A COMPARISON STUDY ON THE REDUCTION OF SEARCH SPACE USING FRAME NODE TECHNIQUE FOR SIMULATION AND COMPUTER GAMES

SUMET PONGPHANKAE, CHUN-CHE FUNG

School of Information Technology, Murdoch University, South Street 6150, Western Australia
E-MAIL: s.pongphankae@murdoch.edu.au, l.fung@murdoch.edu.au

Abstract:

In simulation and computer game domains, two issues related to pathfinding problems are search algorithm and search space. This paper describes and compares four techniques used for the reduction of search space generation in 2-D simulation and game world environments. In order to improve the computational overheads, reducing size of search space is necessary. The size of search space is related to the number of nodes and edges. A high number of nodes and edges will have implications on the usage of computer memory. Therefore, reducing the number of nodes and edges will bring direct benefits to the memory usage and reduction of the search time. In this study, four techniques for the reduction of search space size are compared and investigated. In this paper, frame nodes used in the hierarchical adaptive flood filled (HAFF) technique is presented. This paper also presents the comparison results from the application of four hierarchical search spaces.

Keywords:

Search space generation; Navigation graphs; games and simulation; Frame node; Hierarchical adaptive flood filled technique

1. Introduction

Studies of pathfinding in simulation and computer games have been researched for many years. Many research reports have shown and proposed techniques for pathfinding by developing effective search techniques, or improving the usage of computer memory effectively [1, 2]. Finding the path, in computer games and simulation, is a major component of Artificial Intelligent (AI) agents or Non Player Characters (NPCs) [3]. The task of pathfinding is a key component in order to search for a valid or optimal path in the search space. The search space is normally represented by graph data compose of sets of nodes and edges. Generating the search space from the simulation or game world environment is therefore essential for the execution of the search algorithms. There are several techniques that can be used to transform the context of the simulation or game environments into graphs data or search space. Some techniques can generate the search space automatically and some techniques such as waypoint technique have to be done manually [4]. Manual search space generation can be time-consuming when the world environment is large. Although automatic search space generation may address the timing issue, the size of the search space could be larger than manual search space generation. Moreover, when the game world environment is large, the size of the search space will also be increased proportionally. As the search space has to be stored in the computer memory, a large search space will therefore require more memory more than a small search space. In addition, searching an optimal path in a large search space is time-consuming when comparing with a smaller search space. The study of this research therefore focuses on how to decrease the size of search space that is generated from a simulated world environment. Four techniques have been investigated and compared and the results are shown in the latter section.

This paper is organised as follows. Section 2 describes the background of the problem and related work. Section 3 describes the techniques for search space generation and reduction. Section 4 reports on the experiments and the results are discussed in Section 5. Section 6 provides the conclusion and suggestion for future work.

2. Related work

Search space, in computer games and simulation, represents a world environment with passable areas, block areas and paths within the passable areas. All positions of world environment are represented as set of nodes, and the paths among nodes are represented as set of edges. A game world environment has to be converted to search space that will be used for searching. The number of nodes and edges depends on the size of search space. The process of abstraction is to replace the state of the search space by another abstraction [5] that is called *abstract search space* with an objective to reduce searching in the search space. Sturtevant and Jansen's study has described the varieties of

abstraction techniques for representing the size of search space, and the study also introduced sector abstraction, line abstraction and node limit abstraction [7]. Another technique known as the Clique abstraction for pathfinding problem was also introduced by Sturtevant and Buro [6]. The Clique technique is similar to the concept of cliques graph in graph theory that a graph can be represented completely by subgraphs. Sturtevant and Buro abstract nodes are based on two patterns that are *cliques* and *orphans*. The maximum number of nodes for each clique is four. The number of node, cliques abstraction of each state can be reduced by a factor of four nodes. The nodes of the new state in the higher search space will be assigned with new information for each node. Line abstraction (LA) technique was also introduced by Sturtevant and Jansen [7]. LA works by finding the nodes that are aligned in the vertical or horizontal axes and LA will then abstract those nodes into a single node. The length of line abstraction is denoted by *k* that starts at 2 nodes and the maximun length of line abstraction is in the range between 2 to 6. However, line abstraction has some disadvantages with respect to quality of the paths. The quality of the path could reduce in the case that a path is jagged. In this paper, the length of line abstraction is limited to 3 by setting the value of *k* to 3. This will help in maintaining the quality of the paths and reduce the process for reassigning the spatial information for a new node. Star abstraction was proposed and developed by Holte et al. [2]. The star abstraction is also known as *radius of abstraction*. It works by finding the surrounding nodes within the range of certain radius from a particular node. The length of radius was defined by the user. There are several features of star abstraction that are similar to the proposed technique in this paper and they are described in subsequent sections. However, the process of reducing the search space is different between the two. In addition, star abstraction is not designed to support multi-sized AI agents or NPCs. The next section details the search space reduction technique.

3. Search Space Reduction Techniques

3.1. Frame Node

The Hierarchical Adaptive Flood Filled technique has been proposed previously for the automatic generation and reducing the size of search space using multi-sized frame nodes [8]. Frame node could have multi-size square where the size of each frame node are defined by 2^n+1, where n = 1 to n. By varying the value of n, the result of the frame size can be computed from the *Compute_the_frame_size* algorithm. Figure 1 illustrates different frame nodes in various sizes.

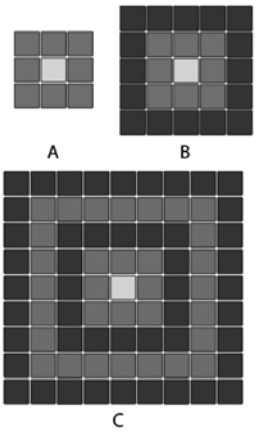

Figure 1. Frame nodes of size 3, 5 and 7

Figure 1A shows a frame node of size 3x3. The second and third sizes of the frame node are shown in Figure 1B and 1C respectively. The size of frame node varied according to the minimal clearance values in the map.

Frame node size could be determined by the algorithms shown as listing 1 and listing 2. These two algorithms are named as *Search_Minimal_Clearance_Value* and *Compute_the_Frame_Size* algorithm and they are used to compute the size of frame node.

3.2. Search Minimal Clearance Value

The search minimal clearance value algorithm shown below is designed to determine the minimal clearance value in the maps. The minimal clearance value is referred to the minimal distance between obstacles. The input to this algorithm is a binary image map and the output is minimal clearance value. The *clearanceValues* is a list of clearance values that is used to store the clearance values.

The *clearanceSpaceCount* is used for counting the length of the clearance values. Lines 1 to 12 and lines 13 to 24 in the *Search_Minimal_Clearance_Value* algorithm are procedures that read all the locations in the map with both x and y axes. Lines 4 to 10 and lines 16 to 22 are processes used for adding the value of *clearanceSpaceCount* into list of clearance values and updating the value of *clearanceSpaceCount* to 0. In line 25, the minimal clearance in *clearanceValuesList* is assigned to *minimal_clearance_value* and the value will be used in the next algorithm.

Listing 1 Algorithm: Search_Minimal_Clearance_Value

```
Input: Binary_game_map Map;

List_Of_ClearanceValue    clearanceValuesList

set clearanceSpaceCount = 0

1: for all location_x in Map
2:      for all location_y in Map
3:          if location_y.isBlock() then
4:              if clearanceSpaceCount not 0 then
5:                  add clearanceSpaceCount to
clearanceValuesList
6:                  set clearanceSpaceCount = 0
7:              end if
8:          else
9:              update by increasing value of
clearanceSpaceCount
10:         end if
11:     end for
12:end for

13: for all location_y in Map
14:     for all location_x in Map
15:         if location_x.isBlock() then
16:             if clearanceSpaceCount not 0 then
17:                 add clearanceSpaceCount to
clearanceValuesList
18:                 set clearanceSpaceCount = 0
19:             end if
20:         else
21:             update by increasing value of
clearanceSpaceCount
22:         end if
23:     end for
24:end for

25: minimal_clearance_value = search minimal value in
clearanceValuesList

Output: minimal_clearance_value
```

The *Compute_the_frame_size* algorithm as shown above is designed to determine the size of frame node by using the minimal clearance value as an input. Lines 1 to 6 of the algorithm is used to compute the frame node's size by comparing the minimal clearance value with 2^n+1, where 2^n+1 is size of frame node *m* x *m* as shown in Figure 1. Line 4 is used to update the value of *framenodeSize* to 2^n+1 when minimal clearance value is greater than 2^n+1. The procedure will end when minimal clearance value is less than 2^n+1.

Listing 2 Algorithm: Compute_the_frame_size

```
Input: minimal_clearance_value ;

1: set framenodeSize = 0
2: set n=1;
3:    while minimal_clearance_value >= 2^n +1
4:          update by set framenodeSize = 2^n +1
5:          update by increasing n by 1
6:    end while

Output: framenodeSize
```

4. Experiment

In this section, experiments and tests that have been carried out to test the proposed frame node of HAFF technique are described. The results from the frame node techniques are compared with the clique technique, the line abstraction technique, and the Star technique. The proposal was tested with a set of 70 game maps. Different types of obstacles are simulated on a variety of maps. The sizes of the map employed in the experiments ranged from 64x64 to 512x512. Each node in map has 8 connections maximum. All the maps used in this experiment are binary images and the maps are in the top views orientation. The walkable areas are represented in white and the obstacles are represented in black on the maps. The complexities of the game maps are categorized by the shapes of the obstacles appeared on the maps. There are a variety of obstacle shapes created in map including squares, rectangles, triangles, polygons, circles, and free forms. In addition, several obstacle shapes are mixed into a single map. All the maps were tested using the four techniques as described previously. The experiment was carried out on 2.0GHz Intel core 2-duo processor with 4GB RAM running OSX 10.6.6 using Xcode 3.2.2. Results from the experiments are described in the following section.

5. Results and Discussion

The experiment results for the different game maps generated using the four techniques are presented. Clique technique, line segment technique, star technique and the proposed frame node of HAFF technique were used to reduce the search space that were generated from the binary maps in the experiment. The search space is referred to the number of nodes and the number of edges that are generated from the walkable areas in the maps. Breadth-First Search is used for validating all the nodes and edges generated by each technique. They should be connected and could be searched by the search algorithm.

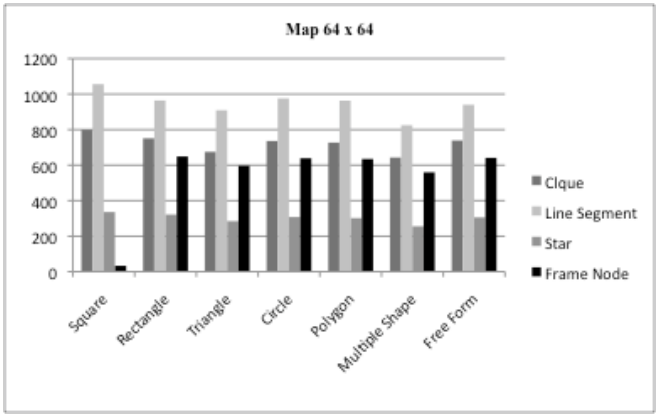

Figure 2. Number of Nodes in Map 64 x 64

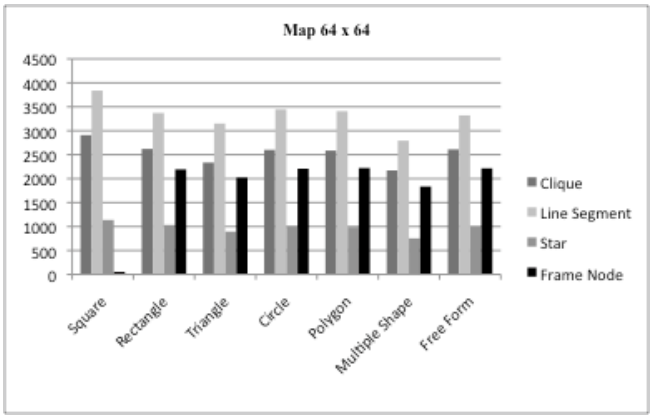

Figure 3. Number of Edges in Map 64 x 64

Figures 2 and 3 show the results from four different techniques for reducing the search space. The results shown in these figures are the number of nodes and the number of edges, which were generated from map size of 64x64. Each map has different types of obstacles including square obstacles, rectangular obstacles, triangular obstacles, circular obstacles, polygonal obstacles, multiple shapes and freeform as shown in Figures 2 and 3. For example, in the case of rectangle, the map will have the rectangular obstacles. For maps that contain multiple shapes, it will have different shapes of obstacles in a single environment.

In Figures 2 and 3, although the star technique could reduce size of search space better than the other techniques in most maps, the frame node techniques could dramatically reduce the number of nodes and number of edges in the maps contains the square obstacles. The number of nodes decreased by approximately 10 times when compare frame node techniques with the star technique. In addition, the number of edges was decreased by approximately 20 times. The size of frame node took into account of the minimal clearance value, therefore the maps size can also has an effect in the reduction of the search space. When the map size is small, there is a high probability that the minimal clearance between each of obstacle might be limited by the map size.

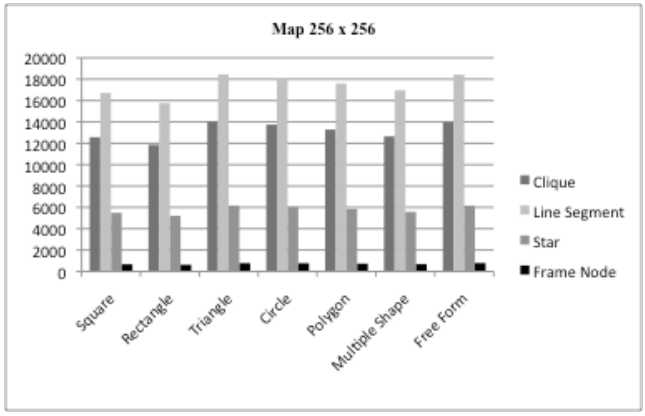

Figure 4. Number of Nodes in Map 256 x 256

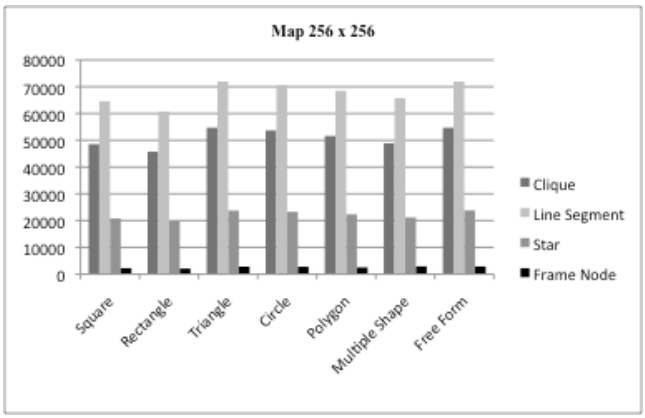

Figure 5. Number of Edges in Map 256 x 256

Figures 4 and 5 show the comparison results that were gathered from maps size of 256 x 256, where each map includes different types of obstacles and the walkable areas were about 81.69 percent on average. The number of nodes and the number of edges was quite high on each map. The number of edges was about 200,000 edges. When Frame node technique was applied to the map of 256x256, it dramatically reduced the size of search space, the number of nodes and the number of edges had reduced significantly. When comparing the results of frame node technique with the other techniques, the results can reduce by about 8.10 to 8.64 times in each map. The results had shown that the frame node technique performs better on larger maps.

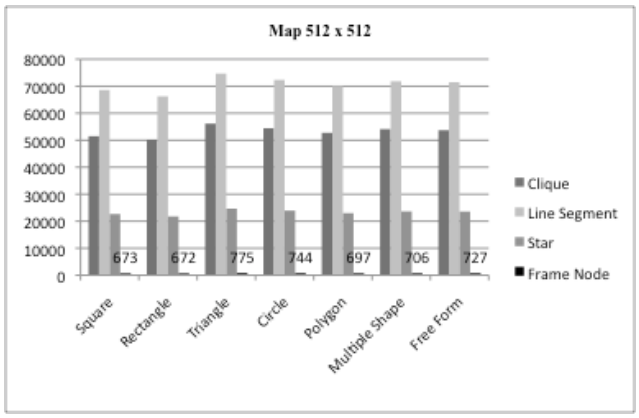

Figure 6. Number of Nodes in Map 512 x 512

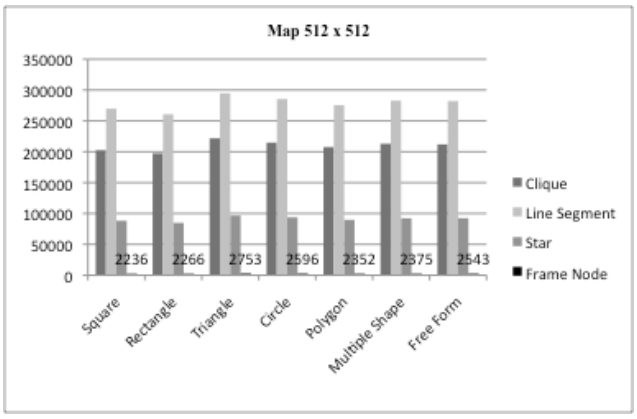

Figure 7. Number of Edges in Map 512 x 512

The comparison results show in Figures 6 and 7 were the number of nodes and the number of edges generated from map size of 512x512. The walkable areas in each map were 82.67 percent on average. The search space size for this larger map was huge. The number of nodes was about 200,000 nodes on average, and the number of edges was over 800,000 edges in each map. Frame node technique was utilized in order to reduce the size of search space. The number of nodes and numbers of edges when compared to star technique had dramatically reduced by about 32.7 and 37.3 times on average. All the results from the experiment show that the proposed frame node technique could significantly reduce the search space size. From this observation, it is deduced that each frame node's size can support different AI agents' sizes.

6. Conclusions

This paper presented the comparison results of four techniques for reducing the search space, and a new frame node technique is proposed. The proposed technique can reduce the size of search space that is generated from the game maps. The frame node technique is simple to implement. The results show that frame node of HAFF technique could dramatically reduce the number of nodes and the number of edges when compared to the other techniques. This could help to save the memory usages and improve the time for computing the navigation paths for NPC's. Frame node techniques can provide better results on larger maps. Moreover, it could handle a variety of obstacle types such as polygonal obstacles, circular obstacles and free form that are in the maps. It could also support multi-size AI agents. AI agents that are smaller size could be used as a guide for pathfinding. In future, improvement to the proposed HAFF technique on the handling of unformed search space and difficult clearance will be investigated.

References

[1] Y. Björnsson and K. Halldórsson, "Improved Heuristics for Optimal Path-finding on Game Maps," in *AIIDE*, 2006, Marina Del Ray, California, USA, 2006, pp. 9-14.

[2] R. C. Holte, T. Mkadmi, R. M. Zimmer, and A. J. MacDonald, "Speeding Up Problem-Solving by Abstraction: A Graph-Oriented Approach," *Artificial Intelligence on Empirical AI*, 1996.

[3] B. Reese and B. Stout, "Finding a Pathfinder," in *Association for the Advancement of Artificial Intelligence*, 1999, pp. 69-72.

[4] P. Tozour, "Search Space Representations," in *AI Game Programming Wisdom 2*, 1 ed, S. Rabin, Ed.: CHARLES RIVER MEDIA, INC., 2004, pp. 85-102.

[5] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald, "Hierarchical A*: Searching abstraction hierarchies efficiently," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[6] N. Sturtevant and M. Buro, "Partial Pathfinding Using Map Abstraction and Refinement," in *AAAI*, 2005, pp. 1392–1397.

[7] N. Sturtevant and R. Jansen, "An Analysis of Map-Based Abstraction and Refinement," in *International Symposium on Abstraction, Reformulation and Approximation*, 2007.

[8] S. Pongphankae, F. C.C., and K. Wong, "Automatic Search Space Generation with Hierarchical Adaptive Flood Filled for Simulation and Computer Game in Confined Environment," in *ICACT*, 2011, pp. 664-669.