A Modified Heuristic Search Algorithm for Pedestrian Simulation

Gao Peng[1], Xu Ruihua[1] and Zou Xiaolei[1]

[1] Department of Transportation Management Engineering, School of Transportation Engineering, TongJi University, 4800 CaoAn Road, JiaDing campus, Shanghai 201804; email: TJ_JT_GP @yahoo.com.cn

Abstract： A* algorithm is a typical heuristic search algorithm, which is widely used in Game Programming, Artificial Intelligence and robotics. However, AI programmers have found again and again that the standard A* algorithm can be woefully inadequate for achieving the kind of realistic movement.

A modified A* algorithm is proposed in this paper to solve path finding problem for an Agent-based pedestrian simulation system. First, several heuristic functions are introduced and compared. Second, a “Trimming Algorithm” has been developed to achieve smooth straight-line movements. Third, the problem of the output path “clinging to obstacle” of the standard A* algorithm has been solved by imposing penalty on the nodes near obstacles in map preprocessing. With these modifications, we can get more realistic path just like human trail, the feasibility and efficiency have been proved in the simulation tests.

Keywords: pedestrian simulation, path finding, path planning, A* algorithm

1 Introduction

Pathfinding is a core component of many intelligent agent applications, ranging in diversity from commercial computer games to robotics. Characters should move in some goal-directed manner, and the program must be able to identify a good path from an origin to a goal, which both avoids obstacles and is the most efficient way of getting to the destination.

Despite the fact that the industry as a whole is quite aware of the "magic A* algorithm", it remains a perennial problem because implementation is so very game specific. For example, AI programmers have found again and again that the standard A* algorithm can be woefully inadequate for achieving the kind of realistic movement they require in their games.

This paper focuses on several techniques for achieving more realistic looking results from path finding. All of the techniques discussed here were used in the development of an Agent-based pedestrian simulation system, which made for startlingly more realistic and human-like movements for larger number of pedestrians in the simulation. The focal topics presented here include:

1. A*’s ability to vary its behavior based on the heuristic functions can be very useful, we can simulate the variety of pedestrian’s route choice by designing diverse heuristics. In chapter 2, several heuristic functions are introduced and compared.

2. A “Trimming Algorithm” is presented in chapter 3, which could reduce the “zigzag effect”

and achieve smooth straight-line movement.

3. The problem of the output path “clinging to obstacle” of the standard A* algorithm has been solved in chapter 4 by imposing penalty on nodes near obstacles in map preprocessing.

2 Heuristic

2.1 The Meaning of heuristic

Our application of interest is grid-based path finding. An agent has to traverse through a two-dimensional map. Simplifying the search area, the map is divided into a square grid, and each square can be identified by a unique ID.

Consider Figure 1: Suppose an agent want to travel from location A (green square) to location B (red square), and there is a wall (blue squares) between them.

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	A	17	18	19	B	21
22	23	24	25	26	27	28
29	30	31	30	33	34	35

Fig.1 Grid map

Within one step, the agent is possible move to one of the 8 adjacent squares which is walk-able. The cost of straight movement can be defined as *“Cost_Straight”* (take 100 in this paper), and the cost of diagonal movement can be defined as *“Cost_Diagonal” (take 141 in this paper). The reason of taking these values is that the “Cost_Diagonal” is* $\sqrt{2}$ times more than “Cost_Straight”, approximately 1.41 times.

Heuristic search means calculating the cost of every location in the search space, and choosing the lowest one for each step, so tremendous unnecessary searching can be avoided and high efficiency can be obtained.

The key to determining which squares to use when figuring out the path is the following equation:

$$F(n) = G(n) + H(n) \quad (1)$$

Where:

G = the cost to move from the starting point A to a given square on the grid, following the path generated to get there.

H = the estimated cost to move from that given square on the grid to the final destination, point B.

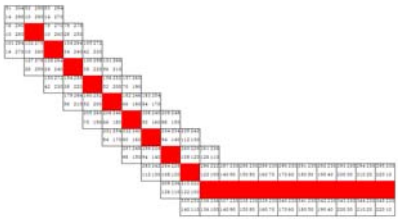
H is often referred to as the heuristic, because we really don't know the actual distance until we find the path, all kinds of stuff can be in the way (walls, water, etc.).

2.2 Heuristic functions

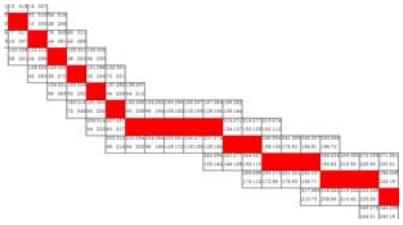
The absolute difference between the row of square n and the row of destination can be defined as "*diffRow*", and the corresponding column difference can be defined as "*diffCol*". So the standard heuristic "Manhattan distance" can be formulated as:

$$H(n) = (diffCol + diffRow) * Cost\_Straight \quad (2)$$

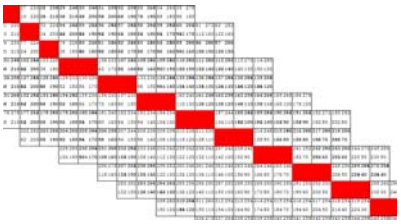
With this simple and low CUP consumption heuristic, we often get bended path in a map which has no obstacle (Fig. 2 a). One of factors contributed to the problem is that the estimated costs are too high, if H(n) is always lower than (or equal to) the cost of moving from square n to the goal, then A* is guaranteed to find a shortest path. The lower H(n) is, the more node A* expands, making it slower. At one extreme, if H(n) is 0, then only G(n) plays a role, and A* turns into Dijkstra's algorithm, which is guaranteed to find a shortest path; at the other extreme, if H(n) is very high relative to G(n), then only H(n) plays a role, and A* turns into BFS.



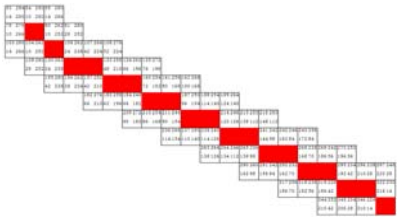
(a) Manhattan distance



(b) Euclidean



(c)Hex



(d)Hex2

Fig.2 various output paths by adopting different heuristic

Euclidean distance is the "straight line distance", which can get a better path (Fig. 2 b) with a higher calculation price:

$$H(n)=Cost\_Straight\times\sqrt{diffRow^2+diffCol^2}$$

$$Cost\_Straight\times\sqrt{diffRow^2+diffCol^2} \quad (3)$$

If you want to get rid of the "polyline" in the output path of Manhattan distance, some modifications is necessary. One of the best solutions is "Hex Distance":

$$H(n) = (Max\{0, diffRow - diffCol / 2 - Correction\} + diffCol) * Cost\_Straight \quad (4)$$

where：

$$Correction = \begin{cases} colTarget \% 2, \text{ when } rowTarget < rowStart \text{ and } diffCol \text{ is odd} \\ colStart \% 2, \text{ when } rowTarget > rowStart \text{ and } diffCol \text{ is odd} \\ 0, \text{ when } diffCol \text{ is even} \end{cases}$$

"*rowStart*" stands for the row index of square n, and "*colStart*" stands for its column index.

"*rowTarget*" stands for the row index of destination, and "*colTarget*" stands for its column index.

The output path by adopting "Hex distance" is more realistic than that of "Euclidean distance" with lower calculation price, but make A* expend more nodes. We can obtain "Hex2 distance", which able to make A* expend less nodes and find similar path, by raise estimated cost:

$$H(n) = (Max\{0, diffRow - diffCol / 2 - Correction\} + diffCol) * Cost\_Diagonal \quad (5)$$

The meanings of variables are consistent with equation (4).

It's worthwhile to note that although we can find similar path by Hex2, we couldn't simply regard Hex2 as a better heuristic than Hex in any case. If the map size is comparatively small and the map structure is somewhat simple, A* might find similar paths, conversely may not.

Finally, there are 7 heuristics, respectively named "Hex", "Hex2", "Hex3", "Manhattan", "Euclidean", "Octile" and "Octile_UniCost", which can be used to develop different kinds of path planning algorithms. So we can simulate various pedestrians' preferences for route choices.

You can see 7 kinds of paths by 7 search algorithms from Fig. 3 (respectively in red, orange, yellow, green, cyan, blue and violet, possible overlapped).

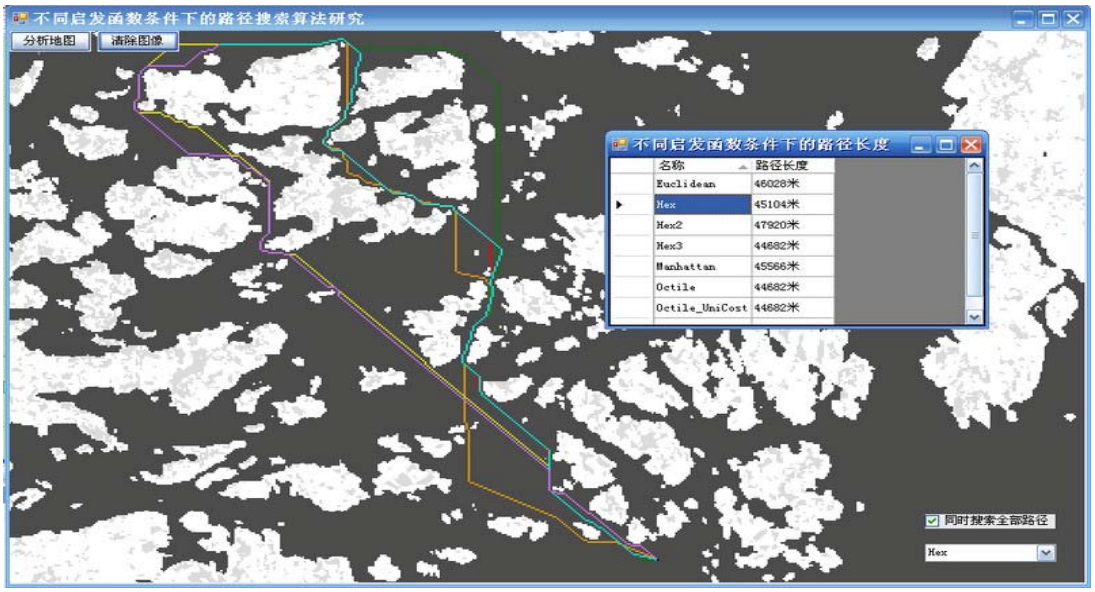


Fig.3 Seven kinds of paths by 7 search algorithms

3 Achieving smooth straight-line movement

As mentioned above, although we can simulate various pedestrians' preferences for route choices by adopting diverse heuristics, some heuristics may lead to an unfortunate "zigzag" effect in Pathfinding (shows in Fig. 3). One of the reasons is the output paths may contain redundant nodes, which must be found out and removed. This paper presents post processing solutions for smoothing the path, whose core component is a recursive algorithm named "Walkable". The fundamental principle is to check every 3 waypoints along the path, if there are no obstacles between the head and the rear, the middle waypoint is redundant. The pseudo code for the "trimming algorithm" as follows:

To judge whether there are any obstacles between the point "P1" and the point "P2":

Step 1: calculate the straight-line distance "d" between P1and P2;

Step 2: if d < tile width, then return and assert "there is no obstacle";

Else turn to Step 3;

Step 3: if the middle point of P1 and P2（named "Center"） is a blocked tile, then return and assert "there are some obstacles";

Else turn to Step 4;

Step 4: judge recursively whether there are any obstacles between P1 and Center, Center and P2, if any, then return and assert "there are some obstacles";

Else return and assert "there is no obstacle";

After the post processing trimming, the output path is very short and include no redundant nodes (shows in Fig. 4 and Fig. 5), so it can serve as Macro-navigation for virtual pedestrians in the simulation.

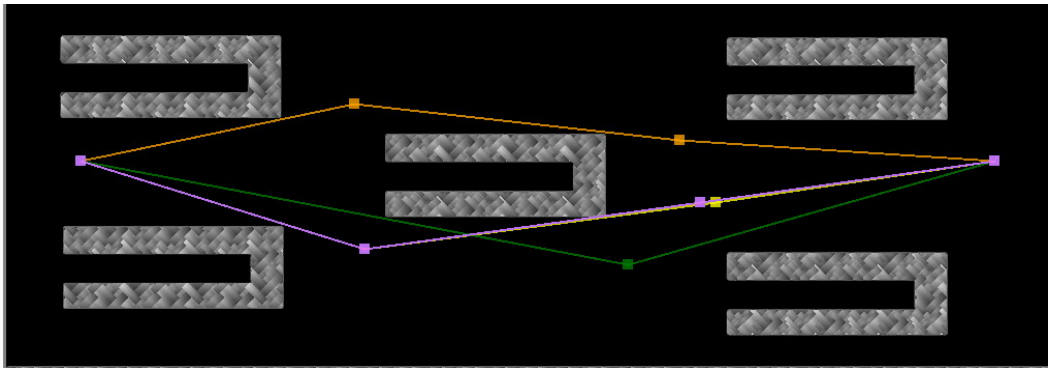
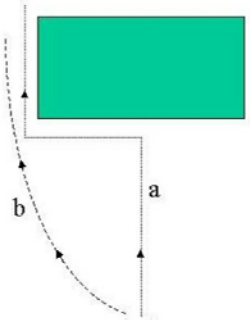


Fig.4 Paths after Trimming

4 Toward more realistic Pathfinding

Most pedestrians are inclined to walk on "the shortest paths" in our real life, and seldom love getting too close to obstacles. While unfortunately, output paths of standard A* algorithm seems "clinging to obstacles" too much, and nobody would move in this way, as Fig. 5 and Fig. 6(a) showing.

With map preprocessing, the problem has been solved by imposing penalty on nodes near obstacles. The map preprocessing



algorithm calculated the distance between every node of the map to its nearest obstacle. Then we can add a penalty function to equation (1):

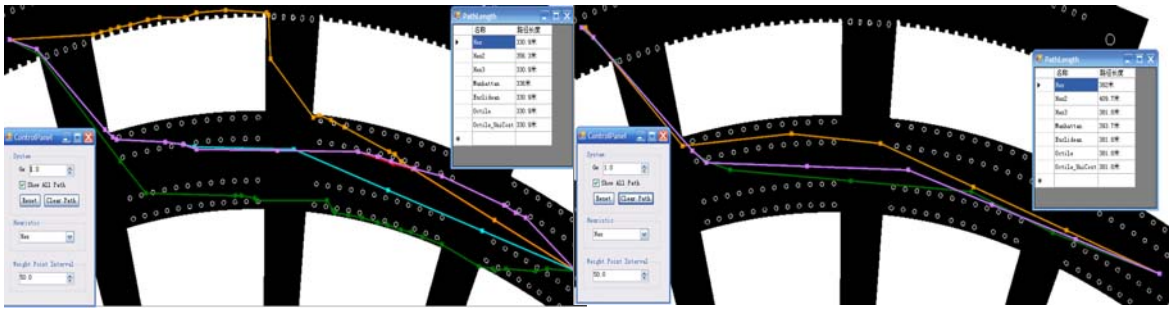
$$F(n) = G(n) + H(n) + P(d) \quad (6)$$

Where:

d= the distance between node n to its nearest obstacle. *Fig. 5 Unrealistic path*

Fig. 6 shows a part of Inner Concourse of Athens Olympics 2004. By comparing (a) and (b) of Fig. 6, it's obvious that the modified algorithm, which achieved "line of sight" move and keeping certain distance away from obstacles, has made the difference.

The fundamental principle of map preprocessing is: to calculate the distance between every node and its nearest obstacle, the pseudo code as follows:



(a) Output paths by standard A* algorithm (b) Output paths by modified algorithm

Fig.6 Paths before and after disposing

Step 1: let R=1, R is the index of "square layer", which is composed of 8*R nodes, around the given node;

Step 2: if "R< map width or map height" then turn to Step 3;

Else turn to Step 4;

Step 3: analyze every node of the "square layer":

If come across a blocked node, turn to Step 4;

Else let R=R+1, turn to Step 2;

Step 4: obtained the distance between given node and its nearest obstacle is R, return.

5 Conclusion

A* is a famous algorithm, widely used in commercial game and AI. To some extent, we should not merely regard A* as a concrete algorithm, but a thought or a framework instead. Based on the framework, implementation can be various. So we can develop a lot of new search algorithms, which could meet some specific needs.

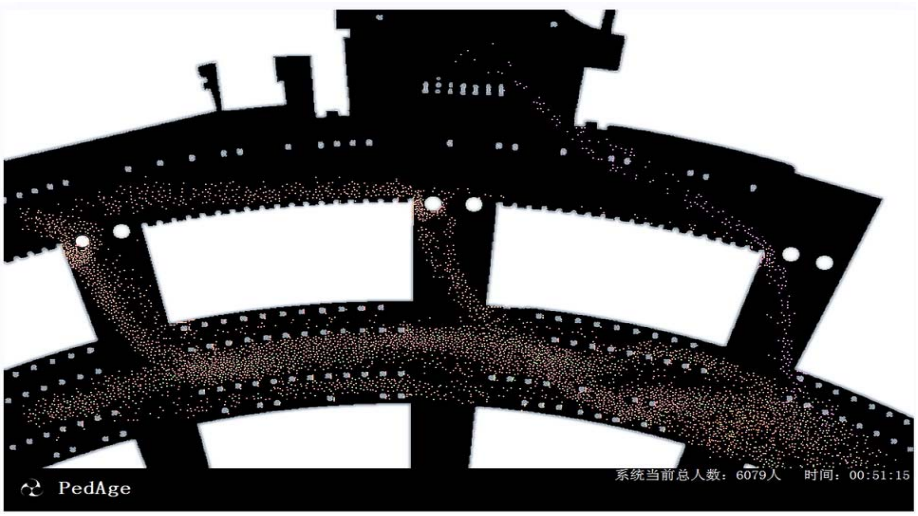


Fig. 7 Scenario of Athens Olympics 2004

A modified A* algorithm is proposed in this paper to solve path finding problem for an Agent-based pedestrian simulation system. Fig. 7 depicts a scenario of Athens Olympics 2004. The effect implies that we can achieve more realistic movements with the modifications proposed in this paper. The point to emphasize here is that the output paths by search algorithm not serve directly as the factual trail, but a kind of macro-navigation. The movement of every pedestrian depends on both macro-navigation and crowd dynamics.

6 Reference

[1] Dirk Helbing , Martin Treiber . Jams, Waves and Clusters [J]. Science , 1998:Vol. 282 Pp2001 - 2003

[2] Dirk Helbing . Modeling the evolution of human trail systems [J]. Nature, 1997: Vol. 388 Pp 47-50

[3] G. Keith Still. Crowd Dynamics [D]. London: University of Warwick, BSc Physical Sciences (Robert Gordon's Institute of Technology, Aberdeen 1981), 2000

[4] Anthony Stentz．The Focused D * Algorithm for Real-Time Re-planning[A]．In：Proceedings of the international Joint conference on A artificial Intelligence[C]． 1995

[5] S.P. Hoogendoorn & W.Daamen, Design assessment of Lisbon transfer stations using microscopic pedestrian simulation, [J].Computers in Rilways , 2004, IX ,135-147

[6] Winnie Daamen , Modeling Passenger Flows in Public Transport Facilities [D]. Delft: Delft University of Technology, 2004

文件名:	A Modified Heuristic Search Algorithm for Pedestrian Simulation.doc
目录:	C:\Users\silver\Documents
模板:	C:\Users\silver\AppData\Roaming\Microsoft\Templates\Normal.dotm
标题:	
主题:	
作者:	高鹏
关键词:	
备注:	
创建日期:	2007/3/10 8:45:00
修订号:	158
上次保存日期:	2007/7/4 22:35:00
上次保存者:	高鹏
编辑时间总计:	3,984 分钟
上次打印时间:	2009/3/2 12:49:00
打印最终结果	
页数:	7
字数:	1,932 (约)
字符数:	11,019 (约)