Proceedings of CONF-APMM 2026 Symposium: Applied Mathematics and Quantitative Modeling
DOI: 10.54254/2753-8818/2026.PU35235

From Traditional Search Techniques to Deep Reinforcement Learning: Intelligent Navigation Approaches

Shining Liu

Department of Information and Mathematical Sciences, Faculty of Science, Tokai University, Hadano, Japan
4CSS2109@tokai.ac.jp

Abstract. Navigation Intelligence is one of the problems in Artificial Intelligence, Robotics and Autonomous Platforms. It is a system that needs to go from a start to an end, avoiding obstacles and selecting an optimal path. This paper studies typical navigation techniques and how they operate. The traditional ways explored here are Breadth-First Search, Depth-First Search, Dijkstra's algorithm and A*. The first way is simple to use and stable in a relatively good and simple environment; however, it is not very convenient for many cases. At the same time, the frameworks of reinforcement learning and deep reinforcement learning, such as Q-learning, Deep Q-Networks, Proximal Policy Optimisation and Asynchronous Advantage Actor-Critic, can be employed to learn the movement pattern of an agent through interaction with its environment. The above ways are relatively more flexible in the area that is difficult to reach or only partially known. However, they also have some limitations. That is to say, the training costs are high, the results are inconsistent, and they cannot respond to new situations quickly. According to the results of the survey, in the future, intelligent navigation architecture may be combined with traditional planning technology.

Keywords: Intelligent navigation, path planning, reinforcement learning, deep reinforcement learning

1. Introduction

Intelligent navigation is one of the current hot topics in Artificial Intelligence, Robotics and autonomous driving. Navigation is to move from one place to another around obstructions and choose a reasonable path. The above problem can be shown by a simple maze-solving exercise and has many applications in life. Mobile robots, warehouse automation frameworks, autonomous vehicles, service robots and game artificial intelligence all have navigation functions. With the wide spread of robots in unpredictable environments, navigation ability has advanced from the foundation of path mapping to a combination of perception, planning, learning and adaptation [1].

The two types of improvements in routing technology can generally be divided into the former exploration-driven type and the latter data-driven way. Traditional ways are graph-traversal algorithms, such as BFS and DFS, Dijkstra's algorithm, A*, etc., along with heuristic search. Such algorithms are relatively simple to understand and perform well in the model of graphs or lattice grids [2]. BFS is used to find the shortest path in an unweighted graph, and A* can be used to

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2753-8818/2026.PU35235

accelerate the search. Generally speaking, they are all static environment variables. If the set is unknown, partially visible or changing dynamically, it needs to be reconfigured more often and sensing should be added.

Recently, Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) have been applied to address the problem of navigation [3]. The old way usually needs to know the whole map or graph structure beforehand; RL does not. The agent learns to acquire knowledge by observing the results of its actions in the environment. To address the problem of a large amount of data and learn complex decision-making rules, deep neural networks are employed in DRL [4, 5]. Q-learning, Deep Q-Network (DQN), Proximal Policy Optimisation (PPO) and Asynchronous Advantage Actor-Critic (A3C) are representative algorithms that have been used in the field of intelligent navigation.

Intelligent navigation technology has evolved from traditional search-based approaches to learning-based navigation strategies. This paper aims to provide a review rather than an experimental study. It does not propose new algorithms or conduct novel experiments, but instead summarizes representative navigation methods and explains their fundamental principles, advantages, limitations, and applicable scenarios. Using maze solving as an example, the paper illustrates the working mechanisms of different navigation strategies in a simple yet effective manner. Based on this analysis, it clarifies the relationship between classical path planning methods and modern learning-based navigation techniques.

2. Method

There are two main types of intelligent navigation strategies in this paper. The first type is a traditional search-oriented method, which includes BFS, DFS and A*, and it is based on graph traversal and heuristic scheduling. The second group is reinforcement learning methods, including Q-learning, DQN, PPO, A3C, etc. The above frames can improve the motion characteristics of objects in a particular environment. The discussion moves from fixed-search rules to adaptive and learning-based methods.

2.1. Traditional algorithms

2.1.1. Breadth-first search

The old graph-search method is Breadth-First Search, and it can solve a maze problem and other path-finding problems. BFS starts at the beginning, considers all adjacent nodes in the current level, and then moves on to the next depth. The above is a level-by-level process; thus, BFS can find the shortest path in an unweighted graph. Moore's work on the shortest path in mazes is one of the first applications of BFS-like algorithms to maze problems and is frequently cited [6].

BFS is complete; it can be used on unweighted graphs. If a path exists, BFS will find one; the first goal path it finds will be the shortest in terms of the number of moves. In short, BFS can find the shortest path from a given starting vertex to all other reachable vertices in an unweighted graph. BFS cannot be used to examine the whole region when it is very large. Since all the frontier vertices in the current search tier are stored, the number of stored vertices can be relatively large in a complex environment. BFS is simple in terms of structure and logic, but it cannot solve problems with a very large number of nodes or real-time navigation [7].

DOI: 10.54254/2753-8818/2026.PU35235

2.1.2. Depth-first search

Depth-First Search is the other way to navigate graphs and solve labyrinths. DFS is deep in one path and then backtracks. It will have a small amount of memory. Depth-First Search (DFS) is used to solve the routing problem when only a valid path to the goal is required, and the highest efficiency is not necessary.

DFS cannot guarantee the shortest path. It may follow a suboptimal route for a long time before reaching the goal, or it may need to backtrack and explore alternative paths. DFS is generally not a good choice for exploring the maze efficiently. BFS and DFS are different in that BFS can find the shortest path of an unweighted graph, but DFS cannot. Therefore, DFS can be used in the case of small storage [7]. DFS is suitable for the first stage of autonomous driving research, but it is not ideal and thus cannot be used in a practical manner independently.

2.1.3. The A* algorithm

A* is a general heuristic search method that can be applied to route planning. The algorithm will choose the lowest-cost path according to the present actual path costs and predict the rest of the expenses [8]. The typical form of the heuristic function for A* is $f(n) = g(n) + h(n)$, where $g(n)$ is the cost from the start node to the current node, and $h(n)$ is the estimated cost from the current node to the goal. A* does not suffer from the problem of repeated exploration that the unguided strategies of BFS and DFS have, so it selects more promising vertices first.

A* is relatively fast and can also be optimal in some cases. Provided that the heuristic function is admissible, it will never overestimate the actual cost to the goal; thus, A* can find the shortest path [8]. Dijkstra's algorithm is a classic shortest-path algorithm, and A* can be considered a kind of directed search method for uniform-cost search that uses a heuristic function to direct the search. If the heuristic value is 0, A* will run the same as Dijkstra's algorithm [8, 9]. A* still needs a good heuristic function and some knowledge about the environment. Therefore, A* is suitable for structured environments, but it cannot learn or change according to the circumstances of a moving object.

2.2. Reinforcement learning algorithms

2.2.1. Q-learning

Q-learning is the older off-policy reinforcement learning method. Thus, an entity will be able to perform all kinds of operations in different circumstances by learning and applying through practice in interaction with its environment [10]. The state of the world in a navigation scenario is the agent's current location, and a decision is generally one of the four directions: up, down, left or right. After running the operation, it will be either a bonus or a fine. If the destination is reached, the recorded path is considered valid; otherwise, the algorithm must continue searching or backtrack to explore alternative routes.

Q-learning is a model-free computation framework that does not need to know the whole environment model. The learner learns a good way of navigating through experience. Q-tables are the basic component of the old Q-learning algorithm, and they perform poorly in large state and action spaces. The state space of the actual navigation problem is generally very large and may contain continuous variables. Tabular Q-learning is the basic theoretical foundation, but it cannot be applied to the problem of high-dimensional spaces in practice.

DOI: 10.54254/2753-8818/2026.PU35235

2.2.2. Deep Q-network

Deep Q-networks are a kind of neural network that can approximate the action-value function, so they fall under the category of Q-learning [4]. DQN does not store all state-action pairs in a table. It is a function that takes the high-dimensional input and returns an action reward, and nothing else. Thus, the integration of visual information and sensor data in reinforcement learning can be handled more effectively.

Deep Q-Networks (DQN) have been used to solve the problem of navigation for exploration and avoidance in autonomous driving. Tai and Liu have put forward a deep reinforcement learning framework, and the depth data that can be obtained from an RGB-D camera is used as the input for robot navigation [11]. DQN is good at learning the logic of path planning through experience and does not need to manually set every maneuvering rule. DQN usually needs a large amount of training data, is very sensitive to the choice of hyperparameters, and has unstable learning. The strategies that have been obtained from simulation may not be feasible in reality. DQN is relatively general-purpose, but it needs more resources and is also not very easy to optimise.

2.2.3. Proximal policy optimization

PPO is a kind of policy gradient method in deep reinforcement learning and is proximal. DQN determines the value of a certain action, but PPO is used to directly update the policy and choose which step the agent takes. The first idea of PPO is to limit how much the new policy deviates from the old policy at every step [12]. Therefore, it is more stable than many of the earlier policy-gradient methods.

PPO is suitable for the navigation problem because it can handle the continuous or multi-dimensional action space of different types of problems and solve the decision-making problem. Policy gradient and actor-critic methods have been used in the field of robotic movement scheduling to improve the navigation behaviour of agents by interacting with the environment, so they can learn [2]. PPO is still relatively difficult to train and needs a good reward function. Therefore, PPO has shown good stability compared with some older methods, but the problems of sample inefficiency and reward design have not been fully solved.

2.2.4. Asynchronous Advantage Actor-critic

Asynchronous Advantage Actor-Critic (A3C) is a kind of deep reinforcement learning based on the actor-critic method. Some concurrent entities are used to communicate with all kinds of instances in the environment and update a shared model independently [13]. This architecture can help the network obtain more effective training by providing it with different views from all the participants.

A3C can be used to solve the problem of visual navigation as well. Zhu and others have applied the framework of goal-directed deep reinforcement learning to indoor visual exploration to train an agent that can reach a goal with the help of visual information [14]. Practical Navigation cannot always use complete maps and is therefore perception-based Navigation. A3C is more complicated than the previous ones. Concurrent training has a relatively high computational cost, and the obtained policy may still fail on the test set that is different from the training set. Therefore, A3C has shown that deep reinforcement learning can be applied to smart navigation, but it also has some problems in actual use.

DOI: 10.54254/2753-8818/2026.PU35235

3. Discussion

3.1. Challenges

Based on the above evaluation, autonomous guidance has gradually been replaced by prediction-based search methods and has started to use flexible learning-centric strategies. BFS, DFS, Dijkstra's method and A* are relatively simple to understand and implement, so they are still popular. The above algorithms are correct in the identified and stationary regions. BFS is for finding the shortest path on unweighted graphs, DFS can be used for simple traversal, and A* works well when a suitable heuristic function is available, which is better than BFS and DFS. However, a problem with the above methods is that they usually require organised spatial information. In the absence of a definite plan or based on some observed information, the original plan will often need to be amended, or it will be revised and updated.

Reinforcement Learning and Deep Reinforcement Learning are generally better for the navigation problem of a complicated environment. The agent will not follow a pre-set search plan; rather, it will learn to explore independently. It will help to address the problem of a lack of environmental models. DRL can also be employed to solve the problem of high-dimensional input data for images and depth maps in perception and action selection [5]. However, there are many problems with such generalisation in practice. The above reasons show that although DRL technology has been making progress, it is not yet suitable for all kinds of navigation problems in the real world.

The first problem is a small sample. Many DRL methods need to run a long time to obtain a good policy. It is acceptable in simulations; many training iterations can be performed effortlessly, but it is not practical for real robots due to the slow speed, high cost and occasional dangerous circumstances of practical learning. Research in the field of visual navigation has also concluded that, in order to help an agent operate normally in the context of perception, a large amount of training data needs to be collected [14].

The next issue is that the training is unstable and not reproducible. Settings of hyperparameters, random seeds, reward functions and many other specific contents of DRL vary widely. Henderson and others have shown that the results of experiments in deep reinforcement learning are often quite different, so they need to be reported rigorously to confirm whether the improvements are real [15]. The mobility study has not yet been solved, and what works well in one simulation may not be suitable for others.

The third is overextension. The plan for a certain maze, room or simulation environment is not suitable for the new situation or actual life. Therefore, a generalisation problem in DRL has arisen due to the difference between the learning environment and the operating environment [16]. The other problem is that the simulation is not the same as reality. If an actor learns through a small number of training scenarios, they will be unable to navigate under various circumstances. Therefore, the assessment should also consider how the agent will behave without prior preparation, face all kinds of barrier distributions and sensor malfunctions, etc.

3.2. Future prospects

There are some restrictions, but many other ways are available. First of all, the combination of navigation will be based on the reliability of the old algorithm and the flexibility of data-driven technology. A* is a general-purpose search algorithm that may be too slow to find the path, so DRL can be used to avoid close collisions or make choices in the face of uncertainty. Social and multi-

DOI: 10.54254/2753-8818/2026.PU35235

agent movement will also need to be achieved more broadly in communal Areas for autonomous units. Therefore, the machine needs to go around fixed obstacles in the scene and at the same time understand human behaviour and group norms. Next, semantic-aware navigation will be incorporated into the framework, enabling higher-level reasoning about objects, locations, instructions, and goals. Thus, guidance architectures will be more convenient for people to use in everyday life and will often be stated in words rather than precise coordinates.

In short, no single navigation way can address all the problems of navigation. The traditional method is relatively simple to understand and reliable, but Reinforcement Learning and Deep Reinforcement Learning have been developed to address the problems of complex environments. In the future, it is hoped that the whole field of autonomous driving will be organised in an integrated way through learning-based adaptation and strong abstraction.

4. Conclusion

Recently, research on navigation strategies has included the old way of search and Deep Reinforcement Learning. As shown in the background, Pathfinding is a typical problem for robots, autonomous driving systems, logistics bases and so on, so it should be learned. BFS, DFS, Dijkstra's algorithm and A* are relatively simple in terms of operation and can be used to find paths. BFS is to find the shortest path of an unweighted graph; DFS is for exploring the graph with little memory; A* adds a heuristic function to speed up the search. The algorithm will still work in an ordered and regular environment.

However, in reality, navigation missions are often not static grids or graphs. The place is unknown, changing, or only partially shown. Therefore, Reinforcement Learning and Deep Reinforcement Learning have been applied to solve this problem. Q-learning, DQN, PPO and A3C are used to learn how to move more often by interacting with the environment. The above ways are more general-purpose in handling all kinds of input and change at all levels, but they also need a large amount of training data and careful design of incentives and variables.

In general, the old way of searching is stable and easy to understand, and the new way of learning has greater flexibility but is more difficult to train and evaluate. Therefore, the new navigation frameworks will not be based on only one method. A practical way is to integrate the old planning with learning-based adaptation, and thus the intelligent entity will be able to move more safely and conveniently in the real world.

References

[1] Xiao, X., Liu, B., Warnell, G. and Stone, P. (2022) Motion planning and control for mobile robot navigation using machine learning: A survey. Autonomous Robots, 46, 569-597.
[2] Zhou, C., Huang, B. and Fränti, P. (2022) A review of motion planning algorithms for intelligent robots. Journal of Intelligent Manufacturing, 33, 387-424.
[3] Arce, D., Solano, J. and Beltrán Castañón, C.A. (2023) A comparison study between traditional and deep-reinforcement-learning-based algorithms for indoor autonomous navigation in dynamic scenarios. Sensors, 23(24), 9672.
[4] Mnih, V. et al. (2015) Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.
[5] Arulkumaran, K., Deisenroth, M.P., Brundage, M. and Bharath, A.A. (2017) Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6), 26-38.
[6] Moore, E.F. (1959) The shortest path through a maze. Proceedings of the International Symposium on the Theory of Switching, Part II, Harvard University Press, Cambridge, MA, 285-292.
[7] Everitt, T. and Hutter, M. (2015) A topological approach to meta-heuristics: Analytical results on the BFS vs. DFS algorithm selection problem. arXiv preprint arXiv: 1509.02709.

DOI: 10.54254/2753-8818/2026.PU35235

[8] 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), 100-107.
[9] Dijkstra, E.W. (1959) A note on two problems in connexion with graphs. Numerische Mathematik, 1, 269-271.
[10] Watkins, C.J.C.H. and Dayan, P. (1992) Q-learning. Machine Learning, 8, 279-292.
[11] Tai, L. and Liu, M. (2016) Towards cognitive exploration through deep reinforcement learning for mobile robots. arXiv preprint arXiv: 1610.01733.
[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017) Proximal policy optimization algorithms. arXiv preprint arXiv: 1707.06347.
[13] Mnih, V. et al. (2016) Asynchronous methods for deep reinforcement learning. Proceedings of the 33rd International Conference on Machine Learning, 48, 1928-1937.
[14] Zhu, Y. et al. (2017) Target-driven visual navigation in indoor scenes using deep reinforcement learning. Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 3357-3364.
[15] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D. and Meger, D. (2018) Deep reinforcement learning that matters. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1), 3207-3214.
[16] Kirk, R., Zhang, A., Grefenstette, E. and Rocktäschel, T. (2023) A survey of zero-shot generalisation in deep reinforcement learning. Journal of Artificial Intelligence Research, 76, 201-264.