Any-Angle Path Planning for Computer Games

Peter Yap and **Neil Burch** and **Robert C. Holte** and **Jonathan Schaeffer**
Computing Science Department
University of Alberta
Edmonton, Alberta, Canada T6G 2E8
blueberrypete@gmail.com, {nburch, rholte, jonathan}@ualberta.ca

Abstract

Path planning is a critical part of modern computer games; rare is the game where nothing moves and path planning is not necessary. A* is the workhorse for most path planning applications. Block A* is a state-of-the-art algorithm that is always faster than A* in experiments using game maps. Unlike other methods that improve upon A*'s performance, Block A* does not require knowledge of the map before the search and its paths are never longer than A* paths. In our experiments, Block A* is ideal for games with randomly generated maps, large maps, or games with a highly dynamic multi-agent environment. In the domain of grid-based any-angle path planning, we show that Block A* is an order of magnitude faster than the previous best any-angle path planning algorithm, Theta*. We empirically show our results using maps from DRAGON AGE: ORIGINS and STARCRAFT. Finally, we introduce "populated game maps" as a new testbed that is a better approximation of real game conditions than the standard testbeds of this field. The main contributions of this paper are a more rigorous set of experiments for Block A* and introduction of a new testbed (populated game maps).

1 Introduction and Overview

Pathfinding is an important topic in numerous domains, including computer games. Path planning in games is particularly challenging for a number of reasons. First, the paths must be calculated in milliseconds due to real-time constraints. Second, the domain is dynamically modified; for example, obstacles like doors or walls may be built or removed in real time. Third, games may have numerous competing players, each controlling an army of mobile agents. For example, RTS (real-time-strategy) games like STARCRAFT may have up to 8 players, each controlling up to 200 units, for a worst case total of 1600 pathfinding agents.

Academia and the games industry are beginning to collaborate, resulting in artificial intelligence (AI) research being incorporated into computer games (e.g., BioWare's DRAGON AGE engine uses University of Alberta research (Sturtevant and Geisberger 2010)). Recently, Yap *et al.* developed a new algorithm called Block A* (a generalization of A)* that returns the same optimal path as A* for traditional 4-way and 8-way grid searches (Yap et al. 2011). Furthermore, in the domain of any-angle pathfinding (paths not constrained by 45° angles), it is an order of magnitude faster than the previously best any-angle path planning algorithm, Theta* (Daniel et al. 2010). Theta* works like A*, but does a line-of-sight check at every node expansion. Field D* (Ferguson and Stentz 2006) is another any-angle algorithm. Field D* uses interpolation during each vertex expansion. Field D* paths can be worse than A* paths, while both Theta* paths and Block A* paths can never be worse than A* paths.

RTS games like the STARCRAFT and DAWN OF WAR series use a grid for its flexibility in dealing with dynamic terrain and multiple path planning agents (Jurney 2010). Other genres, like BioWare's DRAGON AGE RPG series, also use grids. Grids allow one to quickly change the cost of entering a region, for example, a door closing, or a minefield; this is harder to do in a continuous space representation like visibility graphs, nav-meshes or triangulation (Demyen and Buro 2006). A problem with grids is that it often leads to jagged paths due to the 45° turning constraints on grid movements. These paths are often "smoothed" in post-processing to make the paths shorter and more realistic (Rabin 2000). However paths using this post-processing may not be ground-truth optimal.

An alternative is to use an any-angle algorithm like Theta* that searches for a smoother, more human-like path *during* the search, as opposed to *after* the search. Alternatively, the recently published Block A* provides paths of roughly the same length as Theta* but is always faster.

Practically every paper in this field uses two sets of testbeds to test their algorithm: a grid with randomly placed obstacles and an unpopulated game map. The problem with the random obstacle testbed is that the randomness does not properly construct maps that represent real game maps. The problem with an unpopulated game map, is that it is devoid of the moving agents or dynamically placed objects that are placed in the game during run-time. We believe we can better approximate real game conditions by combining both testbeds into one, what we call a "populated game map".

2 Introduction to Any-Angle Path Planning

Figure 1 illustrates the difference between any-angle and normal A* pathfinding. Here we aim to find the shortest path from $B1$ to $E6$. The shaded square bounded by C4 and

D5 is an impassable obstacle, however we can traverse on its edges. The ground-truth shortest path is $B1 - D4 - E6$ (dashed line), with the optimal length of 5.84. In contrast, A* will find a zig-zag-like path $B1 - C2 - C3 - D4 - D5 - E6$ (black circles; cost is 6.24). This is because grid-based A* is constrained to heading changes that are $45°$, resulting in many directions changes (twice as many, in this example).

A* will always find the optimal path, relative to the grid that it searches on. However, the zig-zag path found by A* is not ground-truth optimal, thus post processing is applied to smooth and shorten this path.

In our experiments, we used an efficient post-processing smoothing where we iteratively check the inflection points $x_1, ..., x_n$ found by A*. This path smoothing algorithm removes the inflection point x_i if its neighbors x_{i-1} and x_{i+1} have line of sight (LOS) of each other. This simple LOS smoothing algorithm will result in a ground-truth optimal path in this example, although this is not guaranteed in every situation.

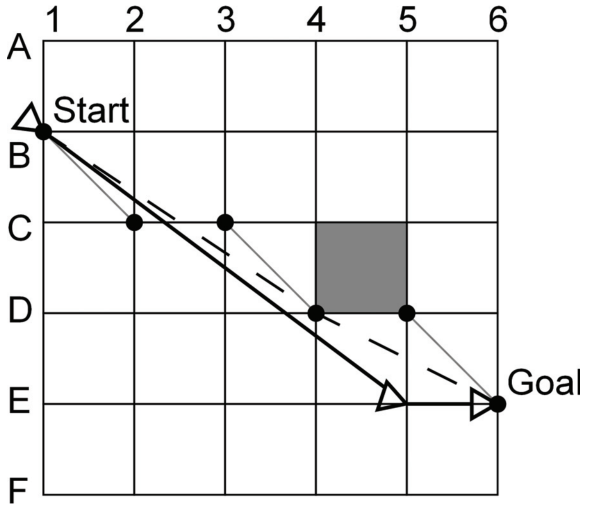

Figure 1: Any-angle search results

Traditional A* on a grid searches with $45°$ constraints; the jagged path found is only smoothed as an afterthought during post-processing. A better alternative for smoother paths is to use Theta*, which considers paths of any-angles (not just $45°$) during the search. While a path-smoothed A* algorithm does a LOS check *after* the search, Theta* does a LOS check *during* the search. Theta* does a LOS check for every child of a vertex expansion, but otherwise performs the same as A*; unlike A*, the parent of a vertex in Theta* is not confined to its neighbor. Theta* finds paths of any-angle while searching on a grid. Normally for A*, when the node s' is expanded, its child s'' has a pointer back to its parent s'. In Theta*, for each expansion of vertex s', a LOS check is made for every child s'' of s'. If a LOS check from the child s'' to s (parent(s')) is successful, then this implies that an unobstructed line can be drawn from s'' to s and the parent of s'' is now set to be s.

In Figure 1, let $E5$ be s' whose parent $B1$ is s. When $E5$ is expanded, it will check if its child s'', $E6$, has a straight line to $B1$. If it does, then $E6$'s parent is set to $B1$. In this example, there is no direct path from $E6$ to $B1$ and thus $E6$'s parent is set to $E5$. The path found by Theta* is $B1 - E5 - D6$ (open triangles; cost is 6). Note that Theta* is actually worse than a post-processed A* path (dashed line; cost is 5.84) in this example. This figure is also described in detail in (Daniel et al. 2010).

Finally, in our quest to find smoother and shorter paths, we can use Block A* for any-angle path planning. Both Theta* and Block A* search for any-angle paths during the search. While Theta* conducts LOS checks, Block A* uses a small pre-computed database that has LOS information about the local area. We call this database containing the local LOS information the LDDB (Local Distance Database). In Figure 1, Block A* finds the optimal path $B1 - D4 - E6$ (dashed line) with a cost of 5.84 instantaneously by reading from its database. However, like A* and Theta*, the ground-truth optimal path is not guaranteed. Block A* uses locally optimal (block-wise) information (from the LDDB) to help improve its path quality, but it lacks the global information needed for a ground-truth optimal answer. The details of how Block A* works is not provided here (see (Yap et al. 2011) for full details), but we explain it conceptually in the next section.

We should note that traditional grid searches on the tiles (cells) of a grid, while the any-angle literature searches on the vertices of the grid. As an example, if our grid resembled a face of a Rubik's Cube, it would be called a 3x3 (cell) grid in traditional search literature, but it would be a 4x4 (vertex) grid in any-angle literature. As such, any-angle paths can traverse the outer edges of a tile obstacle. The Block A* algorithm is unaffected by this, so for brevity we use the term "node" to describe both cells and vertices, as Block A* simply considers both nodes in a graph.

3 Introduction to Block A*

Block A* is a generalization of A*. Instead of searching one *node* per iteration like A*, Block A* searches one *block* per iteration, where a block is a square region of nodes. The performance gain of Block A* derives from it searching in bulk. Consider a shopping analogy of purchasing 100 beer kegs for a party. Instead of driving to the brewery to buy one keg per trip (A*), 100 trips, we drive to the brewery to buy 25 kegs per trip (Block A*), 4 trips. Obviously, the more we buy per trip, then the less trips are needed. For non-trivial cases, where the search space is large (we need to buy a lot of beer!), Block A* will out-perform A*. We will later show this in our experiments with real game maps.

There are two main components to Block A*. First, A* is generalized so that it can handle blocks of nodes. Second, a database is built, called the Local Distance Database (LDDB), that enables the efficient processing of these blocks during the search. A particular set of obstacle combinations in a block is called a "block pattern". The LDDB will store every block pattern, so that it can handle every possible obstacle combinations possible in a block. Each LDDB entry stores the all-pairs-shortest-path results for the perimeter nodes of the block pattern. The all-pairs-shortest-path results can be quickly computed using Dijkstra's algorithm as a one-time cost, and is not part of the runtime cost. Computing the LDDB takes less than one second.

For example in Figure 1, a 5x5 cell LDDB (or equivalently, a 6x6 vertex LDDB) would have stored the all-pairs-shortest-path between every vertex on the boundary. Indeed, both the vertices $B1$ and $E6$ are boundary vertices, and we can simply look up in the LDDB to find the optimal path between these two boundary vertices quickly after the block pattern is identified.

Figure 2 contrasts the area of explored by A* with the area of Block A*. The shaded area is the region searched by A*. The blocks encapsulate the area examined by Block A*. Block A* actually examines more nodes than A*. For every node x searched by A*, Block A* will examine the entire block (of nodes) that x belongs to. Note that when Block A* $examines$ a block, it does not $search$ nor $expand$ every node in the block in the traditional sense. For one, the interior block nodes are ignored. It is somewhat counterintuitive that by examining more nodes, one actually goes faster, yet this is the case for Block A* in our experiments. Firstly, the heap of Block A* is smaller. Instead of dealing with a heap of n nodes, Block A*'s heap deals with $\frac{n}{B \times B}$ blocks of nodes, where B x B is the size of the square block. Secondly, we use a LDDB database to quickly process these blocks. Finally, we only directly deal with the perimeter block nodes; the interior block nodes are ignored.

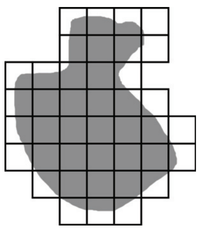

Figure 2: The shaded area is the region searched by A*. The blocks encapsulate the area examined by Block A*

A* and Block A* mainly differ in how each expand a node (A*) or a block (Block A*). We only summarize the main ideas of Block A* here using the simplest example. The full details can be found in (Yap et al. 2011). Consider Figure 3, where we expand a block of 2x2 unobstructed tiles. For simplicity, assume 4-way vertex moves with a zero heuristic.

1) PRE-EXPANSION: This block's parent is from the south. The **ingress** vertices (circled) are the vertices from the parent block that have a finite g-value. The **egress** vertices are all the vertices on the block boundary.

2) LDDB LOADED: The LDDB entry for this block pattern is retrieved, $LDDB(y, x)$ returns the length of the shortest path between any ingress y (circled) and egress x for a given block obstacle pattern. Since we have three ingress vertices, we use the LDDB for each. The circled value is the g-value of that ingress found during search and does not come from the LDDB. The top block is the LDDB entry for the leftmost ingress (g=3). The middle block is the LDDB entry for the middle ingress (g=4). The bottom block is the LDDB entry for the rightmost ingress (g=5).

3) LDDB APPLIED: We add $y.g$ and $LDDB(y, x)$ to find the g-value of the shortest path from $start$ to x via y.

4) POST-EXPANSION: For each egress x, its new g-value is the minimum of its old g-value and the smallest g-value from all paths via these ingresses (circled). $x.g' = min_{y \in Y}(x.g, y.g + LDDB(y, x))$. For example, the shortest path to the top-left vertex is $min(5, 7, 9) = 5$. Finally, we place the four block neighbors of the expanded block into the open list using the $min_{updated\ s'}(s'.g + s'.h)$ as its heap value, where s' are the boundary vertices shared by the neighboring block and the expanded block. With a zero heuristic, the east block neighbor would be added with a heap value of $min(5, 6, 7) = 5$.

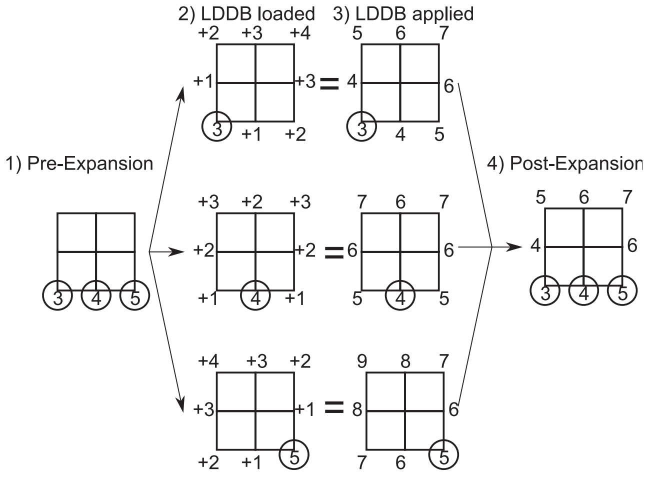

Figure 3: Expanding a block using a 3x3 vertex LDDB of 4-way movement.

Block A* is database-driven in that it is simple to switch from finding 4-way paths to any-angle paths. To demonstrate this, consider the difference between expanding the same block in Figure 3 using a 4-way LDDB and Figure 4 using an any-angle LDDB. Note that the LDDB is any-angle only in the paths within the block. It is only truly any-angle for sufficiently large blocks. In Figure 4, the execution of Block A* is the same, however, the resultant g-values of the expanded block differs. In this simple case, the leftmost ingress (g=3) is dominant over the other ingresses.

Not only is Block A* faster than A*, it also finds shorter paths. Intuitively, Block A* glues together path segments that are any-angle optimal piece-wise (block-wise). See Figure 5. The larger the $B \times B$ block used in the LDDB, the shorter the path in general. One can think of A* as a special case of Block A* that only expands one node (one tile, or a 2x2 vertex block) at a time. At top is the path found by A* using the Euclidean heuristic; the path has many turns (10). The ground truth optimal path is the dashed line. At bottom is the Block A* path found by using a LDDB of 3x3 vertex blocks (equivalently a 2x2 tile block); it has less turns (2) than the A* path. In this trivial example with no obstacles,

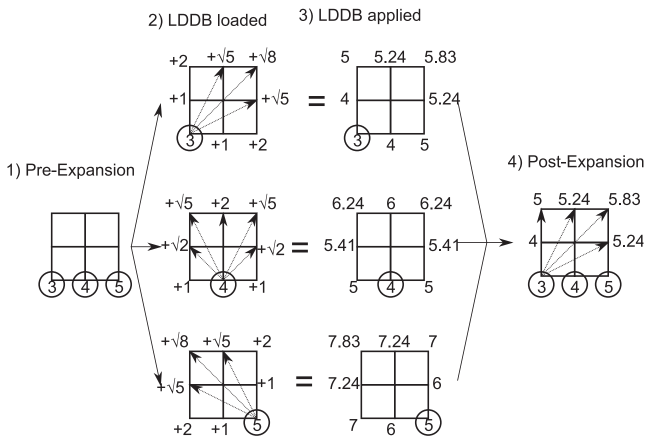

Figure 4: Expanding a block using a 3x3 vertex LDDB of any-angle movement.

smoothing these paths in the post-processing will result in the ground-truth optimal path. However, this is generally not possible in practise because of obstacles.

While not described here in detail, Block A* improves upon the performance of A* by making the computation of g-values during the A* search more efficient. Most search enhancements have gone in the opposite direction by improving the performance of h-values, as noted in (Harabor and Botea 2010). As such, the performance gains of Block A* are orthogonal to other A* speed enhancements such as hierarchical planning or a weighted heuristic.

It should be noted that in traditional 4-way and 8-way pathfinding, Block A* and A* return the same optimal path and is 2-4 times faster in our experiments (not shown here).

4 Experimental Results

Block A* is evaluated using three classes of experiments: 1) grids with randomly placed obstacles; 2) commercial game maps; and 3) commercial game maps populated with random obstacles. Comparisons are made to A*, Theta*, and visibility graphs (v-graphs). Neither A* nor Theta* require any pre-computation. While Block A* requires preprocessing to build the LDDB, it is less than one second (even for the largest 5x5 LDDB used in this paper), and once computed it will work for every map and for every size; the amortized cost is negligible. In contrast, the time required to generate the v-graphs was quite extensive (2 weeks of computing time were needed using Xeon 3.0 GHz computers). Generating these v-graphs were $O(v^3)$ where v is the number of all vertices (obstacles' corners) in the map. The time (in hours) to generate these v-graphs are shown in brackets under the "Data Set" column in the subsequent tables.

It should be noted that the original game maps were provided as grids, not as v-graphs. In reality, most games maps are either exclusively in grid representation or in a continuous representation (e.g. v-graphs). For comparison purposes, we tried our best to do a fair conversion. There are many techniques that trade off between the construction time and the search time of v-graphs (de Berg, van Kreveld, and

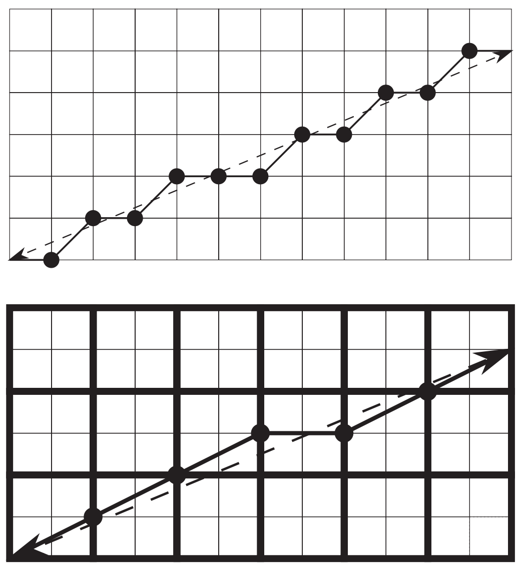

Figure 5: Top: A* path (special case of 2x2 vertex Block A*). Bottom: 3x3 vertex Block A* path. Dashed line is the ground-truth optimal path.

Schwarzkopf 2000). In addition, there are many techniques that forsake v-graph optimality for speeding up the search time or the construction time; for this paper, v-graph always returns the optimal path. While there may never be a fair comparison between discrete and continuous path planning techniques, we have tried our best and warn the reader of our deficiencies.

In our experiments, we also compared using different LDDB block sizes: 3x3, 4x4, and 5x5. A 2x2 (vertex) block is equivalent to a single tile cell with its 4 exterior vertices. Therefore, a 2x2 Block A* search is simply A* searching in 8 directions. The results for 2x2 Block A* are not provided. A simple 3x3 LDDB is only a few bytes, a 5x5 is 60MB, while a 6x6 LDDB (not used here) uses over 2GB.

The average number of nodes expanded during the search is typically provided in these types of experiments shown here. However, they are omitted here as they are *not* accurate indicators of algorithmic performance because the cost to expand a node differs significantly between algorithms. In particular, node counts artificially inflates Block A*'s performance. For A*, each node expansion is a relatively constant cost. For Theta*, each node expansion includes an additional LOS check, whose cost differs significantly depending on how open or cluttered the map is. Theta*'s LOS check is in addition to the normal A* cost of expanding a node, thus Theta* is always slower than A* per expansion. Finally, Block A* expands blocks, not nodes; thus its number of expansions is much fewer.

In Tables 1, 2, and 3, the path length is averaged over 500 searches and is listed in the "Distance" column. The length of the path after smoothing is done is shown in brackets. Theta* did not significantly benefit from this so its smoothing result was omitted, while v-graphs already have the perfect any-angle paths so its smoothing result are also omitted. The time for path smoothing was at most 1% of the total search time.

The average run-time is given in the "Time" column. In addition, v-graphs require a pre-computation to generate the graph to search on; the total pre-computation time (hours) is given in brackets under the "Data Set" column. While LDDBs need to be pre-computed too, they take less than one second, and they work for every map of every size. Thus its pre-computation time is not shown. The "Time" column *excludes* the times for pre-computations.

4.1 Random Maps

We start by comparing the algorithms using a 600x600 grid (typical for modern games) filled with randomly placed obstacles, the probability of a cell being an obstacle ranges from 0% to 50%. A*, Theta*, visibility graphs (v-graphs), and Block A* (with different LDDB block sizes) are compared. These results are in Table 1, with each entry representing the average over 500 randomly generated maps, each with its own random start and goal node.

For the trivial case of maps with zero obstacles, v-graphs clearly win, as the shortest path is simply a straight line between the start and the goal. This is also the only case where any algorithm is faster than Block A*. Theta* also performs poorly compared to A* here because the cost for LOS checks are enormous if the LOS is very long (caused by the large open space).

For non-trivial searches, maps with any positive percentage of obstacles, Block A* is clearly the fastest algorithm. The larger the LDDB used by Block A*, the greater the speed gain. The greatest speed improvement per memory usage is from using a simple 3x3 LDDB. While there are diminishing returns as the block size increases, Block A* still maintain an overall speed improvement. Furthermore, the paths found are shorter than A*.

The performance of Theta* is mixed; while it is faster compared to v-graphs and returns paths that are very close to the ground-truth optimal, Theta* is generally slower than A*. In comparison, Block A* is always faster than A* and gives paths that are within 0.1% of Theta* in length. For games like RTS games, finding a path quickly is more important than finding a smooth path. By this criteria, both Theta* and v-graphs are poor alternatives to A*. In comparison, not only is Block A* at least twice as fast as A*, it returns a shorter path at no extra charge.

4.2 Unpopulated Game Maps

Maps with random obstacles are obviously not indicative of actual game performance. In this section, we use five STARCRAFT maps used in the AIIDE 2010 STARCRAFT Competition. These maps were chosen due to their popularity in human competitions, and partly because they are strategically challenging for the game AI. In addition to these five maps, we also included the largest map, Korcari Wilds, from DRAGON AGE: ORIGINS. These results are displayed in Table 2.

The STARCRAFT maps are strategically interesting with large open spaces with choke points. The large open spaces allow for bases to be built there, and facilitate large-scale battles with hundreds of participants. Additionally, the choke points are narrow corridors between large open spaces to facilitate conflict and control of the map. In terms of pathfinding, large open spaces means more states for A* to explore. Moreover, choke points affect the efficacy of our Euclidean heuristic. Consequently, the pathfinding results for game maps in Table 2 differ from the random maps in Table 1.

With game maps, Theta* performs worse with game maps than with random maps; in the map Python, Theta* is over 11 times slower than A*. Block A* is still at least twice as fast as A*. The effect of different LDDB block sizes for game maps is similar to random maps, in that the larger the block size the faster the search is, and thus the 3x3 and 4x4 entries are not shown. Finally, we see that v-graphs do better for real game maps than for random maps. This is mostly because the number of v-graph vertices is lower, and the pre-computation time of v-graph is in minutes instead of days.

Data Set	Algorithm	Distance	Time
Random 0% (0h)	A*	336.4 (319.6)	0.0090s
	Theta*	319.6	0.0117s
	V-graph	319.6	0s
	3x3 Block A*	323.9 (319.6)	0.0037s
	4x4 Block A*	321.8 (319.6)	0.0025s
	5x5 Block A*	320.8 (319.6)	0.0017s
Random 10% (41h)	A*	335.9 (325.1)	0.0091s
	Theta*	319.5	0.0068s
	V-graph	319.3	0.0099s
	3x3 Block A*	323.5 (321.4)	0.0045s
	4x4 Block A*	321.5 (320.3)	0.0030s
	5x5 Block A*	320.6 (319.9)	0.0023s
Random 20% (62h)	A*	330.9 (321.1)	0.0089s
	Theta*	315.3	0.0079s
	V-graph	315.1	0.0109s
	3x3 Block A*	319.2 (317.1)	0.0051s
	4x4 Block A*	317.4 (316.2)	0.0034s
	5x5 Block A*	316.6 (315.9)	0.0029s
Random 30% (62h)	A*	334.9 (326.1)	0.0093s
	Theta*	320.1	0.0101s
	V-graph	319.9	0.0132s
	3x3 Block A*	324.0 (321.9)	0.0061s
	4x4 Block A*	322.4 (321.1)	0.0042s
	5x5 Block A*	321.6 (320.8)	0.0039s
Random 40% (48h)	A*	341.0 (332.4)	0.0107s
	Theta*	327.5	0.0146s
	V-graph	327.2	0.0108s
	3x3 Block A*	331.6 (329.3)	0.0081s
	4x4 Block A*	329.9 (328.4)	0.0054s
	5x5 Block A*	329.2 (328.2)	0.0052s
Random 50% (30h)	A*	364.0 (354.9)	0.0170s
	Theta*	350.9	0.0282s
	V-graph	350.7	0.0123s
	3x3 Block A*	355.3 (352.7)	0.0142s
	4x4 Block A*	353.6 (351.9)	0.0092s
	5x5 Block A*	352.8 (351.6)	0.0090s

Table 1: Algorithm performance (random obstacles)

Data Set	Algorithm	Distance	Time
Tau Cross 0% (0.1h)	A*	334.4 (319.7)	0.0120s
	Theta*	318.0	0.0841s
	V-graph	318.0	0.0016s
	5x5 Block A*	319.3 (318.1)	0.0048s
Andromeda 0% (0.2h)	A*	328.2 (315.5)	0.0154s
	Theta*	313.3	0.1316s
	V-graph	313.3	0.0021s
	5x5 Block A*	314.4 (313.5)	0.0062s
Destination 0% (0.1h)	A*	299.4 (287.0)	0.0084s
	Theta*	285.2	0.0514s
	V-graph	285.2	0.0011s
	5x5 Block A*	286.4 (285.4)	0.0036s
Python 0% (0.1h)	A*	245.5 (234.0)	0.0155s
	Theta*	232.8	0.1628s
	V-graph	232.8	0.0014s
	5x5 Block A*	233.7 (232.8)	0.0056s
Heartbreak Ridge 0% (0.1h)	A*	288.9 (277.1)	0.0084s
	Theta*	275.5	0.0592s
	V-graph	275.5	0.0010s
	5x5 Block A*	276.5 (275.6)	0.0036s
Korcari Wilds 0% (0.3h)	A*	422.7 (408.8)	0.0106s
	Theta*	403.5	0.0556s
	V-graph	403.5	0.0021s
	5x5 Block A*	405.0 (403.7)	0.0050s

Table 2: Algorithm performance (unpopulated game maps)

4.3 Populated Game Maps

Here we present our final experiment, where we combined the random obstacles from Table 1 with the game maps from Table 2. The results are shown in Table 3. For each open tile in the game map, we placed an obstacle on that tile with 10% probability.

Game maps by themselves are devoid of the numerous obstacles that often inhabit the game space during the course of the game. For example, there may be numerous buildings and objects that clutter the game map. These extra obstacles are not represented by the plain game map because they are placed in real-time by the players and are not known a-priori. The actual number of obstacles and their placements depends on numerous factors. In a multi-player RTS game like STARCRAFT, there may be hundreds of mobile agents and buildings in the game space. In a single-player RPG game like DRAGON AGE: ORIGINS, the maximum number of obstacles is less. In a RTS game, the number of obstacles grows as the game progresses and peaks as the armies collide in the “late game”, while in a RPG game the number of foes is fairly low and constant. Due to time and resource constraints, we simply populated obstacles in the playable game space randomly. Although not ideal, this gives us a rough idea of the relative *worst case* performance of the competing algorithms in a cluttered game map. The average case performance is likely closer to 0% obstacles than 10% obstacles, although this depends on many factors.

In populated game maps, both Theta* and v-graphs are slower than A*. Theta*’s performance here is better than Theta* in an unpopulated game maps, this is due to the shorter and cheaper LOS checks caused by obstacles. While the performance of v-graphs is greatly affected by the addition of random obstacles on game maps, A*’s and Block A*’s performance are robust in this domain. Similar to the results from Table 2 and Table 3, Block A* is always at least twice as fast as A*, while finding shorter paths.

Data Set	Algorithm	Distance	Time
Tau Cross 10% (12h)	A*	326.3 (318.6)	0.0126s
	Theta*	310.8	0.0280s
	V-graph	310.6	0.0163s
	5x5 Block A*	311.9 (311.2)	0.0055s
Andromeda 10% (14h)	A*	327.8 (319.7)	0.0155s
	Theta*	313.2	0.0378s
	V-graph	313.0	0.0212s
	5x5 Block A*	314.2 (313.6)	0.0072s
Destination 10% (7h)	A*	302.9 (295.8)	0.0087s
	Theta*	288.9	0.0200s
	V-graph	288.8	0.0111s
	5x5 Block A*	290.0 (289.4)	0.0043s
Python 10% (17h)	A*	247.2 (240.1)	0.0152s
	Theta*	234.7	0.0354s
	V-graph	234.6	0.0212s
	5x5 Block A*	235.6 (235.1)	0.0065s
Heartbreak Ridge 10% (7h)	A*	287.2 (280.4)	0.0085s
	Theta*	274.1	0.0205s
	V-graph	273.9	0.0118s
	5x5 Block A*	275.0 (274.4)	0.0043s
Korcari Wilds 10% (6h)	A*	417.0 (407.8)	0.0105s
	Theta*	398.6	0.0258s
	V-graph	398.4	0.0122s
	5x5 Block A*	400.0 (399.1)	0.0056s

Table 3: Algorithm performance (populated game maps)

4.4 Discussion of the Different Testbeds

For all three testbeds, Block A* is always at least twice as fast as A*, and often 5-30 times faster than the previously best any-angle algorithm Theta*. Block A* always finds an any-angle path no worse than A*, and is within 0.1% of Theta* paths. The only algorithm faster than Block A* are v-graphs, but only if the preprocessing time is omitted and if the map is unpopulated. For highly dynamic games, the cost to maintain the v-graphs in real-time makes it inferior to Block A*.

V-graphs suffers as the number of random obstacles increase. Additional experiments (not shown here) shows that v-graphs are roughly the same speed as Block A* using populated game maps with just 1% obstacles. It can be a difficult endeavor to optimize v-graph performance if the number of in-game obstacles is high and the map rapidly changes. In contrast, grids are robust with respect to these concerns and are simple to implement, but have slower search times.

It is interesting to note the dramatic performance degradation of v-graphs when tested on populated game maps instead of unpopulated game maps. Most academic papers only tested on vacant game maps without obstacles for their evaluations. We believe that using a testbed of populated maps better approximates actual in-game performance, and uncovers algorithmic flaws that are otherwise undetectable in the standard testbeds.

5 Advantages and Disadvantages of Block A*

Besides finding smoother and shorter any-angle paths faster than A*, we further identify four advantages of Block A*.

1. Generality: *A priori* map knowledge is not necessary.
2. Scalability: It can be used to solve maps of any size.
3. Optimality: It will never find a path worse than A*'s path.
4. Memory: One LDDB will solve any map.

Games like STARCRAFT and DRAGON AGE: ORIGINS have maps that are known before the game is shipped. There are numerous search enhancements that can pre-process these maps to derive run-time speed gains (Sturtevant and Geisberger 2010). They essentially trade domain knowledge for speed. Upcoming dungeon crawler games like DIABLO III or TORCHLIGHT II have randomly generated maps to maximize replay-ability. Randomly generated maps limit the use of pre-processed speed gains. Block A* works well here as no domain knowledge is required.

DRAGON AGE: ORIGINS used 157 maps. A hierarchy or look-up table may need to be built (several hours) for each map to use abstraction techniques. In contrast, only one LDDB (1 second build time) is needed for Block A* to solve all.

Many search abstraction techniques do not guarantee optimality, trading off accuracy for speed. In contrast, Block A* always returns a path no worse than A*.

Many search enhancements generally require more memory to store their hierarchies or look-up tables that speed up A*. They trade off memory for speed. While Block A* also requires memory to store the LDDB, it is possible that Block A* will actually use *less* memory than A* while still finding a path no longer than A*. This is because the grid that A* searches on is node based, while Block A* searches on a block map. As an example, a 600x600 node map can be reduced to a 200x200 block map if Block A* uses a 3x3 LDDB. The size of a block map is less than the size of a node map, and as the size of the map increases, the memory savings from using a block map instead of a node map will eventually out-weight the cost of storing the LDDB. It may be more practical memory-wise to use block maps for large search spaces. Furthermore, many patterns in the LDDB are irrelevant and can be removed for more space.

Recalling Figure 2, we noted that Block A* $examines$ a larger search space as A*. The worst case scenario for Block A* is when A* search finds a direct path to the goal. While the search area is a straight line, Block A* will explore many unnecessary nodes, as it must process the entire block. In game maps with large open spaces, this can be mitigated by a LOS check from start to goal before calling Block A* (this trick is also used in games to check if A* is required). Another issue with Block A* is that the cost of revisiting a block is very expensive, as it must update many nodes in the block. This can be mitigated (but not avoided) by using a consistent heuristic.

6 Conclusion

Block A* is a new algorithm that is a generalization of A* that searches a block of nodes instead of one node per iteration . Block A* is different than other enhancements in that it is more general and more scalable– one LDDB can solve every map of any size without any prior knowledge of the map. This particularly benefits games with randomly generated maps. The performance gain of Block A* is the combination of its LDDB usage and the more efficient exploration of the search space. It should be noted that Block A* works just as well for traditional 4-way and 8-way pathfinding while returning the same optimal path as A*. In our experiments with any-angle path planning, Block A* is at least twice as fast as A* while providing shorter and smoother paths, paths that are very close ground-truth optimal. The next best any-angle path planning algorithm, Theta*, is an order of magnitude slower than Block A* in the worst case, but the difference between their paths is less than 0.1%.

This paper introduces populated game maps to the research community, as we believe it is a more realistic testbed for algorithmic performance compared to the standard testbeds. Our experiments also tested over the standard random obstacle maps and unpopulated game maps. We find that Theta* is always slower than A*, especially for game maps. We find that v-graphs works well for unpopulated (static) game maps but poorly for populated (dynamic) game maps. Finally, Block A* is always faster than A* in all these domains while returning shorter and smoother paths.

7 Acknowledgments

We thank the reviewers for their insightful comments; and Nathan Sturtevant and Dave Churchill for the game maps. This research was funded by NSERC and iCORE.

References

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

de Berg, M.; van Kreveld, M.; and Schwarzkopf, O. 2000. Chapter 15: Visibility graphs. In *Computational Geometry: 2nd Edition*, 307–317.

Demyen, D., and Buro, M. 2006. Efficient triangulation-based pathfinding. In *AAAI*.

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

Harabor, D., and Botea, A. 2010. Breaking path symmetries on 4-connected grid maps. In *AIIDE*.

Jurney, C. 2010. On the war path: Tactical AI in Warhammer 40,000: Dawn of War II. In *AIIDE*.

Rabin, S. 2000. A* aesthetic optimizations. In *Game Programming Gems*, 264–271.

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

Yap, P.; Burch, N.; Holte, R.; and Schaeffer, J. 2011. Block A*: Database-driven search with applications in any-angle path-planning. In *AAAI*. To appear. http://webdocs.cs.ualberta.ca/~holte/Publications/aaai11PeterYapFinal.pdf.