Theta*: Any-Angle Path Planning on Grids

Alex Nash and **Kenny Daniel** and **Sven Koenig**
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781, USA
{anash,kfdaniel,skoenig}@usc.edu

Ariel Felner * †
Department of Information Systems Engineering
Ben-Gurion University of the Negev
Beer-Sheva, 85104, Israel
felner@bgu.ac.il

Abstract

Grids with blocked and unblocked cells are often used to represent terrain in computer games and robotics. However, paths formed by grid edges can be sub-optimal and unrealistic looking, since the possible headings are artificially constrained. We present Theta*, a variant of A*, that propagates information along grid edges without constraining the paths to grid edges. Theta* is simple, fast and finds short and realistic looking paths. We compare Theta* against both Field D*, the only other variant of A* that propagates information along grid edges without constraining the paths to grid edges, and A* with post-smoothed paths. Although neither path planning method is guaranteed to find shortest paths, we show experimentally that Theta* finds shorter and more realistic looking paths than either of these existing techniques.

Introduction

We are interested in path planning for computer games and robotics, where a two-dimensional continuous terrain is discretized into square cells that are either blocked (grey) or unblocked (white). Our goal is to find a short and realistic looking path from the start location to the goal location (both at the corners of cells) that does not pass through blocked cells, as shown in Figure 1. We assume for ease of description that the path can pass through diagonally touching blocked cells. Many methods for discretizing continuous terrain have been investigated in computer science (Choset *et al.* 2005), all of which attempt to balance the inherent tradeoff between two conflicting criteria, namely the path planning runtime and the length of the resulting path:

- **Visibility Graphs:** Visibility graphs contain the start vertex, the goal vertex and the corners of all blocked cells (Lozano-Pérez & Wesley 1979). A vertex is connected via a straight line to another vertex if and only if it has line-of-sight to the other vertex, that is, the straight line from it to the other vertex does not pass through a blocked cell. The shortest paths on visibility graphs are also shortest paths in the continuous terrain, as shown in Figure 1 (left). However, path planning is slow on large visibility graphs since the number of edges can be quadratic in the number of cells. More sophisticated path planning methods can find shortest paths faster. Their runtime complexity, however, remains super-linear in the number of cells (Mitchell & Papadimitriou 1991).
- **Grids:** Path planning is faster on grids than visibility graphs, since the number of edges is linear in the number of cells. However, paths formed by grid edges can be sub-optimal and unrealistic looking since the possible headings are artificially constrained (Yap 2002), as shown in Figure 1 (right).

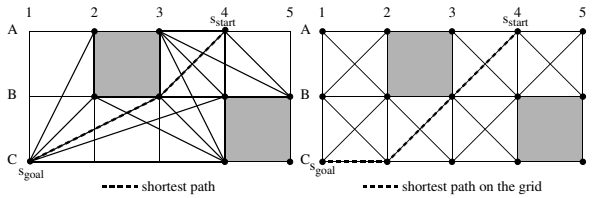

Figure 1: Visibility Graph (L) and Eight-Neighbor Grid (R)

We present Theta*, a variant of A*, which compromises between these two extremes. Theta* propagates information along grid edges (to achieve a short runtime) without constraining the paths to grid edges (to find "any-angle" paths). We show that Theta* is simple, fast and finds short and realistic looking paths.

Notation

We assume an eight-neighbor grid throughout this paper, where S is the set of all grid vertices, $s_{start} \in S$ is the start vertex of the search, and $s_{goal} \in S$ is the goal vertex of the search. $succ(s) \subseteq S$ is the set of neighbors of $s \in S$ that have line-of-sight to s. $c(s, s')$ is the straight-line distance between s and s' (both not necessarily vertices), and $lineofsight(s, s')$ is true if and only if they have line-of-sight.

A*

All path planning methods discussed in this paper build upon A* (Hart, Nilsson, & Raphael 1968), shown in Algorithm 1.[1] [Statements in square brackets are to be ignored for

*We would like to thank Vadim Bulitko from the University of Alberta for making maps from Baldur's Gate II available to us. This research has been partly supported by an NSF award to Sven Koenig under contract IIS-0350584. 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, companies or the U.S. government.

†This work was done while Ariel Felner was on sabbatical at USC.

[1] *open.Insert*(s, x) inserts vertex s with key x into *open*. *open.Remove*(s) removes vertex s from *open*. *open.Pop*() removes a vertex with the smallest key from *open* and returns it.

```
1  Main()
2      g(s_start) := 0;
3      parent(s_start) := s_start;
4      open := ∅;
5      open.Insert(s_start, g(s_start) + h(s_start));
6      closed := ∅;
7      while open ≠ ∅ do
8          s := open.Pop();
9          if s = s_goal then
10             return "path found";
11         closed := closed ∪ {s};
12         [UpdateBounds(s)];
13         foreach s' ∈ succ(s) do
14             if s' ∉ closed then
15                 if s' ∉ open then
16                     g(s') := ∞;
17                     parent(s') := NULL;
18                 UpdateVertex(s, s');
19     return "no path found";
20 end
21 UpdateVertex(s,s')
22     if g(s) + c(s, s') < g(s') then
23         g(s') := g(s) + c(s, s');
24         parent(s') := s;
25         if s' ∈ open then
26             open.Remove(s');
27         open.Insert(s', g(s') + h(s'));
28 end
```

Algorithm 1: A*

now.] To focus its search, A* uses h-values $h(s)$ that approximate the goal distances of the vertices $s \in S$. We use the consistent octile distances in the experiments, that is, the shortest distance on an eight-neighbor grid without blocked cells. A* maintains two values for every vertex s: **(1)** The g-value $g(s)$, which is the length of the shortest path from the start vertex to s found so far. **(2)** The parent *parent*(s), which is used to extract the path after the search halts. Path extraction is done by repeatedly following the parent pointers from the goal vertex to the start vertex. A* also maintains two global data structures: **(1)** The open list, a priority queue that contains vertices to be considered for expansion. **(2)** The closed list, which contains vertices that have already been expanded and ensures that each vertex is expanded only once. A* updates the g-value and parent of an unexpanded successor s' of vertex s (procedure UpdateVertex) by considering the path from the start vertex to s [= $g(s)$] and from s to s' in a straight line [= $c(s, s')$], resulting in a length of $g(s) + c(s, s')$. It updates the g-value and parent of s' if this new path is shorter than the shortest path from the start vertex to s' found so far [= $g(s')$].

A* with Post-Smoothing

Paths formed by grid edges can be sub-optimal and unrealistic looking. A simple approach to improving the paths is to smooth the path found by A* in a post-processing step. A* with Post-Smoothing (A* PS) first runs A* to find a path

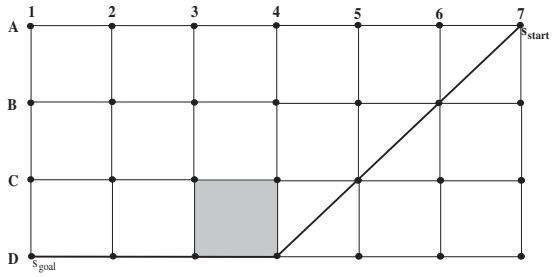

Figure 2: Sub-Optimality of A* with Post-Smoothing

$s_0, s_1, \ldots, s_n$ formed by grid edges, and then smoothes the path as follows (Botea, Müller, & Schaeffer 2004): A* PS uses the first vertex on the path (s_0) as the current vertex. A* PS checks whether the current vertex (s_0) has line-of-sight to the parent of its parent on the path (s_2). If so, A* PS removes the parent (s_1) of the current vertex from the path and repeats the procedure by checking again whether the current vertex (s_0) has line-of-sight to the parent of its parent on the path (s_3), and so on. If not, A* PS uses the parent (s_1) of the current vertex as the current vertex and repeats the procedure by checking whether the current vertex (s_1) has line-of-sight to the parent of its parent on the path (s_3), and so on. To focus its search, we use the consistent straight-line distances $h(s) = c(s, s_{goal})$ in the experiments since the octile distances find paths that are less effectively shortened by this post smoothing technique. A* PS finds significantly shorter paths than A* but is not guaranteed to find shortest paths, as shown in Figure 2. A* finds the path shown in the figure (since it is a shortest path formed by grid edges), and the smoothing method cannot shorten this path. However, the shortest path is $A7$, $C3$ and $D1$. We therefore develop smarter path planning methods, called Basic and Angle-Propagation Theta*, that consider paths that are not constrained to grid edges during the search and can thus make more informed decisions during the search.

Basic Theta*

The key difference between Theta* and A* is that Theta* allows the parent of a vertex to be any vertex, unlike A* where the parent must be a successor. We first introduce a basic variant of Theta* (Basic Theta*), shown in Algorithm 2. Procedure Main is identical to that of Algorithm 1 and thus is not shown. [Statements in square brackets are still to be ignored.] To focus its search, we use the consistent straight-line distances $h(s) = c(s, s_{goal})$ in the experiments since the octile distances can overestimate the goal distances when paths are not constrained to grid edges. Figure 3 shows a trace of Basic Theta*. The vertices are labeled with their g-values and their parents. The hollow circle indicates which vertex is currently being expanded, and the solid circles indicate vertices that have already been expanded. The start vertex $A4$ is expanded first (left) and $B3$ is expanded next (right).

Basic Theta* is identical to A* except that Basic Theta* updates the g-value and parent of an unexpanded successor s' of vertex s (procedure UpdateVertex) by considering the

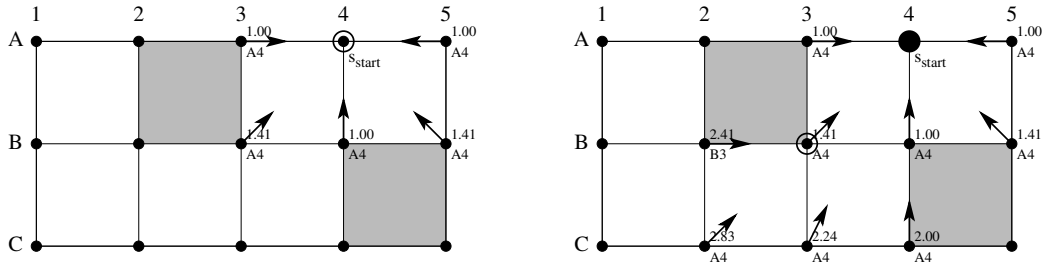

Figure 3: Example Trace of Basic Theta*

```
29 UpdateVertex(s,s')
30     if lineofsight(parent(s), s') then
31         /* Path 2 */
32         if g(parent(s)) + c(parent(s), s') < g(s') then
33             g(s') := g(parent(s)) + c(parent(s), s');
34             parent(s') := parent(s);
35             if s' ∈ open then
36                 open.Remove(s');
37             open.Insert(s', g(s') + h(s'));
38     else
39         /* Path 1 */
40         if g(s) + c(s, s') < g(s') then
41             g(s') := g(s) + c(s, s');
42             parent(s') := s;
43             if s' ∈ open then
44                 open.Remove(s');
45             open.Insert(s', g(s') + h(s'));
46 end
```

Algorithm 2: Basic Theta*

following two paths:

- **Path 1:** As done by A*, Basic Theta* considers the path from the start vertex to s [= $g(s)$] and from s to s' in a straight line [= $c(s, s')$], resulting in a length of $g(s) + c(s, s')$ (line 40).
- **Path 2:** To allow any-angle paths, Basic Theta* also considers the path from the start vertex to *parent*(s) [= $g($*parent*$(s))$] and from *parent*(s) to s' in a straight line [= $c($*parent*$(s), s')$], resulting in a length of $g($*parent*$(s)) + c($*parent*$(s), s')$ if s' has line-of-sight to *parent*(s) (line 32). The idea behind considering Path 2 is that Path 2 is no longer than Path 1 due to the triangle inequality if s' has line-of-sight to *parent*(s).

Basic Theta* updates the g-value and parent of s' if either path is shorter than the shortest path from the start vertex to s' found so far [= $g(s')$]. For example, consider Figure 3 (right) where $B3$ (with parent $A4$) gets expanded. $B2$ is an unexpanded successor of $B3$ which does not have line-of-sight to $A4$ and thus is updated according to Path 1. $C3$ is an unexpanded successor of $B3$ which does have line-of-sight to $A4$ and thus is updated according to Path 2.

Basic Theta* (without a closed list) can re-expand vertices even if the h-values are consistent because the f-values of the expanded vertices are not guaranteed to be monotonically non-decreasing over time. The closed list prevents Basic Theta* from re-expanding vertices, which makes it faster but the path slightly longer.

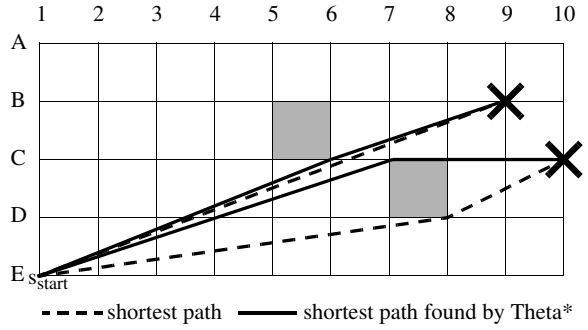

Figure 4: Mistakes of Basic Theta*

Basic Theta* is simple, fast and finds short and realistic looking paths, but is not guaranteed to find shortest paths (even if it re-expands vertices), as shown in Figure 4. Basic Theta* does not find shortest paths from the start vertex $E1$ to the two vertices marked with an X (but even these paths are less than 0.2 percent longer than minimal). The reason for this is that a vertex p can only become the parent of another vertex s if either p is a successor of s (Path 1) or p is the parent of a successor of s (Path 2). $D8$ should be the parent of $C10$ since this results in a shortest path from the start vertex $E1$ to $C10$. However, none of the successors of $C10$ have $D8$ as a parent since the shortest paths from the start vertex $E1$ to them move around the blocked cell in different ways. For example, $C7$ is correctly the parent of $C9$, and $E1$ is correctly the parent of $D9$. Similarly, $E1$ should be the parent of $B9$ but none of the successors of $B9$ have it as a parent because none of them have line-of-sight to $E1$ through the small gap formed by the two blocked cells.

Angle-Propagation Theta*

Basic Theta* needs to perform many line-of-sight checks, whose runtime per vertex expansion can be linear in the number of cells, which can potentially degrade its runtime. Angle-Propagation Theta* (AP Theta*) performs the line-of-sight checks in constant time per vertex expansion by calculating and maintaining angle ranges incrementally prior to expanding vertices, as shown in Algorithm 3. Procedure Main is identical to that of Algorithm 1 and thus is not shown. [Statements in square brackets are now to be ex-

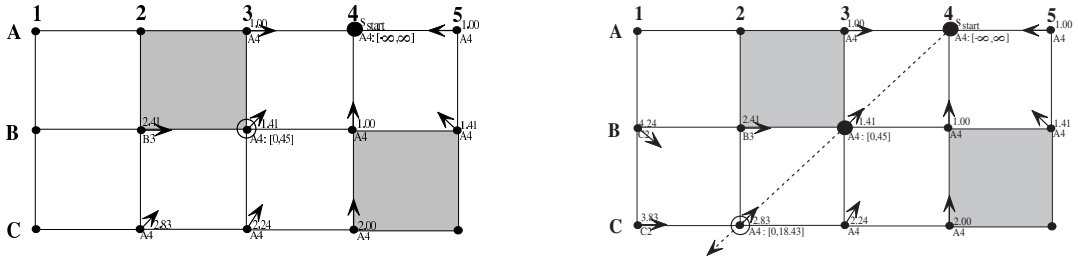

Figure 5: Example Trace of Angle-Propagation Theta* (AP Theta*)

ecuted.] Figure 5 shows a trace of AP Theta*, where the vertices are now also labeled with the angle ranges.

Angle Ranges

AP Theta* maintains two additional values for each vertex s: a lower angle bound $lb(s)$ and an upper angle bound $ub(s)$ that together form the angle range $[lb(s), ub(s)]$. To explain their meaning, we need to define $\Theta(s, p, s')$, which gives Theta* its name. $\Theta(s, p, s')$ denotes the angle $\angle(s, p, s')$ in the range $[-180°, 180°)$. This angle is positive if the ray from p through s' is strictly counterclockwise of the ray from p through s. Now consider a vertex s with successor s'. By constraining the angle range of s appropriately, AP Theta* maintains the following invariant: If $lb(s) \leq \Theta(s, parent(s), s') \leq ub(s)$ then s' is guaranteed to have line-of-sight to $parent(s)$ (line 48).

For example, in Figure 5 (left), $lb(B3) = 0°$ and $ub(B3) = 45°$. $\Theta(B3, A4, C3) = 18°$ and thus $C3$ is guaranteed to have line-of-sight to $A4$. On the other hand, $\Theta(B3, A4, B2) = -18°$ and thus $B2$ is not guaranteed to have line-of-sight to $A4$. AP Theta* thus assumes that $B2$ does not have line-of-sight to $A4$.

Maintaining Angle Ranges

AP Theta* constrains the angle range of a vertex s before it expands the vertex, based on its adjacent cells and successor vertices if $s \neq s_{start}$.

First, AP Theta* uses any blocked cells c adjacent to vertex s to constrain the angle range of the vertex s being expanded (line 68). If each corner s' of c satisfies the following condition (Condition 1): **(1)** $parent(s) = s'$ or **(2)** $\Theta(s, parent(s), s') < 0°$ or **(3)** $\Theta(s, parent(s), s') = 0°$ AND $c(parent(s), s') \leq c(parent(s), s)$, then AP Theta* assumes that vertices do not have line-of-sight to $parent(s)$ if the rays from $parent(s)$ through them are clockwise of the ray from $parent(s)$ through s. It therefore sets $lb(s) = 0°$. Likewise, if each corner s' of c satisfies the following condition: **(1)** $parent(s) = s'$ or **(2)** $\Theta(s, parent(s), s') > 0°$ or **(3)** $\Theta(s, parent(s), s') = 0°$ AND $c(parent(s), s') \leq c(parent(s), s)$, then AP Theta* assumes that vertices do not have line-of-sight to $parent(s)$ if the rays from $parent(s)$ through them are counterclockwise of the ray from $parent(s)$ through s. It therefore sets $ub(s) = 0°$.

For example, in Figure 5 (left), AP Theta* constrains the lower angle bound of $B3$ using this property. All corners of the blocked cell that $B3$ is adjacent to satisfy Condition 1. AP Theta* thus assumes that vertices do not have line-of-sight to $A4$ if the rays from $A4$ through them are clockwise of the ray from $A4$ through $B3$. It therefore sets $lb(B3) = 0°$.

Second, AP Theta* uses any successor s' of vertex s to constrain the angle range of the vertex s being expanded, if s' satisfies the following condition (Condition 2): **(1)** s' is unexpanded or has a parent other than $parent(s)$ and **(2)** s' is closer to $parent(s)$ than s itself and **(3)** s' is not equal to $parent(s)$ (line 73). Such vertices have no line-of-sight information regarding $parent(s)$, yet it would be important to propagate such information from them to s. AP Theta* therefore assumes conservatively that s' barely has line-of-sight to $parent(s)$. This assumption might over constrain the angle range, but avoids paths that pass through blocked cells.

For example, in Figure 5 (left), AP Theta* constrains the upper angle bound of $B3$ using this property. $B4$ is a successor of $B3$, that is unexpanded, closer to $A4$ than $B3$ and not equal to $A4$. Thus it satisfies Condition 2. AP Theta* therefore assumes that $B4$ barely has line-of-sight to $A4$, and sets $ub(B3) = \Theta(B3, A4, B4) = 45°$.

Third, AP Theta* uses any successor s' of vertex s to constrain the angle range of the vertex s being expanded, if s' satisfies the following condition (Condition 3): **(1)** s' is expanded and **(2)** s' has parent $parent(s)$ and **(3)** $s \neq s_{start}$ (line 79). Such vertices have line-of-sight information regarding $parent(s)$, and it is important to propagate such information from them to s. AP Theta* therefore tightens the angle range of s by intersecting it with the angle range of s'.

For example, in Figure 5 (right), AP Theta* constrains the lower angle bound of $C2$ using this property. $B3$ is a successor of $C2$, that is expanded, has the same parent as $C2$ and is not equal to $A4$. Thus it satisfies Condition 3. AP Theta* therefore intersects their angle ranges and sets $lb(C2) = 0°$.

Updating g-values and Parents

AP Theta* updates the g-value and parent of an unexpanded successor s' of vertex s by considering the following two cases:

- **Path 2:** First, AP Theta* considers Path 2 (line 50) as defined for Basic Theta* and updates the g-value and parent of s' in the same way. In Figure 5 (left), $C3$ is updated this way.
- **Path 1:** Otherwise, AP Theta* considers Path 1 (line 58) as defined for Basic Theta* and A* and updates the g-

```
47 UpdateVertex(s,s')
48     if s ≠ s_start AND lb(s) ≤ θ(s, parent(s), s') ≤ ub(s) then
49         /* Path 2 */
50         if g(parent(s)) + c(parent(s), s') < g(s') then
51             g(s') := g(parent(s)) + c(parent(s), s');
52             parent(s') := parent(s);
53             if s' ∈ open then
54                 open.Remove(s');
55             open.Insert(s', g(s') + h(s'));
56     else
57         /* Path 1 */
58         if g(s) + c(s, s') < g(s') then
59             g(s') := g(s) + c(s, s');
60             parent(s') := s;
61             if s' ∈ open then
62                 open.Remove(s');
63             open.Insert(s', g(s') + h(s'));
64 end
65 UpdateBounds(s)
66     lb(s) := −∞; ub(s) := ∞;
67     if s ≠ s_start then
68         foreach blocked cell b adjacent to s do
69             /* Condition 1 */
70             update lb(s) and ub(s) (see main text);
71         foreach s' ∈ succ(s) do
72             /* Condition 2 */
73             if (s' ∉ closed OR parent(s) ≠ parent(s')) AND
               c(parent(s), s') < c(parent(s), s) AND parent(s) ≠ s'
               then
74                 if θ(s, parent(s), s') < 0 then
75                     lb(s) := max(lb(s), θ(s, parent(s), s'));
76                 if θ(s, parent(s), s') > 0 then
77                     ub(s) := min(ub(s), θ(s, parent(s), s'));
78             /* Condition 3 */
79             if s' ∈ closed AND parent(s) = parent(s') AND
               s' ≠ s_start then
80                 if lb(s') + θ(s, parent(s), s') ≤ 0 then
81                     lb(s) :=
                       max(lb(s), lb(s') + θ(s, parent(s), s'));
82                 if 0 ≤ ub(s') + θ(s, parent(s), s') then
83                     ub(s) :=
                       min(ub(s), ub(s') + θ(s, parent(s), s'));
84 end
```

Algorithm 3: Angle-Propagation Theta* (AP Theta*)

value and parent of vertex s' in the same way. In Figure 5 (left), $B2$ is updated this way.

Properties

We can prove that AP Theta* finds a path from the start vertex to the goal vertex if such a path exists, simply because it considers all grid edges. Furthermore, this path indeed does not pass through blocked cells. (We omit the proof due to space limitations.) AP Theta* tends to find slightly longer paths than Basic Theta* because it can over constrain the angle ranges, which incorrectly rules out some paths.

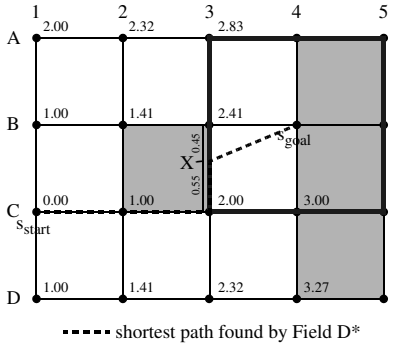

Figure 6: Linear Interpolation Error of Field D* (FD*)

Field D*

Field D* (Ferguson & Stentz 2006) (FD*) is the only other variant of A* that propagates information along grid edges without constraining the paths to grid edges, as far as we know. We implemented a non-incremental variant of FD* that uses A* to search from the start vertex to the goal vertex instead of D* Lite which searches from the goal vertex to the start vertex. However, both of these variants find the same paths. To focus its search, we use the consistent straight-line distances $h(s) = c(s, s_{goal})$ in the experiments.

FD* updates the g-value and parent of an unexpanded successor s' of vertex s by considering the paths from the start vertex to any point X on the perimeter of s' [= $g(X)$] and from X to s' in a straight line [= $c(X, s')$], resulting in a length of $g(X) + c(X, s')$. It updates the g-value and parent of s' if this new path is shorter than the shortest path from the start vertex to s' found so far [= $g(s')$]. For example, the perimeter of $s' = B4$ is formed by connecting all of the neighbors of $B4$ and shown in bold in Figure 6. Consider point X on the perimeter, whose neighbors are $B3$ and $C3$. Since g-values are only stored for vertices, the g-value of X is linearly interpolated using $g(B3) = 2.41$ and $g(C3) = 2.00$ to get $g(X) = 0.55 \times 2.41 + 0.45 \times 2.00 = 2.23$. This value is too small, for a simple reason: There are short paths from the start vertex $C1$ to $B3$ (around the blocked cell in the clockwise direction) and from the start vertex $C1$ to $C3$ (around the blocked cell in the counterclockwise direction). Thus, linear interpolation concludes that there must also be an equally short path from the start vertex $C1$ to X, which is not the case. It turns out that X minimizes $g(X') + c(X', B4)$ among all points X' on the perimeter of $B4$ and thus becomes the parent of $B4$. The path found by FD* in Figure 6 demonstrates that path extraction for FD* must be done with care and that the paths found by FD* are susceptible to unnecessary heading changes. The authors of FD* recognize this problem and suggest to use a one-step lookahead method during path extraction (Ferguson & Stentz 2006). This smoothing method allows FD* to avoid some of the unnecessary heading changes, like the one in Figure 6, but does not eliminate all of them. We used it when generating our experimental results for FD*.

Experimental Results

We now compare the average path lengths and runtimes of Field D* (FD*), Angle-Propagation Theta* (AP Theta*),

		FD*	Basic Theta*	AP Theta*	Shortest Paths	A*	A* PS
100×100	Game Maps	41.98 (0.0126)	41.92 (0.0063)	42.01 (0.0070)	41.89 (0.6490)	43.80 (0.0029)	42.00 (0.0060)
	Random 0%	51.88 (0.0109)	51.80 (0.0026)	51.80 (0.0034)	51.80 (0.0020)	54.63 (0.0015)	51.80 (0.0057)
	Random 5%	48.83 (0.0097)	48.74 (0.0022)	48.74 (0.0038)	48.69 (0.0311)	51.24 (0.0013)	48.99 (0.0048)
	Random 10%	50.64 (0.0120)	50.53 (0.0028)	50.54 (0.0051)	50.45 (0.1173)	53.11 (0.0014)	50.91 (0.0054)
	Random 20%	48.65 (0.0135)	48.54 (0.0034)	48.55 (0.0065)	48.43 (0.4594)	50.86 (0.0019)	49.04 (0.0054)
	Random 30%	50.19 (0.0153)	50.10 (0.0045)	50.11 (0.0081)	49.98 (1.0769)	52.25 (0.0028)	50.61 (0.0058)
500×500	Game Maps	205.60 (0.1916)	205.28 (0.0988)	206.20 (0.1624)	N/A	214.80 (0.0661)	205.64 (0.1040)
	Random 0%	259.65 (0.1231)	259.24 (0.0288)	259.24 (0.0113)	N/A	273.11 (0.0045)	259.24 (0.1688)
	Random 5%	257.19 (0.1538)	256.58 (0.0390)	256.60 (0.0523)	N/A	270.40 (0.0053)	259.14 (0.1747)
	Random 10%	259.37 (0.1795)	258.62 (0.0577)	258.65 (0.0870)	N/A	271.77 (0.0108)	261.62 (0.1783)
	Random 20%	258.71 (0.2219)	257.88 (0.0882)	257.93 (0.1384)	N/A	270.60 (0.0273)	261.36 (0.1871)
	Random 30%	266.49 (0.3207)	265.84 (0.1244)	265.90 (0.2000)	N/A	277.57 (0.0628)	269.60 (0.1951)

Table 1: Experimental Results: Path Lengths (in Parenthesis: Run Times)

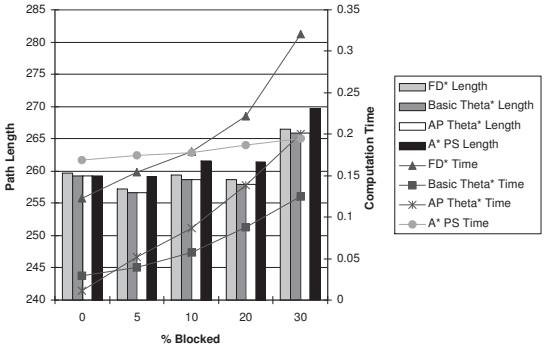

Figure 7: Path Length and Run Time (Random Grids 500×500)

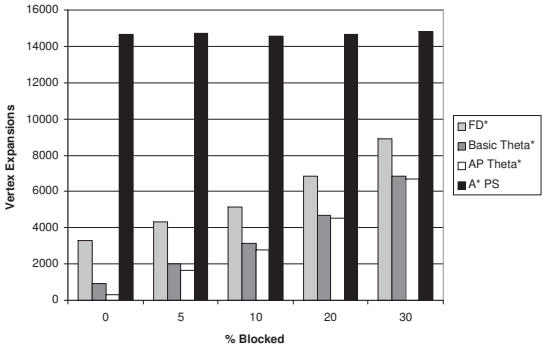

Figure 8: Vertex Expansions (Random Grids 500×500)

Basic Theta*, Shortest Paths (computed with A* on visibility graphs), A* (on an eight-neighbor grid), and A* with Post-Smoothing (on an eight-neighbor grid, A* PS) run on the same path planning problems. The path planning problems are characterized by two different grid sizes and six different kinds of grids, namely grids with a given percentage of randomly blocked cells (random grids) and scaled maps from the real-time strategy game Baldur's Gate II (game maps) (Bulitko, Sturtevant, & Kazakevich 2005). Table 1 summarizes our results, averaged over either 500 path planning problems for random grids of size 100×100, 500 path planning problems for random grids of size 500×500 or 118 path planning problems on game maps, with randomly

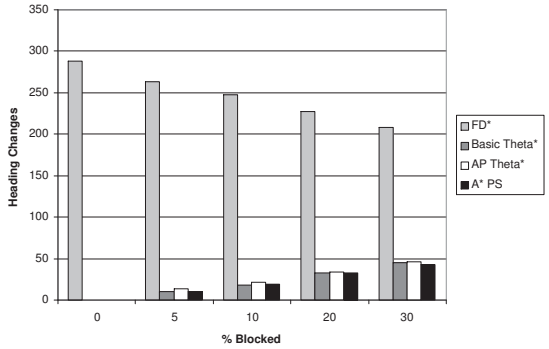

Figure 9: Heading Changes (Random Grids 500×500)

chosen start and goal vertices in all cases. The path lengths are given outside of the parentheses, and the runtimes (measured in seconds) are given inside of the parentheses. All path planning methods were implemented in C# and executed on a 3.7 GHz Core 2 Duo with 2 GByte of RAM. Our implementations are not optimized for performance and can possibly be improved. We break ties among vertices with the same f-values for all path planning methods in favor of larger g-values (as usual), but in the opposite direction for AP Theta* and Basic Theta* because this tie-breaking scheme found shorter paths. It took too long to find the shortest paths on grids of size 500×500, which is why these results are omitted from the table. We graphically depict some relationships for random grids of size 500×500 with different percentages of randomly blocked cells. Figure 7 shows the path lengths and runtimes (from the table), Figure 8 shows the number of vertex expansions, and Figure 9 shows the number of heading changes on the paths.

We found that Basic Theta* and AP Theta* find much shorter paths than A* and shorter paths than both FD* and A* PS on random grids, as expected, and Basic Theta* finds slightly shorter paths than AP Theta*. For example, on random grids of size 500×500 with 20 percent randomly blocked cells, Basic Theta* finds shorter paths than A* PS 95 percent of the time, shorter paths than FD* 93 percent of the time, and shorter paths than AP Theta* 57 percent of the time (and paths with the same length as AP Theta* 17 percent of the time). Overall, Basic Theta* has the best

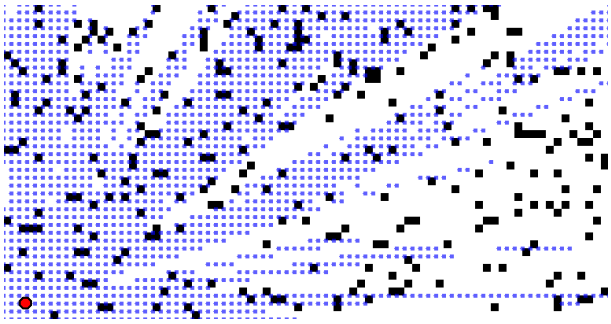

Figure 10: Shortest Paths Found by AP Theta*

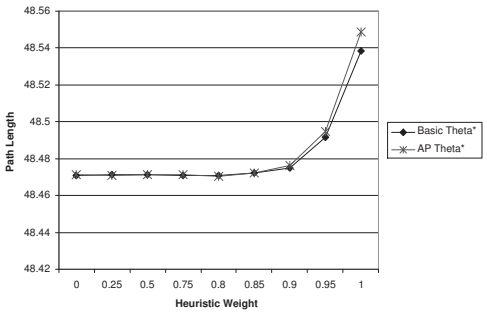

Figure 11: Weighted h-Values (Random Grids 100×100)

trade-off between path length and runtime among all path planning methods. In particular, it finds paths of close-to-minimal lengths faster than any other path planning method.

Both Basic Theta* and AP Theta* find paths of close-to-minimal lengths. Their paths contain far fewer heading changes than the paths of FD*, which explains why the FD* paths are longer and less realistic looking. Because AP Theta* finds paths of close-to-minimal lengths, it does not seem to over constrain the angle ranges very often, or the resulting effect on the path lengths is negligible. Figure 10 illustrates how often AP Theta* finds shortest paths. Blocked cells are represented by black boxes, and the start vertex is represented by the concentric circles in the lower left corner. AP Theta* finds shortest paths from the start vertex to all vertices represented by shaded circles.

It is possible to make both Basic Theta* and AP Theta* find even shorter paths at the expense of larger runtimes. Remember that they both use a closed list to prevent the re-expansion of vertices, which makes them faster at a cost of slightly longer paths. One can control this effect by using the h-values $h(s) = w \times c(s, s_{goal})$ for a constant $0 < w < 1$, which is similar to Weighted A* (Pohl 1973) except that Weighted A* typically uses a constant larger than one. Even with $w = 0$ neither version of Theta* finds shortest paths as we showed earlier in Figure 4. Figure 11 shows the effect of different values for w on path length, for the same 500 random grids of size 100×100 with 20 percent randomly blocked cells.

Conclusions

In this paper, we introduced Theta*, a path planning method that propagates information along grid edges without constraining the paths to grid edges. Theta* is simple, fast and finds short and realistic looking paths. A* with Post-Smoothing and Field D* are more general than Theta*. For example, they can be applied to grids whose cells have different sizes and traversal costs. However, Basic Theta* found shorter paths for our path planning problems faster than both A* with Post-Smoothing and Field D*, without complicated path extraction or smoothing methods. As opposed to A* and A* with Post-Smoothing, Theta* considers paths that are not constrained to grid edges during the search and can thus make more informed decisions during the search. As opposed to Field D*, Theta* exploits the fact that shortest paths for our path planning problems have heading changes only at the corners of blocked cells, which allows it to eliminate long and unrealistic looking paths from consideration.

Future research will be directed towards reducing the length of the paths Theta* finds, by exploiting our understanding of the causes for its sub-optimality. In addition, future research will also be directed towards introducing an incremental variant of Theta* to support re-planning and extending Theta* to grids whose cells have non-uniform sizes and traversal costs.

References

Botea, A.; Müller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. *Journal of Game Development* 1(1):1–22.

Bulitko, V.; Sturtevant, N.; and Kazakevich, M. 2005. Speeding up learning in real-time search via automatic state abstraction. In *Proceedings of the National Conference on Artificial Intelligence*, 1349–1354.

Choset, H.; Lynch, K.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L.; and Thrun, S. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

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

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* SCC-4(2):100–107.

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

Mitchell, J., and Papadimitriou, C. 1991. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM* 38(1):18–73.

Pohl, I. 1973. The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 12–17.

Yap, P. 2002. Grid-based path-finding. In *Proceedings of the Canadian Conference on Artificial Intelligence*, 44–55.