TELKOMNIKA Indonesian Journal of Electrical Engineering
Vol.12, No.7, July 2014, pp. 5537 ~ 5545
DOI: 10.11591/telkomnika.v12i7.4292

Performance Analysis of Pathfinding Algorithms Based on Map Distribution

Yan Li*, Zhenhua Zhou, Wenju Zhao
Key Lab. of Machine Learning and Computational Intelligenceof Hebei University
College of Mathematics and Computer Science, Hebei university, Baoding, 18730272725
*Corresponding author, e-mail: ly@hbu.cn

Abstract

The distribution information of game maps is highly relevant to the execution efficiency of path searching and the degree of game difficulty. This paper analyzes the relationship between the pathfinding performance and the obstacles distribution in maps from two aspects, pathfinding algorithm design and game's map design respectively. A hierarchical pathfinding algorithm called CDHPA is proposed by incorporating the obstacle distribution in traditional HPA* algorithm. It is used to hierarchical path search in those maps where the obstacles are densely distributed. On the other hand, a map complexity metric is defined based on the accumulation of xor calculations of given maps. This measure describes the complexity of a map and hence could reflect the performance of pathfinding algorithms, which could provide references for game maps design. The experimental results have validated the proposed analysis.*

Keywords: *pathfinding algorithm, performance analysis, map distribution, abstract map*

1. Introduction

In the maps of computer games, the quality and response speed of pathfinding directly affect the experience and satisfaction degree of game players. Most of the existing pathfinding algorithms such as A*[1-2] and Dijkstra [3], however, do not consider the distribution information of obstacles, which causes unnecessary time and storage cost. Another example is HPA*[4], which is a typical fast algorithm using the concept of hierarchy and abstract map to speed up the traditional A* and effectively redues the search space. It does not consider the terrain information when forming the abstract map and produces some unnecessary extension nodes and reduces the path optimality, which is especially obvious in a map with a large free area. Besides, M-A*[5] divides the map into unequal sub-regions by multi-scale strategy before forming its abstract map. It only considers the location information of start node and goal node in the partition process, which decreases the speed of pathfinding and increases the extension nodes. Quadtree [6] algorithm does consider the information of map distribution when dividing the map into clusters, but the condition of partition termination is too strict to require each cluster containing only one type of nodes. As a result, both the number of sub areas and abstract nodes are large, which leads to a lot of memory cost. Further, these algorithms only use local A* algorithm to calculate the shortest path in each pair abstract nodes and refine paths. They don't select suitable algorithm according to the distribution information of sub-regions, which reduces the overall execution efficiency of pathfinding algorithm to some extent.

In fact, the search performance could be very different even for one given algorithm when the maps have different distributions of obstacles. We cannot find an algorithm which is working the best in all types of maps with respect to the density of obstacles. Therefore, the distribution information in the map scene should be considered when the pathfinding algorithm is designed. Different pathfinding algorithms should be selected for different sub-regions. On the other hand, we attempt to propose a complexity measure for maps which could reflect the difficulty for path searching (which can be defined by searching time). Then pathfinding algorithms can also be analyzed through the computation of the complexity measure. In this paper, we present a new hierarchical pathfinding algorithm CDHPA* based on the obstacle distribution and a new map complexity metric called ACX based on the accumulation of xor calculation, which further analyzes the relationship between the obstacles distribution and the execution efficiency of pathfinding.

Received August 31, 2013; Revised February 19, 2014; Accepted March 7, 2014

2. Related Reserach

A*[1] is a kind of pathfinding algorithm based on a heuristic function which estimates the distance between the current node and the goal node and the next search node is selected. It can effectively reduce the searching time by moving to the most hopeful direction, and so it runs faster than Dijkstra [3]. But as we have mentioned, the scale and obstacles distribution of maps influences its performance greatly.

There are many improvement versions of A*. HPA*[4] algorithm is a kind of hierarchical pathfinding algorithm which is based on abstract graph. First, it constructs an abstract graph through dividing original map into some uniform clusters and links the key nodes of adjacent clusters. Then A* algorithm is used on the abstract graph to find an abstract path. Finally, it refines the abstract path to get a local near-optimal path between the start node and goal node. When the map is in large-scale, HPA* could divides the map into a multi-layer abstract maps, and further narrows the abstract space. Notice that HPA* does not consider the terrain information in the process of abstraction and refinement. This will reduce the optimality of the found path in large free regions in given maps. Quadtree[6] is also a hierarchical pathfinding algorithm, but it is different from HPA* in its map division method. It first divides the map into four uniform clusters, and checks the state of all cells in each sub-clusters respectively. If all of the cells are obstacles/non-obstacles in current cluster, this cluster will not be divided further. Otherwise, the current cluster will continue to be divided into four uniform sub-clusters until each cluster only contains one type of cells. Then the intermediate node of each cluster and some nodes selected in the boundary will be linked and thus forming the abstract graph. Quadtree algorithm considers the map distribution in the process of division, but it requires each cluster to contain only one type of cells, which increases the number of clusters and then the number of abstract nodes. M-A*[5] is a pathfinding algorithm using multi-scale environmental information and considers the influence of different search tasks in the process of division. It tends to refine the clusters near the start node and destination node. First, M-A* divides the map into four uniform clusters like Quadtrees and judges whether the start/goal node is in the current cluster or not. It only further divides the cluster that contains the start or goal node until the cluster contains only one cell. Then M-A* uses a bottom-up fusion algorithm to calculate the shortest distance of all free cells in each partition boundary, and gets the abstract map. M-A* improves the computational complexity of A* in the worst case. But,the process of division neglects the obstacle distribution and its needs to repeat the division process whenever the the start node and the destination node are changed.

In this paper, we develop a new hierarchical pathfinding algorithm based on the idea of abstract graph, which relaxes the division criteria in Quadtree according to the obstacle density. The purpose is to improve the efficiency and the optimality in maps that obstacles are densely distributed in some sub-regions. To further analyze the relationship between the map distribution and the pathfinding performance, we also study the complexity metrics of a map.

Many scholars have put forward a lot of metrics from the perspective of geography and generalized diagram in data structure for real maps. [7] uses the concept of entropy to calculate the map complexity, and [8] proposes a complexity metric of the geographical distribution and contour figure. [9] measures the complexity using boundary index, boundary contrast index, and mitotic respectively. However, these methods are not applicable to game maps, so some other related methods are put forward in the field of AI. [10] proposes a method based on Hamming Distance, which computes the complexity by calculating the number of different bits between two bit strings. [11] defines a game map complexity using relative Hamming Distance, and introduces the concept of regional variance. Both of the two metrics are defined by the different number of cells with respect to every neighboring rows or colums in a given map, without considering the shape of obstacles or the geometrical property.

3. Research Method

In this section, two methods proposed in this paper will be elaborated.

3.1. Hierarchical pathfinding algorithm based on map distribution—CDHPA*

In large-scale commercial computer games, many obstacles are densely distributed in the map scene such as buildings, lakes and mountains. There are often large free areas exist in this type of map. If we divide the map into uniform clusters simply according to the positions of

start and goal nodes and only use A* algorithm to calculate the shortest distances in all clusters, the path optimality and searching speed will be affected in the free areas. This paper presents CDHPA*, which considers the distribution information of obstacles in the process of division and refinement. First, the given map is divided into unequal sub-regions based on the obstacle distribution, where the number of sub-regions is determined by adjustable predefined threshold. Second, the free boundary nodes in the sub-regions are connected as abstract nodes to form a complete abstract graph. The shortest path between each pair of abstract nodes is calculated by Manhattan distance or bottom-up fusion algorithm depending on the density of obstacles. Finally, we find an abstract path on the generated abstract graph and then convert the abstract path to an local path. Both A* and Bresenham[12-13] algorithm are used in the process of refinement according to differnt obstacle density. A* is used for regions that have many obstacles and Bresenham algorithm for free areas. Bresenham's speed is faster and the extended nodes are less than A* in free areas. Next, we will introduce the details of CDHPA*, and its pathfinding process can be divided into two parts—preprocessing and online pathfinding.

3.1.1. Preprocessing

This part mainly contains map division, key nodes selection and the formation of abstract graph according to the obstacle distribution.

Step 1. Transform a real map to a grid map. Here we require the size of the map to be n*n and $n=2^j$, j is an integer. This is because the map will be divided into four equal sub-regions in each division based on the idea in Quadtree and M-A*. Figure 1 is a map selected from *Baldurs Gate II* [14] and its scale is 64*64. Figure 2 is a grid map that is the transform result of Figure 1. In Figure 2, the obstacles cells gather together, and the white cells represent walkable area.

Step 2. Build a quadtree according to the proportion of the obstacles, in which all final sub-regions are the leaf nodes. First, the grid map is divided into four equal areas and the obstacle proportion of each sub-region α is calculated using formula(1). Among them, *#obstacle* is the number of obstacles in current region, *#total* is the total number of nodes in the current region.

$$\alpha = \frac{\#obstacle}{\#total} \tag{1}$$

In Quadtree, the division will continue until each sub-region containing only one type of cells, i.e., free cell or obstacle cell. This is too strict and tends to generate lots of sub-areas and nodes in abstract graph. Here, we set a threshold β experimentally to relax this stop criterion. The relation between α and β will determine whether the current sub-region is further divided or not. Accordingly, the states of the leaf nodes (sub-regions) in the quadtree is marked by λ. There are three kinds of situations: (1) α = 0, which means the current area has no obstacle and need not to be divided any longer. In this case, assign λ=1 to current sub-region (leaf node and obstacle section); (2) $0<\alpha<\beta$ means that the ratio of obstacle cells and free cells is close and this region should continue to be divided; (3) $\alpha>\beta$, means the current region has enough obstacles and does not need further dividing. The state marker λ is set to 0 (leaf node and free section). This method divides the original map into some non-uniform sub-regions, and a lot of them are free areas. Bresenham algorithm can be used to calculate the shortest distance between each pair of abstract nodes in this type of region.

Figure 1. Real game Map

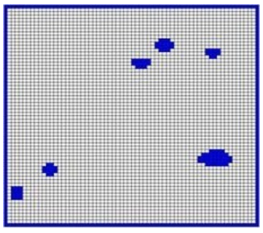

Figure 2. Grid Map

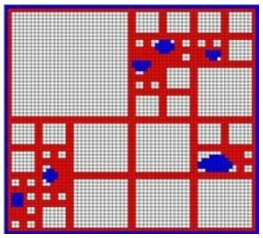

Figure 3. Partition Results

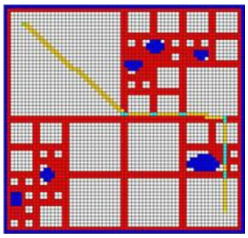

Figure 4. Shortest Path

We should mention that the value of threshold λ is different in different maps. Its value affects the number of sub-regions directly, which further affects the number of abstract nodes, extended nodes, and the searching time. Therefore, the suitable threshold should be found to balance all these aspects. For Figure 2, The threshold λ is 0.3 obtained experimentally. Figure 3 is the division result of Figure 2, and it can be seen that there are many free areas. The red cells represent abstract nodes which are all free cells on region boundaries.

Step 3. From an abstract graph. After the process of division, CDHPA* computes the shortest distance for each pair of abstract nodes, then forms the abstract graph according to λ value of each sub-region. When λ is equal to 1, it means that the current region only contains free cells and the Manhattan distance is the shortest path. When λ is equal to 0, it means that the current region has a lot of obstacle cells and A* is used to calculate the shortest path.

To summarize, in the stage of preprocessing, CDHPA* divides the map and chooses different methods to calculate the shortest distance based on the distribution of obstacles. This algorithm does not need the location information of start and goal nodes in division like M-A* does. Therefore, the preprocessing is needed only once even for multiple tasks. Although the threshold determination needs many experiments in advance, it is still acceptable when we taking the average executing time for many different tasks.

3.1.2. Online Processing

Step 1. Insert the start and goal nodes into the abstract map. If the start/goal node is already an abstract node, then the insertion is unnecessary and we directly go to the next step. Otherwise, we find the corresponding sub-region based on the area indicator in the data structure, and link every boundary free cells in the corresponding sub-region, and finish the insertion step.

Step 2. Find an abstract path between the start and goal node in the abstract map. First, we judge whether the start node and goal node locate in the same region or not. If yes, then CDHPA* directly chooses A* or Bresenham algorithm to find the shortest path according to λ. Otherwise, this algorithm uses A* to search abstract path and continue to execute the next step.

Step 3. Refine the abstract path to a local one. Considering every sequential pair of nodes (one node and its next one) in the abstract path, if the state marker λ of the region containing adjacent nodes of the current pair of nodes is equal to 1, Bresenham algorithm is used to find the final path between these two nodes. If that value of λ equals to 0, A* is used to find the final path correspondingly. The speed of pathfinding of Bresenham is faster than A* in the free areas. Here we have considered the map distribution and selects different algorithms to path refinement. In Figure 4, the yellow line illustrates the found path between (5,5) and (60,59) in Figure 2, which is also a shorest path.

It can be seen from the implementation process of CDHPA* that the map distribution is considered in both map division and path refinement. According to the obstacles distribution, we choose the corresponding method to abstract and refine paths, which can effectively improve the execution efficiency.

On the other hand, the performance of search algorithms should be considered as the important reference when map designers generates a game map scene. The designers incline to form maps in which pathfinding is done quickly, so that reducing the players' waiting time. Further, they can also increase the game's interesting degree through controlling the map's complexity and then control the pathfinding difficulty. Therefore, this paper also puts forward a kind of map complexity metric called ACX. ACX measures map complexity by analyzing obstacle distribution, and provides reference for map design due to the high correlation between ACX and the search efficiency of A* algorithm.

3.2. Metric Index of Map Complexity-Accumulation of Xor

The design of obstacle distribution decides the performance of pathfinding algorithm in the game map scene and also the interesting degree of games. Taken the classic 2D game <Pacman> for example. This game's roles chase each other in a maze-like map and the process of chase is acturally pathfinding. Intuitively, when the map complexity is low, the Ghost can catch up with Pacman easily and the game does not have challenge. On the contrary, if the map complexity is too high, the Ghost will hardly grasp Pacman and the game is too difficult.

This shows the complexity of game map is highly correlated to the search efficiency and the player's experience. In this section, we will define a formal measure to describe the map complexity.

3.2.1. The Definition of ACX

There are several factors related to the map complexity, such as the map scale, the number and the density of obstacles. Different from what we could imagine intuitively, the complexity does not necessarily increase when the map scale becomes larger or the number of obstacles increases. For example, it is obvious that the number of obstacles in Figure 5 is more than that in Figure 6, but it is more difficult to find paths in Figure 6. This is because that the map distribution of Figure 5 is more dense than that of Figure 6, and we can observe that the obstacle regions in Figure 5 is convex from the mathematics point of view. We will incorporate this information in the defintion of the complexity metric.

First, we convert a map to a matrix whose elements represent the states of every cell. Let t (i, j) denote the state of cell located at the i-th row and the j-th column in the map, and it takes value 0(free) or 1(obstacle). Before defining ACX, we need to introduce several functions:

$$F(1):\min_j^i = \arg_j \min(t(i,j)=0);\ F(2):\max_j^i = \arg_j \max(t(i,j)=0).$$

Here, F(1) is defined as the minimum column no. of the cell with t(i, j) =0 in the i-th row. Similarly, F(2) is used to find the maximum j-value through the same condition as F(1). ACX is defined as follows:

$$ACX(M)=\sum_{i=1}^{m}((\sum_{j=\min_j^i+1}^{\max_j^i} t(i,j)\oplus t(i,j-1))/2)+\sum_{j=1}^{n}((\sum_{i=\min_i^j+1}^{\max_i^j} t(i,j)\oplus t(i-1,j))/2)$$

ACX calculates map complexity through accumulating the xor-value of adjacent cells line by line, and the complexity can be get in time O (n^2). The value contains two parts: the first item is obtained by accumulating ACX-value row by row, and another part is obtained by accumulating xor-value column by column. For each row (column), searching the $\min_j^i$ ($\min_i^j$) cell and starting to sums up the xor-value between every two adjacent cells until the $\max_j^i$ ($\max_i^j$) cell. Finally the summation of the xor-value of each row and column is the complexity of map. The ACX measure will take its minimum value of zero if the free cells in rows and columns are not seperated by any obstacle, which implies the lowest map complexity. This is consistent with our intuition.

If the ACX-value of a certain row (column) is not equal to 0, then this row (column) has at least one obstacle cell. As can be seen from the definition, the ACX-value of map is not directly related to the map scale and the number of the obstacles, it is only related to the state of every two adjacent cells. The higher the return value, the higher the complexity of the map. For example, the complexity from Figure 5 to Figure 8 are shown as follows: ACX(M5)=0; ACX(M6)=160; ACX(M7)=86; ACX(M8)=130.

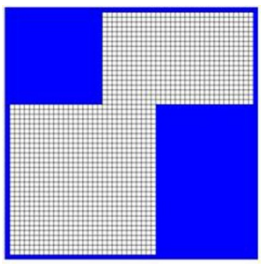

Figure 5. Example 1

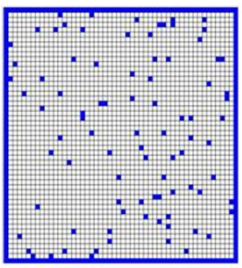

Figure 6. Example 2

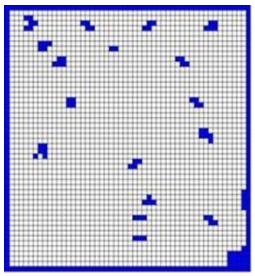

Figure 7. Example 3

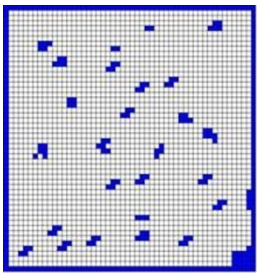

Figure 8. Example 4

Although the number of obstacles in Figure 5 is the largest, this map has the lowest ACX complexity value and resulting best pathfinding efficiency. We can see that ACX value is not only determined by the number of obstacles and the size of map. To summarize, if the obstacles gather into a block or locate at the edge of the map, the ACX complexity is lower relatively even there are many obstacles. In contrast, when the obstacles are scattered in a map like a maze, the complexity is relatively high.

3.2.2. The Correlation of Acx Metric and Search Efficiency of A* Algorithm

Although there are many improved version of A*, the traditional A* is still the most widely used path search algorithm in many applications. In the search process of A*, it checks all the neighboring nodes repeatly for the current node to predict the most promising direction and extend the next node by the heuristic function. When obstacles are near the edge or gathered into a block, the nodes in the block will not be extended any more. The search efficiency will be relatively high. In contrast, when the most cells' states are not consistent with their adjacent cells, the algorithm will take more time to check neighboring nodes and compute their heuristic values. The number of expanded nodes is also increasing. As a result, the search efficiency will be decreased. That is consistent with the principle of ACX metric based on xor accumulation. The higher the value of ACX is, the lower the execution efficiency of A* will be. Thus, the performance of A* can be ajusted by controlling the ACX complexity in the game map design.

In order to prove the correlation between ACX metric and A* algorithm, numerous experiments have been conducted, and the results are shown in Section 4. The actual search efficiency for pathfinding is the ratio of the total number of expanding nodes and the length of the shortest path for the start to goal node, as shown in formula 2.

$$E(A\lg othrim) = \frac{NodeSerche\ d}{PathLength} \quad (2)$$

4. Experiments and Results

In order to testify the impacts of the map distribution information on the performance of pathfinding algorithm, two groups of experiments have been conducted.

4.1. Comparisons of CDHPA* with HPA* and M-A* algorithm.

In this group of experiments, the average performance of above three algorithms is compared based on the same search tasks to verify the effectiveness of considering the map distribution information during the process of pathfinding. 10 maps are selected from *Baldurs Gate II*, among which Map1-Map5 are 64*64 in size and the other 5 maps are 128*128. At the same time, 10 tasks of pathfinding are randomly generated to carry out the CDHPA*, HPA and M-A* in each map respectively,and different algorithms have same tasks in the same map.

Notice that the searching time of M-A* starts computing after the abstract map is completely formed. For the sake of easy comparison, we divided the whole running time of CHDPA* and HPA* into three parts: preprocessing time, the time of insert the start and goal and on-line pathfinding time. The following symbols are used to express all metrics of performance of pathfinding respectively. *t/ms* is the total time of searching abstract path and refining the abstract path. *Len* is the length of full path between start node and goal node. *Vec* is the total number of nodes that are expanded in the process of pathfinding. The final comparison results are shown in Table 1. Note that the data are the average value of ten pathfinding tasks and the threshold β is obtained experimentally. We compare these three algorithms from the following three aspects:

a) On-line searching time. Table 1 show that the on-line searching time of CDHPA* is less than M-A* and HPA*. The reason is that CDHPA* considers the obstacles distribution information in the process of division and performs Bresenham straightline algorithm in the free areas as much as possible, which is faster than HPA* and M-A* using A* in refining path.

b) The number of expanded nodes. The number of expanded nodes of CDHPA* is less than that of M-A* but more than HPA*. CDHPA* expands all free nodes on the boundaries of all sub-regions as M-A* does, which gurantees the optimal path will be found. HPA* uses uniform partition strategy and only chooses a few key nodes on the region boundraries as

abstract nodes to form the abstract map, so it expands less nodes. However, HPA* cannot gurantee to find the optimal path for that reason.

c) Preprocessing time. In addition to the online-pathfinding time, each algorithm includes preprocessing time and time for inserting nodes. M-A* can perfom inserting operation only when the position information of start and goal nods are confirmed. Ever when the task changes, M-A* needs to re-divide the map, which leads to a longer waiting time. CDHPA* divides the map without the location information of start and goal nodes, which reduces the users' waiting time for multiple tasks. HPA* does not consider any informaiton of distribution and start/goal locations and always equally divides a game map, and therefore its preprocessing time is less than CDHPA*.

d) The path optimality. The path optimality is an important index of measuring algorithm performance[15]. Since CDHPA* selects all boundary free cells as nodes in the abstract map and uses A*or Bresenham to find the shortest path according to the value of λ. That guarantees the found path is optimal.

To conclude, without affecting the path optimality, CDHPA* improves the speed of on-line pathfinding and reduces the storage cost of M-A* by considering the map distribution. It is more suitable for pathfinding task that has higher demand for path quality and real-time response.

Table 1. Comparison Results of Different Algorithms

MAP	β	Performance	CDHPA*	M-A*	HPA*
Map1	0.1	t/ms	0.63256	0.65958	2.96582
		len	71.1	71.1	71.5
		Vec	282.4	391.6	192.7
Map2	0.2	t/ms	0.58666	0.65810	2.48668
		len	60.5	60.5	61.5
		Vec	238.1	343.6	167.2
Map3	0.3	t/ms	0.78825	1.48901	2.80040
		len	72.7	72.7	73.5
		Vec	297.3	398.9	183
Map4	0.1	t/ms	0.81666	0.8691	2.70956
		len	73.3	73.3	73.7
		Vec	332.5	428.5	186.8
Map5	0.2	t/ms	0.46617	0.72967	2.63379
		len	64.6	64.6	66.4
		Vec	252.8	358.2	181.2
Map6	0.1	t/ms	3.30625	2.72534	5.44052
		len	121.7	121.7	123.3
		Vec	682.6	737.4	338.5
Map7	0.15	t/ms	2.59991	3.16892	5.56705
		len	138.2	138.2	138.4
		Vec	709.1	872	357.9
Map8	0.15	t/ms	1.6171	2.06603	4.26967
		len	101.4	101.4	103
		Vec	548.7	653.3	280.4
Map9	0.12	t/ms	2.74448	3.68443	4.84008
		len	109.9	109.9	110.9
		Vec	595.9	706.6	317.3
Map10	0.12	t/ms	3.55374	5.69287	5.63222
		len	132.7	132.7	133.9
		Vec	689.6	864.2	353.6
Average		t/ms	1.71118	2.17431	3.93458
		len	94.61	94.61	95.61
		Vec	462.9	575.43	255.86

4.2. Correlation of Acx Metric with the Search Efficiency of A* Algorithm

This group of experiments is conducted to validate the relationship between map complexity and the search efficiency of A*. 600 maps (50*50 size) are generated by random map generator and ACX metric is carried out on each map to calculate the map's complexity. 1500 pairs of (start, goal) nodes are generated randomly to find paths in each map and the

average efficiency of the 1500 searches is computed. To make the results be more obvious, we only use those pair of nodes with its path length larger than 50 units. Figure 9 and 10 show the results, where Figure 9 is the curve figure of ACX complexity, and Figure 10 is the efficiency curve of A*. In Figure 9, the maps are numbered from low to high based on the value of ACX. The x-axis is the map number and y-axis is the corresponding value of complexity. In Figure 10, the y-axis represents the search efficiency of A* in each map corresponding to that in Figure 9.

From the two figures, we can see that the trend of ACX curve is consistent with that of search efficiency curve of A* algorithm except of some small jitters in local areas. That may be caused by the methods of maps' generation, the existence of particularly low efficiency path and no-path nodes pairs and so on. [10] mentioned when the map's complexity based on Hamming distance metric is more than 100, the jitter of the search efficiency curve is more obvious. However, this does not occur in the method of ACX metric. he consistency of ACX metric and the search efficiency does not disappear with the increase of map's complexity. Therefore, ACX can be used to measure the map complexity in designing maps which uses A* or its improved versions to find shortest paths.

From the above two groups of experiments, we can see that the distribution information in a map is highly relevant with the performance of search algorithms. By analyzing their relationship, we can design low-cost and efficient algorithm with reference to the map distribution. On the other hand, the map complexity measure can help in map design which could determine different obstacle distributions, producing different search difficulties.

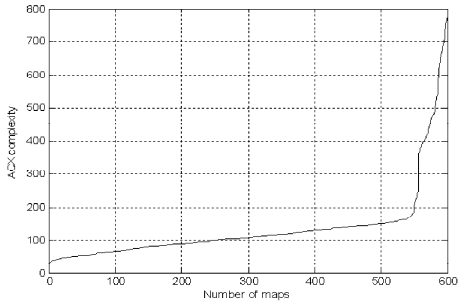

Figure 9. Complexity

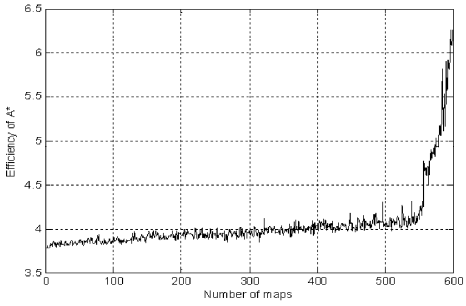

Figure 10. A* Search Efficiency

5. Conclusion

In this paper, the relationship between map distribution and the performance of searching algorithms is discussed. A new hierarchical pathfinding algorithm called CDHPA* is developed to reduce the time and storage cost by incorporating the obstacle distribution. Besides, a map complexity metric based on the accumulation of xor is proposed, which can be used to describe how easy or difficult to search a path in a given map. This is very helpful for game map designers. They could tune the pathfinding efficiency (or the pathfinding difficulty on the contrary) in a map by generating maps with different complexities. Overall, the two proposed methods demonstrate the effects of obstacles distribution on the performance of search algorithm from two angles – the design for pathfinding algorithms in a particular distribution and the design for maps based on complexity metric. In future work, we will study the methods determining threshold λ intelligently and analyze the correlation of ACX and the search efficiency of other pathfinding algorithms.

References

[1] Rabin S. AI Game Programming Wisdom. Beijing: Tsinghua University Press. 2005.
[2] Lulin Chen, Goufeng Qin. K Nearest Neighbor Path Queries Based on Road Networks. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2013; Vol.11(11):6637-6644,
[3] Dijkstra E. A Note on Two Problems in Connexioin with Graphs. *Numerische Mathematik.* 1959; 1(1): 269-271.
[4] Botea A, Muller M, Schaeffer J. Near-optimal Hierarchical Pathfinding. *Journal of Game Development*. 2004; 1(1): 7-28.

[5] Yibiao Lu, Xiaoming Huo, P Tsiotras. *Beamlet-like Data Processing for Accelerated Path-planning using Multiscale Information of the Environment.* Proceeding of 49th IEEE Conference on Decision Control. Atlanta, GA. 2010; 3808-3813.
[6] Hanan Samet. The Quadtree and Related Hierarcherical Data Structures. *ACM Computing Surveys.* 1984; 16(2): 187-260.
[7] Mowshowitz A. Entropy and the Complexity of Graphs. *An Index of the Relative Complexity of a Graph Bulletin of Mathematical Biophysics.*1968; 30(1): 175-204.
[8] MacEachren A. Map Complexity. *The American Cartographer.* 1982; 9(1): 31-46.
[9] David F. Measuring Map Complexity. *Cartographic Journal.* 2006; 43(3): 224-238.
[10] Pan Su, Yan Li, Wenliang Li. *A Game Map Complexity Measure Based on Hamming Distance.* Proc. of the 3rd International Conference on Computational Intelligence and Industrial Application. Wuhan. 2010; 38(7): 332-335.
[11] Yan Li, Tiesong Li, Cai Chen. Map Complexity Measurement Based on Relative Hamming Distance. *Computer Engineering.* 2012; 38(7): 10-12.
[12] Chao Yuan. Reserach of the Bresenham Straight Line Generation Algorithm. *Journal of Sichuan university of science & engineering.* 2006; 19(2): 36-40.
[13] Yingliang Jia, Huanchun Zhang, Yazhi Jing. A Modified Bresenham Algorithm of Line Drawing. *Journal of Iamge and Graphics.* 2008; 13(1): 158-161.
[14] Nathan Sturtevant's Moving AI Lab. Pathfinding Benchmarks, http://www.movingai.com/benchmarks/.
[15] Yonghua Xu, Fei Shao. An Optimal Routing Strategy Based on Specifying Shortest Path. *TELKOMNIKA Indonesian Journal of Electrical Engineering.* 2013; 11(10): 6224-6231.