Pathfinding Algorithm Analysis

**Introduction**

My project consisted of comparing the following 3 algorithms, Dijkstra, A\* and Jump point search (JPS). They were compared on how long they took to find the goal node and as well how many nodes were searched in the process. This was all done on the adaptive depth maps, I choose to do it on 4 of those maps. I would also like to state that the graphs all had a uniform cost since JPS needs this to function properly. The results of each run where stored in text files.

**Algorithms Explained**

In this section I will briefly explain how all 3 of the algorithms work. Dijkstra’s algorithm work by checking all of the nodes in its vicinity and finding the one with the lowest cost from the start node. It then keeps finding the best node every time and adding to the open set which holds all of the explored nodes. A\* builds on this by adding a heuristic component which allows to better gauge if it’s getting closer to the goal node. Without the heuristic component to A\* it becomes Dijkstra’s algorithm. Now, JPS looks to further improve on A\* by removing unnecessary nodes which don’t need to be checked. It also exploits a certain feature inherent to grid based map representation. They are called symmetries, basically a symmetry is multiple same path lengths leading to the goal node. As you will see later on this makes an extreme difference in performance.

**Analysis**

After running all of these algorithms on 4 different maps each. The results came out to be the same for all of them. JPS seemed to outperform both Dijkstra’s algorithm and A\*. It outperformed them both in actual running time of the algorithm and as well as the number of nodes needed to search. I would also like to note here that the heuristic which A\* was using in these tests was the Euclidean distance. As mentioned before JPS allows to make certain assumptions which increases its speed about 10 to 20 times faster than A\*. However, it has weaknesses to it as well. A major one being that it can be only applied to uniform grid based graphs, while A\* is universal and can be applied to many different representations for maps. This really hinders the practicality of JPS.

For comparison and examples of the results achieved on the first map were as follows:

**Dijkstra:**

Time: 0:00:03.854982 seconds

Nodes searched: 8419

**A-Star:**

Time: 0:00:00.027055 seconds

Nodes searched: 660

**JPS:**

Time: 0:00:00.106284 seconds

Nodes searched: 69

As one can see the time difference and nodes explored are extremely large. Dijkstra’s algorithm searches 8419 nodes while A\* searches only 660. This however was nowhere close to the 69 nodes JPS had to search. Clearly, JPS is far superior in performance compared to the other two algorithms.

**Discussion**

I found this an issue in my implementation, I first implemented Dijkstra’s algorithm and A\* as they should be on a non-uniform graphs and then later needed to modify the grid to become uniform cost based for JPS. The many different resources that I used to complete this project also stated that this is a huge issue. However, in recent years there have been modifications on JPS and new version which can handle non-uniform grids have come out, such as JPS+.

**Conclusion**

After testing on all 4 for of the maps I can conclude that JPS is the superior search algorithm in terms of speed and numbers of nodes needed to be explored.