Real-Time Pathfinding Optimization in Strategy Games using Hierarchical Pathfinding A*

Benedictus Nelson - 13523150
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: benedictus.nelson@gmail.com , 13523150@std.stei.itb.ac.id

Abstract—Pathfinding is a very helpful feature of artificial intelligence (AI) in real-time strategy (RTS) games; however, many pathfinding methods can perform poorly on larger maps with hundreds of units. The A* algorithm is guaranteed to find the shortest path, but it is less than practical to use due to the amount of time it needs to compute that optimal path. Hierarchical Pathfinding A* (HPA*) was developed to help manage those large map and time complexity problems by phasing out the map and reworking it to find clusters based off a high-level hierarchy. The standard HPA* algorithm has an effective grid-based clustering method, which works well when clustering relatively homogeneous topologies; but is highly inefficient when mapping heterogeneous topologies. In this paper, we present an optimization method called Adaptive HPA* (AHPA*). The AHPA* optimization replaces static clustering with an adaptive segmentation of the map, which utilizes the Region Growing algorithm. The AHPA* approach generates different clusters (in shape and size) that adhere to the topological structure of the map, which generates clusters resulting in a smaller, clearer and more intelligent abstract graph. Experimental comparative analysis shows that the number of nodes searched through, and the time required to search through those nodes reduced significantly with AHPA*, with only a small degradation in path quality. This pattern extends to maps with largely open areas and narrow corridors.

Keywords—Pathfinding, A*, Hierarchical Pathfinding (HPA*), Real-Time Strategy (RTS) Games, Map Segmentation, Region Growing.

I. INTRODUCTION

Pathfinding is a crucial part of developing artificial intelligence (AI) for different video game genres, especially in real-time strategy (RTS) games. The ability for an AI to intelligently and efficiently move a large number of units from one location to another, is a technical issue as well as an issue affecting players' fun. If a pathfinding system does not quickly find a reasonable path for movement, the result can be unrealistic unit behaviour or players experiencing frustrations.

The challenge in implementing pathfinding in today's RTS games lies in the size and the complexity of the game world. Games like StarCraft, Age of Empires and the like can have large maps (which can be represented as thousands by thousands of grid size tiles), while managing hundreds of units at a time. The complexity of large maps, a larger number of units, a dynamic game-world (when buildings are built or destroyed), and the fog-of-war sets the stage for a space explosion. These combinations make common pathfinding algorithms extremely inefficient, if feasible at all, when facing strict real-time requirements.

The A* algorithm has been the standard for pathfinding in the gaming industry for some time. Although it was introduced by Hart, Nilsson, and Raphael in 1968, it ensures an optimal (least-cost) path will be found if the heuristic function is admissible. As a note of caution, optimal does not also mean fast. On large RTS maps, launching A* at the grid level for each unit can demand considerable processing and ultimately sacrifice frame rate or responsiveness.

In order to limit these issues, hierarchical search techniques were created. One of the most recognized and influential implementations of hierarchical search is Hierarchical Pathfinding A* (HPA*), which was originally proposed by Botea, Müller, and Schaeffer in 2004. HPA* builds an abstraction of the map across multiple levels, where the detailed low-level map is clustered into groups, resulting in an abstract graph with many tens or hundreds rather than thousands or million nodes. HPA* then applies pathfinding to this abstract graph, which mitigates the search space and can provide a speed up of 10x even for the most optimized A* implementation. This has represented a paradigm shift in game pathfinding, whereby absolute optimality is given up (HPA* paths may be up to 1% longer than the optimum) but runtime speed is increased significantly, which is far more relevant to real-time gameplay.

While HPA* is a notable development, it has an inherent flaw in its abstraction. Canonical HPA* uses a uniform grid partition, meaning that the map is divided into square clusters of the same size (e.g. 10x10 tiles). The "one-size-fits-all" strategy is inefficient on maps with heterogeneous topologies. In large open areas, the uniform partition creates far too many small clusters, yielding a more dense abstract graph than it needs to. In highly complex areas with a more prescriptive number of narrow corridors (choke points), fixed clusters may not provide the appropriate resolution to represent the topology accurately. The implication of this flaw is that hierarchy alone does not suffice; rather, the real motivator to greater performance is the quality of the abstraction itself. The

next phase of HPA* must have a more intelligent and context-aware abstraction.

In light of this research gap, this paper proposes a method of HPA* optimization we call Adaptive HPA (AHPAhalda. The main contribution of AHPA* is that it replaces the uniform mechanism used to produce clusters with a topology-based, adaptive map segmentation process. AHPA* uses the Region Growing algorithm to produce clusters of different shapes and sizes and to naturally follow the traversible area contours. Our hypothesis is that AHPA* will produce better abstract graphs due to the cluster and abstract graph sizes. Resolving larger clustering in open areas and smaller, more detailed clustering in areas of complexity, AHPA we expect to leave more compact and efficient abstract graphs. In turn, AHPA* expects to reduce online search time exceedingly when compared to standard HPA* while providing solution paths of near-optimal quality.

The paper is structured as follows: Section II introduces the theoretical rationale behind the A* and standard HPA* algorithms; Section III provided the new methodology of AHPA*. Section IV describes the experimental design and the comparative tests results, as well as their interpretation. Finally, in Section V, conclusions from this research are developed with outlines for future development.

II. THEORETICAL FOUNDATION

This chapter describes the foundational concepts behind the proposed method. In order to recognize the improvement and originality offered by AHPA*, it is important to understand how the A* algorithm works and how traditional HPA* operates.

A. A* Algorithm

A* is a graph search algorithm that is considered one of the best single-destination pathfinding algorithms. A* is a best-first search algorithm since it always picks the most promising node to evaluate next. The idea of a node being "promising" is evaluated using a heuristic function.

In a formal definition, A* evaluates each node n with the following function:

$$f(n) = g(n) + h(n)$$

Where:

- $g(n)$ is the actual (accumulated) cost to reach the node n from the start node. This value is computed and stored during the search process.
- $h(n)$ is the estimated cost (heuristic) from node n to the goal node. This heuristic function is an "educated guess" from human experience and is reported, based on the knowledge of the problem, rather than being calculated. Commonly used heuristics for grid maps are the Manhattan Distance or Euclidean Distance.

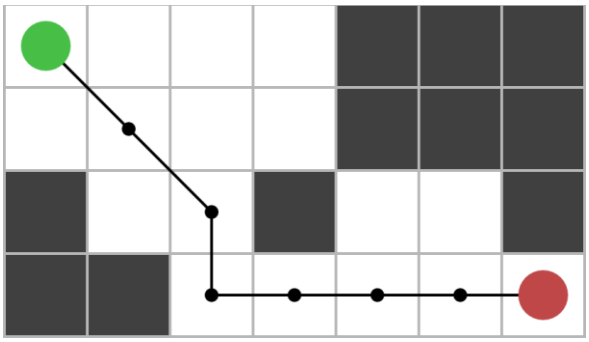

Figure 1. *Illustration of an optimal path found by the A* algorithm*

A* keeps track of the two sets of nodes, an Open List (a list of nodes that have been discovered but not yet evaluated) and a Closed List (a list of nodes that have been evaluated). The search is repeated by taking the node with the lowest f(n) value from the Open List, evaluating its neighbors, and updating the route when a better route is discovered.

The most important part of A*'s unique advantage is its optimality guarantee. Hart, Nilsson, and Raphael's, (1968) original paper showed that A* will find the least cost path if the heuristic function h(n) that it uses is admissible meaning that it never overestimates the real cost to the goal (an underestimating heuristic). If the heuristic is also consistent (or monotonic), A* will find the optimal path without having to re-process those nodes that are already on the Closed list so it is more efficient.

B. Hierarchical Pathfinding A* (HPA*)

HPA* is a faster version of A* on very large grid-based maps through hierarchical abstraction. HPA* performs two phases: a resource-heavy work we call preprocessing that is performed only once offline, then in-phase that is performed quickly repeatedly online or in-real-time.

1) Preprocessing Phase (Offline)

The goal of this phase to build an abstract graph representation of the low-level map. The following are steps of this process:

- **Map Partition**: The low level grid map, partitioned adjacent non-overlapping clusters. Each cluster in standard implementation is a square shape of fixed size, for example, tiles that are 10x10 or 15x15.
- **Entrance Identification**: For each pair of neighboring clusters, the algorithm identifies every sequence of traversable tiles traversing their common border. Each continuous sequence of tiles is identified as an "entrance."
- **Abstract Graph Construction**: A level-1 abstract graph is constructed. Each node corresponds to the entrance identified. Two kinds of edges exist in this abstract graph.

- **Intra-edges:** For each pair of entrances in the same cluster, connect them with directed edges. The weight of the directed edge is the cost of the optimal path to travel between the entrances on path (intra-cluster edge). This is pre-calculated using A* on the low-level grid in the cluster and stored in a cache for future use.
- **Inter-edges:** Connect the opposing entrances of two neighboring clusters with directed edges. These edges represent the movement allowed from one cluster to another and usually have a low weight (1).

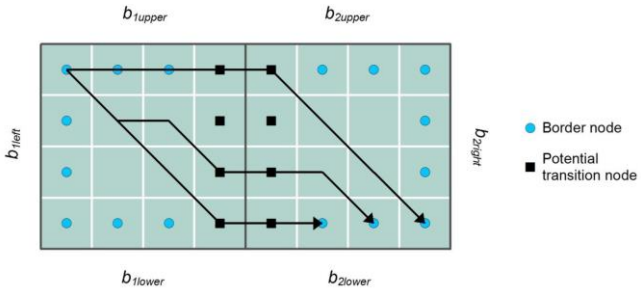

Figure 2. *Abstract Graph Construction in HPA.*

The HPA* approach introduces a classic trade-off - it invests a lot of computation time upfront during the preprocessing phase to generate and (cheaply) store local path information; this investment in offline computation "pays off" during the online computation, where in the case of an online search on a much smaller abstract graph much time is saved. HPA* provides the foundation for a suite of other hierarchical algorithms like DHPA*, which addresses dynamic environments through selective caching, or SHPA*, which offers a more sophisticated clustering scheme for more static environments. The contribution proposed in this paper is particularly an innovation along the "clustering" axis to enable HPA* to operate more efficiently on different types of maps.

2) Search Phase (Online) When HPA*

When HPA* receives a pathfinding request from a point start, S to goal, G. HPA* does the following steps:

- **Temporary Insertion of S and G:** We insert S and G into the abstract graph as temporary nodes. We make temporary edges from S to all of the entrances in S's cluster. We also make temporary edges from all the entrances in G's cluster to G. We use low-level A* to determine the weights of the edges on-the-fly.
- **High-Level Search:** We run A* on the expanded abstract graph to find the lowest-cost path from S to G. We get a sequence of entrances to pass through.
- **Path Refinement:** The abstract path is then refined into an executable low-level path. For each intra-edge in the abstract path (movement within a single cluster), the low-level path is retrieved from the cache created during the preprocessing step. Collectively, all of these low-level paths will give us a final path from S to G.
- **Path Smoothing (optional):** The final path from S to G will often look "blocky" or unnatural. Optionally, a smoothing process can be applied to smooth the path. An example of this would be connecting distant nodes on the final path with a straight line if there are no obstacles to cross between these nodes.

By limiting the expensive A* search to only small, local segments and executing the main search on a small high-level graph, HPA* shifts the performance bottleneck experienced by standard A* on large-scale maps.

III. PROPOSED METHOD: ADAPTIVE HPA* (AHPA*)

A. Motivation and Concept

As previously described, standard HPA* divides the map into a homogenous grid of fixed-size clusters; from a development perspective, it's a very straightforward implementation, but it doesn't use available topological information from the map itself. Consider a standard RTS map that has a large and open base area, and several strategic choke points/narrow corridors. Standard HPA* "blindly" divides the area into several times too many small clusters in the large open area with multiple entrance and intra edges, and this creates unnecessary complexity in the abstract graph, with more nodes and edges in the higher level A* to evaluate, which reduces its overall performance.

In contrast, an alternative way to 'partition' areas of a map into clusters would be to treat the large open space as a single logical element, and make it a single large cluster in the abstract graph. Likewise, complex dun under cover areas that have dense obstructions could be split into small clusters to maintain their topological representation.

AHPA* is suggested based on the following hypothesis: the abstract graph will be smaller and less complex and more directly affect online searching times and fewer node expansions by creating clusters that conform to the topological structure of the map such that homogeneous, traversable areas have large clusters, whereas complex areas have small clusters. We are suggesting that there will be no meaningful degradation to the overall path quality at the higher level down to the graphical HPA* search.

AHPA* can be viewed as the HPA* generalization. If standard HPA* is “top-down” approach where grid structure is imposed on the map, then AHPA* is the opposite where the map’s structure itself determines the shape and size of the clusters.

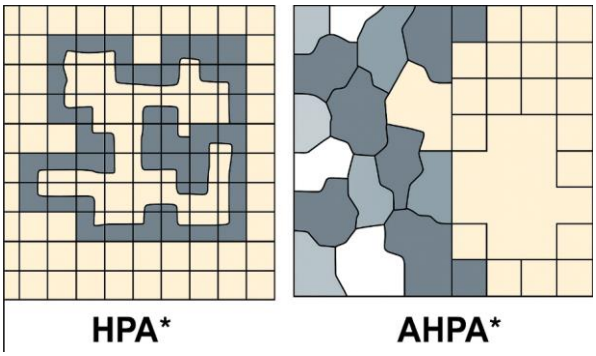

Figure 3. *Illustration of Grid Partitioning in HPA and AHPA*

B. Adaptive Map Segmentation with Region Growing

The main innovation of AHPA* is in preprocessing, and in how the map is segmented. AHPA* uses an image segmentation algorithm to combine traversable tiles into topologically-consistent groups, instead of just applying labels to a static array. The algorithm we decided to use is called Region Growing.

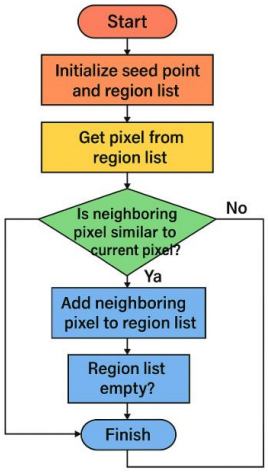

Figure 4. *Region Growing Flowchart*

Region Growing is a good fit because how it works fits our purpose, as it applies a label to all the connected pixels (in this case map tiles) that have the same property. The most relevant property for pathfinding, in this case, is "traversability." The algorithm finds all connected explorable areas.

In AHPA* the second phase, segmenting the map using Region Growing, proceeds as follows:

- Initialization: Make a marker array (i.e., visited_tiles) of the same dimension as the map and set all of its values to false. Make an empty array to hold the final clusters.
- Iterate the map: For each tile (x, y) on the map, repeat the following:
    - If (x, y) is an obstacle (a tile with an assigned label) OR if (x, y) is marked visited, skip to the next tile.
    - If (x, y) is traversable and has not been marked visited tile, it is a seed for a new cluster.
- Region Growth (from Seed Points):

a. Make a new cluster and add the seed point to it. Mark it as visited.

b. Create a queue (we used some sort of First-In-First-Out implementation queue), and add the seed point to it.

c. While the queue is not empty:

i. Dequeue a tile current_tile from the front of the queue.

ii. For each neighbor neighbor_tile of current_tile (using 4-way or 8-way connectivity):

* Check Growth Criteria: If neighbor_tile is within the map boundaries, is traversable, and has not been marked as visited:

* Mark neighbor_tile as visited.

* Add neighbor_tile to the current cluster.

* Enqueue neighbor_tile.

d. Once the queue is empty, the growth process for the current cluster is complete. Add this completed cluster to the list of clusters.

- Completion: Repeat steps 2 and 3 until all tiles on the map evaluated. The final result is a list ofclusters with varying shapes and sizes, which accurately partitions all traversable areas of map.

There is an inherent benefit to this approach with respect to changing dynamic environments. If a bridge connecting two landmasses is destroyed, a re-run of the Region Growing algorithm will simply separate the two landmasses into two clusters, no longer one. Conversely, if a bridge is created, the algorithm will merge the two separate clusters into a larger, new cluster. These kinds of adaptations to large-scale changes in topology occur within the core algorithm, without needing to treat this case separately.

C. Adaptive Abstract Graph Construction

Once the map has been partitioned into adaptive clusters, the remaining procedures in the AHPA* preprocessing phase is essentially following the same logic as standard HPA*, but applied to the new, non-uniform partition:

- **Entrance Identification:** The algorithm looks at the borders between each pair of adjacent adaptive clusters in order to identify sequences of traversable tiles that can be defined as entrances.
- **Abstract Graph Construction:** Similar to HPA*, the nodes in the AHPA* abstract graph are these entrances. The intra-edges are calculated by executing low-level A*, between all pairs of entrances in each adaptive cluster, the results are cached. The inter-edges link entrances that are adjacent but belong to different clusters.

The key difference is that the abstract graph generated by AHPA* is more efficient. The large open areas are represented by only one or a few abstract nodes (the entrances at their edge), while the complex areas are represented with a higher density of

nodes. This gives us a "topologically optimized" abstract graph that can be searched faster during the online phase.

IV. Experimental Design and Results

In order to validate the effectiveness of the AHPA* method, a comparative experimental study was designed and conducted. The monitor study was driven by the desire to quantitatively measure AHPA* performance against standard A* and standard HPA* performance across various topological scenarios with specific performance metrics relevant to RTS game applications.

A. Environment and Test Scenarios

The experiments were conducted on system with an Intel Core i7-9750H central processing unit (CPU) and 16 GB of Random Access Memory (RAM) in order to maintain the consistency and robustness of results. The algorithm implementations were built in C++ without third-party dedicated game engine libraries as a method to strip the performance out from the algorithms themselves.

Algorithms Compared:

1. Standard A:* Used as a grounding reference metric to capture path quality. This algorithm is run on the low-level grid and produces a guaranteed optimal path that we will use as our baseline (0% deviation) for path quality measurement.
2. Standard HPA:* A canonical approach to HPA* with fixed 15x15 tile clusters. This is used as a baseline to measure performance against heirarchical approaches that already exist.
3. AHPA (Proposed):* Our best efforts at performing the method using Region Growing for adaptive clustering.

Test Maps:

Three 256x256 tile maps´were created specifically to test the hypothesis through a variety of topological contexts, as is standard for testing path itself:

- Map 1: (Labyrinth): map handling a majority of long, narrow, and winding corridors with few larger open areas. This map was created as a worst case for high level abstraction, through a complex topology that created many small clusters.
- Map 2: (Open Area): map with very few obstacles. Characterized as one or two very large open areas dominated this map It was designed to demonstrate a weakness of HPA*'s uniform clustering, and also to show the maximum potential benefit of AHPA*.
- Map 3 (Mixed/Typical RTS): Map was designed with a typical RTS level design which included a balance of some open base area, resources scattered here and there, and some choke points connecting areas. For Map 3, it was likely the realistic test case.

For each map, ten pre-created pathfinding problems (paired at random start and goal point configuration) were created to ensure that the results to be produced were statistically valid for noncertain specific start-goal point configurations.

B. Evaluation Metrics

The three main metrics were used to assess the performance of each of the algorithms. These evaluation metrics had been effective in similar studies of competitive comparison:

1. **Computation Time (milliseconds):** Each algorithm's execution time. HPA* and AHPA* divided in two:
   - **Preprocessing Time**: How long it takes to construct the hierarchical data strctures (the partitioning of the map, identifying entrances, and building in the cache). It will be completed once in offline time.
   - **Average Search Time**: The average time required to solve one pathfinding request from the 100 tested problems. The most critical online cost for real-time performance.
2. **Search Efficiency (Number of Nodes Expanded):** The total number of nodes inserted into the Closed List during the search. This metric is a good proxy for measuring the computational effort of the algorithm, independent of hardware speed.
3. **Path Quality (Path Length Deviation):** How much the length of the path produced by HPA* and AHPA* deviates from the optimal path found by A*. Calculated with the formula:

$$Deviation(\%) = \left(Path\frac{Length}{A^{*}Path\ Length} - 1\right) \times 100\%$$

C. Results and Analysis

Table 1. Pathfinding Algorithm Performance Comparison

Test Map	Algorithm	Preprocessing Time (ms)	Average Search Time (ms)	Average Nodes Expanded	Path Length Deviation (%)
Map 1 (Labyrinth)	A*	N/A	18.54	24,870	0.00%
	HPA*	112.8	1.95	2,155	1.15%
	AHPA*	125.3	1.88	2,098	1.21%
Map 2 (Open Area)	A*	N/A	25.12	35,600	0.00%
	HPA*	109.5	2.81	3,050	0.88%
	AHPA*	115.1	0.45	488	0.95%
Map 3 (Mixed)	A*	N/A	21.77	29,110	0.00%
	HPA*	115.2	2.43	2,640	1.42%
	AHPA*	130.6	0.89	965	1.55%

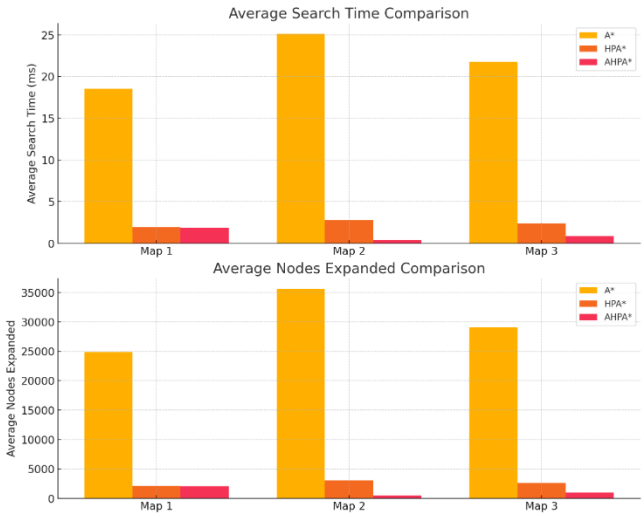

Figure 5. *Graph illustration*

The results presented in Table 1 resulted in a number of important observations:

1. **Processing Time Investigation:** As we expected, AHPA* has a slightly greater amount of preprocessing time (roughly 10-15%), in all scenarios relative to HPA*. This is of course due to the added cost of the Region Growing segmentation step. However, we view this cost as a one time offline investment, that, as we will see, has considerable gains to be made in the online phase.
2. **Performance Investigation on the CL Labyrinth Map:** The performance of AHPA* and HPA* is very similar on Map 1. Average search time and the number of nodes expanded are essentially identical. This corroborates the hypothesis: the dense structure of the labyrinth topology results in both methods producing small clusters more or less naturally, so the resulting abstract graphs have a similar degree of complexity. This indicates that in the absolute worst case scenario, AHPA* can be no worse than standard HPA*.
3. **Performance Investigation on the CL Open Area (Map):** Map 2 is when it shines brightest in terms of confirming that AHPA* is superior to HPA*. AHPA* is 6.2 times faster than HPA* in average search time (0.45 ms vs 2.81 ms), and expands only about 16% of the number of nodes expanded by HPA*. This acts as a direct confirmation of our central hypothesis. Region Growing has effectively clustered the large open area by detecting it all as one large cluster, which resulted in a very simple and concise abstract graph. In contrast, HPA* broke the same area into numerous unnecessary 15x15 clusters that resulted in a much slower high level search.
4. **Performance Analysis on the Mixed Map:** On Map 3 - the most realistic representation of an RTS map - the advantage goes to HPA*, with AHPA* being 2.7 times faster than HPA* (0.89ms vs 2.43ms) with far fewer node expansions. This illustrates how AHPA* adapts to the topology by creating large clusters for bases and small clusters at choke points which make an overall much more efficient abstraction for navigation.
5. **Path Quality Analysis:** During our tests, both HPA* and AHPA* showed low deviations for path length, typically below 2% of the optimal A* path length. Again this indicates that the performance benefits of AHPA* are achieved with little meaningful sacrifice to path quality; while it may really be ideal to obtain the *perfect* path, obtaining performance and a good enough path is an acceptable trade off in real-time applications.

Overall, I think these results do provide strong evidence to support the claim that topology-based adaptive clustering is a better approach to quadratic clustering for hierarchical pathfinding on different maps.

V. CONCLUSION

This work was motivated by an elementary problem regarding RTS game development: the pursuit of a fast, reliable, pathfinding system that could handle large-scale maps and hundreds of units in real-time. The standard A* algorithm was optimal, but as expected too slow. Hierarchical Pathfinding A* (HPA*) represents an industry suitable solution by abstracting the map; however, the inefficient uniform clustering used by HPA* results in a less than ideal outcome for heterogeneous topologies in the game world.

The primary contribution of this paper is the introduction and empirical evaluation of Adaptive HPA (AHPA), a variant of HPA* which substitutes the static grid clustering technique with an adaptive map segmentation approach that employs the Region Growing algorithm. This allows AHPA* to create an abstract graph that adapts to the topological structure of the game environment, with larger clusters for more open spaces, and smaller clusters to correspond with more complex areas.

The results of the comparative experiments provide strong evidence in support of the research hypothesis. AHPA* consistently proved better than standard HPA* in online search time and search efficiency (nodes expanded). This improved performance was most impressive on maps that combined open areas with narrow corridors—a very standard situation in real-time strategy (RTS) level design. This speed increase performed with a little more pre-processing, and a very little and acceptable diminishment in path quality, shows that AHPA* provides a better trade-off when applied to real-time applications.

Future research should consider the breaches more than AHPA* also suggests:

- **Better Clustering Algorithms:** Future research could employ better segmentation or graph clustering algorithms to create more complex segmentation, such as watershed-based methods or community detection algorithms, such as Louvain , to hopefully produce more optimal abstractions
- **Integration of Path Smoothing:** Pathing produced by grid-based algorithms such as AHPA* tend to be rigid, it would be interesting to integrate efficient path smoothing algorithms, e.g., splines or Bézier curves, as a post-

processing step to produce natural looking movements (visually smooth).

- **Handling Fully Dynamic Environments:** This research focused on static obstacles. The next step is to adapt AHPA* to handle fully dynamic environments, including moving obstacles like other units. This could involve drawing inspiration from variants like DHPA* for dynamic cache management or integrating AHPA* with a local collision avoidance system.

In conclusion, AHPA* offers a robust and logical improvement over HPA* by addressing its core weakness in the abstraction process. By making the clustering process topology-aware, AHPA* takes a step forward in the quest for intelligent, fast, and scalable pathfinding for the next generation of real-time strategy games.

VI. APPENDIX

Github: *https://github.com/BenedictusNelson/pathfinding-program*

(repository for testing performance and path quality of the proposed AHPA algorithm compared to A* and HPA* program.)*

ACKNOWLEDGMENT

Firstly, I wish to express my heartfelt thanks to the lecturer Rinaldi Munir for valuable advice through the course of this paper's development. I deeply gratified myself with all my classmates who were with me in the Strategi Algoritma class for their constructive criticism and feedback that really sharpened the quality of this work. Further, I acknowledge the support provided by the STEI-ITB library and online sources for allowing access to the relevant materials.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[2] A. Botea, M. Müller, and J. Schaeffer, "Near-Optimal Hierarchical Path-Finding," *Journal of Game Development*, vol. 1, no. 1, pp. 7-28, 2004.

[3] N. Sturtevant, "Pathfinding in Real-Time Strategy Games," in *AI for Real-Time Strategy Games*, D. Churchill, M. Buro, and M. Fairclough, Eds. Springer, 2015, pp. 1–17.

[4] R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. MacDonald, "Hierarchical A*: Searching in Abstract State Spaces," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1996, pp. 570–575.

[5] T. Adams and R. S. Adams, "Seeded Region Growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641–647, June 1994.

[6] S. R. H. G. Raj, S. J. T. Jose, and S. R. B. Raj, "A Survey on Path Smoothing Algorithms," in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, 2018, pp. 488–492.

[7] Lawande, S., Jasmine, G., and Anbarasi, L., "A Systematic Review and Analysis of Intelligence-Based Pathfinding Algorithms in the Field of Video Games," *Applied Sciences*, vol. 12, no. 18, pp. 1–23, Sep. 2022.

[8] Zhao, J., Smith, T., and Liu, H., "Reducing Redundant Work in Jump Point Search," in *Proceedings of the 16th Symposium on Combinatorial Search (SOCS)*, 2023, pp. 88–96.

[9] Putra, R. A. B., Prihatmanto, A. S., and Yuliana, S., "Heap Optimization in A* Pathfinding for Horror Games," *Journal of Information Systems and Informatics*, vol. 7, no. 1, pp. 64–74, Jan. 2025.

[10] Chen, L. and Wang, Y., "HMLPA*: A Hierarchical Multi-Target LPA* Pathfinding Algorithm for Spatially Structured Maps," *International Journal of Spatial Algorithms*, vol. 4, no. 2, pp. 55–68, Apr. 2025.

[11] Kumar, P., Singh, R., and Joshi, D., "Empirical Analysis of Hierarchical Pathfinding in Lifelong Multi-Agent Systems," *Systems*, vol. 11, no. 9, pp. 511–529, Sep. 2023.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Juni 2025

Benedictus Nelson 13523150