Artificial Intelligence Coursework
SET09122

Dimitrios Tsolis

Matriculation Number: 40204497

2018-2019

Table of Contents

1. Abstract:

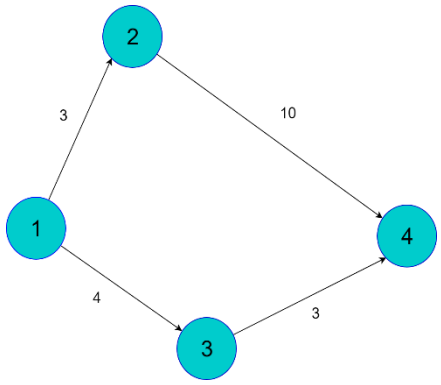
In this coursework, three pathfinding algorithms will be analysed and explained. An algorithm is basically a system for solving a problem. The problem in this coursework is finding the shortest route from one node to another, to which many solutions exist. Specifically, this coursework will include a description for Dijkstra's algorithm, A Star, and Breadth-First Search.

2. Methods

a. Dijkstra's Algorithm

Dijkstra's algorithm was invented by a Dutch computer Scientist "Edsger Dijkstra" in 1956, and it was published in 1956 (Yagvalkya Sharma). Dijkstra's algorithm will always find the shortest route from the start to the end node if a route exists for a map with nonnegative edge path costs.

Figure 1



For the purposes of this report, a small map has been drawn (See Figure 1). The blue circles represent nodes and the black lines are node paths. Each path from node to node has a direction, and a cost associated with it. Each node in the figure has a label, and the numbers on the nodes do not represent the cost of each node. The arrows display which direction the algorithm is allowed to take when travelling from node to node.

The problem here is to find the most efficient route from Node 1 to Node 4. The numbers next to the lines represent the "cost" of going between nodes. The shortest path in this scenario is from Node 1 to Node 2, and from there to Node 4, which is the end goal. Specifically, the cost to get there is 7(4+3), while for the other path it is 13(3+10). For this example, it's quite easy to figure out the shortest route, but for more complex maps it would take more time, which is the reason algorithms are used. Dijkstra's algorithm will find all viable routes to the goal from a map, compare them, and output the shortest path to the route.

In detail, this algorithm finds the path with the lowest cost between one node and every other node. To start with, we have a list of unvisited nodes. After we choose the starting node in a graph, we set the cost in the list for that node to be 0. We set all the other nodes to have a distance of infinity from the starting node to begin with. The next step is updating the list with the cost of getting to the neighbouring nodes from the current one. After processing all the nodes and all the possible paths, the algorithm compares all the distance travelled to get to a specific node and returns the path with the shortest possible distance.

b. A*

A* is an informed search algorithm, and the best-known form of best-first search. Best-first search is a search algorithm, which explores a graph by expanding the most promising node chosen according to a specific rule (**f(n)**). In order to choose the node, A* uses the following formula:

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

- **g(n):** The cost to reach the node(so far)
- **h(n):** The estimated cost to get from the node to the goal(heuristic).
  If **n** is the goal then **h(n)=0**
- **f(n):** Estimated total cost of the path through the end goal

A* is similar to Dijkstra's algorithm in that it can be used to find a shortest path, and it is the most commonly used pathfinding algorithm due to its flexibility. A* is so favourable because it combines the pieces of information from Dijkstra's algorithm and information that Greedy Best-First search uses. Specifically, A* favours vertices that are closer to the starting point (Dijkstra's Algorithm), and it favours vertices that are closer to the goal (Best-First Search).

The algorithm terminates when the path it has selected is a path from the starting node to the end node, or if there are no more paths permitted to be extended. The heuristic function for A* is problem specific, but for this coursework Euclidian Distance will be used as the heuristic. Its node has an X and a Y position, which describe its location, and the distance from one node to another is represented by the following formula:

$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, where (x_1, y_1) and (x_2, y_2) are the coordinates of the respective caverns

In order to make the algorithm more effective, a priority queue can be used. The queue will make the algorithm more efficient as it will prioritize which nodes are selected first depending on their estimated heuristic cost. Therefore, the nodes that will be processed first are the ones with the minimum estimated cost. This implies that at each step of the algorithm, the node with the lowest **f(n)** value is removed from the queue, the **f** and **g** values are update accordingly, and these neighbours are added to the queue. The algorithm will keep going until the queue is empty or if a goal node has an **f** value that is lower than any node in the queue (Wikipedia, n.d.). The **f** value of the goal represents the cost of the shortest path.

For this coursework, A* was used, and the algorithm was changed so that each node keeps track of the previously visited node, in order to find the sequence of steps. Since Euclidian distance is used in this implementation, h(n) represents the distance from the goal mathematically calculating the distance of a straight line from the current node to the goal.

c. Breadth First Search

Breadth First Search is a traversing algorithm. It starts from a selected node (the starting node for this scenario) and traverses the graph layer by layer, therefore exploring the neighbouring nodes. After that, the algorithm moves to the next layer of neighbouring nodes (Garg, n.d.).

Neighbouring nodes in this scenario are only the nodes the current node has access to. For example, in Figure 1, while at Node 1, the algorithm will check nodes 2 and 3 before travelling, and after it has made a choice, it will only check Node 4, as it doesn't have access to Node 1 due to the directional arrows.

The Breadth First Search traverses the graph breadthwise in the following way:

1. First, the algorithm moves horizontally and visits all the nodes in the current layer
2. It then moves to the next layer.

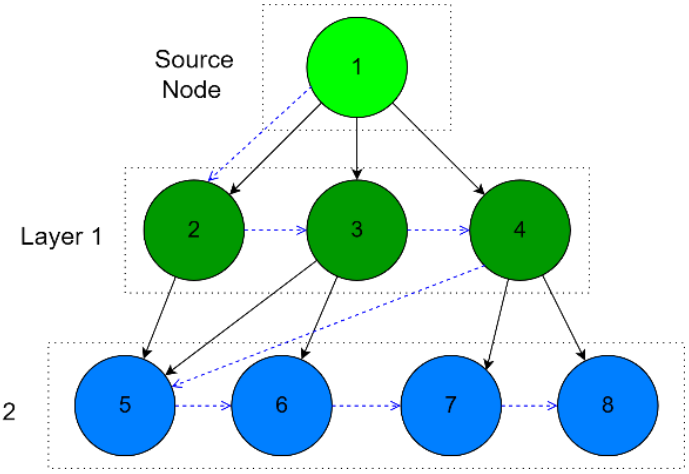


Figure 2

For the sake of the explanation, a graph has been drawn (See Figure 2). The black arrow lines are the directions the algorithm is allowed to go when travelling from node to node. The distance of the nodes from the starting node in layer 1 is less than the distance from the nodes in layer 2. Consequently, using a Breadth First algorithm, all the nodes in Layer 1 will be traversed first before moving on to the ones in Layer 2.

A graph may contain cycles, which may lead the algorithm to traverse nodes which it has already visited. In order to avoid calculating the routes from nodes the algorithm has already visited, an array can be used in order to mark the nodes that have already been visited. While processing nodes on a layer, they should be stored in a way so that they may traverse in a similar order. In order to achieve this non-recursive implementation, a First in First Out queue may be used. This implies that the neighbouring nodes that will be visited are inserted in the queue in the order they are visited, therefore making it easier to visit the child nodes of the current neighbouring nodes later on.

In order to better understand the implementation, take Figure 2. Starting from Node 1, the algorithm will visit Nodes 2,3 and 4. They will be stored in the order they are visited, which will allow the algorithm to visit the child nodes of 2(Node 5) first, then 3(Nodes 5 and 6), and then 4(Nodes 7 and 8) etc.

3. Explanation

a. Dijkstra's Algorithm

Dijkstra's algorithm will visit all the nodes in a graph, and it keeps a track of the total cost from the source node to all the nodes that it's visited. It uses this feature to determine the best order to traverse the graph from the source to the node goal. Dijkstra's algorithm always returns the shortest path and it works with weighted (i.e. the distance between each adjacent node is not always the same) graphs, but it is not as efficient as other algorithms computationally since it checks every node in a graph, so it takes a longer time to calculate the final path. Dijkstra's algorithm would work for this coursework, but it would not be as time efficient as other algorithms, and since one of the aims of this coursework was for then algorithm to find the route as quick as possible, this algorithm was not used.

b. A*

A * is similar to Dijkstra, but it also uses a heuristic to determine how likely each node is close to the goal in order to make the best decision. Due to this heuristic property of A*, it will always return the shortest path if one exists, and it is much more time efficient compared to Dijkstra. For this coursework, it was required to find the shortest path in a timely manner, which is why this algorithm was used.

c. Breadth First Search

The Breadth First Search algorithm performs simple graph traversal by visiting each layer of children at a time, and it stops when the path is found. For weighted graphs, Breadth First search is not guaranteed to find the shortest path, but it will find a path if one exists. If the graph is unweighted, it finds the shortest path but not as efficiently computationally and time-wise as A*, due to the fact that it processes all the nodes in a graph. For this coursework in particular, this algorithm could have been used, but due to the lack of time efficiency compared to A*, it was not used.

4. References

Dijkstra's Algorithm. (n.d.). Retrieved from https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Game Dev Stack Exchange. (n.d.). Retrieved from Path Finding Algorithms: https://gamedev.stackexchange.com/questions/28041/path-finding-algorithms

Garg, P. (n.d.). *Breadth First Search*. Retrieved from HackerEarth: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/?fbclid=IwAR1Kx_uKgoZJ8N8Lg_k25RAU_5j7QpgF24m4UqpZuhFIW1NaIyombGEYWSM

Hazard, E. (2010, May 22). *Dijkstra's Algorithm - Shortest Path*. Retrieved from Vasir: http://vasir.net/blog/game_development/dijkstras_algorithm_shortest_path

Thomas Cormen, D. B. (n.d.). *The breadth-first search algorithm*. Retrieved from Khan Academy: https://www.khanacademy.org/computing/computer-science/algorithms/breadth-first-search/a/the-breadth-first-search-algorithm?fbclid=IwAR1XPx6GYgOmOLWZMWsjD7JqwKbhkZgrpMn1XpZYC6Z2npr2Y1koRVLMIa4

Wikipedia. (n.d.). Retrieved from A Star Search Algorithm: https://en.wikipedia.org/wiki/A*_search_algorithm

Yagvalkya Sharma, S. C. (n.d.). *International Journal of Electronics and Computer Science Engineering .* Retrieved from www.ijecse.org .