A Comprehensive Survey and Analysis on Path Planning Algorithms and Heuristic Functions

Bin Yan[1], Tianxiang Chen[2], Xiaohui Zhu[1,2(✉)], Yong Yue[2], Bing Xu[1], and Kai Shi[1]

[1] School of Information and Science, Nantong University, Nantong 226019, Jiangsu, China
ntuzxh@hotmail.com

[2] School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou 215123, Jiangsu, China

Abstract. This paper explores the performance of four commonly used path planning algorithms of A*, D*, LPA* and D* Lite in both static and dynamic environments. To assess the effect of heuristic functions for path planning algorithms, four commonly used heuristic functions of Manhattan distance, diagonal distance, Euclidean distance and squared Euclidean distance are used based on three simulation environments with different map size and complexity. Experimental results show that the heuristic function significantly affects the computing performance and the final path of path planning algorithms, and Manhattan distance is the most efficiency heuristic function for all the four path planning algorithms. We apply A*, D*, LPA* and D* Lite algorithms to static and dynamic environments, respectively with different map size and complexity. We find that A* has the best performance in relatively simple static environments. When the map size and complexity of the static environment is highly increased, D* Lite is the best choice. In contrast, D* Lite has the best performance for path planning in dynamic environments. However, we also find that A* also has an excellent performance in dynamic environments when the map size and complexity are decreased.

Keywords: Path planning · Heuristic function · A* · D* · LPA* · D* Lite · Path planning performance

1 Introduction

With the development of robotics and artificial intelligence, a variety of path planning algorithms have been proposed [1]. Hart and Nilsson developed A* algorithm in 1968 [2]. It is one of the best heuristic algorithms to find an optimal path in static environments. It uses a heuristic function to estimate the cost from any node to the goal node, which can reduce the search space of the original Dijkstra algorithm [3,4]. It has been widely used in kinds of path planning solutions [5]. To further decrease the computing time of A*, J. Szczerba proposed

K. Arai et al. (Eds.): SAI 2020, AISC 1228, pp. 581–598, 2020.
https://doi.org/10.1007/978-3-030-52249-0_39

Sparse A* Search (SAS) [6]. It uses a constraint function to reduce search space resulting in a faster search and less computing time. However, both algorithms are only suitable for path planning in static environments. They have to plan the whole path again when the environment is changed, which is inefficient. To achieve efficient path planning when the environment is partially known or is continuously changing, Anthony Stentz developed D* algorithm (also known as Dynamic A* algorithm) [8–10]. It was applied to the Mars probe car as the core path planning algorithm in December 1996 and achieved a huge success [11,12]. To further decrease the computing complexity of path planning algorithms, Sven Koening proposed the Lifelong Planning A* algorithm (LPA*). Due to the repetitive use of the surrounding environment information, it can sufficiently improve the search efficiency [13,14]. However, LPA* only aims to plan a path from a particular start node to the goal node. The path has to be re-planned entirely when a new obstacle comes up. To solve the issue of dynamic start node and particular goal node, Sven Koening improved the LPA* and developed D* Lite [15,16]. Combined with heuristic searching and incremental searching, D* Lite can plan a path from an uncertain start node to a particular goal node. Because it only deals with path information from the current node to the goal node, it has higher performance than other algorithms when re-plan the path in dynamic environments. D* Lite has been widely applied on the navigation of mobile robots and autonomous vehicles, including the prototype system that was tested on 'opportunity' and 'courage' Mars rovers, as well as the navigation system that won the match of the Defense Advanced Research Projects agency [17]. In 2011, Mansour K A applied D* Lite algorithm to multi-agent dynamic path planning [18].

At present, A*, D*, LPA* and D* Lite algorithms are widely used, and many researchers have made improvements using different heuristic functions. However, little analysis and comparison have been carried out to identify the effect of heuristic functions to all these path planning algorithms. Furthermore, the performance of these four algorithms has not been completely assessed and compared in different static and dynamic environments with different map size and the number of obstacles.

In this paper, we first introduce the four path planning algorithms in Sect. 2. Section 3 describes four heuristic functions we used and the simulation environment designed to evaluate the performance of four commonly used path planning algorithms. In Sect. 4, we first compare the performance of four heuristic functions based on four path planning algorithms. After that, we test four path planning algorithms in different static and dynamic environments and assess their performance, respectively. The advantages and disadvantages of all these four algorithms are concluded in the last section.

2 Review of Path Planning Algorithms

2.1 A*

A* is a classic heuristic algorithm for path planning. It has been widely applied in path planning for robots, games and navigation of vehicles [19]. The heuristic

function of A* is shown in Eq. 1.

$$F(n) = G(n) + H(n) \tag{1}$$

where G(n) denotes the cost from the start node to any node n and $H(n)$ represents the estimated cost from node n to the goal node. When moving from the start node to the goal node, A* algorithm weighs these two costs and evaluates surrounding nodes using a heuristic function. It always chooses a node with the lowest cost of $F(n)$ as the next node. The selection of $H(n)$ determines the efficiency and accuracy of the A* algorithm. We consider the heuristic function $H(n)$ in the following situations:

- When $H(n)$ always equals to 0, then only $G(n)$ works and A* transforms into the Dijkstra algorithm, which ensures A* to find the shortest path while is extremely time-consuming [20].
- When $H(n)$ is smaller than the actual cost from the node n to the goal node. In this case, A* algorithm can find an optimized path. The smaller $H(n)$ is, the more nodes A* explores, and the slower the searching efficiency is.
- When $H(n)$ is larger than the actual cost from the node n to the goal node, A* cannot guarantee to find the optimal path but can accelerate the searching procedure.
- When $H(n)$ is extremely larger than $G(n)$. It means only $H(n)$ works, and A* is converted to Breadth-First Search algorithm (BFS) [21].

So, A* can control the efficiency of the algorithm by balancing the values of $G(n)$ and $H(n)$ via different heuristic functions. The Manhattan distance is one of the commonly used heuristic functions by A*. The main procedure of A* is shown in Fig. 1.

A* uses two lists. An open list saves nodes that may be visited in the future. A closed list saves nodes that have been visited. At first, it puts the start node S into the open list. Then, A* traverses the open list to find a node with the smallest value of $F(n)$, and put it into the closed list for further processing as well as setting it as the current node. Third, checking all the successor nodes around the current node. If one successor node is unreachable or is in the closed list, ignore it, otherwise do the following operations:

- If the successor node is not in the open list, put it in and set the current node as its parent node, calculate its $F(n)$ value using Eq. 1.
- If the successor node is already in the open list, check if the new $F(n)$ value of the successor node is smaller than its original $F(n)$ value. If it is, set current node as its parent node, and recalculate its $F(n)$, $G(n)$ and $H(n)$.
- Repeat steps 2 and 3 until the goal node is in the open list, which means that the path is found. If the open list is empty, it means that there is no path exists.
- Finally, We obtain the path by moving from the goal node to the start node via the parent node of each node.

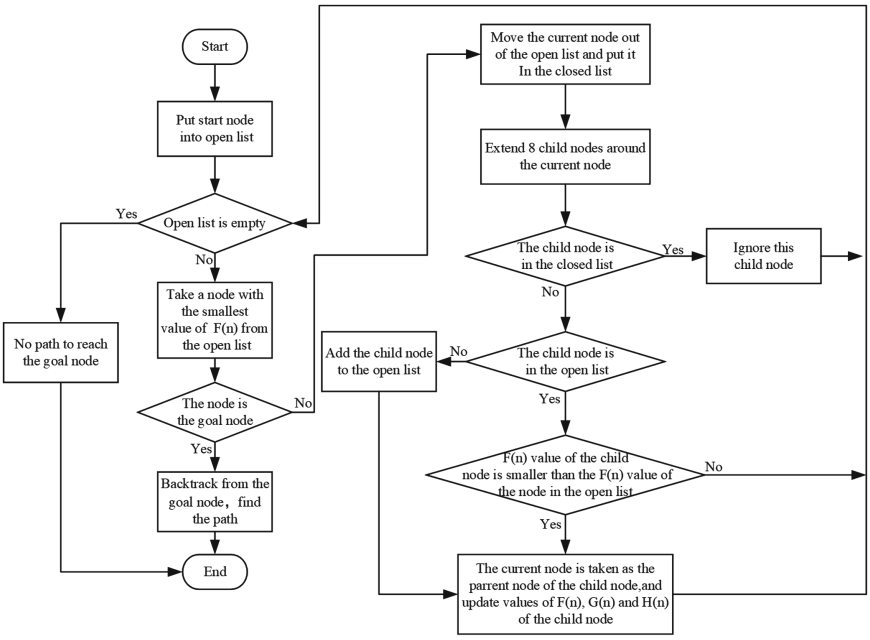


Fig. 1. Main procedure of A*

2.2 D*

A* is very efficient in static environments. However, its performance is extremely decreased in dynamic environments. It is because it needs to recalculate the whole path when new obstacles appear in dynamic environments. Compared to the A*, D* is more efficient in dynamic environments. In contrast to the forward-searching in A*, D* searches reversely from the goal node to the start node. Additionally, D* can process dynamic environment information, which is more suitable in unknown or dynamic environments.

The main idea of D* is to use static path planning algorithm to calculate the shortest path from each node to the goal node, then backtrack from the start node to the goal node. If the status of the next node is changed, then change the cost of the effected nodes for re-searching. D* mainly consists of two functions: "PROCESS-STATE()" and "MODIFY-COST()". "PROCESS-STATE()" is used to calculate the shortest path form the start node to the goal node. "MODIFY-COST()" is used to change the cost between two nodes and put affected nodes into an open list when detecting the status of the next node is changed. Figure 2 shows the main procedure of D*.

The main procedure of D* is as follows.

– First, we initialize the status of all the nodes. Each node has a status function $t(X)$ shown in Eq. 2. We initialize the status of all nodes as NEW, set $H(G)$ as 0 and put the goal node G in an open list.

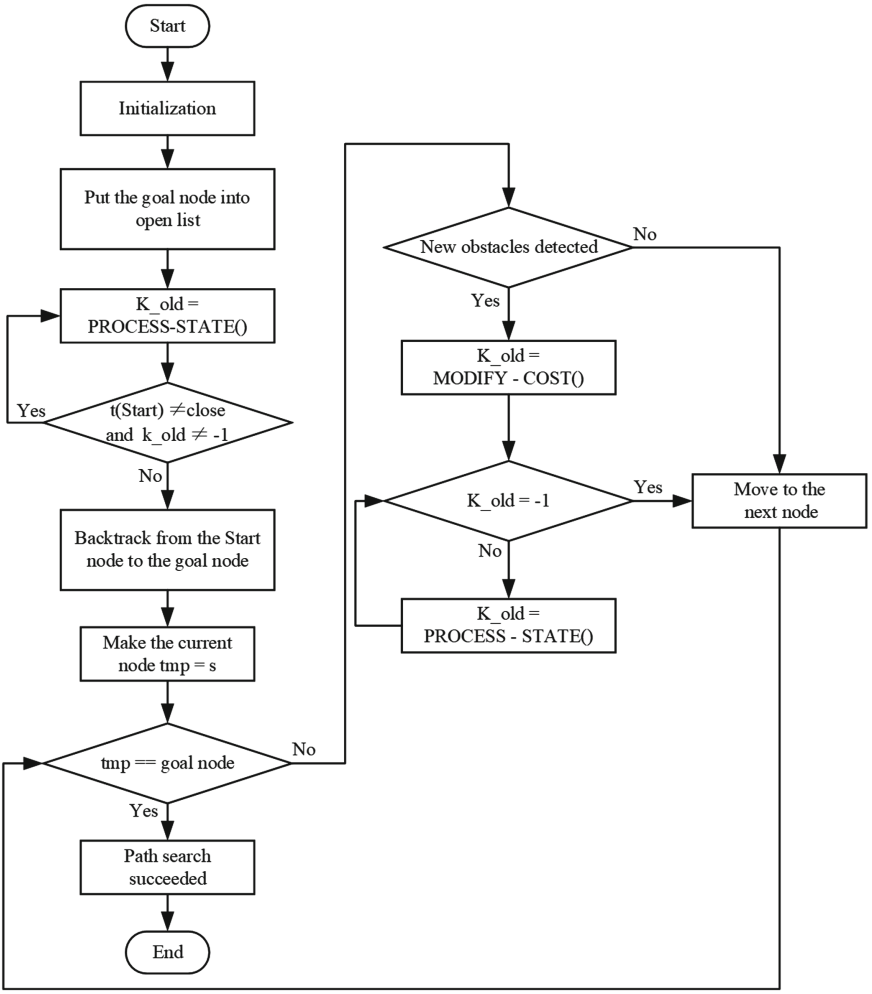


Fig. 2. Main procedure of D*

$$t(X) = \begin{cases} NEW, & \text{X has never been in the open list} \\ OPEN, & \text{X is in the open list} \\ CLOSED, & \text{X is not in the open list} \end{cases} \quad (2)$$

– "PROCESS-STATE()" function is executed repetitively until the start node X is pop up from the open list, which means the shortest path is found successfully.
– The robot starts to move from the start node to the goal node. When a new obstacle is found on the path, "MODIFY-COST()" function is run to update

the costs of related nodes. In addition, these affected nodes are put in the open list.
– "PROCESS-STATE()" function is rerun until a new path is planned,
– Repeat Steps 3 and 4 until reach the goal node.

2.3 LPA*

LPA* algorithm is an incremental and heuristic searching version of A*, which is more suitable for dynamic environments. The main difference between incremental and heuristic searching is that in a heuristic search, the algorithm uses a heuristic function to guide the search towards the goal node, which can reduce the searching space and the computing time, and result in an efficient path planning. However, incremental searching can repetitively use previously explored information to improve search efficiency. When the surrounding environment changes, only the affected nodes need to be recalculated.

Assume $Succ(s)$ is the set of successor nodes of node s; $Pred(s)$ denotes the set of predecessor nodes of node s; $c(s, s')$ represents the path cost from node s to node s'; s_{start} denotes the start node; s_{goal} represents the goal node. Each node in LPA* has two values of $g(s)$ and $rhs(s)$. $rhs(s)$ is the minimal value for the sum of predecessor node's $g(s)$ and the cost from the node s to node s', which is shown in Eq. 3.

$$rhs(s) = \begin{cases} 0, & \mathrm{s} = s_{start} \\ min_{\mathrm{s}' \subseteq \mathrm{pred(s)}}(g(s') + c(s, s')), & \text{otherwise} \end{cases} \tag{3}$$

where $g(s')$ denotes the cost from node s' to the start node s_{start}; $c(s, s')$, named as edge cost function, denotes the cost from node s to node s'. Similar to the A*, LPA* also has a priority list, named Openlist for saving nodes that have been searched. However, LPA* uses $key(s)$ to rank these nodes. The definition of $key(s)$ is shown in Eq. 4.

$$key(s) = \begin{cases} k_1(s) = min(g(s), rhs(s)) + h(s, s_{goal}) \\ k_2(s) = min(g(s)), rhs(s)) \end{cases} \tag{4}$$

LPA* chooses a node with minimal value of $key(s')$ from the surrounding nodes of the current node s' as the next node. Equation 5 shows the comparison of $key(s)$ for nodes s_1 and s_2.

$$key(s_1) \leq key(s_2) \Rightarrow \begin{cases} k_1(s_1) \leq k_2(s_2) \\ k_1(s_1) = k_2(s_2) \quad k_2(s_1) \leq k_2(s_2) \end{cases} \tag{5}$$

The main procedure of LPA* shown in Fig. 3 is as follows.

– Initialization: Set $g(s)$ and $rhs(s)$ of all nodes as infinite; set $rhs(s)$ of the start node S_{start} to 0; calculate the value of its $key(S_{start})$ and put it into the priority list.

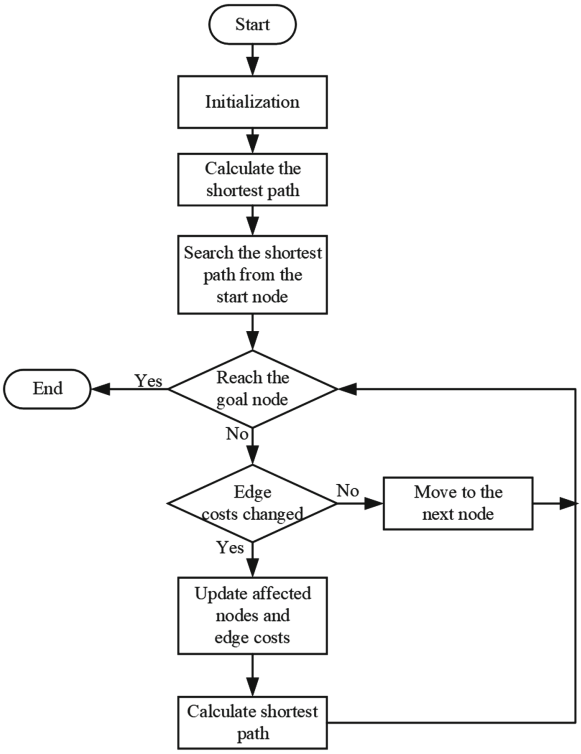


Fig. 3. Main procedure of LPA*

- Searching the shortest path: Calculate $key(s)$ for all surrounding nodes of S_{start}; (1) If $g(s) > rhs(s)$, let $g(s) = rhs(s)$ and recalculate $rhs(s)$ for all the sub nodes. (2) If $g(s) < rhs(s)$, set $g(s) = \infty$, recalculate the $rhs(s)$ values for its sub nodes and all the nodes in the priority list. (3) If $g(s) = rhs(s)$, delete the node from the priority list; repetitively select the node with the smallest $key(s)$ value to expand until $rhs(s_{goal}) = g(s_{goal})$.
- Path generating: The path is planned by moving from the goal node to the parent node with the smallest $rhs(s)$ value until reaching the start node.
- Path updating: Recalculate $rhs(s)$ when the environment is changed; If $g(s) = rhs(s)$, delete the node from the priority list; Otherwise, put the node into the priority list, repeat until $g(s) = rhs(s)$ for the all surrounding nodes of the current node.
- Repeat step 3 to get the new shortest path.

2.4 D* Lite

LPA* finds the shortest path from the start node to the goal node. It still needs to maintain the whole path, even it detects obstacles. D* Lite is an improved

version of the LPA*. It only updates the path between the current node and the goal node when the environment is changed. Thus the searching space is decreased when moving towards the goal node. Similar to D*, D* Lite searches from the goal node to the start node. So the definition of $g(s)$ and $rhs(s)$ is contrary to the LPA*: $g(s)$ denotes the minimal estimation cost from the goal node to current node s, and $rhs(s)$ is the sum of the subsequent node's $g(s)$ and the cost between nodes s and s', which is shown in Eq. 6.

$$rhs(s) = \begin{cases} 0, & \mathrm{s} = s_{goal} \\ min_{\mathrm{s}' \subseteq \mathrm{succ(s)}}(g(s') + c(s, s')), & \text{otherwise} \end{cases} \tag{6}$$

D* Lite also uses $key(s)$ to assess the cost of each node. $key(s)$ consists of two values $[\mathrm{k_1(s)};\mathrm{k_2(s)}]$. The node with the smallest key value will be selected as the next node. The calculation of $k_1(s)$ and $k_2(s)$ is shown in Eq. 7.

$$\begin{cases} k_1(s) = min(g(s), rhs(s)) + h(s, s_{start}) + km \\ k_2(s) = min(g(s), rhs(s)) \end{cases} \tag{7}$$

Different from Eq. 4, there is a parameter km in Eq. 7 to denote the sum of the actual distance that has moved. The km can improve the computing efficiency by preventing the priority list from being recalculated when the edge cost changes. $h(s, s_{start})$ denotes the estimated cost from the current node to the start node, which is shown in Eq. 8.

$$h(s, s_{start}) = \begin{cases} 0, & s = s_{start} \\ c(s, s') + h(s', s_{start}), & \text{otherwise} \end{cases} \tag{8}$$

The main procedure of D* Lite is shown in Fig. 4.

The main procedure of D* Lite is as follows.

- Initialization: Set $g(s)$ and $rhs(s)$ of all nodes as infinite; set rhs value of the goal node s_{goal} as 0; calculate its $key(s_{goal})$ value and put it in the priority list.
- Calculating the shortest path: calculate the $key(s)$ of the surrounding nodes starting from the goal node. If $g(s) > rhs(s)$, then let $g(s) = rhs(s)$. Select the node with the smallest $key(s)$ to expand until $rhs(s_{start}) = g(s_{start})$.
- Path generating: move from the current node to a node with the smallest $rhs(s)$ value. During the moving, set the new node as a start node when it is reached.
- Path updating: Update the value of $key(s)$ for the current node it's status is changed and update the value of affected surrounding nodes.
- Repeat steps 3 and 4 until $s_{start} = s_{goal}$.

3 Heuristic Functions and Simulation Environment

3.1 Heuristic Functions

Heuristic functions are essential for path planning algorithms. Four heuristic functions are widely used in path planning, that is Manhattan distance, diagonal distance, Euclidean distance and squared Euclidean distance. Table 1 shows

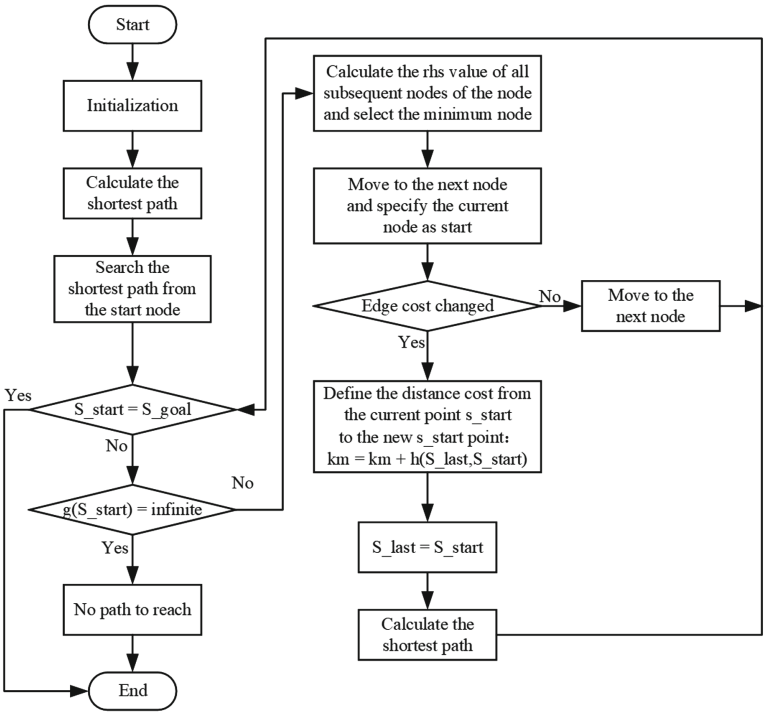


Fig. 4. Main procedure of D* Lite

the definition of these four functions. In the table, $n.x$ and $n.y$ represent the coordinate of the current node; D_1 denotes the minimal cost of moving one step vertically or horizontally, which by default is 1; D_2 denotes the minimal cost of moving one step diagonally, which by default is $\sqrt{2}$; $goal.x$ and $goal.y$ are the coordinates of the goal node respectively.

3.2 Simulation Environment

To evaluate the effect of heuristic functions to path planning algorithms, three path planning environments with different map sizes of 500*500 pixels, 800*800 pixels and 1000*1000 pixels are used to simulate different environments with different complexity (shown in Fig. 5). The black area in the map represents obstacles, and the rest of the white area denotes the free space for path planning.

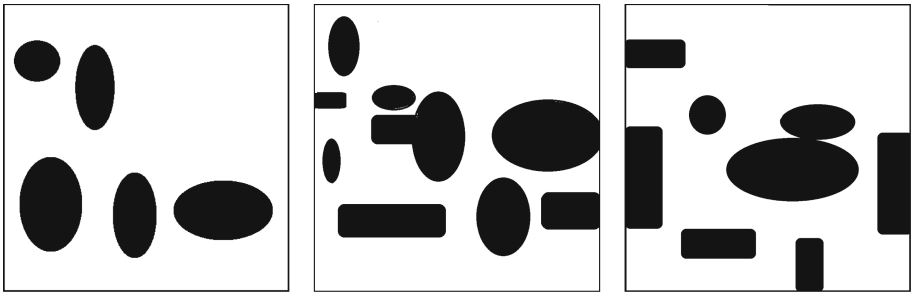
4 Experiments and Analysis

4.1 Performance Analysis of Heuristic Functions

To explore how much a heuristic function can affect the planning efficiency of path planning algorithms, we apply four heuristic functions introduced in

Table 1. Heuristic functions($H(n)$)

Function ($H(n)$)	Formula
Manhattan distance	$H(n) = D_1 \times (abs(n.x - goal.x) + abs(n.y - goal.y))$
Diagonal distance	$d_x = abs(n.x - goal.x)$ $d_y = abs(n.y - goal.y)$ $H(n) = D_1 \times (d_x + d_y) + (D_2 - 2 \times D_1) \times min(d_x, d_y)$
Euclidean distance	$H(n) = D_1 \times \sqrt{(n.x - goal.x)^2 + (n.y - goal.y)^2}$
Squared of Euclidean distance	$H(n) = D_1 \times ((n.x - goal.x)^2 + (n.y - goal.y)^2)$



(a) 500*500 map (b) 800*800 map (c) 1000*1000 map

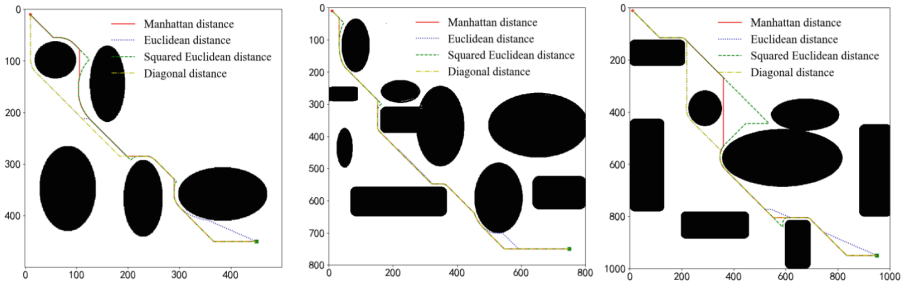
Fig. 5. Three environments with different map size and obstacles

Sect. 3.1 to A*, D*, LPA* and D* Lite, respectively based on three simulation environments in Fig. 5. Path planning results are shown in Figs. 6, 7, 8 and 9. The red star in figures represents the start node, and the green square denotes the goal node.

From Fig. 6(a) we find that the final path of A* is quite different when it uses different heuristic functions. When we increase the map size to 800*800 and 1000*1000, respectively, we find that the paths planned by different heuristic functions are also different (Figs. 6(b) and 6(c)). We obtain similar results in D*, LPA* and D* Lite algorithms (Figs. 7, 8 and 9). Compared to the path results planned by different algorithms, we find that all these path planning algorithms obtain different paths when use different heuristic functions. According to this observation, we conclude that heuristic functions can significantly affect the final path planned by path planning algorithms.

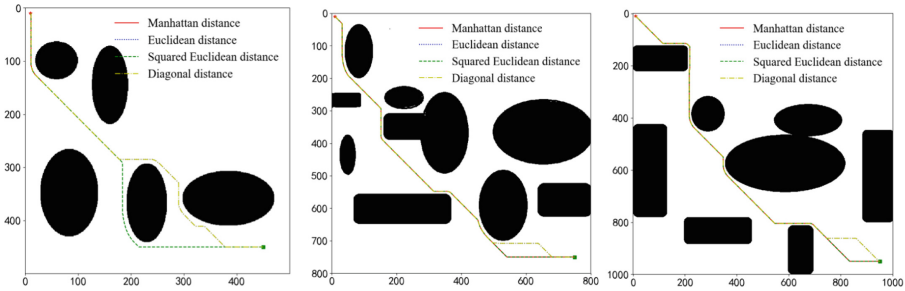
To further explore the effect of heuristic functions to the performance and efficiency of path planning algorithms, we assess their performances using three indicators of the path planning time, number of searched nodes and the path length.

Figure 10 shows the path length, number of searched nodes and the planning time, respectively of the A* based on four different heuristic functions and three different map sizes. We find from Fig. 10(a) that the path length of all heuristic functions is almost the same. When we increase the map size



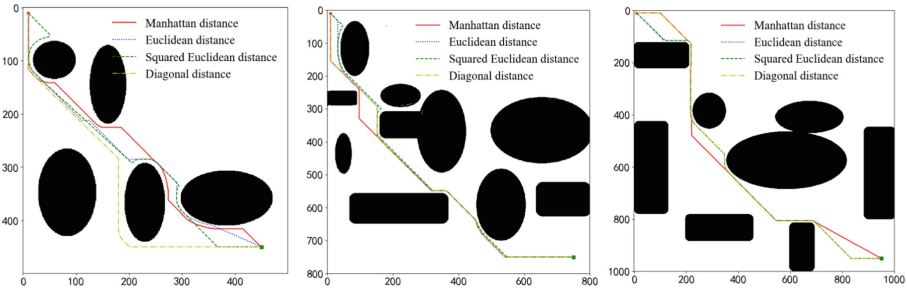
(a) 500*500 map (b) 800*800 map (c) 1000*1000 map

Fig. 6. Path planning results of four heuristic functions using A*



(a) 500*500 map (b) 800*800 map (c) 1000*1000 map

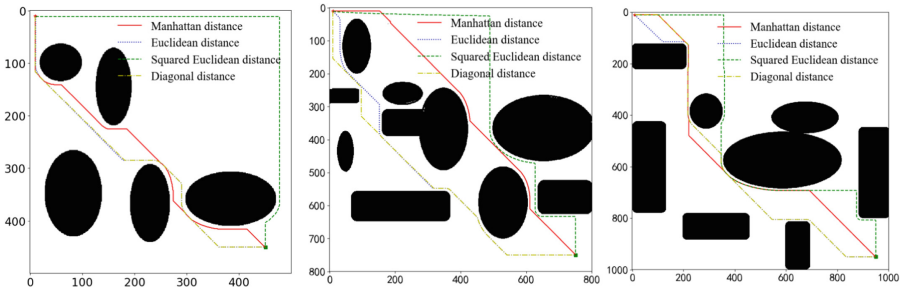
Fig. 7. Path planning results of four heuristic functions using D*



(a) 500*500 map (b) 800*800 map (c) 1000*1000 map

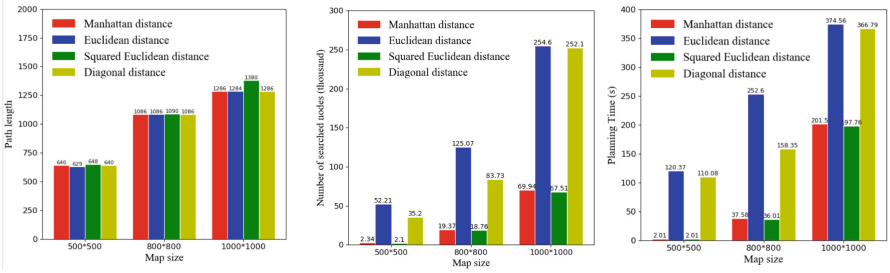
Fig. 8. Path planning results of four heuristic functions using LPA*

from 500*500 to 800*800 and 1000*1000, respectively, the path length is also increased accordingly. Figure 10(b) shows that the number of searched nodes for these four heuristic functions varies significantly. the number of searched nodes of Manhattan distance and Squared Euclidean distance are much smaller than the number of Euclidean distance and Diagonal distance. We find similar



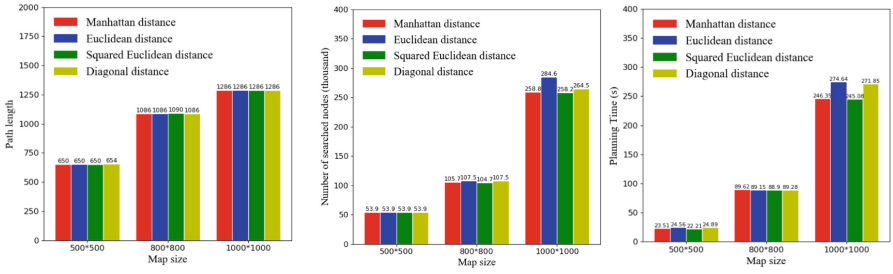
(a) 500*500 map (b) 800*800 map (c) 1000*1000 map

Fig. 9. Path planning results of four heuristic functions using D* Lite



(a) Path length (b) Number of searched nodes (c) Planning time

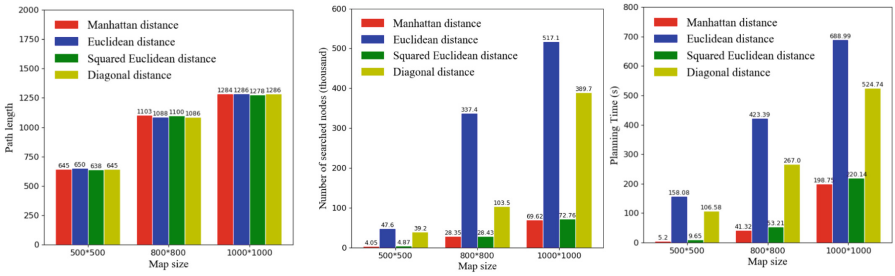
Fig. 10. Performance of four heuristic functions on A*



(a) Path length (b) Number of searched nodes (c) Planning time

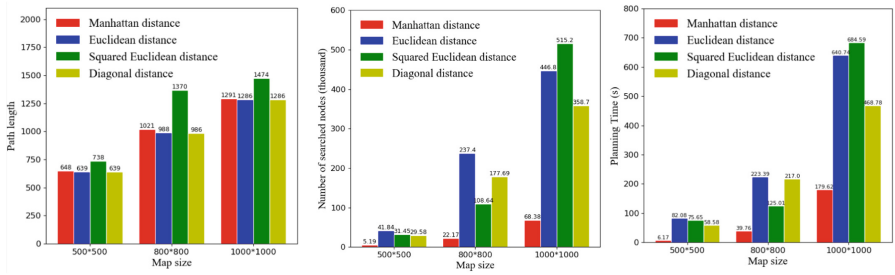
Fig. 11. Performance of four heuristic functions on D*

results when increasing the map size to 800*800 and 1000*1000, respectively. Affected by the searching space explored by different heuristic functions, we find in Fig. 10(c) that Squared Euclidean distance and Manhattan distance also have the shorter planning time than Euclidean distance and Diagonal distance.



(a) Path length (b) Number of searched nodes (c) Planning time

Fig. 12. Performance of four heuristic functions on LPA*



(a) Path length (b) Number of searched nodes (c) Planning time

Fig. 13. Performance of four heuristic functions on D* Lite

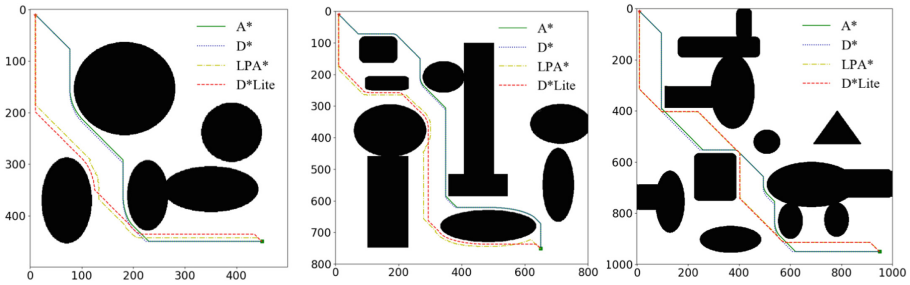
When we apply the four heuristic functions to D* algorithm (Fig. 11), we find that the performance difference between these four heuristic functions is tiny. However, we find in Figs. 12 and 13 that Manhattan distance has a least number of searched nodes and shortest planning time both on LPA* and D* Lite.

Comparing the efficiency of the four heuristic functions based on four path planning algorithms and three maps, we conclude that Manhattan distance is the best heuristic function with higher efficiency and lower computing time. We will use Manhattan distance to explore the efficiency of A*, D*, LPA* and D* Lite in the next section.

4.2 Performance Analysis of Path Planning Algorithms in Static Environment

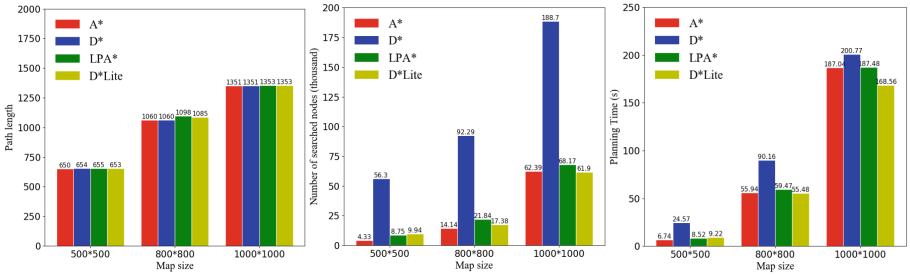
First, we compare the paths planned by A*, D*, LPA* and D* Lite based on Manhattan distance and three static environments with different map sizes. We assume that no dynamic obstacle shows up during the path planning process. Figure 14 shows the paths planned by A*, D*, LPA* and D* Lite, respectively.

Figure 14 shows that paths planned by A* and D* are almost the same. Similar to A* and D*, paths found by LPA* and D* Lite are also very close. To



(a) 500*500 map (b) 800*800 map (c) 1000*1000 map

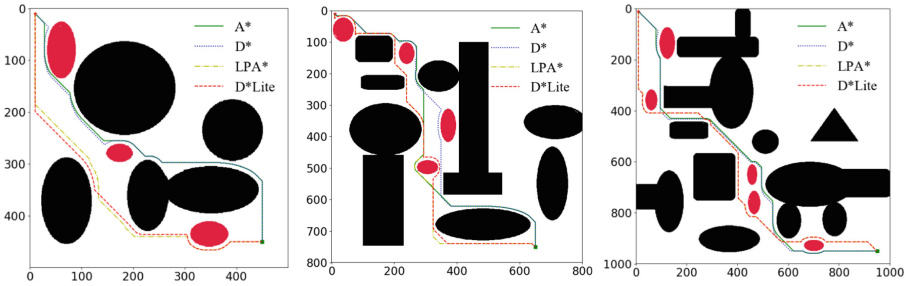
Fig. 14. Path planning results of A*, D*, LPA* and D* Lite using Manhattan distance in static environment



(a) Path length (b) Number of searched nodes (c) Planning time

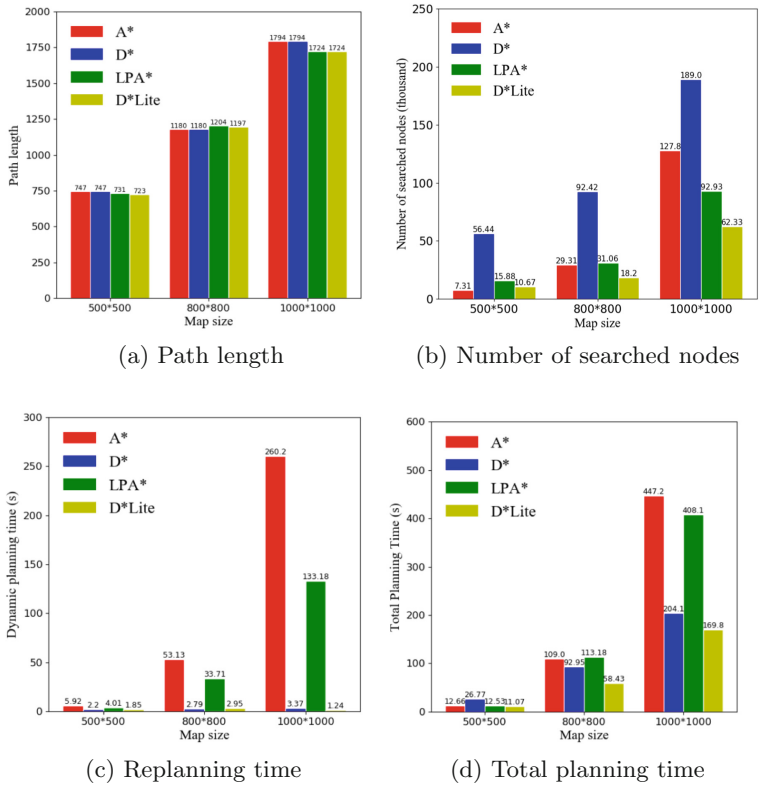
Fig. 15. Path planning performance of A*, D*, LPA* and D* Lite in static environment

compare the path planning efficiency of these four algorithms, we list the path length, number of searched nodes and planning time in Fig. 15. Figure 15(a) shows the length of paths planned by A*, D*, LPA* and D* Lite, respectively based on three different maps. When the map size increases from 500*500 to 800*800 and 1000*1000 respectively, the path length is also increased accordingly. We find that though the path length of four algorithms varies very little on the same map, A* has the shortest path on three maps. Figure 15(b) shows the number of nodes searched by all the four path planning algorithms based on three maps, respectively. We also find that D* has the largest number of searched nodes, while A* has the smallest number of searched nodes. The number of searched nodes for all the four path planning algorithm is increased when the map size is increased. From Fig. 15(c) we observe that A* has the shortest planning time when the map size is 500*500. When the map size increases to 800*800 and 1000*1000, respectively, D* Lite has the shortest planning time. It means that A* has much better efficiency and performance than other algorithms in simple static environments. When the environment complexity and map size are increased, D* Lite has the best planning efficiency and the shortest planning time than all the other path planning algorithms.



(a) 500*500 map (b) 800*800 map (c) 1000*1000 map

Fig. 16. Path planning result of A*, D*, LPA* and D* Lite in dynamic environment



(a) Path length (b) Number of searched nodes

(c) Replanning time (d) Total planning time

Fig. 17. Path planning performance of A*, D*, LPA* and D* Lite in dynamic environment

According to the observation above, we argue that A* is the most efficient algorithm in simple static environment. However, to a complex static environment, D* Lite have better path planning performance than other algorithms.

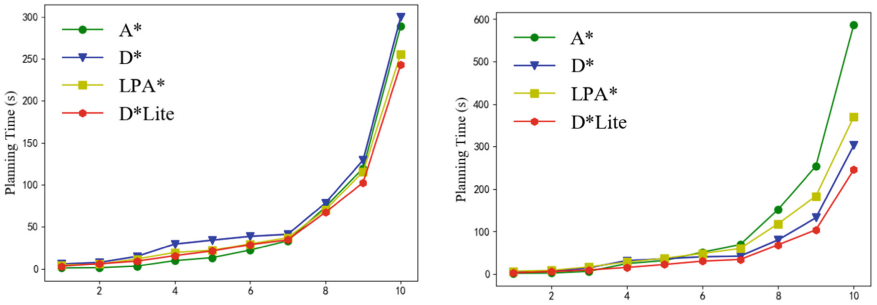
4.3 Performance Analysis of Path Planning Algorithms in Dynamic Environment

To simulate a dynamic environment, we place some obstacles (marked in red) randomly into the environment during the path planning process. Figures 16 and 17 show the path planning results based on three different maps.

Figure 16 shows that each path planning algorithm can find a path from the start node to the goal node and avoid obstacles during planning. From Fig. 17(a) we find that D* Lite has the shortest path in three dynamic environments. Figure 17(b) shows that when the dynamic environment is 500*500, A* has the smallest number of searched nodes. However, D* Lite has the smallest number of searched nodes when the map size of the dynamic environment is increased to 800*800 and 1000*1000, respectively. From Fig. 17(c), we find that the re-planning time of A* is increased quickly when the map size is increased. However, the re-planning time of D* and D* Lite changes very little when the map size is increased. From Fig. 17(d) we know that D* Lite has the shortest path planning time. It means that D* Lite is very suitable for path planning in dynamic environments. However, we also find that when the map size is 500*500, the path planning time of A* is just a little bit larger than D* Lite. So we argue that we can both use A* and D* Lite in simple dynamic environments.

4.4 Performance Analysis of Path Planning Algorithms in a Same Environment with Different Complexity

To compare the four path planning algorithms in the same map but with different complexity and obstacles, we randomly create ten maps with the same size of 500*500 but add random obstacles one by one to increase the complexity of environment respectively. Figure 18 shows the path planning time of all the four algorithms both in static and dynamic environments.



(a) Path planning performance in a static environment

(b) Path planning performance in a dynamic environment

Fig. 18. Path planning performance in a same environment with different complexity

In Fig. 18(a), we add one obstacle to the 500*500 map each time to create ten different environments. We find that when the number of obstacles in the static environment is smaller than 7, A* has the shortest path planning time. However, the path planning time increases sharply when the number of obstacles is increased from 7 to 10. Based on this observation, we confirm that usually, A* has the best performance and efficiency of path planning in static environments. However, when the static environment is very complex, D* Lite algorithm may have a better performance than A*. From Fig. 18(b) we confirm that A* also has an excellent performance in a simple dynamic environment. However, D* Lite is much better than all the other path planning algorithms when the dynamic environment becomes complex.

5 Conclusion

In this paper, we first compared the performance of four widely used heuristic functions based on four path planning algorithms. Experimental results show that Manhattan distance is the best heuristic function for path planning algorithms of A*, D*, LPA* and D* Lite. After that, we analysed and compared the path planning performance of A*, D*, LPA* and D* Lite both in static and dynamic environments with different map sizes and complexity. Results show that A* is much faster and more efficient than other three path planning algorithms in simple static environments. However, if the static environment is very complex, D* Lite may have a better performance than other algorithms. Further experiments reveal that D* Lite is the best algorithm for path planning in dynamic environments. However, in a simple dynamic environment, the path planning performance of A* is almost as good as D* Lite.

The original path planning algorithms do not consider the safety distance between a robot and obstacles when it pass through obstacles. In the future, We will integrate safety distance into D* Lite and apply to an unmanned surface vehicle we have designed for water quality monitoring.

Acknowledgments. This work was partly supported by the AI University Research Centre (AI-URC) through XJTLU Key Programme Special Fund (KSF-P-02 and KSF-A-19), Natural Science Foundation of Suzhou City (SYG201837), Natural Science Foundation of Nantong City (JC2018075) and Nantong University-Nantong Joint Research Center for Intelligent Information Technology (KFKT2017A06).

References

1. Zhang, G.L., Hu, X.M., Chai, J.F., Zhao, L., Yu, T.: Summary of path planning and its application. Mod. Mach. **05**, 85–90 (2011)
2. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968). https://doi.org/10.1109/TSSC.1968.300136
3. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**(1), 271 (1959)

4. Wang, X.S., Wu, Z.X.: Improved Dijkstra shortest path algorithm and its application. Comput. Sci. **39**(05), 223–228 (2012)
5. Jia, Q.X., Chen, G., Sun, H.X., Zheng, S.Q.: Path planning for space manipulator to avoid obstacles based on A* algorithm. J. Mech. Eng. **46**(13), 109–115 (2010)
6. Szczerba, R.J., Galkowski, P., Glicktein, I.S., et al.: Robust algorithm for real-time route planning. IEEE Trans. Aerosp. Electron. Syst. **36**(3), 869–878 (2000)
7. Stentz, A.: The D* algorithm for real-time planning of optimal traverses. CMU Technical report CMU-RI-TR-94-37 (1994)
8. Stentz, A.: The focussed D* algorithm for real-time replanning. In: International Joint Conference on Artificial Intelligence. Morgan Kaufmann Publishers Inc. (1995)
9. Saranya, C., Rao, K.K., Unnikrishnan, M., Brinda, V., Lalithambika, V.R., Dhekane, M.V.: Real time evaluation of grid based path planning algorithms: a comparative study. IFAC Proc. Vol. **47**, 766–772 (2014)
10. Qiu, L., Zhang, H.: Implementation of path finding on 2D game maps. J. Hunan Univ. Technol. **26**(01), 66–69 (2012)
11. Liu, L., Guo, J.M.: Improved D* algorithm for robots in partially-known environment. In: First Intelligent Transport and Artificial Intelligence Conference: Research and Application of Intelligent Transport Technology, pp. 260–264. Guangzhou Publisher of Southern China University of Technology (2006)
12. Xu, K.F.: Research of route planning of mobile robots based on D* Lite. Harbin Institute of Technology (2017)
13. Koenig, S., Likhachev, M., Furcy, D.: Lifelong planning A*. Artif. Intell. **155**(1–2), 93–146 (2004)
14. Zhang, Y.N., Sun, F.C., Shi, X.H.: Route planning of mobile robots based on fast D* Lite algorithm. Data Commun. **01**, 46–51 (2018)
15. Koenig, S., Likhachev, M.: D* Lite. In: AAAI/IAAI, pp. 476-483 (2002)
16. Koenig, S., Likhachev, M.: Fast replanning for navigation in unknown terrain. IEEE Trans. Rob. **21**(3), 354–363 (2005)
17. Zhang, Z.S.: Designing and research of autonomous robots based on visual navigation. Harbin Institute of Technology (2008)
18. Mansour, K.A., Muhammad, A., Ramdane, E.H., et al.: D* Lite based real-time multi-agent path planning in dynamic environments. In: 3rd International Conference on Computational Intelligence. IEEE Computer Society (2011)
19. Gu, Y.Z., Meng, Q.D., Gao, R.Z.: Research of route planning on military robots in street fighting. Inf. Technol. **35**(02), 30–33+36 (2011)
20. Zhou, C.H., Li, S.G.: Research of Dijkstra and A* algorithm. J. Softw. **01**, 102–103 (2007)
21. Chen, S.Q., Teng, Z.J., Hong, Q., Chen, Q.H.: Analysis of Instances of 4 shortest route planning algorithms. Comput. Sci. Technol. **16**, 1030–1032 (2007)