A-star algorithm

A* Search Algorithm - GeeksforGeeks
A* search algorithm - Wikipedia

> **Tldr**
>
> 一个带有启发式指引（greedy）的两点间最短路的搜索的Dijkstra算法。

背景

- 一种图遍历的路径搜索算法；
- 给定一个带权图，不是处理单个源节点到所有节点的最短路问题，是处理单个源节点到某一个确定的目标节点的**最短路**；

  > Compared to Dijkstra's algorithm, the A* algorithm only finds the shortest path **from a specified source to a specified goal**, and not the shortest-path tree from a specified source to all possible goals.

- 时间复杂度O(|E|log|V|)，空间复杂度O(|V|);
- 可以视为Dijkstra算法的拓展；其使用了一个启发式函数指导其搜索；

  > This is a necessary trade-off for using a specific-goal-directed heuristic.

- 虽然每次只选择当前的最小成本路径，不过该greedy策略最终可以得到一个全局最优的结果；

算法内容

https://en.wikipedia.org/wiki/File:Astarpathfinding.gif

- It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until the goal node is reached.
- 每次主循环，算法需要挑选一条最小成本的路径进行extend；代价函数分为两部分，路径的cost和把路径延伸到目标的估计cost。

$$f(n) = g(n) + h(n)$$

- n 是路径上的下一个节点，g(n) 是从起始节点到 n 的路径成本，h(n) 是一个启发式函数，用于估计从 n 到目标的最小成本路径的成本（the cost of the cheapest path from n to the goal）；

  > We define ‘**g**’ and ‘**h**’ as simply as possible below
  > **g** = the **movement cost** to move from the starting point to a given square on the grid, following the path generated to get there.
  > **h** = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of ***smart guess***.

实现

- A* 的典型实现使用优先队列（作为frontier）来执行最小（估计）成本节点的选择以进行扩展。
- 在算法的每一步，具有最低 f(x) 值的节点从队列中pop，其邻居的 f 和 g 值相应更新，并将这些邻居添加到队列中。
- 终止条件：pop的节点恰好为目标节点。
- 被访问过的节点，都存储了对其父节点的引用，因此从目标节点倒推回去就能得到最短路；
- $h(x)$ 可以被设置为是物理距离，比如L1距离，L2距离；

优先队列的实现通常是斐波那契堆，复杂度为O(log V);

伪代码

```
function reconstruct_path(cameFrom, current)
    total_path := {current}
    while current in cameFrom.Keys:
        current := cameFrom[current]
        total_path.prepend(current)
    return total_path

// A* finds a path from start to goal.
// h is the heuristic function. h(n) estimates the cost to reach goal from node n.
function A_Star(start, goal, h)
    // The set of discovered nodes that may need to be (re-)expanded.
    // Initially, only the start node is known.
    // This is usually implemented as a min-heap or priority queue rather than a hash-set.
    openSet := {start}

    // For node n, cameFrom[n] is the node immediately preceding it on the cheapest path from the start
    // to n currently known.
    cameFrom := an empty map

    // For node n, gScore[n] is the cost of the cheapest path from start to n currently known.
    gScore := map with default value of Infinity
    gScore[start] := 0

    // For node n, fScore[n] := gScore[n] + h(n). fScore[n] represents our current best guess as to
    // how cheap a path could be from start to finish if it goes through n.
    fScore := map with default value of Infinity
    fScore[start] := h(start)

    while openSet is not empty
        // This operation can occur in O(Log(N)) time if openSet is a min-heap or a priority queue
        current := the node in openSet having the lowest fScore[] value
        if current = goal
            return reconstruct_path(cameFrom, current)

        openSet.Remove(current)
        for each neighbor of current
            // d(current,neighbor) is the weight of the edge from current to neighbor
            // tentative_gScore is the distance from start to the neighbor through current
            tentative_gScore := gScore[current] + d(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                // This path to neighbor is better than any previous one. Record it!
                cameFrom[neighbor] := current
                gScore[neighbor] := tentative_gScore
                fScore[neighbor] := tentative_gScore + h(neighbor)
                if neighbor not in openSet
                    openSet.add(neighbor)

    // Open set is empty but goal was never reached
    return failure
```

一个很清晰的演示：https://upload.wikimedia.org/wikipedia/commons/9/98/AstarExampleEn.gif

难点

- 如何获取有效的启发式函数？path的cost是非常直接就能得到，但是到目标的距离是一个并不显然的估计值；
- 一个错误的、过于简单的启发式函数，可能会造成误导的效果，比如在目标点之前有一堵墙，那一开始直接朝着目标点探索并不是最优的路径；
- 核心：对于目标距离的估计（或者是对未来reward的期望，对于正确得到答案的概率，anyway随便怎么样都是同一套）如何基于当前的state进行估计？回归到经典的RL的问题，对于state-value/action-value的估计；

- 对于抽象概念的距离，譬如“语义”，“数学推理的正确性”，这种距离如何设计？如何朝着有效的方向进发？如何设计模型进行近似计算？