Jump Point Search Analysis

Bryan Tanner
Florida State University
bst12@my.fsu.edu

Abstract

The *A* algorithm* was developed to be an improvement over *Edsger Dijkstra's* original graph search algorithm, commonly known as *Dijkstra's algorithm*. A* was able to achieve this by using heuristics to intelligently choose paths that lead to the goal more quickly. This allows neighbors with higher cost to be pruned resulting in an optimal path being discovered. This algorithm has been detrimental in game development and robotic pathfinding. Now, nearly forty years after A* was proposed, new approaches are able to improve even more on this performance. By using *path expansion*, *jump points*, and *symmetry reduction* the *Jump Point Search* algorithm may be the future of AI in games and robotics.

1 Introduction

Jump Point Search (JPS) is a specialized, optimal algorithm used for traversing uniform-cost grid environments. The algorithm is ideal for traversing 8-way grid-based map representations, though it can be customized to accommodate other types of grids. JPS is consistently more than ten times faster than traditional A* implementations in benchmarks for modern games such as *Baldur's Gate II: Shadows of Amn* and *Dragon Age: Origins* (Harabor & Grastien, 2011). The general concept of JPS is path expansion and symmetry reduction in which the path is expanded quickly in the best known direction. This can be thought of as *intelligent expansion*, where neighbors can be cleverly pruned for a highly-focused, yet agile, search algorithm.

The common representation of maps as uniform-cost grids are found in a multitude of pathfinding environments. This contributes to a high level of path symmetry. "Unless handled properly, symmetry can force search algorithms to evaluate many equivalent states and prevents real progress toward the goal." (Harabor & Grastien, 2011)

JPS uses jump points to to expand selected nodes on the grid. Jump Points allow a traversal between two points without the need to expand the intermediate nodes. Along with reduced computational overhead, JPS also features optimal pathfinding, no need for preprocessing, and no memory overheads.

2 Prior Work

Several variants of the A* algorithm have addressed similar issues as JPS (Patel, 2013). In an attempt to reduce the memory overhead, *Beam Search* places a limit on the number of nodes stored in the open set. When the limit has been reached then the node with the worst-possible chance of finding the goal is dropped.

Iterative Deepening attemps to move ahead using a technique where a path is examined until its value only increases marginally, and it's assumed that this is as close to the goal as the current path will get. Next, another path is examined similarly, until a complete path has been found

Bidirectional search conducts two searches in parallel. One search starts at the beginning node and searches for the goal node while the other search starts at the goal node and works towards the start node. When the two paths meet, then the final path has been found.

*Theta** is very similar to JPS. The difference is that Theta* typically uses precomputation to find corners on the map. These corners are usually based on obstacles in the map. Theta* links paths between these corners to the goal while mainly ignoring the actual grids on the map during the search process.

Other specialized variations exist, however JPS is a more universal model. This holds true because grids are a very common way of representing maps whether it is for games, robotics, global positioning systems (GPS), simulations, or other AI applications.

3 Pathfinding with Jump Point Search

JPS can be implemented as an optimization to the A* algorithm with minor changes. JPS excels in large, open areas of a map. It is in these open areas that JPS can skip, or *jump*, over a large number of intermediate nodes that would otherwise be expanded using a traditional A* algorithm. Recognition of symmetry allows JPS to eliminate many other potential nodes as well. With a little more focus on calculation at each expanded node, JPS is able to eliminate large amounts of potential path nodes.

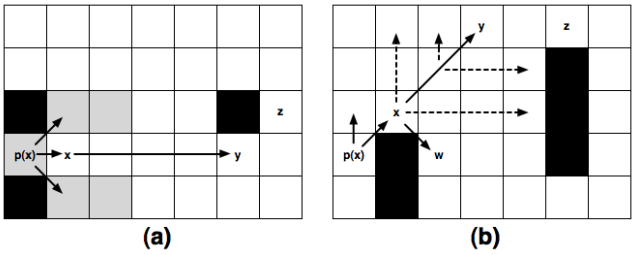

Figure 1: Examples of straight (a) and diagonal (b) jump points. Dashed lines indicate a sequence of interim node evaluations that reached a dead end. Strong lines indicate eventual successor nodes. (Harabor & Grastien, 2011)

3.1 Jump Points

Jump points are the basis of the JPS algorithm. An example of this idea for straight and diagonal jump points can be seen in Figure 1, respectively. The figure above makes it obvious that when moving in a stright line to the successor node that there is no need to evaluate any neighbor nodes along the path.

3.2 Intelligent Neighbor Pruning

Using A*, when expanding a new node, the neighbors of that node are usually added to the *open set* of nodes to be looked at. However, this isn't necessary for optimal pathfinding in most cases. If the parent nodes location in relation to the expanded node is considered then it is found that any of the expanded node's neighbors that could have been reached directly from the parent can be eliminated (Podhraski, 2013). This is a simple realization considering that if the neighbor node had been a better choice from the parent then it would have been expanded in the first place.

The exception to the rule is when expanding a node adjacent to a blocked node. In this case, the paths that could not be reached directly from the parent must be considered. It is important to understand that the currently expanded node is already the optimal path to that node, otherwise it would not have been expanded. This is the basis of pruning neighbors that would only serve as an intermediary to the currently expanded node (Witmer, 2013).

3.3 Symmetry Reduction

JPS experiences the largest efficiency improvements in large, open areas of maps. This allows jumping to the next *waypoint*, or checkpoint where the expanded node being jumped to is closer to the goal. But the calculation to choose the next jump node can be computationaly demanding. This is where symmetry reduction steps in to drastically reduce the size of the problem domain.

Explained simply, consider that the start node and goal node are on the same vertical row, meaning the goal node can be reached in a straight horizontal line from the start node. Then, one blocked node is placed directly in the center. This creates two parallel optimal paths around the blocked node to the goal. If the goal can be reached at the same cost using both paths then symmetry reduction should be used to remove one of these paths from consideration (Harabor D. , Shortest Path - Fast Pathfinding via Symmetry Breaking, 2011). This is one of the benefits of intelligent neighbor pruning.

4 Path Comparisons

The testing environment for the path comparisons were conducted using a Manhattan heuristic. The results show how A* and JPS pathfinding algorithms navigate through a simple maze.

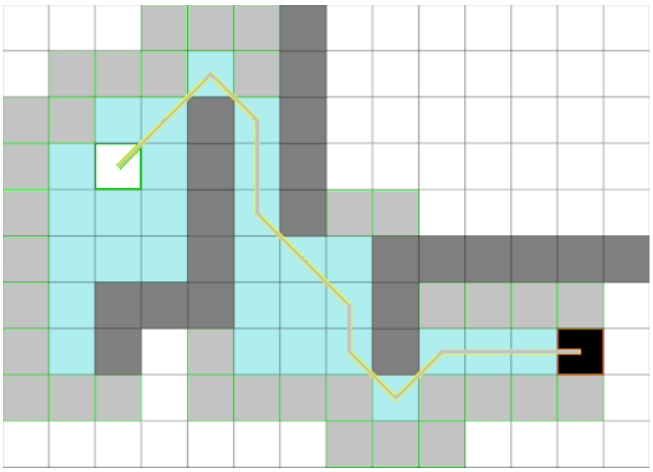

Figure 2: A path results (Xu, 2013)*

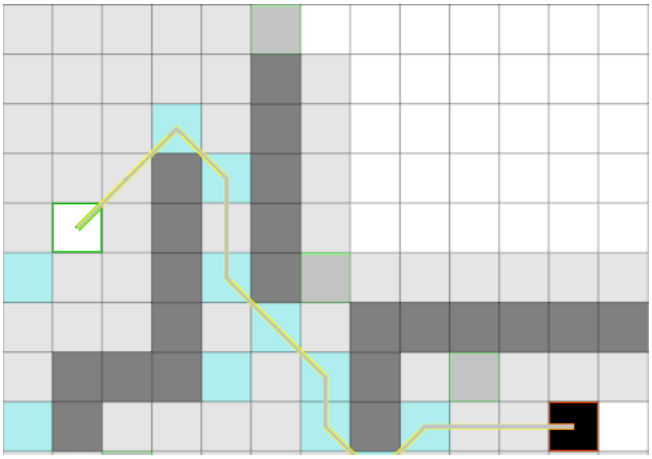

Figure 3: JPS path results (Xu, 2013)

In the above figures, white represents the start node, black is the goal node, dark gray nodes are blocked, light blue nodes are expaned nodes, and light gray nodes are frontier nodes. It's easy to see the reduced overhead of JPS. The algorithms expands on the jump points which are usually wall edges that create a path around the object. The result is far less nodes on the frontier and far less nodes expanded. It is also obvious from the figures that both return the optimal path.

4.1 Future Work

Figure 4: Output from JPS_Algorithm

Figure 4, above, shows the output from the JPS_Algorithm implementation I have constructed. This program takes in a mapfile consisting of blank nodes represented by the dash symbol, '-', a start node represented by the letter 'S', a goal node represented by the letter 'G', and blocked nodes represented by the letter 'X'. It then uses JPS and a Manhattan heuristic to navigate to the goal node. The resulting path is shown represented by the letter 'o'. The output is shown here. Not shown in Figure 4, the list of steps taken to arrive at the goal are displayed along with the calculated time it took to find the optimal solution in the program.

In the next iteration of this project I would like to abstract the grid and map logic into separate classes and substitute different search algorithms to devise comparison metrics. I would be interested to find out the speed differences between popular search algorithms such as Best-First, Breadth-First, HPA*, Djikstra, A* and Theta* to name a few. I would also be curious as to how these algorithms perform on different map types. It is my theory that JPS could rival Best-First Search in a straight line to the goal. My curiousity is more concerned with complex maps with several dead-ends and multiple paths to the goal with trivial differences.

5 Conclusion

When the A* algorithm was developed nearly a decade after Dijkstra's Algorithm, it revolutionized pathfinding methods. A* has stood strong as a versatile tool for searches. Now, new techniques are being developed that can exploit A* to get better performance in specific applications. Jump Point Search offers enhanced performance and lower memory cost than a traditional A* implementation for uniform-cost grid-based maps.

It's hard not to see a future for Jump Point Search in AI. A* has been established as a basic universal pathfinder, but the specialized algorithms are making headway and improving the metrics of A* tenfold in some areas.

References

Harabor, D. (2011, Aug 26). *Shortest Path - Fast Pathfinding via Symmetry Breaking*. Retrieved from www.wordpress.com: http://harablog.wordpress.com/2011/08/26/fast-pathfinding-via-symmetry-breaking/

Harabor, D. (2011, Sep 7). *Shortest Path - Jump Point Search*. Retrieved from www.wordpress.com: http://harablog.wordpress.com/2011/09/07/jump-point-search/

Harabor, D. (2011, Sep 1). *Shortest Path - Rectangular Symmetry Reduction*. Retrieved from www.wordpress.com: http://harablog.wordpress.com/2011/09/01/rectangular-symmetry-reduction/

Harabor, D., & Grastien, A. (2011). Online Graph Pruning for Pathfinding on Grid Maps. *Association for the Advancement of Artificial Intelligence (AAAI).*

Patel, A. (2013, Jul 18). *Variants of A**. Retrieved from www.stanford.edu: http://theory.stanford.edu/~amitp/GameProgramming/Variations.html

Podhraski, T. (2013, Mar 12). *How to Speed Up A* Pathfinding With the Jump Point Search Algorithm.* Retrieved from www.tutsplus.com: http://gamedev.tutsplus.com/tutorials/implementation/speed-up-a-star-pathfinding-with-the-jump-point-search-algorithm/

Witmer, N. (2013, May 5). *Jump Point Search Explained*. Retrieved from www.zerowidth.com: http://zerowidth.com/2013/05/05/jump-point-search-explained.html

Xu, X. (2013). *Pathfinding Visual*. Retrieved from www.github.com: http://qiao.github.io/PathFinding.js/visual/