An Empirical Comparison of Any-Angle Path-Planning Algorithms

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

Abstract

We compare five any-angle path-planning algorithms, Theta*, Block A*, Field D*, ANYA, and Any-Angle Subgoal Graphs in terms of solution quality and runtime. Any-angle path-planning is a fairly new research area, and no direct comparison exists between these algorithms. We implement each algorithm from scratch and use similar implementations to provide a fair comparison.

Introduction

For path planning in video games and robotics, one usually discretizes a continuous environment into a graph (such as a grid) and searches this graph with an optimal path-planning algorithm (such as A*) to find a shortest path on the graph. The resulting path can be unrealistic looking and longer than a shortest path on the continuous environment. This path can be smoothed after the search by replacing local parts of the path by straight lines (Botea, Müller, and Schaeffer 2004). However, the smoothed path can still be long since its homotopy often remains unchanged.

Any-angle path-planning algorithms interleave the smoothing and search in order to find paths that are shorter on the continuous environment. They propagate information along the edges of the graph during search, but the movements are no longer constrained to the edges. There have been comparisons between any-angle path-planning algorithms (Nash and Koenig 2013; Daniel et al. 2010; Yap et al. 2011b; 2011a; Uras and Koenig 2015), but these comparisons either lack an experimental evaluation or provide results for only a small number of any-angle path-planning algorithms. Combining the results from different papers into a larger comparison between the various algorithms is tricky because different papers provide results for different implementations of the algorithms and compare them using different experimental setups.

In this paper, we therefore implement and compare five state-of-the-art any-angle path-planning algorithms, namely Theta* (Daniel et al. 2010), Block A* (Yap et al. 2011b; 2011a), Field D* (Ferguson and Stentz 2006), ANYA (Harabor and Grastien 2013), and Any-Angle Subgoal Graphs (Uras and Koenig 2015) in terms of runtime and path length. Of particular note among these algorithms is ANYA, which has no published experimental results. The implementation details can affect the performance of the algorithms, so we implement these algorithms from scratch with similar implementations to provide a fair comparison among them. We make the code of our implementation publicly available. [1]

Preliminaries

In our comparison, we use 2D grids with uniform traversal costs where all blocked cells are known before the search since most of the algorithms in our comparison can only be used in this setting. We call a contiguous set of blocked cells an obstacle. As all the algorithms in our comparison typically do, we place the vertices of a grid graph at the corners of grid cells, rather than their centers. A grid path is a sequence of cell corners where consecutive pairs of cell corners must lie on the same unblocked cell. An any-angle path is a sequence of points $(p_0, \ldots, p_n)$ where p_i and p_{i+1} have line-of-sight for all i, that is, the straight line between them does not pass through any blocked cells. We allow the straight line to pass between two diagonally adjacent blocked cells, but not between two cardinally adjacent blocked cells. p_i is called a *turning point* iff the line segments (p_{i-1}, p_i) and (p_i, p_{i+1}) form an angle $\alpha \neq 180°$. The Euclidean distance between two points is the length of the straight line between them. The Octile distance between two cell corners is the length of a shortest grid path between them assuming that there are no blocked cells on the grid.

Algorithms and Implementation Details

The resulting path length is mostly determined by the algorithms themselves but there may be various implementation tricks with different memory/runtime trade-offs. Because of this and because searching grids typically does not require much memory, we have decided to exclude memory requirements from the comparison and implement the algorithms as efficiently as we can, provided that the memory requirements scale linearly with the grid size (except for Subgoal Graphs, see below). All algorithms use the same implementation of a binary heap as the priority queue. All

[1] Source code and more detailed results can be found at http://idm-lab.org/anyangle

algorithms break ties towards larger g-values and use Euclidean distance as heuristic (unless noted otherwise) since Octile distance is not an admissible heuristic for any-angle path-planning algorithms.

A*: A* is the only algorithm in our comparison that can use the Octile distance as an admissible heuristic. We include two variants, A*-Euc and A*-Oct, that use Euclidean distance and Octile distance as heuristic, respectively.

Theta*: Theta* is a variant of A* that interleaves path smoothing with A*. When expanding a vertex s, it checks for each successor s' of s whether s' and the parent of s have line-of-sight. If so, it sets the parent of s' to the parent of s and assigns the g-value of s' accordingly. We also include Lazy Theta* (L-Theta*), a variant of Theta*, in our comparison. When expanding a vertex s, Lazy Theta* delays visibility checks by assuming that s' is visible from the parent of s. When Lazy Theta* expands s', it first checks if s' is actually visible from its parent. If not, it updates the g-value of s' by using the g-values of the predecessors of s' and proceeds to expand s'.

Block A*: Block A* partitions the grid into equal-sized blocks and uses an A* search that expands blocks, rather than vertices. Before a search begins, Block A* finds shortest paths from the start and goal vertices to the fringe vertices (that is, vertices on the boundary of the block) of the blocks that contain the start and the goal, respectively. When a block is expanded, the g-values of its fringe vertices are updated by querying a pre-computed Local Distance Database (LDDB) that stores the (lengths of) shortest paths between all fringe vertices of a block, for all possible blocks with different configurations of blocked cells. The memory required to store the LDDB is exponential in the size of the blocks. In our experiments, we use 5x5 blocks (that is, there are at most 5 vertices along each side of a block), which is the block size used in (Yap et al. 2011a).

Field A*: Field D* is a variant of D* Lite (Koenig and Likhachev 2005) that considers setting the parent of a vertex to any point on the convex hull of all of its neighboring vertices. The g-value of a non-vertex point p is computed by linearly interpolating the g-values of the two vertices at the ends of the edge that contains p. At the end of the search, a path is extracted by moving between vertex or non-vertex points on grid edges, by greedily following the (interpolated) g-values. Field D* works on grids with non-uniform traversal costs where some of the blocked cells might be unknown in the beginning of a search. In our comparison, we have simplified the g-value interpolation to work on grids with uniform traversal costs and used A* expansions instead of D* Lite expansions to improve runtime. We have also changed the direction of the search from backward (that D* Lite uses) to forward (that all the other algorithms in this comparison use). We call the resulting algorithm Field A*.

ANYA: ANYA is an optimal any-angle path-planning algorithm that works on 2D grids. Its state space consists of tuples of the form (I, r), where I is an interval (a contiguous set of points along a row of the grid), and r (root) is a cell corner so that any point $p \in I$ is visible from r. When a state (I, r) is expanded, ANYA generates successors of the form (I', r'), where I' is in the same row as I or in an adjacent row and $r' \in \{r\} \cup I$. We omit the details of exactly how the successors of a state are generated. Intuitively, ANYA makes sure that, for any state (I, r), r is the most recent turning point on a taut path[2] from the start point s to any point $p \in I$. The search terminates when a state whose interval contains the goal is expanded (goal state). The path is extracted by following the parent pointers from the goal state to the start state. The root points of the encountered states are the turning points on a shortest any-angle path from the start to the goal. ANYA uses a heuristic that estimates for a state (I, r) the shortest distance from r to the goal through any point $p \in I$.

Subgoal Graphs: Simple Subgoal Graphs (SSGs) (Uras, Koenig, and Hernández 2013) are constructed from grids by placing subgoals at the convex corners of obstacles and connecting pairs of subgoals that are *direct-h-reachable* (description omitted). Using SSGs, one can find shortest grid paths by connecting the start and goal to their respective direct-h-reachable subgoals, searching the resulting graph to find a high-level path of subgoals, and following the shortest grid paths between consecutive subgoals on this high-level path. Intuitively, SSGs can be considered as an adaptation of visibility graphs to grids. Any two vertices that are direct-h-reachable also have line-of-sight (Uras and Koenig 2015). This has two implications: 1) SSGs are sparser than visibility graphs and therefore searching them can be significantly faster. 2) The high-level paths found by searching SSGs are any-angle paths (although they are not necessarily optimal). One can use Theta* instead of A* to search SSGs to find shorter any-angle paths.

N-Level Subgoal Graphs (Uras and Koenig 2014) are constructed from SSGs by creating a hierarchy among its vertices. This hierarchy is very similar to Contraction Hierarchies (Geisberger et al. 2008; Dibbelt, Strasser, and Wagner 2014), and can be used to exclude many vertices from the search (while maintaining optimality on grids), resulting in faster searches. During construction, one can add new edges to the graph (which allow searches to ignore even more vertices) to further improve runtime. Adding new edges only between vertices that have line-of-sight guarantees that the high-level paths found by searching N-Level Subgoal Graphs are also any-angle paths.

In our comparison, we use a single any-angle variant of Subgoal Graphs, namely, 2-Level Subgoal Graphs searched with Theta* (Sub-2), although using SSGs or N-Level Subgoal Graphs instead of 2-Level Subgoal Graphs and/or A* instead of Theta* may result in different runtime/path-length trade-offs (Uras and Koenig 2015).

Experimental Results

The experiments are run on a PC with a dual-core 3.2GHz Intel Xeon CPU and 2GB of RAM. We use game maps and maps with randomly blocked cells in our comparison, which are available at Nathan Sturtevant's repository, along with the instances used for each map.[3] For each map type and al-

[2] A taut path is a path that cannot be made shorter without changing its topology.

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

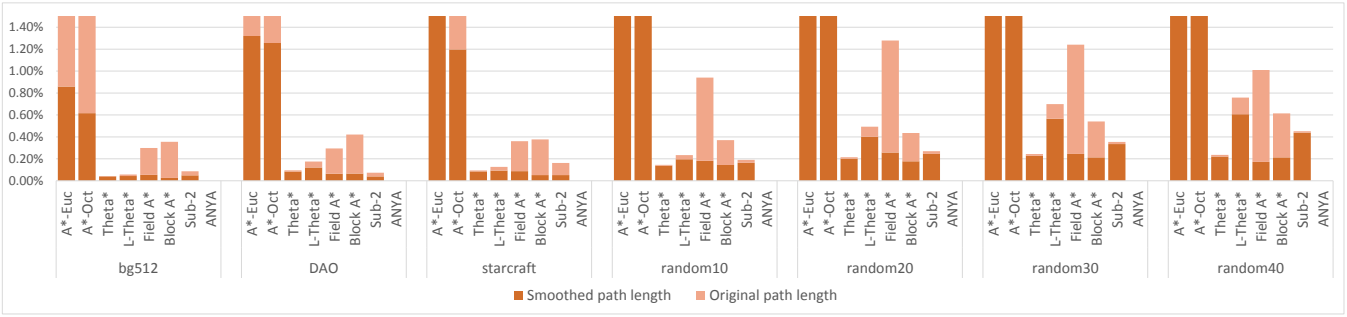


Figure 1: Average path-length suboptimality. Chart is truncated at 1.50% suboptimality. A*-Euc and A*-Oct produce paths that are more than 5% suboptimal on Starcraft maps and between 4-5% suboptimal on other maps, before smoothing.

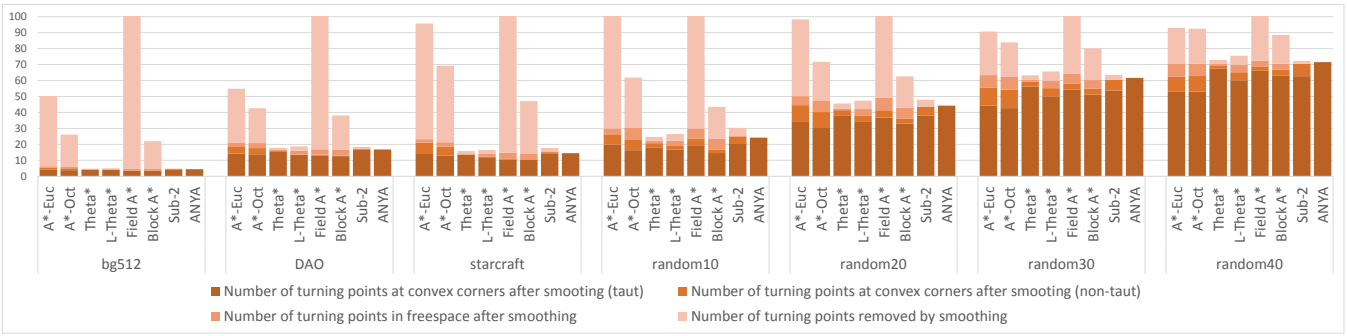
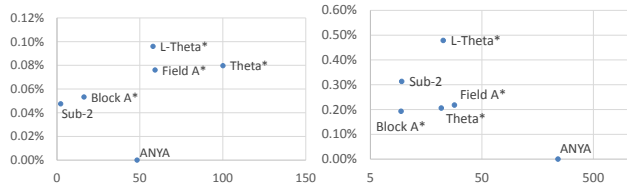


Figure 2: Average number of turning points on paths. Chart is truncated at 100 turning points. Paths found by Field A* have around 450 turning points on Starcraft maps and between 200-300 turning points on other maps, before smoothing.



(a) Game maps. (b) Random maps.

Figure 4: Path-length suboptimality (y-axis) and runtime (x-axis, in ms) after smoothing.

gorithm, we provide results on the suboptimality (Figure 1) and number of turning points[4] (Figure 2) for the paths found by the algorithm and the runtime (Figure 3) of the algorithm. We also smooth the paths found by all algorithms, using the smoothing method described in (Botea, Müller, and Schaeffer 2004), and provide results for the smoothed paths in Figures 1 and 2. Smoothing paths usually takes less than 1% of the runtime of the algorithm (up to 1.63% for Field A* on bg512) and is thus not included in the results.

Game maps: We use three different sets of game maps in our comparison, namely, maps from the game Baldur's Gate II (bg512, resized to 512×512), maps from the game Dragon Age: Origins (DAO, ranging from 22 × 28 to 1260 × 1104) and maps from the game StarCraft (starcraft, ranging from 384 × 384 to 1024 × 1024). On game maps, Sub-2 is the fastest algorithm in our comparison (17.9 times faster than A*-Oct), followed by Block A* (2.52 times faster than A*-Oct). The average solution time for A*-Oct on game maps is 41.04ms. ANYA is the slowest algorithm on DAO but it is 1.87 times faster than Block A* (and 4.67 times faster than A*-Oct) on bg512 and 1.25 times faster than A*-Oct on starcraft. The paths produced by all algorithms except for A*-Euc and A*-Oct are very close to optimal, at most 0.43% longer than the shortest path before smoothing (Block A* on DAO) and at most 0.12% longer after smoothing (L-Theta* on DAO). Figure 4(a) shows that Sub-2 dominates Block A*; ANYA and Block A* dominate Field A*, Theta* and L-Theta*; and Field A* dominates Theta* in terms of the runtime/path-length suboptimality trade-off after smoothing, on results averaged over all game maps.

Random maps: We use 512 ×512 maps with randomly blocked cells, where the percentage of blocked cells vary from 10% to 40%. The variance in the runtimes of the algorithms is lower compared to game maps, with Block A* being the fastest algorithm (2.05 times faster than A*-Oct), followed closely by Sub-2 (2.02 times faster than A*-Oct). The average solution time for A*-Oct on random maps is 19.25ms. The paths produced by all algorithms are less opti-

[4]We distinguish between turning points at convex corners of obstacles that produce taut or non-taut turns and turning points in freespace. Shortest any-angle paths only have taut turns at the convex corners of obstacles.

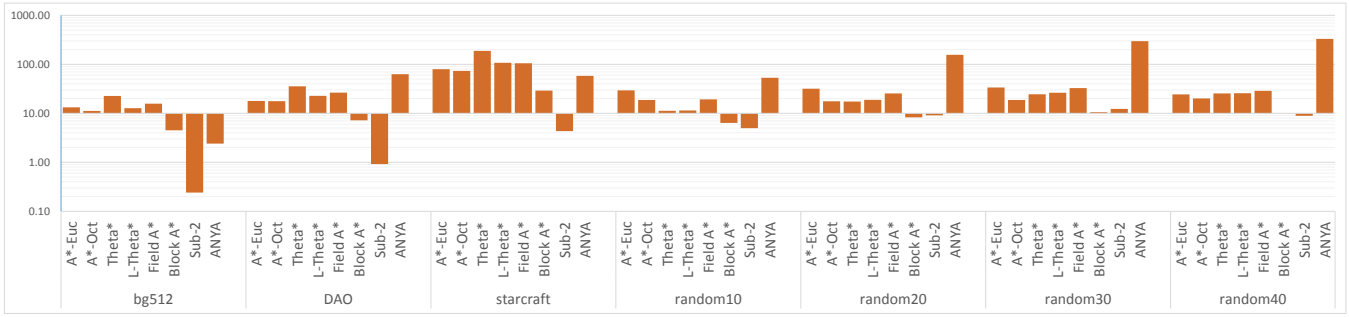


Figure 3: Average runtime (ms).

mal compared to game maps. The paths produced by all algorithms except for A*-Euc and A*-Oct are at most 1.28% longer than the shortest path before smoothing (Field A* on random20), and at most 0.61% longer after smoothing (L-Theta* on random40). Figure 4(b) shows that Block A* dominates Theta* and Sub-2; Theta* dominates L-Theta* and Field A*; and Sub-2 dominates L-Theta* in terms of the runtime/path-length suboptimality trade-off after smoothing, on results averaged over all random maps.

Further observations: ANYA is the algorithm with the highest variance in runtime between different types of maps. This can be explained by ANYA using intervals as states, rather than just vertices. Large, open spaces allow ANYA to explore a map quickly, using long intervals to cover space more quickly than most other algorithms. However, in cluttered environments, such as random maps (and also on Dragon Age: Origins maps, which have many tight corridors, where ANYA is the slowest algorithm), its state space can blow up (since similar intervals can be paired with different root points), requiring many expansions before reaching the goal.

A*-Oct benefits the most from smoothing on game maps, with an average of 3.6% reduction in path-length, whereas A*-Euc benefits the most from smoothing on random maps, with an average of 2.27% reduction in path-length. Since Theta* performs a similar smoothing during its search, it does not benefit much from the smoothing after the search, with an average of 0.01% reduction in path-length on both game and random maps. The paths produced by Field A* have many small turns because of its interpolation-based approach. Smoothing removes most of these turns, but the resulting paths still have the most number of turns among the algorithms (except for A*-Euc and A*-Oct). Paths produced by Sub-2 have no turns in freespace because Sub-2 uses convex corners of obstacles as vertices.

It should be noted that even if some algorithms are dominated by others in terms of the runtime/path-length suboptimality trade-off in our experiments, they have other qualities that makes them suitable in different contexts. Theta* was the simplest algorithm to implement (besides A*) and can be applied to other graphs embedded in 2D or 3D environments, such as navigation meshes. Field D* is the only algorithm that addresses non-uniform cost grids and can easily be used for incremental path-planning. Even though Sub-2 is non-dominated on game maps, it requires time for preprocessing (up to 35 seconds for starcraft).

Comparison with previous results: Our results mostly match the results in (Yap et al. 2011b; 2011a), especially on game maps that use the same scenarios. If we assume that their implementation of A*-Euc is similar to ours, then our Block A* and Theta* implementations are 1.17 and 2.84 times faster than theirs, respectively, on Dragon Age: Origins maps. The substantial difference in the Theta* implementations might be due to a more efficient implementation of line-of-sight checks on our part. Our results confirm their result that Block A* is faster than Theta* on both game and random maps. Contrary to their results, in our experiments, smoothed Block A* paths are slightly shorter than smoothed Theta* paths (in both our and their experiments, the difference is very small). This is not necessarily a contradiction, however, because they do not provide averaged results over game maps, and the random maps used in their experiment are significantly different from the ones that we use. In our maps, after a given percentage of cells are randomly blocked, only the cells in the largest connected component are left unblocked in the map, increasing the actual percentage of blocked cells (Sturtevant 2012).[5]

Compared to (Daniel et al. 2010), if we assume that their implementation of A*-Oct is similar to ours, then our Theta* implementation is 1.32 times slower that theirs, but our Field A* implementation is 1.78 times faster than theirs on maps from Baldur's Gate II.[6] Our results confirm their result that Theta* produces shorter paths than Field D*, A*, and A* with smoothing.

Acknowledgments

We thank Alex Nash and Robert Holte for their helpful insights and Romain Jacob for sharing his implementation of ANYA in MATLAB. 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.

[5]The connected components are identified by using the centers of grid cells as the vertices rather than their corners, and diagonal movement between two diagonally adjacent blocked cells is not allowed.

[6]The maps in their experiments are scaled to 500 $\times$ 500 and they use a different set of instances than ours.

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.

Koenig, S., and Likhachev, M. 2005. Fast replanning for navigation in unknown terrain. *Transactions on Robotics* 21(3):354–363.

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

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

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., and Koenig, S. 2015. Speeding-up any-angle path-planning on grids. In *Proceedings of the International Conference on Automated Planning and Scheduling*.

Uras, T.; Koenig, S.; and Hernández, C. 2013. Subgoal graphs for optimal pathfinding in eight-neighbor grids. In *Proceedings of the 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.