搜索算法综述

1951976　李林飞

日期：2021 年 3 月 27 日

摘　要

搜索在现实生活中无处不在，它是解决许多问题的最基本的方法。而搜索算法则是为了寻找一种或多种解的序列的过程，其输入的是问题，输出的是问题的解，以行动序列的方式返回问题的解。为便于现实问题的讨论，我们通常将问题形式化，去除那些并不重要的信息，只留下问题的本质。本文对目前人工智能领域中三大类搜索算法进行探讨与分析，即盲目搜索、启发式搜索、局部搜索。着重分析各类中几个经典算法的实现思想与四大性质：完备性、最优性、时间复杂度、空间复杂度，并采用伪代码的形式给出算法实现的具体过程。

关键词： 盲目搜索, 启发式搜索, 局部搜索, 性能分析

目录

1 基础知识

搜索算法需要一个数据结构来记录搜索树的构造过程。对树中的每个结点，我们定义的数据结构包含四个元素：

$n.STATE$: 对应状态空间的状态；

$n.PARENT$: 搜索树中产生该结点的结点（父结点）；

$n.ACTION$: 父结点生成该结点所采取的行动；

$n.PATH-COST$: 代价，一般用 $g(n)$ 表示，指从初始状态到达该结点的路径消耗。

除此之外，我们一般采用队列作为算法实现的数据结构，现定义几个基本的操作:

$EMPTY?(queue)$: 返回值为真当且仅当队列中没有元素；

$POP(queue)$ 返回队列中的第一个元素并将它从队列中删除;

$INSERT(element, queue)$: 在队列中插入一个元素并返回结果队列。

注意，栈只是一种被限制的特殊队列。

而对于一个算法，我们通常从四个方面进行讨论，即

完备性: 当问题有解时，是否保证可以找到解？

最优性: 是否可以找到最优解？

时间复杂度: 找到解花费的时间；

空间复杂度: 搜索过程中需要多少内存？

接下来，我将对三大类搜索算法中的一些经典算法谈谈自己的理解。

2 符号说明

表 1: 文中用到的符号定义

符号	意义
$\|V\|$	图的顶点数
$\|E\|$	图的边数
b	分支因子，即任意节点的最多后继数
d	目标结点所在的最浅的深度
m	状态空间中任何路径的最大长度
$f(n)$	评价函数
$h(n)$	启发函数 (结点 n 到目标结点的最小代价路径的代价估计值)
$g(n)$	路径消耗：从开始结点到结点 n 的路径代价

3 盲目搜索

3.1 深度优先搜索 (DFS)

DFS 算法的基本思想

首先，鉴于深度优先搜索的过程中需要回溯，故一般用 LIFO 队列；其次，需要创建一个 $visited[]$ 数组，用于记录所有被访问过的顶点。然后执行以下步骤：

（1）从图中 v_0 出发，访问 v_0；

（2）找出 v_0 的第一个未被访问的邻接点，访问该顶点。以该顶点为新的顶点，重复此步骤，直至刚访问过的顶点没有未被访问的邻接点为止；

（3）返回前一个访问过的访问过的仍有未被访问邻接点的顶点，继续访问该顶点的下一个未被访问临界点；

（4）重复（2）、（3）步骤，直至所有顶点均被访问。

DFS 算法伪代码

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

图 1: DFS 算法伪代码 [1]

DFS 算法分析

完备性：DFS 算法如果是在避免重复状态和冗余路径的图搜索，则在有限状态空间是完备的，因为它之多扩展所有结点。但对于树搜索，是不完备的，因为由于一个有解的问题树可能含有无穷分枝，深度优先搜索如果误入无穷分枝（即深度无限），则不可能找到目标节点。

最优性：无论是图搜索还是树搜索，DFS 都不是最优的。因为 DFS 算法只要找到一种存在的解就会返回，但并不能保证该解是最优的。

时间复杂度:：在简单的算法分析中，如果使用邻接矩阵存储图结构，则使用 DFS 遍历的时间复杂度为 $O(|V|^2)$, 若使用邻接表，则为 $O(|V|+|E|)$。更普遍的分析中，DFS 的时间复杂度取决于状态空间的规模。对于树搜索，其最坏情况是目标结点在某最长路径上的末端，此时其时间复杂度为 $O(b^m)$，其中 m 可能会远远大于 d, 也可能是无限的。

空间复杂度：DFS 在搜索过程中使用 LIFO 队列存储一条从根结点到叶结点的路径，以及该路径上每个结点的所有未被扩展的兄弟结点。在最坏情况下，路径的最大长度为 m，在该路径上每个结点未扩展的兄弟结点数最多为 b, 故其空间复杂度为 $O(bm)$。

3.2 广度优先搜索 (BFS)

BFS 算法的基本思想

广度优先搜索一般使用 FIFO 队列作为数据结构。其算法流程为：

（1）需创建一个 $visited[]$ 数组，用来记录已被访问过的顶点；创建一个队列，用来存放每一层的顶点；

（2）从图中的 v_0 开始访问，将 $visited[v_0]$ 设置为 $true$，表示已访问，同时将 v_0 入队列；

（3）判断队列是否为空，若空，则结束；否则，重复以下步骤：

a. 队列顶点 u 出队；

b. 依次检查 u 的所有邻接顶点 w, 若 $visited[w]$ 的值为 $false$, 则设为 $true$, 表示已访问，同时将 w 入队。

BFS 算法伪代码

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure

function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

图 2: BFS 算法伪代码 [2]

BFS 算法分析

完备性：显然，BFS 算法一定是完备的。因为对于一个网络，我们总能遍历其全部结点。

最优性：只有满足路径代价是基于结点深度的非递减函数时，BFS 算法才满足最优性。简单地说，即路径权值不能为负数，否则，BFS 算法找到最浅深度的目标并不一定是最优（路径长度最小）的目标。

时间复杂度：在简单的算法分析中，如果使用邻接矩阵存储图结构，则使用 BFS 遍历的时间复杂度为 $O(|V|^2)$, 若使用邻接表，则为 $O(|V|+|E|)$. 套用标准的分析框架，遍历算法的时间

复杂度取决于需要遍历对象的个数。假设一颗深度为 d 的一致树，每个状态都有 d 个结点，则总结点数为

$$N(DFS) = b + b^2 + b^3 + ... + b^d$$

利用等比数列求和有

$$N(DFS) = \frac{b(1-b^d)}{1-b} = \frac{b^{d+1}}{b-1} - \frac{b}{b-1}$$

对于一些人工智能领域中的大型网络，通常 b 远远大于 1，故可近似认为 $b \approx b-1$, 则上式可化简为

$$N(DFS) = \frac{b^{d+1}}{b-1} - \frac{b}{b-1} \approx \frac{b^{d+1}}{b} = b^d = O(b^d)$$

即，对于普遍的 BFS 算法，其时间复杂度为 $O(b^d)$。

空间复杂度：由于在深度搜索算法中使用 FIFO 队列进行辅助存储，对于一个有层次的图结构，其需要的最大存储空间取决于最外层结点的数量。由上述假设可知，最外层结点数为 b^d，故 BFS 算法的空间复杂度为 b^d。

3.3 迭代加深的深度优先搜索 (IDS)

IDS 算法的基本思想

迭代加深的深度优先搜索 (Iterative Deepening Search) 经常和深度优先搜索结合使用来确定更好的深度界限。其做法是不断增大深度限制。其流程如下：

（1）将深度限制设为 0；

（2）深度限制加 1，判断是否找到目标若找到，结束；否则深度限制继续加 1；

（3）重复步骤（2）直到深度界限达到 d，即最浅的目标结点所在深度时，找到目标结点，结束。

IDS 算法伪代码

IDS 算法效率分析

完备性：由上分析，DFS 算法在树搜索中是不完备的。但在有深度限制的情况下显然是完备的，即 IDS 算法是完备的。

最优性：与 BFS 算法相近，IDS 算法只有在路径代价是结点深度的非递减函数时该算法才是最优的。

时间复杂度: 在 IDS 算法中，搜索过程在目标结点所在的最浅深度层 (d 层) 结束, 自底向上，第 i 层的结点被生成 i 次，因为其每到一层的新结点时都会进行回溯判断，即根结点共被生成 d 次。因此，生成结点的总数为:

$$N(IDS) = (d)b + (d-1)b^2 + ... + (1)b^d$$

令上式左右两边同乘 b 可得：

$$bN(IDS) = 0 + (d)b^2 + (d-1)b^3 + ... + (2)b^d + (1)b^{d+1}$$

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution node or failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns a node or failure or cutoff
  frontier ← a LIFO queue (stack) with NODE(problem.INITIAL) as an element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        add child to frontier
  return result
```

图 3: IDS 算法伪代码 [2]

两式相减有：

$$(b-1)N(IDS) = -db + b^2 + b^3 + ... + b^d + b^{d+1} = \frac{b^2(1-b^d)}{1-b} - bd = \frac{b^{d+2}}{b-1} + \frac{bd - b^2 - b^2 d}{b-1}$$

即:

$$N(IDS) = \frac{b^{d+2}}{(b-1)^2} + \frac{bd - b^2 - b^2 d}{b^2 - 2b + 1}$$

同 BFS 时间复杂度的分析方法，当 b 远远大于 1 时, 其时间复杂度为 $O(b^d)$。

空间复杂度：由于 IDS 在 DFS 的基础上增加了深度限制的条件，此时最长深度为 d，故其空间复杂度为 $O(bd)$。

4 启发式搜索

4.1 贪婪最佳优先搜索 (GBFS)

GBFS 算法的基本思想

贪婪最佳优先搜索 (Greedy Best First Search) 试图扩展离目标最近的结点，即只用启发式信息，故 $f(n) = h(n)$。其中 $h(n)$ 主要使用欧氏距离 (Euclidean Distance) 或曼哈顿距离 (Manhatan Distance) 来表示，具体含义如下：

假设有两点坐标为 (x_1, y_1) 和 (x_2, y_2)，则欧式距离为

$$D_{Euclidean} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

曼哈顿距离为

$$D_{Manhatan} = |x_1 - x_2| + |y_1 - y_2|$$

显然，曼哈顿距离避免了复杂的开方计算。

GBFS 算法的流程与 DFS、BFS 是相似的，但其使用的数据结构是优先队列 (Priority Queue)，因为它在每一步都在寻找离目标最近的结点，即“优先级”高的结点。优先级的评判标准是代

价函数 $f(n)$，$f(n)$ 越小，优先级越高，反之优先级越低。

GBFS 算法分析

完备性：实质上，GBFS 算法和 DFS 算法只是增加了 $f(n)$ 的评估机制，但这并不影响其搜索的实质过程。故其在有限的空间上，图搜索是完备的；但在树搜索上，即使在有限状态空间，也是不完备的。

最优性：显然，GBFS 算法找到解就会返回，并不能保证该解是最优解。

时间复杂度：与 DFS 算法一样的分析思路，为 $O(b^m)$。

空间复杂度：由于 GBFS 算法使用优先级队列进行存储，故在最坏情况下需要存储在搜索路径上的所有结点，即与时间复杂度一样，为 $O(b^m)$。

4.2 A* 搜索算法

A* 算法的基本思想

在 GBFS 算法的基础上，我们缩小总评估代价，对结点的评估结合 $g(n)$，即到达此结点已经花费的代价，和 $h(n)$，从该结点到目标结点所花的代价为：

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

由此可知，$f(n)$ 表示经过结点 n 的最小代价解的估计代价。

A* 算法步骤如下：

(1) 从起点 A 开始，并把它就加入到一个由方格组成的 open list(开放列表) 中。这个 open list 有点像是一个购物单。当然现在 open list 里只有一项，它就是起点 A，后面会慢慢加入更多的项。Open list 里的格子是路径可能会是沿途经过的，也有可能不经过。基本上 open list 是一个待检查的方格列表。

(2) 查看与起点 A 相邻的方格 (忽略其中墙壁所占领的方格，河流所占领的方格及其他非法地形占领的方格)，把其中可走的 (walkable) 或可到达的 (reachable) 方格也加入到 open list 中。把起点 A 设置为这些方格的父亲 (parent node 或 parent square)。当我们在追踪路径时，这些父节点的内容是很重要的。

(3) 把 A 从 open list 中移除，加入到 close list(封闭列表) 中，close list 中的每个方格都是现在不需要再关注的。

A* 算法伪代码

A* 算法分析

完备性：如果在搜索的过程中每步代价都超过从根结点到达目标结点的估计代价与实际代价的相对误差并且 b（任意结点的最多后继数）是有穷的，则 A* 算法是完备的。

最优性：如果 $h(n)$ 是可采纳的，那么 A* 的树搜索版本是最优的；如果 $h(n)$ 是一致的，那么图搜索的 A* 算法是最优的。

The goal node is denoted by node_goal and the source node is denoted by node_start

We maintain two lists: **OPEN** and **CLOSE**:

OPEN consists on nodes that have been visited but not expanded (meaning that sucessors have not been explored yet). This is the list of pending tasks.

CLOSE consists on nodes that have been visited *and* expanded (sucessors have been explored already and included in the open list, if this was the case).

```
1  Put node_start in the OPEN list with f(node_start) = h(node_start) (initialization)
2  while the OPEN list is not empty {
3    Take from the open list the node node_current with the lowest
4         f(node_current) = g(node_current) + h(node_current)
5    if node_current is node_goal we have found the solution; break
6    Generate each state node_successor that come after node_current
7    for each node_successor of node_current {
8      Set successor_current_cost = g(node_current) + w(node_current, node_successor)
9      if node_successor is in the OPEN list {
10       if g(node_successor) ≤ successor_current_cost continue (to line 20)
11     } else if node_successor is in the CLOSED list {
12       if g(node_successor) ≤ successor_current_cost continue (to line 20)
13       Move node_successor from the CLOSED list to the OPEN list
14     } else {
15       Add node_successor to the OPEN list
16       Set h(node_successor) to be the heuristic distance to node_goal
17     }
18     Set g(node_successor) = successor_current_cost
19     Set the parent of node_successor to node_current
20   }
21   Add node_current to the CLOSED list
22 }
23 if(node_current != node_goal) exit with error (the OPEN list is empty)
```

图 4: A* 算法伪代码 [3]

时间复杂度：A* 算法的复杂度严重依赖于对状态空间的所做的假设。在只有一个目标状态的状态空间，其时间复杂度也是指数级的。

空间复杂度：同时间复杂度一样，其空间复杂度也严重依赖于对状态空间所做的假设。

5 局部搜索

5.1 模拟退火 (Simulated Annealing)

Simulated Annealing 算法的基本思想

模拟退火算法的思想来源于对固体退火降温过程的模拟。即将固体加温至充分高，再让其徐徐冷却。再加热固体时，固体中原子的热运动不断加强，内能增大，随着温度的不断升高，固体粒子运动状态有序被彻底破坏，固体内部粒子岁温度的升高而变为无序状态。冷却时，粒子逐渐趋于有序，在每个温度下都达到平衡状态，最后在常温下达到基态，同时内能也降至最低。

假设要解决的问题时一个寻找最小值的问题。将物理学中模拟退火的思想应用于优化问题中就可以得到模拟退火寻优方法。

算法的一般过程为：

（1）初始化退火温度 T_k(令 $k = 0$)，产生随机初始解 x_0；

（2）在温度 T_k 下重复执行如下操作，直至达到温度 T_k 的平衡状态;

a. 在解 x 的领域中产生新的可行解 x'；

b. 计算 x' 的目标函数 $f(x')$ 和 x 的目标函数 $f(x)$ 的差值 δf；

c. 依照概率 $min1, exp-\delta f/T_k > random[0,1]$ 接受 x'，其中 $random[0,1]$ 是 $[0,1]$ 区间内的随机数。

（3）退火操作：$T_{K+1} = CT_k, k = k+1$，其中 $C \in (0,1)$。若满足收敛判据，则退火过程结束；否则，转至步骤（2）。

其中退火温度控制着求解过程中向最优值得优化方向进行，同时它又以概率 $exp(-\delta f/T_k)$ 来接受劣质解，因此算法可以跳出局部极值点，只要初始温度足够高，退火过程足够满，算法就能收敛到全局最优解。

Simulated Annealing 算法伪代码

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

图 5: Simulated Annealing 算法伪代码 [2]

Simulated Annealing 算法分析

完备性：在爬山法中加入随机因素的 SA 算法是完备的，因为从后继集合中完全等概率的随机选取后继是可以找到解的，但是效率会很低。

最优性：由物理退火过程思想的启发，如果在温度下降过程中的梯度足够小，则到达最低温度的概率就会很大。同理，如果调度随机移动的距离，使其下降的足够慢，则算法找到全局最优解的概率就会逼近 1。

5.2 遗传算法 (Genetic Algorithm)

Genetic Algorithm 算法的基本思想

遗传算法是借鉴生物学中适者生存、优胜劣汰遗传机制等进化规律而提出得随机优化搜索算法。其主要特点是直接对结构对象进行操作，不存在求导和函数连续性得限定；采用概率化的寻优方法，能自动获取和指导优化的搜索空间，能自适应地调整搜索方向，不需要确定的规则。

其实现的基本步骤为：

（1）编码和产生初始群体；

（2）重复执行以下步骤：

a. 计算各个个体的适应度值；

b. 判断是否满足算法收敛准则。若满足，则输出收敛结果，否则执行步骤 c；

c. 执行选择操作；

d. 执行交叉操作；

e. 执行变异操作；

由于遗传算法中许多都是源于生物遗传中的概念，所有许多名称会让人感到不明所以。对应染色体中基因的编码过程，算法是根据具体问题确定可行解域，确定一种能用字符串或数值表示可行域的每一解的编码方法；然后采用适应度函数（一般为目标函数）对每一个解的好坏进行度量；最后需要确定进化参数群体规模 M、交叉概率 P_c，变异概率 P_m、进化终止条件。

Genetic Algorithm 算法伪代码

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

图 6: Genetic Algorithm 算法伪代码 [2]

Genetic Algorithm 算法分析

完备性：遗传算法从问题解的串集开始搜索，而不是从单个解开始。传统算法从单个初始值迭代求最优解的，容易误入局部最优解。遗传算法从串集开始搜索，覆盖面大，利于全局择优。

最优性：许多传统搜索算法都是单点搜索算法，容易陷入局部最优解。遗传算法同时处理群体中的多个个体，即对搜索空间中多个解进行评估，减少了陷入局部最优解的风险，同时算法本身易于实现并行化。

6 综述

理性认识

盲目搜索只是按照预定的策略搜索解空间的所有状态，而不会考虑到问题本身的特性。因此其适用范围较广，但这种“蛮力思想”所带来的往往是空间和时间的代价。比如 DFS 和 BFS 算法的复杂度都是指数级的，且对问题的解答也存在一定的遗漏。

启发式搜索就是在状态空间中的搜索对每一个搜索的位置进行评估，得到最好的位置，再从这个位置搜索直到目标。由部分信息对计算产生推理，个体能够基于经验或者个体经验交流改变搜索方式。这种搜索方式可以减少盲目性带来的重复搜索和冗余搜索，但加入的评估函数无疑会是过程变得复杂。比如 A* 算法中 $f(n)$ 函数的计算。

局部搜索是指能够无穷接近最优解的能力，而全局收敛能力是指找到全局最优解所在大致位置的能力。该算法主要是为解决一些难以找到最优解的近似算法，比如遗传算法解决 TSP 等 NP 问题等。缺点就是有时只能找到局部最优解，但相比于找到最优解所带来的时间和空间代价，这是值得的。

感性认识

算法是什么？维基百科上将其定义为："a set of rules that precisely defines a sequence of operation." 但在我探究这些算法的过程中，我觉得算法是人类从自然界中发现的解决基本问题的基本方案。在前面的几种经典算法中，其核心思想几乎都是源于自然界中事物发展的规律。在盲目搜索中，就像是一个原始人被困在迷宫里所能采取的基本的方法。在启发式搜索中，很多评价概念都是生活中常见的模型。尤其是在局部搜索中，爬山法是一个人在爬山过程中的策略；模拟退火则是基于对固体冷却过程的观察；遗传算法则是源于生物学进化理论的思想。每一种算法的思想都只是自然界事物面对一些问题所能运用的本能方法，就像物理学中事物的存在总是趋于熵最小的状态，化学的原子模型总是使其保持最低能量状态，而使其完成这种状态的过程就是一种完美的算法。每一种自然界中的算法都等待着人们的发现，然后在加工之后使用于更广泛的人类生活中。因此，对于人工智能领域中的一些难题，或许我们需要在自然界中的客观规律中寻找答案。

参考文献

[1] Goodrich and Tamassia; Cormen, Leiserson, Rivest, and Stein.

[2] http://aima.cs.berkeley.edu/algorithms.pdf

[3] https://mat.uab.cat/ alseda/MasterOpt/AStar-Algorithm.pdf