Fast Approximate Pathfinding Based on 2D Convolution

Marius Teleiša [1] and Dalia Čalnerytė [1]

[1] *Kaunas University of Technology, Studentų str. 50, Kaunas, LT-51368, Lithuania*

Abstract

The goal of pathfinding algorithms is to find a path between the desired points. An optimal path is more complex and time consuming to find, which is why some industries, such as the video game industry, can sacrifice optimality for reduced run-time. A grid map can be represented as an image, so techniques used in image processing, such as filtering, may be applied to pathfinding. In this paper we propose a convolution inspired hierarchical pathfinding algorithm that achieved 4% longer paths and 97% shorter runtime than A* on average.

Keywords

Hierarchical pathfinding, Grid graphs, Heuristic search

1. Introduction

Pathfinding algorithms solve the problem of finding a path (usually shortest) between two points. Finding a path between two points is not difficult and can be done by simple algorithms such as breadth-first search, which has a linear time complexity [1], however, this only works for unweighted graphs. The time complexity of the Dijkstra algorithm used to solve this problem is non-linear [1] due to the need to sort current paths by length and guarantee an optimal path.

Search spaces are usually represented using graphs, which may even have a geometric structure, and it is not uncommon to see video games utilizing grid graphs [2], [3]. In video games, various logic must be completed within a short time frame (e.g., 16ms for 60 updates per second or more), so it is critical for developers to employ a pathfinding algorithm that can perform a search in these time constraints. Video games are not required to have true-to-life graphical fidelity, physics simulation, or pathfinding, so game developers often use more computationally efficient methods to solve those problems. One such example is finding a close-to-optimal path at a fraction of the time, when compared to traditional methods, such as A*.

Grid maps can be represented as images with free tiles and obstacles, which can then be processed to form a grid graph. A grid map can be processed into various different graphs, depending on tile connection logic. The two classical ways to connect the neighboring tiles for a grid map are 4-way (cardinal directions) and 8-way (cardinal and diagonal directions). Both methods introduce the path symmetry problem, but the 8-way connection option also doubles the edge count, all of which can easily result in unreasonable search times when using pathfinding algorithms, most of which have non-linear time complexity.

Filtering is a powerful technique for image processing that can be used to extract features in raster images. Image filtering is generally performed as a series of local neighborhood operations using a sliding-window-based principle [4]. The sliding window partitions the grid, where a convolution operation can be utilized for each partition to extract information. A similar technique may be employed for pathfinding, so in this paper we propose a pathfinding algorithm based on 2D convolution, to perform fast and approximate pathfinding in 4-way connected uniform-cost grid maps.

2. Pathfinding problems and methods

IVUS2023: Information Society and University Studies 2023, May 12, 2023, Kaunas, Lithuania
EMAIL: marius.teleisa@gmail.com (M. Teleiša); dalia.calneryte@ktu.lt (D. Čalnerytė)
ORCID: 0000-0003-4185-0397 (D. Čalnerytė)

2.1. Path symmetry

One of the biggest problems with grid maps is path symmetry, where there are multiple equally optimal paths for two given points, as shown in **Figure 1**. Path symmetry is often a property of uniform-cost grid maps, where the order of steps can often be rearranged to form another equally optimal path.

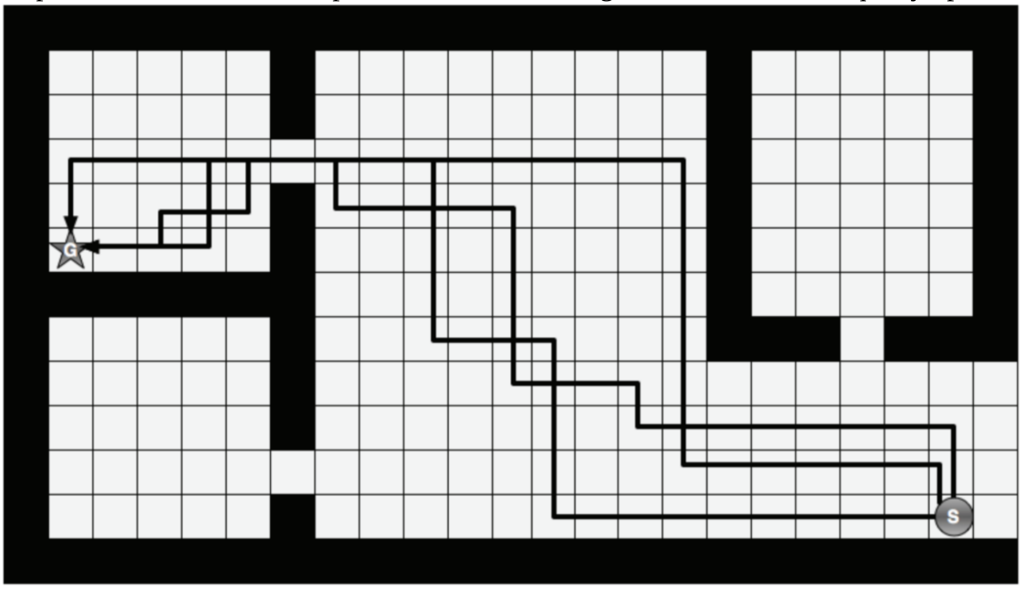


Figure 1 A highly symmetric pathfinding instance with three highlighted paths[5]

The number of symmetric paths increases with path length, and a pathfinding algorithm, such as A*, must evaluate all the redundant symmetric paths to find an optimal path. JPS is a pathfinding algorithm based on elimination of symmetric paths and can run up to 3.5 times faster than A* [5], which shows that path symmetry can hinder pathfinding algorithm performance significantly.

2.2. A*

One of the most popular and effective pathfinding algorithms is A* [6], [7]. A* is a best-first search algorithm that can use various heuristic functions to adapt to the available space, its rules or the required solution. This algorithm can find the optimal path if a suitable heuristic function is chosen, which depends on graph type.

Each iteration A* adds a node to current paths, which is determined by current cost of the path and the estimated cost from this node to goal by a heuristic function. The nodes keep track of which node they were reached from, and once the algorithm reaches the goal, it traces back to the start to form a path and terminates.

One of the ways to reduce search time for A* is to have a better heuristic. Standard online heuristics, such as Manhattan distance, do not consider obstacles and can only estimate an optimistic scenario without obstacles, which leads to the pathfinding algorithm having to expand many nodes when encountering obstacles. A heuristic function that can accurately evaluate distance between nodes may result in the pathfinding algorithm expanding only nodes on the optimal path [8].

2.3. HPA*

Hierarchical Pathfinding A* (HPA*) is a pathfinding algorithm that was developed in 2004, with the aim of reducing the pathfinding time by sacrificing path optimality [9]. HPA* uses a hierarchical pathfinding approach, which allows the information of the network-type space to be processed once, so that this information can be used to speed up the performance of A*.

HPA* algorithm creates an abstraction layer by dividing the map into rectangular parts called clusters, as marked by the red rectangles in **Figure 2**. Next, the passage points, shown in green, are searched between the clusters and added to the graph of the abstraction layer. This algorithm has been applied to 4-way connected graphs and experimental results have shown up to a 10-fold speedup in pathfinding and a 1% degradation in path quality compared to the optimal [9].

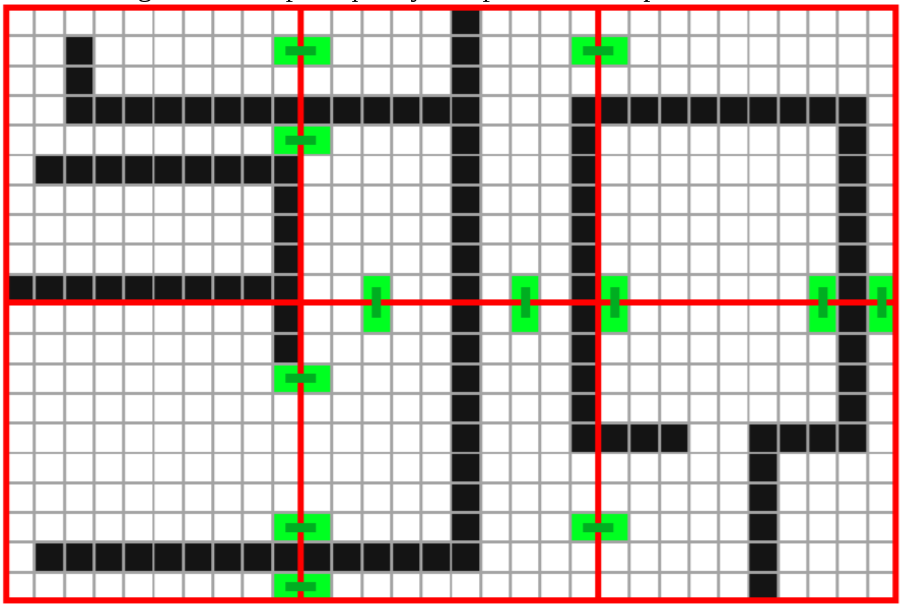


Figure 2 HPA* abstract layer creation [10]

2.4. Heuristic functions

The goal of a heuristic in pathfinding is to guide an algorithm to the target node. A standard heuristic function for 4-way connected maps is Manhattan Distance (MD) (1). Given two points $\boldsymbol{x} = (x_1, x_2, \dots, x_n)$ and $\boldsymbol{y} = (y_1, y_2, \dots, y_n)$ MD is calculated as the sum of distances in each dimension [11]:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} |x_i - y_i| \tag{1}$$

MD is fast to calculate, but this method assumes only orthogonal movement, which overestimates travel distance in 8-way connected grids where diagonal movement is allowed. Euclidean distance (ED) (2) is more suitable for 8-way connected grids; however, the calculation process is computationally expensive as it utilizes a square root operation.

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2}$$

To speed up the computation, square root can be removed from the equation (2). This makes the heuristic function overestimate the distance to the destination, which results in the pathfinding algorithm finding paths faster at the cost of no longer guaranteeing optimality [7]. This version of the heuristic function is called squared Euclidean Distance (SED).

3. Proposed pathfinding algorithm

Convolutional Hierarchical Pathfinding A* is a pathfinding algorithm that utilizes offline preprocessing to construct an abstraction layer, which is used to perform an online search. The abstraction layer is smaller than the original search space, which results in reduced search time. As the name suggests, the process for creating the abstraction layer is based on 2D image convolution, where a sliding window is used to generalize information inside the window.

During preprocessing, a non-square sliding window is used to partition the map into square segments, as shown in **Figure 3**. If necessary, a map is padded to the required length by copying the nearest pixel of the original image. The shape of the sliding window is derived from the square segment shape and should be one tile longer to allow overlap with neighboring segments.

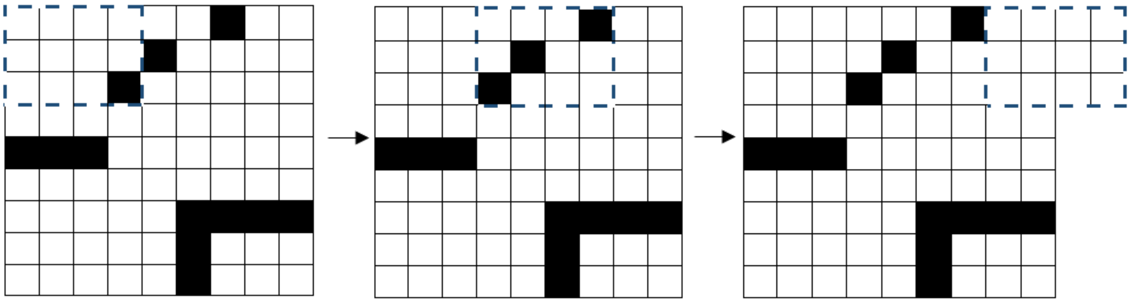


Figure 3 Horizontal map slicing with 4x3 window

Within each window, a search is performed to find any valid path along the window, and the purpose of the overlap with neighboring segment is to guarantee, that the neighboring segment can be entered. A clockwise rotated window is also used to perform the same operations and store vertical traversal information. The abstraction layer graph is created by using the segments as nodes, and the traversal information from sliding windows to connect the nodes.

Increasing the segment size will result in a smaller abstraction layer and faster search time, however more information will be lost during preprocessing, which can reduce path optimality. For this paper a segment size of 3x3 was chosen to introduce some data loss and evaluate the effect of it on pathfinding performance.

During phase 1 of online search, an initial path is found in the abstraction layer using A*, which is presented in **Figure 4**. In phase 2, the nodes of this path are then used as checkpoints and guide the A* algorithm in the real grid. For this to work, a coordinate translation must be performed between abstraction layer path nodes and real grid nodes. The translation method first determines the direction of movement and shifts the translated center point of the target segment towards the origin segment. A valid unoccupied tile is then needed as a goal, which is searched in a predetermined order along the wall of the origin segment.

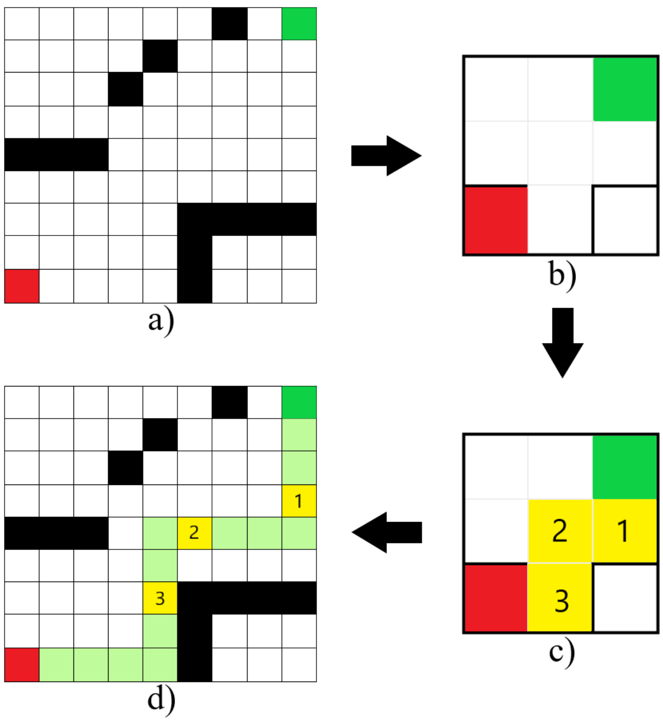


Figure 4 CHPA pathfinding example. a) real map, b) abstraction layer, c) pathfinding in abstraction layer, d) pathfinding using checkpoints in real map. Green – start, red – goal, yellow – checkpoint, light green – path

This method of coordinate translation may reduce path optimality. To reduce errors caused by the coordinate translation, a Pstep parameter is introduced, which defines the interval of checkpoints to be used for pathfinding in the real grid. An example of Pstep effect on final path can be seen in **Figure 5**, where the resulting final path using Pstep=2 is more optimal than Pstep=1. Increasing Pstep value reduces the checkpoint, and the associated coordinate translation count, which results in fewer opportunities for sub-optimal translations and should on average increase path optimality.

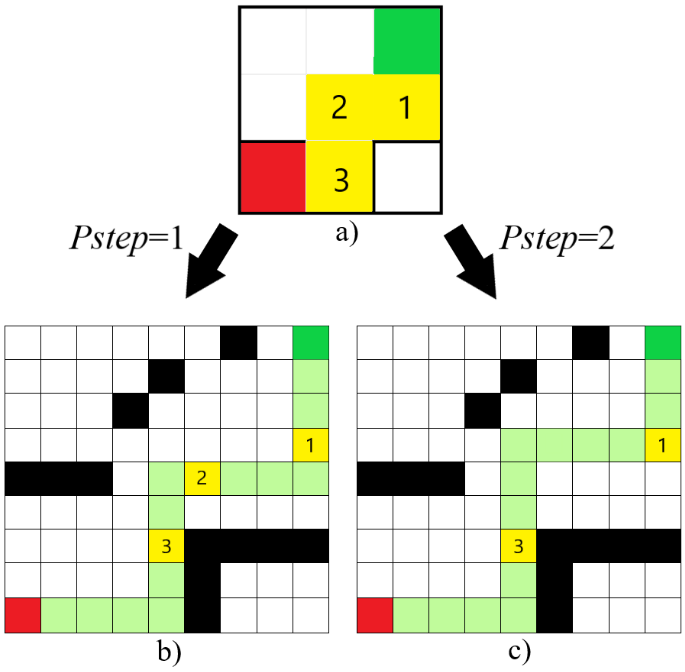


Figure 5 CHPA* pathfinding results for a) abstraction layer with b) Pstep=1 and c) Pstep=2

4. Experimental setup

The proposed algorithm was tested against the A* algorithm using various heuristic functions. The same A* implementation was also used for CHPA*, which will make for a fair comparison as there will not be an implementation optimization difference.

The benchmark set of maps and scenarios used for testing were created by Sturtevant [12], and features maps from games such as Starcraft, Warcraft III, labyrinths, randomly generated maps, etc. The algorithms were implemented using Python, and pathfinding for path lengths above 700 tiles can take more than a minute depending on obstacles, so path counts had to be reduced to have a reasonable testing time. The scenarios were ordered by path length and divided into 200 segments, where one path was chosen at random from each segment. The maps chosen for testing were:

- ArcticStation
- BrokenSteppes
- Enigma
- Nightshade

The tests were carried out using a personal desktop computer, and the specifications are as follows:

- CPU – AMD Ryzen 5 3600
- GPU – Nvidia GEFORCE GTX 1080Ti GPU
- RAM – 16GB
- OS – Windows 10

5. Results

The effects of CHPA* parameter Pstep were tested and can be seen in **Figure 6**. As the value of Pstep increases, the path length approaches optimal, however that also causes more nodes to be explored in the real grid, which will increase search time.

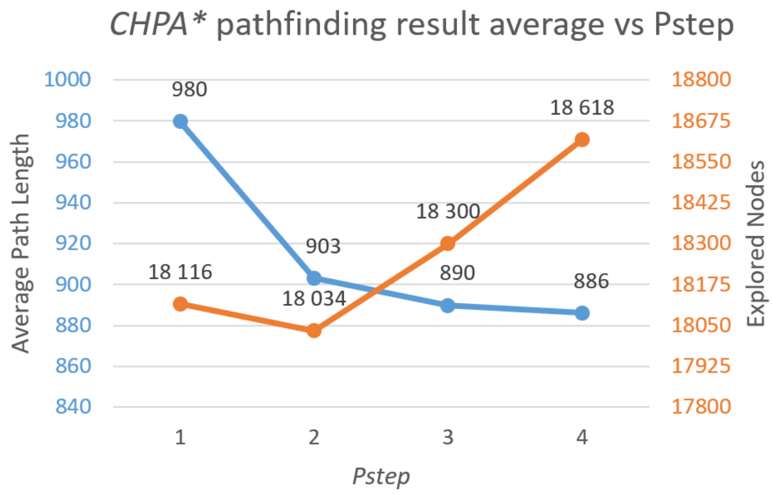


Figure 6 CHPA* test results using various Pstep values

We can also see that on average, with Pstep=2 the path length was reduced by 8%, and explored node count by 0.5%. Increasing Pstep further has diminishing returns on path length reduction of 1.5% and 0.5% for Pstep value of 3 and 4 respectively, and increases explored node count by 1.5%. Pstep=2 resulted in the best balance of path quality and search area, so further experiments will be carried out only using this value.

Next, A* and CHPA* algorithms were tested using various heuristic functions, and the results can be seen in **Table 1**.

Table 1

Pathfinding result of A* and CHPA* with various heuristic functions. F1 and F2 for CHPA* denote phase 1 and phase 2, where phase 1 is the search in the abstraction layer

Pathfinding method	Average path length	Average explored nodes
A* + ED	873.37	120005.71
A* + SED	1040.63	16770.77
A* + MD	873.37	80696.25
CHPA* + F1-ED F2-ED	903.09	18033.84
CHPA* + F1-ED F2-SED	903.79	16766.83
CHPA* + F1-MD F2-MD	**904.23**	**13446.26**
CHPA* + F1-MD F2-ED	904.23	14255.29

A* algorithm with ED and MD heuristics achieved the expected optimal average path length, however MD heuristic explored 33% less tiles and was less computationally expensive, so it led to significantly shorter search times. Using SED heuristic results in 19% longer paths on average, but it explores 80% less nodes than MD, which can be valuable in situations where computation time is strict.

Shifting focus to CHPA*, phase 1 and phase 2 ED heuristic on average explored only 8% more nodes than A* with SED, while having only 4% longer path than optimal. Other CHPA* configurations explored even less nodes, while preserving average path length within 0.2% of other configurations, so choosing a heuristic for CHPA* largely comes down to minimizing explored nodes. Out of the tested configurations with CHPA*, MD heuristic resulted in longest average path, which is still 14% better than A* with SED, and 20% less explored nodes.

Surprisingly, ED heuristic outperforms MD heuristic on CHPA* average path length by 0.2%, but explored 34% more nodes. MD and ED heuristics both find optimal paths, however those paths may differ and cause different errors in CHPA* coordinate translation, which results in paths of different length in the real map.

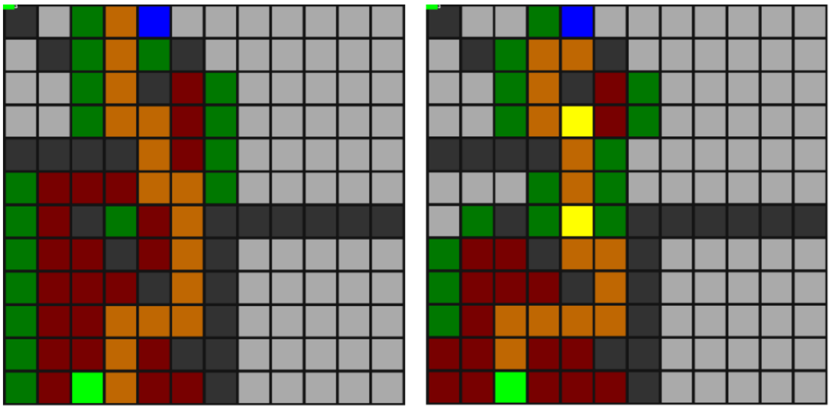
The last test was performed to compare average search time between the algorithms, and the results are shown in **Table 2**.

Table 2
Pathfinding search time results

Pathfinding method	Average path length	Average search time, s
A* + MD	873.37	29.23
CHPA* + F1-MD F2-MD	884.04	0.79

During this testing, CHPA* achieved only 2% longer path, however on average it took 97% less time to find them, which is the result of heavily reducing search space.

The implemented system includes capability to visualize the pathfinding process, which was used to observe general pathfinding algorithm behavior. In **Figure 7** a comparison between A* and CHPA* is shown for a short path, where CHPA* has a smaller exploration footprint than A*. Unfortunately, this system does not visualize the abstraction layer, and the total explored nodes count is very similar to A* in this instance.



a) b)

Figure 7 Pathfinding result visualization on 12x12 map. a) A* b) CHPA*. Heuristic – MD, Pstep=1, green – start, blue – goal, orange – path, yellow – checkpoint, red – closed list tile, green – open list tile

Figure 8, **Figure 9**, and **Figure 10** present pathfinding algorithm behavior over a longer distance, and here the benefits of CHPA* can be seen. A* explored most of the map due to the walls obstructing the optimal path, whereas CHPA* performs pathfinding in the compressed abstraction layer, which greatly reduces node expansion.

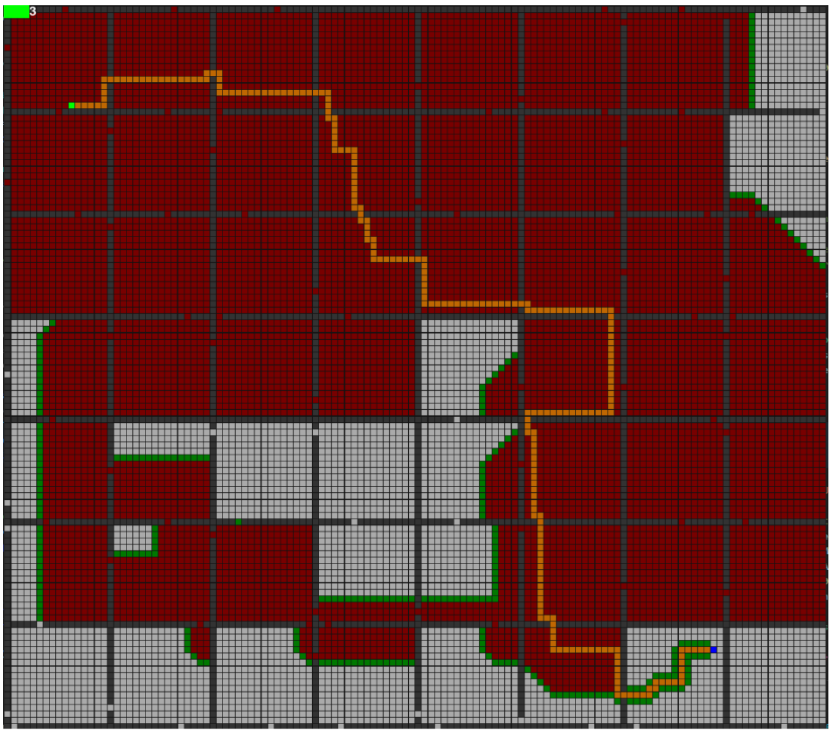


Figure 8 A* + MD result visualization on 128x128 map. Green – start, blue – goal, orange – path, red – closed set tile, green – open set tile

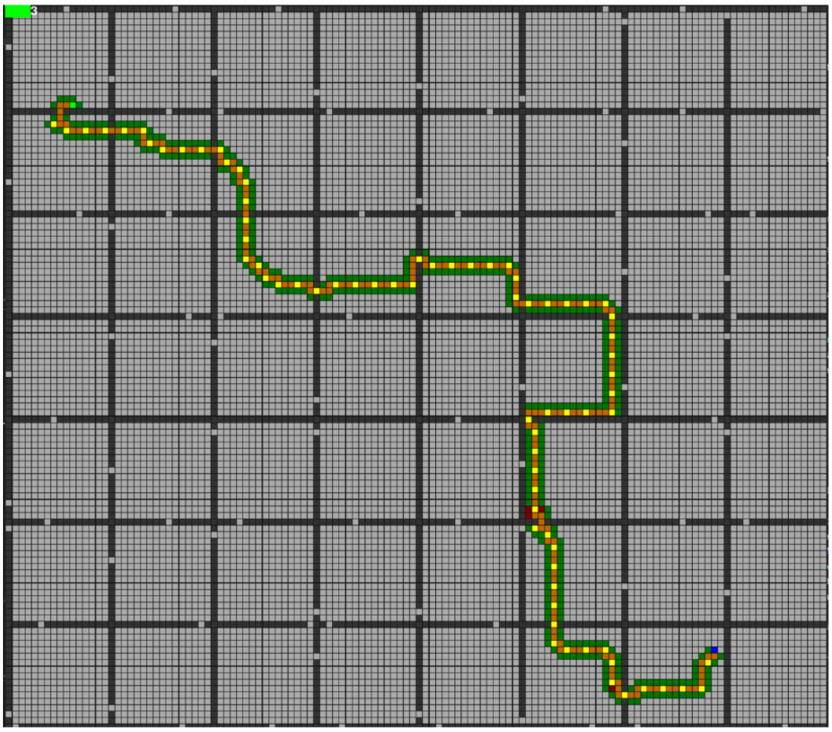


Figure 9 CHPA* Pstep=1 result visualization on 128x128 map. Green – start, blue – goal, orange – path, yellow – checkpoint, red – closed set tile, green – open set tile

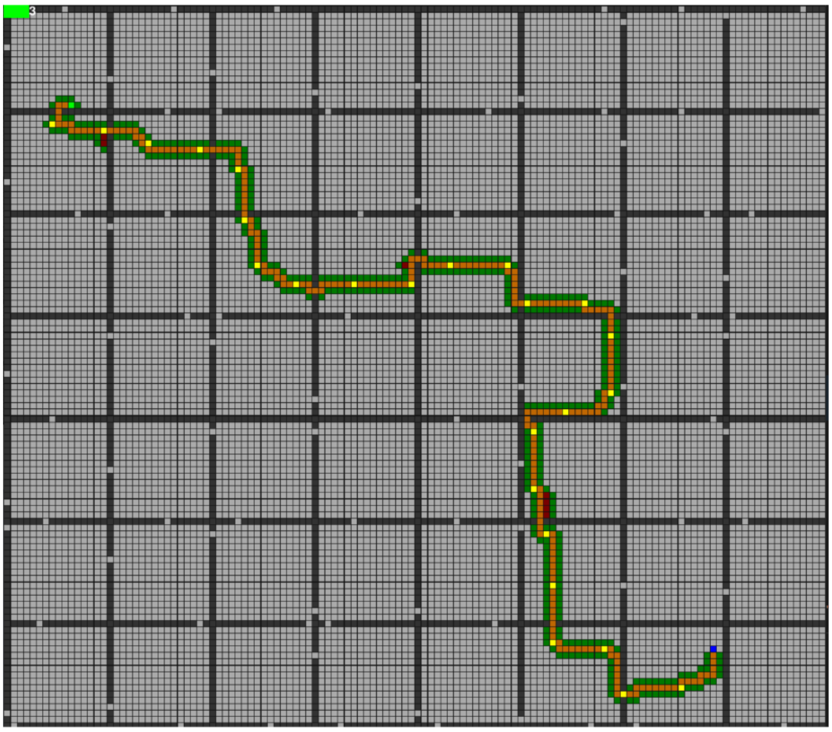


Figure 10 CHPA* Pstep=3 result visualization on 128x128 map. Green – start, blue – goal, orange – path, yellow – checkpoint, red – closed set tile, green – open set tile

6. Future work

Main problems of current CHPA* implementation:
1. Only works with 4-way connected graphs.
2. Simplistic coordinate translation logic reduces path quality.
3. Preprocessing edge cases lead to obstacles not being recognized and producing extremely long paths.

In the future, an analysis on the effect of different window sizes for path quality and search times can be performed.

The proposed algorithm finds paths between multiple checkpoints, which could be done in any order, so parallel processing may be applied to speed up the search even further.

7. Conclusion

In this paper a new pathfinding algorithm is proposed for fast approximate pathfinding called CHPA*. The algorithm utilizes preprocessing, inspired by 2D image convolution, to create an abstraction layer, reduce search space and improve path search times.

Results in worst case show 4% path length degradation on average, more than 80% less explored nodes, and takes 97% less time to find a path compared to A* with an optimal MD heuristic. During our testing, the heuristic function for CHPA* had very low impact on path length (only 0.2% difference), but significantly changed explored node count, so the heuristic choice in most cases will come down to minimizing search space.

The drawbacks of the algorithm are preprocessing edge cases that can lead to not recognizing obstacles in abstraction layer, creating extremely long paths during online search, or even not being able to find a path.

8. References

[1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, "Introduction to Algorithms, Second Edition," 2001, pp. 451–507.

[2] X. Cui and H. Shi, "A * -based Pathfinding in Modern Computer Games," International Journal of Computer Science and Network Security, vol. 11, no. 1, pp. 125–130, 2011.

[3] A. Botea, B. Bouzy, M. Buro, C. Bauckhage, and D. Nau, "Pathfinding in Games," in Artificial and Computational Intelligence in Games, S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, Eds., in Dagstuhl Follow-Ups, vol. 6. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, pp. 21–31. doi: 10.4230/DFU.Vol6.12191.21.

[4] C. Solomon and T. Breckon, "Fundamentals of Digital Image Processing," Wiley, 2010, p. 87. doi: 10.1002/9780470689776.

[5] D. Harabor and A. Botea, "Breaking Path Symmetries on 4-Connected Grid Maps.," in Proceedings of the 6th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, 2010.

[6] Y. Björnsson, M. Enzenberger, R. C. Holte, and J. Schaeffer, "Fringe search: Beating A* at pathfinding on game maps," IEEE 2005 Symposium on Computational Intelligence and Games, CIG'05, no. January 2005, pp. 125–132, 2005.

[7] A. Suryadibrata, J. C. Young, and R. Luhulima, "Review of Various A* Pathfinding Implementations in Game Autonomous Agent," IJNMT (International Journal of New Media Technology), vol. 6, no. 1, pp. 43–49, Aug. 2019, doi: 10.31937/ijnmt.v6i1.1075.

[8] C. Hu, Q. Yin, Y. Hu, J. Zeng, and L. Qin, "Speeding up FastMap for Pathfinding on Grid Maps," in 2019 IEEE International Conference on Mechatronics and Automation (ICMA), IEEE, Aug. 2019, pp. 2501–2506. doi: 10.1109/ICMA.2019.8816354.

[9] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical path-finding (HPA*)," Journal of Game Development, vol. 1, Mar. 2004.

[10] A. Strand-Holm Vinther and M. Strand-Holm Vinther, "Pathfinding in Two-dimensional Worlds. A survey of modern pathfinding algorithms, and a description of a new algorithm for pathfinding in dynamic two-dimensional polygonal worlds," Aarhus University, 2015.

[11] J. Fürnkranz et al., "Manhattan Distance," in Encyclopedia of Machine Learning, C. Sammut and G. I. Webb, Eds., Boston, MA: Springer US, 2011, pp. 639–639. doi: 10.1007/978-0-387-30164-8_506.

[12] N. R. Sturtevant, "Benchmarks for Grid-Based Pathfinding," IEEE Trans Comput Intell AI Games, vol. 4, no. 2, pp. 144–148, Jun. 2012, doi: 10.1109/TCIAIG.2012.2197681.