) Comparison of optimal path planning algorithms. In 2018 14th international conference on advanced trends in radioelecrtronics, telecommunications and computer engineering (TCSET), IEEE, Ukraine, https://doi.org/10.1109/TCSET.2018.8336197

[4] A.J. Dalton, (2008) Autonomous Vehicle Path Planning with Remote Sensing Data. Virginia Tech.

[5] B. Wang, S. Li, J. Guo, Q. Chen, Car-like mobile robot path planning in rough terrain using multi-objective particle swarm optimization algorithm, Neurocomputing, 282, (2018) 42–51. https://doi.org/10.1016/j.neucom.2017.12.015

[6] X. Zhou, Q. Xie, M. Guo, J. Zhao, J. Wang, Accurate and efficient indoor pathfinding based on building information modeling data, IEEE Transactions on Industrial Informatics, IEEE, 16(12), (2020) 7459–7468. https://doi.org/10.1109/TII.2020.2974252

[7] Y. Ma, S. Liu, B. Sima, B. Wen, S. Peng, Y. Jia, A precise visual localisation method for the Chinese Chang'e-4 Yutu-2 rover. The Photogrammetric Record, 35(169), (2020) 10–39. https://doi.org/10.1111/phor.12309

[8] H.B. Duan, (2005). Ant colony algorithms: theory and applications. Chinese Science.

[9] E. Shi, M. Chen, J. Li, Y. Huang, Research on method of global path-planning for mobile robot based on ant-colony algorithm. Transactions of the Chinese society for agricultural machinery, 45(6), (2014) 53-57.

[10] C. Zheng, H. Yin, J. Li, M. Lu, Multi objective evolutionary algorithm for shortest path with maximal visual coverage, In 2011 International Conference on Intelligent Computation and Bio-Medical Instrumentation IEEE, 232-235.

[11] J. Leng, X. Yu, (2023) Research on personalized travel route recommendation AI algorithm based on simulated annealing, IEEE 3rd International conference on electronic communications, Internet of Things and Big Data (ICEIB), IEEE, Taiwan. https://doi.org/10.1109/ICEIB57887.2023.10169965

[12] A. Panov, K. Yakovlev, R. Suvorov, Grid path planning with deep reinforcement learning: Preliminary results, Procedia Computer Science, 123, (2018) 347–353, https://doi.org/10.1016/j.procs.2018.01.054

[13] H.D. Pham, S.M. Narasimhamurthy, B. Mehran, E. Manley, A. Ashraf, "Reinforcement learning based estimation of shortest paths in dynamically changing transportation networks, Frontiers in Future Transportation, 6, (2025) 1524232. https://doi.org/10.3389/ffutr.2025.1524232

[14] Z. Hong, P. Sun, X. Tong, H. Pan, R. Zhou, Y. Zhang, Y. Han, J. Wang, S. Yang, L. Xu, Improved A-star algorithm for long-distance off-road path planning using terrain data map. ISPRS International Journal of Geo-Information, 10(11), (2021) 785. https://doi.org/10.3390/ijgi10110785

[15] P. Upadhyay, V. Marriboina, S.J. Goyal, S. Kumar, E.S.M. El-Kenawy, A. Ibrahim, A.A.

Alhussan, D.S. Khafaga. An improved deep reinforcement learning routing technique for collision-free VANET, Scientific Reports, 13(1), (2023) 21796. https://doi.org/10.1038/s41598-023-48956-y

[16] W. Alhoula, J. Hartley, (2014) Static and time-dependent shortest path through an urban environment: Time-Dependent Shortest Path, in Science and Information Conference (SAI), IEEE, London, UK. https://doi.org/10.1109/SAI.2014.6918315

[17] Z. Sun, (2022) Applying reinforcement learning for shortest path problem International Conference on Big Data, Information and Computer Network (BDICN), IEEE, China https://doi.org/10.1109/BDICN55575.2022.00100

[18] D. Shah, A. Bhorkar, H. Leen, I. Kostrikov, N. Rhinehart, S. Levine. (2022) Offline reinforcement learning for visual navigation. arXiv preprint arXiv:2212.08244. https://doi.org/10.48550/arXiv.2212.08244

[19] S. Huang, X. Wu, G. Huang. (2023) Deep reinforcement learning-based multi-objective path planning on the off-road terrain environment for ground vehicles. arXiv preprint https://doi.org/10.48550/arXiv.2305.13783

[20] Y. Hu, L. Yang, Y. Lou, Path planning with Q-learning. Journal of Physics: Conference Series, 1948, (2021) 012038.

[21] B. Jang, M. Kim, G. Harerimana, J. Kim, Q-learning algorithms: A comprehensive classification and applications, IEEE Access, 7, (2019) 133653–133670. https://doi.org/10.1109/ACCESS.2019.2941229

[22] A.P. Mohamed, B. Lee, Y. Zhang, M. Hollingsworth, C.R. Anderson, J.V. Krogmeier, D.J. Love. (2024) Simulation-Enhanced Data Augmentation for Machine Learning Pathloss Prediction. In ICC 2024-IEEE International Conference on Communications, IEEE, USA. https://doi.org/10.1109/ICC51166.2024.10622237

[23] S. Wuqiang, S. Guiquan, C. Lei, Z. Jijun, (2023) Data Preprocessing Optimization Algorithm Based on Improved Particle Swarm Optimization. In International Conference on Networking, Informatics and Computing (ICNETIC), IEEE, Palermo, Italy, https://doi.org/10.1109/ICNETIC59568.2023.00152

[24] M. Dorigo, M. Birattari, T. Stützle, Ant colony optimization. IEEE computational intelligence magazine, IEEE, 1(4), (2006) 28-39. https://doi.org/10.1109/MCI.2006.329691

[25] P. Liashchynskyi, P. Liashchynskyi, (2019) Grid search, random search, genetic algorithm: A big comparison for NAS. arXiv preprint. https://arxiv.org/pdf/1912.06059

Authors Contribution Statement

Jay Kansara: Conceptualization, visualization, methodology, Writing original Draft. Tanmay Mistry: Methodology, Data analysis, Writing, Review & Editing. Vedant Bhawnani: Methodology, Validation, Formal Anlaysis, Writing, Review & Editing. Dwiti Choksi: conceptualization and visualization, Data Curation, Writing, Review & Editing. Lakshmi Kurup: Supervision, Formal Analysis, Validation. Pratik Kanani: Supervision, Formal Analysis, Validation, Writing-reviewing. All the authors read and approved the final version of the manuscript.

Funding

The authors declare that no funds, grants or any other support were received during the preparation of this manuscript.

Competing Interests

The authors declare that there are no conflicts of interest regarding the publication of this manuscript.

Data Availability

The data supporting the findings of this study can be obtained from the corresponding author upon reasonable request.

Has this article screened for similarity?

Yes

About the License