The Practicality of ANYA: An Any-Angle Pathfinding Algorithm

Michael McMahon | mwr.mcmahon@outlook.com

Abstract

In domains such as video games, autonomous agents must be able to move around the world towards goals such as the player, or objects in the world; agents must be able to do so in a manner that is realistic (i.e. paths reflect intelligent behaviour). In order to find such a path, pathfinding algorithms must be used; both the optimality and runtime of pathfinding is dependant on the algorithm chosen. This paper describes Theta* and ANYA, two any-angle pathfinding algorithms, and uses Theta* as a benchmark to determine the practicality of using ANYA in the domain of 2D video games. Using an online pathfinding algorithm which uses less memory, computes faster, and gives shorter paths than other algorithms would greatly benefit games such as Total War: Shogun II, where many agents must find paths in large environments.

Table of Contents

Introduction: Any-Angle Pathfinding

Pathfinding is the process of finding a path from one coordinate (the start) to another (the goal) in a world [Graham et al., 2003]. A path is a series of points between the start and goal points which an agent (i.e. NPC: non-playable character) follows to reach its destination [Graham et al., 2003]. $P = \{v_1, v_2, v_3, \dots, v_k\}$; where P is a path, v_i is a point along this path, and k is the number of points in the path. There are many different algorithms used to find such paths (called pathfinding algorithms), and factors such as optimality (a path is said to be optimal if it is the shortest possible) and runtime differ depending on the algorithm used.

Before a pathfinding algorithm can be used, the continuous world in which the agents operate must be represented in a discrete form. Since this paper is concerned with pathfinding in video games, all algorithms will be discussed on the basis that the world in which they operate is a 2D grid: a discretization of a continuous 2D world "map" into atomic cells; each cell is either "free" (traversable by an agent) or "blocked" (not traversable by an agent) [Graham et al., 2003]. For this paper, all grids will be regular square grids: all cells are equally sized squares. Each square cell contains four vertices (one at each corner) connected by edges (line segments). Each vertex in the grid map has up to eight neighbouring vertices.

A path is said to "turn" whenever the direction of the path changes; more specifically: if acute the angle formed by $v_{i-1}, v_i, v_{i+1} \neq 180°$; where v_i is the point along the path currently being considered. A common approach in the video game industry is to use A* on a square grid map to create a path which is "grid constrained": agents can only travel along edges formed between two neighbouring cells. This means that the path only turns at $90°$ or $45°$; such paths are said to be "unsmooth". Unsmooth paths are considered unrealistic because they do not reflect intelligent behaviour: the agent turns in places where travelling in a straight line would be ideal. In order to "smooth-out" the paths produced by A*, it is common practice to use "string pulling": a post processing technique which modifies the path so that it only turns at the corners of blocked cells; the path can turn at any angle. However, string pulling not only requires additional computation on top of the path finding algorithm itself, but also produces a non-optimal path as a result (i.e., the path is no longer the shortest possible) [Harabor and Grastien, 2013]. In response to the downsides of post-smoothing an unsmooth path, "any-angle" pathfinding algorithms will be discussed as an alternative to string pulling.

Preliminaries: Theta*

Theta* is a popular any-angle pathfinding algorithm which is essentially A* with smoothing "mixed in": path smoothing is done entirely online (i.e. no pre/post-computation required). In A*, a node's parent must be one of it's (maximum) eight neighbours; however, in Theta*, a node's parent can by any other node which is mutually visible. Two nodes are said to be mutually visible if the line segment connecting them does not pass through any blocked cells, or a vertex shared by two blocked cells. Figure 1 illustrates the expansion of nodes in Theta*: starting at figure 1 (a), then progressing to figure 1 (d), where:

- The column along the right side of the sub-figures and the row along the top side of the sub-figures will be used to identify nodes (think battleship)
- The node currently being expanded is circled in red
- The arrow of each node points to that node's parent
- Nodes with blue arrows are newly discovered neighbours of the node currently being expanded
- The number next to each node indicate that node's g-value

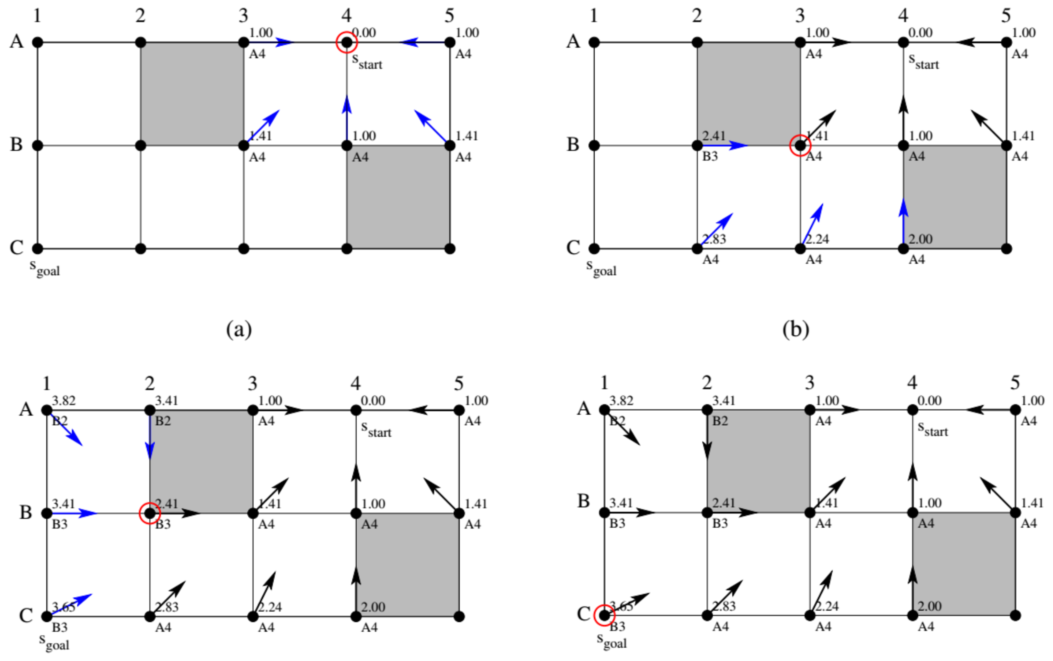

Figure 1: an example execution of Theta*, progressing from (a) to (d) [Daniel et al.,

This algorithm will be illustrated by example using figure 1:

(a) Starting at A4: A3, A5, B3, B4, B5 are visible from A4 → their parent becomes A4

(b) Proceeding to B3 (from A4): B2 is visible from B3 → B2's parent becomes B3. C2, C3, and C4 are visible from B3's parent (A4) → their parent becomes A4 (**not** B3!)

(c) Proceeding to B2 (from B3): A1 and A2 are visible from B2 → their parent becomes B2. B1 and C1 are visible from B2's parent (B3) → their parent becomes B3 (**not** B2!)

(d) Proceeding to C1 (from B2): C1 has no **newly** discovered neighbours → no need update anything

However, because Theta* essentially uses string pulling, it inherits a big disadvantage of string pulling: the path is not optimal. Additionally, the g-value between a parent node and all of it's successors may not increase by a consistent amount; it is necessary for g-values to increase monotonically (between a parent node and all of it's successors) to ensure that an optimal solution is found [Harabor and Grastien, 2013]. As a result, Theta* can't guarantee an optimal solution.

Preliminaries: ANYA

Principle of ANYA

Unlike Theta*, whose path only contains turning points, ANYA's path contains turning points, and points between these turning points, which will be referred to as "intermediate points". An intermediate point is the intersection between the line segment connecting two turning points, and a row of the grid map. Consider figure 2; suppose n_1 and n_2 are turning points in a path, and $y_1 \dots y_3$ are intermediate points. The path produced by Theta* would only store n_1 and n_2, but the path of ANYA would store n_1, y_1, y_2, y_3, and n_2. This means that unlike Theta*, the g-value of each parent node increases by a consistent amount with respect to each of it's successors. As a result, ANYA always produces the optimal path.

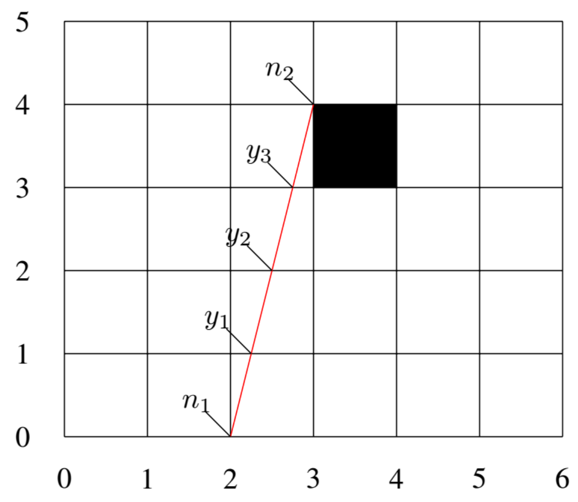

Figure 2: Points in Theta*'s path vs points in ANYA's path [Harabor and Grastien, 2013]

Generating Nodes

In ANYA, rather that each node corresponding to a vertex on the grid map, each node corresponds to a tuple, (I, r); where:

- I is a line segment formed between two mutually visible vertices along a row of the grid map
- r is the "root" of the node (i.e. a turning point in the path), $r \notin I$

[Harabor and Grastien, 2013]

Every point along I is visible to r. So, if (I, r) is the node currently being expanded, it's successor is the node (I', r'); where:

- If every point in the successor interval is visible from r, then $r' = r$ (this is what results in a smooth path), **OR**
- If every point in the successor interval is not visible from r, but is visible from some other point, b (which is an endpoint of I), then $r' = b$
- In all cases, $r' \in \{r\} \cup I$ [Uras and Koenig, 2015]

[Harabor and Grastien, 2013]

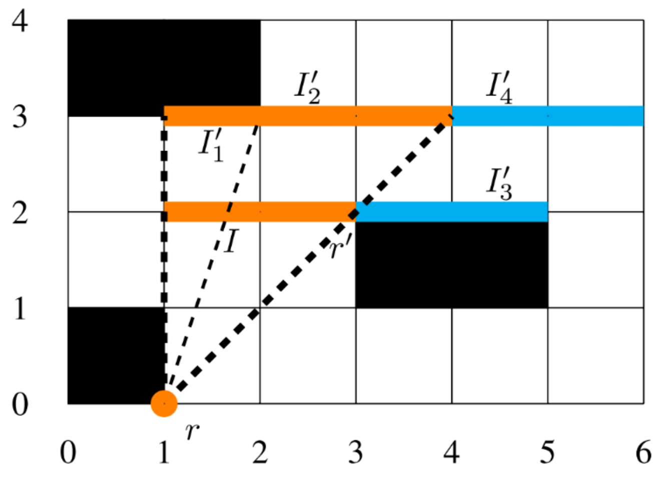

Figure 3: The node currently being expanded, (I, r), it's observable successors (orange), and it's non-observable successors (blue) [Harabor and Grastien, 2013]

An interval "splits" (is partitioned into two smaller intervals) whenever it's visibility w.r.t. r changes. Consider row 2 in figure 3: $I = [(1,2),(3,2)]$ rather than $[(0,2),(6,2)]$. In this case, I splits just to the right of (3, 2) to create the successor I'_3, since none of the points contained in I'_3 are visible from r. An interval also splits whenever it contains a corner vertex of a blocked cell; this is why I'_1 splits at (2, 3) to create I'_2.

Finding a Node's Representative Point

Once a node has been generated, a point $p \in I$ must be chosen to represent this node (called a "representative point"). Each intermediate point in figure 2 is the representative point of some node. Each $p \in I$ has a three-part heuristic value (lower is better) which estimates the cost from p to the goal, t.The representitivbe point of a node is the one with the lowest f-value; luckily, it is not necessary to check every $p \in I$.

For each point on an interval ($\forall p \in I$), it's (3-part) heurisitic, $f(p)$, is defined as:

$f(p) = g(r) + d(r,p) + h(p,t)$; where:

- $g(r)$ is the length of the optimal path from the start point to the root
- $d(r,p)$ is the direct distance between the root and the point being considered;

  $d(p_1, p_2) = \sqrt{\left(x_{p2} - x_{p1}\right)^2 + \left(y_{p2} - y_{p1}\right)^2}$
- $h(p,t)$ is an admissible heuristic between the point being considered, and the goal point, t. In this case, $h(p,t) = d(p,t)$

[Harabor and Grastien, 2013]

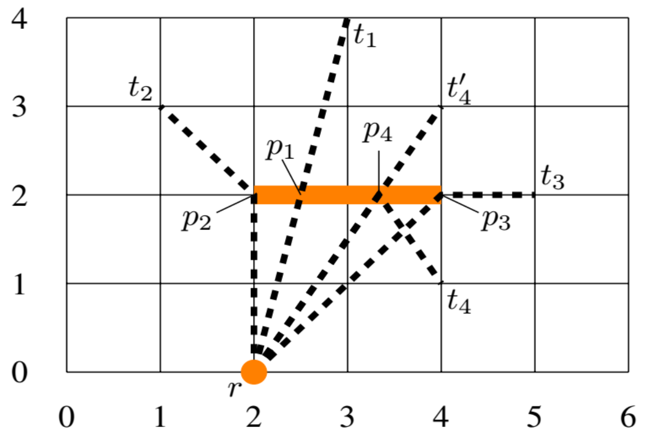

Figure 4: The four cases in which $p \in I$ with the minimum f-value can be found [Harabor and Grastien, 2013]

As seen in the figure 4, there are four possible cases for choosing a representative point:

1. t is above I, and the line segment between r and t intersects I. In this case, p is the intersection point of this line segment and I
2. t is above I, and the line segment between r and t doesn't intersect I. In this case, p is the endpoint of I that is closest to t
3. t is on the same row as I ($t \notin I$). In this case, p is the endpoint of I that is closest to t
4. t is below I: "mirror"/reflect t along the axis formed by I; reflecting does not change $d(p,t)$. Once reflected, t is above I: now we have either case 1 or 2 on our hands.

Steps of the Algorithm

Now that the processes of generating and processing nodes has bee n covered, the following is a summary of the ANYA algorithm:

1. On the first iteration of the algorithm, I only contains the start vertex; r is located off of the map, and is only visible to I; the cost from r to I is zero. The goal vertex, t, has been found if the interval of the node currently being expanded contains t (i.e. $t \in I$)

2. Find the representative point of the node currently being expanded; the represented point of the first node is the start vertex
3. Generate the successor nodes of the node currently being expanded, and repeat step #2 for each successor node (nodes are typically stored using a priority queue, such as the ANYA implementation in [Uras and Koenig, 2015])
4. Repeat steps #2-3 until the interval of the node currently being expanded contains the goal vertex (i.e. $t \in I$)

The path is extracted by following parent node pointers from the goal until the start is reached. The path is the series of all representative points; the path turns at root points.

Discussion: ANYA vs Other Algorithms

As mentioned earlier, unlike Theta*, ANYA always produces an optimal path, and does so online, without any preprocessing or memory overheads [Harabor and Grastien, 2013]. However, this does not speak of the speed of the algorithm. In [Uras and Koenig, 2015], several different pathfinding algorithms were implemented and tested for sub-optimality (based on path length), as well as runtime; because the path length and runtime can vary based on the implementation used, the implementation for each algorithm was made as similar as possible; more specifically:

- All algorithms used the same implementation of a priority queue using a binary heap
- All algorithms break ties towards larger g-values
- All algorithms used Euclidean distance (i.e. direct distance) as a heuristic between a node and the goal

[Uras and Koenig, 2015]

Figure 5 demonstrates that ANYA outperformed all other algorithms in terms of optimality. Although ANYA performed quite well on game maps (left), it was the slowest algorithm on random maps (right). This may be due to the way ANYA generates nodes: each node corresponds to one interval, and not every interval is long. This is because an interval always splits whenever it's visibility w.r.t. r changes, or I contains a corner vertex of a blocked cell.

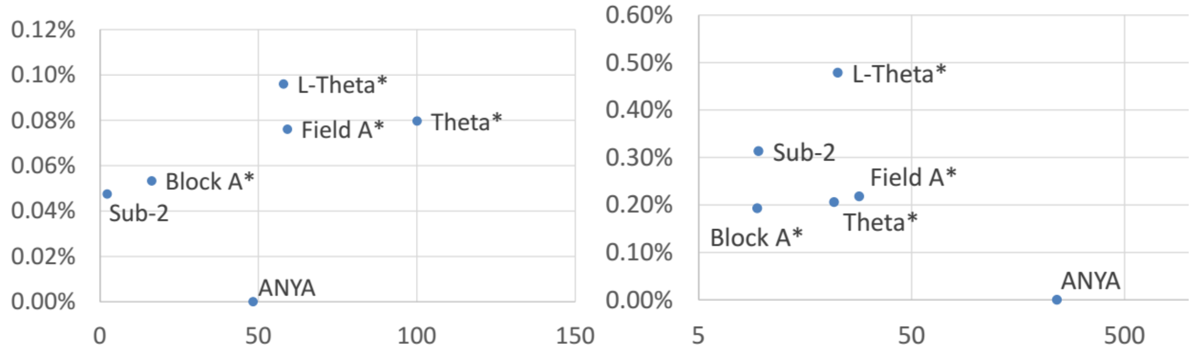

Figure 5: sub-optimality defined by path length (y-axis) vs runtime in ms (x-axis) of pathfinding algorithms on game maps (left), and random maps (right) [Uras and Koenig, 2015]

As a consequence of these facts, ANYA generates a new node each time an interval is split; this means that the more intervals there are, the more computations that need to be done: each node needs to find it's representative point, and generate its successors. In the worst possible case, the number of nodes per row is equal to the number of cells in that row [Harabor and Grastien, 2013]. On the other hand, ANYA performs better than most other pathfinding algorithms in large, open spaces: in these cases, intervals rarely need to be split, allowing for nodes to contain long intervals [Uras and Koenig, 2015]. To summarize, ANYA can cover more space faster than (most) other algorithms in fairly open grids, but performs slower than other algorithms in cluttered girds. Theta* on the other hand, only needs to process up to eight neighbours (maximum) per node. As a result, Theta* performs more consistently than ANYA: Theta*'s difference in runtime (~70ms) is less than that of ANYA (~350ms).

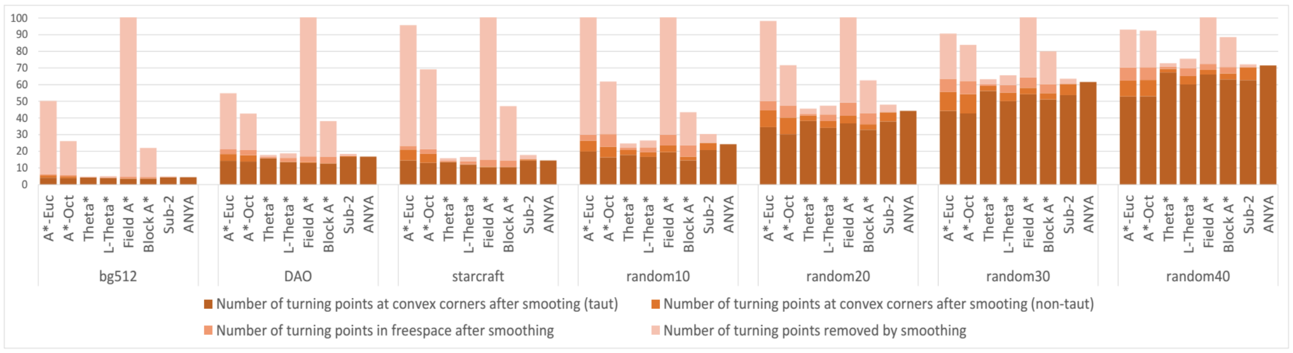

Figure 6: Histogram of the number of turning points removed after post-smoothing paths. The number in the names of the random maps indicates the percentage of blocked cells [Uras and Koenig, 2015]

Figure 6 shows that ANYA doesn't always have the least number of turning points; this means that the resulting path will look less realistic from an efficiency perspective (i.e. the agent is changing direction more than it needs to). This appears to be a small negative consequence when weighed against the benefits associated with its optimality: ANYA is the **only** any-angle pathfinding algorithm above that produces an optimal path [Uras and Koenig, 2015]. Furthermore, ANYA does not benefit at all from post-smoothing; this is evident by the singly-coloured columns associated with ANYA in Figure 6. As a result, post-smoothing can be omitted entirely from ANYA, further reducing the number of computations that need to be run, thereby reducing runtime.

Results & Conclusion

This paper has both described how both Theta* and ANYA work, and has used Theta* as a reference point to determine the practicality of using ANYA in the context of 2D video games. Unlike the other any-angle pathfinding algorithms discussed, ANYA was the only algorithm to consistently produce optimal paths. However, this optimality comes with the consequence of inconsistent performance: ANYA is faster than most other algorithms on maps with large, open spaces, but performs slower than other algorithms in cramped, cluttered maps (e.g. Dragon Age: Origin maps contain many tight corridors) [Uras and Koenig, 2015]. These results suggest that ANYA would be ideal in situations where very large, open maps need to be explored quickly. An example of such a situation would Total War: Shogun II: a strategy game consisting of very large, open maps where many agents (both player and computer) must find paths.

References

[Daniel et al., 2010] Daniel, K., Nash, A., Koenig, S., & Felner, A. (2010). Theta*: Any-Angle Path Planning on Grids. *Journal of Artificial Intelligence Research, 39*, 533-579.

[Graham et al., 2003] Graham, R., McCabe, H., & Sheridan, S. (2003). Pathfinding in Computer Games. *The IBT Journal, 4*(2), 57-81.

[Harabor and Grastien, 2013] Harabor, D., & Grastien, A. (2013). An Optimal Any-Angle Pathfinding Algorithm. *Conference on Automated Planning and Scheduling*, (pp. 308-311). Rome, Italy.

[Uras and Koenig, 2015] Uras, T., & Koenig, S. (2015). An Empirical Comparison of Any-Angle Path-Planning Algorithms. *Eighth Annual Symposium on Combinatorial Search.* Ein Gedi, Israel: Association for the Advancement of Artificial Intelligence.