A* Algorithm

The first difference between wave-front is that A* needs to discretize the world.

The A* algorithm searches a graph efficiently, with respect to a chosen heuristic. A* will produce an optimal path if its heuristic is optimistic.

An optimistic, or admissible, heuristic always returns a value less than or equal to the cost of the shortest path from the current node to the goal node within the graph.

The A* search has a priority queue which contains a list of nodes sorted by priority, which is determined by the sum of the distance traveled in the graph thus far from the start node, and heuristic.

The first node to be put into the priority queue is naturally the start node. Next, we expand the start node by popping the start node and putting all adjacent nodes to the start node into the priority queue sorted by their corresponding priorities. Note that only unvisited nodes are added to the priority queue.

After one path to the goal with a cost is found, the nodes in the priority queue which have a priority value (its heuristic value to goal plus the edge cost from start) greater than the cost to the goal can be discarded.

The explicit path through the graph is represented by a series of back pointers. A back pointer represents the immediate history of the expansion process.

For the nodes that their priority values are less than the found path, they are still the candidates.

We will use two additional data structures, an open set O and a closed set C. The open set O is the priority queue and the closed set C contains all processed nodes. Other notation includes

- Star(n) represents the set of nodes which are adjacent to n.
- $c(n_1, n_2)$ is the length of edge connecting n_1 and n_2.
- $g(n)$ is the total length of a backpointer path from n to q_{start}.
- $h(n)$ is the heuristic cost function, which returns the estimated cost of shortest path from n to q_{goal}.
- $f(n) = g(n) + h(n)$ is the estimated cost of shortest path from q_{start} to q_{goal} via n.

The algorithm can be found in algorithm 24.

H.2.2 Discussion: Completeness, Efficiency, and Optimality

Here is an informal proof of completeness for A^*. A^* generates a search tree, which by definition, has no cycles. Furthermore, there are a finite number of acyclic paths in the tree, assuming a bounded world. Since A^* uses a tree, it only considers acyclic paths. Since the number of acyclic paths is finite, the most work that can be done,

Algorithm 24 A^* Algorithm

Input: A graph

Output: A path between start and goal nodes

1: **repeat**
2: Pick n_{best} from O such that $f(n_{best}) \leq f(n), \forall n \in O$.
3: Remove n_{best} from O and add to C.
4: If $n_{best} = q_{goal}$, EXIT.
5: Expand n_{best}: for all $x \in$ Star(n_{best}) that are not in C.
6: **if** $x \notin O$ **then**
7: add x to O.
8: **else if** $g(n_{best}) + c(n_{best}, x) < g(x)$ **then**
9: update x's backpointer to point to n_{best}
10: **end if**
11: **until** O is empty