Proceedings of the Ninth International
Symposium on Combinatorial Search (SoCS 2016)

Generalizing JPS Symmetry Detection: Canonical Orderings on Graphs

Nathan R. Sturtevant
University of Denver
Denver, CO, USA
sturtevant@cs.du.edu

Abstract

The Jump Point Search (JPS) algorithm is designed explicitly for search on grids; it uses grid-specific properties to reduce symmetry and provide faster optimal search without pre-computation. Recent work has broken the algorithm down into three components: a best-first search, a canonical ordering of states, and a jumping policy. This paper shows how a canonical ordering can be built on general graphs and used in a similar manner to the canonical ordering of JPS. This approach is able to significantly reduce the number of states generated by an A* search, but more work is needed to optimize and fully characterize the correctness of the approach.

Introduction and Background

There has been significant work on improving the speed of grid-based pathfinding algorithms (Sturtevant et al. 2015), particular using pre-computed storage. One of the few algorithms that performs well without pre-processing is Jump Point Search (JPS) (Harabor and Grastien 2011). Unfortunately, JPS has only been shown to be effective on grid maps, although it has potential applications on other domains with regular structure.

Recent work (Sturtevant and Rabin 2016) analyzed JPS, breaking it down into three independent components. These are (1) a best-first search, (2) a canonical ordering of states, and (3) a jumping policy. While JPS is only designed to work on grids, the pieces of this decomposition are not specific to grids. This paper shows how a canonical ordering can be used for successor pruning in general graphs, similar to JPS; we do not study jumping policies here.

Although canonical orderings have been used previously on general graphs, they have, to our knowledge, only been used to improve the efficiency of other techniques (Goldberg, Kaplan, and Werneck 2006; Antsfeld et al. 2012) as opposed to as a technique on their own. Furthermore, since we use canonical orderings for move pruning, they are not guaranteed to always be correct with a best-first search (Holte and Burch 2014).

This paper shows how a canonical ordering can be used, discusses the need for correctness, shows that a canonical ordering can reduce node generations, and shows that different canonical orderings have different performance.

Canonical Orderings

Canonical orderings have commonly been used to prune implicit state spaces (Holte and Burch 2014), but their use for pruning explicit state spaces has not been widely studied.

Potential Gain of a Canonical Ordering

One application of a canonical ordering is to reduce symmetries and thus the number of equal-cost optimal paths between two vertices in a graph. By removing redundant paths, the canonical ordering can prune successors, reducing the number of node generations during search.

In the JPS canonical ordering, the successors of a state are determined by the action that was used to reach the state. In the basic JPS canonical ordering, if a state was reached by an action moving to the east, then the only child generated from that state will be to the east. If the action reaching a state is north-east, then three children are generated via the actions north, east, and northeast. Analogous rules are defined for all directions, along with special rules for obstacles (Harabor and Grastien 2011). These rules reduce nodes generations, speeding up the search independent of the other components of JPS (Sturtevant and Rabin 2016).

We illustrate this using the first line of Table 1 and the graph in Figure 1. State A has neighbors S, B, and C. If, during search, we know that the parent of A is S, we can prune all successors except B, since C can be reached optimally through B, and S is the parent of B. This avoids generating two successors, and makes the search more efficient.

Need For Correctness

One approach for producing canonical orderings is to add a small amount of random noise to all edges in the graph, as this implicitly breaks ties. Goldberg et. al. (2006) used a variant of this approach, noting that they couldn't prove correctness, but in practice that it worked. Using Table 1, we illustrate a canonical ordering that prunes all possible paths between two states. In the second line we decide that B should have no successors when B is generated as a successor of A, because all successors can be reached optimally by other paths. However, when combined with the pruning from state A, a search from S can no longer reach C or D. The canonical ordering in this example can be fixed trivially; the point is that we must consider the possibility of a canonical ordering being incorrect.

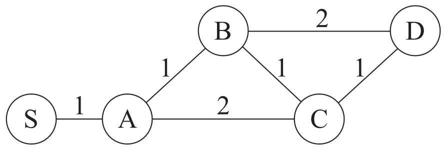

Figure 1: Canonical ordering example.

Table 1: A canonical ordering that cannot reach D from S.

(Parent →) State	All Successors	Canonical Successors
(S →) A	S, B, C	B (C via B)
(A →) B	A, C, D	– (C via A; D via C)

Local Computation Algorithm

We now propose a simple algorithm for generating a canonical ordering that we can use for pruning. We have not proved the correctness of the result, but we haven't yet, in limited experiments, found cases where it produces incorrect canonical orderings.

Recall that the valid successors of a state are determined by the parent of that state. For every state s in the graph we perform a Dijkstra search until the *neighbors of the neighbors* of s are reached. Then, the parent pointers are followed back to s. We illustrate this on the graph in Figure 1. We search from S until we reach A, B, and C. We then fill in the canonical successors of A according to the parent points of B and C. When coming from S, A is on the shortest path to both B and C, but not to S. So, S is not A's successor in the canonical ordering when reaching A via S. C can be reached optimally both directly from A and through B, so, depending on the tie-breaking in the search, C may or may not be considered a successor of A. This choice can influence the effectiveness of a canonical ordering; the question of finding the best canonical ordering is open.

Experimental Results

We have experimented on a number of maps and grids; due to space restrictions we present results on a single map here so we can compare the grid- and graph-based canonical orderings. We search on the map converted to a graph using A* (G-A*) and Canonical A*[1] (G-CA*). We also search directly on the grids using A*, and CA*. The canonical ordering on the graph is computed with our local algorithm from the last section; on grids we use the JPS canonical ordering. We report average results over 5000 random problems on the *brc203d* Dragon Age: Origins map (from (Sturtevant 2012)) in Figure 2. Results on the *AR0300SR* Baldur's Gate II map (not shown) are similar. All searches use the octile-distance heuristic.

We report time, states expanded, states generated, and the size of the open list at the end of the search. We store the canonical ordering in a hash table, not directly in the graph, so the time is not particularly relevant. The most important metric is the number of states generated. The canonical ordering in the graph reduces the number of state generations by a factor of 6.7. This is the primary impact of the canonical ordering. But, note that JPS's canonical ordering reduces state generations by a factor of 7.4. Since the search spaces and problems are identical, this indicates that there is a choice of canonical orderings, and the choice can impact performance.

[1]This is A* with a canonical ordering (see (Sturtevant and Rabin 2016))

Table 2: Results on the brc203d map with 20,712 states and 75,330 edges. The average path length was 192.1.

Alg	Time	Expanded	Generated	Open
G-A*	3.27ms	3,382	25,059	229
G-CA*	2.93ms	3,521	3,731	152
A*	2.36ms	3,382	25,059	229
CA*	1.14ms	3,387	3,386	91

Conclusions

This paper has shown that we can build and use canonical orderings on graphs to reduce the number of states generated during a search, similar to what is done in JPS. More work is needed to prove general correctness, study different canonical orderings, and to compare against other techniques which use constraints to prune states in graph search.

References

Antsfeld, L.; Harabor, D. D.; Kilby, P.; and Walsh, T. 2012. TRANSIT routing on video game maps. In *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE-12, Stanford, California, October 8-12, 2012.*

Goldberg, A. V.; Kaplan, H.; and Werneck, R. F. 2006. Reach for A*: Efficient point-to-point shortest path algorithms. In *Proceedings of the Eighth Workshop on Algorithm Engineering and Experiments, ALENEX 2006, Miami, Florida, USA, January 21, 2006*, 129–143.

Harabor, D. D., and Grastien, A. 2011. Online graph pruning for pathfinding on grid maps. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011.*

Holte, R. C., and Burch, N. 2014. Automatic move pruning for single-agent search. *AI Commun.* 27(4):363–383.

Sturtevant, N., and Rabin, S. 2016. Canonical orderings on grids. *International Joint Conference on Artificial Intelligence (IJCAI).*

Sturtevant, N. R.; Traish, J.; Tulip, J.; Uras, T.; Koenig, S.; Strasser, B.; Botea, A.; Harabor, D.; and Rabin, S. 2015. The grid-based path planning competition: 2014 entries and results. In *Eighth Annual Symposium on Combinatorial Search*, 241–251.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144 – 148.