The JPS Pathfinding System

Daniel Harabor and **Alban Grastien**
NICTA and The Australian National University
Email: firstname.lastname@nicta.com.au

Abstract

We describe a pathfinding system based on Jump Point Search (JPS): a recent and very successful search strategy that performs symmetry breaking to speed up optimal pathfinding on grid maps. We first modify JPS for grid maps where corner-cutting moves are not allowed. We then describe JPS+: a new derivative search strategy that reformulates an input graph into an equivalent symmetry-reduced form that can be searched more efficiently. JPS and JPS+ were both submitted to the 2012 Grid-based Path Planning Competition.

Introduction

Symmetry in grid-based pathfinding manifests itself when we consider a path, traditionally defined as an ordered sequence of nodes, as consisting instead of an ordered sequence of vectors. Each vector $\vec{d}$ is associated with one of the eight allowable movement directions (up, down, left, right etc.) and has a magnitude of either 1 or $\sqrt{2}$, depending on whether it represents a straight or a diagonal move. Such a formulation allows us to see that many paths on a grid map, which share the same start and end node but which pass through different intermediate nodes, are often just symmetric permutations of each other; i.e. they are identical save for the order in which the individual moves occur.

In the presence of symmetry, A* unnecessarily considers permutations of all shortest paths: from the start node to *every expanded node*. Jump Point Search (Harabor and Grastien 2011) is a simple but highly effective strategy that eliminates many such symmetries. JPS is fast, optimal, requires zero preprocessing, has zero memory overhead and appears orthogonal to recent pathfinding algorithms; e.g. (Björnsson and Halldórsson 2006; Pochter et al. 2010; Goldenberg et al. 2010). We first adapt JPS to grid domains where "corner-cutting" movement is not allowed. Then, we introduce JPS+: a new search method which reformulates an input graph into a symmetry-reduced equivalent that can be searched much faster.

Jump Point Search

Jump Point Search (JPS) is the combination of A* search with a simple node expansion operator that prunes potential successors if they can be reached by path which is shorter than, or symmetric to, the current path. JPS employs two simple sets of rules: pruning rules and jumping rules.

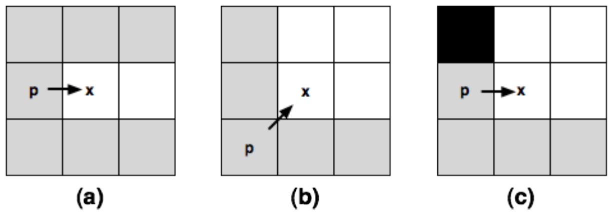

Figure 1: (a) When the move from p to x is straight only one natural neighbour remains. (b) When the move from p to x is diagonal, three natural neighbours remain. (c) Obstacles around x cause some neighbours to become forced.

Pruning Rules: Given a node x, reached via a parent node p, we prune from the neighbours of x any node n for which one of the following rules applies:

1. there exists a path $\pi' = \langle p, y, n \rangle$ or simply $\pi' = \langle p, n \rangle$ that is strictly shorter than the path $\pi = \langle p, x, n \rangle$;
2. there exists a path $\pi' = \langle p, y, n \rangle$ with the same length as $\pi = \langle p, x, n \rangle$ but π' has a diagonal move earlier than π.

We illustrate these rules in Figure 1(a) and 1(b). Observe that to test each rule we need to look only at the neighbours of the current node x. Pruned neighbours are marked in grey. Remaining neighbours, marked white, are called the *natural* successors of node x. In Figure 1(c) we show that obstacles can modify the list of successors for x: when the alternative path $\pi' = \langle p, y, n \rangle$ is not valid, but $\pi = \langle p, x, n \rangle$ is, we will refer to n as a *forced* successor of x. The set of forced successors in Figure 1(c) is different to the set identified in (Harabor and Grastien 2011). In that work we assumed corner-cutting (a.k.a. taking a diagonal shortcut around a corner) is allowed. Here we explicitly require that both π and π' respect any such domain-specific movement rules. When corner-cutting is not allowed this change has the following effect: only straight steps from p to x may produce forced neighbours and each x may have up to 4 such neighbours. This change preserves optimality; the argument is identical to the one in (Harabor and Grastien 2011).

Jumping Rules: JPS applies to each forced and natural

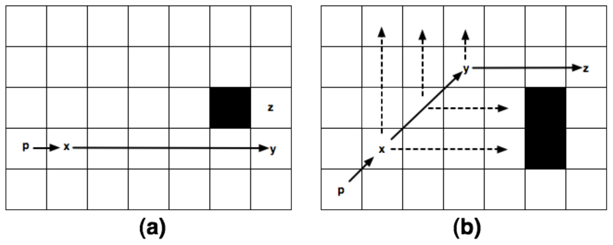

Figure 2: (a) A straight jump from x to y; the recursion stops due to node z which is forced. (b) A diagonal jump from x to y; the recursion stops due to a non-failed straight jump. Continuing would mean missing a potential turn due to z.

neighbour of the current node x a simple "jumping" procedure; the objective is to replace each neighbour n with an alternative successor n' that is further away. Precise details are given in (Harabor and Grastien 2011); we summarise the idea here using a short example:

Example 1. In Figure 1(a) pruning reduces the number of successors of x to a single node n. JPS exploits this property to immediately and recursively explore n. If the recursion stops due to an obstacle that blocks further progress (which is frequently the case), all nodes on the failed path, including n, are ignored and nothing is generated. Otherwise the recursion leads to a node n' which has a forced neighbour (or which is the goal). JPS generates n' as a successor of x; effectively allowing the search to "jump" from x directly to n' – without adding to the open list any intermediate nodes from along the way. In Figure 1(b) node x has three natural neighbours: two straight and one diagonal. We recurse over the diagonal neighbour only if both straight neighbours produce failed paths. This ensures we do not miss any potential turning points of the optimal path. □

In Figure 2(a) we illustrate a straight jump and in 2(b) a diagonal jump. By jumping, JPS is able to move quickly over the map without inserting nodes in the A* open list. This is doubly beneficial as (i) it reduces the number of operations and (ii) it reduces the number of nodes in the list, making each list operation cheaper. Notice that this version of JPS is performed entirely online, involves no preprocessing and has no memory overhead.

JPS+

Jumping from one point to another in the grid avoids many unnecessary A* open list operations. However, identifying these jump points becomes the new bottleneck of the algorithm. We therefore propose JPS+: an symmetry-breaking method which replaces each adjacent neighbour of a node with a jump point that lies in the same relative direction.

Figure 3(a) illustrates our graph reformulation idea for a single node x. We simply search for a jump point in the direction of each grid neighbour of x. In JPS, we discard all nodes along a failed path. By comparison, JPS+ must store the last node along a failed path. These *sterile jump points* are required to guarantee optimality during search but are never added to the A* open list. To see why they are necessary, consider Figure 3(b). Here we reach x from p and try to jump from x to y. Notice that each such jump may cross the goal or column of the target node t. In JPS the diagonal recursion would have terminated at node y', having detected the goal t along a non-failed straight jump. JPS+ simulates this behaviour by explicitly inserting an intermediate node y' at the point where the jump to y crosses the column of t. This condition is sufficient to preserve optimality during search. The proof involves showing that JPS+ simulates exactly the behaviour of JPS. We omit it for brevity.

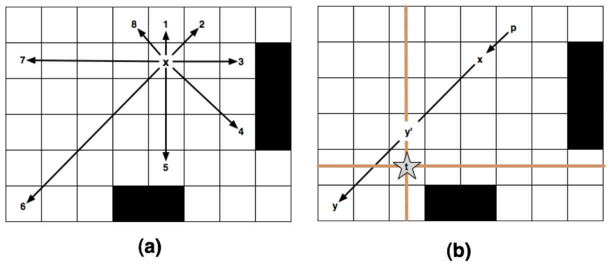

Figure 3: (a) A jump point is computed in place of each grid neighbour of node x. (b) When jumping from x to y we may cross the row or column of the target t (here, both). To avoid jumping over t we insert an intermediate successor y' on the row or column of t (whichever is closest to x).

Discussion

For the 2012 GPPC we configured JPS+ to reformulate the entire map during an offline phase. Precomputation time is at worst quadratic in the size of the input graph and produces a symmetry-reduced graph which is much faster to search. An alternative approach is to replace, during search, each forced or natural neighbour with a newly identified jump point successor. JPS+ requires very little memory: the reformulated graph is stored as an adjacency list that replaces the original input graph. When the input graph is also stored as an adjacency list, JPS+ has zero memory overhead.

Acknowledgments

We thank Patrik Haslum for helpful comments and discussion. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

Björnsson, Y., and Halldórsson, K. 2006. Improved heuristics for optimal path-finding on game maps. In *AIIDE*, 9–14.

Goldenberg, M.; Felner, A.; Sturtevant, N.; and Schaeffer, J. 2010. Portal-based true-distance heuristics for path finding. In *SoCS*.

Harabor, D., and Grastien, Al. 2011. Online graph pruning for pathfinding on grid maps. In *AAAI*.

Pochter, N.; Zohar, A.; Rosenschein, J. S.; and Felner, A. 2010. Search space reduction using swamp hierarchies. In *AAAI*.