Proceedings of the 3rd International Conference on Software Engineering and Machine Learning
DOI: 10.54254/2755-2721/150/2025.22521

An Improved Jump Point Search Pathfinding Algorithm for Hexagonal Grid Maps

Xuxuan Peng

International School, Beijing University of Posts and Telecommunications, Beijing, China
2023213448@bupt.cn

Abstract: Pathfinding in hexagonal grid maps is a common problem in some specific areas, especially in video games (like the highly acclaimed Civilization series). The main challenge of these problems lies in how to handle the paths that seemingly different, but in fact equivalent. Jump Point Search (JPS) is regarded as the best algorithm in the current research based on square grid maps. It effectively solved this problem. The efficiency of JPS far exceeds other similar algorithms, including the widely used A-Star algorithm. This paper presents an adaptation of the JPS algorithm for hexagonal grid maps and presents a comparative analysis of its efficiency against the widely-used A-Star algorithm. The result proved that the adjusted JPS algorithm has significantly improved the efficiency of solving pathfinding problem compared to the A-Star algorithm when dealing with medium to long paths and complex paths.

Keywords: Pathfinding, Jump Point Search, Hexagonal Grid Map, Algorithm

1. Introduction

The pathfinding problem is a prevalent challenge across various domains. To address it, researchers have developed a range of algorithms, starting from the fundamental Dijkstra algorithm, depth-first search, and breadth-first search, to the extensively employed A-Star algorithm, and the Jump Point Search (JPS) algorithm [1-2], which is renowned for its speed. These algorithms, designed for square grid environments, demonstrate robust performance. Moreover, the A-Star and JPS algorithms have spawned a suite of enhanced variants, such as JPS+ [2] and HPA-Star [3], among others. Researchers have also expanded the applicability of the JPS algorithm, including its adaptation to three-dimensional spaces as the JPS-3D algorithm [4] and its use as an advanced solution for pipeline routing [5]. The A-Star algorithm, notable for its balance between straightforward implementation and computational efficiency, has found widespread use, particularly in applications like video games [6]. Nevertheless, it is regrettable that the faster JPS algorithms have not gained comparable popularity.

This paper introduces an adjusted JPS algorithm that can run in a hexagonal grid environment. This paper presents an adjusted JPS algorithm tailored for hexagonal grids, along with a comparative analysis based on running time between the adjusted JPS algorithm and the A-Star algorithm, as evidenced by the results of several simple experiments. The research provides a faster method than the A-Star algorithm for pathfinding problems in hexagonal grid maps, which is beneficial for fields that need to solve pathfinding problems in hexagonal grids maps.

DOI: 10.54254/2755-2721/150/2025.22521

2. Notation and Terminology

This article presents an algorithm designed for hexagonal grid maps. In such maps, each node can have up to six adjacent nodes, and nodes are categorized as either passable or impassable (i.e., obstacles). The distance between any two adjacent passable nodes is defined as 1, and any attempt to move from a passable node to an impassable node is deemed invalid.

For hexagonal grid maps, we will employ the form to denote the coordinates of each node, as illustrated in Figure 1:

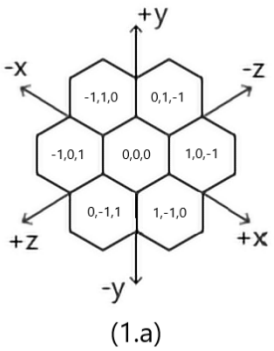

Figure 1: Coordinate System

Most of the following terminologies are referenced from Harabor and Grastien [1], and are mentioned here for ease of reading:

The notation $\vec{d}$ refers to one of the six allowable movement directions.

$\pi = \langle a_0, \dots, a_n \rangle$ represents a path from node a_0 to node a_n, $\pi = \langle a_0, \dots, a_n \rangle \backslash x$ represents paths from a_0 to a_n without passing through node x.

$\text{len}(\pi)$ represents the length of path π.

$p(x)$ represents the previous node (i.e. parent node) of node x in a path.

For a direction $\vec{d}$, $\overrightarrow{d_L}$ and $\overrightarrow{d_R}$ represent the directions of left and right rotation $\pi/3$ along $\vec{d}$, respectively.

$\vec{d} = \langle x, y \rangle$ indicates that $\vec{d}$ is the direction from node x to node y for a pair of nodes $\langle x, y \rangle$ on a straight line.

$y = x + k\vec{d}$ represents that node y can be reached by taking k unit moves from node x in direction $\vec{d}$.

3. Jump Point Search in Hexagonal Grid Maps

Jump Point Search (JPS) is a pathfinding algorithm for square grid maps. It combines the best-first expansion strategy used by popular A-Star search with a neighbour pruning strategy.

However, for hexagonal grids, the rules of the original JPS algorithm require adjustments. It is evident that diagonal movements do not exist in hexagonal grid maps. Instead, it exists turning movement - meaning that the movement is not straight (e.g. $y = x + \overrightarrow{d_1}$ is a *turning movement* if $x = p(x) + \overrightarrow{d_2}$ and $\overrightarrow{d_1}$ is inequal to $\overrightarrow{d_2}$). Meanwhile, a *good turning movement* should be restricted to $\frac{\pi}{3}$ degrees, as a path with valid larger angle turns imply that the path is not the shortest. Also, in hexagonal grid maps, there are only 6 directions, and the cost of moving between adjacent passable nodes will always be 1.

Therefore, we need to make some adjustments to the original definition of JPS to adapt to the characteristics of hexagonal grid maps.

DOI: 10.54254/2755-2721/150/2025.22521

Definitions 1-3 are referenced from Harabor and Grasstien [1], and there are slight differences in these definitions:

Definition 1. *Node* $n \in neighbours(x)$ *is natural if:*

$\vec{d}\langle x, n\rangle = \vec{d}\langle p(x), x\rangle$

In hexagonal grid maps, each node has only 1 natural neighbour.

Definition 2. *Node* $n \in neighbours(x)$ *is forced if:*

1. node n is not the only natural neighbour of x.

2. $len(\langle p(x), x, n\rangle) < len(\langle p(x), \dots, n\rangle \backslash x)$

Definition 3. Node y is a jump point from node x, heading in direction $\vec{d} = \langle x, y\rangle$ *if y both minimizes the value k such that* $y = x + k\vec{d}$ *and lies* k *steps from x in direction* $\vec{d}$ *and one of the following conditions holds:*

1. node y is the goal node.

2. node y has at least one forced neighbour.

3. $\vec{d}$ *is a turning move and there exists a node* $z = y + k_i \overrightarrow{d_i}$ *which lies* $k_i \in N$ *steps in direction* $\overrightarrow{d_i} \in \{\overrightarrow{d_L}, \overrightarrow{d_R}\}$ *such that* z *is a jump point from* y *by condition 1 or condition 2.*

The following are Pruning Rules and Jumping Rules for hexagonal grid maps. Pruning Rules have slight differences, while Jumping Rules are consistent with Harabor and Grasstien [2], for ease of reading only:

Pruning Rules: Given a node x, reached via a parent node p, we prune from the neighbours of x any node n for which one of the following rules applies:

1. there exists a path $\pi' = \langle p, y, n\rangle$, or simply $\pi' = \langle p, n\rangle$ that is strictly cheaper than the path $\pi = \langle p, x, n\rangle$;

2. there exists a path $\pi' = \langle p, y, n\rangle$ with the same cost as $\pi = \langle p, x, n\rangle$, but π' has an earlier turning move than π.

Jumping Rules: JPS applies to each forced and natural neighbour of the current node x a simple "jumping" procedure; the objective is to replace each neighbour n with an alternative successor n_0 that is further away.

Here is a simple example to demonstrate these concepts:

Referring to the scenario depicted in Figure 2.a, in the absence of obstacles, when addressing node 2 with node 1 as its parent, the neighboring nodes highlighted in gray in Figure 2.a can be entirely disregarded. This is because the paths to these nodes are either not the shortest or are symmetrically equivalent in length to paths that can be traversed by turning and moving from the preceding node, without necessitating passage through node 2. Therefore, in the absence of any obstacles, for the situation shown in Figure 2.a, we only need to consider one path: $\pi = \langle 1,2,3\rangle$.

However, if obstacles are also taken into account, the situation will become different. Considering the situation shown in Figure 2.b, where black nodes represent obstacles, we must consider two paths: $\pi_1 = \langle 1,2,3\rangle$ and $\pi_2 = \langle 1,2,4\rangle$. Due to the existence of obstacles, $\pi_2 = \langle 1,2,4\rangle$ becomes the only shortest path from node 1 to node 4. Node 2 becomes a jump point and node 4 is its forced neighbor. Similarly, for the situation shown in Figure 2.c, we need to consider three paths: $\pi_1 = \langle 1,2,3\rangle$, $\pi_2 = \langle 1,2,4\rangle$, and $\pi_3 = \langle 1,2,5\rangle$. For 2.c, node 2 has two forced neighbors: node 4 and node 5.

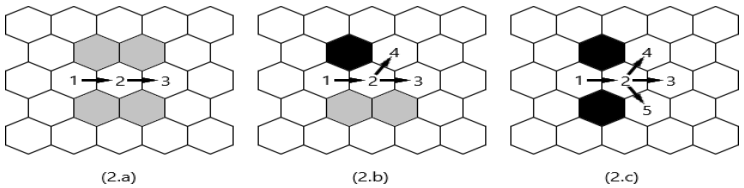

Figure 2: Simple Examples

DOI: 10.54254/2755-2721/150/2025.22521

For each search, assuming the algorithm is processing node x with direction $\vec{d}$, according to the definition of jump points, we need to find potential jump points by the following methods to ensure that we do not miss any jump points:

1. Keep searching along direction $\vec{d}$ until the algorithm find a jump point or the next node reached along direction $\vec{d}$ is an obstacle or outside of the grid. The algorithm will return at this time.

2. After moving $n \in [0, +\infty]$ nodes along direction $\vec{d}$ and reach a node x_n, search in both directions $\overrightarrow{d_L}$ and $\overrightarrow{d_R}$ until a jump point is found, or the next node reached along direction $\overrightarrow{d_L}$ or $\overrightarrow{d_R}$ is an obstacle or outside of the grid. For this situation, if the algorithm find a jump point, it not only need to record the found jump point, but also need to record the node x_n, and node x_n has a predecessor successor relationship with the found jump point. (Definition 3.3)

Based on these steps, we can create a function to find the jump points with given a direction.

Algorithm 1 Function find_jump_point

Require: x: initial node, $\vec{d}$: direction

current $\leftarrow$ x
jump_points(x) $\leftarrow \emptyset$
repeat
  current $\leftarrow$ step(current, $\vec{d}$)
  if current is a jump point then
    jump_points(x) $\leftarrow$ current
    Return jump_points(x)
  for $\overrightarrow{d_n} \in \{\overrightarrow{d_L}, \overrightarrow{d_R}\}$ **do**
    current_RL $\leftarrow$ current
    repeat
      current_RL $\leftarrow$ step(current_RL, $\overrightarrow{d_n}$)
      if current_RL is a jump point then
        jump_points(x) $\leftarrow$ current, current_RL
        break
    until current_RL is an obstacle or current_RL is outside of the grid
  until current is an obstacle or current is outside of the grid
return jump_points(x)

This function is able to find jump points by given point and given direction.

In addition, the algorithm needs to process the obtained jump points in a certain order. This part is similar to A-Star Algorithm and Original JPS Algorithm.

In the hexagonal grid maps, consider two nodes $a = (x_1, y_1, z_1)$ and $b = (x_2, y_2, z_2)$ in it, the distance $H\langle a, b\rangle$ between them will be:

$$H\langle a, b\rangle = \max\left(|x_1 - x_2|, |y_1 - y_2|, |z_1 - z_2|\right) \tag{1}$$

Also, the algorithm can efficiently record the path length for each jump point identified during its processing. Then, the algorithm is able to put all the jump points that it has already found but unprocessed into a minimum heap according to a value f such that:

$$f = length\_of\_path(n) + H\langle n, goal\rangle \tag{2}$$

Which n is a jump point it has got but unprocessed, $length\_of\_path(n)$ represents the length of the path currently found from the starting node to node n.

DOI: 10.54254/2755-2721/150/2025.22521

4. Experiment Setup, Result and its Analysis

Owing to personal constraints, the following is a list of the relevant programs and devices that were utilized in the course of this experiment:

1. Grid Generation: The program is written in Python. The hexagonal grid maps used are randomly generated by generating them with a specified radius, and then traversing the nodes in each grid. During the traversing process, each node has a probability of becoming an obstacle node with an input value (i.e. obstacle rate).

2. Algorithms: The A-Star algorithm applicable to hexagonal grids and the adjusted JPS algorithm are both written in python. The operation of the A-Star algorithm is not elaborated here, and the general logic of the adjusted JPS algorithm can be seen in the previous text.

3. Devices: A 2.5 GHz AMD Ryzen 9 7945HX processor with 16GB RAM, running Windows 11 24H2.

The experimental method is as follows: the experiment will be conducted in several rounds, with several tests conducted in each round. For each test, the program will randomly generate a hexagonal grid with the value of that round as the radius. Then, the program will randomly specify two nodes with a distance (not path length) greater than or equal to the value of the round but less than or equal to 1.25 times the value of the round as the starting and ending points (the program will ensure that both the starting and ending points are passable nodes). For each test, if there exists a path and the path length is less than or equal to 1.25 times the total number of rounds, the program will record the length of the path and the time required for both algorithms. If there are multiple data with the same path length, the program will calculate their average time as the data displayed in the chart.

Firstly, this study conducted an experiment with a total of 300 rounds, 50 times per round, and an obstacle rate of 0.25. The result is shown in Figure 3:

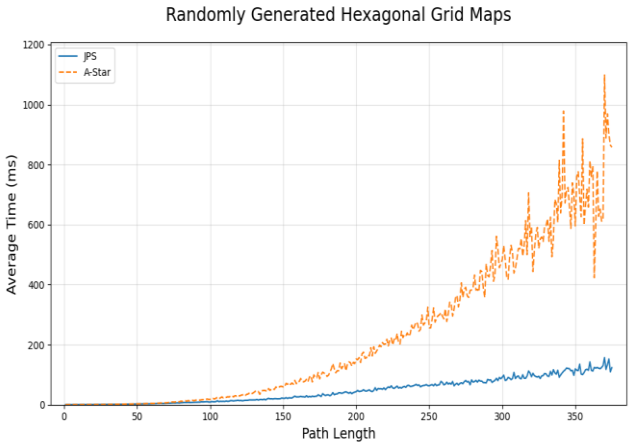

Figure 3: Result of the first experiment with rounds 300, times per round 50 and obstacle rate 0.25.

The experimental results indicate that for hexagonal grid maps, the adjusted JPS algorithm has higher efficiency than the A-Star algorithm in long path situations. As evidenced by the preceding chart, the efficiency disparity between the modified JPS algorithm and the A-Star algorithm increases with the length of the path. Regarding short path scenarios, here are the result from another experiment, comprising 50 rounds and 100 times per round, with an obstacle density of 0.25, the result is as shown in Figure 4:

DOI: 10.54254/2755-2721/150/2025.22521

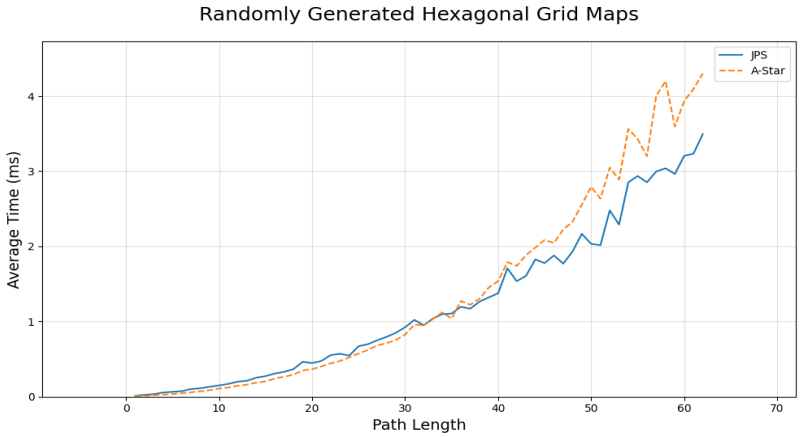

Figure 4: Result of the experiment with rounds 50, times per round 100 and obstacle rate 0.25.

It can be seen that in the case of shorter paths, the A-Star algorithm is comparable to the adjusted JPS algorithm, and the adjusted JPS algorithm is even slower in the case of extremely short paths. I infer that this phenomenon is related to the order of search directions: when the adjusted JPS algorithm processing maps, for situations that the path is short, the algorithm may only get fewer jump points - meaning fewer searches. At this stage, the processing sequence of the "correct direction" search for jump points during the algorithm's execution significantly influences the runtime, particularly at the starting point where a search in all six directions is undertaken despite only one being necessary. This aspect represents a potential avenue for the optimization of the algorithm. In conclusion, the modified JPS algorithm demonstrates substantially higher efficiency compared to the A-Star algorithm when applied to complex scenarios and medium to long-distance paths.

5. Conclusion

This paper provides a JPS algorithm that is adjusted to hexagonal grid maps, which still follows the basic framework of the original JPS algorithm, namely Forced Neighbour, Jump Point, Pruning Rules, and Jumping Rules are basic principles, but necessary adjustments have been made to these frameworks for to adapt to new hexagonal grid map scenarios. This research offers a comprehensive elucidation of the implementation of the newly adjusted JPS algorithm, encompassing essential aspects such as the identification of jump points and the application of heuristic functions. Furthermore, through controlled comparative experiments, this study has substantiated that the adjusted JPS algorithm exhibits substantial advantages over the widely employed A-Star algorithm, particularly in scenarios characterized by medium to long distance paths and complex trajectories. The algorithm's processing speed is notably superior, and the disparity in running time between the new algorithm and the A-Star algorithm becomes increasingly pronounced as the path length extends. Naturally, the new algorithm has some potential issues, and one possible optimization direction is to identify the priority of the direction that needs to be searched through some method to improve its performance in short paths. Meanwhile, the direction of expanding properties, such as improving the algorithm into 3D hexagonal grid scenarios and optimize according to the needs of certain specific professional fields, is also an interesting direction. In addition, changing heuristic functions to optimize their performance may also be one of the potential optimization directions.

References

[1] *Harabor, D., & Grastien, A. (2011). Online Graph Pruning for Pathfinding On Grid Maps. Proceedings of the AAAI Conference on Artificial Intelligence, 25(1), 1114-1119.*

DOI: 10.54254/2755-2721/150/2025.22521

[2] *Harabor, D.; and Grastien, A. (2012). The JPS pathfinding system. In International Symposium on Combinatorial Search, volume 3.*
[3] *Foead, D.; Ghifari, A.; Kusuma, M.B.; Hanafiah, N.; Gunawan, E. (2021). A Systematic Literature Review of A* Pathfinding. Procedia Comput. Sci., 179, 507–514.*
[4] *Nobes, T.; Harabor, D.; Wybrow, M.; and Walsh, S. (2022). The Jump Point Search pathfinding system in 3D. Proceedings of the 15th International Symposium on Combinatorial Search (SoCS), 15(1): 145–152.*
[5] *Min J.-G., Ruy W.-S., Park C. S. (2020). Faster pipe auto-routing using improved jump point search. International Journal of Naval Architecture and Ocean Engineering, 12, 596–604.*
[6] *Mehta P, Shah H, Shukla S, Verma S. (2015). A Review on Algorithms for Pathfinding in Computer Games. 2nd International Conference on Innovations in Information Embedded and Communication Systems.*