Automatic Search Space Generation with Hierarchical Adaptive Flood Filled for Simulation and Computer Game in Confined Environment

Sumet Pongphankae*, Fung C.C.*, Wong K.W.*

*School of Information Technology, Murdoch University, 90 South Street 6150 Western Australia

s.pongphankae@murdoch.edu.au, l.fung@murdoch.edu.au, k.wong@murdoch.edu.au

Abstract— In simulation and computer game domains, pathfinding is an important capability for mobile and artificial agents. The problems of pathfinding have been widely studied for decades. The two significant aspect of pathfinding are related to search algorithm and search space. This paper mainly focuses on the search space generation aspect. Search space could be generated manually or automatically and the size of the search space has significant effects on the computational time and memory usage. Therefore, improving the search space generation techniques is essential to improve the system performance. In this paper, a Hierarchical Adaptive Flood Filled (HAFF) technique for automatic search space generation from binary image of a map is presented. From the experimental results, it is shown that the search space can be significantly reduced with the proposed method.

Keywords— Search Space Generation, Simulation, Computer Game, Graph, Hierarchical Adaptive Flood Filled technique.

I. INTRODUCTION

Simulations and games are useful for the design of communication systems in confined environments. For example, an autonomous mobile agent could be used to patrol or monitor a designated area for surveillance purposes. The agent may be required to navigate within its territory, or it could be sent as a matter of urgency to specific destinations and reports to the control centre. As such, it is necessary for the agent to possess the ability of pathfinding. Similar applications and requirements can be found in computer games for artificial agents or non-player characters (NPC). There are several factors related to the pathfinding problem that have been widely investigated in various aspects [1]. Some of the research reports have focused on improving the search algorithms' efficiency and some reports have studied other aspects on the search space and search algorithms (e.g. [2-4]). In simulation and computer games, the problems of search space generation are related to search space representation, that is, how to convert the map data into graph data structure. In real world, objects and environment are in continuous forms. Searching for a route in a continuous environment requires knowledge about the environment with information such as the agent's position, learning about the passable or blocked areas etc. It has been shown that there are various techniques such as online path planning, offline path planning, or a combination of both of these techniques that can perform the searching task. This paper aims to investigate the techniques for search space generations that could be used in the simulation of real world environments and in computer games. It also introduces the Hierarchical Adaptive flood filled (HAFF) technique.

This paper is organised as follows. Section II describes the background of the problem from related work. Section III explains search space generation, the HAFF technique and how to use it. Section IV reports on the experiments and Section V discusses on the results. Section VI provides the conclusion and suggestion for future work.

II. RELATED WORK

Transforming a world environment into a map for simulation and computer game is a problem of search space generation. The environment has to be transformed into a data structure that could are utilized by the agents. Graph data structure is used and it forms the fundamental description of search space. The search space could be visualized in various forms such as grid, triangle, hexagon and others. Several approaches used to transform the game world environment in computer game domain have been described by Rabin, Stout and Tozour [5-7]. Rabin [5] described several approaches of search space, and has pointed out the advantages and disadvantages of each. Stout [6] examined the solution for path planning and describes many techniques for partitioning the environment into search space. In addition, Tozour [7] explained various types of search space representation and discussed the advantages and disadvantages of each type of search space.

Grid base search space representations are often found and used widely in many fields such as video games and robotics. With respect to the grid approach, Yap [8] discussed a variety of grid representations and he also introduced texes that have the advantage of hexes. Bjornsson et al. [9] compared four different grid topologies and analysed the result of A* and IDA*search algorithms that were tested on the different grid representations. Although Grid-based search space is easy and simpler than other approaches, the major problem of grid is memory usage. Visibility Graphs approach was introduced by

Lozano-Pérez and Wesley in [10]. The Visibility graphs use the corners of the environment and the obstacles as vertices and all of them are connected by straight line provided the lines are not blocked by any obstacle. This approach is not as popular as the grid approach, but the benefit of visibility graph is the reduced memory requirement. However, in cases where the environments contain a large number of obstacles with complex shapes, they could result in a huge number of edges leading to highly complex search space and adverse effects on the computational time. Tozour [7] introduced two methods related to the space filling volume technique that could be used for generating the search space. The first is to place a "seed" into the environment and then expand the seed until it reaches an obstacle or the boundary. This volume forms the walkable area. The second method is to join the adjacent walkable regular grids into a larger grid thereby reduce the memory requirements. However, the space-filling volume approach has some disadvantages with respect to the handling of non-aligned geometry, complex obstacles and they are not guaranteed to completely fill all the areas. Heckel et al. [11] has improved and introduced the adaptive space filling volume approach to generate search space in the form of navigation mesh that could handle complex game map environments and they are also simpler than the triangulation-based approach. Flood filled technique is adapted from the paint technique in computer graphics domain. In addition, there are similarities with the space filling volume technique as regard to the ways that the seeds are being placed in the maps. Bjornsson and Halldorsson [12] also used the flood filled technique to generate the search space. The advantages of flood filled are its ability to generate graphs with more fine details and it could also handle difficult problems better than the space filling volumes approach and in particular, non-axis-aligned geometry. Navigation mesh is often used in 3D games in particular first person and third person views. There are various forms of navigation mesh such as rectangles and triangles. Creating search space in the form of triangulation could be done by decomposing the game world into triangular regions. The Hertel and Mehlhorn [13] algorithm could be used to generate navigation mesh by connecting each vertex in the world geometry into a set of triangles. Demyen and Buro [3] introduced Triangulation A* and the Triangulation Reduction A* technique to provide an abstract of the environment using constrained Delaunay triangulations. However, dealing with non-polygons such as circular objects might be difficult. Some recent studies [14-17] have used hierarchical approaches to improve computational search time by reducing the search space into abstracts of hierarchical levels commonly known as abstraction. Harabor and Botea [16] introduced the Hierarchical Annotated A* approach that has some similarity with the proposed approach in this paper as regard to the clearance value between the obstacles. However, the clearance value algorithm is different in this paper that will be explained subsequently. Optimizing the search space is necessary in order to improve the efficiency of the search algorithm and to reduce the workflow of search algorithm. The next section details the generation of the search space.

III. Generate Search Space

Using flood-filled technique to generate search space is more effective for area coverage and has the ability to generate graph with more detail. The algorithm used to generate search space is adapted from flood-filled technique used in computer graphics. However, for fine grain graph problem, it requires more memory storage and computational time. In order to reduce the computational time to handle complex graph, the hierarchical approach is proposed here.

A. Hierarchical Adaptive Flood Filled

In this paper, the propose Hierarchical Adaptive Flood Filled (HAFF) technique is adapted and enhanced from the flood filled technique. This technique is easy to implement by scanning through the entire binary image game map to determine walkable area in order to generate the search space. This algorithm works by finding the walkable areas and then adds the nodes into set of nodes in graph. There are some modifications to the conventional flood filled technique in order to generate the hierarchical search space in the proposed HAFF. The algorithm shown below is named as search Minimal Clearance Value algorithm, used to compute the HAFF level, which will be used by the HAFF algorithm later.

Algorithm: searchMinimalClearance

```
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

B. Minimal clearance value

A search minimal clearance algorithm shown above 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 used to store the clearance values. The clearanceSpaceCount is used for counting the length of the clearance values. The searchMinimalClearance algorithm lines 1 to 12 and lines 13 to 24 are procedure that read all the location in the map with both x and y axes. The 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.

Algorithm: ComputeLevel

Input: minimal_clearance_value ;

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

Output: HAFF_Highest_level

C. Determine the highest HAFF level

The ComputeLevel algorithm as shown above is designed to compute the level of HAFF by using the minimal clearance value as an input. Lines 1 to 6 of the ComputeLevel algorithm is used to compute the level of HAFF by comparing the minimal clearance value with 2^n+1, where 2^n+1 is size of frame node as shown in figure 1. Line 4 is used to update the HAFF level to the next level when minimal clearance value is greater than 2^n+1. The procedure will end when minimal clearance value is less than 2^n+1.

Algorithm: HAFF

Input: Binary_game_map Maps;

1: set MinimalClearance = searchMinimalClearance(Maps)
2: set HAFF_Highestlevel = computeLevel (MinimalClearance)
3: set HAFF_level = 0;
4: while HAFF_level <= HAFF_Highestlevel
5: for all location in Maps
6: if location.isWalkableArea() then
7: if location.canFramenodeFilled(HAFF_level) then
8: insert the nodes to HAFF Graphs[HAFF_level]
9: end if
10: end if
11: end for
12: validate all nodes and edges in Graphs [HAFF_level];
//BFS is used for validating all nodes and edges.
13: update HAFF_level to next level;
14: end while

Output: A set of hierarchical flood filled graph

D. HAFF Algorithm

The HAFF algorithm as shown above described the entire process of the proposed HAFF technique that used to generate the search space from binary image maps. The inputs of the HAFF algorithm are binary image maps and the outputs are search space or navigation graph that are generated from the maps. Line 1 of the HAFF algorithm is used to calculate the minimal clearance value, and in line 2, it computes the highest level of HAFF that are used for defining the number of levels that HAFF can generated. Line 3, HAFF_Level is set to 0. Lines 4 to 14 are the processes that used to generate the search space. In line 4, the HAFF_level is compared to HAFF_Highestlevel. When the HAFF_level is less than the HAFF_Highestlevel, the process will be continued to line 5s to 7. Lines 5 and 6 scan the entire map to determine walkable areas in each location of the map. In line 7 of the algorithm, walkable areas are checked to see if they can be filled with frame nodes. Each HAFF level uses different size of the frame node to build the search space. In line 8 of the algorithm, the nodes are inserted into HAFF graphs if those locations can be filled by frame nodes. A centre node in each frame node is inserted into the HAFF graphs when each HAFF graph has different level of HAFF. In line 12, it is used to verify all nodes and edges in the HAFF graph of each level to see if they can be explored further using Bread-First Search. In line 13, the process proceeds to the next HAFF level. When in the next level, the while loop in line 4 is repeated. The while loop in line 4 will terminate if HAFF_level is greater than the HAFF_Highestlevel.

E. Frame node characteristics

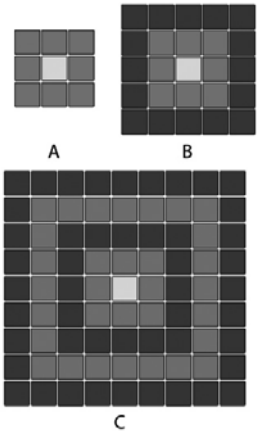

Figure 1. A, B, and C are Frame node in multi-size.

In figure 1, the illustrations of the Frame node that are used for building search space are shown. The Frame node has

multi-size square where the sizes of each frame node are defined by 2^n+1 where n = 1 to n by n vary according to the maximum number of HAFF level, as shown in figure 1, which represents the different frame node in each level of the hierarchy. The first level of HAFF uses the frame node size of 3x3 as shown in figure 1 (A). HAFF scans the entire game map using this frame node size in order to generate the search space by calculating the distance between the centre of the frame node to the centre of the frame, which is 2^n where n=1 to n. The second and third level will use the frame node as shown in figure 1 (B) and (C) respectively. The sizes of frame node varied according to the minimal clearance values in the map. Frame node will use only the centre node in each frame to create the reference node for the search space. In addition, for all the nodes surrounding the centre node can be used as a reference for determining the boundary of the AI agents. This could imply that the largest AI agent will use the largest frame in the map. All the nodes are also used for cross checking all the walkable areas in the lower level. Figure 2 illustrates an example of the detail of a frame node.

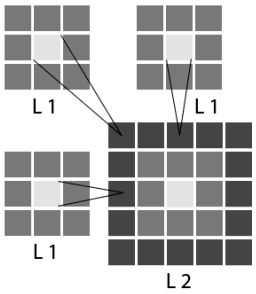

Figure 2. Relation between frame node size 1 and size 2 that are shown in figure L1 and L2.

Figure 2 shows the relationship between the frame node size 1 and frame node size 2. In figure 2 (L2), the centre node of the frame node is size 2, which in turn is surrounded by other frame nodes that use frame node size 1. Each frame node using size 1 is located at the border of frame node in eight directions from the centre of the frame node using size 2. Other similar patterns can exist by using frame node with larger size to holds other smaller size frame node.

IV. Experiment

In this section, experiments and tests are carried out to test the proposed HAFF technique. It is tested on a set of game map of 70 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, where some including arbitrary size of x x y. 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 categorised by the shapes of the obstacles appeared on the maps. There are a variety of obstacle shapes created in map including square, rectangle, triangle, polygon, circle, and free form. In addition, several obstacle shapes are mixed into a single map. All the maps were tested using the two techniques, which are normal flood filed and the proposed HAFF. Both algorithms are utilized to generate the search space from all of maps in the comparison experiment. The experiment is carried out on 2.0GHz Intel core 2-duo processor with 4GB RAM running OSX 10.6.4 using Xcode 3.2.2

V. Empirical Results

In this section, the experiment results for the different game maps generated using the two techniques are presented. The normal flood filled technique and the proposed HAFF technique are used to generate the search space 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 nodes and edges in each level of the HAFF, which are connected and could be searched by the search algorithm. Figure 3 shows two examples maps. Figure 3 (A) has the size of 300x500, which has 35.42 percentage of walkable area. Figure 3 (A2) has a size of 192x320 that has 74.10 percent walkable area. Figures 3 (B) and 3 (B2) are result that generated from normal flood filled (NFF) technique. Figure 3 (C) is the result from using the normal flood filled technique, and figures (D), (E), (F), (C2), (D2), and (E2) are the result from using HAFF technique. The image result of figures (D, C2), (E, D2) and (F, E2) are created by using HAFF level 1, HAFF level 2 and HAFF level 3 techniques respectively.

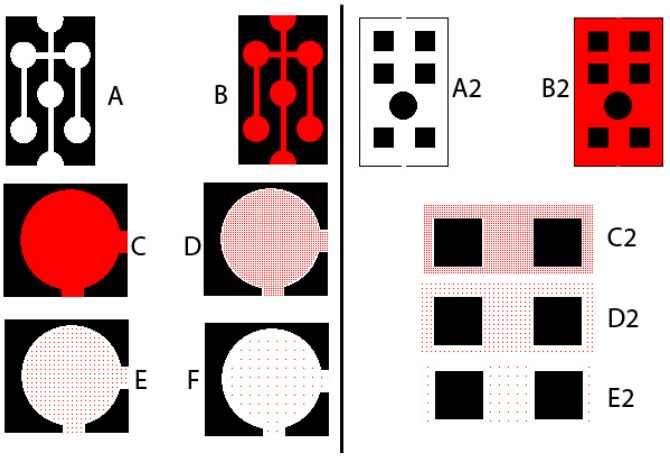

Figure 3. A and A2 are input maps; C is the result from Normal Flood Filled techniques; D, E, F, C2, D2, F2 are result from HAPP techniques.

TABLE 1. Result from NFF and HAFF

Map Size		Normal Flood Filled (NFF) and HAFF			
		NFF Level 0	HAFF Level 1	HAFF Level 2	HAFF Level 3
Map A 300x500	nodes	53129	12403	2891	593
	edges	207858	47296	10415	1809
Map A2 192x300	nodes	45526	10826	2576	598
	edges	178760	41608	9449	1950

Table 1 shows that the number of nodes and number of edges generated from using NFF and HAFF. In map A, the number of nodes generated from HAFF level 1 technique has decreased by about 4.28 times when compare with those generated from the NFF technique. In addition, the number of edges has reduced to 4.39 times. The number of nodes and number of edges generated from Map A by HAFF level 2 technique as shown in the table 1 have dramatically reduced by about 18.38 and 19.96 times respectively when compare with NFF techniques. Moreover, the outputs of HAFF level 3 have reduced to 89.59 and 114.90 times. Likewise, in table 1, Map A2 shows the number of nodes and the number of edges that are created by NFF and HAFF techniques. The results generated from HAFF technique have reduced dramatically when compare to NFF in each level of HAFF. From table 1, under the results of Map A2, the results generated from HAFF level 3 when compared to the NFF technique shows that the number of nodes and the number of edges have decreased by about 76.13 and 91.63 times respectively. In addition, both map A and A2 when compare the results of HAFF in each level have decreased by about 4 times or more. Next the results that are gathered from experiments are presented.

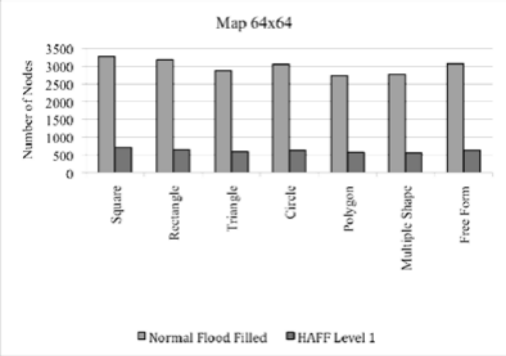

Figure 4. Number of nodes in Map 64x64

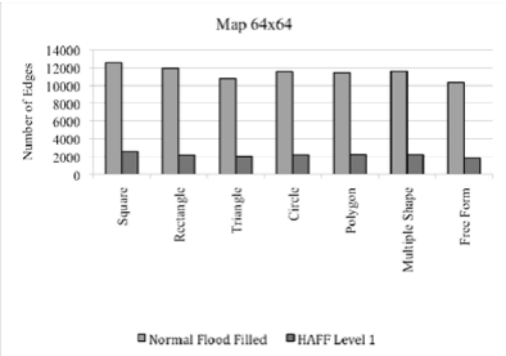

Figure 5. Number of edges in Map 64x64

Figure 4 and figure 5 shown the differences of the results between using NFF technique and using HAFF technique. The results of the number of nodes and the number of edges are shown in figures 4 and 5 respectively. The results are generated from the map size of 64x64, where each map contained different types of obstacles including square, rectangle, triangle, circle, polygon, multiple shape and free form as show in the x-axis of graph in the figures 4 and 5, i.e. if the map contains square, this map will has square shaped obstacle. For map that contains multiple shapes, it will hold different shape of obstacles in a single map. The walkable areas of all maps are more than 65 percent. The results gathered from both techniques are shown in Figure 4. The number of nodes generated from the HAFF technique is reduced by at least 4 times in each different map when compare to those generated from the normal flood filled technique. In addition, the number of edges as shown in figure 5 also decreased by 5.3 times on average in every maps. The highest level of HAFF that could be generated in the map size of 64x64 is level 1. Although the level of HAFF are defined by the size of frame node that involved minimal clearance value, maps size can also has an effect to the number of levels in HAFF. 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.

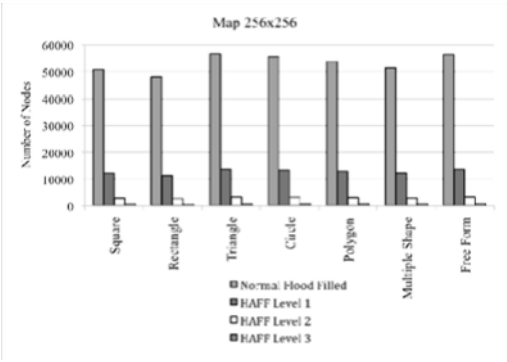

Figure 6. Number of nodes in Map 256x256

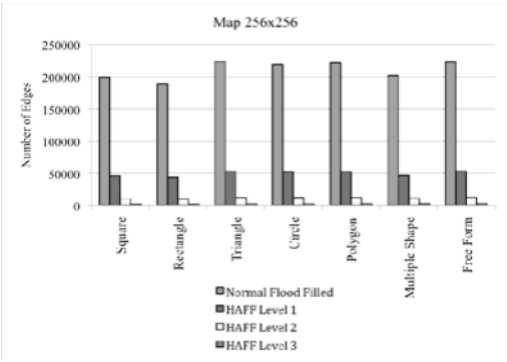

Figure 7. Number of edges in Map 256x256

Figures 6 and 7 show the results that are generated from maps size of 256x256, where each map includes different types of obstacles and the walkable areas are about 81.69 percent on average. The number of nodes and the number of edges that are generated from the NFF technique are quite big on each map. The number of edges from the results generated from using NFF technique is about 200,000 edges. When HAFF technique is utilized for the map of 256x256, it can attain 3 level of hierarchy. In each level of the HAFF, the

number of nodes and the number of edges has reduced significantly. When comparing the results of HAFF using 3 levels with NFF, the results can reduce by about 70.01 to 91.06 times in each map. The HAFF technique performs better on larger maps, as it can generate more level of hierarchy.

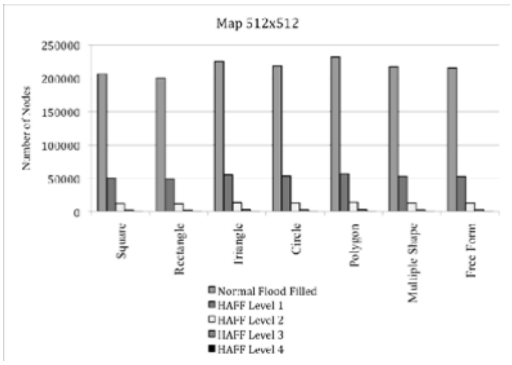

Figure 8. Number of nodes in Map 512x512

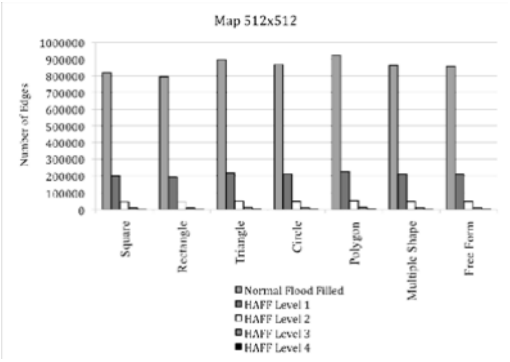

Figure 9. Number of edges in Map 512x512

The results show in figures 8 and 9 are the number of nodes and the number of edges that are generated from the map size of 512x512, and the walkable areas in each map are 82.67 percent on average. The highest level of HAFF can reach level 4 in this case. The search space size for this larger map is huge. Figures 8 and 9 show that the number of nodes is about 200000 nodes on average, and the number of edges is over 800000 edges in each map. Searching in these space could take a long time even efficient search algorithm is employed. HAFF is utilized in order to reduce the size of search space. The number of nodes and numbers of edges generated from HAFF level 4 when compare to those from the NFF has dramatically reduced by about 297.77 and 342.85 times on average. All the results from the experiment show that the proposed HAFF technique could reduce search space size. From the observation, it is deduced that each level of HAFF can support different AI agents' sizes.

VI. CONCLUSIONS

In this paper, a hierarchical adaptive flood filled technique is presented. The proposed technique can improve the search space generation from an environment. The HAFF technique is simple to implement. The results generated from HAFF technique could dramatically reduce the number of nodes and number of edges when compared to the NFF. This could help to save the memory usages and improve the time for computing the paths. HAFF can provide better result on larger maps. Moreover, it could improve the search time and support multi-size mobile AI agents. AI agents that have the biggest size could use the HAFF graph at the highest level, and the AI agents that are smaller size could use the HAFF at lower level as a guide for pathfinding. The proposed HAFF could handle a variety of obstacle types such as polygonal obstacles, circular obstacles and free form that are in the maps. In future, improvement to the proposed HAFF technique to handle grey scale and color images will be investigated.

REFERENCES

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

[2] S. Koenig, "A Comparison of Fast Search Methods for Real-Time Situated Agents," in Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2 New York, New York: IEEE Computer Society, 2004.

[3] D. Demyen and M. Buro, "Efficient Triangulation-Based Pathfinding," in AAAI, 2006, pp. 942-947.

[4] S. Rabin, "A* Speed Optimizations," in Game Programming Gems 1ed. vol. 1, M. DeLoura, Ed.: Charles River Media, Hingham, MA, USA, 2000, pp. 272-287.

[5] S. Rabin, "A* Aesthetic Optimizations," in Game Programming Gems 1ed. vol. 1, M. DeLoura, Ed.: Charles River Media, 2000, p. 600.

[6] B. Stout, "The Basics of A* for Path Planning," in Game Programming Gems 1ed. vol. 1, M. DeLoura, Ed.: Charles River Media, 2000, pp. 254-263.

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

[8] P. Yap, "Grid-based path-finding," in the Canadian Conference on Artificial Intelligence, 2002, pp. 44-55.

[9] Y. Bjornsson, M. Enzenberger, R. Holte, J. Schaejfer, and P. Yap, "Comparison of different grid abstractions for pathfinding on maps," in Proceedings of the 18th international joint conference on Artificial intelligence Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003.

[10] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," Commun. ACM, vol. 22, pp. 560-570, 1979.

[11] F. W. P. Heckel, G. M. Youngblood, and D. H. Hale, "Influence points for tactical information in navigation meshes," in Proceedings of the 4th International Conference on Foundations of Digital Games Orlando, Florida: ACM, 2009.

[12] Y. Bjornsson and K. Halldorsson, "Improved Heuristics for Optimal Pathfinding on Game Maps," in American Association for Artificial Intelligence, 2006.

[13] S. Hertel and K. Mehlhorn, "Fast triangulation of the plane with respect to simple polygons," Inf. Control, vol. 64, pp. 52-76, 1985.

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

[15] A. Botea, M. Muller, and J. Schaeffer, "Near Optimal Hierarchical Path-Finding," Journal of Game Development, vol. 1, pp. 7-28, 2004.

[16] D. Harabor and A. Botea, "Hierarchical Path Planning for Multi-Size Agents in Heterogeneous Environments," in Symposium on Computational Intelligence and Games, 2008, pp. 258-265.

[17] 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.