Proceedings of the 3rd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/150/2025.22529

Exploring the Comparison Between Dijkstra's Algorithm and Rapidly-exploring Random Trees Algorithm for Robot Path Planning

Yuxiao Liu

Faculties of Engineering, University of Bristol, Bristol, UK
IG22046@BRISTOL.AC.UK

Abstract: Since the inception and evolution of artificial intelligence, control systems, particularly automated pathfinding, have constituted a pivotal area of significance. It is reflected in today's world of life in all aspects of autonomous driving, rescue robots and so on. Algorithms as the basis for the realization of this field exist in thousands of situations, which are reflected in different logic, different computational efficiency, different time and space complexity, etc. We will introduce the development history and basic principles of Dijkstra’s algorithm and Rapidly-exploring Random Trees algorithm, and analyze the advantages and disadvantages of each of these two algorithms to determine the domains in which they are applicable. In this paper, we will use MATLAB to set up a test environment and will investigate the effect of different environments comparing Dijkstra’s algorithm and RRT algorithm on the automatic control system in artificial intelligence. In addition, some experimental data and icons will be cited to support the experimental results by comparing the algorithms in terms of distance and efficiency. It is concluded that Dijkstra’s algorithm will be more suitable for static and low dimensional environments, while Rapidly-exploring Random Trees algorithm will be suitable for more complex and high dimensional environments.

Keywords: Pathfinding Algorithms, Dijkstra’s Algorithm, Rapidly-exploring Random Trees, MATLAB Simulation

1. Introduction

With the advancement of technology, Artificial Intelligence (AI) is playing an increasingly important role in today's world and has become the foundation of some industries such as automated driving systems. According to Atakishiyev, Salameh and Yao [1], with the rapid progress in computationally powerful artificial intelligence (AI) techniques, AVs can sense their environment with high precision, make safe real-time decisions, and operate reliably without human intervention [1]. In the case of autonomous driving, for example, today's AI technology cannot be separated from the help of path-planning algorithms, and as Lin [2] said, automatic driving has gradually become a hot topic of people's attention. Among them, path planning algorithms play a central role in the field of automatic driving [2]. Among many path planning algorithms, Dijkstra's algorithm and Rapidly-exploring Random Trees algorithm are two representative algorithms. Dijkstra’s algorithm was conceived by computer scientist Edsger W. Dijkstra in 1956. It can be used to find the shortest path to a specific goal node by terminating the algorithm after determining the shortest path to the goal node. RRT algorithm was developed by Steven M. LaValle and James J. Kuffner Jr. They can easily handle problems with obstacles and differential constraints (incomplete and dynamic) and have been widely used for autonomous robot motion planning. This study conducts a comparative analysis of the Dijkstra and RRT algorithms by applying them to a single robot operating in both simple and complex environments, thereby assessing their suitability for different types of environments. This will be useful for future industries using AI path planning systems as the algorithms can be selected according to different environments, thus completing the task more efficiently and reducing errors and redundancy.

2. Introduction to dijkstra and RRT algorithm

2.1. Dijkstra algorithm and application

Dijkstra’s algorithm, introduced by the Dutch computer scientist Edsger W. Dijkstra in 1956, represents a foundational and extensively employed method in computer science for determining the shortest path between two nodes within a graph. It is particularly effective for graphs with non-negative edge weights and has thus become a cornerstone in areas such as network routing, transport systems and robotics. The algorithm iteratively selects the node with the smallest tentative distance from the starting node, updates the distance to its neighboring nodes, and marks it as ‘visited’. This process continues until the shortest path to the goal node has been determined or until all reachable nodes have been explored. Dijkstra’s algorithm employs a greedy approach and always selects the locally optimal step at each iteration, thus guaranteeing the shortest path. Although Dijkstra's algorithm is efficient in many applications, it has limitations such as not being able to handle graphs with negative edge weights. And in some cases the computational efficiency is too high, as in Fan and Shi, Dijkstra's algorithm is the most classical and mature algorithm for searching a shortest path in the graph, however this algorithm has the highly time complexity and takes up a larger storage space [3]. Nevertheless, Dijkstra's algorithm is still an important tool in the field of computer science and is highly regarded for its simplicity, reliability and effectiveness in solving the shortest path problem.

Dijkstra's algorithm plays an important role in Artificial Intelligence (AI), especially in pathfinding and optimisation tasks. One of its main applications is robot navigation, where a robot or autonomous agent must find the shortest path from a starting point to a goal while avoiding obstacles. By modelling the environment as a graph, Dijkstra's algorithm can efficiently compute optimal paths, enabling robots to move efficiently in complex spaces such as warehouses, factories or even outdoor terrain. In game AI, Dijkstra's algorithm is commonly used to determine the shortest path for a non-player character (NPC) to traverse the game world. While more advanced algorithms such as A* have been widely adopted for their heuristic-based efficiency, Dijkstra's algorithm remains a fundamental approach, especially when heuristic information is unavailable or unreliable. Furthermore, Dijkstra’s algorithm is employed in AI-driven logistics and transportation systems, including courier services and route planning for public transit networks. By calculating the shortest paths between locations, the algorithm helps to optimise resource allocation and reduce operational costs. For illustration, Behún, M., Knežo, and D., Cehlár demonstrate the development of a model for determining the optimal path based on data derived from our algorithm, which was utilized to identify the most financially advantageous connection between customer and supplier locations [4].

2.2. RRT alogrithm and application

The Rapid Exploration Random Tree (RRT) algorithm is a probabilistic path planning method widely used in robotics and artificial intelligence to solve complex motion planning problems. Developed by

Steven M. LaValle in the late 1990s, RRT is particularly effective in high-dimensional spaces and environments with obstacles where traditional path planning algorithms may struggle. RRT works by incrementally building a tree-like structure of feasible paths from a starting point to a goal. It randomly samples points in the configuration space and connects them to the nearest node in the tree structure, ensuring that paths do not collide. This random sampling allows RRT to explore the environment efficiently, even in cluttered or dynamic environments. Over time, the tree expands into unexplored regions, gradually finding paths to the goal. One of the main strengths of RRT is its ability to deal with non-holonomic constraints and high-dimensional problems, such as those encountered in robotic arm manoeuvring or automated vehicle navigation. Variants such as RRT* and RRT-Connect further enhance its capabilities by optimising the smoothing and convergence of the paths. Nevertheless, the RRT algorithm possesses certain limitations. As noted by Tian, Yan, and Park, the termination condition of tree formation is contingent upon the success in locating the goal [5]. This condition may result in the tree containing many nodes in the order of hundred. Due to these drawbacks, there are some strategies that can help the Due to these drawbacks, there are some strategies that can help the original RRT algorithm to reduce the number of nodes. The number of nodes is an important factor in computation to find an optimal path using the tree.

In the field of robotics, RRT is widely used for autonomous navigation and manipulation tasks. For example, mobile robots and drones use RRT to plan collision-free paths in dynamic or cluttered environments such as warehouses, urban areas or disaster zones. Similarly, robotic arms can use RRT to find feasible trajectories to perform tasks such as pick-and-place operations to avoid obstacles while moving in tight spaces. The probabilistic nature of the algorithm allows it to adapt to real-time changes in the environment, making it suitable for applications that require fast decision-making.

In self-driving cars, RRT plays a crucial role in path planning and obstacle avoidance. It helps self-driving cars navigate complex road networks, car parks or off-road terrain by generating safe and efficient routes. In addition, RRT can be used to simulate environments to test and validate self-driving systems before deployment, as is shown in Figure 1:

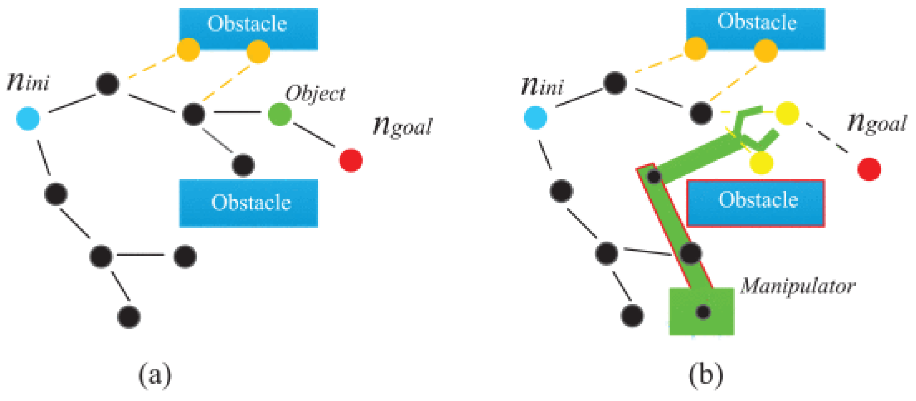

Figure 1: Applications in complex environments: path planning for robot manipulators [6]

3. Comparative analysis

3.1. Environment

This study will use MatLab to construct two 2D test environments, simple and complex, with the same size and starting and ending points. The simple environment is an environment without any obstacles and the robot only needs to move from the start point to the end point, while the complex environment generates a number of points as obstacles to block the robot's planned path.

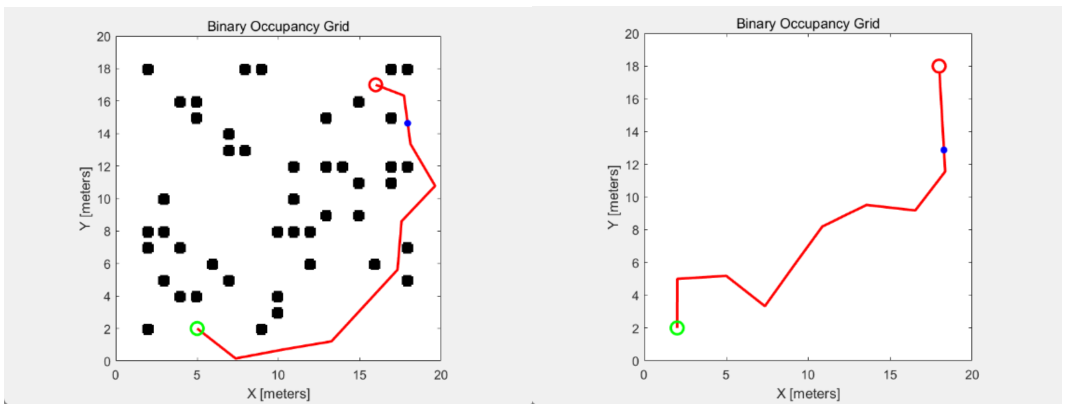

Figure 2: Examples of complex and simple environments (personal matlab test)

In order to reach a conclusion, this research will introduce a few criteria for judgement:

* Runtime: the time it takes for the computer to run the algorithm, which allows us to consider the cost of time in order to judge which algorithm can plan a path in less time.

*Running distance: Calculate the actual distance travelled by the robot from the starting point to the end point, which is also the key criterion of this test, it can intuitively see the difference between the two algorithms in terms of running efficiency.

3.2. Simple environment

In the simple environment, the map is set up as a 20x20 two-dimensional map with the start point set to (2,2) and the end point set to (18,18). Here are the results of the nine sets of tests.

Table 1: Run distance in simple environment

Distance	1	2	3	4	5	6	7	8	9
RRT	31.05	28.77	29.99	28.89	25.64	26.54	28.88	31.23	26.44
Dijkstra	22.63	22.63	22.63	22.63	22.63	22.63	22.63	22.63	22.63

Table 2: Runtime in simple environment

Runtime	1	2	3	4	5	6	7	8	9
RRT	0.0215	0.0151	0.0078	0.0208	0.0047	0.0023	0.0061	0.0036	0.0021
Dijkstra	0.0123	0.0075	0.0186	0.0057	0.0055	0.0055	0.0055	0.0056	0.0053

According to Table 1 and 2, we can know that at the runtime level, excluding the possible outliers of test1 and 3, the runtime of Dijkstra is around 0.0055 seconds, which reflects the stability of having only one fixed route. Conversely, the RRT algorithm must generate distinct paths for each iteration, resulting in a running time that can vary significantly—ranging from very large to very small—thus embodying its inherently random nature. Regarding distance, Dijkstra's algorithm consistently identifies the shortest path, yielding a fixed distance value. In contrast, the RRT algorithm's inherent randomness causes the distance to fluctuate, and it typically exceeds that of the path length determined by Dijkstra's algorithm.

3.3. Complex environment

Table 3: Runtime in complex environment

Runtime	1	2	3	4	5	6	7	8	9
RRT	0.0089	0.0055	0.0064	0.0045	0.0032	0.0088	0.0067	0.0084	0.0092
Dijkstra	0.0212	0.0209	0.0224	0.0241	0.0281	0.0215	0.0266	0.0234	0.0204

Table 4: Run distance in complex environment

Distance	1	2	3	4	5	6	7	8	9
RRT	31.23	27.77	26.94	28.43	26.42	28.41	28.22	33.42	28.32
Dijkstra	25.43	26.41	23.22	26.54	23.32	24.32	24.57	25.42	25.11

As illustrated in Table 3, at the runtime level, Dijkstra’s algorithm generally exhibits a higher running time compared to the RRT algorithm. This discrepancy may be attributed to the increased time required for calculating and planning paths in complex environments. It is noteworthy that, despite the numerical difference in running times between the two algorithms being relatively small, this is influenced by the map size. Specifically, the running time of Dijkstra’s algorithm is substantially longer than that of the RRT algorithm, particularly when the map size is not excessively large. It is important to note that although numerically the RRT algorithm is only slightly smaller than Dijkstra's algorithm, this is due to the size of the map, which means that at the practical level, the actual running time of the RRT algorithm may be much smaller. In terms of distance, as is shown in Table 4 that Dijkstra's algorithm still represents the smallest path in the environment, so RRT will still take more distance, but because of the effect of time, it can still be considered more suitable for complex environments.

The running time of an algorithm is influenced by various factors, including computer hardware and environmental conditions. In this study, the algorithm’s running time was measured using a single computer, thereby acknowledging the potential for differing performance outcomes when executed on alternative hardware configurations. In addition, the test environment chosen for this test is relatively single, only 2d 20x20 size, and is not enough to cover the three-dimensional world and larger and more complex environments.

4. Conclusion

Both Dijkstra's algorithm and the RRT algorithm are fundamental tools for path planning, but they excel in different scenarios due to their respective uniqueness. The Dijkstra algorithm is very effective in simple environments with low-dimensional spaces and static obstacles. It guarantees the shortest path by systematically exploring all possible paths, making it ideal for applications such as network routing or grid-based navigation. However, in high-dimensional or complex environments, the computational cost of the algorithm increases substantially, limiting its usefulness in real-time systems. Conversely, RRT demonstrates superior performance in complex, high-dimensional, and dynamic environments. Its probabilistic nature enables efficient exploration of expansive spaces and adaptability to evolving conditions, thereby rendering it the preferred methodology for motion planning in robotics, autonomous vehicles, and cluttered or unstructured settings. Although RRT does not guarantee the shortest path, its ability to quickly find feasible solutions makes it invaluable in real-time applications. However, in low-dimensional and simple environments, it will not be able to complete the task with the same efficiency as Dijkstra's algorithm. In summary, Dijkstra's algorithm is best suited for simple, static environments where optimality is critical, while RRT shines in complex, dynamic scenarios where adaptability and speed are priorities. The choice between the two depends on the specific requirements of the application, which highlights the importance of choosing the right tool for the problem at hand.

References

[1] *Atakishiyev, S., Salameh, M., Yao, H., & Goebel, R. (2024). Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions. IEEE Access.*

[2] *Lin, Y. (2024). Dijkstra and A* algorithms in automated vehicle driving. Proceedings of SPIE 13071, International Conference on Mechatronic Engineering and Artificial Intelligence (MEAI 2023), 130712F. https://doi.org/10.1117/12.3025670.*

[3] *Fan, D., & Shi, P. (2010). Improvement of Dijkstra's algorithm and its application in route planning. Seventh International Conference on Fuzzy Systems and Knowledge Discovery, Yantai, China, 1901-1904. https://doi.org/10.1109/FSKD.2010.5569452.*

[4] *Behún, M., Knežo, D., Cehlár, M., Knapčíková, L., & Behúnová, A. (2022). Recent application of Dijkstra's algorithm in the process of production planning. Applied Sciences, 12(14), 7088. https://doi.org/10.3390/app12147088.*

[5] *Tian, Y., et al. (2007). Application of RRT-based local path planning algorithm in unknown environment. 2007 International Symposium on Computational Intelligence in Robotics and Automation, Jacksonville, FL, USA, 456-460. https://doi.org/10.1109/CIRA.2007.382896.*

[6] *Zhang, H., Wang, Y., Zheng, J., & Yu, J. (2018). Path planning of industrial robot based on improved RRT algorithm in complex environments. IEEE Access, 6, 53296-53306. https://doi.org/10.1109/ACCESS.2018.2871222.*