Journal of Computer and Communications, 2020, 8, 1-13
https://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

An Intelligent Bi-Directional Parallel B* Routing Algorithm

Xueyu Zhang*, Caihong Li

School of Computer Science and Technology, Shandong University of Technology, Zibo, China
Email: *zxy.980622@gmail.com

How to cite this paper: Zhang, X.Y. and Li, C.H. (2020) An Intelligent Bi-Directional Parallel B* Routing Algorithm. *Journal of Computer and Communications*, **8**, 1-13.
https://doi.org/10.4236/jcc.2020.87001

Received: June 13, 2020
Accepted: June 27, 2020
Published: June 30, 2020

Abstract

Pathfinding is a kind of problem widely used in daily life. It is widely used in network games, map navigation and other fields. However, the traditional A* algorithm has some shortcomings, such as heuristic function needs to be designed according to different problems, path has many inflection points, and algorithm stability is poor. B* algorithm also has the shortcoming of inaccurate pathfinding. In order to solve the problems existing in A* and B* algorithms, obstacle avoidance regeneration mechanism, pre-exploration mechanism and equivalent waiting strategy are proposed. It adds a bidirectional parallel search mechanism to form an IBP-B* algorithm (Intelligent bi-directional parallel B* routing algorithm). The simulation results show that the speed of IBP-B* algorithm is 182% higher than that of A* algorithm and 366% higher than that of BFS algorithm. Meanwhile, compared with B* algorithm, IBP-B* algorithm improves the pathfinding accuracy of the algorithm.

Keywords

B* Algorithm, Path Planing, Parallel

1. Introduction

The problem of pathfinding is a key issue in graph theory, and it is widely used in many fields such as intelligent transportation [1] and online games [2]. Regarding this problem, common pathfinding methods include BFS algorithm, Dijkstra algorithm, and A* algorithm. Among them, BFS algorithm and Dijkstra algorithm belong to blind search [3]. BFS algorithm is based on queue implementation, adding all feasible points to the queue. A lot of memory space is wasted [4]. Dijkstra updates according to the shortest path from the starting point to other points, which is a waste of time for the problem of finding a path between two points. The A* algorithm is an improved heuristic algorithm based on the BFS algorithm [5]. Compared with other algorithms, it is currently the most widely used algorithm and can basically meet the requirements of pathfinding, but also has many obvious disadvantages, such as heuristic The function needs to be designed according to different situations [6] [7]. The path searched by the algorithm is often tortuous, which is inconsistent with the actual situation [8], and the dynamic programming effect is poor [9].

In recent years, scholars at home and abroad have conducted in-depth research on this problem. In view of the shortcomings of the A* algorithm, there are currently two solutions. One is to improve based on the characteristics of the A* algorithm heuristic function or optimize it in conjunction with the parallel mechanism, such as Xiaoyan Guo and Xun Luo explored the influence of different heuristic functions on the algorithm [10], Vahid Rahmani and Nuria Pelechan proposed a multi-objective hierarchical parallel A* algorithm [11]; Li Wencong proposed a searchable infinite neighbor based on steering restrictions Bidirectional heuristic algorithm in the domain [12]. Qin Feng *et al.* adopted a bidirectional preprocessing structure to reduce the number of redundant nodes in the A* algorithm [13]. Wang Zhongyu *et al.* optimized the path by improving the weight ratio of the evaluation function [14]. The other is the targeted improvement of the algorithm based on the characteristics of the problem to be solved. For example, Pan Changan studied the urban traffic routing problem based on the A* algorithm [15], using a binary fork structure and adjusting the heuristic function in the valuation function. Xue Shuangfei used the A* algorithm for optimization research on the obstacle avoidance problem of ships at sea [16]. The A* algorithm was used to extract each inflection point and then tested, and the redundant points were removed. In addition, some scholars correspondingly improved other algorithms for specific problems. For example, Yang Weidong and others solved the dynamic pathfinding problem based on the genetic algorithm of niche immunity [17]. Yu Lingjie and Gu Peng combined neural network and Q-learning to propose In addition to the DQN algorithm [18], Liu Xiaotao and others used support vector machines to optimize the D* algorithm to solve the problem of unmanned vehicle pathfinding [19], and so on.

Zhao Qingsong proposed a new B* algorithm based on mimicking nature animal pathfinding in 2010 [20]. This algorithm is mainly used to solve the problem that a large number of pathfinding requests need to be completed quickly in the game server, and the efficiency can reach dozens of times of the traditional A* algorithm. Compared with the A* algorithm, the algorithm is simple in principle and does not require complex heuristic function design for different problems. However, the algorithm is prone to inaccurate pathfinding, and sacrifices the accuracy of pathfinding in pursuit of efficiency. On the basis of this, this paper improves the problem that the original algorithm cannot find the correct path in some cases through the cooperation of pre-detection mechanism, obstacle avoidance mechanism and the same-level waiting strategy. Then, by adding parallel search mechanism, an intelligent bidirectional B* pathfinding algorithm is proposed. Experimental results show that the new algorithm is more practical, with higher stability, faster speed and less path turns than the traditional A* algorithm.

2. Problem Description

For the convenience of research, the problems studied in this paper are expressed as follows: given a grid map T, it is represented by $N * M$ squares of the same size, and each square is represented by N_{ij}.

$$T = \sum N_{ij}; i \in [1, N], j \in [1, M] \tag{1}$$

The information of each square in the figure is specifically expressed as

$$N_{ij} = \begin{cases} 0, & \text{No obstacles} \\ 1, & \text{There are obstacles} \end{cases} \tag{2}$$

When the value is 0, it indicates that there is no obstacle in the current square, and it can move freely. When the value is 1, it indicates that there is an obstacle in the current square, which is impassable. This article selects four connection methods for research, that is, there are four directions to choose from when walking, which are up, down, left and right. In the grid map T, let S be the starting point, E is the end point, P is a road from S to E in the grid map T, and define the weight of the road P as the sum of the weights of all the edges in the road, denoted as. The problem required in this paper can be described as finding the path with the smallest weight from all the paths from S to E. That is to find a way from S to E, so that

$$W(P_0) = \min\{W(P)\}, \quad P \in D \tag{3}$$

3. B* Pathfinding Algorithm

3.1. Algorithm Introduction

B* pathfinding algorithm is an efficient pathfinding algorithm invented by Zhao Qingsong in 2010 inspired by the pathfinding process of real animals in nature. It combines the greedy algorithm and the breadth-first search algorithm, and selectively adds points to the search queue through the greedy strategy, which effectively reduces the space waste of the breadth-first search algorithm and improves the search speed. Moreover, because the algorithm is based on a greedy strategy, its search path is more in line with the law of natural biological pathfinding than the A* algorithm, and there are fewer inflection points.

3.2. Algorithm Principle

The B* algorithm is inspired by the pathfinding process of real animals in nature, and is improved to solve various blocking problems. During the pathfinding process, the exploration nodes are divided into two states: free exploration nodes and crawling exploration nodes. The initial exploration node is free, and moves from the origin to the goal. When encountering an obstacle, it branches along the obstacle into left and right. Each branch constitutes a crawling exploration node, and attempts to bypass the obstacle. After the obstacle is bypassed, the crawling node becomes a free node and advances to the target. This process iterates until the path finding success or exploration is reached. The disappearance of the node indicates that there is no reachable path.

The algorithm process is shown below.

Step 1. Initially, the exploration node is a free non-climbing node, starting from the origin and moving towards the self-marking point.

Step 2. Determine whether the front is an obstacle: it is not an obstacle, move forward one step to the target, and it is still a free non-climbing node; it is an obstacle, bifurcating into two branches along the obstacle, trying to bypass the obstacle from two directions, this two branch nodes become the crawl around exploration nodes.

Step 3. After crawling around the exploration node to bypass the obstacle, it becomes a free non-climbing node again, and return to 2.

Step 4. After the exploration node advances, determine whether the current map grid is the target grid, if so, the pathfinding is successful, and a complete path is constructed according to the pathfinding process. .

Step 5. During the pathfinding process, if the search node is gone, the pathfinding ends and the target grid is unreachable.

4. IBP-B* Algorithm Introduction

The B* algorithm loses some accuracy while improving speed. Based on this, this paper adopts new strategies, pre-exploration mechanism and obstacle avoidance rebirth mechanism to improve the accuracy of its algorithm. At the same time, it adds a parallel mechanism and proposes an intelligent two-way parallel B* pathfinding algorithm.

4.1. Greedy Strategy

The core part of the B* algorithm is a greedy strategy that imitates animals in nature. The strategy is as follows: Assume that the current point position is point F and the end point position is point E.

When there is no obstacle in front of the current movement direction of point F, the difference between the horizontal and vertical and the end point is compared, and the judgment is made according to the relative position between the current position F and the end position E, thereby determining the next direction of movement, according to the greedy strategy, if the absolute value of the lateral difference between the current position and the end position is greater than the absolute value of the longitudinal difference, then the relative position between the two points will be judged, and it will move laterally.

When an obstacle appears in front of the current movement direction of point F, it is divided into two cases. The first is the first encounter with the obstacle, and then it is extended in the other directions except the direction of travel. If it is encountered multiple times, the judgment Whether there is still an obstacle in the original collision direction, if it does not exist, it will expand in the direction of the original obstacle, if it exists, it will continue to expand in the current direction, especially if there is an obstacle in the current direction, it will proceed in the opposite direction of the original obstacle Expand accordingly.

4.2. Obstacle Avoidance Rebirth Mechanism

As shown in Figure 1, the center of the grid in the upper left corner of the figure is the coordinate origin (1, 1), the positive x-axis is downward, and the positive y-axis is right. Grid map. Then the starting point S is located at (3, 1), and the point E is located at (3, 7). According to the greedy strategy algorithm in 2.2, the correct path cannot be found. In order to solve this problem, this paper proposes an obstacle avoidance rebirth mechanism. Let the current movement point be A and the current movement direction be D during the algorithm. When point A moves along direction D to point B, first determine whether there are obstacles on the left and right of the direction D of point B. If there are obstacles, add point A to the queue while adding A to the queue. The point after the point moves in that direction.

4.3. Pre-Exploration Mechanism

As shown in Figure 2, with the center of the grid in the upper left corner of the figure as the coordinate origin (1, 1), the downward x-axis is positive, and the right is the y-axis positive, to construct a plane rectangular coordinate system and build the corresponding Grid map. Then point S is at (3, 1) and point E is at (3, 7). It can be found that when the S point expands to the right to the position (3, 3), because there are no obstacles above and below, the algorithm will enter a "trap", and the obstacle avoidance rebirth mechanism has no effect on this situation. Therefore, this paper proposes a pre-exploration mechanism based on the idea of reconnaissance and detection, and pre-judgment at the current point position. If the current point next entry position is a concave glyph, it will expand in other directions at the same time.

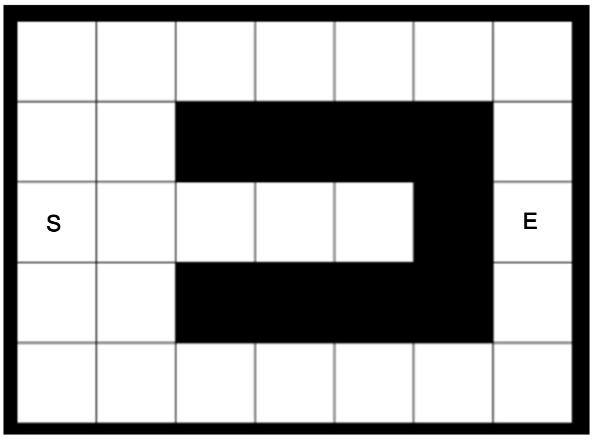

Figure 1. Example of obstacle avoidance rebirth mechanism.

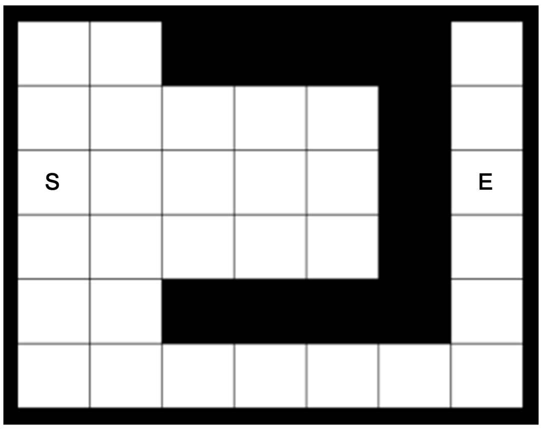

Figure 2. Example of pre-exploration mechanism.

4.4. Two-Way Parallel Search

In order to make up for the time overhead caused by the obstacle avoidance rebirth mechanism and the pre-exploration mechanism, and at the same time improve the efficiency of the algorithm, this algorithm implements a two-way parallel search. The specific operations are as follows: first, the starting point S is added to the queue Q1, and the ending point E is added to the queue at the same time. Q2, each time the head of the queue is taken from queue Q1 and queue Q2 at the same time. The search strategy adopted by forward search and reverse search is exactly the same. When the search reaches the end point in the search process, the search ends, that is, the search ends when the forward search reaches the end point E, and the search ends when the reverse search reaches the start point S. When queue Q1 and queue Q2 are both empty or the forward and reverse search ends at the same time, the algorithm ends.

4.5. Peer Waiting Strategy

First, let the point where the end position is encountered for the first time in the search process be point H. Due to obstacle avoidance rebirth and the setting queue of the pre-exploration mechanism, some points that are not optimal are added to the queue. Secondly, in the queue, the best and other non-best advantages cannot be prioritized. Therefore, at the end of the algorithm, it is not easy to determine that the result obtained after a certain point reaches the end point is the final result. You need to wait for point H to pop up the queue one by one in the queue with the same search level before you can judge the result obtained at this time. The result is the final result.

Secondly, in the process of running the algorithm, there is also a situation that may produce errors. As shown in Figure 3, when the algorithm is in this situation, since the algorithm has no priority difference between the four directions when expanding the multiple directions, an error of 2 will be generated. In order to make up for this error, it is mandatory to wait for all points that differ from the search depth of point H by 2 to pop out of the queue before terminating the operation of the algorithm.

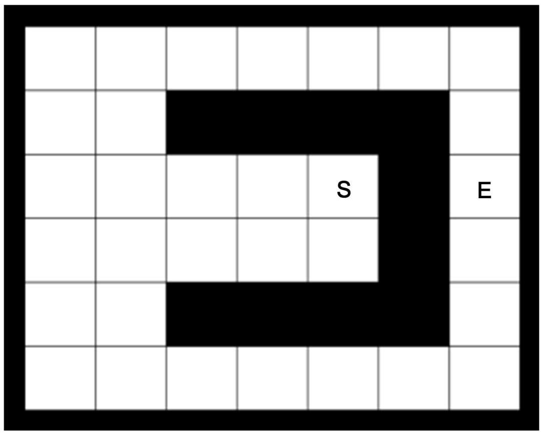

Figure 3. Example of peer waiting strategy.

4.6. Algorithm Structure

The intelligent two-way parallel B* pathfinding algorithm is based on two queues when searching for paths. One is used to store the search queue starting from the start point, and one is used to store the search queue starting from the end point. During the running of the algorithm, the first points of the two queues are taken from the two queues for expansion. First, it is judged whether they can move directly according to the greedy strategy. Secondly, if an obstacle is touched during the movement, it will be discussed in different situations. The first time the obstacle is touched. The object expands in other directions. If the obstacle has been touched before, it is judged whether the obstacle in the direction before the current point exists and expands according to the state of the obstacle. When both the forward and reverse search have ended, the algorithm ends and the output, the best results found during the search. The overall structure of the algorithm is shown in Figure 4.

The specific steps of the algorithm are as follows:

Step 1. Create queues Q1 and Q2, add the start point to queue Q1, and the end point to queue Q2.

Step 2. Check whether the queues Q1 and Q2 are empty, or both the forward and reverse searches have found the results and ended. If both are empty or both ends, skip to step 7, otherwise go to step 3.

Step 3. First, perform a forward search. If queue Q1 is not empty, then take point F1 from the head of queue Q1 and pop it from the queue, go to step 4, otherwise go to step 5.

Step 4. Then determine whether the point F1 can move in accordance with the greedy strategy, whether it is the first time it has encountered an obstacle, or whether it has been encountered multiple times. If the F1 point can move directly according to the greedy strategy, the point after the F1 point moves is calculated according to the greedy strategy; if the F1 point encounters an obstacle for the first time, the state of the obstacle is recorded, and it expands in the remaining movement direction; In order to encounter the obstacle many times,

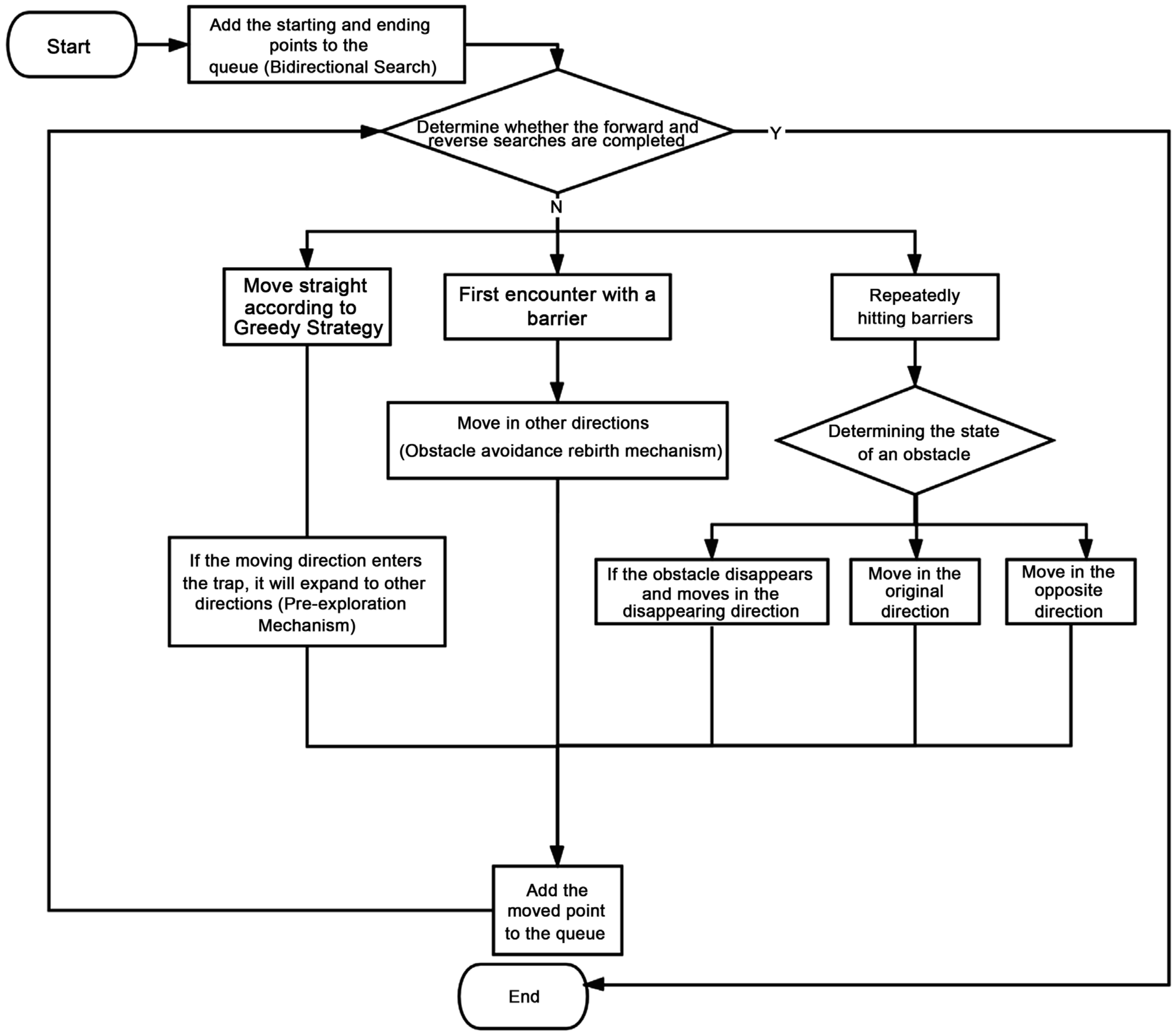
Start
Add the starting and ending points to the queue (Bidirectional Search)
Determine whether the forward and reverse searches are completed
Y
N
Move straight according to Greedy Strategy
First encounter with a barrier
Repeatedly hitting barriers
Move in other directions (Obstacle avoidance rebirth mechanism)
Determining the state of an obstacle
If the moving direction enters the trap, it will expand to other directions (Pre-exploration Mechanism)
If the obstacle disappears and moves in the disappearing direction
Move in the original direction
Move in the opposite direction
Add the moved point to the queue
End

Figure 4. Algorithm flowchart.

the state of the obstacle in the current moving direction is judged, and the corresponding movement is made according to its state. If there is an obstacle in the original direction, it moves in the opposite direction of the obstacle. Finally, the points calculated according to different strategies are re-added to the queue Q1.

Step 5. Next, perform a reverse search. If queue Q2 is not empty, take point F2 from the head of queue Q2 and pop it from the queue, go to step 6, otherwise go to step 2.

Step 6. In turn, determine whether point F2 can move according to the greedy strategy, whether it is the first time it has encountered an obstacle, whether it has been encountered multiple times. The status moves accordingly, and finally the points moved according to different strategies are added to the queue Q2 again, and go to step 2.

Step 7. The operation of the program ends.

5. Simulation

In order to verify the performance of the new algorithm, based on C++ programming under the Windows operating system, this paper conducted experiments on breadth-first search (BFS), A* algorithm, B* algorithm and intelligent two-way parallel B* search algorithm. The experimental operating environment is Windows 10 Professional Edition, the processor is Intel(R) Core i7-10710U 1.10 GHZ ten-core processor, 16 GB memory, 1T hard disk, and the compiler version is GCC 5.1.0.

5.1. Experiment Procedure

First, according to the principle of the B* algorithm, five sets of test samples are designed to test the correctness of the four algorithms. The five test samples are shown in Figure 5. The path length results found by each algorithm are shown in Table 1, where −1 indicates that the path cannot be found.

We imitate the examples in [15] and generate a set of test cases 6 of size 35 * 35 to explore the characteristics of the path generated by the A* algorithm and the IBP-B* algorithm. The path searched by the A* algorithm is shown in Figure 6, and the path generated by the IBP-B* algorithm is shown in Figure 7.

Then we randomly generate a set of sample 7 with a size of 1000 * 1000 that does not exist, and test the four algorithms. The experimental results are shown in Table 2.

Table 1. Running results of different algorithms.

Algorithm	BFS	A*	B*	IBP-B*
Sample 1	12	12	16	12
Sample 2	12	12	18	12
Sample 3	11	11	−1	11
Sample 4	12	12	14	12
Sample 5	12	12	−1	12

Table 2. Running time and number of search nodes.

Algorithm	BFS	A*	B*	IBP-B*
The elapsed time(s)	1.351	0.329	0.002	0.078
Number of search nodes	376,748	376,748	4619	136,641

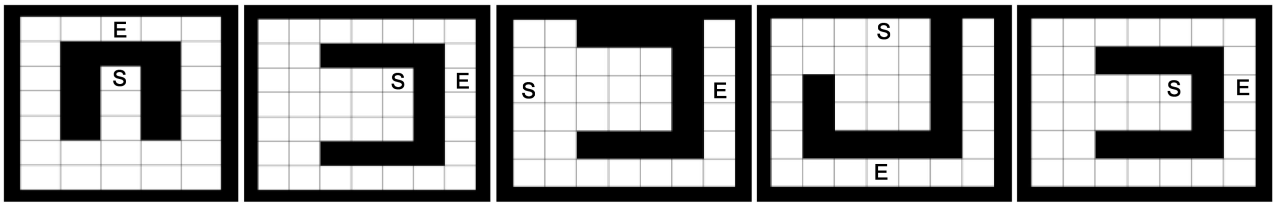

Figure 5. Five groups of test cases.

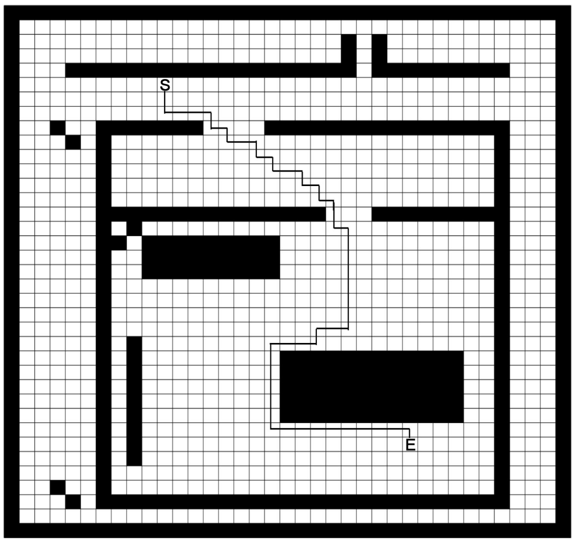

Figure 6. A* Algorithm pathfinding results.

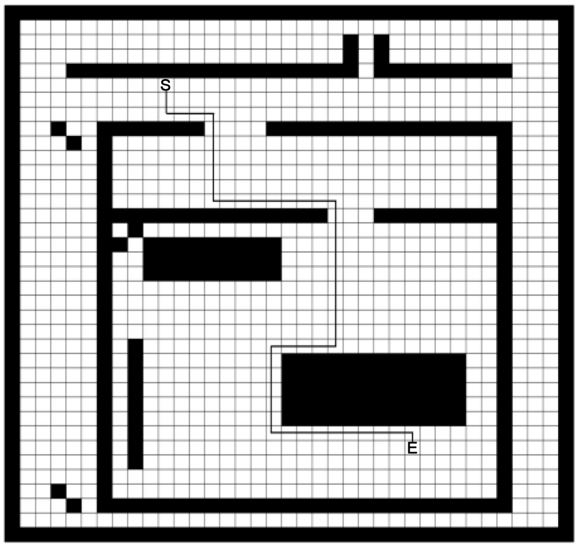

Figure 7. IBP-B* algorithm pathfinding results.

Next, we test the efficiency of the four algorithms. In order to ensure the test accuracy, we first generate a set of 1000 * 1000 sample 8 and use each algorithm to run the sample ten times to explore the operating system under the same hardware environment. The degree of influence of factors such as scheduling on the running time of the algorithm. The results are shown in Figure 8.

From the experimental results, the running time of the algorithm under the same hardware environment is slightly different at different times, but the overall gap is not large, so you can use the results of a certain run as the results of the algorithm. We randomly generated 15 sets of samples with a size of 1000 * 1000, and each algorithm was run once for each algorithm to explore the performance of each algorithm in the case of random data. The experimental results are shown in Table 3 and Figure 9.

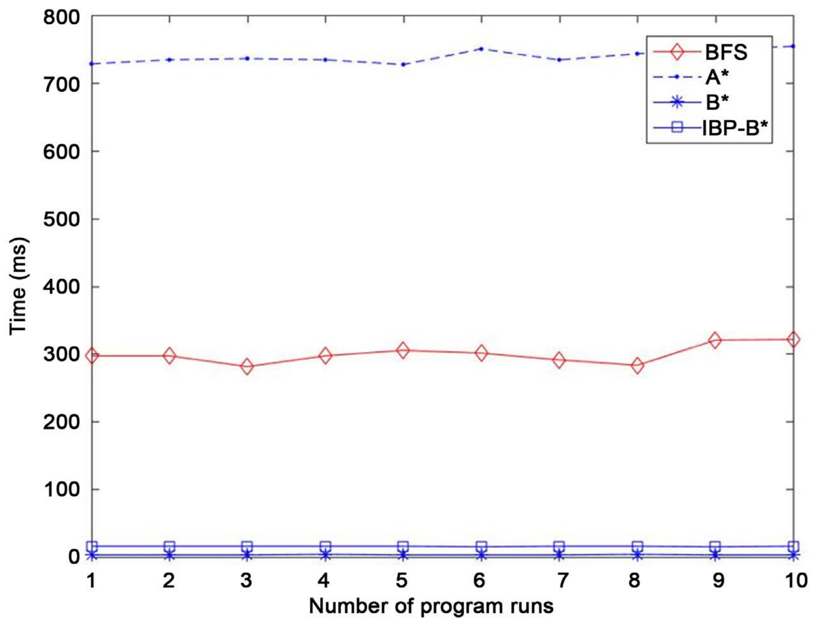

Figure 8. Running time of each algorithm.

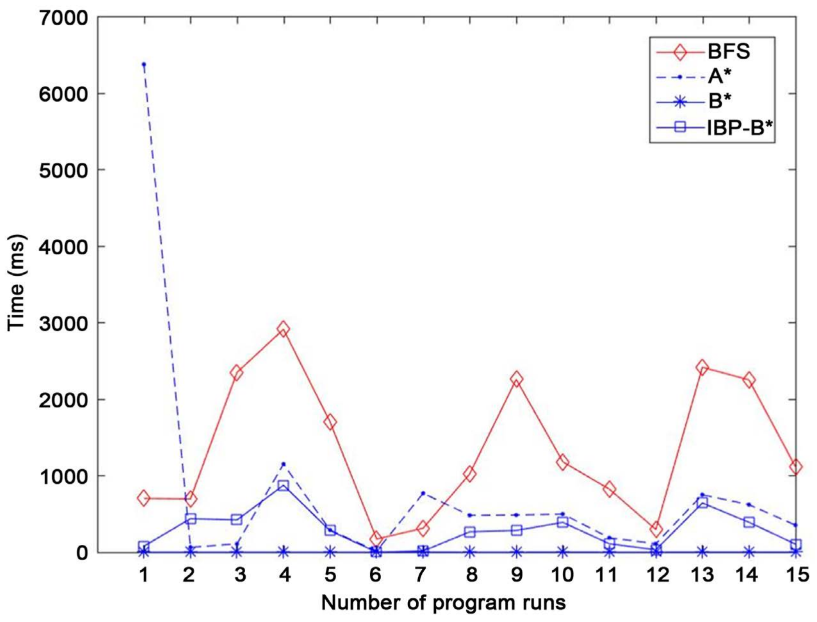

Figure 9. Running time of different algorithms.

Table 3. Average running time and path-finding accuracy.

Algorithm	BFS	A*	B*	IBP-B*
Average running time(s)	1.443	0.876	0.001	0.309
Pathfinding accuracy	100%	66.7%	87%	100%

5.2. Analysis of Results

First of all, this paper uses the sample designed for the B* algorithm to test the four algorithms. The results show that the IBP-B* algorithm, BFS algorithm, and

A* algorithm can correctly calculate the results in the test and verify the IBP-B*. The correctness of the algorithm. Then, through the test of sample 6, the test results are obtained in **Figure 6** and **Figure 7**, and it can be clearly seen that the path generated by the IBP-B* algorithm is smoother than the A* algorithm, and the inflection point in the path less. This is because the principle of the IBP-B* algorithm is based on a greedy strategy. From the test of Example 7, we find that when there is no path, the BFS algorithm and the A* algorithm will traverse the entire graph, and thus search for the most nodes. The B* algorithm and the IBP-B* algorithm are based on the greedy strategy, which greatly reduces some unnecessary expansion in the search process, so the number of search nodes is relatively small. In order to ensure the accuracy of the results, the IBP-B* algorithm adds some search branches to ensure the accuracy of the algorithm, so the number of search nodes has increased compared to the B* algorithm. By analyzing the results of 15 sets of random test samples, we can find that the A* algorithm runs fast and slow in various tests and has poor stability. The running time of each group of test cases of the IBP-B* algorithm is lower than that of the A* algorithm and the BFS algorithm. The algorithm has good stability and fast running speed. The experimental results show that the speed of the IBP-B* algorithm is improved by 182% compared to the A* algorithm and 366% compared to the BFS algorithm. At the same time, compared with the B* algorithm, the IBP-B* algorithm improves the algorithm's pathfinding accuracy.

6. Conclusion

The traditional A* algorithm is prone to redundant points and inflection points during the pathfinding process, and the heuristic function needs to be designed according to different problems. Zhao Qingsong proposed a B* algorithm based on bionics, which is efficient and easy to implement and the planned path is smooth, but it is prone to inaccurate pathfinding when planning the path. In this paper, the accuracy of the original algorithm is improved by adding obstacle avoidance rebirth mechanism, pre-exploration mechanism and sibling waiting strategy. At the same time, bidirectional parallel mechanism is added, and an intelligent bidirectional parallel B* pathfinding algorithm is proposed. Through experimental simulation, compared with other algorithms, the IBP-B* algorithm works through a variety of mechanisms to maintain the advantages of the B* algorithm's smooth path generation, practicality and wide applicability, while improving the accuracy of pathfinding. Compared with the BFS algorithm and the traditional A* algorithm, the IBP-B* algorithm greatly shortens the pathfinding time and provides a new solution for the global path planning in various pathfinding problems.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Jin, M.C. (2019) Research on Traffic Route Based on Hybrid Intelligent Algorithm.

Inner Mongolia University of Technology, Hohhot.

[2] Han, Y.Z. (2017) Path-Finding Algorithm in Online Games. *Electronic Technology and Software Engineering*, No. 24, 16.

[3] Cai, Z.X. (1996) Artificial Intelligence and Its Applications. Tsinghua University Press, Beijing.

[4] Zhu, L. (2010) Research on the Way-finding Problem in the Game. *The Science Education Article Gallery*, No. 2, 69, 83.

[5] Lian, Y., Wang, C.L, He, L., Zeng, X.M, Cui, T.J. and Chen, L. (2017) Solving the Shortest Path Based on Improved Heuristic Ant Algorithm. *Journal of Tianjin Normal University* (*Natural Science*), **37**, 54-57.

[6] Jiang, Y.Q., Li, Z.Y., Guan, Q.X. and Guan, S.J. (2020) Research on UAV Path Planning Based on Improved A* Algorithm. *Journal of Ordnance Equipment Engineering*, 1-5. http://kns.cnki.net/kcms/detail/50.1213.TJ.20191227.1324.004.html

[7] Wan, B.W., Chen, J., Zhu, R.C., Zhu, D.W. and Pan, Z.Y. (2019) Application and Research of Advanced A~* Algorithm in Game Pathfinding Function. *Information Research*, **45**, 51-55.

[8] Yang, K.X. (2009) Artificial Intelligence Path-Finding Algorithm and Its Application in Games. Central South University, Changsha.

[9] Zang, C. and Sun, Q.Q. (2018) Dynamically Adjustable A~* Path Optimization Algorithm Based on Path State. *Industrial Control Computer*, **31**, 101-102.

[10] Guo, X.Y. (2018) Global Path Search Based on A~* Algorithm. *Proceedings of the* 2018 *International Conference on Transportation & Logistics*, *Information & Communication*, *Smart City* (*TLICSC* 2018). Wuhan Zhicheng Times Cultural Development Co., Ltd., Wuhan, 380-385.

[11] Rahmani, V. and Pelechano, N. (2020) Multi-Agent Parallel Hierarchical Path Finding in Navigation Meshes (MA-HNA*). *Computers & Graphics*, **86**, 1-14. https://doi.org/10.1016/j.cag.2019.10.006

[12] Qin, F., Wu, J., Zhang, X.F. and Zhao, J.L. (2019) Improved Search Algorithm Based on A~* Bidirectional Preprocessing. *Computer System Application*, **28**, 95-101.

[13] Yu, L.J. and Gu, P. (2019) Labyrinth Pathfinding Based on DQN Algorithm. *Information and Computer* (*Theoretical Edition*), No. 11, 31-32, 36.

[14] Li, W.Y., Gao, Z.Z., Wu, D.F. and Li, S.G. (2019) Improved Bidirectional Heuristic Shortest Path Algorithm Based on Turn Limit. *Science Technology and Engineering*, **19**, 169-174.

[15] Wang, Z.Y., Zeng, G.H., Huang, B. and Fang, Z.J. (2019) Global Optimal Path Planning of Robot Based on Improved A~* Algorithm. *Computer Application*, **39**, 2517-2522.

[16] Pan, C.A. (2015) Research on Urban Traffic Routing Based on Improved A Star Algorithm. Huaqiao University, Xiamen.

[17] Xue, S.F., Xie, L., Wang, S.W., Xia, W.T. and Bao, Z. (2018) Ship A~* Collision Avoidance Algorithm for Offshore Wind Farms. *China Navigation*, **41**, 21-25.

[18] Yang, W.D. (2009) Genetic Algorithm Based on Niche Immunity to Solve the Path-finding Problem of Dynamic Obstacle Avoidance Maze. *South China Financial Computer*, **17**, 34-36.

[19] Liu, X.T., Cai, Y.F. and Wang, T.C. (2017) Application of Constrained D* Algorithm Based on SVM in Unmanned Vehicle Path-Finding. *Computer and Digital Engineering*, **45**, 1748-1754.

[20] Zhao, Q.S. (2010) A New Efficient Path-Finding Algorithm in Game Development—B~*Pathfinding Algorithm. *Programmer*, No. 8, 110-111.