Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling

Speeding-Up Any-Angle Path-Planning on Grids

Tansel Uras and **Sven Koenig**
Department of Computer Science
University of Southern California
Los Angeles, USA
{turas, skoenig}@usc.edu

Abstract

Simple Subgoal Graphs are constructed from grids by placing subgoals at the corners of obstacles and connecting them. They are analogous to visibility graphs for continuous terrain but have fewer edges and can be used to quickly find shortest paths on grids. The vertices of a Simple Subgoal Graph can be partitioned into different levels to create N-Level Subgoal Graphs, which can be used to find shortest paths on grids even more quickly by ignoring subgoals that are not relevant for the search, which significantly reduces the size of the graph being searched. Search using Two-Level Subgoal Graphs was a non-dominated entry in the Grid-Based Path Planning Competitions 2012 and 2013.

In this paper, we take advantage of the similarities between Subgoal Graphs and visibility graphs to show that Subgoal Graphs can be used, with small modifications, to quickly find "any-angle" paths, thus extending their applicability. Any-angle paths are usually shorter and more realistic looking than grid paths since the movement along any-angle paths is not constrained to grid edges. Our algorithm has the advantage that it is a simple extension of searching Subgoal Graphs and is up to two orders of magnitude faster than Theta* and up to an order of magnitude faster than Block A* (using 5×5 blocks), two of the most well-known any-angle path-planning algorithms, while still finding any-angle paths of comparable lengths.

Introduction

Grids are often used in video games and robotics to discretize a continuous environment into a graph, which can be searched with an optimal path-planning algorithm, such as A* (Hart, Nilsson, and Raphael 1968), to find shortest paths. Movement on grids is typically constrained to grid edges and, as a result, the paths found (grid paths) can be unrealistic looking and longer than shortest paths in the continuous environment. One can address this issue by smoothing grid paths by replacing local, sub-optimal parts of the paths by straight lines (Thorpe 1984; Botea, Müller, and Schaeffer 2004). These smoothed paths can still be long, however, since their homotopy often remains unchanged.

Any-angle path-planning algorithms, such as Theta* (Nash et al. 2007; Daniel et al. 2010; Nash, Koenig, and Tovey 2010; Nash 2012), Block A* (Yap et al. 2011b; 2011a) and Field D* (Ferguson and Stentz 2006), address this issue by interleaving path smoothing with search. Similar to A*, they propagate information along grid edges during search, but the movement is no longer constrained to grid edges. Although they are not guaranteed to find shortest paths in the continuous environment, the paths tend to be shorter than smoothed grid paths. To find shortest any-angle paths one can use visibility graphs (Lozano-Pérez and Wesley 1979), Accelerated A* (Sislak, Volf, and Pechoucek 2009b; 2009a), or Anya (Harabor and Grastien 2013). Visibility graphs tend to have very high vertex degrees and, as a result, searching them is usually slow. Accelerated A* is a variant of Theta* that is only assumed to find shortest any-angle paths. We are not aware of any work that evaluates Anya's efficiency.

The contribution of this paper is to show that Subgoal Graphs (Uras, Koenig, and Hernández 2013; Uras and Koenig 2014) can, with small modifications, be used to quickly find any-angle paths. For many search problems, the graph is known beforehand and there is time to preprocess it to make the search faster. Simple Subgoal Graphs are constructed from grids during a preprocessing phase by placing subgoals at the corners of obstacles and connecting pairs of subgoals that are *direct-h-reachable*. The resulting graph is essentially a sparser visibility graph that can be used to find shortest grid paths. N-Level Subgoal Graphs are constructed from Simple Subgoal Graphs by partitioning the subgoals into different levels (similar to Contraction Hierarchies (Geisberger et al. 2008; Dibbelt, Strasser, and Wagner 2014), which we discuss later), allowing the searches to ignore many subgoals while still finding shortest grid paths. We discuss the details of these algorithms and how they can be used to find any-angle paths in the following sections.

Preliminaries

The algorithms described in this paper work on 8-neighbor grids. Any-angle path-planning algorithms typically place the vertices at the corners of the grid cells, rather than their centers. A grid path is a sequence of cell corners where consecutive pairs of cell corners must belong to the same grid cell. For the purpose of this paper, we define an any-angle

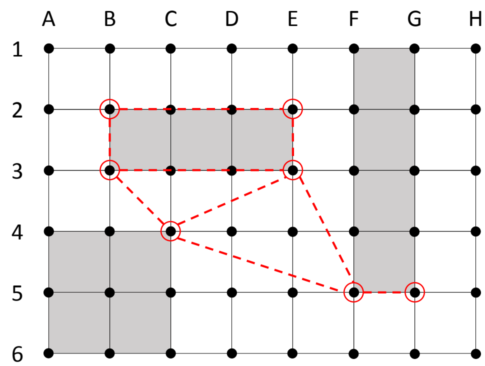

Figure 1: Simple Subgoal Graph.

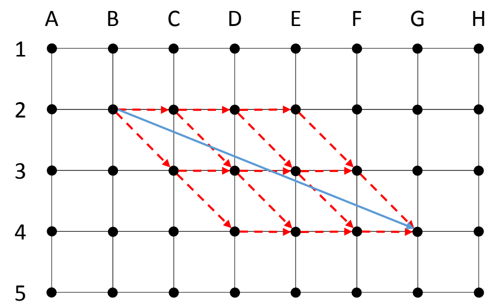

Figure 2: Shortest grid paths and shortest any-angle path between B2 and G4.

path as a sequence of cell corners where consecutive pairs of cell corners have line-of-sight (the straight line between them does not pass through the interior of blocked cells).

Simple Subgoal Graphs

Simple Subgoal Graphs (SSGs) are constructed from grids whose vertices are placed at the centers of grid cells. In this paper, however, we place the vertices of the grid at the corners of grid cells as any-angle path-planing algorithms typically do. Except for some small implementation details, this does not change how SSGs are used.

We start with some definitions: A cell corner s is called a *subgoal* if and only if it is a convex corner of an obstacle, that is, exactly one or two of its four adjacent grid cells are blocked and, in the latter case, the two blocked cells are on the same diagonal. Two cell corners s and u are called *h-reachable* if and only if there is a shortest grid path between them whose length is equal to the Octile distance (that is, the length of a shortest grid path assuming the grid has no blocked cells) between them. They are called *direct-h-reachable* if and only if they are h-reachable and none of the shortest grid paths between them pass through a subgoal (except for s and u).

Simple Subgoal Graphs are constructed by adding edges between all pairs of direct-h-reachable subgoals. The length of each edge is the Octile distance between the subgoals it connects. Figure 1 shows an example of an SSG. Observe that B3 and F5 are h-reachable but not direct-h-reachable (due to the subgoal at C4), so there is no edge between them.

To find shortest grid paths using SSGs, one connects the given start and goal vertices s and g to all of their respective direct-h-reachable subgoals and searches this graph with A* to find a sequence of direct-h-reachable subgoals between s and g, called a *shortest high-level path*. One can then determine a shortest grid path between consecutive subgoals to find a shortest grid path between s and g. For instance, if we were to use the SSG in Figure 1 to find a shortest grid path between A2 and G4, we would add the edges (A2, B2), (A2, B3) and (G4, G5) to the SSG and search this graph to find the shortest high-level path A2-B3-C4-F5-G5-G4. Following this high-level path on the grid, we obtain the shortest grid path A2-B3-C4-D5-E5-F5-G5-G4.

Identifying all direct-h-reachable subgoals from a given cell corner can be done efficiently with a dynamic programming algorithm that uses precomputed clearance values. Using this algorithm, SSGs can be constructed within milliseconds and the start and goal vertices can be connected to the SSGs quickly before a search. Previous results indicate that, using SSGs, one can find shortest grid paths $\sim$24 times faster than A* on some video game maps, while requiring only a couple of MBs of extra memory to store the SSGs.

Any-Angle Simple Subgoal Graphs

We start this section by discussing the relationship between direct-h-reachability and visibility. Figure 2 shows all shortest grid paths between B2 and G4 with red dashed lines and the shortest any-angle path between them with a solid blue line. The shortest grid paths between B2 and G4 cover a parallelogram-shaped area. If two cell corners s and u are direct-h-reachable, then the parallelogram-shaped area between them, called P, cannot intersect with the interior of any blocked cells. This is so, because, otherwise, the set of blocked cells whose interiors intersect with P either cause s and u to be no longer h-reachable, or introduce a subgoal on one of the shortest paths between them (Uras, Koenig, and Hernández 2013). The straight line between s and u traverses only cells whose interiors intersect with P, which are unblocked if s and u are direct-h-reachable. Therefore, *if two cell corners are direct-h-reachable, then they have line-of-sight*.

This property implies that all edges of an SSG are visibility (graph) edges (connecting subgoals that have line-of-sight). Therefore, the high-level path found by searching an SSG is also an any-angle path. However, not all visibility edges are included in an SSG. For instance, in Figure 1, there is no edge between E2 and F5 even though they have line-of-sight. Since SSGs can be used to find shortest grid paths and since shortest grid paths are at most 8% longer than shortest any-angle paths (Nash and Koenig 2013), we can consider SSGs as sparser visibility graphs with a suboptimality bound of 8% if one does not refine the high-level path.

We therefore make the following changes to use SSGs to find short any-angle paths quickly: 1) The high-level paths are already valid any-angle paths, so we do not need to refine them into a low-level paths on the grid. 2) We use the

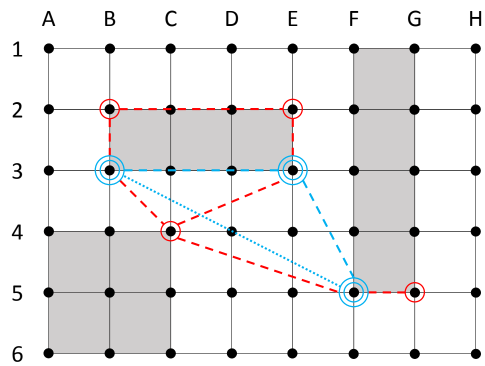

Figure 3: A Two-Level Subgoal Graph constructed from the Simple Subgoal Graph in Figure 1. The number of circles around a cell corner depict the level of the subgoal. Dashed lines are the original edges of the subgoal graph, and the dotted line depicts the extra edge added to the graph during partitioning.

Euclidean distance rather than the Octile distance as edge lengths and heuristics for the search. This results in shorter any-angle paths because the search is now trying to minimize the length of an any-angle path rather than the length of a grid path. 3) We use Theta* for the search instead of A*. Theta* is a variant of A* that finds any-angle paths on graphs embedded in 2D or 3D environment, such as SSGs. When expanding a vertex s, Theta* checks for each successor u of s if the parent of s and u have line-of-sight. If so, it sets the parent of u to the parent of s and sets the g-value of u accordingly. This results in shorter any-angle paths because the search can generate shortcut edges that are not in the graph.

(Any-Angle) N-Level Subgoal Graphs

N-Level Subgoal Graphs are constructed from SSGs by repeatedly performing the following procedure, called *partitioning*, starting with the SSG: 1) Identify a maximal set of subgoals S (also called *local subgoals*), such that removing any subset of S from the graph does not increase the lengths of shortest paths between the remaining subgoals. 2) Set the level of all subgoals in S to i, where i is the number of times partitioning has already been performed, including the current instance. 3) Remove all subgoals $s \in S$ from the graph (along with their adjacent edges). 4) If $S = \emptyset$ or the remaining graph has no subgoals (or a user defined number of levels have been created), set the level of all remaining subgoals to $i + 1$ and stop. Otherwise, repeat the procedure with the remaining graph.

One can allow partitioning to add new edges to the graph (discussed below in more detail) in order to classify more vertices as local subgoals. Figure 3 shows a Two-Level Subgoal Graph constructed from the SSG in Figure 1. Without the extra edge between B3 and F5, removing C4 would increase the length of a shortest path between B3 and F5 and, therefore, C4 could not be classified as a local subgoal during the first partitioning.

To find shortest paths using N-Level Subgoal Graphs, one first connects the given start and goal vertices s and g to all their respective direct-h-reachable subgoals, identifies all subgoals reachable from s and g via *ascending edges* (edges from a subgoal to higher-level subgoals, s and g are assumed to have level 0 if they are not subgoals) and searches the graph consisting of those subgoals and all highest-level subgoals (and the edges between them), thus ignoring other subgoals during the search. For instance, if one were to use the Two-Level Subgoal Graph in Figure 3 to find a path between A2 and G4, the graph searched would include B2 and G5 but not C4 or E2. Without the extra edge between B3 and F5, C4 would be a highest-level subgoal and also be included in the search.

If one does not specify a constraint on the edges that partitioning can add to the graph, it can add edges between all pairs of subgoals and classify all subgoals as local subgoals, essentially creating a pairwise distance matrix, which would require a lot of memory to store. Typically, N-Level Subgoal Graphs require new edges to connect h-reachable vertices. However, this is problematic for finding any-angle paths since two vertices that are h-reachable do not necessarily have line-of-sight. For example, in Figure 1, B2 and C4 are h-reachable but do not have line-of-sight. For the any-angle version of N-Level Subgoal Graphs, we allow partitioning to add edges only between vertices that have line-of-sight to ensure that the high-level paths found by searching N-Level Subgoal Graphs are also any-angle paths.

Generalizing beyond Grids

Even though SSGs are specific to grids, the idea of partitioning subgoals can be generalized to undirected graphs (Uras and Koenig 2014). The generalized method is called N-Level Graphs and can be modified to speed-up any-angle path-planning for all undirected graphs embedded in 2D or 3D environments (such as navigation meshes or waypoint graphs), by allowing the partitioning to add edges only between vertices that have line-of-sight.

Similarities to Contraction Hierarchies

N-Level Graphs are closely related to Contraction Hierarchies (CH) (Geisberger et al. 2008; Dibbelt, Strasser, and Wagner 2014), a method that was developed before N-Level Graphs. Both methods create a hierarchy among the vertices of the original graph during a preprocessing phase, which can be used to find shortest paths on the original graph quickly by ignoring vertices that are not relevant for the search. There are two main differences between the two methods: 1) CH allow the addition of edges between any two vertices; and 2) CH choose only a single vertex to remove from the graph at a time (that is, $|S| = 1$), called *contraction*, resulting in a graph where each level contains a single vertex.

Note that CH can also be modified to add edges only between vertices that have line-of-sight, although limiting the edges that can be added to CH can prevent one from contracting all the vertices, resulting in multiple vertices rather than a single vertex at the highest level. The resulting CH can then be used to find any-angle paths in the 2D or 3D environment that the original graph is embedded in.

N-Level Graphs and CH have not been experimentally compared yet and the trade-offs of the different design decisions are therefore, at the moment, unknown. Such a comparison is beyond the scope of this paper and thus subject to future work.

Experimental Results

We compare the runtimes and resulting path lengths of A* and Theta* on grids, SSGs (S_1), Two-Level Subgoal Graphs (S_2) and N-Level Subgoal Graphs (S_N). We also include Block A* (using 5×5 blocks, which is the block size used in (Yap et al. 2011b)) in our comparison, an any-angle path-planning algorithm that partitions the grid into blocks of equal sizes and uses a version of A* that expands blocks rather than vertices. It uses pre-computed shortest any-angle path lengths between all vertices on the perimeter of a block to speed-up block expansions.

The experiments are run on a PC with a dual-core 3.2GHz Intel Xeon CPU and 2GB of RAM. We use game maps, maps with randomly blocked cells, room maps and maze maps in our comparison, available from Nathan Sturtevant's repository.[1] All variants of Subgoal Graphs use the Euclidean distance as edge lengths. All searches use the Euclidean distance as heuristics, except for A* on grids, which uses the Octile distance. We smooth the paths found by each algorithm in a post-processing step, using a simple path smoothing method that provides a good trade-off between runtime and path length ((Daniel et al. 2010), Algorithm 2). In Table 1, we report for each algorithm its average smoothed path length and runtime (including smoothing time) on each type of map. The preprocessing time and memory requirements of Subgoal Graph variants are similar to previous results and therefore not reported.

The results show that the lengths of the paths found by Theta* on Subgoal Graphs are comparable to those found by Block A* and Theta* on grids. The paths found by A* on grids are typically long, whereas the lengths of the paths found by A* on Subgoal Graphs are closer to (but still longer than) those found by Theta* and Block A*, which can be attributed to Subgoal Graphs using the Euclidean distance as edge lengths. The fastest algorithm on game, room and maze maps is A* on Subgoal Graphs, followed by Theta* on Subgoal Graphs, Block A*, A* on grids and Theta* on grids, in that order. Searches on N-Level Subgoal Graphs are typically faster than searches on SSGs ($\sim$3 and 6 times faster on game maps for A* and Theta*, respectively) and much faster than searches on grids ($\sim$100 and 128 times faster on game maps for A* and Theta*, respectively), similar to previous results on Subgoal Graphs. On game maps, Theta* on N-Level Subgoal Graphs is $\sim$20 times faster than Block A*. Theta* searches are faster than A* searches on maps with randomly blocked cells, even though Theta* performs expensive line-of-sight checks with each expansion. This is so because Theta* tends to perform fewer expansions per search than A* (with $\sim 60\%$ fewer expansions on maps with 10% blocked cells) since it typically finds shorter paths.

[1] http://movingai.com/benchmarks/

Conclusions

We have shown how Subgoal Graphs can be used for finding any-angle paths with some simple modifications. Our experiments demonstrate that Subgoal Graphs can be used to find any-angle paths of comparable lengths, but up to two orders of magnitude faster than Theta* and up to an order of magnitude faster than Block A* (using 5×5 blocks).

Acknowledgments

Our research was supported by NSF under grant numbers 1409987 and 1319966. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

References

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

Daniel, K.; Nash, A.; Koenig, S.; and Felner, A. 2010. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research* 39:533–579.

Dibbelt, J.; Strasser, B.; and Wagner, D. 2014. Customizable contraction hierarchies. *arXiv preprint arXiv:1402.0402*.

Ferguson, D., and Stentz, A. 2006. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics* 23(2):79–101.

Geisberger, R.; Sanders, P.; Schultes, D.; and Delling, D. 2008. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Proceedings of the International Conference on Experimental Algorithms*, 319–333.

Harabor, D., and Grastien, A. 2013. An optimal any-angle pathfinding algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling*, 308–311.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Lozano-Pérez, T., and Wesley, M. 1979. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM* 22(10):560–570.

Nash, A., and Koenig, S. 2013. Any-angle path planning. *Artificial Intelligence Magazine* 34(4):85–107.

Nash, A.; Daniel, K.; Koenig, S.; and Felner, A. 2007. Theta*: Any-angle path planning on grids. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1177–1183.

Nash, A.; Koenig, S.; and Tovey, C. 2010. Lazy Theta*: Any-angle path planning and path length analysis in 3D. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 147–154.

	Average Runtime (ms)									Average Path Length								
	A*				Theta*				Block	A*				Theta*				Block
	Grid	S_1	S_2	S_N	Grid	S_1	S_2	S_N	A*	Grid	S_1	S_2	S_N	Grid	S_1	S_2	S_N	A*
bg512	11.52	0.16	0.11	**0.09**	22.69	0.42	0.24	0.16	4.51	238.64	237.66	237.52	237.52	237.27	237.29	237.29	237.36	**237.23**
DAO	18.18	0.88	0.48	**0.26**	36.01	1.95	0.93	0.45	7.21	397.71	393.64	393.38	393.55	393.11	392.94	**392.93**	393.31	393.04
starcraft	75.68	2.49	1.38	**0.70**	190.23	9.09	4.29	1.35	29.75	551.12	546.71	546.12	545.73	545.06	544.93	**544.88**	545.12	**544.88**
random10	19.67	7.93	7.92	7.93	11.22	4.82	**4.81**	**4.81**	6.34	325.10	320.82	320.82	320.82	**319.00**	319.09	319.09	319.09	319.03
random15	18.59	10.85	10.78	10.80	14.41	7.22	**7.19**	7.20	7.33	329.73	325.36	325.35	325.35	323.05	323.13	323.14	323.14	**323.00**
random20	17.95	13.07	12.86	12.88	17.54	9.10	8.99	9.00	**8.31**	330.24	325.85	325.82	325.82	323.13	323.26	323.27	323.27	**323.05**
random25	18.46	14.74	14.16	14.14	21.11	11.59	11.24	11.24	**9.55**	332.19	327.85	327.77	327.76	324.75	324.96	324.98	324.99	**324.68**
random30	18.73	15.30	14.17	13.94	24.84	12.61	11.86	11.74	**10.67**	328.32	324.21	324.05	324.02	320.90	321.19	321.25	321.28	**320.86**
random35	22.58	15.69	13.66	12.94	29.92	14.16	12.59	**12.06**	12.36	330.29	326.45	326.16	326.12	323.12	323.50	323.62	323.72	**323.07**
random40	20.50	10.50	8.32	**7.14**	25.92	10.92	8.73	7.52	10.06	301.68	298.62	298.30	298.30	295.88	296.29	296.53	296.84	**295.86**
room8	44.02	3.67	3.24	**2.97**	55.95	4.29	3.71	3.36	18.03	353.61	348.37	348.58	348.76	348.62	**348.25**	348.43	348.63	348.36
room16	47.53	1.03	0.91	**0.83**	63.52	1.35	1.17	1.06	18.64	358.68	354.37	354.48	354.57	354.61	**354.34**	354.43	354.54	354.48
room32	53.95	0.31	0.28	**0.25**	83.68	0.47	0.40	0.36	21.23	369.46	365.95	366.01	366.05	366.07	**365.93**	365.99	366.04	366.05
room64	65.56	0.11	0.10	**0.09**	131.19	0.19	0.16	0.14	24.80	404.39	401.65	401.69	401.72	401.69	**401.64**	401.68	401.71	401.72
maze4	149.28	3.14	1.85	0.21	207.87	3.66	2.02	**0.18**	60.22	1771.55	1759.78	1759.61	1759.60	**1759.33**	1759.43	1759.53	1759.59	1760.28
maze8	161.86	0.93	0.54	**0.12**	253.11	1.18	0.66	**0.12**	60.05	1798.85	1788.46	1788.14	1787.91	**1787.51**	1787.84	1787.94	1787.89	1788.02
maze16	171.19	0.29	0.18	**0.07**	318.84	0.41	0.23	0.08	60.39	1646.59	1639.57	1639.16	1638.90	**1638.23**	1638.60	1638.65	1638.78	1638.55
maze32	167.89	0.10	0.07	**0.05**	417.73	0.14	0.09	**0.05**	56.35	1198.77	1194.48	1194.09	1193.71	**1193.43**	1193.86	1193.70	1193.61	1193.62
GAME	42.09	1.44	0.80	**0.42**	101.09	4.78	2.26	0.79	16.57	432.31	428.72	428.34	428.23	427.73	427.62	**427.59**	427.85	427.62
ALL	79.18	2.96	2.34	1.90	149.61	4.25	2.76	**1.79**	30.03	811.15	805.76	805.48	805.36	**804.62**	804.69	804.71	804.85	804.74

Table 1: Experimental results.

Nash, A. 2012. *Any-Angle Path Planning*. Ph.D. Dissertation, University of Southern California. http://idm-lab.org/project-o.html.

Sislak, D.; Volf, P.; and Pechoucek, M. 2009a. Accelerated A* path planning. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 1133–1134.

Sislak, D.; Volf, P.; and Pechoucek, M. 2009b. Accelerated A* trajectory planning: Grid-based path planning comparison. In *Proceedings of the Workshop on Planning and Plan Execution for Real-World Systems at the International Conference on Automated Planning and Scheduling*, 74–81.

Thorpe, C. 1984. Path relaxation: Path planning for a mobile robot. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 318–321.

Uras, T., and Koenig, S. 2014. Identifying hierarchies for fast optimal search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 878–884.

Uras, T.; Koenig, S.; and Hernández, C. 2013. Subgoal graphs for optimal pathfinding in eight-neighbor grids. In *Proceedings of the 23rd International Conference on Automated Planning and Scheduling*.

Yap, P.; Burch, N.; Holte, R.; and Schaeffer, J. 2011a. Any-angle path planning for computer games. In *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Yap, P.; Burch, N.; Holte, R.; and Schaeffer, J. 2011b. Block A*: Database-driven search with applications in any-angle path-planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 120–126.