Planning Paths with Fewer Turns on Grid Maps

Hu Xu and **Lei Shu** and **May Huang**
AI Research Laboratory
International Technological University
San Jose, CA, USA
aaronhuxu@gmail.com leishu02@gmail.com mhuang@itu.edu

Abstract

In this paper, we consider the problem of planning any-angle paths with small numbers of turns on grid maps. We propose a novel heuristic search algorithm called Link* that returns paths containing fewer turns at the cost of slightly longer path lengths. Experimental results demonstrate that Link* can produce paths with fewer turns than other any-angle path planning algorithms while still maintaining comparable path lengths. Because it produces this type of path, artificial agents can take advantage of Link* when the cost of turns is expensive.

Introduction

In robotics and video games, grids with blocked and unblocked cells are commonly used as navigation maps (Yap 2002). On these grids, the path planning problem is usually defined as finding a non-blocked path between two vertices, which can be solved as standard shortest path problem. To save running time, A* (Hart, Nilsson, and Raphael 1968) and its variants use heuristic search to provide optimal or sub-optimal solutions. Furthermore, any-angle path planning algorithms improve A* by not constraining heading changes on the paths along grid edges. Among these algorithms, Theta* (Nash et al. 2007; Daniel et al. 2010) produces any-angle paths with shorter lengths and fewer heading changes (a.k.a turns) by a line-of-sight check at every node expansion.

Because they do not figure the number of heading changes into path quality determination, these shortest-path algorithms can easily produce paths with many unnecessary turns. In practice, paths with fewer turns are sometimes preferred when turning costs are high. For example, in trajectory planning of robotics, this kind of path can simplify control complexity (Choset et al. 2005); robotic vehicles can also maintain a higher average speed by following a path smoothed from this kind of path. In order to meet these needs, minimizing turns is factored into the determination of path quality. In computational geometry, this is known as the minimum-link path problem: given a set of disjoint simple polygons with n vertices and two points s and g, the minimum-link path is a polygonal chain connecting s to g with the minimum number of edges without intersecting the polygons. Given a fixed starting point and a simple polygon without holes, such a path can be generated in $O(\log n)$ query time after $O(n)$ preprocessing time (Suri 1986). Path queries between any two points can be generated after $O(n^3)$ preprocessing time (Arkin, Mitchell, and Suri 1992). Given a fixed starting point and a set of disjoint simple polygons, a path can be generated after $O(n^4)$ preprocessing time (Mitchell, Rote, and Woeginger 1990; 1992). When applied to grids, similar to computing shortest-path using a visibility graph on grids (Nash et al. 2007; Daniel et al. 2010), these algorithms take expensive running time.

Extending from the minimum-link path problem, we propose a heuristic search algorithm producing any-angle paths with fewer turns on grid maps. Borrowing the term "link" from computational geometry, we named our algorithm "Link*". Link* encompasses three related algorithms: Basic Link*, Enhanced Link* and Weighted Link*. Experimental results show that Link* produces paths with fewer than 60% of the turns and no more than 122% of the path length of those produced by Theta*, while their running times remain comparable.

In this paper, we first describe grid map representation and notation; we review Theta* and the framework; and then we present three Link* algorithms: Basic Link*, Enhanced Link* and Weighted Link*. After that, we provide experimental results comparing Link* with A* and Theta*.

Preliminaries

In this section, we briefly review grid map representation and notation. Then, we describe the Theta* algorithm and the framework that is used by Link*.

Grid Map Representation and Notation

There are two major grid map models: corner-vertex and center-vertex (Choi and Yu 2011), as depicted in Figure 1. The corner-vertex model defines all vertices at the corners of square cells. The center-vertex model defines vertices at the centers of square cells. We use the corner-vertex model for grid maps in this paper, but Link* can be applied to either model. We assume each vertex has eight neighboring vertices and non-blocked paths can pass through the vertex

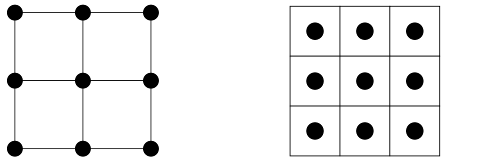

(a) Corner-vertex model (b) Center-vertex model

Figure 1: Two models of grid representation

shared diagonally by two blocked cells, as depicted in Figure 2.

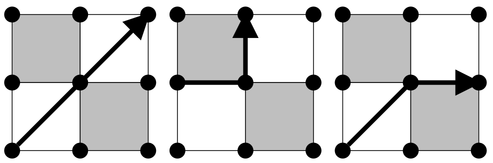


Figure 2: Examples of paths passing through the vertex shared diagonally by two blocked cells

Let S be a set of all vertices in grids and s be a vertex in S. Let the vertex sequence: $s_s s_1 \dots s_n s_g$ denotes a non-blocked path connecting the start vertex s_s to the goal vertex s_g. Three consecutive vertices in this sequence can form a turn (heading change). A vertex sequence with $n+2$ vertices has exactly n turns.

Theta* and the Framework

Theta* (Nash et al. 2007; Daniel et al. 2010) differs from A* primarily in that it can perform any-angle path planning. Link* algorithm proposed in this paper is based on the framework of Theta*.

Each vertex s in Theta* contains three values: a parent pointer $p(s)$, an f-value $f(s)$ and a g-value $g(s)$. The parent vertex pointer $p(s)$ stores the pointer to the previous vertex of s in the path between s and the start vertex s_s. The f-value is defined as a distance function: $f(s) = g(s) + h(s)$, where $g(s)$ is the g-value that represents the length of currently obtained path between s_s and s and $h(s)$ is the Euclidean distance between s and s_g. Initially, $p(s_s)$, $f(s_s)$ and $g(s_s)$ are initialized as s_s, the Euclidean distance from s_s to s_g and 0, respectively; $p(s)$, $f(s)$ and $g(s)$ of other vertices are initialized as NULL, infinity and infinity accordingly. The *expansion* operation of a vertex s examines its visible neighboring vertices and updates three values of these vertices, accordingly.

The framework of Theta* is described in Algorithm 1, which maintains two global sets of vertices: an open set O and a closed set C, where O keeps track of all vertices to be expanded and C contains all expanded vertices. A priority queue is built on top of O to enable the query of the vertex with the smallest f-value. Initially, vertex s_s is initialized by procedure InitializeVertex(s) and put into $O = \{s_s\}$. Then vertex $s_e = s_s$ is expanded and moved from O to C. Procedure VisibleNeighbors(s_e) only returns

Algorithm 1: The Framework of Theta*

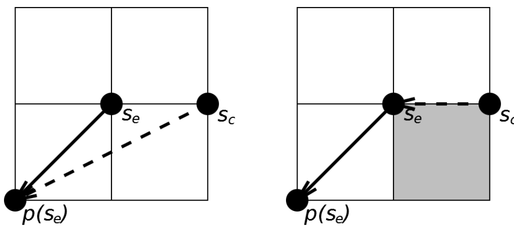
```
Input: a grid map with blocked and unblocked cells, a
       start vertex $s_s$ and a goal vertex $s_g$.
Output: a non-blocked path connecting $s_s$ to $s_g$.

InitializeVertex($s_s$)
$O \leftarrow s_s$
$C \leftarrow \emptyset$
while $O \neq \emptyset$ do
    remove $s_e$ with the smallest $f$-value from $O$
    if $s_e = s_g$ then
        return ExtractPath($s_s$, $s_g$)
    end
    $C \leftarrow C \cup \{s_e\}$
    //Vertex Expansion
    foreach $s_c \in$ VisibleNeighbors($s_e$) do
        if $s_c \notin C$ then
            if $s_c \notin O$ then
                InitializeVertex($s_c$)
            end
            if LineofSight($p(s_e)$, $s_c$) then
                ChoosePath1($s_e, s_c$)
            else
                ChoosePath2($s_e, s_c$)
            end
        end
    end
end
return NULL        //Path Not Found
```

visible neighbors of s_e. During vertex expansion, every vertex s_c ∈VisibleNeighbors(s_e) is examined and all vertex values are updated and then s_c is added into O if not already in it and priority queue is updated. Algorithm 1 keeps expanding vertices and terminates only when the goal vertex s_g is expanded or every vertex in the same connected component as s_s in grid graph is expanded. The former case indicates a path is found and ExtractPath(s_s, s_g) retrieves the path by repeatedly following parent vertex from s_g to s_s. The latter case indicates that s_g is not reachable from s_s.

When s_c is visited as a visible neighbor of s_e, the framework examines $p(s_c)$ based on the procedure LineofSight($p(s_e), s_c$), which checks whether vertex s_c is visible to $p(s_e)$ using Brensenham's algorithm (Bresenham 1965). It updates $p(s_c)$ and $f(s_c)$ by considering the following two paths:

- **Path1:** when LineofSight($p(s_e), s_c$) returns true, ChoosePath1(s_e, s_c) evaluates the new path from s_s to $p(s_e)$ and from $p(s_e)$ to s_c and updates vertex values if necessary. Theta* updates s_c by computing the length of the new path from s_s to $p(s_e)$ and from $p(s_e)$ to s_c. If the new path is shorter than $g(s_c)$, Theta* updates $p(s_c)$, $f(s_c)$ and $g(s_c)$ for the new path.
- **Path2:** when LineofSight($p(s_e), s_c$) returns false, ChoosePath2(s_e, s_c) evaluates the new path from s_s to s_e and from s_e to s_c and updates vertex values if necessary.

When update happens, a new turn is introduced at vertex s_e. Theta* updates s_c by computing the length of the new path from s_s to s_e and then from s_e to s_c. If this path is shorter than $g(s_c)$, it updates $p(s_c)$, $f(s_c)$ and $g(s_c)$ for the new path.



(a) Path1 (b) Path2

Figure 3: Path1 and path2 of the framework of Theta*

Figure 3 illustrates an example of the choice between path1 and path2. When the bottom right cell is an unblocked cell, vertex s_c is visible to vertex $p(s_e)$. Vertex $p(s_e)$ is considered as the parent of s_c. In this case path1 is formed, which allows the generation of any-angle paths. When the bottom right cell is a blocked cell, vertex s_c is not visible to $p(s_e)$. Vertex s_e is considered as the parent of s_c and path2 is formed.

Algorithm 2: Basic Link* Algorithm

```
InitializeVertex($s$)
if $s = s_s$ then
    $p(s) \leftarrow s_s$
    $f_B(s) \leftarrow 0$
else
    $p(s) \leftarrow NULL$
    $f_B(s) \leftarrow \infty$
end

ChoosePath1($s_e$, $s_c$)
$f'_B(s_c) \leftarrow f_B(p(s_e)) + \Theta(s_g, p(s_e), s_c)$
if $f'_B(s_c) < f_B(s_c)$ then
    $p(s_c) \leftarrow p(s_e)$
    $f_B(s_c) \leftarrow f'_B(s_c)$
    if $s_c \notin O$ then
        $O \leftarrow O \cup \{s_c\}$
    end
end

ChoosePath2($s_e$, $s_c$)
$f'_B(s_c) \leftarrow f_B(s_e) + \Theta(s_g, s_e, s_c)$
if $f'_B(s_c) < f_B(s_c)$ then
    $p(s_c) \leftarrow s_e$
    $f_B(s_c) \leftarrow f'_B(s_c)$
    if $s_c \notin O$ then
        $O \leftarrow O \cup \{s_c\}$
    end
end
```

Link*

Since Theta* finds sub-optimal shortest paths, simply changing it may not work for planning paths with fewer turns. For example, if we take the edge cost as 1 rather than its length, we can simply count the number of turns and take it as part of the f-value. This causes a tie-breaking problem since lots of different paths may have the same number of turns. It's also hard to find a good heuristic value ($h(s)$ of Theta*). So it may be difficult for this algorithm to reach the goal vertex.

We choose the framework of Theta* as the main procedure of our solution and take angle values into f-values. We present three variants of Link*: Basic Link*, Enhanced Link* and Weighted Link*. Basic Link* is fast and easy to implement. Enhanced Link* improves path quality and produces shorter paths than Basic Link*. Weighted Link* can provide a better balance between number of turns and vertex expansions than Basic Link* and Enhanced Link* by controlling the f-value.

Basic Link*

The f-value of each vertex in Basic Link* and its two variants is derived from angles formed by grid vertices.

Given three points A, B and C in the plane, the angle formed by vectors $\overrightarrow{AB}$ and $\overrightarrow{AC}$ is calculated as follows :

$$\Theta(B, A, C) = \begin{cases} 0, & \text{if } A = B \text{ or } A = C \text{ (null vector);} \\ \arccos\left(\frac{\overrightarrow{AB} \cdot \overrightarrow{AC}}{\|\overrightarrow{AB}\| \cdot \|\overrightarrow{AC}\|}\right), & \text{otherwise.} \end{cases}$$

Let $s_s s_1 \ldots s_i \ldots s_k$ denotes a non-blocked path connecting s_s to s_k on grids. Given the goal vertex s_g, the *backward-angle* of vertex s_i is defined as $\Theta(s_g, s_{i-1}, s_i)$, where $1 \le i \le k$. The α-value of vertex s_k is the sum of all backward-angles of vertices along the path from s_1 to s_k:

$$\alpha(s_k) = \sum_{i=1}^{k} \Theta(s_g, s_{i-1}, s_i).$$

In Basic Link*, the f-value of vertex s_k is simply defined as:

$$f_B(s_k) = \alpha(s_k).$$

Algorithm 2 describes Basic Link*. We skip the framework of Theta* and only provide procedures related to vertex updates. Each vertex of Basic Link* contains only two values: a parent vertex $p(s)$ and an f-value $f(s)$. In procedure InitializeVertex, the value $p(s_s)$ and the value $f(s_s)$ of the start vertex s_s are initialized as s_s and 0, respectively; the value $p(s)$ and the value $f(s)$ of the other vertices are initialized as NULL and infinity accordingly, which indicate unknown.

Let $f'_B(s_c)$ denote the f-value of vertex s_c for the new path to be evaluated between s_s and s_c. When considering Path1, we compute $f'_B(s_c)$ for the new path from vertex s_s to $p(s_e)$ and from $p(s_e)$ to s_c by adding $\Theta(s_g, p(s_e), s_c)$ to $f_B(p(s_e))$. If $f'_B(s_c)$ is smaller than $f_B(s_c)$, Algorithm 2 will update $p(s_c)$ and $f_B(s_c)$.

Similarly, when considering Path2, we compute $f'_B(s_c)$ for the new path from vertex s_s to s_e and from s_e to s_c. If $f'_B(s_c)$ is smaller than $f_B(s_c)$, Algorithm 2 will update $p(s_c)$ and $f_B(s_c)$ and a new turn is formed at vertex s_e.

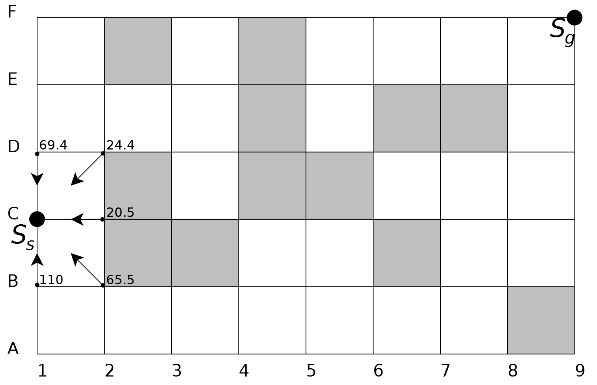


Figure 4: Example trace of Basic Link* with f-values and parent pointers (1)

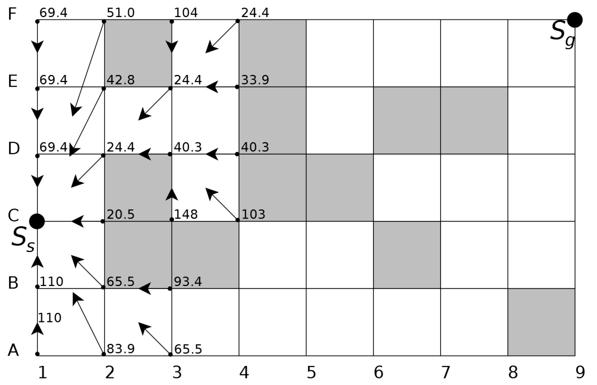


Figure 5: Example trace of Basic Link* with f-values and parent pointers (2)

Figure 4 to 6 illustrate an example trace of Basic Link*. The vertices are labeled with their f-values in degrees. Arrows of these vertices point to their parent vertices. Vertex C1 and vertex F9 are s_s and s_g accordingly. As depicted in Figure 4, we put C1 into closed set C; expand C1 and find D1, D2, C2, B2 and B1; compute their f-values and put them into open set O. Since the f-value of C2, $\Theta(F9, C1, C2), isequalto 20.5^\circ$ that is smaller than other vertices in O, C2 will be expanded next. Because two blocked cells are located on the right side of C2, no new vertices are added into O. We choose D2 as the next vertex being expanded and compute the f-values of E1, E2, E3, D3 and C3. Among these vertices, E3 is visible to C1 so it's f-value is the same as D2; D3 is blocked so it's f-value is $\Theta(F9, C1, D2) + \Theta(F9, D2, D3)$, which is 40.3°. We then expand E3 and F4 and find F4 is blocked by obstacles. As depicted in Figure 5, after a few steps, when the f-value of C4 is higher than B2, we expand C2 instead of C4 as the next vertex. In this way, we skip the path with more

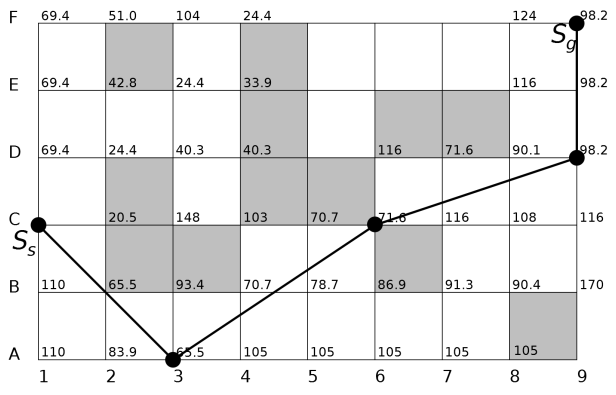


Figure 6: Example trace of Basic Link* with f-values (3)

turns. We keep exploring grids until F9 is expanded and find a path from C1 to F9, as depicted in Figure 6. Please notice that the last edge of the path is parallel to y-axis, which is constrained on one of grid edge directions.

Figure 7 shows a comparison of paths generated by Basic Link* and Theta*. In this example, Basic Link* generates a path with 4 fewer turns by not following the convex boundary at the bottom of the map. A more ideal path with only one turn is shown by the dashed line. Another set of traces is shown in Figure 8, where Basic Link* generates a path with 9 fewer turns by passing through the upper tunnel. This path is smoother but longer than the lower one with zigzag shape.

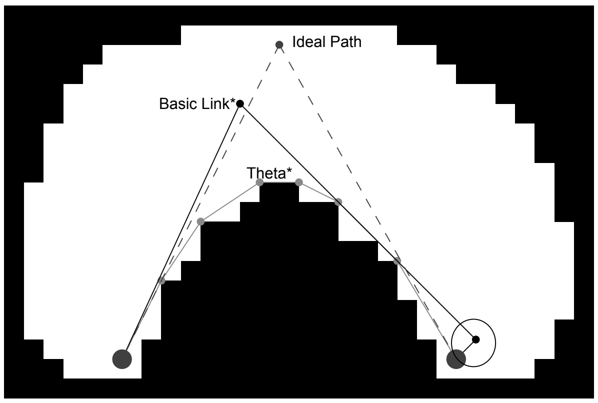


Figure 7: Paths generated by Theta* and Basic Link* compared with an ideal path

Enhanced Link*

Enhanced Link* is introduced to produce paths with more natural turns and shorter length than those in the paths produced by Basic Link*. Sometimes, Basic Link* constrains the direction of turns on the direction of grid edges. For example, in Figure 6 and Figure 7, the last turns of the paths generated by Basic Link* are constrained in multiples of 45°.

Given a goal vertex s_g, the *forward-angle* γ-value of vertex s_k is defined as:

$$\gamma(s_k) = \Theta(s_{k-1}, s_g, s_k).$$

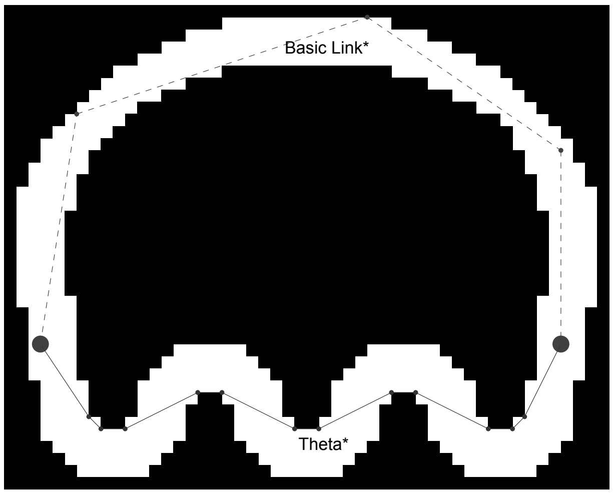


Figure 8: Comparison of paths generated by Theta* and Basic Link* in a map with two tunnels

In Enhanced Link*, the f-value of vertex s_k is defined as:

$$f_E(s_k) = \alpha(s_k) + \gamma(s_k).$$

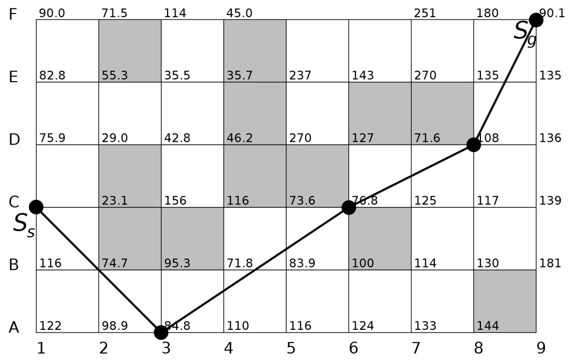


Figure 9: Example trace of Enhanced Link* with f-values

Algorithm 3 describes Enhanced Link*. Each vertex s contains an extra value $\alpha(s)$ to store the α-value, which is initialized to be the same value as $f(s)$.

Let $\alpha'(s_c)$ and $f'_E(s_c)$ denote the α-value and f-value of vertex s_c for the new path to be evaluated between s_s and s_c. When considering Path1, we compute $\alpha'(s_c)$ and $f'_E(s_c)$ for the new path from vertex s_s to $p(s_e)$ and from $p(s_e)$ to s_c. Value $\alpha'(s_c)$ is the sum of $\Theta(s_g, p(s_e), s_c)$ and $\alpha(p(s_e))$ and value $f'_E(s_c)$ is the sum of $\gamma(s_c)$ and $\alpha'(s_c)$. If $f'_E(s_c)$ is smaller than $f_E(s_c)$, Algorithm 3 will update $p(s_c)$, $\alpha(s_c)$ and $f_E(s_c)$ for the new path.

Similarly, when considering Path2, we compute $\alpha'(s_c)$ and $f'_E(s_c)$ for the new path from vertex s_s to s_e and from s_e to s_c. If $f'_E(s_c)$ is smaller than $f_E(s_c)$, Algorithm 3 will update $p(s_c)$, $\alpha(s_c)$ and $f_E(s_c)$.

Figure 9 illustrates an example trace of Enhanced Link* using the same grids and s_s and s_g as Figure 6. Most f-values of Enhanced Link* are higher than those of Basic Link*. For example, we compute the f-value of C2 as

Algorithm 3: Enhanced Link* Algorithm

```
InitializeVertex($s$)
if $s = s_s$ then
    $p(s) \leftarrow s_s$
    $\alpha(s) \leftarrow 0$
    $f_E(s) \leftarrow 0$
else
    $p(s) \leftarrow NULL$
    $\alpha(s) \leftarrow \infty$
    $f_E(s) \leftarrow \infty$
end

ChoosePath1($s_e$, $s_c$)
$\alpha'(s_c) \leftarrow \alpha(p(s_e)) + \Theta(s_g, p(s_e), s_c)$
$\gamma(s_c) \leftarrow \Theta(p(s_e), s_g, s_c)$
$f'_E(s_c) \leftarrow \alpha'(s_c) + \gamma(s_c)$
if $f'_E(s_c) < f_E(s_c)$ then
    $p(s_c) \leftarrow p(s_e)$
    $\alpha(s_c) \leftarrow \alpha'(s_c)$
    $f_E(s_c) \leftarrow f'_E(s_c)$
    if $s_c \notin O$ then
        $O \leftarrow O \cup \{s_c\}$
    end
end

ChoosePath2($s_e$, $s_c$)
$\alpha'(s_c) \leftarrow \alpha(s_e) + \Theta(s_g, s_e, s_c)$
$\gamma(s_c) \leftarrow \Theta(s_e, s_g, s_c)$
$f'_E(s_c) \leftarrow \alpha'(s_c) + \gamma(s_c)$
if $f'_E(s_c) < f_E(s_c)$ then
    $p(s_c) \leftarrow s_e$
    $\alpha(s_c) \leftarrow \alpha'(s_c)$
    $f_E(s_c) \leftarrow f'_E(s_c)$
    if $s_c \notin O$ then
        $O \leftarrow O \cup \{s_c\}$
    end
end
```

$\Theta(F9, C1, C2)$+$\Theta(C1, F9, C2)$, which is 23.1°. By adjusting the f-values, Enhanced Link* finds paths with better turning directions.

Another sample path generated by Enhanced Link* is shown in Figure 10 in comparison with Basic Link*. Note that Enhanced Link* outputs a slightly shorter path and that the direction of the turns in this path is more natural.

Weighted Link*

Weighted Link* differs from Enhanced Link* in that it weights angles. That is, it controls the balance between the number of turns and vertex expansions by applying different weights to backward-angles and forward-angles.

Let $s_s s_1 s_2 \ldots s_k$ be a non-blocked path connecting s_s to s_k. Let c^i be the weight function for the i-th backward-angle, where $1 \leq i \leq k$ and c is a constant number. The α-value of

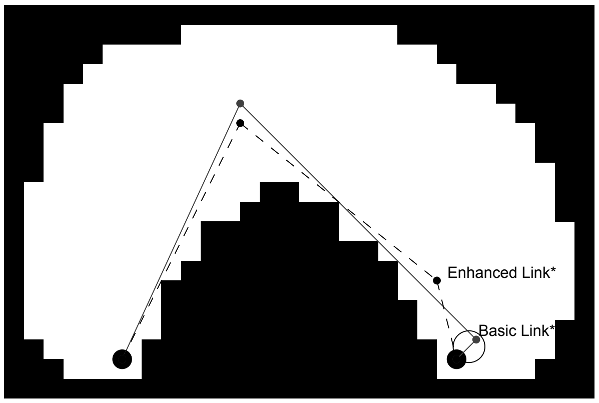


Figure 10: Sample paths of Enhanced Link* and Basic Link*

vertex s_k is defined as:

$$\alpha_W(s_k) = \sum_{i=1}^{k} c^i \cdot \Theta(s_g, s_{i-1}, s_i).$$

The γ-value of vertex s_k is defined as:

$$\gamma_W(s_k) = c^k \cdot \Theta(s_{k-1}, s_g, s_k).$$

The f-value of vertex s_k is defined as follows:

$$f_W(s_k) = \alpha_W(s_k) + \gamma_W(s_k).$$

Note that when $c = 1$, Weighted Link* behaves the same as Enhanced Link*.

Algorithm 4 describes Weighted Link*. Each vertex s contains an extra l-value $l(s)$ to store the number of backward-angles in the path between s_s and s. In procedure InitializeVertex, $l(s_s)$ is initialized as 1 and $l(s)$ of all other vertices is initialized as infinity, which indicates unreachable.

Let $l'(s_c)$, $\alpha'_W(s_c)$ and $f'_W(s_c)$ denote the l-value, α-value and f-value of vertex s_c for the new path to be evaluated between s_s and s_c. When considering Path1, we compute $l'(s_c)$, $\alpha'_W(s_c)$ and $f'_W(s_c)$ for the new path from vertex s_s to $p(s_e)$ and from $p(s_e)$ to s_c. The new path between s_s and s_c has the same number of backward-angles as the path between vertex s_s and vertex s_e. Algorithm 4 copies $l'(s_c)$ from $l(s_e)$. Value $\alpha'_W(s_c)$ is the sum of $c^{l'(s_c)} \cdot \Theta(s_g, p(s_e), s_c)$ and $\alpha_W(p(s_e))$ and value $f'_W(s_c)$ is the sum of $\gamma_W(s_c)$ and $\alpha'_W(s_c)$. If $f'_W(s_c)$ is smaller than $f_W(s_c)$, Algorithm 4 will update $l(s_c)$, $p(s_c)$, $\alpha_W(s_c)$ and $f_W(s_c)$.

Similarly, when considering Path2, we compute $l'(s_c)$, $\alpha'_W(s_c)$ and $f'_W(s_c)$ for the new path from vertex s_s to s_e and from s_e to s_c. The new path between s_s and s_c has one more backward-angle at vertex s_e. The $l'(s_c)$ is assigned by adding 1 to $l(s_e)$. If $f'_W(s_c)$ is smaller than $f_W(s_c)$, Algorithm 4 will update $l(s_c)$, $p(s_c)$, $\alpha_W(s_c)$ and $f_W(s_c)$.

Figure 11 illustrates an example trace of Weighted Link* when $c = 1.2$. By adjusting f-values of Enhanced Link*, Weighted Link* reduces one turn in the final path and still maintains better turning direction.

Algorithm 4: Weighted Link* Algorithm

```
InitializeVertex($s$)
if $s = s_s$ then
    $l(s) \leftarrow 1$
    $p(s) \leftarrow s_s$
    $\alpha_W(s) \leftarrow 0$
    $f_W(s) \leftarrow 0$
else
    $l(s) \leftarrow \infty$
    $p(s) \leftarrow NULL$
    $\alpha_W(s) \leftarrow \infty$
    $f_W(s) \leftarrow \infty$
end

ChoosePath1($s_e$, $s_c$)
$l'(s_c) \leftarrow l(s_e)$
$\alpha'_W(s_c) \leftarrow \alpha_W(p(s_e))$
        $+c^{l'(s_c)} \cdot \Theta(s_g, p(s_e), s_c)$
$\gamma_W(s_c) \leftarrow c^{l'(s_c)} \cdot \Theta(p(s_e), s_g, s_c)$
$f'_W(s_c) \leftarrow \alpha'_W(s_c) + \gamma_W(s_c)$
if $f'_W(s_c) < f_W(s_c)$ then
    $l(s_c) \leftarrow l'(s_c)$
    $p(s_c) \leftarrow p(s_e)$
    $\alpha_W(s_c) \leftarrow \alpha'_W(s_c)$
    $f_W(s_c) \leftarrow f'_W(s_c)$
    if $s_c \notin O$ then
        $O \leftarrow O \cup \{s_c\}$
    end
end

ChoosePath2($s_e$, $s_c$)
$l'(s_c) \leftarrow l(s_e) + 1$
$\alpha'_W(s_c) \leftarrow \alpha_W(s_e)$
        $+c^{l'(s_c)} \cdot \Theta(s_g, s_e, s_c)$
$\gamma_W(s_c) \leftarrow c^{l'(s_c)} \cdot \Theta(s_e, s_g, s_c)$
$f'_W(s_c) \leftarrow \alpha'_W(s_c) + \gamma_W(s_c)$
if $f'_W(s_c) < f_W(s_c)$ then
    $l(s_c) \leftarrow l'(s_c)$
    $p(s_c) \leftarrow s_e$
    $\alpha_W(s_c) \leftarrow \alpha'_W(s_c)$
    $f_W(s_c) \leftarrow f'_W(s_c)$
    if $s_c \notin O$ then
        $O \leftarrow O \cup \{s_c\}$
    end
end
```

Figure 12 illustrates the results of Weighted Link* using two different c-values on the map with two tunnels. When $c = 1.2$, Weighted Link* selects the upper tunnel and generates a path with 4 turns; when $c = 0.8$, it selects the lower tunnel and generates a path with 9 turns. We also include the path generated by Theta* depicted by the gray dashed line for comparison. Weighted Link* finds a path with 4 fewer turns when passing through the lower tunnel.

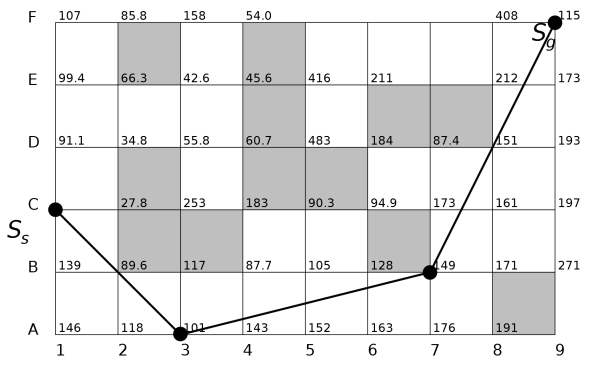


Figure 11: Example trace of Weighted Link* with f-values when $c = 1.2$

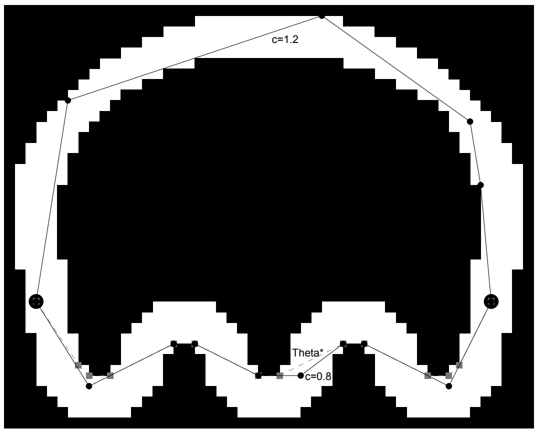


Figure 12: Paths of Weighted Link* under different weights

Correctness and Completeness

All Link* variants use the same framework as Theta* and the proof of the correctness and completeness of Link* is similar to that for Theta* (Daniel et al. 2010). We omit the proof here for brevity.

Theorem 1 *Link*(Basic Link*, Enhanced Link* and Weighted Link*) terminates upon finding a non-blocked path from vertex s_s to vertex s_g or determining that no such path exists.*

Experimental Results

Experiments were performed mainly on four types of grid maps: random maps, video game maps, room maps and maze maps, which are provided by the Pathfinding Benchmarks (Sturtevant 2012). All provided maps were tested, except that the only tested video game maps were from Warcraft III and Baldurs Gate II. All maps are based on 512 by 512 grids. We randomly chose 200 pairs of s_s and s_g vertices from the problem set of each map.

We implemented five planning algorithms, including A*, Theta*, Basic Link*, Enhanced Link* and Weighted Link*. Among these algorithms, A* performs path planning with paths on grid edges, while the others perform any-angle path planning without that constraint. All algorithms are implemented in Java, running on a 2.4 GHz Core i5 2430m laptop with 4GB RAM. Our implementation is not optimized for performance, so further improvements are possible.

Table 1 compares average number of turns, path length and running time of A*, Theta* Basic Link* and Enhanced Link* in four map categories. Since Weighted Link* is identical to Enhanced Link* when the c-value equals to 1.0, we omitted Weighted Link* here. Overall, we found that Basic Link* and Enhanced Link* generated paths with far fewer turns and slightly longer length than those in paths produced by Theta*. On random maps, Basic Link* generated paths with fewer than 60% of the turns and no more than 122% of the path length of Theta*. With respect to the number of turns, Basic Link* performed best on random maps with 10% blocked cells. As depicted in Figure 13, the output of Basic Link* had only 20% of the turns of the output of Theta*. Enhanced Link* usually generated only a few more turns than did Basic Link* but the path length of its output was shorter.

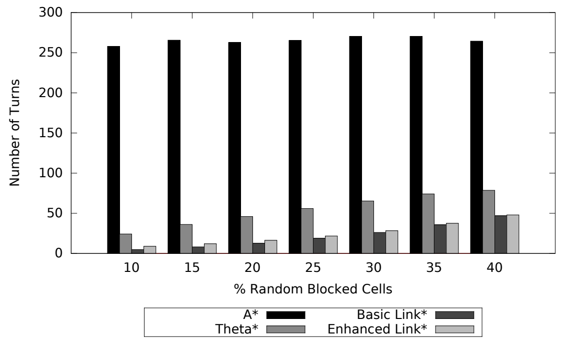


Figure 13: Number of turns in random maps

Given the same map and the same s_s and s_g vertices, Weighted Link* may generate different paths under different c-values. So the path quality of Weighted Link* can be adjusted by the c-value, which can be determined by statistical results of running Weighted Link* with different c-values in a fixed map, or within similar maps. For example, Figure 14 illustrates the number of turns and vertex expansions for Weighted Link* under different c-values on random maps with 10% blocked cells. The x-axis represents c-value and the y-axis represents the number of turns or vertex expansions. As the c-value increased, Weighted Link* generated paths with fewer turns but more vertex expansions. When $c = 1.2$, the number of turns reached a low level but the number of vertex expansions was sub-maximal. So Weighted Link* can choose $c = 1.2$ as the weighing parameter in maps with 10% blocked cells. The number of turns in paths is optimized in this map category.

In Table 1, the running time of Link* is usually longer than that of Theta* primarily for two reasons: (1) the cost of computing angles is more expensive than that of computing distances; (2) the number of vertex expansions is larger in Link* than in Theta*. This is because Link* usually tries to

Map Types	A*	Theta*	Basic Link*	Enhanced Link*
Random10	257.94(312.28, 49.28)	24.28(297.33, 16.70)	4.91(362.58, 215.10)	8.85(332.96, 650.47)
Random15	265.72(322.61, 47.40)	36.13(307.81, 20.59)	8.16(372.06, 362.43)	12.09(350.92, 650.04)
Random20	262.74(318.55, 46.82)	46.04(304.42, 24.38)	12.67(358.65, 399.03)	16.33(344.20, 584.87)
Random25	265.33(321.96, 45.78)	56.00(308.10, 28.67)	18.85(355.02, 368.15)	21.71(345.52, 516.70)
Random30	270.43(324.88, 45.06)	65.37(311.27, 32.09)	26.17(351.59, 314.84)	28.34(345.54, 436.38)
Random35	270.44(323.96, 45.17)	74.17(310.68, 36.76)	35.68(345.28, 257.51)	37.48(340.69, 366.30)
Random40	264.48(308.43, 24.17)	78.64(296.14, 25.72)	46.98(326.41, 112.17)	47.98(324.07, 176.46)
Warcraft III	189.65(223.72, 17.69)	4.42(212.87, 28.26)	2.04(242.97, 54.91)	2.36(225.16, 93.63)
Baldurs Gate II	197.06(236.52, 14.19)	4.75(225.59, 31.83)	2.35(249.38, 51.10)	2.56(236.86, 87.68)
Room8	296.14(346.27, 74.37)	56.64(329.89, 50.42)	41.84(356.56, 235.07)	42.28(353.57, 363.65)
Room16	300.62(352.85, 82.59)	31.81(335.48, 54.00)	25.79(358.77, 216.55)	26.14(354.19, 350.17)
Room32	312.75(367.35, 87.18)	17.02(348.79, 76.96)	14.23(372.35, 217.70)	14.52(365.70, 360.35)
Room64	337.94(396.14, 87.97)	9.30(376.66, 149.73)	7.73(407.10, 267.71)	7.94(396.42, 427.21)
Mazes1	2022.86(2241.79, 78.78)	770.99(2185.93, 96.74)	713.42(2466.49, 243.88)	713.36(2465.24, 461.43)
Mazes2	1635.01(1822.90, 82.57)	421.24(1754.13, 108.65)	374.25(2022.10, 259.56)	374.37(2019.15, 491.80)
Mazes4	1574.07(1767.66, 94.52)	239.89(1698.03, 150.78)	208.33(1994.31, 315.18)	209.45(1986.51, 593.30)
Mazes8	1624.90(1827.41, 108.31)	139.80(1758.73, 230.72)	118.27(2103.09, 406.25)	118.44(2090.22, 735.36)
Mazes16	1459.99(1644.80, 120.66)	67.84(1588.44, 370.05)	57.34(1911.47, 543.35)	57.26(1886.55, 905.11)
Mazes32	1051.73(1189.38, 133.38)	26.45(1150.13, 604.61)	21.24(1394.50, 782.88)	21.45(1356.50, 1149.88)

Table 1: Comparison of average number of turns, path length, and running time (ms) (path length and running time are in parenthesis)

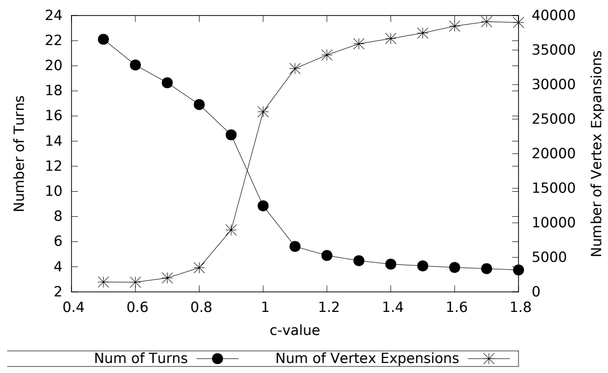


Figure 14: Number of turns and vertex expansions in Weighted Link* with different weights on random maps with 10% blocked cells

explore multiple passages and finds more straight ones than Theta*. According to our test, the running time of Link* is close to that of Theta* on maps of applications such as robot path planning and video games. Among the three variants of Link*, Basic Link* is faster than either Enhanced Link* or Weighted Link* because it makes only one angle calculation in determining its f-values.

Conclusion and Future Work

In the paper, we present Link* as a solution to the problem of finding fewer turns in any-angle path planning. Link* encompasses three related algorithms: Basic Link*, Enhanced Link* and Weighted Link*. Basic Link* is the simplest. It has better performance as measured by number of turns and running time. Enhanced Link* delivers superior path quality and shorter length than Basic Link*. Weighted Link* provides the flexibility to adjust the balance between the number of turns and the number of vertex expansions. Link* finds paths with fewer than 60% of the turns at the cost of no more than 122% of the path length of those found by Theta* on random maps. Link* thus offers a big advantage when turns in paths are expensive. Further research can be focused on extending Link* by considering vertex re-expansion and incremental heuristic search. Paths generated by Link* can also be smoothed with arcs for trajectory planning in robot motion planning.

Acknowledgements

The authors would like to thank International Technological University (ITU) for its financial support on AI Research Lab and this research. We would also like to thank Dr. Greg O'Brien, Dr. Steve Miller and Dr. Richard Riehle for their editing and comments.

References

Arkin, E. M.; Mitchell, J. S.; and Suri, S. 1992. Optimal link path queries in a simple polygon. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, 269–279. Society for Industrial and Applied Mathematics.

Bresenham, J. E. 1965. Algorithm for computer control of a digital plotter. *IBM Systems journal* 4(1):25–30.

Choi, S., and Yu, W. 2011. Any-angle path planning on non-uniform costmaps. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 5615–5621. IEEE.

Choset, H.; Lynch, K. M.; Hutchinson, S.; Kantor, G.; Burgard, W.; Kavraki, L. E.; and Thrun, S. 2005. *Principles of robot motion: theory, algorithms, and implementations*. MIT press.

Daniel, K.; Nash, A.; Koenig, S.; and Felner, A. 2010. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research* 39(1):533–579.

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

Mitchell, J. S.; Rote, G.; and Woeginger, G. 1990. Minimum-link paths among obstacles in the plane. In *Symposium on Computational Geometry*, 63–72.

Mitchell, J. S.; Rote, G.; and Woeginger, G. 1992. Minimum-link paths among obstacles in the plane. *Algorithmica* 8(1):431–459.

Nash, A.; Daniel, K.; Koenig, S.; and Felner, A. 2007. Theta*: Any-angle path planning on grids. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 22, 1177–1183.

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

Suri, S. 1986. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing* 35(1):99–110.

Yap, P. 2002. Grid-based path-finding. In *Advances in Artificial Intelligence*. Springer. 44–55.