Pathfinding : HPA^* in StarCraft

Adrien Bodineau, Santiago Ontañón and Alberto Auriarte
Computer Science Department
University of Nantes, France
Drexel University

Abstract

Standard algorithms such as A^* are not adequate for pathfinding in Real Time Strategy (RTS) games, because of the tight time constraints the Artificial Intelligence (AI) of these games are subject to. In this paper we will present the outcome of integrating HPA^*[1], a hierarchical variant of A^* into *StarCraft*, a popular RTS game. We describe both A^* and HPA^*, and how were they integrated into StarCraft. Then we will present experimental results, and finally, present directions for future work.

I. Introduction

In RTS game AI, one of the main challenges is the tight time constraints under which the AI needs to execute. This is because RTS games are *real time* games, which means that the AI has to make all its computation in a very short amount time.

Algorithms exists, designed to reduce the time required by the AI to perform certain tasks, or to save in some computation time in order to allocate this time to other computations. In this paper we will present our experimental results of using the HPA^* algorithm for pathfinding, in the popular RTS game : StarCraft Broodwar. HPA^* is a variant of A^*, designed to significantly reduce computation time, at the cost of not ensuring optimal paths.

Section II introduces RTS games and A^* pathfinding. **Section III** introduces HPA^* pathfinding. **Section IV** present BroodWar Terrain Analyzer (BWTA) and how, using the c++ library BroodWar API (BWAPI) which provide an interface with StarCraft, and BWTA, a map analyzer for StarCraft in c++, we have integrated the algorithm. Then, **Section V** explains results of our experiments. Finally, **Section VI** is a brief conclusion with ideas too improve our results.

II. Background

RTS games

Contrary to classic board strategy games, like chess, RTS games are significantly harder from an AI point of view. In particular, the fact that they are *real time* means that both players could make actions simultaneously, and have to make decisions very quickly. Moreover, most RTS games are also *partially observable*, meaning that players can only see parts of the map (this is called $\ll$ fog of war $\gg$). But, in particular, the difference on which we will focus on is the size of the maps. In RTS games the size of the maps could potentially be very large, which means that analyzing the map, or finding a path between two locations may take a significant amount of time and resources. Finding a better way to compute shortest paths could save time for other AI tasks, making the AI stronger.

In this paper, we focus our study on the game StarCraft Broodwar, a RTS game made by Blizzard, which is very popular and has raised as the standard tesbed for AI in RTS games.

Pathfinding with A^*

The A^* algorithm is a variant of Djikstra, and can be used to find optimal paths. Actually, it's the standard algorithm for pathfinding, used in many commercial games.

This algorithm uses a heuristic to choose the better node to explore in a list of possible nodes called open list. Nodes that are already explored are insert into a list called closed list. While the open list is not empty we explored the neighbor nodes of the current one, if a node is not in both open and closed list, we add it into the open list and we change its parent node by the current one. At the end, if a path exists, it can be constructed by taking the parent of target, and the parent of its parent until the parent is the start node.

The **Figure 1** show an example of path find with A^*, the empty circles represent the nodes in the open set, those that remain to be explored, and the filled ones are in the closed set. Color on each closed node indicates the distance from the start : the greener, the farther.

III. Pathfinding with HPA^*

The HPA^* algorithm is a variant of A^* presented by Adi Botea, Martin Mller and Jonathan Schaeffer. It works in the following way. First the graph over which we are going to perform pathfinding is divided in clusters, group of neighboring nodes. You can see an example of cluster on **Figure 2** using the same map as **Figure 1**. Then, we construct a high-level graph where each node is a cluster, then we use

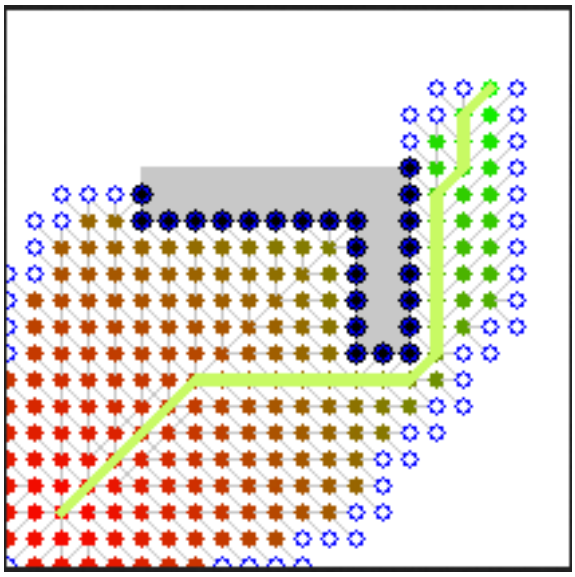

FIGURE 1 – Source : A^* example from the English version of A^* wikipedia.

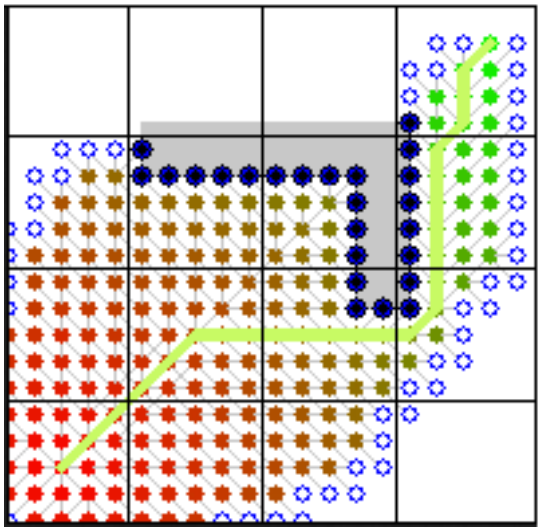

FIGURE 2 – Clusters are represented by black squares of 5×5 nodes. Source of original picture : A^* example from the English version of A^* wikipedia.

A^* to find the path from the start cluster to the target cluster, as you can see on **Figure 2** instead of searching a path in a graph with about $4^2 * 5^2 = 400$ nodes, we will use the high-level graph which only have $4^2 = 16$ nodes.

Before we continue, we have to define what we will called "entrance" in our situation. An entrance is a set of nodes you can go through in order to come in/out of a cluster, in our case, entrances are the Chokepoints [1], because they are the only way to navigate between region, and the particular point we will focus on is the center of Chokepoints, because BWTA provide a function to have it.

Finally, to find our path we can use A^*, we compute distance from start/target to each entrances of the cluster, and for all other clusters in the path we compute the path from each entrance to others in the cluster. This method makes a significant improvement on the number of nodes visited, making pathfinding much faster at the price of not guaranteeing finding the optimal path. To be closer to the optimal path, the algorithm includes a refinement step, to improve the final path.

IV. Integration into StarCraft

BWTA

The standard way to interface with StarCraft is via BWAPI, a C++ library with which AI for StarCraft can be developed. A common addition to BWAPI is BWTA, which contains functionality to analyze a map and give some information. After analysis. BWTA can construct an high-level graph, the one you can see in **Figure 3** in which blue nodes are region, red nodes are the center of Chokepoints and red lines are pixels parts of the Chokepoints.

1. A Chokepoint is a point between two zones you must go through in order to pass from one to the other

With this graph BWTA could make an abstraction of the map into a set of Regions (that can be used as the clusters required by *HPA**) and Chokepoints (that can be used as the entrance of regions in *HPA**) like we can see in **Figure 4**. It also give us access to the tile position of the map which are in four categories : pixels, walk-tiles (4×4 pixels), build-tiles (16×16 pixels) and Position units (32×32 pixels).

How did we did the integration

We integrated the *HPA** implementation into a custom version of the BWTA library. When BWTA is asked to analyze a map, it proceeds as follows :

— It check if the map has already been processed before (by checking if a file with the analysis data already exists)
— If it exists, load the data from disk.
— Otherwise, analyze the map then save the results to a file for future use.

We made a function called buildChokeNodes which computes the cost and the path between all Chokepoints on the map. Then we called this function during the analyze process of BWTA, and also modified the load and save procedures to load and save this information together with all the other information saved by BWTA, so we don't have to recompute all those information for a map we have already analyzed before. Then, when a path using *HPA** needs to be generated. We compute a path from the start to the end regions, using the precomputed paths between each chokepoint. Then, we called A^* to find a path from the start and end position to the first and last chokepoint in the path between the regions. Finally, we made a function which constructs the complete path with Tile positions, that are the units used for pathfinding, by concatenating precom-

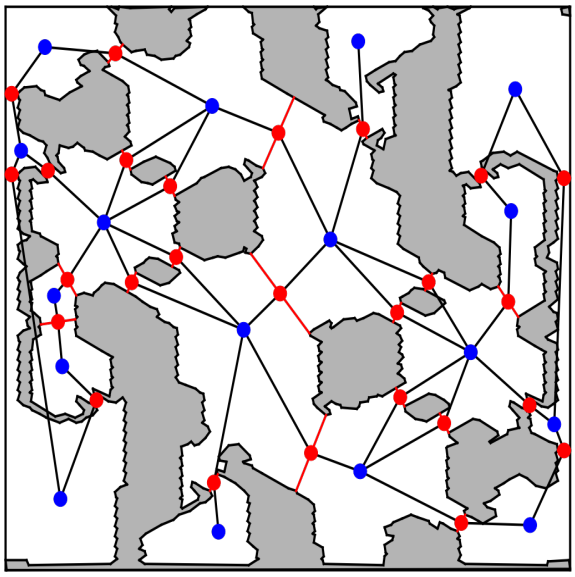

FIGURE 3 – High-level graph of the map. Region nodes in blue, Chokepoint nodes in red

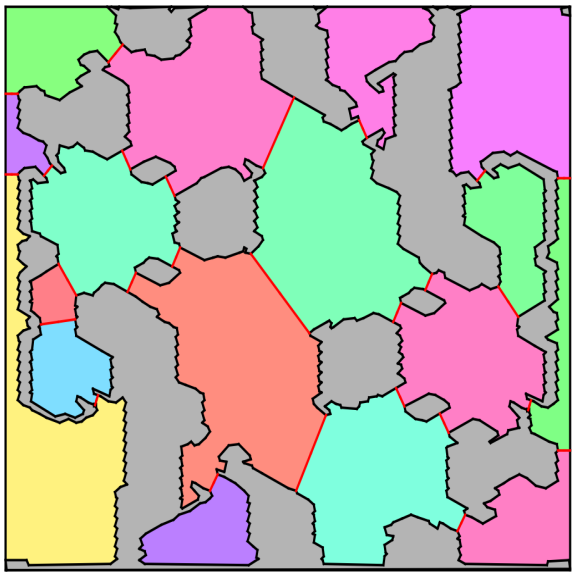

FIGURE 4 – Region of the map represented by colored area, and Chokepoints represented by red lines

puted path between Chokepoints and computed path from start/target Tile position to the Chokepoints.

V. Our results

We made tests on a map called Benzene, you can see it on the **Figure 5**. The started locations on the map are in the upper-right region and lower-left region. The x-axis goes from the left to the right and the y-axis goes from the top to the bottom. For our experiment we always take the blue point in the lower-left corner as start point for our path. On this figure you could also see an approximation of the position of the point we took, all points exept the point for the length of 40 are located aproximately on the red line.

For each path we compared the times of A^* and HPA^*. We computed each path 500 times to have a good average of the time.

As we can see when we compare **Figure 6** and **Figure 7**, HPA^* is really fast compared to A^*, and really stable in the time of execution. Now we will see if the path is still good or not.

As we can see in **Figure 8** the refinement make the path closer to the optimal but the difference of length between the A^* and HPA^* grows up. This is explain by the fact that we always cross Chokepoints througth the center so we are not completely optimal.

VI. Conclusion

In this paper we have presented and compared HPA^* and A^* and we have seen that HPA^* provide a really good and interesting improvment in time without loosing too much in optimality. However, we could improve the optimality of the path by computing better path between each Chokepoint. Since all those paths are computed during the off-line analysis, we should be able to improve the optimality *without* making the total time of finding a path longer.

This new pathfinding algorithm could be very interesting for RTS game-AI, because all the time saves with HPA^* could be used for other computations.

Références

[1] Adi Botea Martin Mller Jonathan SCHAEFFER. *Near Optimal Hierarchical Path-Finding*. URL : `http://www.cs.ualberta.ca/~mmueller/ps/hpastar.pdf`.

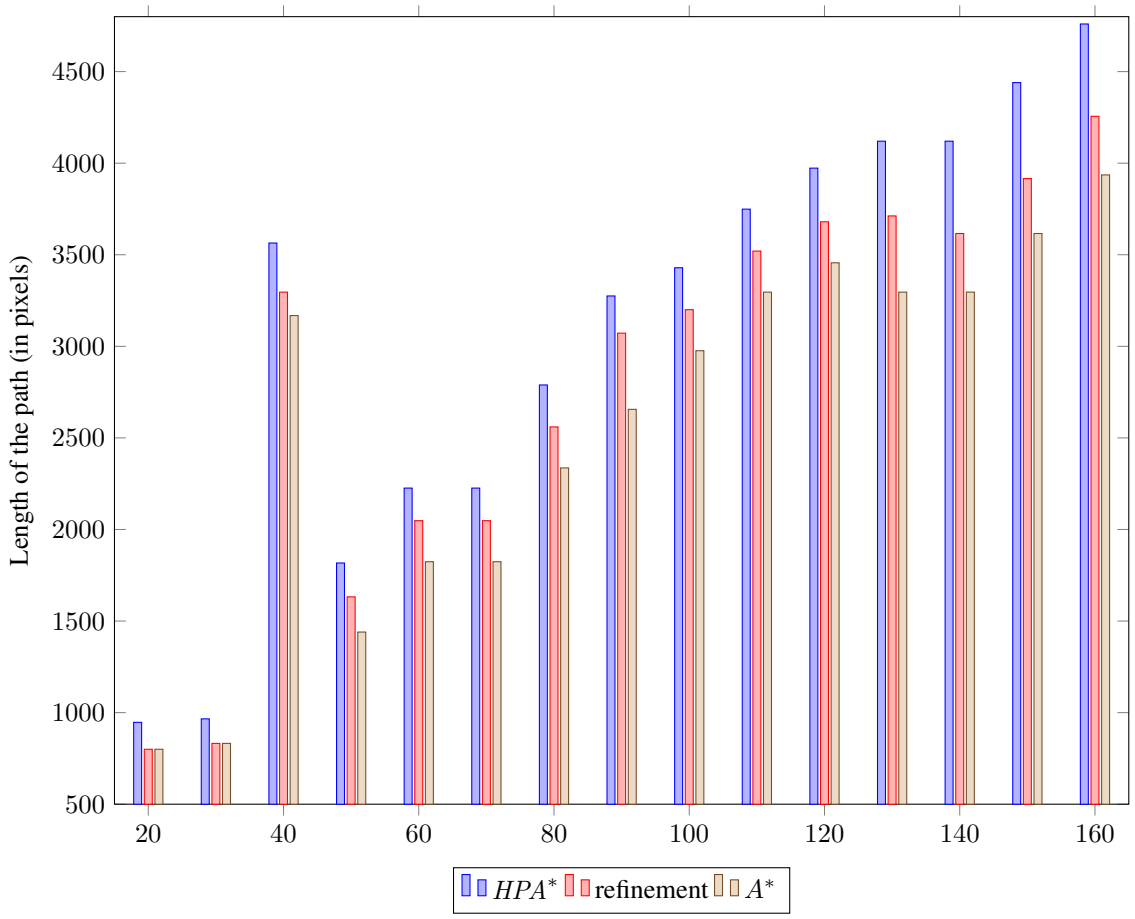

FIGURE 8 – comparison of length of the path for HPA^* before and after refinment, and for A^* in term of pixel

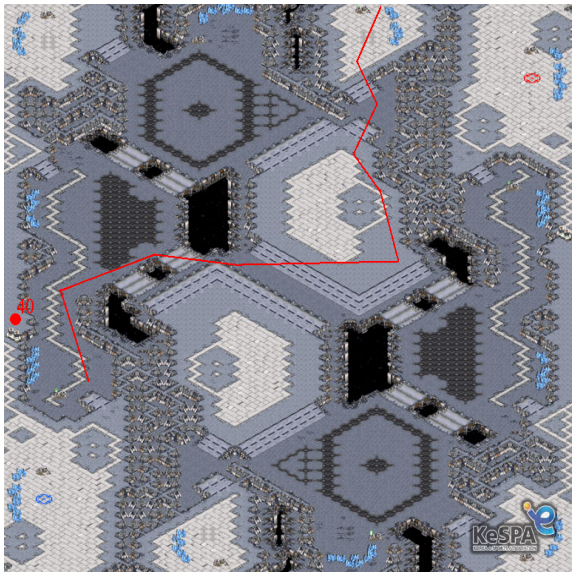

FIGURE 5 – Benzene map, the red line is an approximation of the location of points we took, exept for the length 40. Source of original picture : liquipedia BroodWar (http ://wiki.teamliquid.net/starcraft/Benzene)

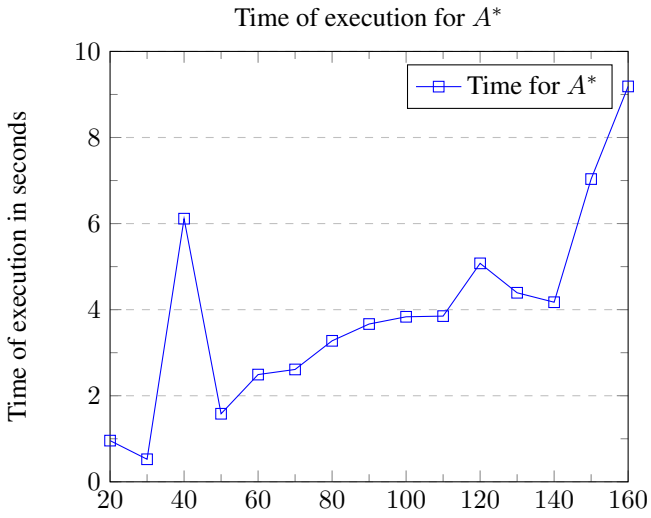

FIGURE 6 – Time of execution of A^* depending on the length of the path in term of Tile position

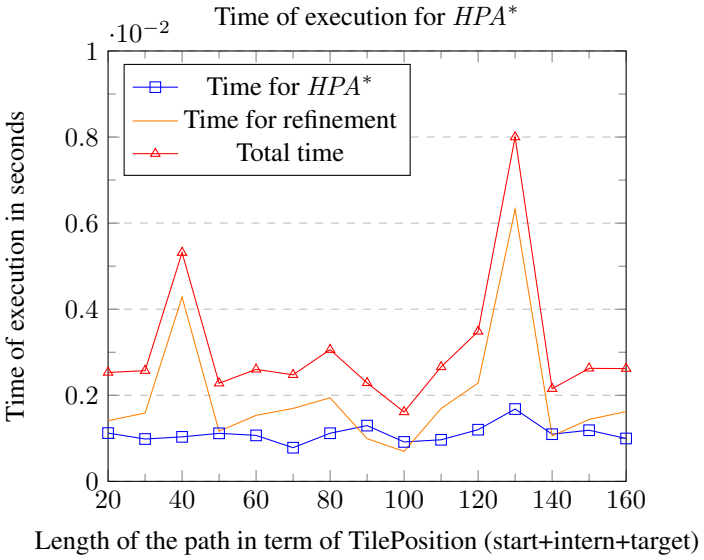

FIGURE 7 – Time of execution of HPA^* depending on the length of the path in term of Tile position