Path Planning Using A-star Algorithm

Hussien Hayder Mohammed Al-Kafaji
Graduate School of Electrical and
Computer Engineering
Altınbaş University
redwolf.hacker@gmail.com

***Abstract* — One of the most significant challenges in designing realistic Artificial Intelligence (AI) in computer games is that we usually want to find routes from one place to another. We are not just trying to find the shortest path; we also want to consider travel costs. We can use a graph search algorithm to find this path, which works when the map is represented as a graph. A * is a famous graph search option. Dijkstra's algorithm is the most straightforward graph search algorithm, so let us start there, and we will work our way up to A*.**

Keywords—A-star algorithm, path planning, a shorter Path.

I. INTRODUCTION

When studying the algorithm, the first thing to understand is the data. What is our input? What about our output?

- The input in Graph search algorithms, A* included, takes a ("graph") as our input. A graph is an array of locations ("nodes") and the connections ("edges") between them. A* does not see anything else. It only considers the graph. It does not know whether something is indoors, outdoors, a doorway, a room, or how big an area is. It only sees the graph.
- The output path found by A* is a graph of nodes and edges. What A* does is that it tells people to move from one node to another but, it will not tell how. Remember that the algorithm does not know anything about rooms or doors; it only sees the input graph. We will have to decide whether a graph returned by A* means moving from one node to another, walking in a straight line, opening a door, swimming, or running along a curved path.

The pathfinding graph does not have to be the same as what a video game map uses. Instead, it can use a non-grid pathfinding graph or vice versa/the other way around. Grids are, in most cases, easier to work with, but a con can be the many nodes as a result. On the contrary, A* is faster and uses fewer graph nodes.

II. ALGORITHMS

Dijkstra's algorithm will find the shortest paths between all the nodes in a graph. It was created by computer scientist Edsger W. Dijkstra in 1956 and published three years later.[2]

There are many different variations of the algorithm. Dijkstra's original algorithm was made to find the shortest path between two given nodes[3]; however, there is a variant that produces a shortest-path tree. The variant does this as it fixes a single node and uses this as the "source" node, and from here, it finds the shortest paths to all the other notes in the said graph.

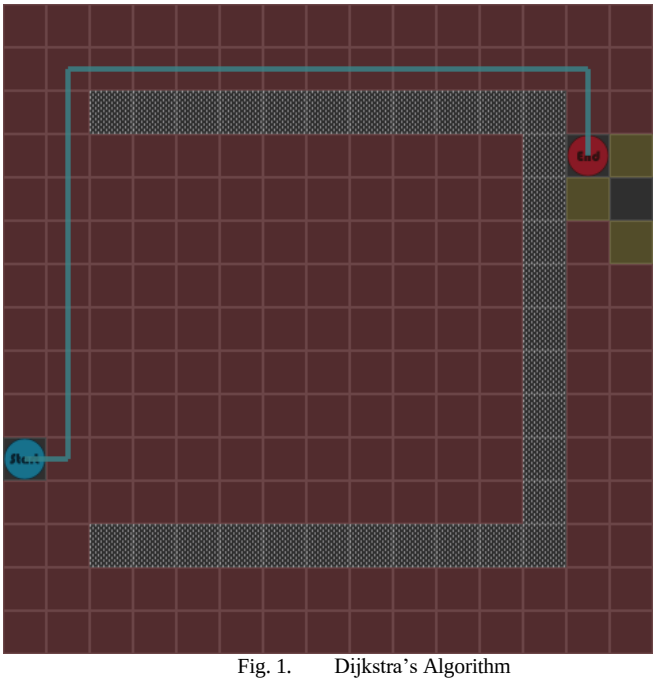

Fig. 1. Dijkstra's Algorithm

Hence, the algorithm is capable of finding the shortest path between a given source node and every other node in the graph. Additionally, this can also be applied to find the shortest paths from a single node to a given (single) destination node. Beforehand mentioned, can be done by stopping the algorithm once the goal node is reached. [4]: 196–206

Best-First Search is search algorithms that explore the graph by expanding the most promising node chosen according to a specified rule. As Judea Pearl described the BFS as estimating the node n by a "heuristic evaluation function $F(n)$ which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most importantly, on any extra knowledge about the problem domain." [1][2] The figure below shows that Best-First Search finds the goal fast but is not the shortest or most optimal.

A. Movement Cost

In pathfinding scenarios, there are different costs for different types of movement. For example, in Civilization, moving through plains cost less than moving through the desert; moving through forest or hills might cost double move-points, walking through water costs ten times as much as walking through grass. We want the pathfinder to take these costs into account.

B. *Heuristic search*

With Dijkstra's Algorithm, the current node expands in all directions. It is a reasonable choice if one is trying to find a path to all locations or multi-able locations. However, a typical case is to find a path to only one location. Let us make the current node expand towards the goal more than in other directions. Usually, we use Manhattan distance to calculate the distance between two nodes or between the current node and the end node.[5]

$$\mathrm{F(n)} = |X_1 - X_2| + |Y_1 - Y_2|$$

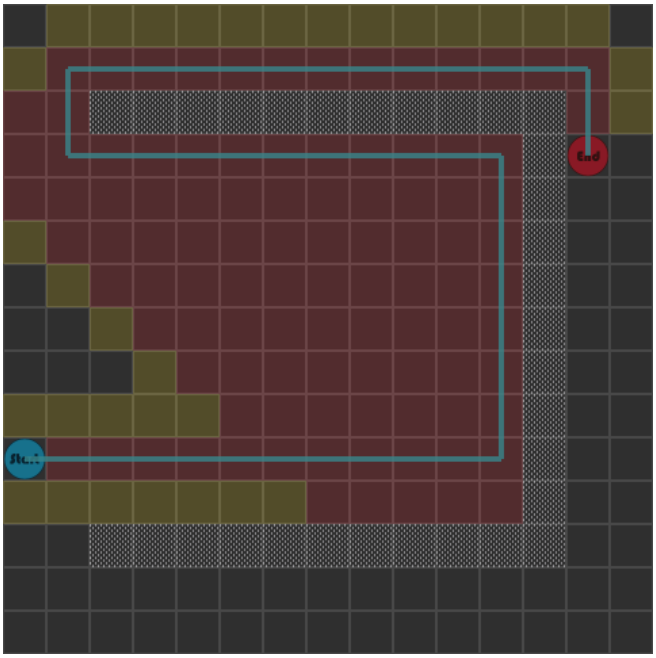

Fig. 2. Best-First Search

III. A* ALGORITHM

A* is an improvement of Dijkstra's Algorithm that is optimized for a single destination.

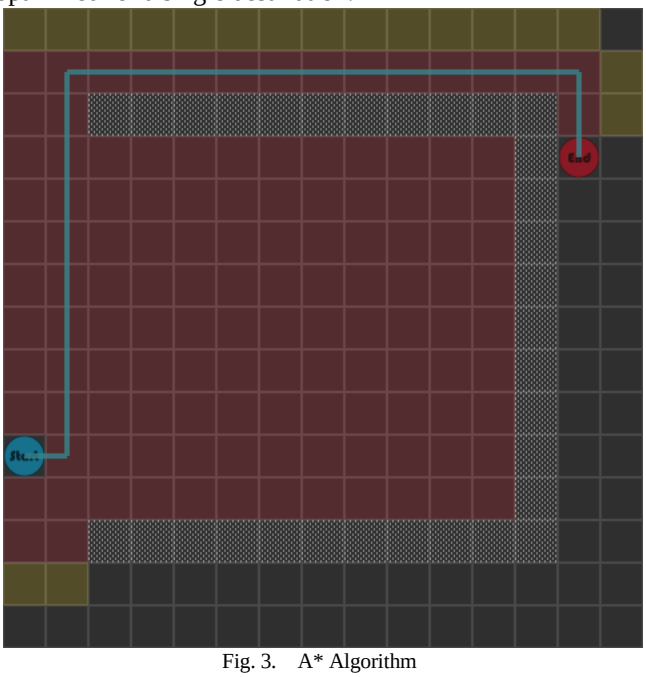

Fig. 3. A* Algorithm

Dijkstra's Algorithm can find paths to all nodes; A* finds paths to one node or the closest of several nodes. It prioritizes paths that seem to be closer to the goal.[6]

Dijkstra works well to find the shortest path, but it wastes time exploring directions that are not promising "fig. 2." Best-First Search explores promising directions, but it may not find the shortest path as seen in "fig. 1." The A* algorithm uses the actual distance from the start and the estimated distance to the goal "fig. 3."

A*, therefore, achieves the best of both worlds. By using the heuristic to reorder the nodes in A*, the possibility of encountering the goal node faster becomes higher. However, this is only possible if the heuristic does not overestimate the distances.

A* is using the following equation $F(n) = G(n) + H(n)$ to determent the best node to explore next. Where $G(n)$ is the distance between the start node and the current node while $H(n)$ is the distance between the goal and the current node, it works well if the movement cost between each node is 1, but usually, that is not the case. As said before, moving through different biomes has different costs, and we want to encourage moving on roads more than off-road.

$$G(current) = G(previous) + distance(current, previous) + movement\ Cost$$

Now A* should prioritize moving on roads over cutting through the land "fig. 4,5".

Fig. 4. A* without movement cost

Fig. 5. A* with a movement cost

ACKNOWLEDGMENT

I can not begin to express my thanks to Sarah Lykke Tost, a student at Aarhus University, and my best friend, the best person I know, for her expert notes, incredible heart, and, as in all things, her invaluable support.

REFERENCES

[1] Pearl, J. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984. p. 48.

[2] Russell, Stuart J.; Norvig, Peter (2003), Artificial Intelligence: A Modern Approach (2nd ed.), Upper Saddle River, New Jersey: Prentice-Hall, ISBN 0-13-790395-2. pp. 94 and 95 (note 3).

[3] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs" (PDF). Numerische Mathematik. 1: 269–271.

[4] Mehlhorn, Kurt; Sanders, Peter (2008). “Chapter 10. Shortest Paths” (PDF). Algorithms and Data Structures: The Basic Toolbox. Springer. doi:10.1007/978-3-540-77978-0. ISBN 978-3-540-77977-3.

[5] Pearl, Judea (1984). Heuristics: intelligent search strategies for computer problem-solving. United States: Addison-Wesley Pub. Co., Inc., Reading, MA. p. 3. OSTI 5127296.

[6] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” IEEE Transactions on Systems Science and Cybernetics. 4 (2): 100–107.