A optimization of A* algorithm to make it close to human pathfinding behavior

ZhenGuo Zhao[1, a], RunTao Liu[2,b]

[1] College of Applied Sciences ,Harbin University of Science and Technology, Harbin 150080, China

[2] Institute of Information and Scientific Computing Technology, Harbin University of Science and Technology, Harbin 150080, China

[a] zhaozhenguoxyz@163.com, [b] johnlee@comp.polyu.edu.hk

Keywords: A* algorithm, Pathfinding optimization, Space vector operations, Common sense illation

Abstract. A* algorithm is considered to be the optimal heuristic search algorithm over time. But the biggest drawback is the consumption of resources as the growth risk of the processing scale is a exponential level. This article based on common sense reasoning of artificial intelligence and the vector calculation of mathematics put forward three strategies to optimize the number of intermediate nodes and the calculation times of heuristic function. These strategies make the computer in the process of pathfinding closer to human thinking. By comparison with the experimental data with the traditional algorithm, the result show that runtime is generally reduced by 40%~70% and count of intermediate node is reduced by 30%~60%. And exponential growth risk is also effectively reduced.

Introduction

A Star algorithm is a heuristic search algorithm, which widely used in the field of pathfinding and traveling in graph, especially in the game development and GIS. Its accessibility and efficiency has been widely recognized[1].It combines the pieces of information that Dijkstra's algorithm uses which expected choice is close to the starting point and information that Best-First-Search uses which choice is close to the goal[2].But with the enlargement of the search space, because of the need to implement a breadth-first search, the calculation times of heuristic function ,which affects the time complexity of algorithm ,is greatly increased and finally tending to exponential growth[3].Therefore, the optimization of breadth-first can reduce the intermediate node number of visits and the number of valuation function calculation and this will greatly reduce resource consumption and time consumption of the algorithm. This paper based on common sense illation of artificial intelligence and mathematics algebra calculation presents an efficient optimization algorithm, and will give a satisfactory experimental results in the algorithm comparison.

Introduction of A * algorithm

Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute first described the algorithm in 1968[1].It's like Dijkstra's algorithm in that it can be used to find a shortest path. It's like Greedy Best-First-Search in that it can use a heuristic to guide itself. Eq. 1 is widely used as a heuristic function ,where g_n is the exact cost of the path from the starting point to any vertex n, h_n is the heuristic estimated cost from vertex n to the goal. When $h_n \le h_n*$,where h_n* is the exact cost from vertex n to the goal, an accessible path will be found[4].In this paper, Manhattan distance is selected for calculating h_n and is has been proved it is accessible[5].The main steps of A * algorithm description[6]:

$$f_n = g_n + h_n \tag{1}$$

The main steps of A * algorithm description[6]:

1. Start node is pushed into the OPEN List, which is a priority queue;

2. Pick the first node from the open list and keep it as the current node, which is the lowest rank by f_n from OPEN and is not the GOAL;

3. Get the neighboring nodes of this current node, which are not obstacle types and not in CLOSED List. Calculate f_n and Store it in the neighbor node object. Set neighbor's parent to current node and add neighbor to OPEN List;

4. Remove current node from OPENList and add it to CLOSED List, then go to step 2 .

The introduction of A * optimization algorithm

Calculation the f_n of the current's all neighbor nodes result in the growth of the processing scale is a exponential level. In this paper ,4-neighborhood is used to pick current's neighbors. According to the human experience，which called common sense illation of artificial intelligence, not all the direction of the current's neighbor needs to be accessed, and the neighbor between Start and Goal may be accessed, and the neighbor, which is a key position from Start to Goal, such as corner or the only neighbor node, must be accessed. So some unnecessary neighbors can be pruned and the number of intermediate nodes and the calculation times of heuristic function are reduced. But in order to record the abandoned intermediate nodes, extra storage space is necessary. Based on space vector operations of mathematics algebra calculation, computer can have a sense of direction. And two list called Negative Direction List and Reverse Move List is used to store the abandoned intermediate nodes. Therefore, our optimization mainly includes three strategies:

Strategy 1: Exploration in the direction from Start node to Goal node.

Let collection NS be collection of all the neighbor nodes of current node S and collection VNS be collection of vectors in the composition of nodes S and every neighbor node. While the dot product between every vector of VNS, which called $\overrightarrow{VNS_i}\ (i = 0,1,2,3)$,and $\overrightarrow{SG}$ is greater than zero or the count of NS is one, the neighbor node which related with $\overrightarrow{VNS_i}$ is in the goal direction ,e.g. In Fig. 1, S'will be added into Negative Direction List which contains not in the goal direction and S" will be processed later.

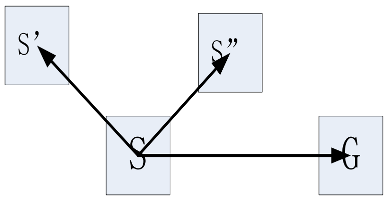
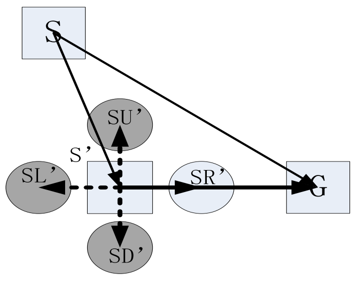


Fig. 1 Decision Node which between S and G

Strategy 2: Move toward the Goal node.

Some nodes, which not in the goal direction, are pruned by Strategy 1. However ,not all the nodes in the goal direction must be calculated the heuristic function, e.g. in Fig. 3，although SU',SL' ,SD' and SR' are both in the goal direction, SD' and SR' are expected based on the common sense illation of artificial intelligence, which showed.

Let collection NS' be collection of all the neighbor nodes of current node S', which 4-neighborhood nodes are in the goal direction, and collection VNS' be collection of vectors in the composition of nodes Sand every neighbor node. While the dot product between every vector of VNS', which called $\overrightarrow{VNS'_i}\ (i = 0,1,2,3)$,and $\overrightarrow{SG'}$ is greater than zero or the count of NS' is one, the neighbor node which related with $\overrightarrow{VNS'_i}$ is in the expected moving direction.



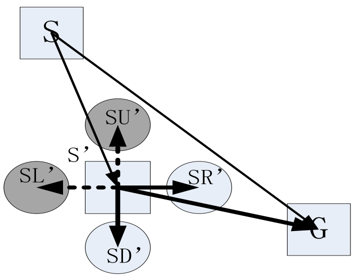


Fig. 2 Node in horz moving target direction   Fig. 3 Node in not horz moving target direction

Deduction 1 :

According to Strategy 2,at least 50 fifty percent of neighborhood nodes are reduced when calculation nodes heuristic function.

Proof : While G in S' horizontal or vertical position, three times of calculation node's heuristic function are reduced, e.g.SU', SL' and SD' (75% neighborhood nodes)will be added into Reverse Move List which contains not in the expected moving direction and SR' will be added into OPEN List, which showed in Figure 2 ;Other situations two times of calculation node's heuristic function are reduced, e.g.SL' and SD' (50% neighborhood nodes)will be added into Reverse Move List and SD' and SR' will be added into OPEN List, which showed in Fig 3.

Strategy 3: When encounter a dead-end, return the nearest position which in

Deduction 2 :

According to Strategy 1 and Strategy 2， the worst case is that encountering a dead-end, if there is a way to goal, we need to return the recent node position which in Negative Direction List and Reverse Move List and continue to search. Finally, the shortest path will be found. In this case, although a few nodes are pruned, it is better than traditional A* algorithm.

Proof: Negative Direction List and Reverse Move List contains the nodes which has been pruned ,these nodes and those nodes which has been push in OPEN List are the same as the OPEN List which used in traditional A* algorithm. Therefore ,optimized A* algorithm has the same nature with traditional A* algorithm ,such as accessibility and term inability. So the shortest path will be found finally.

The pseudo-code of optimized A* algorithm is outlined in Algorithm A Star Optimize.

```
Algorithm   A Star Optimize (Start, Goal)
Input:      A Grid MAP, Start node and Goal node
Output :    The shortest path from Start to Goal
1：         CLOSED List, Right Out List, Reverse Move List:=empty set, OPEN List := {start}
2：         While OPEN List is not empty set
3：             current := OPEN List's first node ,which having the lowest heuristic function value
4：             if current = Goal then
5：                 return reconstruct_ path(current)
6：             Start To Goal Vector := the vector from start to goal
7：             Current To Goal Vector := the vector from current to goal
8：             for each neighbor which not in CLOSED List of current
9：               start To Neighbor Vector :=the vector from start to neighbor
10：                Current To Neighbor Vector :=the vector from current to neighbor
11：              if (neighbor_ nodes. length>1  and  Dot(start To Neighbor Vector, start To Goal Vector)<0)
12：                  add neighbor to Right Out List
13：              else
14：                  if(Dot(current To Neighbor Vector, current To Goal Vector)>0 or neighbors. Count==1)
15：                      Calculate neighbor's heuristic function
16：                      neighbor. parent := current
17：                      add neighbor to OPEN List
18：                  else
```

```
19:                     add neighbor to Reverse Move List
20:               remove current from OPEN List
21:               add current to CLOSED List
22:               if(OPEN List is empty)
23:                   if (current is not Goal)
24:                       Add  Right Out List and Reverse Move List to OPEN List
25:          Reconstruct reverse path from goal to start.
```

Complexity Analysis

The time complexity of A* algorithm depends on the heuristic function. In the worst case,it is exponential. And when the search space is a tree, it is polynomial. The time complexity of optimized A* algorithm is the same with traditional A* algorithm, because it lower down the growth rate by reducing the times of calculation node's heuristic function. Therefore, optimized A* algorithm lower down the factor of exponential or polynomial formula and does not change the time complexity.

Experimental evaluation

We refer to the experimental analysis section of DEC - A * [7] . we compare two measures between A* algorithmand optimized A* algorithm.One is runtime ,another one is total times of accessing nodes in grid map. We create a platform which could generate a grid map with two parameters:

Grid size: the size of maps,e.g.20*20

Probability of obstacle (Prob): during generating grid cells, a random number is given. If it is less than the probability,the grid cell is a obstacle

One experimental result is showed in Figure 4.The number of red node which is accessed by optimized A* algorithm is 88 less than the number of green node which is accessed by A* algorithm.The runtime of optimized A* algorithm is 5.194 ms and A* algorithm is 11.714 ms. Runtime is reduced by 55.6%.So optimized A* algorithm is effective. More sets of test data are given in Table 1 and Table 2.In each experiment, the start is at center location on the map and the goal is at lower right corner. All experiments are performed on an Intel(R) Core(TM) 2 Duo CPU and 2GB memory.

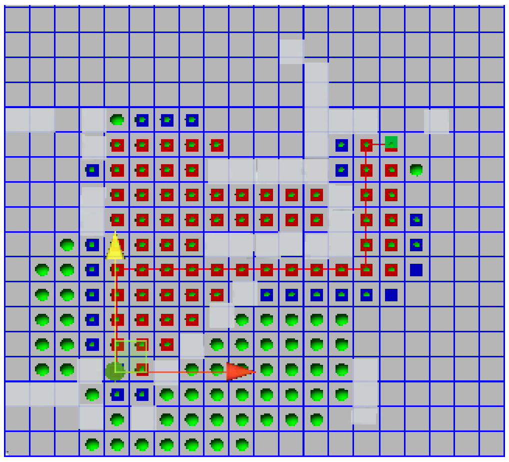


Fig. 4 One experimental result while size=20

Table 1 The results of runtime comparison

Prob / Size	0		0.1		0.2		0.3	
	optimized A*	A*	optimized A*	A*	optimized A*	A*	optimized A*	A*
20	0.0081	0.0137	0.0046	0.0082	0.0032	0.0078	0.0027	0.0069
30	0.0311	0.0736	0.016	0.0487	0.0108	0.0487	0.0039	0.008
40	0.0817	0.1689	0.0537	0.1184	0.0264	0.0721	0.0164	0.0414
50	0.2041	0.3959	0.1101	0.2723	0.0319	0.1683	0.0503	0.1083
60	0.4116	0.7861	0.2875	0.6163	0.1007	0.3911	0.0454	0.187
70	0.7876	1.5152	0.5806	1.5112	0.2613	0.7233	0.0639	0.4918
80	1.3394	2.688	0.7482	1.962	0.2987	1.2739	0.0394	0.4587
90	2.2313	4.3637	1.3392	3.229	0.3504	1.9554	0.4571	1.339
100	3.444	6.7301	1.947	4.9354	1.0407	3.2114	0.2007	1.7105
120	7.0134	14.1236	4.0787	10.6219	0.8211	6.4374	0.2963	3.6539
140	13.319	26.69	8.0425	20.0936	2.0423	13.3212	0.2232	6.7863

Table 2 The results of accessed node’s count comparison

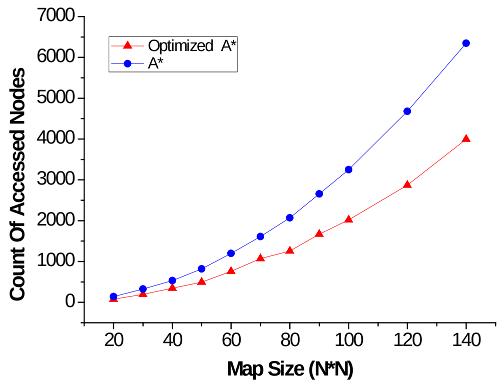
Prob / Size	0		0.1		0.2		0.3	
	optimized A*	A*	optimized A*	A*	optimized A*	A*	optimized A*	A*
20	120	180	77	139	64	118	61	111
30	255	379	195	324	161	270	66	176
40	419	610	348	532	257	426	205	341
50	649	948	493	818	289	677	296	546
60	929	1349	760	1198	496	991	310	700
70	1259	1822	1073	1609	770	1307	353	1109
80	1639	2374	1256	2070	842	1741	314	1101
90	2069	2997	1671	2657	902	2127	813	1773
100	2549	3676	2020	3249	1520	2703	571	2034
120	3659	5286	2872	4678	1375	3788	863	2918
140	4969	7164	3995	6347	2101	5309	712	3925

Table 3 show percentage of relative optimization which calculated by (parameter of A*－parameter of optimized A*)/parameter of A*.Parameter time acts as runtime and Parameter count acts as accessed node’s count. The result show that runtime is generally reduced by 40%~70% and count of accessed node is reduced by 30%~60%.

Table 3 Percentage of relative optimization

Prob / Size	0		0.1		0.2		0.3	
	time	count	time	count	time	count	time	count
20	40.9%	33.30%	43.9%	44.60%	59.0%	45.80%	60.9%	45.00%
30	57.7%	32.70%	67.1%	39.80%	77.8%	40.40%	51.3%	62.50%
40	51.6%	31.30%	54.6%	34.60%	63.4%	39.70%	60.4%	39.90%
50	48.4%	31.50%	59.6%	39.70%	81.0%	57.30%	53.6%	45.80%
60	47.6%	31.10%	53.4%	36.60%	74.3%	49.90%	75.7%	55.70%
70	48.0%	30.90%	61.6%	33.30%	63.9%	41.10%	87.0%	68.20%
80	50.2%	31.00%	61.9%	39.30%	76.6%	51.60%	91.4%	71.50%
90	48.9%	31.00%	58.5%	37.10%	82.1%	57.60%	65.9%	54.10%
100	48.8%	30.70%	60.6%	37.80%	67.6%	43.80%	88.3%	71.90%
120	50.3%	30.80%	61.6%	38.60%	87.2%	63.70%	91.9%	70.40%
140	50.1%	30.60%	60.0%	37.10%	84.7%	60.40%	96.7%	81.90%

We analysis the case that probability of obstacle equal 0.1, both runtime and count of accessed nodes which generated by A* are greatly increased and finally tending to exponential growth, but optimized A* are increased slowly and the velocity of increasing is below exponential level, which showed in Fig 5 and Fig 6.



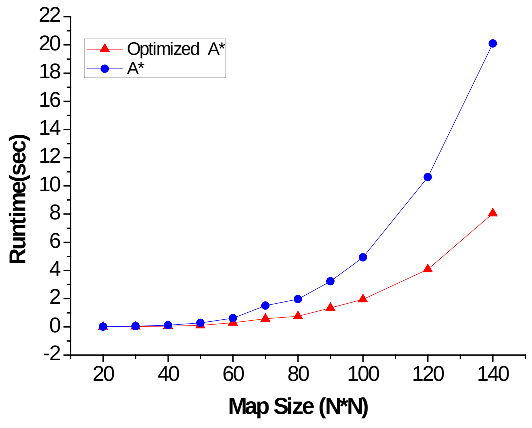


Fig 5 Runtime while prop=0.1 Fig 6 Count of accessed nodes while prop=0.1

The strategies that prune intermediate node and experimental results prove the ability of reducing the risk of exponential growth and the efficiency and of optimized A*.

Summary

A* algorithm is widely used in the field of network game and GIS, because it is optimal about time in theory. The less time algorithm consumed, the higher quality improved for customer experience(CEIP).This paper based on common sense illation of artificial intelligence and space vector operations of mathematics algebra calculation, put forward three optimization strategies. According to three optimization strategies and simulation experiment results, optimized A* is proved that it will reduce the consumption of time and the number of accessed nodes. So, optimized A* is a effectively improved algorithm.

References

[1] P. E. Hart, N. J.Nilsson, B. Raphael, A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 4(2): 100–107 (1968).

[2] S. Anthony : Optimal and Efficient Path Planning for Partially-Known Environments[A]. The Robotics Institute; Cmegie Mellon University; Pittsburgh, PA 15213

[3] M. Nosrati, R. Karimi, and H. A. Hasanvand. Investigation of the * (star)search algorithms: Characteristics, methods and approaches. World Applied Programming 2, 4 (2012), 251-256

[4] Junfeng Yao, Chao Lin, Xiaobiao Xie, Andy JuAn Wang, and Chih-Cheng Hung. 2010. Path Planning for Virtual Human Motion Using Improved A* Star Algorithm. In Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations (ITNG '10). IEEE Computer Society, Washington, DC, USA, 1154-1158.

[5] Miao Wang; Hanyu Lu, "Research on Algorithm of Intelligent 3D Path Finding in Game Development," Industrial Control and Electronics Engineering (ICICEE), 2012 International Conference on , vol., no., pp.1738,1742, 23-25 Aug. 2012.

[6] A. Patel.Introduction to A* From Amit’s Thoughts on Pathfinding Retrieved from http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html.

[7] Mohamad El Falou, Maroua Bouzid, and Abdel Illah Mouaddib.. DEC-A*: A Decentralized Multiagent Pathfinding Algorithm. In Proceedings of the 2012 IEEE 24th International

Conference on Tools with Artificial Intelligence - Volume 01 (ICTAI '12), Vol. 1(2012). IEEE Computer Society, Washington, DC, USA, 516-523.