

## A\* Search

Note: before reading this article you should understand the concepts presented in the tutorial on BFS and DFS.

A\* is a best first, informed graph search algorithm. A\* is different from other best first search algorithms in that it uses a heuristic function  $h(x)$  as well as the path cost to the node  $g(x)$  in computing the cost  $f(x) = h(x) + g(x)$  for the node. The information comes into play with  $h(x)$  which we will come back to later in the tutorial. The A\* algorithm is presented below in Figure 1.0

```
Initialize queue to an empty priority queue (min queue)
Initialize closed to be an empty set
Insert the start state into the queue
While (queue is not empty)
    node ← Dequeue an element off queue
    If (node is a goal state) //Solution is found
        If (node ∉ closed)
            Add node to closed
            Add all successors of node to queue
```

A\* search algorithm : Figure 1.0 <sup>[1]</sup>

For implementation details I suggest [1] and [2].

The heuristic function  $h(x)$  is used to tell A\* an *estimate* of the minimum cost from a node to a goal. The heuristic can also control the behavior of A\*.

Note:  $C^*$  is the minimum cost of moving from a node to a goal

$h(x) = C^*$       A\* will always follow the shortest path from the node to a goal. As a bonus the search will never expand any nodes that are not on the shortest path making the algorithm as fast as possible.

$h(x) < C^*$       As in the above case A\* will find the shortest path. However the search will expand more nodes than necessary to find the shortest path.

$h(x) > C^*$       A\* is not guaranteed to find the shortest path. However the algorithm is much faster than the previous case.

$h(x) \gg g(n)$     A\* turns into a Best First Search

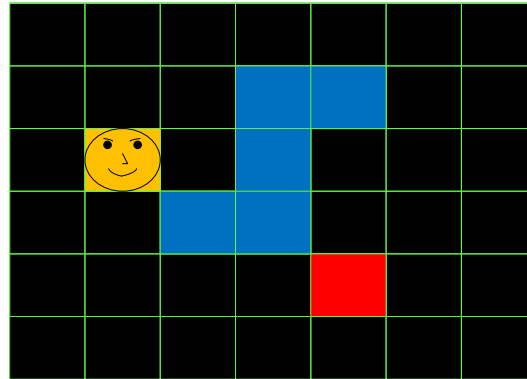
$h(x) = 0$         A\* turns into Uniform Cost Search

[1] Russell S., Norvig P. Artificial Intelligence – a modern approach (2ed, PH, 2003) pg. 83 Figure 3.19

[2] Wikipedia article on A\* Search [http://en.wikipedia.org/wiki/A-star\\_search\\_algorithm](http://en.wikipedia.org/wiki/A-star_search_algorithm)

Before we start exploring heuristics and an example of the A\* search let's check what the complexities are for both memory and time. The time complexity of the worst case is  $O(b^m)$  where  $m$  is the depth of the search tree. The memory complexity is what gives A\* problems. In the worst case memory requirements are exponential which is not good. Usually a good heuristic can cut down the memory used as well as the time required. For more details on runtime complexity see [1] and [2] for more details.

Now we will explore various heuristics and their use in a grid world show below in Figure 1.1



Grid World : Figure 1.1

In the grid world walls are blue the goal square is red and the orange square is where our friend Smiley is currently at. Our job is to get Smiley from the orange square to the red square in as few steps as possible. Smiley can move in his world by going Up, Down, Left, or Right but not diagonally.

Before we get too involved an interesting question is, given any heuristic will we be able to find the minimum path? Terminology for this is called *admissibility*. For our heuristic to be admissible the heuristic must always **never overestimate** the cost to reach the goal. For a more in depth coverage of heuristics and other properties such as *consistency* and the *triangle-inequality* consult [1] pgs 97 – 101.

Now we explore the Manhattan distance and Euclidean distance heuristics for A\* on our grid world.

### Manhattan Distance

$$h(n) = |(node.x - goal.x)| + |(node.y - goal.y)|$$

Manhattan distance is admissible because Smiley must always go at least  $abs(node.x - goal.x)$  distance horizontally and  $abs(node.y - goal.y)$  distance vertically to get to the red square. If there are walls, then  $h(n)$  is at least the minimum cost if no walls get in the way of Smiley and if walls do get in his way then the cost of Smiley's path is  $h(n) + d$ , where  $d > 0$  and still  $h(n)$  never overestimates the true cost of getting to the red square.

### Euclidean Distance

- [1] Russell S., Norvig P. Artificial Intelligence – a modern approach (2ed, PH, 2003) pg. 83 Figure 3.19  
 [2] Wikipedia article on A\* Search [http://en.wikipedia.org/wiki/A-star\\_search\\_algorithm](http://en.wikipedia.org/wiki/A-star_search_algorithm)

$$h(n) = \sqrt{(node.x - goal.x)^2 + (node.y - goal.y)^2}$$

Euclidean distance is admissible because the fastest Smiley could get to the red square is to go in a straight line and Euclidean distance is the distance of the straight line. If there are walls, then  $h(n)$  is at least the minimum cost if no walls get in the way of Smiley and if walls do get in his way then the cost of Smiley's path is  $h(n) + d$ , where  $d > 0$  and still  $h(n)$  never overestimates the true cost of getting to the red square.

Now that we know either Manhattan or Euclidean distance will get us the optimal path we can go ahead and safely compute a path for Smiley to get to the red square. For ease of calculation (so the reader can easily follow along) I am choosing Manhattan distance as my heuristic.

For the following steps I will use a blue outline to show the square currently being analyzed and I will show the  $f(x)$ ,  $g(x)$ , and  $h(x)$  values as in Figure 1.2

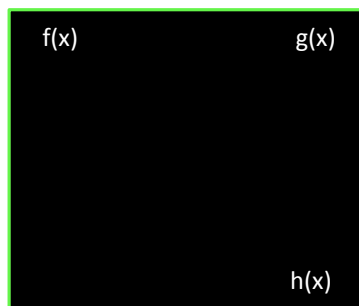


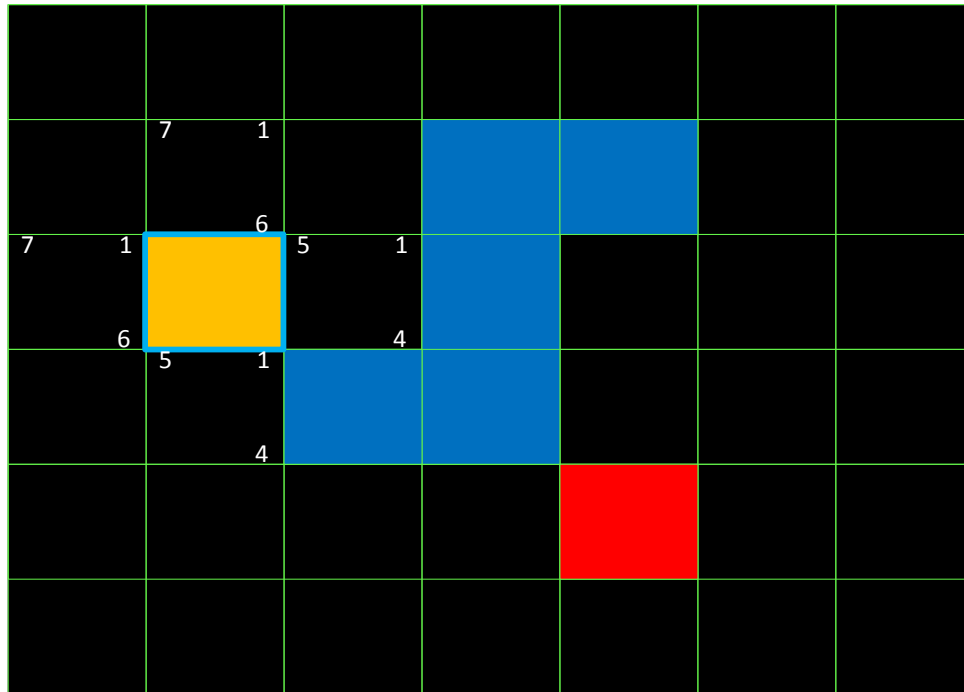
Figure 1.2

### Initial Setup

Initially the orange square is put onto the priority queue with  $f(x) = h(x)$ ,  $g(x) = 0$ , and  $h(x) = 5$

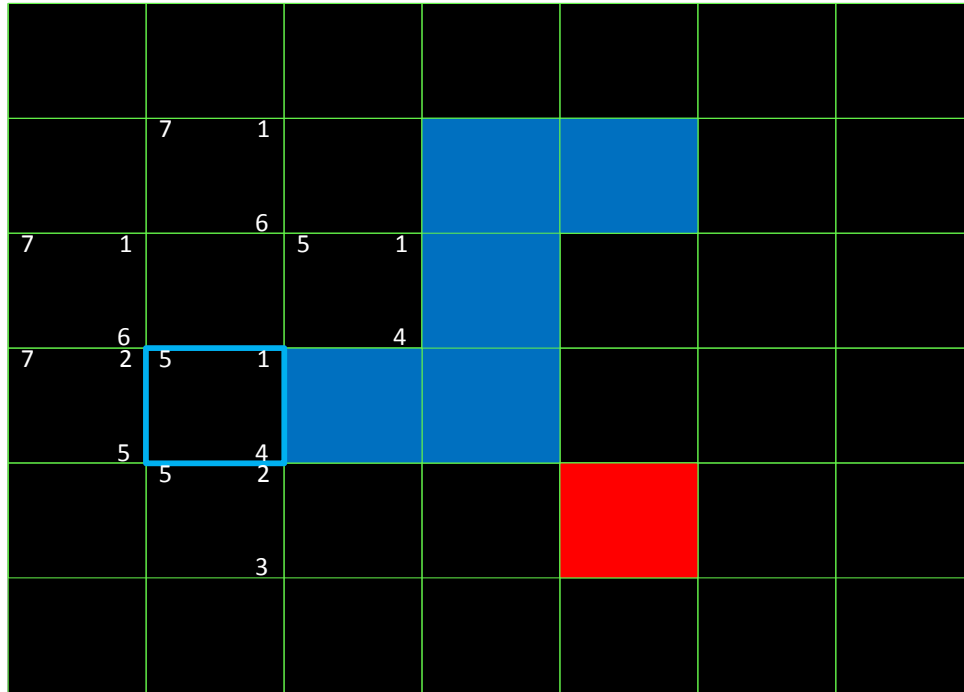
### First Step

Dequeue the orange square and enqueue the successors of the orange square in the order left, up, right, and down (the order was chosen to make the example short). The successors'  $f(x)$ ,  $g(x)$ , and  $h(x)$  values are shown below.



## Second Step

Dequeue the down square (there is a tie between it and the right square) and push on its' successors (same order as the above step). The successors'  $f(x)$ ,  $g(x)$ , and  $h(x)$  values are shown below.



Notice that the square we came from did not get pushed onto the queue, why? Because our heuristic is admissible we know that once a node is in the closed list it will never need to be popped off the priority queue.

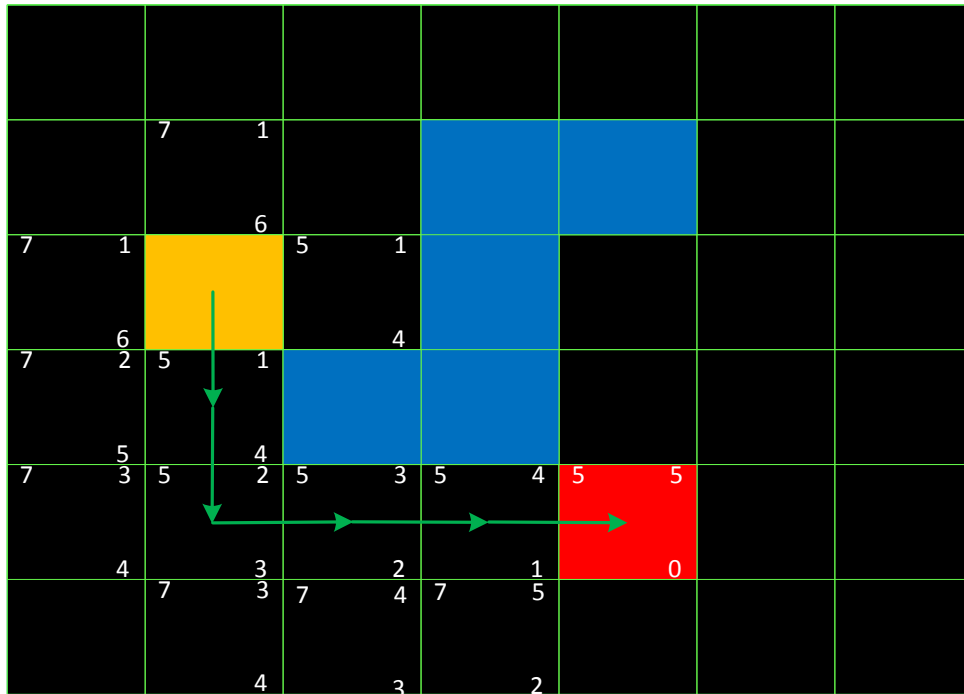








At this point A\* is done and the minimum cost path has been found. However, there is one looming question, how do we get the shortest path in a form that is usable? The most common approach is to store a parent field in the node class and then have a procedure that builds the path given a goal node. If we re-run A\* with parents on the example above we get the following picture.



Where the green arrows were computed using the parent fields

A\* is a flexible and powerful search algorithm that uses information to achieve results at least as good as Best First Search or Uniform Cost Search. By changing the heuristic used A\* can obtain optimal results with a slow runtime or sub-optimal results with a faster runtime. The process of creating good heuristics is very creative and yet analysis is needed to prove admissibility and consistency.

[1] Russell S., Norvig P. Artificial Intelligence – a modern approach (2ed, PH, 2003) pg. 83 Figure 3.19  
 [2] Wikipedia article on A\* Search [http://en.wikipedia.org/wiki/A-star\\_search\\_algorithm](http://en.wikipedia.org/wiki/A-star_search_algorithm)