人工智能 - 搜索

杜小勤

武汉纺织大学数学与计算机学院

2016/02/20

搜索

- 深度优先搜索
- 宽度优先搜索
- $A^{\star}$ 搜索
- $\alpha - \beta$ 搜索

概述

早期的人工智能（AI）重视搜索算法的研究。例如，深度优先搜索（DFS）、宽度优先搜索（BFS）、minmax 搜索、$\alpha-\beta$ 搜索、$A^{\star}$ 算法等。

因为许多 AI 任务可以通过定义状态空间并通过搜索技术来得到有效地解决。

状态空间

问题被表达成状态空间（State Space），然后再使用各种搜索技术在状态空间中进行搜索，以得到问题的解答。

本质上，状态空间由节点和边构成。节点表示问题的每一个状态，边表示一个节点到另一个节点的合法移动。状态空间也定义了一个初始状态和一个终止状态。

状态空间

状态空间可以采取图（包括网格 Grid）和树的形态。

几个经典的问题

旅行商（Traveling Salesman）问题

汉诺塔（Towers of Hanoi）问题

8 数码（8-Puzzle）问题

Tic Tac Toe 游戏

汉诺塔问题

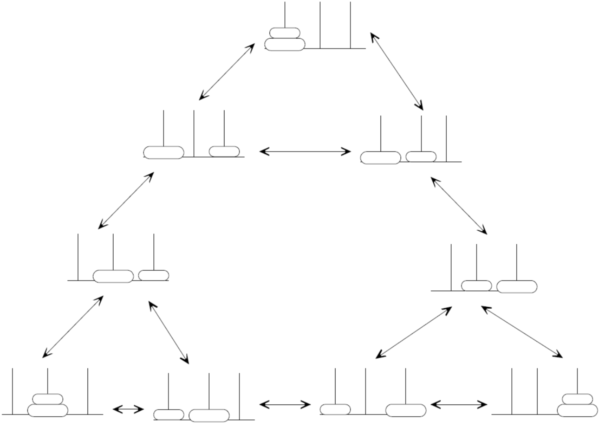

图 1-1: 汉诺塔的状态空间

8 数码问题

图 1-2: 8 数码问题:(a) 初始状态；(b) 目标状态

8 数码问题

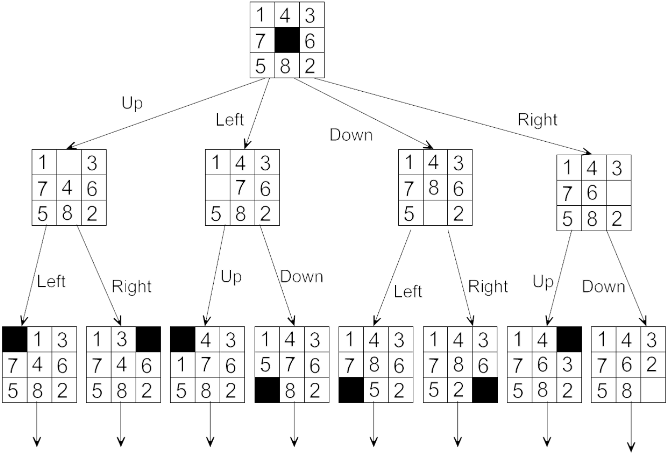

图 1-3: 8 数码的状态空间

Tic Tac Toe 游戏

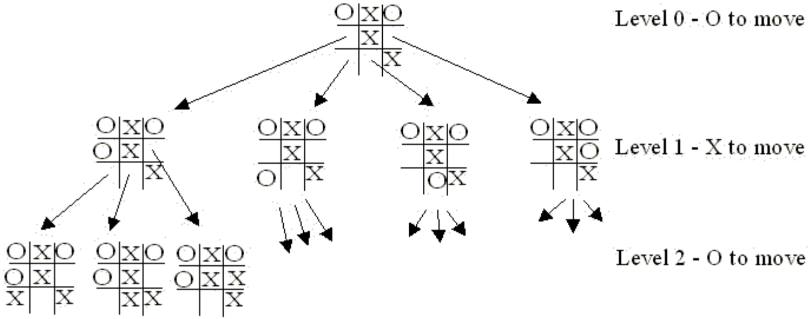

图 1-4: Tic Tac Toe 的状态空间

合法动作集

节点在进行状态转移生成子节点时，都要依据合法动作集。

例如，8 数码问题的合法动作集是：LEFT、RIGHT、UP、DOWN。

状态空间的分支因子越大，合法动作集也越大。有些问题的分支因子非常大，例如围棋。

一般遍历方式（DFS）

使用深度优先搜索对图或树中的节点进行排序：

- Preordering：以节点首次被访问的次序进行排序；
- Postordering：以节点最后被访问的次序进行排序；
- Reverse Postordering：Postordering 的反序；

二叉树遍历方式（DFS）

- Preordering;
- Inordering;
- Postordering;

还有一种是 Level-ordering，它实际上是宽度优先搜索。

Level-ordering: 宽度优先搜索

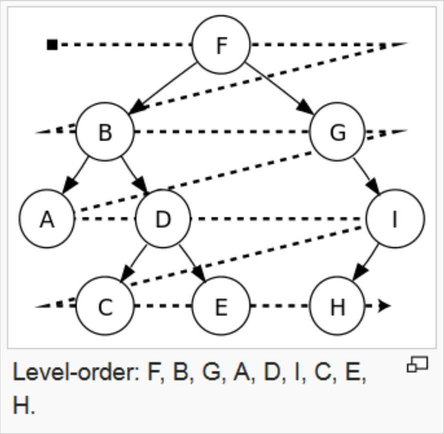

图 1-5: Level-ordering: 宽度优先搜索

算法伪代码：递归实现

Input: A graph G and a vertex v of G
Output: All vertices reachable from v labeled as discovered

```
procedure DFS(G, v):
    label v as discovered
    for all edges from v to w in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G, w)
```

图 1-6: 深度优先搜索的递归实现

算法伪代码：非递归实现（Stack）

```
procedure DFS-iterative(G,v):
    let S be a stack
    S.push(v)
    while S is not empty
        v = S.pop()
        if v is not labeled as discovered:
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

图 1-7: 深度优先搜索的非递归实现（Stack）

示例 Graph

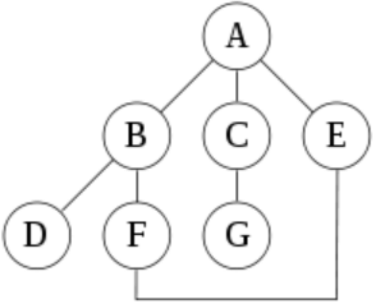

图 1-8: 示例 Graph

递归实现：A, B, D, F, E, C, G;

非递归实现：A, E, F, B, D, C, G;

算法伪代码：非递归实现（Open、Closed 表）

```
def dfs (in Start, out State)
 open = [Start];
 closed = [];
 State = failure;
 while (open <> []) AND (State <> success)
   begin
    remove the leftmost state from open, call it X;
    if X is the goal, then
      State = success
    else begin
      generate children of X;
      put X on closed
      eliminate the children of X on open or closed
      put remaining children on left end of open
    end else
 endwhile
 return State;
enddef
```

图 1-9: 深度优先搜索的非递归实现（Open、Closed 表）

一个例子

Graph/Tree

A to B and C
B to D and E
C to F and G
D to I and J
I to K and L

Start state: A
Goal State: E, J

执行过程

	OPEN	CLOSED	X	X's Children	State
1.	A	-			failure
2.			A	B, C	failure
3.	B, C	A			failure
4.	C	A	B	D, E	failure
5.	D, E, C	A, B			failure
6.	E, C	A, B	D	I, J	failure
7.	I, J, E, C	A, B, D			failure
8.	J, E, C	A, B, D	I	K, L	failure
9.	K, L, J, E, C	A, B, D, I			failure
10.	L, J, E, C	A, B, D, I,	K	none	failure
11.	J, E, C	A, B, D, I	L	none	failure
12.	E, C	A, B, D, I	J		success

图 1-10: 例子的执行过程

迭代加深的深度优先搜索

Depth First Search with Iterative Deepening, DFID

每次迭代有不同的深度上限值（深度值加 1），该算法可以确保找到最短路径。

算法伪代码

```
procedure DFID (intial_state, goal_states)
begin
        search_depth=1
        while (solution path is not found)
        begin
                dfs(initial_state, goals_states) with a depth bound of search_depth
                increment search_depth by 1
        endwhile
end
```

图 1-11: 迭代加深的深度优先搜索算法

执行过程

Iteration 1: search_depth =1

	OPEN	CLOSED	X	X's Children	State
1.	A	-			failure
2.			A	B, C	failure
3.	B, C	A			failure
4.	C	A	B		failure
5.	C	A, B			failure
6.		A, B	C		failure
7.		A, B, C			failure

图 1-12: 例子执行过程（DFID）- 深度 1

执行过程

Iteration 2: search_depth = 2

	OPEN	CLOSED	X	X's Children	State
1.	A	-			failure
2.			A	B, C	failure
3.	B, C	A			failure
4.	C	A	B	D, E	failure
5.	D, E, C	A, B			failure
6.	E, C	A, B	D		failure
7.	E, C	A, B, D			failure
8.	C	A, B, D	E		success

图 1-13: 例子执行过程（DFID）- 深度 2

DFS 应用

- Finding connected components
- Finding strongly connected components
- Topological sorting
- Generating words in order to plot the Limit Set of a Group
- Planarity testing
- Solving puzzles such as mazes
- Maze generation may use a randomized depth-first search

演示

Maze generation

播放视频文件 MAZE_30x20_DFS.ogv

宽度优先搜索

Breadth First Search，BFS

从一个节点开始，由近到远的方式逐层进行搜索，能够保证找到最短路径。

宽度优先搜索 - 示例

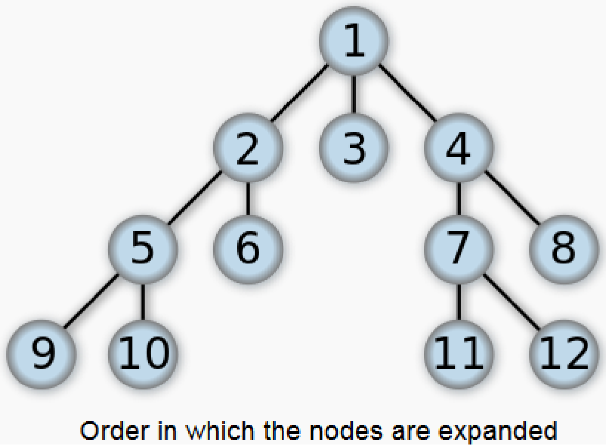

图 1-14: 宽度优先搜索算法的示例

非递归实现（Queue）

Input: A graph Graph and a starting vertex root of G
Output: All vertices reachable from root labeled as explored

非递归实现（Queue）

```
Breadth-First-Search(Graph, root):

    for each node n in Graph:
        n.distance = INFINITY
        n.parent = NIL

    create empty queue Q

    root.distance = 0
    Q.enqueue(root)

    while Q is not empty:

        current = Q.dequeue()

        for each node n that is adjacent to current:
            if n.distance == INFINITY:
                n.distance = current.distance + 1
                n.parent = current
                Q.enqueue(n)
```

图 1-15: 宽度优先搜索的非递归实现（Queue）

非递归实现（Open、Closed 表）

```
def bfs (in Start, out State)
 open = [Start];
 closed = [];
 State = failure;
 while (open <> []) AND (State <> success)
   begin
    remove the leftmost state from open, call it X;
    if X is the goal, then
      State = success
    else begin
      generate children of X;
      put X on closed
      eliminate the children of X on open or closed
      put remaining children on right end of open
    end else
 endwhile
 return State;
enddef
```

图 1-16: 宽度优先搜索的非递归实现（Open、Closed 表）

示例

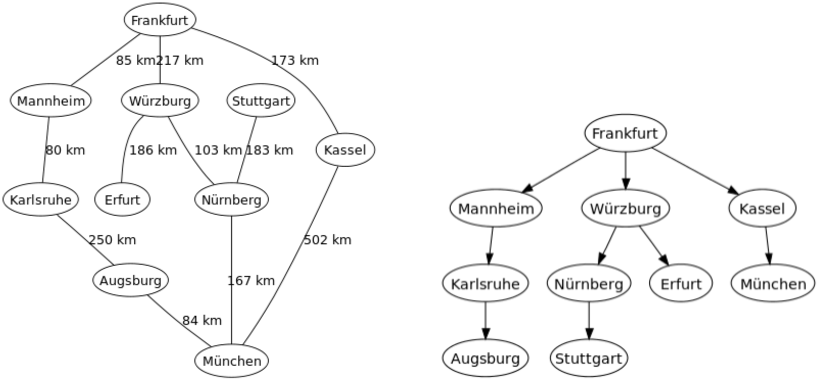

图 1-17: 示例:(a)Germany 城市 Graph；(b)Germany 城市 Tree

DFS v.s. BFS

- BFS 能够保证找到最短路径；
- DFS 在搜索时可能会“迷路”，必须施加一个搜索深度限制；
- BFS 可能会碰到“组合爆炸”问题，当分支因子很高时；
- DFS 搜索到的结果可能不是最优的；
- DFS 可能更有效率，当分支因子很高时；

参考文献

[1] Mark Watson. Practical Artificial Intelligence Programming in Java, 2005.
[2] Web Page: State Space Representation and Search
[3] Wikipedia: Depth First Search
[4] Wikipedia: Tree Traversal
[5] Wikipedia: Bread First Search