中国科学技术大学

University of Science and Technology of China

人工智能讲义

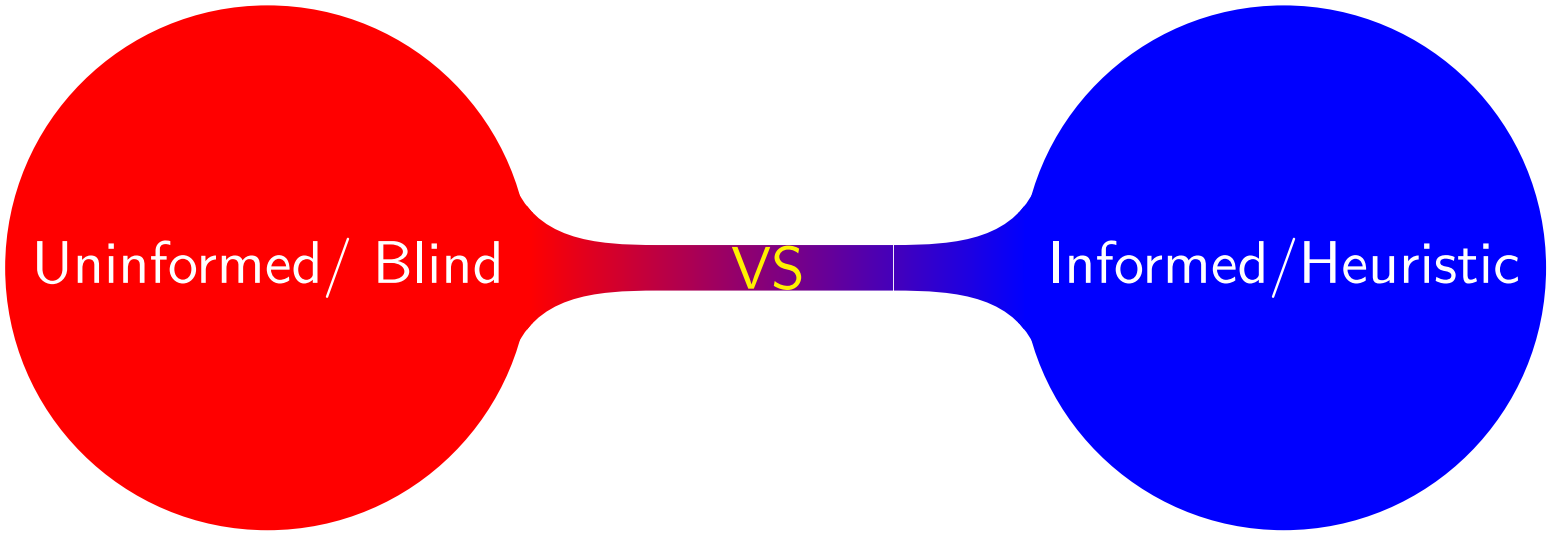
盲搜索

March 8, 2018

Outline

Outline

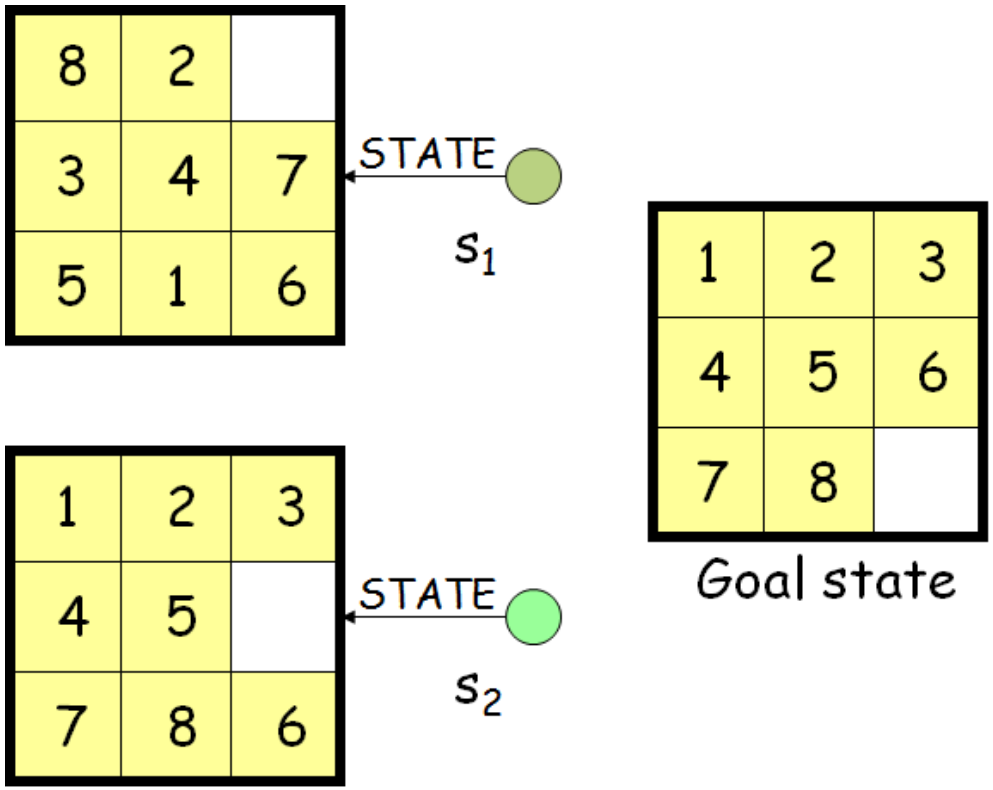
有/无信息搜索算法



FRINGE 是无序的

将更有希望的节点
放在未扩展节点集合
FRINGE 的前面，优先扩展

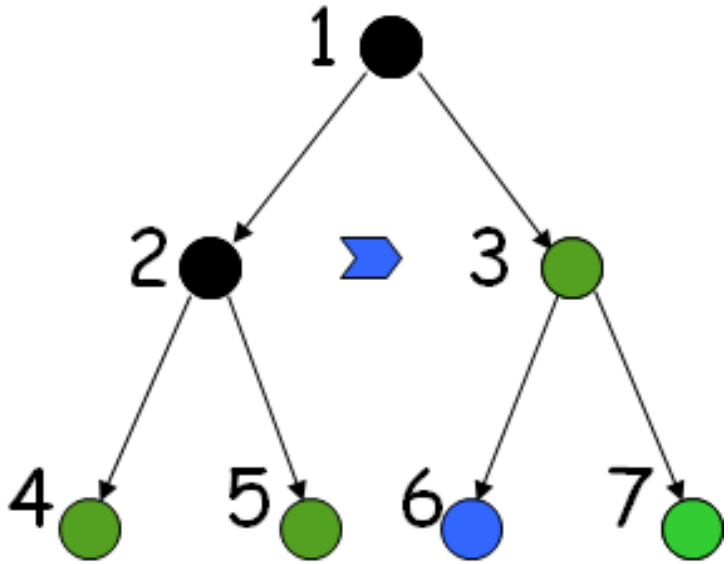
例子：(un)informed search



盲搜索和启发式搜索

- 如左图所示例子：
- 盲搜索：s_1 和 s_2 的次序是“随机的”，树的结构确定的，在算法实现时预先“确定”下来的
- 启发式搜索：状态 s_2 更接近目标状态（错误位置更少），因此可以优先扩展状态 s_2

基准算法的进一步解释 1



基准算法是盲搜索算法

- 如上图所示的节点扩展过程:
- 新节点/状态插入到未扩展节点队列 FRINGE 的队尾
- FRINGE = (3, 4, 5) $\rightarrow$ FRINGE = (4, 5, 6, 7)

基准算法的进一步解释 2

基准算法：宽度优先搜索算法

- 算法的重要参数:
  - 分支因子 b，后继函数返回的最大状态数目
  - 从初态到目标状态的最小深度 d，或者说是宽度优先生成树/搜索树上“埋藏最浅”的目标节点的深度

基准算法的进一步解释 3

基准算法: 评价

- 完备性？有解时给出解；无解时告知无解。
- 最优性？返回的是否是路径耗散最小的路径。
- 复杂性？时空代价。

基准算法评价结论

- 完备的；
- 如果每条边的路径耗散相等，则返回最优解；否则不一定；
- 访问节点数 $\leq 1 + b + b^2 + \ldots + b^d = \frac{b^{d+1}-1}{b-1} = O(b^d)$

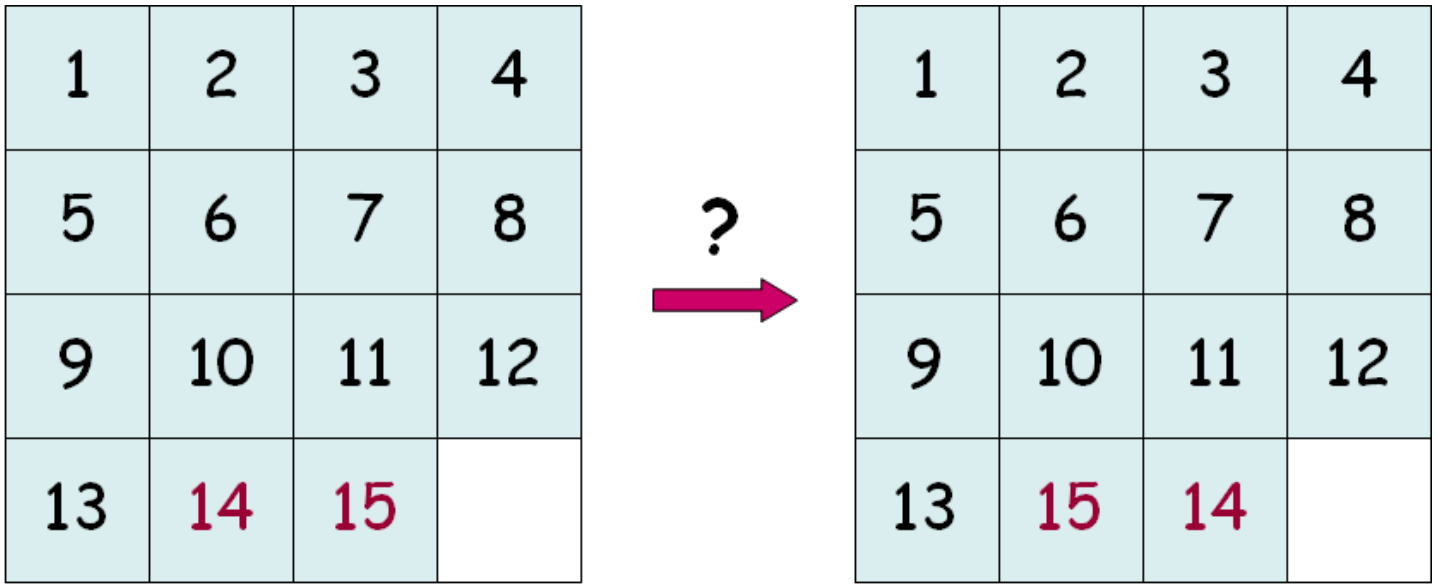
基准算法的进一步解释 4

d	# Nodes	Time	Memory
2	111	.01 msec	11 Kbytes
4	11,111	1 msec	1 Mbyte
6	~10^6	1 sec	100 Mb
8	~10^8	100 sec	10 Gbytes
10	~10^{10}	2.8 hours	1 Tbyte
12	~10^{12}	11.6 days	100 Tbytes
14	~10^{14}	3.2 years	10,000 Tbytes

时间和内存需求的直观认识，如上表，假设：

- 分支因子：$b = 10;$
- 节点处理速度：$1,000,000 nodes/sec;$
- 节点大小 $100 bytes/node$

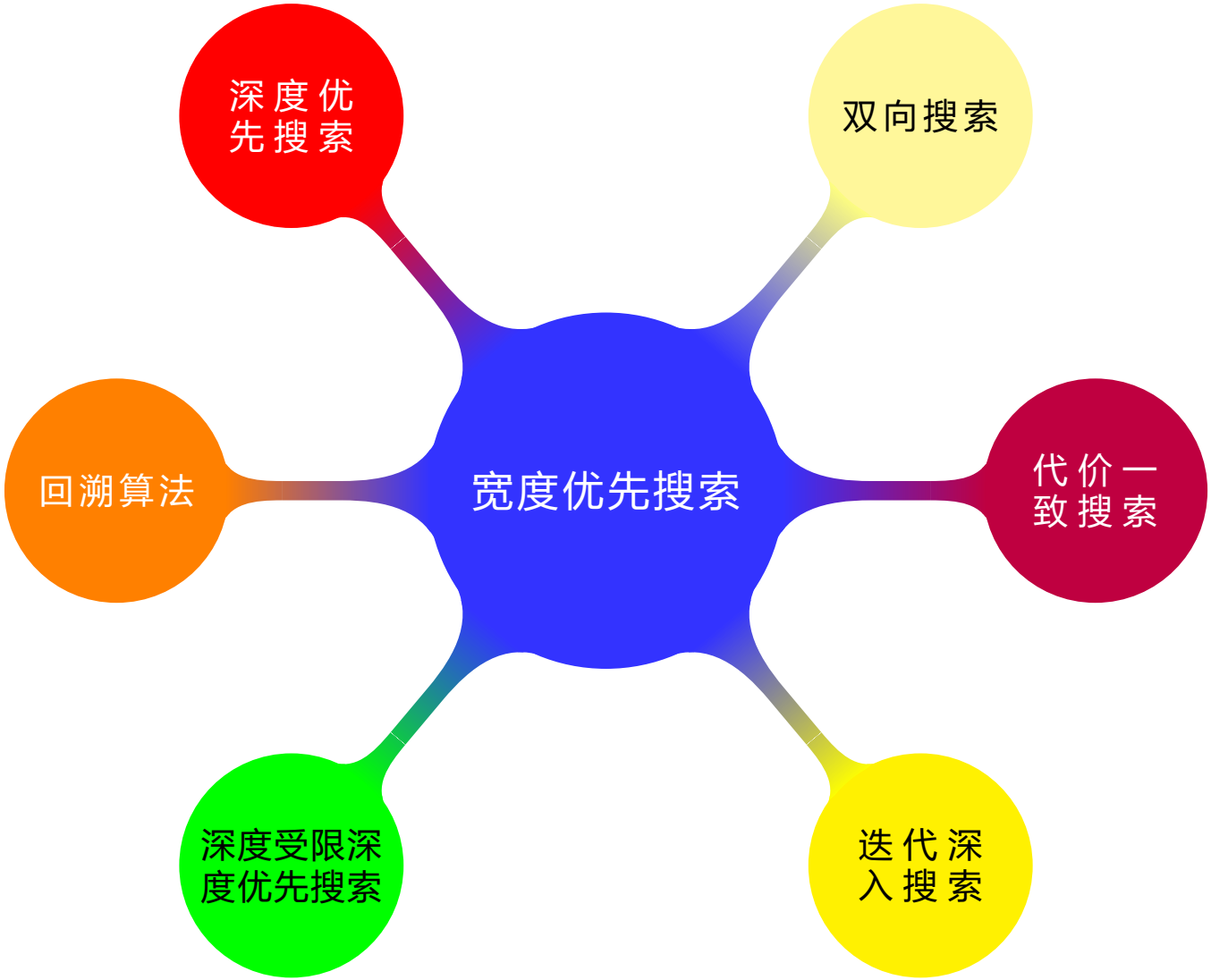
基准算法的进一步解释 5



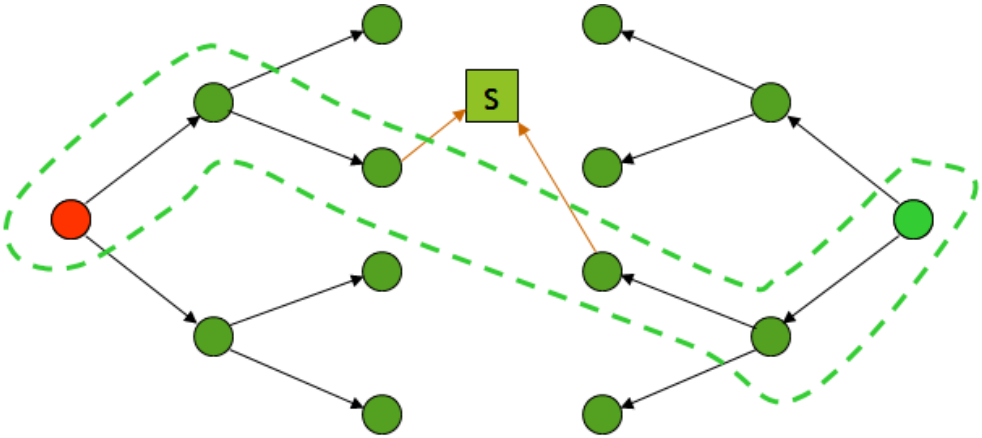
无解时的情况

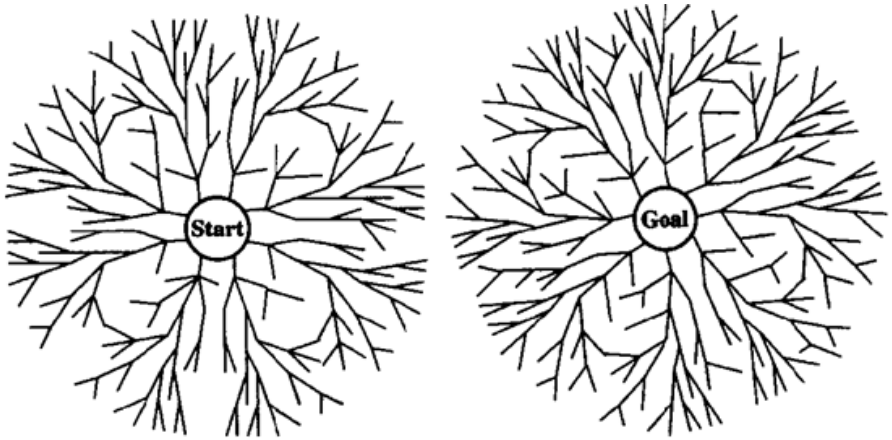
- 问题无解时，若状态空间无限大或者任意状态可被任意次重复访问，则宽度优先搜索算法不会停止

盲搜索算法的改进



改进 1：双向搜索算法





双向搜索算法

- 分别从初态和终态启动两个宽度优先搜索算法；
- 维护 2 个未扩展节点集合：FRINGE1 和 FRINGE2，分别记录从初态和终态开始搜索的未扩展节点集合；当两个集合相交时，算法结束。
- 时间和空间复杂度是 $O(b^{d/2}) << O(b^d)$（假设两个方向的分支因子都是 b）；
- 问题：两个方向的分支因子不一样会怎样？

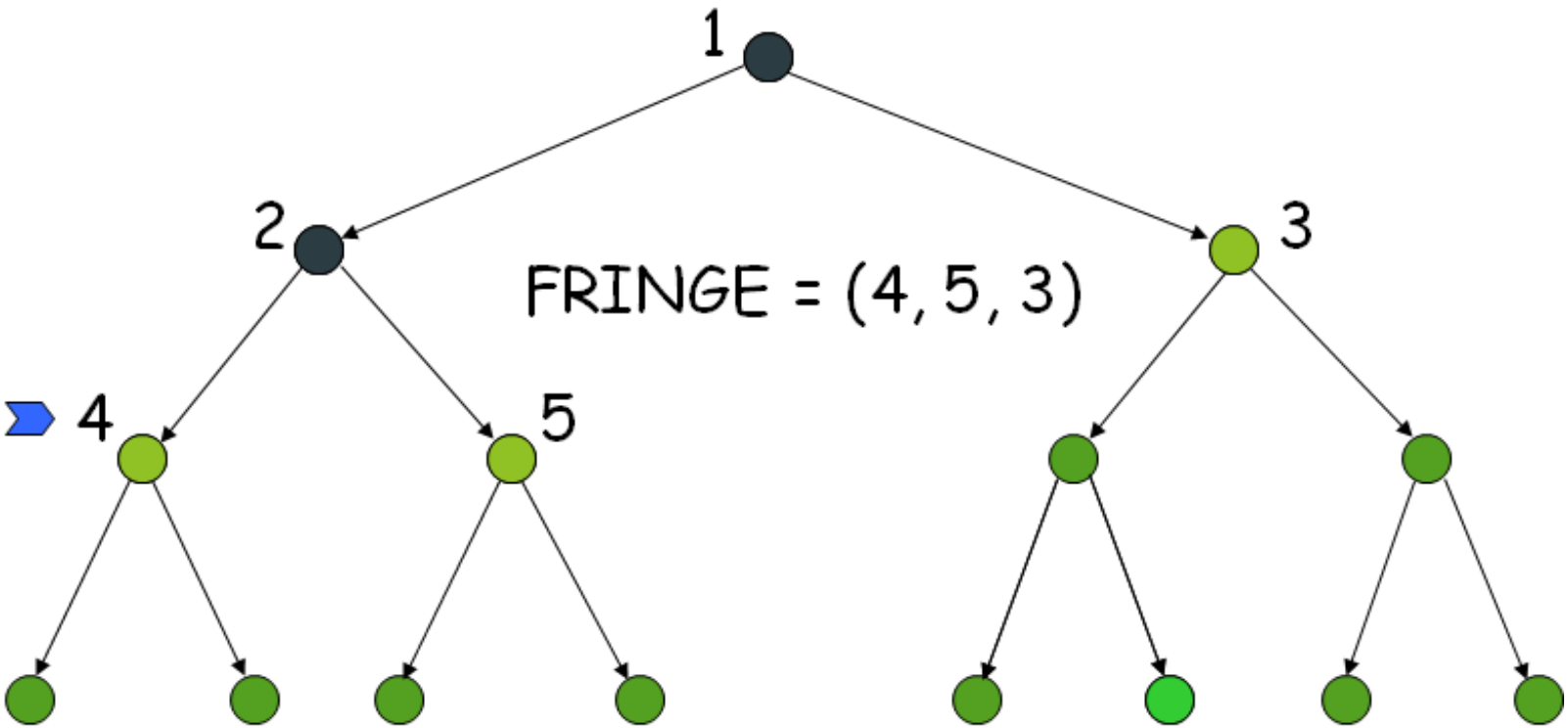
双向搜索：解释

- 双向搜索，是策略/算法框架的改进，而非“原子的”的搜索算法，双向搜索的两个方向（前向/反向）搜索算法，可以一样，也可以不一样
- 问题：总是能找到“好”的反向搜索算法吗？目标状态是单个精确描述的状态？满足某个条件的状态集合？后继函数的逆函数“前驱函数”能方便地描述或表达吗？(比如象棋的目标状态：获胜的布局，个数？如何描述，等等)

双向搜索：评价

- 时间复杂度：获得极大的优化 $O(b^{d/2}) << O(b^d)$，但是“问题本质”（指数时间复杂度）没有变化。这已经是双向搜索带来的明显进步；
- 完备性：完备的；
- 最优性：边的代价都为 1 时，能保证最优性；否则不一定。

改进 2：深度优先搜索



深度优先搜索算法

- 节点扩展时，新状态/节点总是插入到队列/FRINGE 的“队头”（栈）

改进 2：深度优先搜索

评价准则

- 完备性？有解时给出解；无解时告知无解。
- 最优性？返回的是否是路径耗散最小的路径。
- 复杂性？时空代价。

深度优先搜索算法

- 若搜索树有限，则是完备的；
- 不一定是最优的；
- 访问节点数（最坏情形）：$1 + b + b^2 + \ldots + b^m = O(b^m)$，空间复杂度 $O(bm)$，其中 m 是叶子节点的最大深度。

改进 3：回溯法

回溯法：对深度优先搜索的改进

- 每次扩展节点的时候，只扩展一个节点，节省内存，最多同时保存 $O(m)$ 个节点
- 若深度优先搜索树是无限的，则回溯搜索可能是不完备的，也可能无法得到最优解

改进 4：深度受限深度优先搜索

深度受限搜索：深度优先搜索的改进

- 设置扩展节点的深度阈值 k，当节点深度大于 k 时，节点不再扩展
- 算法返回结果：
  - 解
  - 无解 failure
  - 深度阈值 k 内无解

进一步思考与讨论

- 如何容易得到 k，算法性能将会得到提升
- Q: k 比 d(最浅目标状态的深度) 大或小时，会怎样?

改进 5：迭代深入搜索

迭代深入：当不知到受限深度阈值 k 时，从小到大一个个试

- 使用不同的受限深度参数 $k = 0, 1, 2, \ldots$ 不断重复执行“深度受限搜索”；
- 目的：寻找合适的深度受限深度参数 k 值。

对迭代深入搜索的思考

- 带来的好处：结合了宽度优先和深度优先二者的长处（时空复杂度，完备性和最优性）
- 付出的代价？

改进 5：迭代深入搜索

评价准则

- 完备性？有解时给出解；无解时告知无解。
- 最优性？返回的是否是路径耗散最小的路径。
- 复杂性？时空代价。

迭代深度搜索

- 当 $k = d$ 时，是完备的；
- 当单步路径耗散相同时，是最优的；否则不一定；
- 访问节点数（最坏情形）：$(d+1)(1) + db + (d-1)b^2 + (d-2)b^3 + \ldots + (1)b^d = O(b^d)$，空间复杂度 $O(bd)$

改进 5：迭代深入搜索

d = 5 and b = 2

宽度优先	迭代深入
1	1 x 6 = 6
2	2 x 5 = 10
4	4 x 4 = 16
8	8 x 3 = 24
16	16 x 2 = 32
32	32 x 1 = 32
63	120

d = 5 and b = 10

宽度优先	迭代深入
1	6
10	50
100	400
1,000	3,000
10,000	20,000
100,000	100,000
111,111	123,456

进一步讨论

- 对未知问题，解的深度未知，付出较小的代价，迭代深入搜索是首选的盲搜索算法

改进 6：代价一致搜索

代价一致搜索：总是优先扩展使得总路径耗散最小的节点

- 基准宽度优先搜索算法，每次在深度最浅的节点中（随机）选择一个扩展；
- 代价一致搜索：每次在 FRINGE 中选择让目前获得的路径耗散最小的节点扩展，适用于单步路径耗散不同时的情形；
- 当单步路径耗散相同时，代价一致搜索等价于基准算法；
- 要求单步耗散有下界，即 $c_i \geq \epsilon > 0$;

代价一致搜索：讨论

- 路径耗散引导搜索过程；
- 搜索的时空复杂度和 ϵ 相关；(最坏时 $O(b^{\lceil C*/\epsilon \rceil})$)
- 能保证最优性和完备性，一般来说时空复杂性较基准算法大。

盲搜索算法比较

评价标准	广度优先	代价一致	深度优先	深度有限	迭代深入	双向搜索（如果可用）
是否完备?	是[a]	是[a,b]	否	否	是[a]	是[a,d]
时间	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
空间	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
是否最优?	是[c]	是	否	否	是[c]	是[c,d]

图 3.17　各种搜索策略的评价。b 是分支因子；d 是最浅的解的深度；m 是搜索树的最大深度；l 是深度限制。右上角标的含义如下：[a] 如果 b 是有限的，则是完备的；[b] 如果对于正值常数 ϵ 有单步耗散 $\geq \epsilon$，则是完备的；[c] 如果单步耗散都是相同的，则是最优的；[d] 如果每个方向的搜索都使用广度优先搜索

搜索算法：避免状态重复访问

状态被重复访问的产生原因

- 行动可逆，则有可能会出现重复状态，如“三国华容道”；搜索树是无限的
- 行动不可逆，不会出现重复状态，如“8 皇后问题”的形式化方法 2；搜索树有限

基准算法：避免状态重复访问

- 来自数据结构中的技术：设置一个标识数组，标识状态是否被访问过/扩展过。
- 若扩展节点得到的后继状态已经被扩展过/访问过，就直接丢弃该状态及其对应节点。

需要多大的存储空间来存放标识数组？

避免状态重复访问

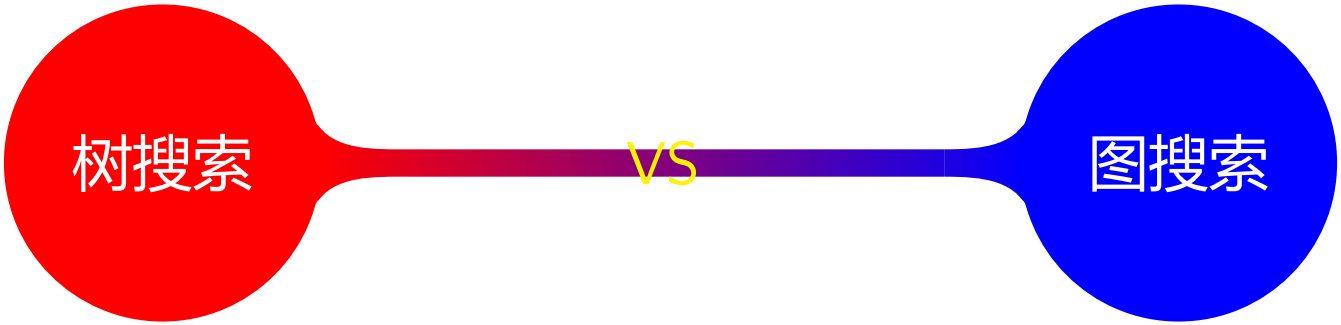
更完善的方法

- 用 CLOSED 表，存储所有访问过（扩展过）的状态
- 未扩展的状态用 OPEN 表来标识
- 若当前待扩展的状态已经在 CLOSED 表中，则丢弃该状态；否则扩展当前状态

OPEN 和 CLOSED 表方法的评述

- 采用该方法避免状态重复访问的算法框架称为“图搜索”算法，直接探索“状态图”
- 内存需求巨大!
- 最优性如何保证？宽度优先搜索。

图搜索和树搜索



相同与不同

- 树搜索中，不同的节点 N_1, N_2 可能表示的是相同的状态 s，分别表示从初态出发，到达状态 s 的两条不同路径（可能具有不同的路径耗散）
- 图搜索中，相同的状态只有一个节点来表示；不同的节点代表不同的状态；可以想象成把“树搜索”中具有相同状态的节点合并为一个节点，得到了“图”
- 树的“层次遍历”算法类似于图的“宽度优先搜索”遍历算法

避免状态重复访问：例子

代价一致搜索

- CLOSED 表中保存每个状态的最优路径耗散（从初态出发）
- 若 s 在 CLOSED 表中，则检查 s 的代价（从初始状态到 s 的路径耗散），并和当前路径 $+s$ 的总路径耗散进行比较，取较小的路径耗散，更新状态 s 的路径耗散；若 s 不在 CLOSED 表中，则放入 CLOSED 中，并开始扩展；
- 将 s 的所有不在 CLOSED 表中的后继状态放入 OPEN 表中
- 重复上两步

关于完备性的一点讨论

状态的重复访问与搜索的完备性

- 若状态空间是无限的，一般来说，搜索是不完备的
- 若状态空间有限，但允许状态被任意次重复访问，搜索一般是不完备的
- 若状态空间有限，访问时重复访问的节点被丢弃，则搜索是完备的，但是一般不是最优的