Edinburgh Napier UNIVERSITY

Coursework Report

Samuel Cattanach
40276600
Edinburgh Napier University - SET09122 Artificial Intelligence

1 Dijkstra's Shortest Path First Algorithm

First published in 1959 by Edsger Dijkstra, Dijkstra's shortest path first algorithm is the basis of many path-finding artificial intelligence and has been expanded upon by several algorithms such as A*. One of the main applications of Dijkstra's algorithm is finding the quickest routes between physical locations using paths such as roads. The algorithm is implemented to be able to find the optimal route, if a route exists, in any scenario. the information required to find a path is a specific starting node, an end node, and either the coordinates of all nodes or the distances between each node. Starting at the specified node, Dijkstra's algorithm will first use the connectivity matrix in order to go through all of the other nodes in order to find which ones can be visited from the starting node. For each neighbouring node that can be visited the distance of each is calculated. the neighbor which has the shortest distance traveled so far is selected as the next node regardless of whether it is closer to the specified end node or not. Two lists are maintained throughout the calculation of the shortest path. The first list is a priority queue consisting of nodes which have not been fully investigated yet which is sorted by the shortest distance first. The second list contains all of the nodes which have been fully investigated and will not be looked at again. Firstly, the traversable neighbouring nodes from the start point will be added to the priority queue and will have their traveled distances and parent nodes recorded along with their coordinates. The starting node, once all traversable neighbouring nodes are added to the priority queue, will be added to the second list of investigated nodes. The algorithm then continues through the nodes, calculating the shortest routes first, until the end node is reached. Once the goal has been reached, the final path traversed is established by following the parent of each node until a path from the first node to the end node has been established. Through proper implementation of Dijkstra's algorithm the optimal path from start to finish will always be found.

In regards to the cave problem described previously, Dijkstra's algorithm can be implemented in order to find the optimal route of any series of caves, and will also be able to identify if no route to the end cave is available. As the Euclidean distance can be calculated using each caves coordinates and a connectivity matrix of all of the caves is supplied, Dijkstra's algorithm will be able to start at cave one and work through every traversable neighbouring node until the last cave is reached. Dijkstra's shortest path first algorithm was first implemented on the ARMAC computer which had only 15 kilobytes of memory, because of this small amount of memory, there were limitations in how advanced Dijkstra's algorithm could be. The most major limitation in comparison to some of its successors was that it did not take into account any heuristic value for the nodes.

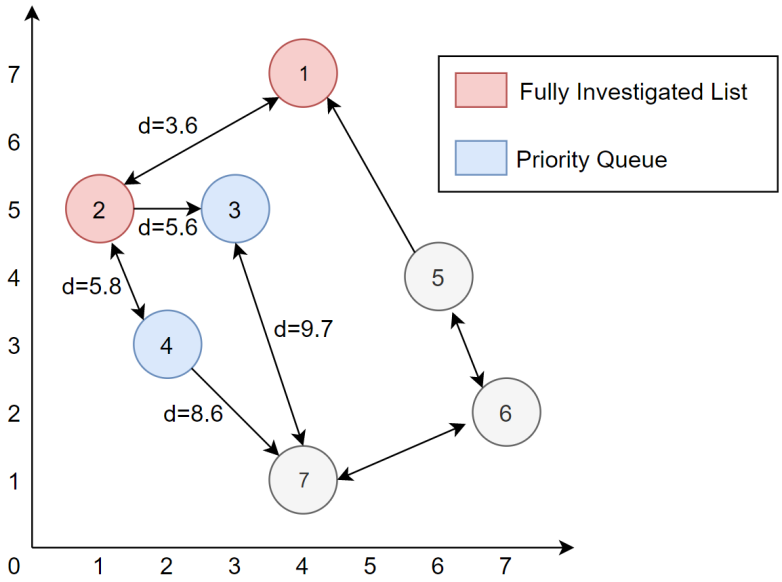

Figure 1: Dijkstra's algorithm

Since only the distance travelled so far is taken into consideration and not the distance from the end node, the algorithm may first calculate nodes leading away from the end node before going back towards it. Dijkstra's algorithm is usually implemented in uninformed environments, where a heuristic value can not be assigned to nodes, and the distance or cost between nodes may not be previously known. In this situation, with the absence of knowledge of the complete graph, Dijkstra's algorithm performs the same as with the A* algorithm as h(n) is 0 for every node and calculations can only be made based on g(n). Unlike with greedy best-first-search algorithms, because Dijkstra's algorithm does not move towards the end goal specifically but in an outwards motion from the start node, it can be very slow in comparison. As a result of this slow computational speed, the algorithm is inferior to others when used as an AI path-finding algorithm in some video games. Where there may not be many obstacles in the way of the start and end node, or the connectivity between nodes is high, the algorithm will waste time computing nodes which are farther away from its goal, shown in figure 2.

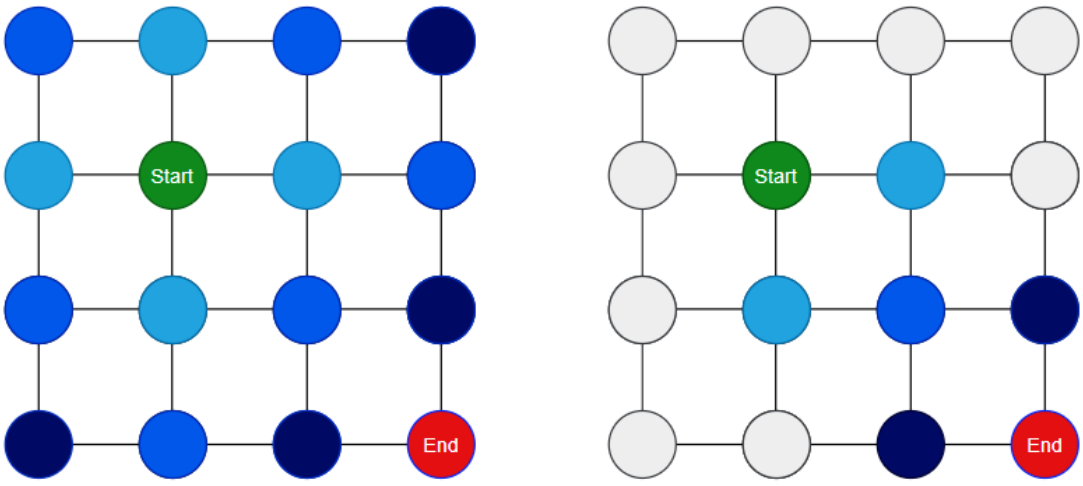

Figure 2: Dijkstra's algorithm (left) compared to A* (right)

2 The A* Path-Finding Algorithm

The informed best-first A* algorithm, first published in 1968 is widely considered the "most popular choice for path-finding, because it is fairly flexible and can be used in a wide range of contexts" (Patel, 2009)[1]. Given the limitations of Dijkstra's algorithm, one of its successors, the A* search algorithm allows for an optimal route to be discovered much faster in informed environments. A* would be a more suitable choice of algorithm for the given problem of navigating a system of connected caves because it takes into consideration not only the distance travelled, but a heuristic value as well. This heuristic value may be the Euclidean distance or the Manhattan distance for example. By calculating the Euclidean distance of each node (h(n)) and the exact distance travelled so far (g(n)), using the below formula (Fig 3) to determine which node to explore first provides an optimal solution at a much faster rate. If an ineffective heuristic value is given for each node the computational time will increase, although still faster or just as fast as Dijkstra's algorithm, but an optimal route will always be found.

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

Figure 3: A* shortest-first algorithm

In a situation where instead of using the Euclidean distance as the distance travelled, each tunnel in a particular direction had a specified cost, the A* algorithm would not be the most suitable choice in finding an optimal path. In this scenario, the only given information to use when deciding which node to explore first is g(n), the exact distance travelled so far. No heuristic value can be assigned to each node and therefor A* should not be used as it is an informed best-first search. Instead of using A* in this situation, Dijkstra's algorithm is the more suited implementation as it only requires a g(n) value and is preferable in uninformed environments.

References

[1] Patel, A. (2009). Introduction to A*. Retrieved from http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.