人工智能 - 搜索

杜小勤

武汉纺织大学数学与计算机学院

2016/02/25

从 BFS 到 $A^\star$ 搜索

- 宽度优先搜索（Breadth First Search）；
- 迪杰斯特拉算法（Dijkstra's Algorithm）；
- 最佳优先搜索（Best First Search）；
- $A^\star$ 搜索；

宽度优先搜索（BFS）

宽度优先搜索：本质上，是一种 Level-ordering 遍历 Graph 中节点的方式，像水波那样均匀地、一层一层地向外传播。

播放 GIF 动画：
<BFS-Animation.gif>
<BFS-Contour-Lines.gif>

BFS Python code

```
frontier = Queue()
frontier.put(start)
visited = {}
visited[start] = True

while not frontier.empty():
    current = frontier.get()
    for next in graph.neighbors(current):
        if next not in visited:
            frontier.put(next)
            visited[next] = True
```

图 1-1: 宽度优先搜索的 Python 代码

BFS Python code

frontier 相当于 Open 表，visited 相当于 Closed 表。

相当于 BFS 的非递归（Open-Closed 表）实现方式。

保存路径

```
frontier = Queue()
frontier.put(start)
came_from = {}
came_from[start] = None

while not frontier.empty():
    current = frontier.get()
    for next in graph.neighbors(current):
        if next not in came_from:
            frontier.put(next)
            came_from[next] = current
```

图 1-2: 增加了保存路径的功能

程序运行结果图

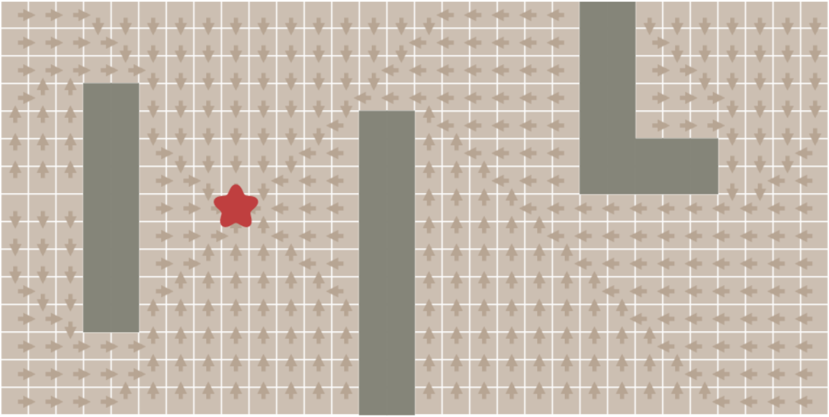

图 1-3: 路径指示图

路径重构

```
current = goal
path = [current]
while current != start:
    current = came_from[current]
    path.append(current)
path.reverse()
```

图 1-4: 路径重构

考虑 Early Exit

Early Exit：找到一条路径后退出。

方法：在循环体中增加如下语句

```
if current == goal:
    break
```

< 播放 GIF 动画：BFS-EarlyExit.gif>

路径步数 V.S. 路径代价

BFS：所有的路径代价相同，相当于只考虑路径步数；

迪杰斯特拉（Dijkstra）算法：路径有不同的代价；

< 播放 GIF 动画：BFS-Dijkstra.gif>

Dijkstra 算法

算法需要修改：

1. 追踪每个节点的代价（从 start 节点到该节点）；
2. 将 Queue 修改成 Priority Queue：始终优先访问代价最低的节点：从代价低的节点逐渐扩展到代价高的节点，与 BFS 一样，也是层层外扩；
3. 一个节点可能访问多次：如果一个已访问节点的新代价更低，则需重新访问；

Dijkstra 算法

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] + graph.cost(current, next)
      if next not in cost_so_far or new_cost < cost_so_far[next]:
         cost_so_far[next] = new_cost
         priority = new_cost
         frontier.put(next, priority)
         came_from[next] = current
```

图 1-5: Dijkstra 算法

BFS 与 Dijkstra 算法：例子

< 播放 GIF 动画：BFSDijkstra.gif>

如何更快地找到一个目标?

将 BFS 和 Dijkstra 算法用于搜寻许多目标是合适的。

但是，假如只需要搜索一个目标，是否存在更快的算法?

BFS 和 Dijkstra 算法都不能快速地找到目标，因为它们都是“盲目地”层层外扩，并没有使用能够快速导向目标的“启发信息”。

启发函数

启发函数：提供当前节点接近目标程度的启发信息。

```
def heuristic(a, b):
    # Manhattan distance on a square grid
    return abs(a.x - b.x) + abs(a.y - b.y)
```

图 1-6: 启发信息——Manhattan 距离

贪心最佳优先搜索

贪心最佳优先搜索（Greedy Best First Search，GBFS）使用了启发信息——当前节点到目标的（估算）距离。

算法

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
came_from[start] = None

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      if next not in came_from:
         priority = heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

图 1-7: 贪心最佳优先搜索

从优先级队列的角度比较：Dijkstra、GBFS

两者的队列都是优先级队列：按代价从小到大排序，代价小的优先搜索；

Dijkstra 中的代价信息是：start 节点到当前节点的代价；

GBFS 中的代价信息是：当前节点到 goal 节点的（估算）代价——优先扩展代价小的节点，利于快速导向目标；

无启发信息 V.S. 启发信息

< 播放 GIF 动画：GBFS-Heuristic-Search1.gif>

一个不成功的例子

< 播放 GIF 动画：GBFS-Heuristic-Search2.gif>

当存在障碍时，GBFS 没有能够找到最短路径。
如何解决？

$A^\star$ 算法

Dijkstra 算法使用了 start 节点到当前节点的实际代价信息，能够找到最短路径，但不能快速地导向目标；

贪心最佳优先搜索算法使用了当前节点到 goal 节点的估算代价信息，能够快速地导向目标，但不一定能够找到最短路径；

$A^\star$ 算法将两者的优点结合到了一起：代价信息 = 实际代价 + 估算代价；

算法

```
frontier = PriorityQueue()
frontier.put(start, 0)
came_from = {}
cost_so_far = {}
came_from[start] = None
cost_so_far[start] = 0

while not frontier.empty():
   current = frontier.get()

   if current == goal:
      break

   for next in graph.neighbors(current):
      new_cost = cost_so_far[current] + graph.cost(current, next)
      if next not in cost_so_far or new_cost < cost_so_far[next]:
         cost_so_far[next] = new_cost
         priority = new_cost + heuristic(goal, next)
         frontier.put(next, priority)
         came_from[next] = current
```

图 1-8: $A^\star$ 算法

三个算法的执行情况对比

< 播放 GIF 动画：ThreeAlgorithms.gif>

限制条件——可接受的启发信息

$A^\star$ 算法能够找到最短路径——

只要当前节点到 goal 节点的启发信息（估算值）不超过当前节点到 goal 节点的实际代价值。

该启发信息被称为是可接受的启发信息（Admissible Heuristic）。

评估函数

$f(n) = g(n) + h(n)$，n 是当前节点；

$f(n)$：评估函数；

$g(n)$：从 start 节点到当前节点 n 的代价（已知）；

$h(n)$：从当前节点 n 到 goal 节点的代价（估算）；

$A^\star$ 算法的启发函数

设 $h^\star(n)$ 是从当前节点 n 到 goal 节点的实际代价，那么当条件 $\forall n, h(n) \le h^\star(n)$ 满足时，称为该启发函数是可接受的，此时 $A^\star$ 算法一定可以找到最短路径，否则，算法将可能漏掉最短路径。

一个启发函数 h_1 优于另一个启发函数 h_2，如果前者扩展的节点数少于后者扩展的节点数。

8 数码问题中的启发函数

- 海明距离（Hamming distance）：当前状态与目标状态相比，不相符的数字格子的个数；
- 曼哈顿距离（Manhattan distance）：$h(n) = \sum_{all-tiles} distance(tile, correct - position)$；

这 2 个启发函数都是“可接受的”，因为从当前状态移动到目标状态的实际代价都要高于估计代价；

例子-目标状态

1	2	3
8		4
7	6	5

图 1-9: 8 数码问题的目标状态

启发函数的计算

2	8	3
1	6	4
	7	5

2	8	3
1		4
7	6	5

2	8	3
1	6	4
7	5	

图 1-10: 三个状态的启发函数：(a)、$h_1 = 5, h_2 = 6$；(b)、$h_1 = 3, h_2 = 4$；(c)、$h_1 = 5, h_2 = 6$；

单调一致的启发信息

如果 $\forall n, h(n) \le c(n,p) + h(p)$，并且 $h(g) = 0$，那么称启发信息 $h(n)$ 是单调一致的（Consistent Heuristic）。

此处，n 是任意节点，p 是 n 的任意后继节点，g 是任意目标节点，$c(n,p)$ 是节点 n 到节点 p 的代价。

“单调一致”一定是“可接受的”

可以证明，单调一致的启发信息一定是可接受的。

归纳法：
依据“单调一致的启发信息”的定义，可以假设 $h(N_m) \leq h^{\star}(N_m)$ 成立，此处，$h^{\star}(N_m)$ 表示节点 N_m 到目标节点的最短路径代价（可以从目标节点的父节点开始，归纳假设）。

“单调一致”一定是“可接受的”

那么可以推出：
$h(N_{m+1}) \leq c(N_{m+1}, N_m) + h(N_m) \leq$
$c(N_{m+1}, N_m) + h^{\star}(N_m) = h^{\star}(N_{m+1})$

但是，“可接受的”不一定是“单调一致的”。

将“可接受的”转换成“单调一致的”

利用 pathmax 公式可以将可接受的启发信息转换成单调一致的启发信息：

$$h'(p) \leftarrow max(h(p), h(n) - c(n, p))$$

单调一致性启发函数的性质

最短路径上的$f(N_j)$ 序列是单调非递减的，其中 N_j 是节点，并且：

$$f(N_j) = g(N_j) + h(N_j),$$
$$g(N_j) = \sum_{i=2}^{j} c(N_{i-1}, N_i)$$

$f(N_j)$ 为什么是单调非递减的？

设 N_i、N_{i+1} 是最短路径上连续的二个节点，那么依据启发信息的单调一致性有：$h(N_i) \le c(N_i, N_{i+1}) + h(N_{i+1})$。

可以推出：

$f(N_{i+1}) = g(N_{i+1}) + h(N_{i+1}) = g(N_i) + c(N_i, N_{i+1}) + h(N_{i+1}) \ge g(N_i) + h(N_i) = f(N_i)$。

单调一致性启发函数的意义

$A^{\star}$ 搜索中，如果使用单调一致性的启发函数，一旦一个节点被扩展，那么它的代价已经是最低的了，意味着该节点不会再被扩展。

而在一个使用可接受的但非单调一致性的启发函数的 $A^{\star}$ 搜索中，一个节点可能需要扩展多次，只要搜索到了一个更好代价的路径。

特定条件下的启发信息

如果单调一致性条件满足：

$$h(N_i) = c(N_i, N_{i+1}) + h(N_{i+1})$$

并且，$c(N_i, N_{i+1})$ 具有非负值，

那么，最短路径上的 $h(N_j)$ 序列是单调非递增的。

算法小结

- 宽度优先搜索与 Dijkstra 算法：如果要找到一个节点到所有其它节点的最短路径。如果移动代价相同，使用前者，否则使用后者；
- $A^\star$ 算法：如果只需要找到一个节点到另一个节点的最短路径。

参考文献

[1] Introduction to $A^\star$
[2] Wikipedia: Admissible Heuristic
[3] Wikipedia: Consistent Heuristic
[4] Web Page: State Space Representation and Search