Path Symmetries in Undirected Uniform-Cost Grids

Daniel Harabor and **Adi Botea** and **Philip Kilby**
NICTA and The Australian National University
Email: firstname.lastname@nicta.com.au

Abstract

We explore a symmetry-based reformulation technique which can speed up optimal pathfinding on undirected uniform-cost grid maps by over 30 times. Our offline approach decomposes grid maps into a set of empty rectangles, removing from each all interior nodes and possibly some from along the perimeter. We then add macro-edges between selected pairs of remaining perimeter nodes to facilitate provably optimal traversal through each rectangle. To further speed up search, we also develop a novel online pruning technique. Our algorithm is fast, memory efficient and retains both optimality and completeness during search.

Introduction

Pathfinding on undirected uniform-cost grid maps commonly appears in areas such as robotics (Lee and Yu 2009), artificial intelligence (Wang and Botea 2009) and video games (Davis 2000; Sturtevant and Geisberger 2010). To solve problems quickly practitioners usually apply hierarchical decomposition techniques such as HPA* (Botea, Müller, and Schaeffer 2004; Sturtevant and Geisberger 2010) or develop more accurate heuristics to guide search (Björnsson and Halldórsson 2006; Sturtevant et al. 2009; Goldenberg et al. 2010). Each of these has disadvantages: either the returned solutions are not guaranteed optimal or a substantial memory overhead is incurred.

In this paper we present Rectangular Symmetry Reduction (RSR): a graph pruning algorithm for undirected uniform-cost grid maps which is fast, memory efficient, optimality preserving and which can, in some cases, eliminate entirely the need to search. The central idea that we will explore involves identifying and eliminating path symmetries from the search space. Along the way we generalise a similar pruning method proposed in (Harabor and Botea 2010), which is limited to 4-connected uniform-cost grid maps.

When compared to related methods from the literature we find RSR has complementary strengths. We identify classes of instances where RSR is clearly the better choice and show that it can often dominate convincingly across a variety of synthetic and realistic benchmarks.

Related Work

In the presence of symmetry, search algorithms often waste time evaluating many equivalent states and make little real progress toward the goal. The problem of how to deal with symmetry has received significant attention in other parts of the literature (e.g. (Rossi, Beek, and Walsh 2006)) but few studies focus on symmetry in pathfinding domains, such as grid maps. We are only aware of Empty Rectangular Rooms (Harabor and Botea 2010): an offline symmetry-breaking technique limited to 4-connected uniform-cost grid map; the connection with RSR is discussed in the next section.

The *dead-end heuristic* (Björnsson and Halldórsson 2006) and *Swamps* (Pochter et al. 2010) are two closely-related pruning techniques that identify areas in the search space not relevant for reaching the goal. This is a similar yet complementary goal to RSR, which tries to reduce the search effort involved in exploring any given area.

The *gateway heuristic* (Björnsson and Halldórsson 2006) and the *portal heuristic* (Goldenberg et al. 2010) are two typical memory-based techniques for optimal pathfinding on grids. The main idea is to reduce the number of A* node expansions by improving the accuracy of cost-to-go estimates during search. The portal heuristic also identifies, online, areas not relevant to the pathfinding instance at hand.

Contraction Hierarchies (Geisberger et al. 2008) is a method for very fast optimal pathfinding on road networks. Based on a combination of Dijkstra's algorithm and memory-intensive abstractions, this approach relies on the existence of "highway edges" that appear on most shortest paths between nodes. Largely orthogonal to RSR, there is little work applying these ideas to searching on grid maps. One recent result (Sturtevant and Geisberger 2010) suggests they are less effective when the underlying graph contains a high degree of path symmetry.

Rectangular Symmetry Reduction

In the AI literature a path in a graph is usually written as: $\langle v_0, \ldots, v_n \rangle$. Here v_0 is the initial (or start) node, v_n is the target (or goal) node and each pair of adjacent nodes represents an edge in the graph. In the context of grid maps, it is sometimes helpful to equivalently describe a path as an ordered sequence of actions $\langle \vec{d_0}, \ldots, \vec{d_n} \rangle$ where each action $\vec{d_i}$ is a vector representing a direction of travel. When executed

by an agent, the effect of each action is to move the agent from the node it currently occupies on the grid (initially v_0) to one of the eight immediately neighbouring nodes. The last action results in the agent occupying the target node v_n. Using this formulation, a path is valid if no action results in the agent moving to a location which is an obstacle.

Definition 1. *Two valid paths π_1 and π_2 are symmetric if they share the same start and goal node and one can be derived from the other by swapping the order of the actions.*

To identify and eliminate path symmetries from the grid we will employ the high level strategy in Algorithm 1.

Algorithm 1 Graph reduction based on empty rectangles

Require: A grid map

1. Decompose the grid map into a series of disjoint obstacle-free rectangles; e.g. as in (Harabor and Botea 2010). The size and placement of rectangles can vary, depending on the positions of obstacles.
2. Prune all tiles from the interior of each rectangle R and possibly some from the perimeter (border).
3. Add to each rectangle R a series of *macro edges* between selected pairs of tiles from the perimeter. The cost of each edge is equal to the Octile (or Manhattan, on 4-connected grids) distance between endpoints.
4. During search, temporarily re-insert tiles back into the map to handle cases where the start or goal is a location which has been previously pruned.

RSR is similar to 4ERR (Harabor and Botea 2010), a symmetry breaking algorithm limited to 4-connected grid maps. The main differences are: (i) we generalise 4ERR to 8-connected grid maps (ii) we give a stronger offline pruning operator to eliminate more nodes from the grid (iii) we give a new online pruning operator that further speeds up search.

Optimal Travel via Macro-Edges

After interior nodes are eliminated, macro-edges have to be added between perimeter nodes to ensure rectangles are traversed optimally. We will describe a strategy that adds only *non-dominated* macro edges; i.e. edges whose lengths are strictly less than the length of any alternative path between the same pair of nodes. There are three cases to discuss. In each case the length of each added macro-edge is equal to the heuristic (or Octile) distance between its two endpoints.

Case 1: nodes on the same side of the perimeter are connected just as in the original grid. **Case 2:** nodes on orthogonal sides of the perimeter are connected *iff* the shortest path between them is a diagonal (45-degree) line; this is illustrated in Figure 1 (a). **Case 3:** nodes on opposite sides of the perimeter. For each such node we generate a "fan" of neighbours from the opposite side; this is shown in Figure 1 (b). Starting from a node such as t_1 we step to the closest neighbour from the opposite side and extend the fan away from the middle, adding each node we encounter. The last node on either side of the fan is placed diagonally, at 45 degrees, from t_1 (such as t_2) or located in the corner of the perimeter

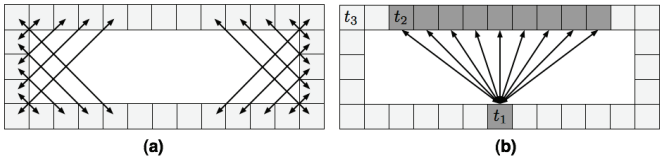

Figure 1: (a) Macro edges between nodes on orthogonal sides of an empty rectangle. (b) Each node on the perimeter is connected to a set of nodes on the opposite side.

(whichever is encountered first). Other nodes, such as t_3, can be reached optimally via the path $\langle t_1, t_2, \ldots, t_3 \rangle$.

Lemma 1. *Let R be an empty rectangle in an 8-connected grid map. Let m and n be two perimeter locations. Then, m and n can be connected optimally through a path that contains only non-dominated macro-edges.*

Proof. : We split the proof over the 3 cases discussed earlier. In the first case we walk along the perimeter from m to n; the optimality of this path is immediate. In the second and third case the two nodes can be connected through an optimal path that has one diagonal macro-edge (at one end of the path) and zero or more straight macro-edges. See again the example of travelling from t_1 to t_3 in Figure 1 (b). □

Node Insertion: When the start or goal is located in the interior of an empty rectangle we will temporarily re-insert them into the graph for the duration of a search. If the start and goal are from the same room no insertion is necessary; an optimal path is trivially available. Otherwise, we add four "fans" (collections) of macro edges. Each fan connects the start (goal) node to a set of nodes on one side of the rectangle's perimeter. Fans are built as shown earlier. To prove optimality we run the argument given for Case 3 of Lemma 1, substituting m for the newly inserted node.

Theorem 1. *For every optimal path π on an original grid, there exists an optimal path π' on the reformulated grid such that π and π' have the same cost.*

Proof. Consider a rectangle R that is crossed by an optimal path π in the original grid. Let m and n be the two perimeter points along π. According to Lemma 1, there is a way to connect m and n optimally in the modified graph. Thus, we replace the original path segment $\langle m, \ldots, n \rangle$ in π with a cost-wise equivalent from the modified grid. The case when m (or n) is the start or goal node is addressed similarly. By performing such a replacement for all rectangles intersected by π, we obtain a path π' that satisfies the desired properties. □

Reducing The Branching Factor Further

In this section we discuss an offline perimeter pruning technique and an online branching factor reduction strategy. Both retain solution optimality (proofs omitted).

Perimeter Reduction: To speed up search we will prune all perimeter nodes which have no neighbours in any adjacent rectangle. To preserve optimality, we connect the neighbours of each pruned node directly to each other. The weight

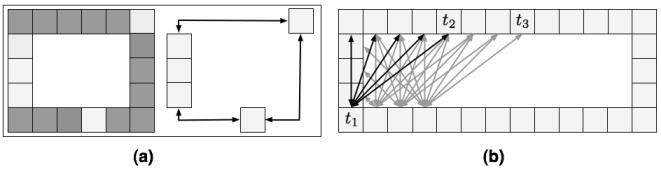

Figure 2: (a) We prune all (dark grey) nodes which have no neighbours in any adjacent rectangle (left). Remaining nodes (right) are then connected directly. (b) Assume t_1 is the parent of t_2. When t_2 is expanded, we do not generate neighbors from the opposite side. These can be reached from t_1 via a shorter or equal-length path.

of each new edge is equal to the octile distance between the two neighbours. Figure 2 (a) shows an example.

Online Node Pruning: When expanding a node during search we observe that if the current node, and its parent, belong to the same rectangle then it is not necessary to consider any successors from the opposite side of the rectangle. Figure 2 (Right) shows an example of such a situation. When the current node has no parent, or the parent belongs to a different rectangle, we process all its successors.

Memory Requirements

RSR requires a memory overhead which is linear in the size of the search graph: i.e. $O(|V|)$. In our implementation we stored the id of the parent rectangle for each node in the original grid. We also stored, for each identified rectangle, its height, width and the coordinates of its origin in the grid. Further overheads, such as storing the set of macro-edges for each perimeter node, can be avoided by exploiting the simple geometric nature of empty rectangles. During node expansion we calculate, on-the-fly and in constant time, the exact position of the two nodes at the edge of each "fan" of neighbours from the opposite side of the perimeter. We can then simply travel along the perimeter, from one end of the fan to the other, and generate each non-pruned node we encounter. Other neighbours, from the same side of the perimeter as the current node, or an orthogonal side, can be similarly identified on-the-fly and in constant time.

Experimental Setup

We evaluate RSR on three benchmarks from the University of Alberta's freely available pathfinding library Hierarchical Open Graph (HOG): **Adaptive Depth** is a set of 12 synthetic maps of size 100×100. **Baldur's Gate** is a set of 120 maps from BioWare's popular roleplaying game *Baldur's Gate II: Shadows of Amn*. They range in size from 50×50 to 320×320 and often appear as a standard benchmark in the literature (Björnsson and Halldórsson 2006; Harabor and Botea 2010; Pochter et al. 2010). **Rooms** is a set of 300 maps of size 256×256, each divided into small rectangular areas (7×7). Rooms has previously appeared in (Sturtevant et al. 2009; Pochter et al. 2010; Goldenberg et al. 2010).

We used two copies each map: one in which diagonal transitions are allowed and another in which they are not.

	Adaptive Depth	Baldur's Gate	Rooms
4ERR	3.05	2.18	2.11
4ERR+PR	4.24	2.47	17.81
4ERR+OP	3.66	2.50	2.67
RSR	4.89	2.78	18.19
8-connected Maps			
Swamps	1.60	2.91	4.66
RSR	3.94	2.18	7.91
Scaled-up Maps (8-connected)			
Swamps	2.04	4.03	7.22
RSR	9.38	4.98	30.99

Table 1: Avg. A* search time speedup on each benchmark.

For each map we generated 100 valid problem instances, checking that every instance could be solved both with and without the use of diagonal transitions. Our test machine had a 2.93GHz Intel Core 2 Duo processor, 4GB RAM and ran OSX 10.6.2. Our implementation of A* is based on one provided in HOG, which we adapted to facilitate our online node pruning enhancement.

Results

To evaluate RSR we use a generic implementation of A* and discuss performance in terms of search time speedup. That is, the relative improvement to the average time A* needs to solve an instance when running on a pruned vs. unpruned grid. For example, a speedup of 2.0 is twice as fast (higher is better). Note that on approximately 2% of all instances the start and goal are located in the same rectangle and RSR computes the optimal solution without search. We exclude these instances from our results on the basis that they are outliers, even though RSR solves them in constant time.

Pre-processing Times: RSR took very little time to preprocess each map from our three benchmark sets. Adaptive Depth required 0.1 seconds per map on average; Rooms and Baldur's Gate required 0.39 and 0.65 seconds respectively.

Comparison to 4ERR: We now compare the performance of RSR against the 4ERR (Harabor and Botea 2010) symmetry-breaking algorithm. Here we restrict our attention to 4-connected maps. To assess the individual impact of both perimeter reduction (PR) and online node pruning (OP) we also develop and compare two variant algorithms: 4ERR+PR and 4ERR+OP. Table 1 (rows 1-4) presents our main result. RSR dominates convincingly across all instances allowing us to conclude it is the better choice on 4-connected maps. Of the variants, 4ERR+PR yields the biggest improvement, speeding up A* by up to 20 times. 4ERR+OP compares well on Adaptive Depth and Baldur's Gate but is of little benefit on Rooms where perimeter pruning has already reduced the branching factor.

Comparison to Swamps: Next, we compare RSR with Swamps (Pochter et al. 2010). We used the authors' source code, including their own implementation of A*, and ran all experiments using their recommended parameters: a swamp seed radius of 6 and "no change limit" of 2. Table 1 (rows 5-6) gives the main result on the 8-connected variants of

Algorithm	Extra Memory	Baldur's Gate	Rooms
PH-e	$2\|V\|$	3.16	11.9
PH-e	$8\|V\|$	3.07	17.54
RSR	$\|V\|$	2.8	18.2

Table 2: Avg. A* search time speedup: RSR vs PH-e. RSR figures are across all maps on each benchmark. PH-e figures are for a small subset selected by its authors (1 of 120 from Baldur's Gate and 5 of 300 from Rooms).

our three benchmarks. On Adaptive Depth and Rooms, where the terrain can be naturally decomposed into rectangles, RSR achieves higher speedups and is shown consistently better than Swamps. On Baldur's Gate, where this is not the case, Swamps-based pruning is more effective.

To measure the effect that larger open areas have on search time, we scaled every map in each benchmark by a factor of 3. We then generated 100 new instances per map. Table 1 (rows 7-8) shows the results. The overall gain for Swamps is very small while RSR shows dramatic improvement. This is not surprising: symmetry reduction quickly explores large areas that must be searched while Swamps-pruning avoids areas that do not need to be searched. Since the two ideas are orthogonal, a natural extension would be to combine them: first, apply symmetry reduction to a grid; then, apply a Swamps-based decomposition to the resultant graph.

Comparison to Portal Heuristic: Table 2 compares RSR against published results for PH-e: the enhanced Portal Heuristic algorithm (Goldenberg et al. 2010). As in that work we focus on 4-connected variants of Baldur's Gate and Rooms. PH-e seems to perform well when it can decompose the map into areas of similar size with few transitionary nodes. Smaller is better but requires more memory. Coarser decompositions require less memory but paths take longer to refine. By contrast, RSR performs well when it can decompose the map into (ideally large) rectangles with few perimeter nodes. On Rooms, both decomposition approaches are highly effective. On Baldur's Gate both are comparatively less effective. Notice however that PH-e requires up to 7 times more memory than RSR to achieve similar results. As with Swamps, we believe PH-e is entirely orthogonal to RSR: e.g. it could be used to guide search on a map pruned by RSR. Alternatively, symmetry elimination could be used to speed up pathfinding during PH-e's refinement phase.

Conclusion

We introduce Rectangular Symmetry Reduction (RSR), a new symmetry-breaking algorithm applicable to pathfinding on undirected uniform-cost grid maps. RSR is fast, memory efficient, optimality preserving and can, in some cases, eliminate entirely the need to search. After symmetry elimination A* can search some grids over 30 times faster.

Compared to the 4ERR (Harabor and Botea 2010) pruning algorithm, on which it is based, RSR's performance dominates convincingly. We also show that RSR is complementary to and often faster than Swamps-based pruning (Pochter et al. 2010). We find that Swamps are more useful on maps with small open areas while RSR becomes more effective as larger open areas are available on a map. When compared to the enhanced Portal Heuristic (Goldenberg et al. 2010), we find that RSR has similar or improved performance but requires up to 7 times less memory. As with Swamps, this method can also be combined with RSR.

An interesting direction for future work is applying RSR in settings involving dynamic environments: for example real-time strategy games where existing obstacles may be destroyed or new ones introduced. Another interesting topic is combining RSR with Swamps or the Portal Heuristic.

Acknowledgements

We would like to thank Alban Grastien and Patrik Haslum for providing feedback on early drafts of this work. We also thank Ariel Felner and Meir Goldenberg for providing us with assistance in comparing RSR with their enhanced Portal Heuristic algorithm.

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.

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *J. Game Dev.* 1(1):7–28.

Davis, I. L. 2000. Warp speed: Path planning for Star Trek Armada. In *AAAI Spring Symposium (AIIDE)*, 18–21.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *WEA*, 319–333.

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

Harabor, D., and Botea, A. 2010. Breaking path symmetries in 4-connected grid maps. In *AIIDE*, 33–38.

Lee, J.-Y., and Yu, W. 2009. A coarse-to-fine approach for fast path finding for mobile robots. In *IROS*, 5414 –5419.

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

Rossi, F.; Beek, P. v.; and Walsh, T. 2006. *Handbook of Constraint Programming*. New York, NY, USA: Elsevier Science Inc.

Sturtevant, N. R., and Geisberger, R. 2010. A comparison of high-level approaches for speeding pathfinding. In *AIIDE*, 76–82.

Sturtevant, N. R.; Felner, A.; Barrer, M.; Schaeffer, J.; and Burch, N. 2009. Memory-based heuristics for explicit state spaces. In *IJCAI*, 609–614.

Wang, K.-H. C., and Botea, A. 2009. Tractable multi-agent path planning on grid maps. In *IJCAI*, 1870–1875.