Intelligent Navigation System: Leveraging Dijkstra, A* Search Algorithms, and Reinforcement Learning for Traffic-Aware Shortest and Most Efficient Pathfinding in Real-World Scenarios

Victor Robinson
School of Computing, Engineering and Digital Technology
Teesside University
Middlesbrough, England
D3739635@live.tees.ac.uk

***Abstract*—This report details the design and implementation of an intelligent navigation system that incorporates AI for real-time traffic-aware, optimal pathfinding in real-world settings. The system utilizes Dijkstra's uninformed search and A-star informed search for initial static shortest pathfinding from source to destination, along with reinforcement learning (RL) for real time dynamic path optimization and rerouting. This approach addresses path finding and optimization issues and real-world challenges such as traffic congestion, road closures, one-way roads, and other obstacles, making it applicable to human navigation, autonomous vehicles, robotic systems, and more. The performance of Dijkstra's uniformed search algorithm is evaluated based on various costs represented as weights, such as distance and travel time. A-star uses geodesic and haversine distance formulas as heuristics to calculate the great-circle distance along the Earth's surface, providing more accurate results and effectively guiding the search toward the goal. Reinforcement learning is evaluated based on the time steps required to reach the goal, its effectiveness in identifying a route, and its ability to determine alternative routes in scenarios such as road blockades and traffic congestion. The results highlight the importance of travel time as the true cost of traversing a route for search algorithms, as well as the static nature of traditional search methods. Additionally, it also highlights the effectiveness of RL in adapting and rerouting for certain road issues, along with its computational complexity in optimal pathfinding.**

Keywords—Pathfinding, Path Optimization, Dijkstra, A*, Reinforcement Learning, Shortest Path, Navigation System, AI

I. Introduction

The utility of Artificial Intelligence has grown rapidly over the years, gaining significant attention. AI has evolved into a valuable tool for performing various tasks and is utilized in numerous application areas, including intelligent navigation systems that determine the optimal path from a source to a destination within a road network for a specific location. In a broad sense, navigation refers to the process of determining position to plan and follow a route. Navigation system for road networks based on digital maps, aids this process by providing information on suggested directions, traffic conditions, and alternative routes to assist both humans and autonomous vehicles [8].

Intelligent navigation systems utilize pathfinding methods for initial route planning. In road networks, pathfinding involves identifying the shortest route between two points on a coordinate system. This process utilizes a variety of both uninformed and informed search algorithms to determine the shortest and most efficient path from a starting location to a destination [13]. The significance of route planning lies in its ability to reduce travel time complexity for traversing between these points [15]. This initial process is considered static, as it is applied to fixed environments under the assumption that certain details of the environment do not change [8] and it does not have the ability to explicitly account for real-world challenges.

For advanced dynamic route planning, it is essential to consider real-time traffic conditions, road closures, and other dynamic blockades in real-world scenarios. This involves identifying the optimal path and rerouting when such challenges arise. Reinforcement learning provides significant advantages in addressing complex routing optimization problems [9]. It leverages the concepts of intelligent agents and environments, where the agent represents a vehicle, and the environment represents the dynamic road network. Through iterative learning and adaptation, agents can determine the most efficient routes. The primary objective of the agent is to learn an optimal path that allows it to navigate a dynamic road network efficiently, reach its destination, avoid real-time obstacles and delays, maximize performance, and minimize travel time. By interacting with the environment, agents develop decision-making capabilities that closely mimic real-world behavior. This approach enables the agent to explore the network, identify the most effective routes, and dynamically adjust to real-world changes such as traffic congestion or road closures.

References [13], [15] and [4] explored various search algorithms for route planning, providing a comparative analysis of their performance based on costs such as distance and travel time to determine the shortest path from a source to a destination. While their research yielded valuable results, it did not account for dynamic road networks or real-world challenges like traffic congestion and road closures. Additionally, the static search approach in [15] used Euclidean distance as the heuristic function for the A* search algorithm. The Euclidean distance calculates the straight-line distance between two points on a flat plane. However, since the Earth is not flat and is more accurately represented as a sphere or ellipsoid, a more realistic heuristic, such as the haversine distance, would have been more appropriate for calculating distances..

This work aims to develop a traffic-aware intelligent navigation system capable of identifying the shortest and most efficient route from a starting location to a destination in a dynamic environment using reinforcement learning. The

system combines Dijkstra's uninformed search and A* informed search for initial static path planning with reinforcement learning to tackle real-world challenges in real time, such as traffic congestion, road closures, one-way streets, and other dynamic obstacles. The system is designed to operate with minimal real-world data, simulating a scenario where only basic map and coordinate tools are available for navigation, without relying on advanced mapping technologies. This approach is applicable to a variety of use cases, including autonomous vehicles, robotic movement, and human navigation. The ultimate goal is to identify the shortest and most efficient path while dynamically adapting to real-time traffic conditions and road blockades.

II. Methodology

This section entails methods utilized for this study, which are highlighted in the subsections below.

A. Graph Model

The road network of the selected location is modeled as a graph consisting of nodes and edges. Each node represents a specific point on the map, identified by its unique ID and coordinates, while each edge connects two nodes. Edges also store detailed information about the roads, including the road name, type of highway, number of lanes, maximum speed, one-way status, distance, and estimated travel time. A road network can be modelled as a multidimensional-weighted graph $G = (V, E)$, where $V = \{v_1, v_2, \dots v_n\}$ represents n vertices or nodes in the graph and $E = e_1, e_2, \dots e_m$ represents m edges (roads) in the graph [Li et al]. This process is done using [3], a Python package to easily download, model, and visualize road networks. [3] utilize data from OpenStreetMap, a free and open-source mapping tool. The road network is downloaded with the 'network_type' parameter set to 'drive', which retrieves a street network specifically for driving. Two-way streets are represented as bidirectional by default, while one-way streets are directed, allowing travel only in the specified direction. This ensures that the graph accurately mirrors the structure and features of the real-world road network. To manage computational complexity, the map and its graph model is limited to a 2100-meter radius around the center of Middlesbrough, England.

Fig. 1. OSMnx graph model of the location

B. Pathfinding with Dijkstra

Dijkstra's uninformed search algorithm is a widely used method for finding the shortest path between nodes in a weighted graph. It is classified as a greedy algorithm because it selects the best immediate option at each step without considering future consequences [18]. This algorithm is especially effective for solving shortest path problems in road networks, calculating the optimal route from a starting node to a destination node based on edge weights, such as distance or travel time. These weights must be positive numbers [21].

Dijkstra's algorithm initializes the start node with a weight of 0 and assigns an infinite weight to all other nodes. In this context of road network, the weight can represent travel time or distance. The algorithm operates by repeatedly identifying the unvisited neighboring node with the smallest weight from the current node. It then updates the weights of these neighboring nodes if a small weight is found. This is achieved by adding the weight of the current node to the weight of the edge connecting it to the neighboring node. The description of the algorithm is as follows:

1. Create a set containing all unvisited nodes.
2. Assign an initial weight to every node in the set of unvisited nodes. Set the value of the starting node to 0 and assign infinity to the remaining nodes.
3. From the set of unvisited nodes, select the node with the smallest weight as the current node. Initially, this will be the starting node, which has a weight of 0. The process terminates if the current node is the goal node.
4. For the current node, identify its unvisited neighbor nodes. Then, update the weight of each neighbor by adding the edge weight from the current node to that neighbor, but only if the new weight is smaller than the current weight of that neighbor. For example, if the current node A has a weight of 5 and the edge connecting it to the unvisited neighbor B has a weight of 3, the total weight from A to B will be 8. If the current weight of B is greater than 8, update it to 8, as this represents a shorter path. Otherwise, keep the existing weight for B.
5. Remove the current node from the unvisited node set after processing all its neighbors, ensuring that a visited node is never rechecked. Then, repeat step 3.
6. As the loops terminates, each visited nodes represent its shortest path tree from the start node.

The runtime complexity of Dijkstra's algorithm is $O(|V|^2)$, where $|V|$ represents the total number of nodes, which is always a positive value [18]. This complexity applies when the algorithm uses a basic min-priority queue. However, an advanced priority queue proposed by [10], called the Fibonacci heap priority queue, can be used to optimize the runtime complexity to $O(|E| + |V|\log |V|)$, where $|E|$ denotes the total number of edges.

The multidimensional weighted graph model of the map represents the road network, with edge weights such as distance (in meters) and travel time (in seconds), making Dijkstra's algorithm an ideal choice used to compute the shortest path between the specified start node and the destination node, using distance and travel time as weights.

Fig. 2. Optimal path by Dijkstra with distance as the weight

Fig. 3. Optimal path by Dijkstra with travel time as the weight

Fig. 4. Realtime Optimal path by OSM

The path generated by OpenStreetMap (OSM) in Fig. 4 may differ because it accounts for real-time factors such as traffic congestion, road closures, and other dynamic conditions. In contrast, Dijkstra's search algorithm, using travel time as the weight, as shown in Fig. 3, identifies the most optimal path, aligning closely with the live data from OpenStreetMap.

C. Pathfinding with A-Star (A)*

A* is a pathfinding algorithm similar to Dijkstra's but with key differences. While both solve shortest path problems, A* focuses on finding the shortest path from a source node to a destination node [14]. It does this by combining the weight of edges, with a heuristic function [11]. This heuristic function estimates the remaining distance to the destination, allowing A* to prioritize paths that are more likely to lead to the goal efficiently.

A* is an informed search algorithm, meaning it uses additional information to guide the search towards the goal efficiently. The primary objective of the algorithm is to find the least-cost path from the start node to the goal node. At each step, the algorithm evaluates the neighboring nodes of the current node. For each neighbor, it calculates:

1. $g(n)$: The cost of the path from the start node to that neighbor n using its weight (distance or travel time).
2. $h(n)$: A heuristic estimate of the cost from n to the goal node.

It then selects the neighbor with the smallest total cost, computed as: $f(n) = g(n) + h(n)$ [11].

The heuristic $h(n)$ plays a critical role in directing the search toward the goal. For A* to guarantee the least-cost path, the heuristic must be admissible (Hart, Nilsson and Raphael, 1968) which means that it must never overestimate the true cost to reach the goal. In other words, $h(n)$ must always be less than or equal to the actual cost from the current node to the goal.

A* uses a priority queue to repeatedly select the node with the lowest f-value that needs to be processed. At each step, the node with the lowest cost is removed from the queue, its neighbors are processed, and their g and h values are updated before being added to the queue. This continues until a removed node is the goal. A mechanism can then be used to keep track of the parent node of each processed node, allowing the processed nodes to be traced in reverse to get all nodes from the start to the goal.

Fig. 5. Shortest path by A* search algorithm with Geodesic (WGS-84) distance formula as the heuristics function and travel time as the weight

The time complexity of A* depends on the quality of the heuristic, as it significantly impacts the performance of the

algorithm. However, the space complexity of A* is denoted as $O(b^d)$ where d is the depth of the shortest path, and b is the branching factor as it stores all nodes in memory [14].

To achieve more accurate results, the Haversine and Geodesic (WGS-84) formulas were used to compute the heuristics. These formulas determine the shortest distance between two points on the Earth's surface, considering its curvature. This distance, known as the great-circle distance, is calculated along a sphere or ellipsoid, unlike the Euclidean distance, which measures a straight-line distance between points on a flat plane.

1) Haversine Distance: distance measured along the surface of a spherical earth [2].

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + cos\varphi_1 * cos\varphi_2 * sin^2\left(\frac{\Delta\lambda}{2}\right)$$
$$c = 2 * atan2\left(\sqrt{a}, \sqrt{1-a}\right)$$
$$d = R.c$$

Where: φ is latitude, λ is longitude, R is earth's Radius (mean radius = 6,371e3 meters) and all angles in radians .

2) Geodesic (WGS-84) Distance: for a more accurate result, distance measured along the surface of an ellipsoidal earth [1].

$$f = (a-b)/a$$
$$tanU_{1/2} = (1-f) * tan\varphi_{1/2}$$
$$L = \lambda_2 - \lambda_1$$
$$s = b * A(\sigma - \Delta\sigma)$$

Where: $a\ and\ b$ are major & minor semi-axes of the ellipsoidal, f is flattening, U is the reduced latitude, φ is the latitude, L is the difference in longitude and s is the geodesic distance along the surface of the ellipsoid

D. Pathfinding with Reinforcement Learning

Unlike supervised and unsupervised learning techniques that ingest, analyze, and learn from data to make predictions, RL learns to solve problems without relying on large datasets, predefined solutions, or preprogrammed actions. In RL, agents learn from their environment by interacting and receiving feedback for each action. The ultimate goal of the agent is to learn a policy that maximizes long-term rewards [19].

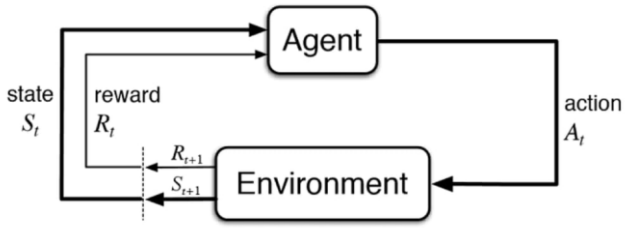

Fig. 6. Reinforcement learning process. Image from https://www.altexsoft.com/

Search algorithms are static, meaning they cannot explicitly detect real-world issues in dynamic environments. This limitation highlights the need for RL to adapt to real-world scenarios. RL can find the optimal route by accounting for real-time factors such as traffic and road closures. The agent explores the environment and learns from each interaction to determine the best route, without relying on predefined cost functions or heuristics, guided solely by the consequences of its actions.

1) Agent: An agent perceives its environment through sensors and interacts with it to gather information. In the context of dynamic route planning, the agent represents a vehicle that navigates the road network to determine the most efficient path to its goal while avoiding real-time issues like traffic and closures.

2) Environment: The environment is the scenario the agent interacts with. In this case, it is the driving road network of Middlesbrough, UK, limited to a 2100-meter radius to manage computational complexity. As explained in [A], the map is modeled as a multidirectional weighted graph, with nodes and edges connecting them. As it is a real world map, the environment is continuous, dynamic and has an infinite state space meaning it has an almost infinite possibilities and the features like traffic change over time.

3) Markov Decision Process (MDP): The MDP is a useful mathematical framework used to model RL problems with full observability. In this case, the system can be modeled as a tuple (S, A, R, P, γ), where S denotes the state space, A denotes the action space, R denotes the reward function, P denotes the transition probability, and γ denotes the discount factor [12].

a) State Space: This provides a complete description of the environment, with all the necessary information required for the agent to reach its goal. In the context of the road network, the state space consists of all the nodes in the graph, including traffic nodes, road closure nodes, as well as the start and goal nodes. Even though the agent can only visit a node once, the current state is 1,772 which is the total number of nodes in the graph.

b) Action Space: The action space represents all possible moves the agent can make in the environment, considering that each node represents a point on the coordinate system. At each time step, the agent can only move to one of its neighbors, excluding its parent node. This makes the action space dynamic, as different nodes may have a varying number of neighbors.

c) Reward: The reward is a value that indicates how well an agent is performing [19]. Each action is assigned a reward value, which can be either positive or negative. As the agent learns from interactions, the reward provides feedback on each action. In the case of route planning, the agent receives a penalty of -1 for each step taken, encouraging it to prioritize the shortest routes or steps. The agent receives a penalty of -10 if it reaches a dead end, meaning there are no further nodes beyond that point, and the episode terminates. Similarly, it receives -10 for closure nodes, which are treated as dead ends, and -5 for traffic. The agent is rewarded with +100 for reaching the goal.

d) Discount Factor: The discount factor is another parameter used to balance exploration and exploitation. It ensures that the agent prioritizes future rewards over immediate gratification. It is denoted as $(0 < \gamma \leq 1)$.

4) Q-learning RL algorithm: Q-learning is a model-free, value-based RL algorithm designed to identify the best actions for an agent based on its current state [20]. It achieves this by updating Q-values for each state-action pair, which represent the expected reward for taking a specific action in

a given state [22]. The ultimate goal is to identify the action with the highest Q-value to maximize future rewards. The Q-function evaluates the expected reward, providing a measure of the quality of each action in terms of its potential to maximize long-term rewards. The algorithm is based on the Bellman equation, and the Q-value is updated using the formula [22]:

$$Q^{new}(S_t, A_t) \leftarrow (1-\alpha).Q(S_t, A_t) + \alpha.(R_{t+1} + \gamma.max_a\, Q(S_{t+1}, a))$$

Where: α is the learning rate which controls the training speed and $(0 < \alpha \leq 1)$, γ is the discount factor which determines the importance of future reward, R_{t+1} is the reward received after moving from S_t to a new state S_{t+1} as a result of taking action A_t, $Q(S_t, A_t)$ is the current Q-value for the state-action pair, $max_a\, Q(S_{t+1}, a)$ is the largest Q-value from all actions (a) in the new state, and $Q^{new}(S_t, A_t)$ is the new Q-value.

The steps involved are as follows:

1. Create a Q-table with the number of rows to match the number of observations and columns to match the number of possible actions. In the context of pathfinding, the highest neighbor value for all nodes is used to avoid out-of-bound actions. Initialize the Q-table entries to 0.
2. Start an episode and use the epsilon-greedy approach for the agent to begin its exploration and exploitation process.
3. The agent selects an action and transitions to the next state.
4. Measure the reward and update the Q-table.
5. The episode terminates when the agent performs an out-of-bound action (highly unlikely), reaches a dead end, encounters a closed road, or successfully reaches the goal.
6. Begin a new episode and reset the environment.
7. Repeat steps 2 to 6 for a specified number of episodes.

Fig. 7. Optimal path returned by Q-learning RL algorithm. The agent explores without any guide and it able to find a path.

5) Epsilon Greedy Policy: The epsilon-greedy policy is a strategy used to balance exploration and exploitation. The agent explores by trying new actions, allowing it to gather information and improve its understanding of the environment. During exploitation, the agent selects the action with the highest Q-value for a given state, relying on previously learned information instead of discovering new possibilities. The parameter is denoted as ε and $(0 < \varepsilon \leq 1)$. At each decision point, a random number is generated. If the number is less than epsilon, the agent explores by trying a new action. If the number is greater, the agent exploits its existing knowledge.

6) Agent Training: The agent is trained for 10,000 episodes with parameters learning rate 0.1, discount factor 0.9, and epsilon 0.1.

7) Agent Evaluation: At this point, the agent does not learn; it exploits its learned knowledge. This process is repeated for 10 episodes, and the results are evaluated.

8) Hyperparameter Tuning: This technique is used to determine the optimal Q-learning parameters. The aim is to find the best combination of values that will yield the best performance, such as either actually finding a path or the shortest path to reach a goal or rerouting in cases of traffic issues. In this case, only the learning rate is adjusted to 0.7, while the other initially selected parameters were found to be optimal.

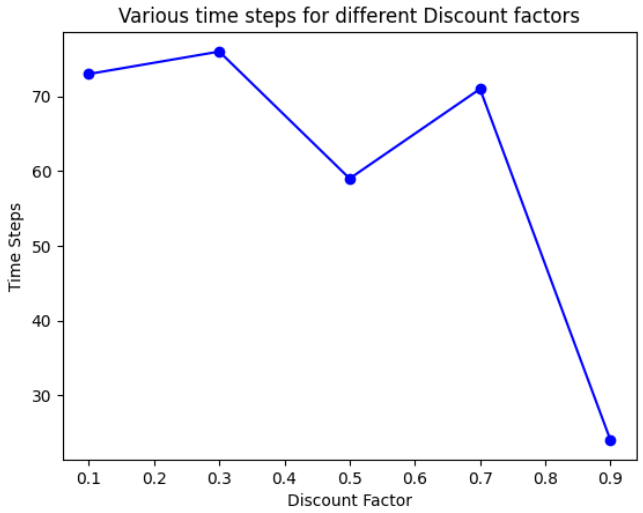

Fig. 8. Hyperparameter results shows discount factor of 0.9 yields the best performance.

E. Traffic Awareness and Dynamic Road Blockades

The agent can adapt in real time to real-world scenarios, such as traffic congestion and road blockages, by dynamically rerouting and continuing toward the goal if alternative paths are available. A mechanism is used to simulate this by randomly selecting nodes to represent traffic or road closure issues.

The agent prioritizes avoiding road closures over traffic congestion because, in reality, closure nodes represent dead ends, meaning no further routes are available beyond them, whereas traffic congestion may eventually clear up. The agent may pass through a traffic node if it is the only path to the goal, but it would never go through a closure node, even

if that is the only path to the goal. In reality, going through a traffic node could indicate that the congestion has cleared.

Fig. 9. Shows the traffic node in yellow and the road closure node in red

From fig. 9 above, The green dot represents the start node. The purple dot represents the end node. The red dot indicates a closed road. The yellow dot indicates traffic congestion.

Fig. 10. New path discovered by the agent showing its ability to bypass road issues in real time and reroute accordingly

Fig. 10 above shows the newly discovered path by the agent. It efficiently reroutes to avoid dynamic road issues. The blue line represents the agent's best-learned route after bypassing closed roads or traffic congestion. The thin red line represents the default route, assuming the road were not closed as seen in fig. 7.

III. Results and Discussion

An analysis of the paths returned by different route planning algorithms reveals variations in both distance and travel time. One notable observation is that Dijkstra's algorithm with distance as weight returns a shorter route, but with a higher travel time. On the other hand, Dijkstra's algorithm with travel time as weight returns a longer route but with a significantly shorter travel time. This difference arises because driving roads have varying speed limits. As a result, the shortest distance may not always be the fastest route when driving. Fig. 2 shows the path generated by Dijkstra's algorithm with distance as weight, which follows residential streets with lower speed limits. In contrast, Fig. 3 shows the path optimized for travel time, which utilizes highways with higher speed limits.

TABLE I. DISTANCE AND TRAVEL TIMES

Optimal Path Algorithm	*Distance (km)*	*Time (m)*
Dijkstra by distance	4.47	5.22
Dijkstra by travel time	4.82	4.00
A* with Geodesic distance heuristic	4.82	4.00
A* with Haversine distance heuristic	4.82	4.00
Reinforcement Learning agent	4.82	4.00
Reinforcement Learning agent with simulation	5.19	5.22

The findings suggest that the true cost of traversing a road is typically measured in time, not distance, because the speed at which a vehicle can travel depends on several factors such as speed limits, road type, traffic congestion, and road closures. In contrast, the distance between two locations remains constant as long as the road exists and both locations remain fixed.

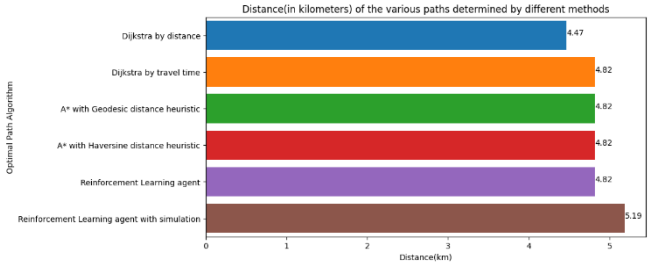

Fig. 11. Shows distance of the routes returned by the different route planning methods

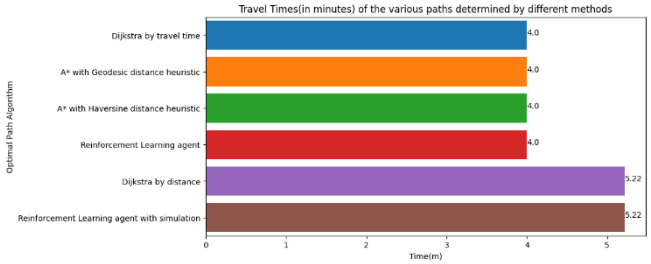

Fig. 12. Shows the travel time of the routes returned by the different route planning methods

Furthermore, RL has proven to be a powerful tool for dynamic pathfinding and route planning tasks. Unlike search algorithms, RL explores the environment without any guidance, heuristics, or weights, yet it is still able to determine the most optimal path. Most importantly, it adapts to dynamic, real-world road issues in real time and reroutes

accordingly, as demonstrated in Fig. 10. This effectively mimics real-life decision-making in real-world scenarios.

IV. Future work and Conclusion

This project yielded exciting results. However, there are a few limitations. While a path may exist, reinforcement learning might fail to identify it within the given number of training episodes. Even when a path is found, it may not be optimal, this highlights a key challenge. Achieving sufficient exploration can sometimes require a significantly higher number of episodes, which may result in computational inefficiencies.

Future work could focus on developing a hybrid approach that first uses search algorithms for initial path planning, which would help reduce the computational complexity of reinforcement learning. Then, RL could be applied to optimize the path and adapt to dynamic road and traffic conditions. Additionally, incorporating a heuristic or guidance mechanism for RL could significantly reduce the computational complexity of the exploration process.

In conclusion, this work aimed to develop an intelligent navigation system for dynamic, optimal pathfinding and route planning. The results demonstrate that reinforcement learning is a powerful tool for enhancing pathfinding in navigation systems, effectively addressing real-world challenges such as traffic congestions, road closures, and other road conditions.

.

References

[1] Veness, C. (2011). Vincenty solutions of geodesics on the ellipsoid in JavaScript | Movable Type Scripts. [online] Movable-type.co.uk. Available at: https://www.movable-type.co.uk/scripts/latlong-vincenty.html.

[2] Veness, C. (2019). Calculate distance and bearing between two Latitude/Longitude points using haversine formula in JavaScript. [online] Movable-type.co.uk. Available at: https://www.movable-type.co.uk/scripts/latlong.html.

[3] Boeing, G. (2024) 'Modeling and Analyzing Urban Networks and Amenities with OSMnx', Available at: https://geoffboeing.com/publications/osmnx-paper/

[4] Chiou, S. (2024) 'A knowledge-assisted reinforcement learning optimization for road network design problems under uncertainty', Knowledge-Based Systems, 292, pp. 111614. Available at: https://doi.org/10.1016/j.knosys.2024.111614

[5] Dayan, P. and Watkins, C. (1992) 'Q-learning', Machine Learning, 8(3), pp. 279–292.

[6] Dijkstra, E.W. (2022) 'A Note on Two Problems in Connexion with Graphs', in 'A Note on Two Problems in Connexion with Graphs', Edsger Wybe Dijkstra: His Life,Work, and Legacy. 1st edn. New York, NY, USA: Association for Computing Machinery, pp. 287–290.

[7] Hart, P.E., Nilsson, N.J. and Raphael, B. (1968) 'A formal basis for the heuristic determination of minimum cost paths', IEEE Transactions on Systems Science and Cybernetics, 4(2), pp. 100–107.

[8] He, K. et al. (2023) 'Accelerated and Refined Lane-Level Route-Planning Method Based on a New Road Network Model for Autonomous Vehicle Navigation', World Electric Vehicle Journal, 14(4) Available at: https://doi.org/10.3390/wevj14040098

[9] Li, D. et al. (2024) 'A reinforcement learning-based routing algorithm for large street networks', International Journal of Geographical Information Science, 38(2), pp. 183–215. Available at: https://doi.org/10.1080/13658816.2023.2279975

[10] M. L. Fredman and R. E. Tarjan (1984) 'Fibonacci Heaps And Their Uses In Improved Network Optimization Algorithms', 25th Annual Symposium onFoundations of Computer Science, 1984. Available at: 10.1109/SFCS.1984.715934.

[11] Nilsson, N.J. (2009) The quest for artificial intelligence. Cambridge University Press.

[12] Puterman, M.L. (1990) 'Markov decision processes', Handbooks in Operations Research and Management Science, 2, pp. 331–434.

[13] R. K. Yadav et al. (2020) 'Comparative Analysis of Route Planning Algorithms on Road Networks', 2020 5th International Conference on Communication and Electronics Systems (ICCES). Available at: 10.1109/ICCES48766.2020.9137965.

[14] Russell, S.J. and Norvig, P. (2016) Artificial intelligence: a modern approach. Pearson.

[15] S. Belhaous et al. (2019) 'Parallel implementation of A* search algorithm for road network', 2019 Third International Conference on Intelligent Computing in Data Sciences (ICDS). Available at: 10.1109/ICDS47004.2019.8942279.

[16] Schrijver, A. (2012) 'On the history of the shortest path problem', Documenta Mathematica, 17(1), pp. 155–167.

[17] Shanmugasundaram, N. et al. (2019) 'Genetic algorithm-based road network design for optimising the vehicle travel distance', International Journal of Vehicle Information and Communication Systems, 4(4), pp. 344–354. Available at: https://doi.org/10.1504/IJVICS.2019.103931

[18] Sniedovich, M. (2006) 'Dijkstra's algorithm revisited: the dynamic programming connexion', Control and Cybernetics, Vol. 35, no 3, pp. 599–620.

[19] Sutton, R.S. and Barto, A.G. (1999) 'Reinforcement learning', Journal of Cognitive Neuroscience, 11(1), pp. 126–134.

[20] Sutton, R.S. and Barto, A.G. (2018) Reinforcement learning: An introduction. MIT press.

[21] Szcześniak, I., Jajszczyk, A. and Woźna-Szcześniak, B. (2019) 'Generic Dijkstra for optical networks', Journal of Optical Communications and Networking, 11(11) Available at: https://doi.org/10.1364/jocn.11.000568

[22] Watkins, C.J. and Dayan, P. (1992) 'Q-learning', Machine Learning, 8, pp. 279–292.