人工智能原理与算法

第4章 复杂环境中的搜索

中国科学院自动化研究所
国科大人工智能学院
朱翔昱

- 局部搜索算法和最优化问题
- 连续空间中的局部搜索
- 使用不确定动作的搜索
- 使用部分可观察信息的搜索
- 联机搜索智能体和未知环境

- 局部搜索算法和最优化问题
- 连续空间中的局部搜索
- 使用不确定动作的搜索
- 使用部分可观察信息的搜索
- 联机搜索智能体和未知环境

4.1 局部搜索算法和最优化问题

- 上一章介绍的算法都是系统地探索空间，保留一条或多条路径和记录路径中的每个结点的选择；
- 当找到目标时，到达此目标的路径就是这个问题的一个解；

4.1 局部搜索算法和最优化问题

- **搜索**：在**可观察的、确定的、已知**的环境之下的搜索。通过在内存中保留一条或多条路径和记录路径中的每个结点的选择。当找到目标时，达到此目标的路径就是这个问题的一个**解**；
- **局部搜索**：不关心路径代价，但是关注解状态。从单个当前结点（而不是多条路径）出发，通常只移动到它的邻近状态。一般情况下**不保留搜索路径**；
- **优点**：
  - 通常只用**常数级**的内存；
  - 通常能在系统化算法不适用的很大或无限的（连续的）状态空间中找到合理的解。

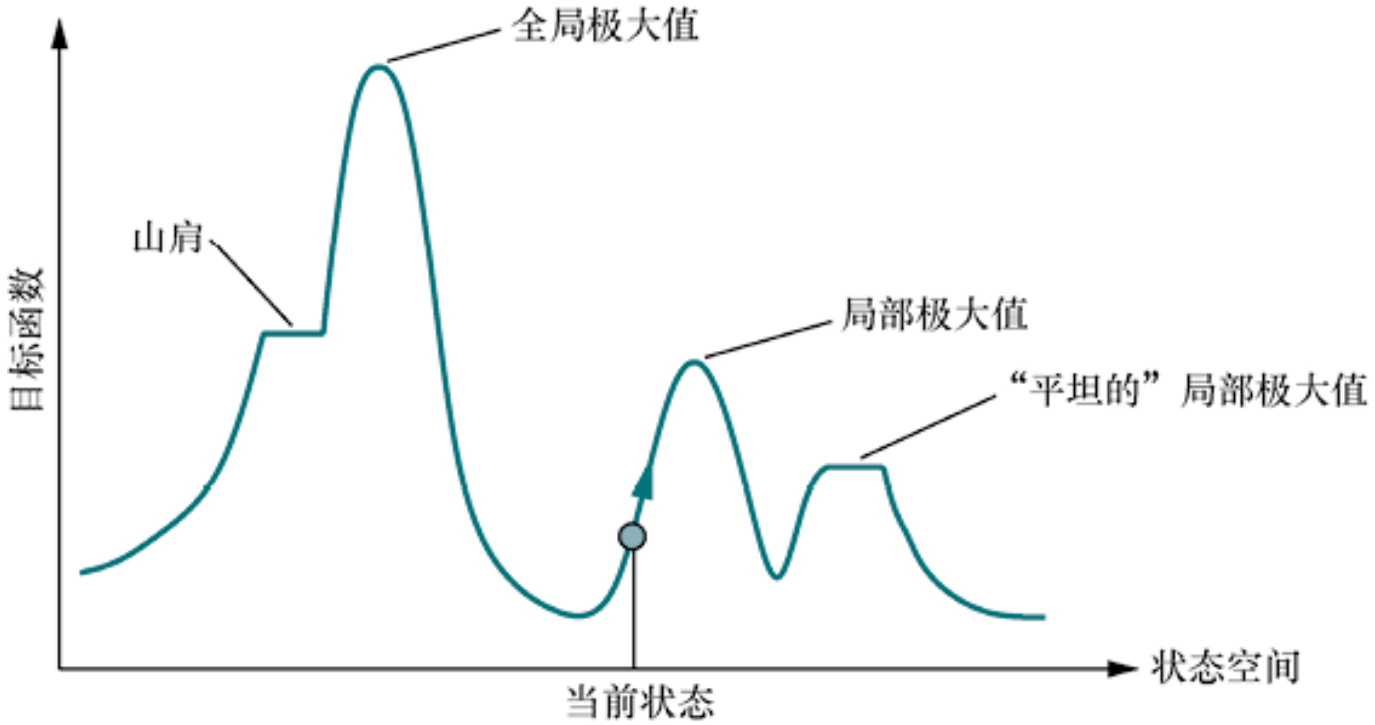
4.1.1 爬山法

- **局部搜索：**
  - 爬山搜索算法是最基本的局部搜索技术。在每一步中，当前节点被其最优邻居节点替换：

function HILL-CLIMBING(*problem*) **returns** 一个位于局部极大值的状态
  current ← *problem*.INITIAL
  while true **do**
    neighbor ← *current*的值最大的后继状态
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*

4.1 局部搜索算法和最优化问题

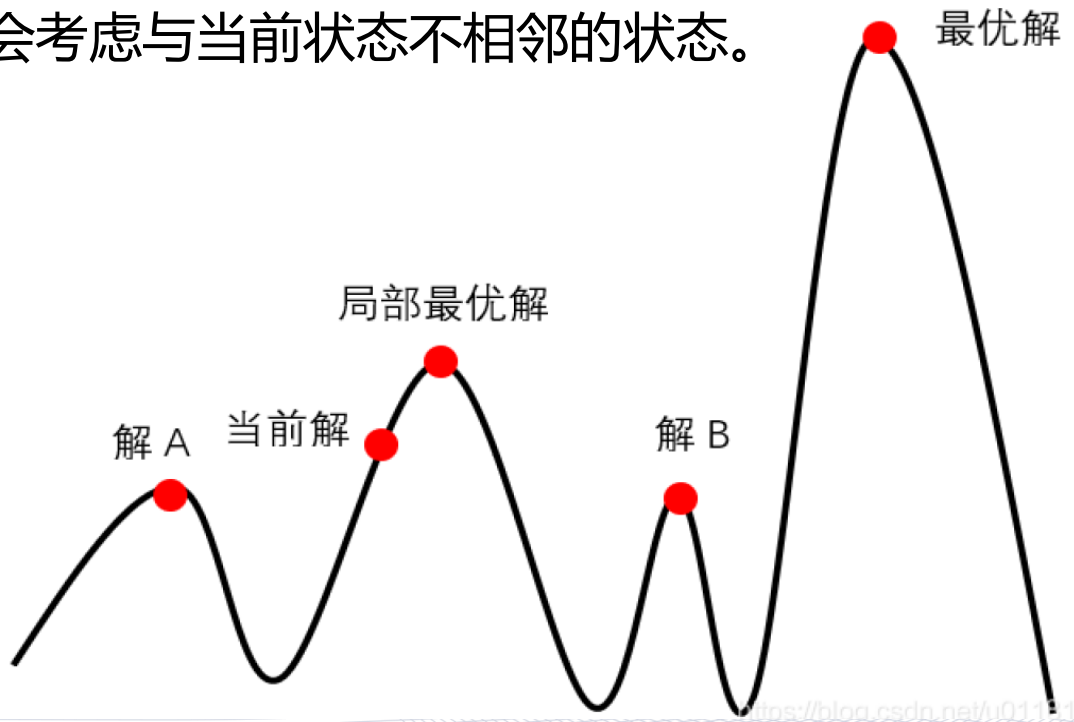
- **局部搜索：**



  - 局部搜索算法就是探索这个地形图：
    - 如果存在解，那么完备的局部搜索算法**总能找到解**；
    - 最优的局部搜索算法总能找到**全局最小值/最大值**。

4.1.1 爬山法

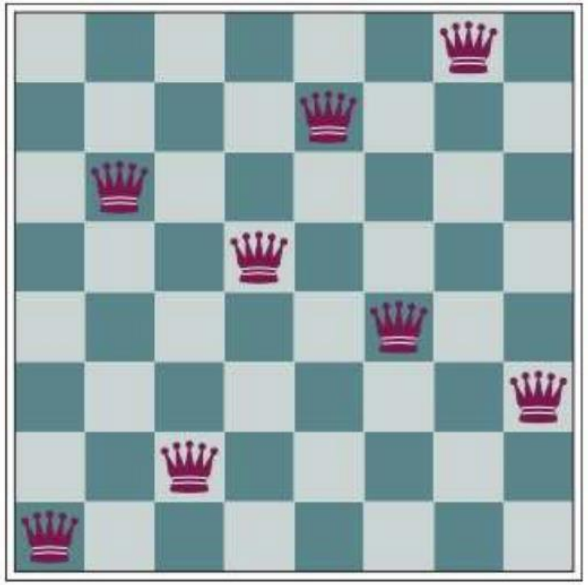
- **特点：**
  - 算法在达到一个“**峰顶**”时终止，邻接状态中没有比它的值更高的；
  - 算法**不维护搜索树**，当前结点的数据结构只需要记录当前状态和目标函数值；
  - 爬山法不会考虑与当前状态不相邻的状态。



4.1.1 爬山法

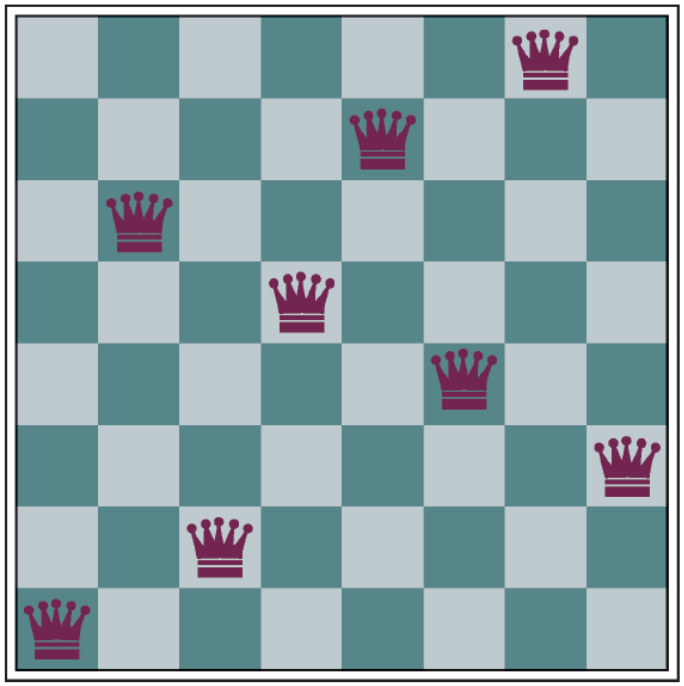
八皇后问题：

- 局部搜索算法一般使用完整状态形式化，即每个状态都包括在棋盘上放置8个皇后，每列一个；
- 后继函数指的是移动某个皇后到这列的另一个可能方格中（因此每个状态有8*7=56个后继）；
- **启发式评估函数h**是形成相互攻击的皇后对的数量；
- 该函数的全局最小值是0，仅在找到解时才会是这个值。

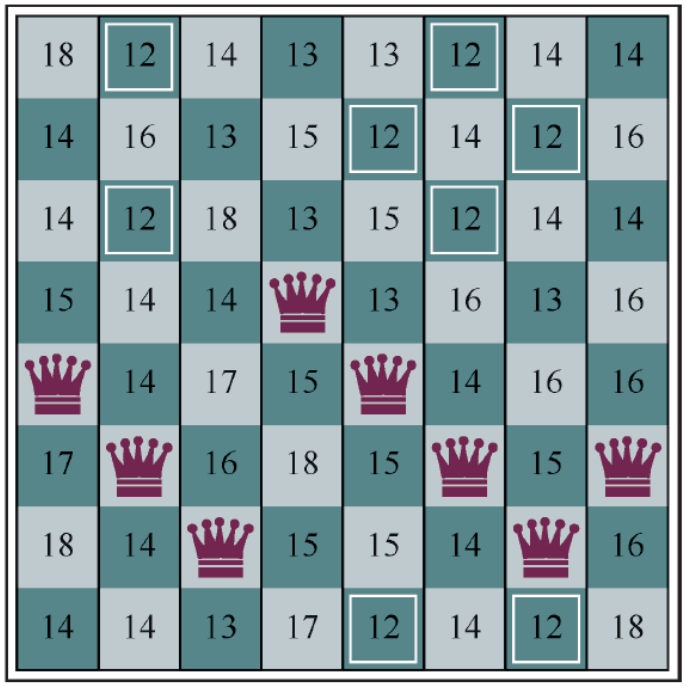


4.1.1 爬山法

- 如果有多个后继同是最小值，爬山法会在最佳后继集合中随机选择一个进行扩展：



（a）



（b）

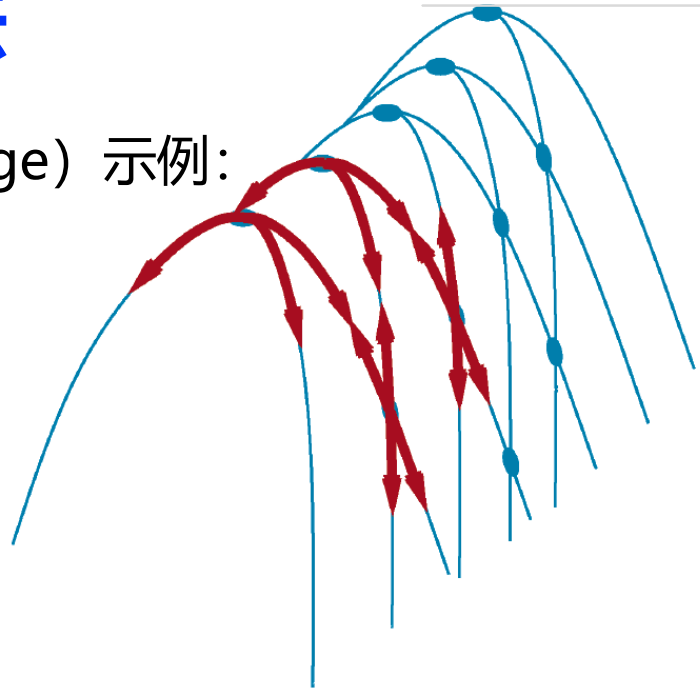
图 4-3 （a）8 皇后问题：在棋盘上放置 8 个皇后，使得它们不能互相攻击。（皇后会攻击同一行、同一列或对角线上的任何棋子。）当前状态非常接近于一个解，除了第 4 列和第 7 列的两个皇后会沿对角线互相攻击。（b）一个 8 皇后状态，其启发式代价估计值 $h = 17$。棋盘显示了通过在同一列移动皇后而获得的每一个可能后继的 h 值。有 8 个移动并列最优，其 $h = 12$。爬山法将选择它们中的一个

4.1.1 爬山法

- **爬山法的缺陷：**
  - **局部极大值**（local maxima）：局部极大值是一个比它的每个邻接结点都高的峰顶，但是比全局最大值要小。爬山法算法达到局部极大值附近就会被拉向峰顶，然后就卡在局部极大值处无处可走；
    - **山脊**（岭，ridge）：山脊造成一系列的局部极大值，贪婪算法很难处理这种情况；
    - **高原**（平台区，plateau）：高原是状态空间地形图上的一块平原区域。它可能是一块平的局部极大值，不存在上山的出口，或者是山肩。爬山法在高原可能会迷路。

4.1.1 爬山法

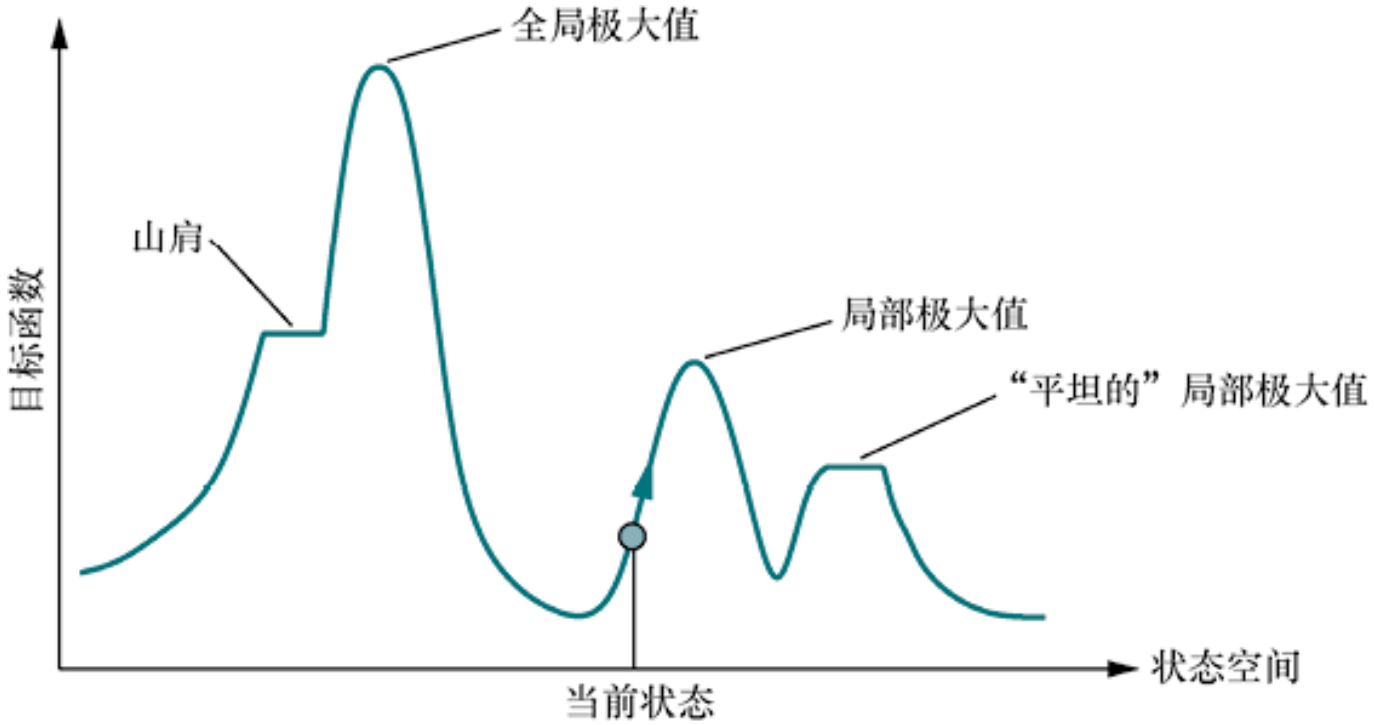
- 山脊（岭，ridge）示例：



- 状态网格（蓝色圆点）叠加在从左到右上升的岭上，形成了一个彼此不直接相连的局部极大值序列；
- 从每个局部极大值出发，所有可选动作都指向下坡。
- 这样的拓扑在低维状态空间中很常见，例如二维平面中的点。但是在具有成百上千个维度的状态空间中，这种直观图并不成立，而且通常至少存在几个维度使得算法有可能漏掉岭和平台区。

4.1.1 爬山法

- 高原（平台区，plateau）示例：



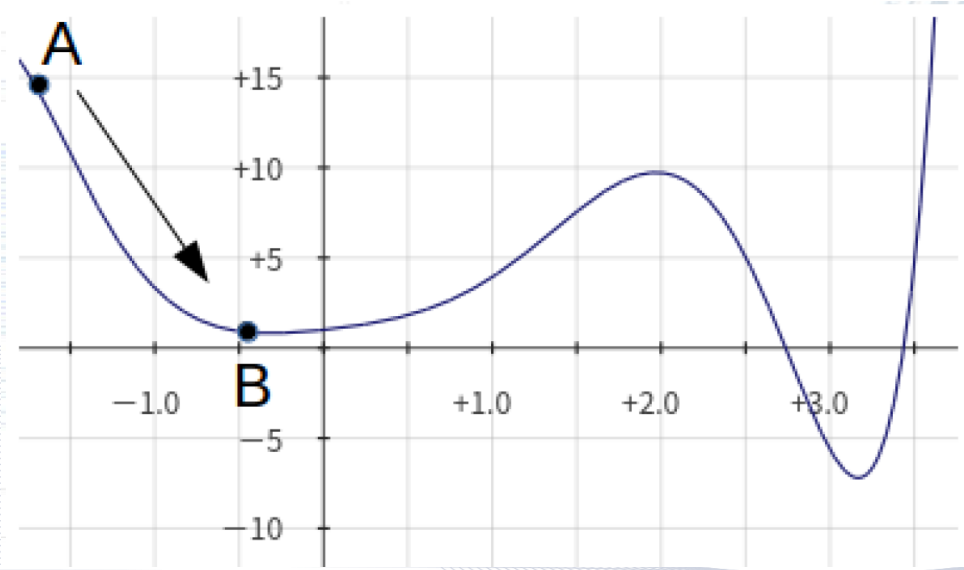
- 从每个局部极大值出发，所有可选动作都指向不动。
- 高原是状态空间地形图上的一块平原区域。它可能是一块平的局部极大值，不存在上山的出口，或者是山肩。爬山法在高原可能会迷路。

4.1.1 爬山法

- **克服局部极大值问题：**
  - **随机爬山法**：在上山移动中随机选择下一步，被选中的概率可能随着上山移动的**陡峭程度**不同而不同：
    - 这种算法虽然收敛速度比最陡上山法慢不少，但是在某些状态空间地形图上它能找到最好的解。
  - **首选爬山法**：基于随机爬山法，随机地生成后继结点直到生成一个优于当前结点的后继：
    - 这种算法在**后继结点很多的时候**是个好策略。
  - **随机重启爬山法**：通过随机生成初始状态导引爬山法搜索，直到找到目标。如果没有成功，那么尝试，再尝试（重新开启搜索）：
    - 这种算法完备的概率**接近于1**。

4.1.2 模拟退火搜索

- **模拟退火搜索：**
  - 爬山法搜索从来不“**下山**”，即不会向值比当前结点低的方向搜索，因此有可能**卡在局部极大值上**；
  - 随机游走搜索从后继中等概率选择后继结点，**效率极低**；
  - **模拟退火**：先高温加热，后逐渐冷却（冶金淬火）。



4.1.2 模拟退火搜索

- **当移动到一个位置上：**

① 如果该移动视情况改善，该移动则被接受；否则，算法以某个**小于1的概率**接受该移动；

② 如果移动导致状态“变坏”，概率则成指数级下降：**评估值ΔE变坏**；

- 这个概率也随“温度”T降低而下降：开始T高的时候可能允许“坏的”移动，T越低则越不可能发生。

③ 如果调度让T下降得足够慢，算法找到全局最优解的**概率逼近于1**。

$$e^{\Delta E/T}$$

4.1.2 模拟退火搜索

- 模拟退火算法，一种允许某些下坡移动的随机爬山法。输入的schedule 是关于时间的函数，它决定了“温度”T的值：

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** 一个解状态

current ← *problem*.INITIAL

for $t = 1$ **to** ∞ **do**

T ← *schedule*(*t*)

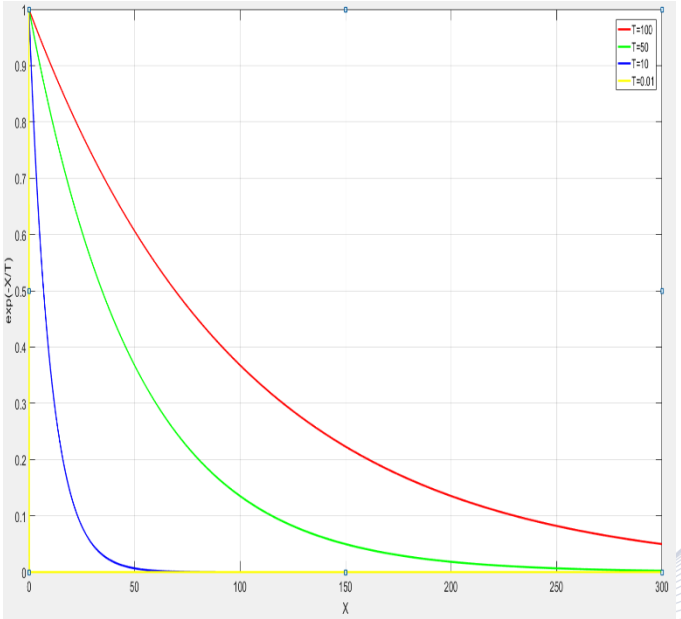
if $T = 0$ **then return** *current*

next ← *current*的一个随机选择的后继状态

ΔE ← VALUE(*current*) – VALUE(*next*)

if $\Delta E > 0$ **then** *current* ← *next*

else *current* ← *next*仅以$e^{\Delta E/T}$的概率



4.1.3 局部束搜索

- **局部束搜索：**
  - **局部束搜索算法**记录k个状态而**不是只记录一个**，他从k个随机生成的状态开始，每一步k个状态的所有后继状态全部被生成；
    - 如果其中有一个是目标状态，则算法停止；否则，它从整个后继列表中选择k个最佳的后继，重复这个过程；
    - 局部束搜索与随机重启搜索的不同。随机重启搜索，每个搜索进程完全独立。而在局部束搜索中，有用信息将在并行的搜索线程之间传递。
  - **随机束搜索算法**不是从候选后继集合中选择最好的k个后继状态，而是随机选择k个后继状态，其中选择给定后继状态的概率是状态值的递增函数。

4.1.4 遗传算法

- **遗传算法**（generic algorithm/GA）是随机束搜索的一个变形——不是通过修改**单一状态**而是通过把**两个父状态结合**以生成后继状态：
  - 与束搜索一样，遗传算法也是从k个随机状态开始——这k个状态称为**种群**，每个状态称为**个体**；
  - 个体用有限长的字符串（通常为0/1串）表示；
  - 个体状态用其评价函数（适应度函数）给出评价值（适应值）；
  - 随后的操作包括——**选择/杂交/变异**。

4.1.4 遗传算法 –遗传算法的操作

- **选择**：按照一定概率随机地选择两对个体进行繁殖（即生成后继状态）；
- **交叉**：交叉点是表示状态的字符串中随机选择的一个位置，以此形成新状态——后代是父串在杂交点上进行杂交（各取一部分）得来的；
- **变异**：在新生成的串中各个位置都会按照一个独立的小概率随机变异。

4.1.4 遗传算法

- 每个字符串代表8皇后问题的一个状态，第c位数字表示第c列中皇后的行号；
- 每个状态根据**适应度函数**进行评级。适应度越高越好，本例使用**非攻击皇后对的数量**作为适应度，解的适应度为8×7/2 = 28。

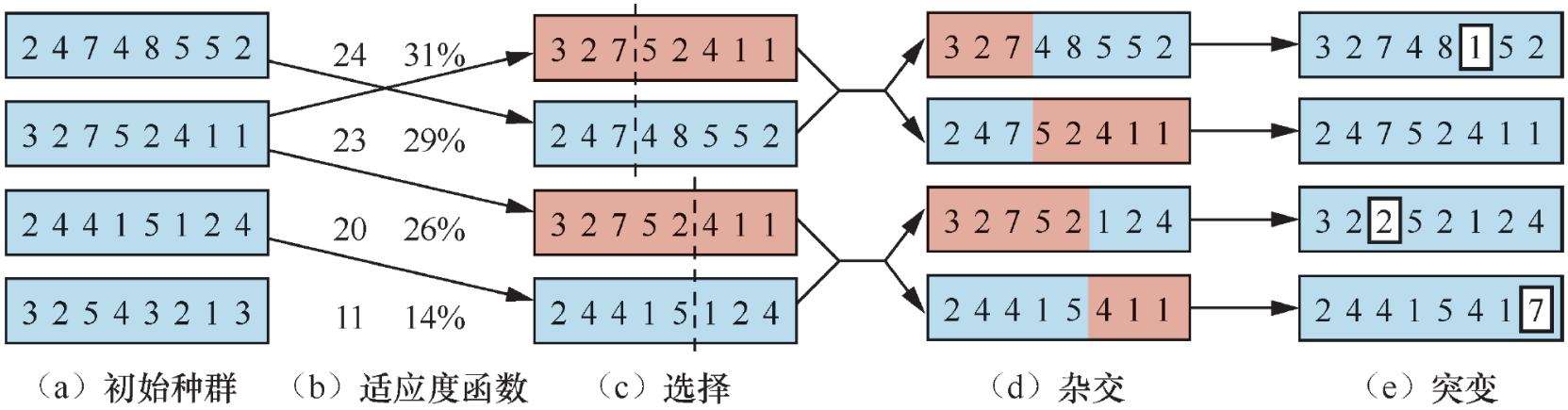


图 4-6　遗传算法，图示为表示 8 皇后状态的数字字符串。（a）中的初始种群根据（b）中的适应度函数进行排序从而得到（c）中的配对，（d）是产生的后代，（e）是可能发生的突变

4.1.4 遗传算法

- (b)中4个适应度分别为24、23、20 和11。然后将适应度得分**归一化**为概率，结果显示在(b)的适应度旁边。按照此概率选出两对父字符串（可重复选，也可能有遗漏）。对于每一对被选择的亲本，随机选择一个杂交点（虚线）得到（c）。

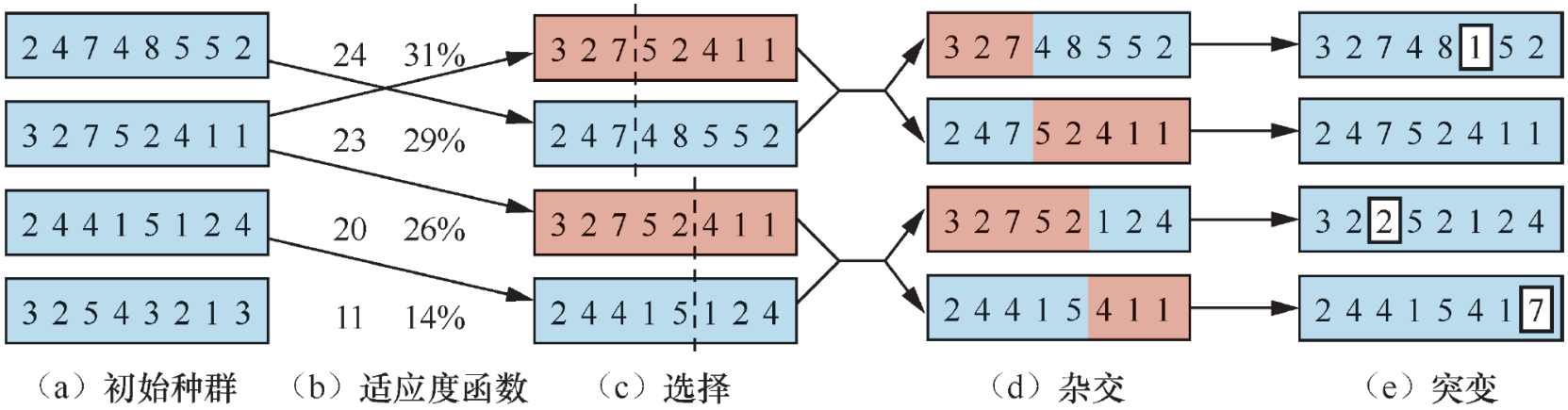
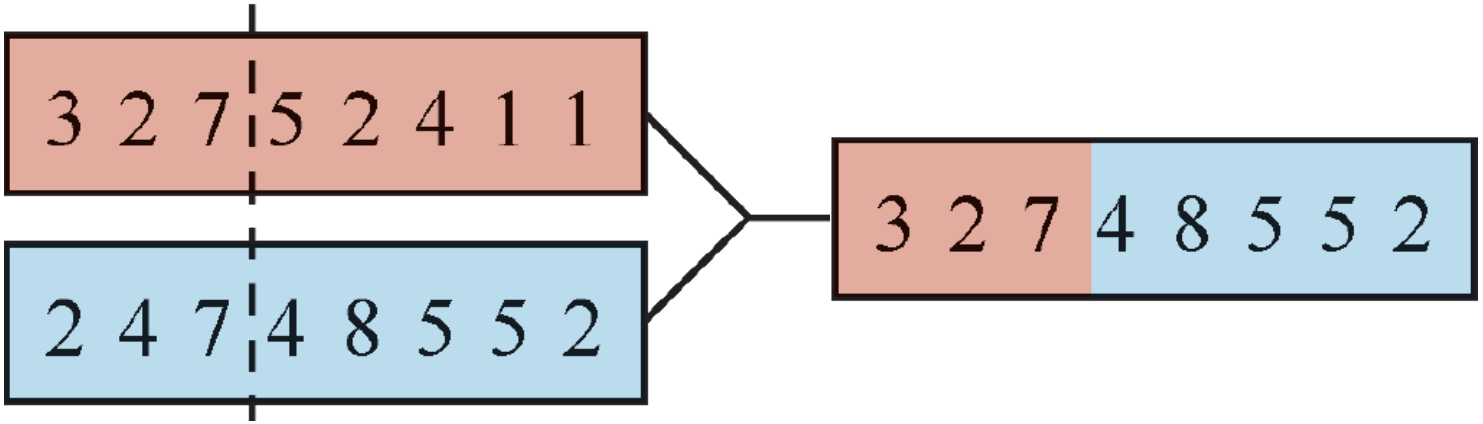
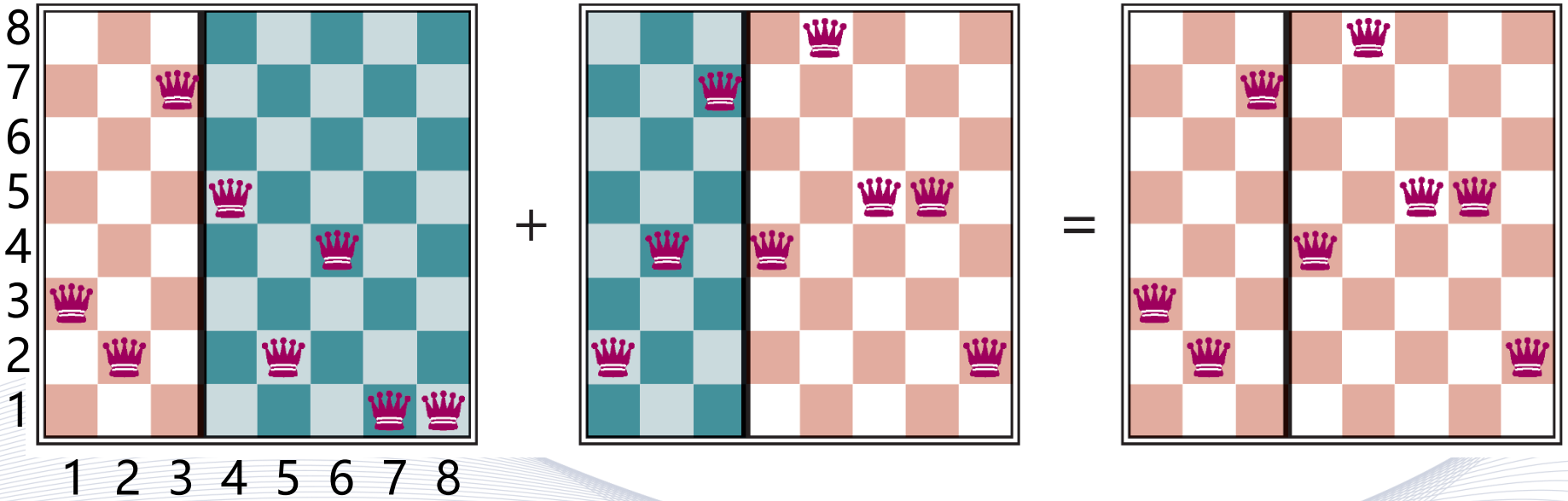


图 4-6　遗传算法，图示为表示 8 皇后状态的数字字符串。（a）中的初始种群根据（b）中的适应度函数进行排序从而得到（c）中的配对，（d）是产生的后代，（e）是可能发生的突变

4.1.4 遗传算法





4.1.4 遗传算法 –遗传算法的描述

- **遗传算法的步骤：**

① 定义问题和目标函数；

② 选择候选解作为**初始种群**，每个解作为个体用二进制串表示（个体相当于染色体，其中的元素相当于基因）；

③ 根据目标函数，对于每个个体计算适应函数值；

④ 为每一个个体指定一个与其适应值成正比的**被选择概率（繁殖概率）**；

⑤ 根据概率选择个体，所选个体通过**交叉/变异**等操作产生新一代种群；

⑥ 如果找到了解或者某种限制已到，则过程结束；否则转③。

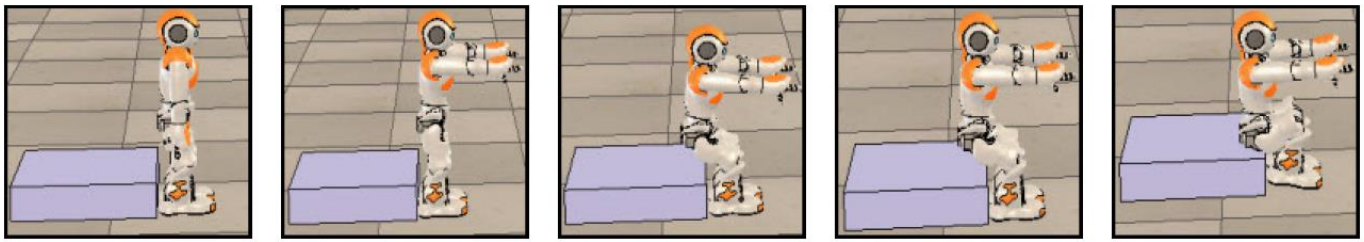
4.1.4 遗传算法 –遗传算法的模式

- **遗传算法的原理：**
  - 遗传算法通过**模式（schema）**来运作——模式是指其中某些位未确定的子串：
    - 能够匹配模式的字符串称为该模式的**实例**；
    - 如果一个模式的实例的**平均适应值超过均值**，则种群内这个模式的实例数量**会随时间而增长**；
    - 遗传算法在模式和解的有意义成分相对应时才会工作得最好，因此字符串顺序很重要，**字符串的子串最好具有明确的物理意义**。

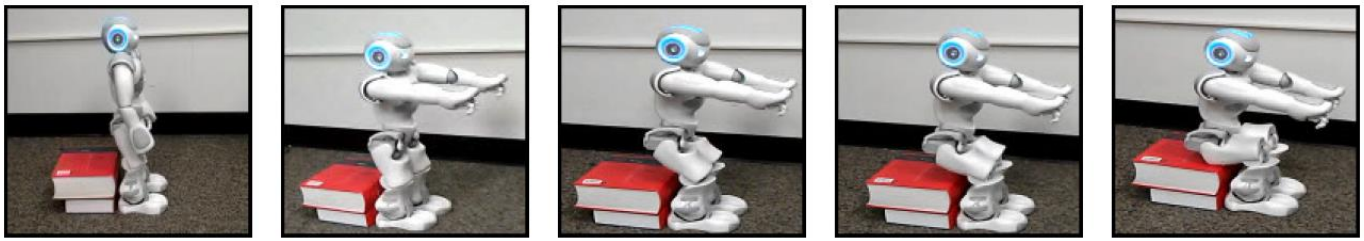
4.1.4 遗传算法 –其它应用

- 优化机器人姿态相关参数（以NAO机器人为例）：
  - 机器人的动作由一系列时序参数控制（**子串代表一个部位的动作**）；
  - 通过提取和固定关键帧的参数（**作为Parent**），优化整个时序的参数；
  - 使机器人更好地执行某些动作，如踢球更远，摔倒后站起来更快等。

	Right Arm					Left Arm					Right Leg						Left Leg					
$Parent_1$	−28	58	84	0	89	28	58	−86	0	−89	−1	−100	0	4	0	0	−1	−100	0	4	0	0
$Parent_2$	−51	18	84	101	42	44	−2	−82	−101	−38	−38	−67	82	9	−9	13	−38	−80	87	9	−13	20
Crossover	−28	58	84	0	89	44	−2	−82	−101	−38	−1	−100	0	4	0	0	−1	−100	0	4	0	0
Mutation	0	0	0	+3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Child	−28	58	84	+3	89	44	−2	−82	−101	−38	−1	−100	0	4	0	0	−1	−100	0	4	0	0



(a) Initial Pose (b) Zero Pose (c) Intermediate Pose (d) Intermediate Pose (e) Initial Sitting Pose



(f) Initial Pose (g) Zero Pose (h) Intermediate Pose (i) Intermediate Pose (j) Initial Sitting Pose

- 局部搜索算法和最优化问题
- 连续空间中的局部搜索
- 使用不确定动作的搜索
- 使用部分可观察信息的搜索
- 联机搜索智能体和未知环境

4.2 连续空间中的局部搜索

- 连续空间的难点：很多搜索算法不能处理连续的状态和动作空间，因为连续空间里的分支因子是无限的。
- 例如：
  - 假设希望新建3个机场，使得地图上每个城市到其最近机场的直线距离平方和最小。状态空间定义为3个机场的坐标：(x_1, y_1)、(x_2, y_2) 和 (x_3, y_3)。这是一个六维空间，也可以说状态由6个**变量**（variable）定义。
  - 一般地，状态定义为n维向量$\boldsymbol{x}$，在这个空间中移动对应于移动地图上的一个或多个机场。
  - 对于任一特定状态，一旦计算出最近城市，目标函数$f(\boldsymbol{x}) = f(x_1, y_1, x_2, y_2 x_3, y_3)$的计算就会变得相对容易。设$C_i$是最近机场（在状态$\boldsymbol{x}$下）为机场 i 的城市集合，则有：

$$f(\boldsymbol{x}) = f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2$$

4.2 连续空间中的局部搜索

难度在于：

- 位置是连续的，每个位置的邻域很难定义
- 随着机场位置的变化，距离机场最近的城市可能突然变化，

  位置1： 最近城市为a,b,c城市

  位置2： 虽然只改变了一点位置，但是最近的城市变为d,b,c

4.2 连续空间中的局部搜索

- **解决方法：离散化**
  - 避免连续性问题的一种简单途径就是将每个状态的邻接状态**离散化**。
  - 例如，一次只能将一个飞机场按照x方向或y方向移动一个固定的量$\pm\delta$。有**6个**变量，每个状态就有**12个**后继。这样就可以应用之前描述过的局部搜索算法。
  - 或者，可以直接应用随机爬山法和模拟退火。这样算法随机选择后继，通过随机生成长度为δ的向量来完成。

4.2 连续空间中的局部搜索

- **解决方法：计算梯度**
  - 很多方法都试图利用地形图的**梯度**来找到最大值。目标函数的梯度是向量∇f，它给出了最陡斜面的长度和方向，对于上述问题，则有：

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3} \right)$$

  - 在某些情况下，可以通过解方程▽f=0找到最大值（如只建一个飞机场，解就是所有城市坐标的算数平均，凸问题）。
  - 然而，在很多情况下，该等式不存在闭合解。例如，要建三个机场时，梯度表达式依赖于当前状态下哪些城市离各个机场最近。这意味着我们只能在**最近城市不变的一个局部区域**内计算梯度（而不是全局地计算），此时至少我们可以用梯度法找到一个局部极值。

4.2 连续空间中的局部搜索

- 给定梯度的局部正确表达式，可以通过下述公式更新当前状态来完成最陡上升爬山法:

$$\boldsymbol{x} \leftarrow \boldsymbol{x} + a\nabla f(\boldsymbol{x})$$

- 其中α是很小的常数，称为**步长**。在一些情况下，目标函数可能无法用微分形式表示（经常不可导）。例如，机场位置的特定集合的值要由某个大型经济仿真程序包来决定（甚至写不出公式）。在这些情况下，**可以通过评估每个坐标上小的增减带来的影响来绝对所谓的经验梯度**（试出来的梯度，而不是算出来的）。在离散化的状态空间中，经验梯度搜索和最陡上升爬山法是一样的。

- 局部搜索算法和最优化问题
- 连续空间中的局部搜索
- 使用不确定动作的搜索
- 使用部分可观察信息的搜索
- 联机搜索智能体和未知环境

4.3 使用不确定动作的搜索

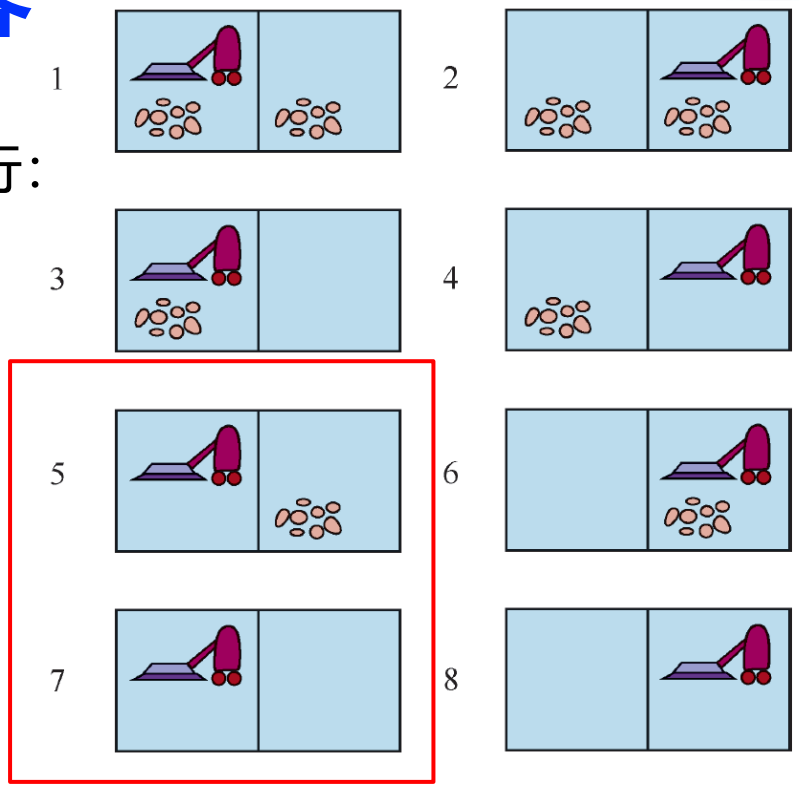
- **不确定性的引入：**
  - 前述问题：假设环境
    - 完全可观测：知道环境的完整状态
    - 确定的：行动的结果是固定的
  - 进入现实世界：不确定性动作
    - 部分可观测：智能体不确定处于什么状态
    - 非确定性：智能体不知道在执行某个动作后将转移到什么状态
    - 两者都有

4.3 使用不确定动作的搜索

感知的必要性

- 如果环境是**部分可观察的或是不确定的**（也可能两者都有），**感知信息**就变得十分有用，感知信息告知智能体某一行动的结果到底是什么。
- 每个感知信息都可能缩小智能体可能的状态范围，这样也就使得智能体更容易到达目标。
- 由于无法**预知**感知信息，智能体的未来行动依赖于未来的感知信息。所以问题的解**不是一个序列，而是一个应急规划（也称作策略）**，应急规划描述了根据接收到的感知信息来决定行动。
- 这在现实世界中很常见，智能体要依赖传感器确定现在的状态。

4.3.1 不稳定的吸尘器世界



- 不稳定的吸尘器世界中，吸尘如下进行：
  - 在一块脏的区域中进行“吸尘”可以使该区域变得干净，有时也会同时清洁邻近区域；
  - 如果是干净区域进行此动作有可能使脏东西掉在地毯上。

动作的结果是不确定的！

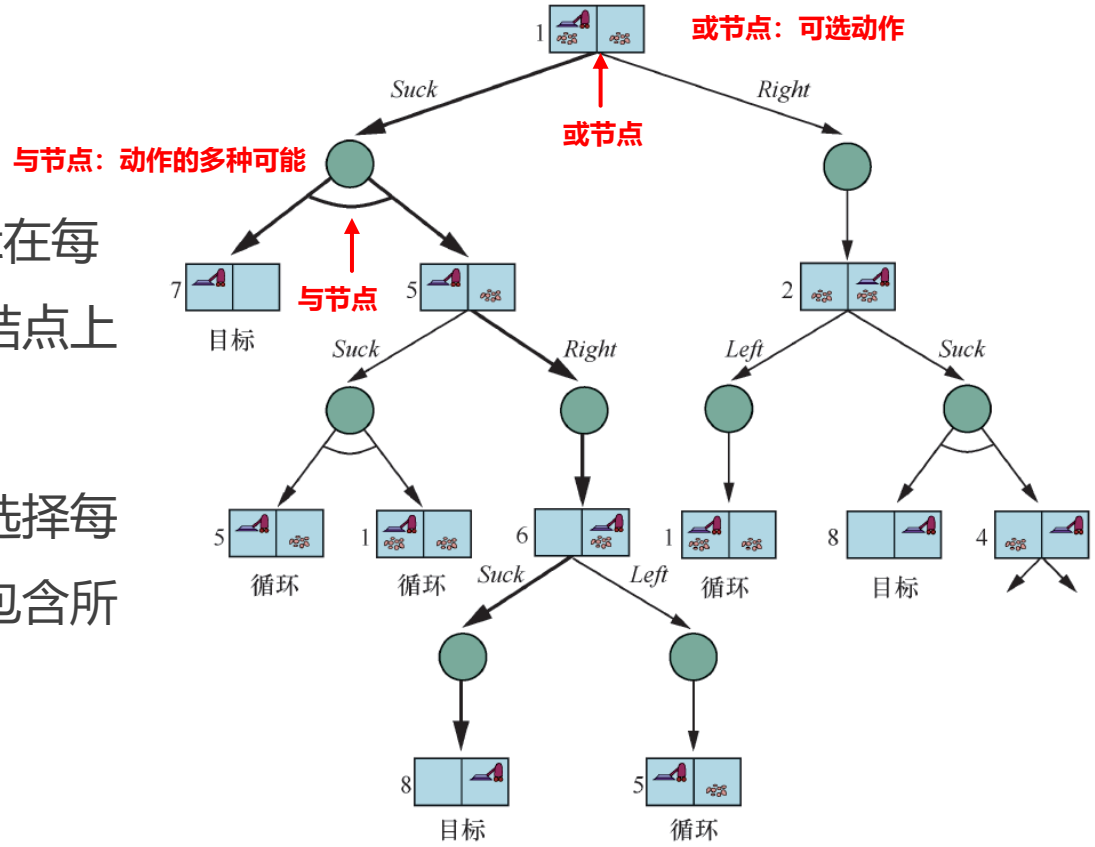
- 返回一组可能状态：状态1下实施吸尘结果为状态集{5,7}，因为他不知道隔壁方格中有没有被一起清理了
- 从状态1开始，没有一个序列可以求解问题。需要一个如下所示的应急规划：

$$[Suck, \textbf{if } State = 5 \textbf{ then } [Right, Suck] \textbf{ else } []]$$

搜索树变成与或树

4.3.2 与或搜索树

- **与或搜索树：**
  - 或结点（动作节点）：Agent在每个状态下的动作选择（在或结点上选择一个活动）
  - 与结点（状态节点）：环境选择每个行动的后果（在与结点上包含所有可能后果）



- 右图示例：
  - 不稳定的真空吸尘器世界搜索树的**前两层**。
  - **或节点上**，必须选择某个动作的。**与节点**上，每个结果都必须处理，结果分支间用弧线连接。

4.3.2 与或搜索树

- **与或图搜索的深度优先递归算法：**
  - 非确定性环境生成的与或图的搜索算法。解是一个条件规划，**它考虑每一个非确定性的结果**，并为每个结果制定规划：

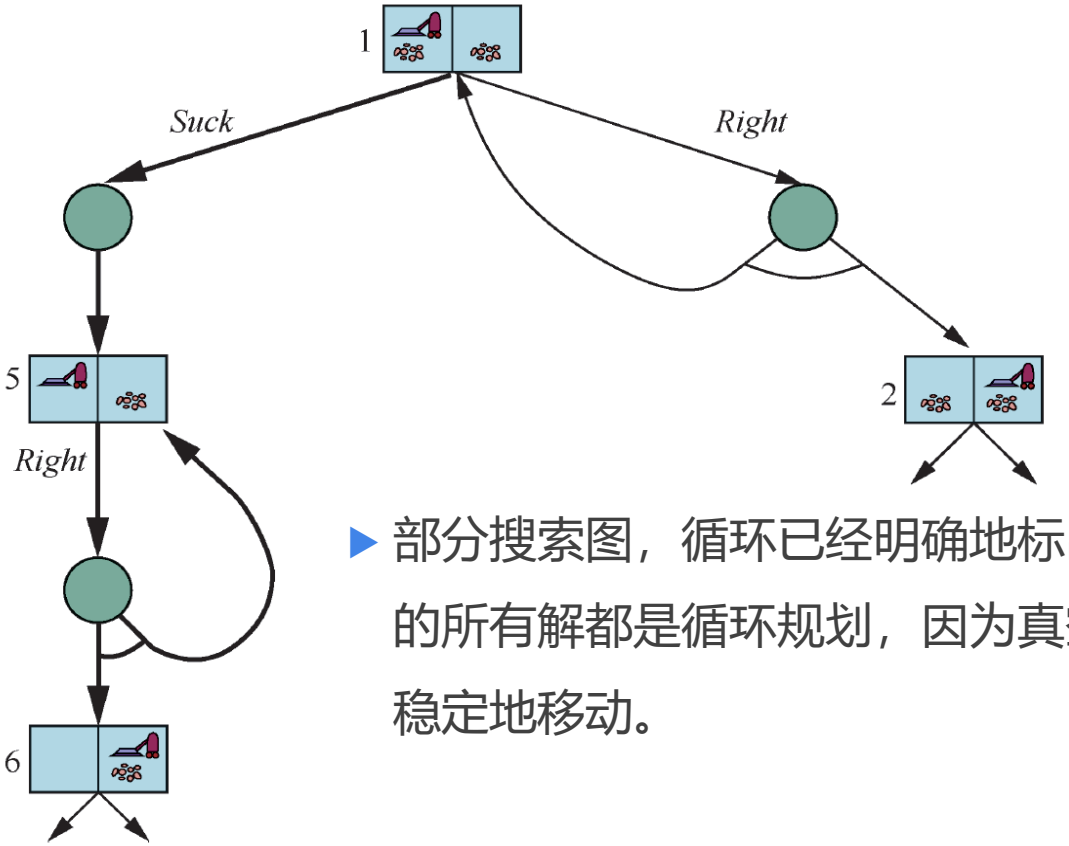
```
function AND-OR-SEARCH(problem) returns 一个条件规划或failure
  return OR-SEARCH(problem, problem.INITIAL, [])

function OR-SEARCH(problem, state, path) returns 一个条件规划或 failure
  if problem.IS-GOAL(state) then return 空规划
  if IS-CYCLE(state, path) then return failure
  for each action在 problem.ACTIONS(state)中 do
      plan ← AND-SEARCH(problem, RESULTS(state, action), [state] + [path])
      if plan ≠ failure then return [action] + [plan]
  return failure

function AND-SEARCH(problem, states, path) returns 一个条件规划或 failure
  for each s_i in states do
      plan_i ← OR-SEARCH(problem, s_i, path)
      if plan_i = failure then return failure
  return [if s_1 then plan_1 else if s_2 then plan_2 else ⋯ if s_{n-1} then plan_{n-1} else plan_n]
```

4.3.3 不断尝试

- 动作会失败的场合
  - 不稳定吸尘器世界：有些时候移动动作会失败，Agent在原地不动。
  - 产生**循环解**：$[Suck, \textbf{while } State = 5 \textbf{ do } Right, Suck]$



- 部分搜索图，循环已经明确地标出。这个问题的所有解都是循环规划，因为真空吸尘器无法稳定地移动。

4.3.3 不断尝试

- 什么时候可以将循环规划作为解?
- 完全可观测，**非确定性**：可行

- 部分可观测的，**确定性的**：不适合用循环规划

- 考虑吸尘器右移失败的两种情况：
  - 地面光滑：环境完全可观测，行动有非确定性；**值得到所尝试几次，可以循环**
  - 皮带断裂：环境部分可观测，行动确定。**怎么尝试也没用，不适合循环**

- 局部搜索算法和最优化问题
- 连续空间中的局部搜索
- 使用不确定动作的搜索
- 使用部分可观察信息的搜索
- 联机搜索智能体和未知环境

4.4.1 无观察信息的搜索

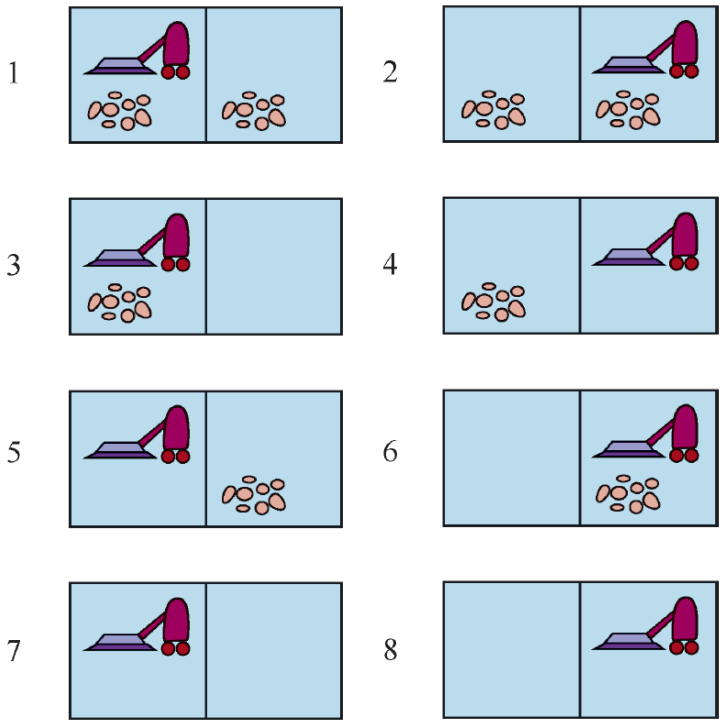
- 无观察信息的情况（无传感器）
  - 如果Agent感知不到任何信息，称之为无传感问题，有时也称为相容问题。
  - 假设Agent知道世界的地图情况，但并不知道自己的位置和地上垃圾的分布。

① 初始状态可能是{1，2，3，4，5，6，7，8}中的一个

② 执行Right会导致Agent在{2，4，6，8}的某一个状态

③ 执行Suck会导致状态集{4，8}

④ 最后，执行{left，Suck}将彻底确保Agent到状态7



4.4.1 无观察信息的搜索

- **无传感器问题的定义：**
  - 信念状态：
    - **智能体认为他可能位于的状态集合称为信念状态（belief state）。**
      如果原问题P有N个状态，那么信念状态问题有2^N个信念状态。
  - 初始状态：
    - 初始信念状态包含P中的所有状态。
  - 动作：
    - $\text{ACTIONS}(b) = \bigcup_{s \in b} \text{ACTIONS}_P(s)$
  - 转移模型：转移模型是一个集合到另一个集合的转移
    - $b' = \text{RESULT}(b, a) = \{s' : s' = \text{RESULT}_P(s, a) \text{ 且 } s \in b\}$

b和b'都是集合

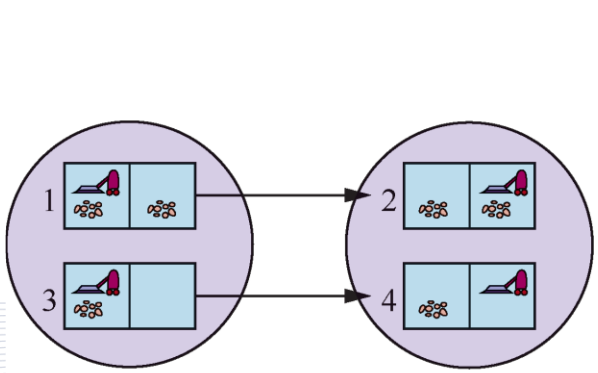
信念状态是建模不确定性的有效手段

4.4.1 无观察信息的搜索

- **无传感器问题的定义：**
  - 目标测试：
    - 如果信念状态中的所有状态都满足 $\text{IS-GOAL}_P(s)$，则必定达到了目标。
  - 动作代价：
    - 简化：假定同一动作在所有状态下具有相同代价。

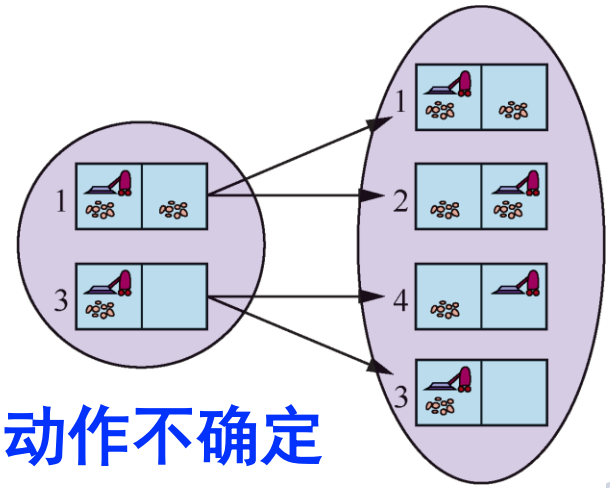
4.4.1 无观察信息的搜索

- **无传感器问题的特点：**
  - 对于确定性动作， b' 不会大于 b （信念状态数量逐渐收敛），而对于非确定性动作， b' 可能会大于 b （越行动越不确定）；
  - （a）预测在无传感器真空吸尘器世界执行确定性动作Right后的下一个信念状态；（b）在光滑的无传感器真空吸尘器世界中的同一状态下执行同一动作的预测:



动作确定

（a）

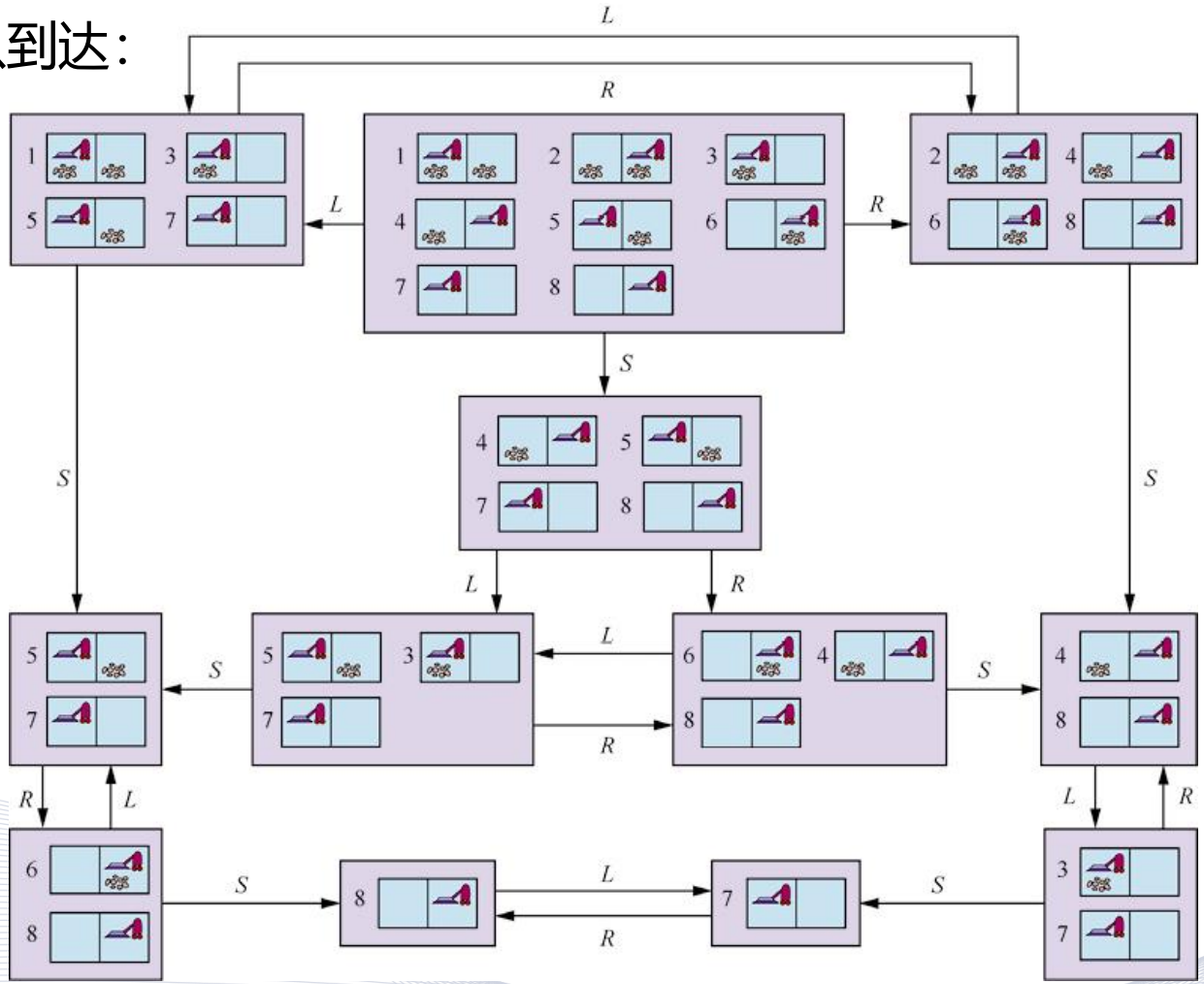


动作不确定

（b）

4.4.1 无观察信息的搜索

- **12个可达的信念状态**，在$2^8 = 256$种可能信念状态中只有12种可以到达：



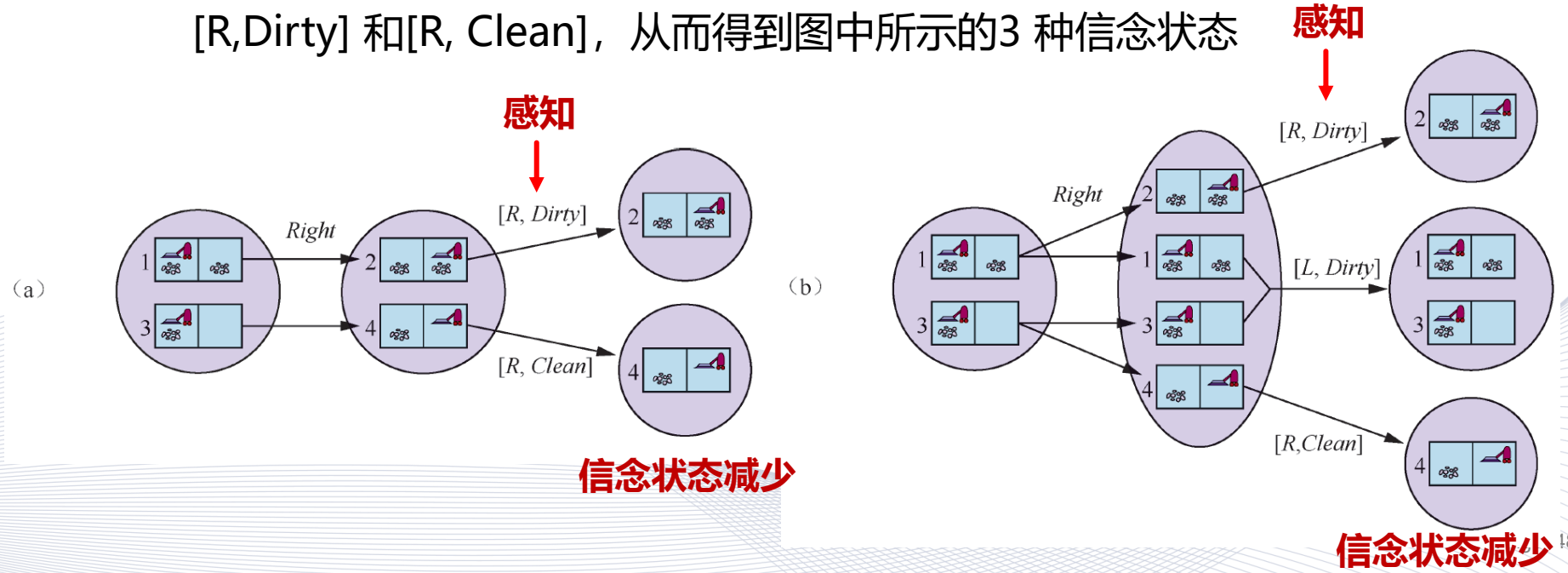
随着行动信念状态会收敛

4.4.2 有部分观察信息的搜索

- **引入传感器：有部分观察信息的搜索**
  - **吸尘器世界中的传感器：**
    - 位置传感器：在左侧方格中生成感知L，右侧方格中生成感知R
    - 灰尘传感器：当前方格内有灰尘时生成感知Dirty，否则生成Clean。
  - **加入传感器后的转移模型：**
    - 预测：与无传感器问题相同，计算由动作所导致的信念状态。
    - 观察：在预测的信念状态下可以观测到的感知集合。
    - 更新：为每个可能感知计算其可能得到的信念状态。

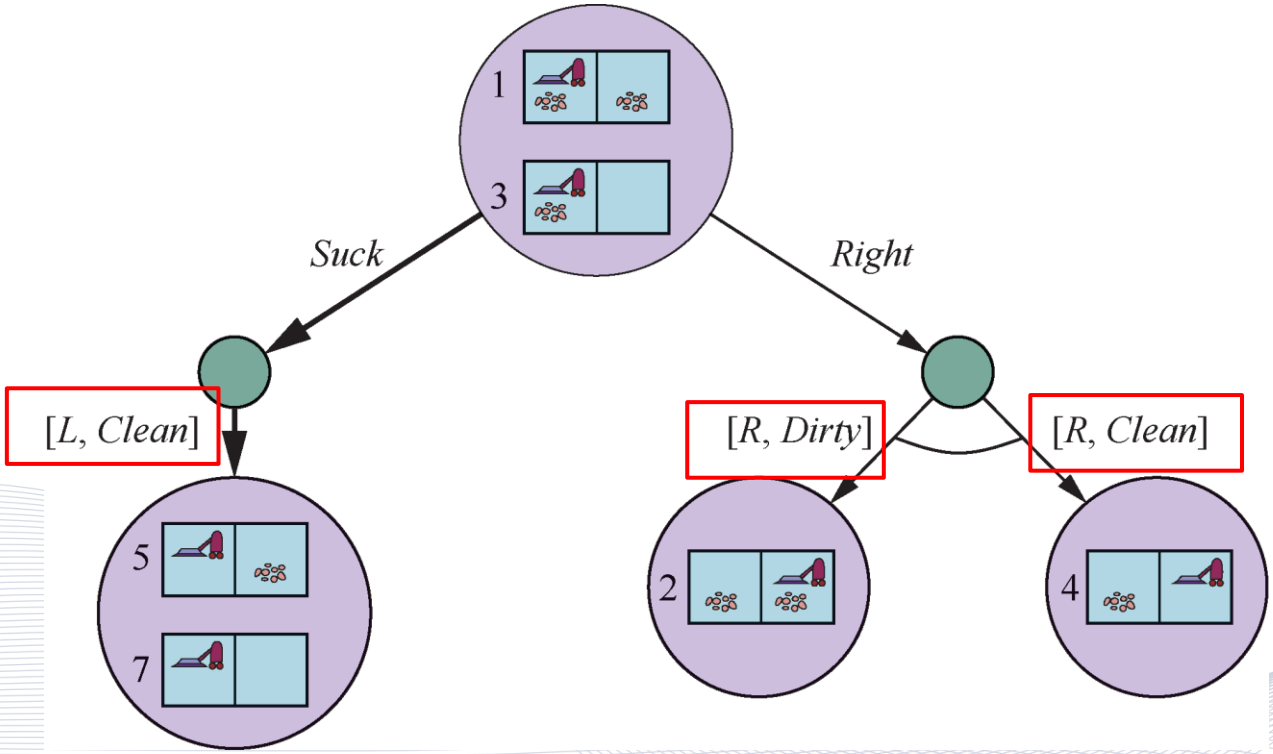
4.4.2 有部分观察信息的搜索

- **示例：**
  - （a）确定性世界中，在初始信念状态下执行Right动作，所得到的新的预测信念状态有两个可能的物理状态；对于这些状态，可能的感知是[R, Dirty] 和[R,Clean]，从而得到两种信念状态。
  - （b）光滑世界中，在初始信念状态下执行Right 动作，所得到的新的信念状态具有4 个物理状态；对于这些状态，可能的感知是[L, Dirty]、[R,Dirty] 和[R, Clean]，从而得到图中所示的3 种信念状态



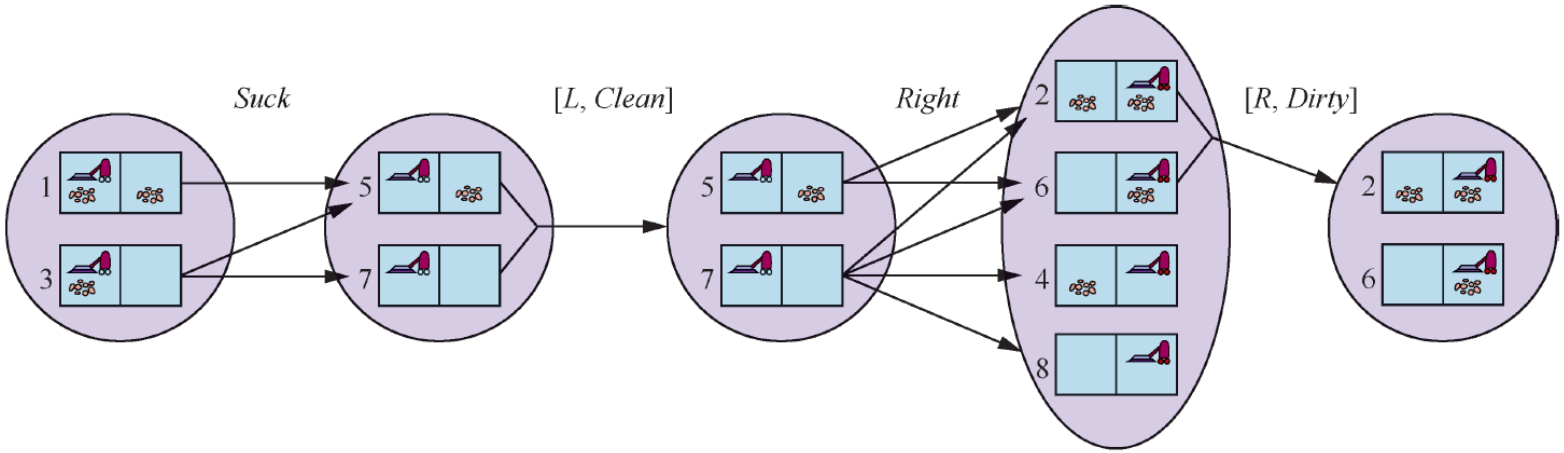
4.4.3 求解部分可观察环境中的问题

- **局部感知吸尘器世界的搜索树**，即：局部感知真空吸尘器世界问题的第一层与或搜索树，Suck 是解序列中的第一个动作：
- 解是一个条件规划，与结点的边上加上了传感信息。
- 相比无感知信息的与或树更小，因为传感信息剪枝了与节点



4.4.4 部分可观察环境中的智能体

- 例：幼儿园中，吸尘器世界中信念状态的维护，任一方格在任一时刻都可能变脏，除非Agent在那一时刻吸尘：



- 局部搜索算法和最优化问题
- 连续空间中的局部搜索
- 使用不确定动作的搜索
- 使用部分可观察信息的搜索
- 联机搜索智能体和未知环境

4.5 联机搜索智能体和未知环境

- **脱机搜索智能体**：实现对问题计算出解决方案，然后再进入现实世界执行解决方案；
- **联机搜索智能体**：通过计算和行动交叉完成；
  - 智能体首先采取一个行动；
  - 然后观察问题的环境并且计算下一个行动。

- **脱机搜索**：先通常需要考虑所有可能发生的情况而制定指数级大小的偶发事件处理计划；
- **联机搜索**：只需要考虑实际发生的情况。

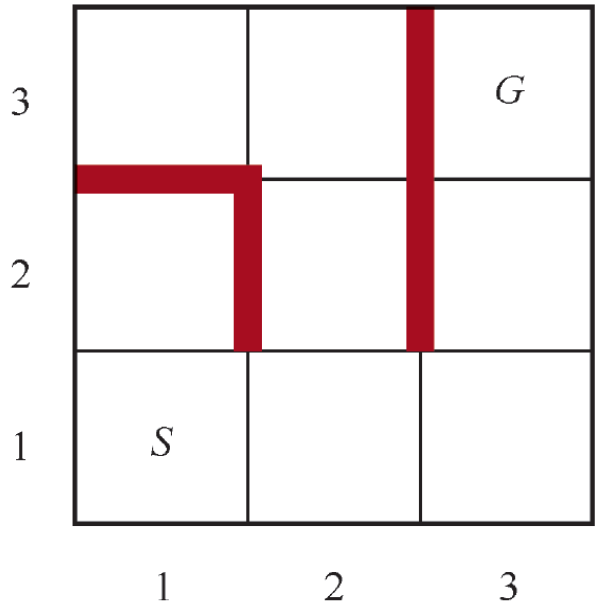
4.5 联机搜索智能体和未知环境

- 应用领域：
  - 动态或半动态的问题领域，对于停留不动或者计算时间太长都会有惩罚的领域；
  - 不确定的领域，可以集中精力在实际发生的偶然事件上，而不需要规划很多不太可能发生的情况，甚至是一个完全随机的领域。
- 必须用联机搜索的搜索问题：
  - 例如将一个智能体放在新建的大楼里，要求它探索大楼，并且绘制出一张从A到B的地图。
- 联机搜索问题必须依靠智能体执行**行动**解决，而不是纯粹的计算过程。

4.5 联机搜索智能体和未知环境

联机搜索问题的定义

- 联机智能体仅知道如下信息：
  - ACTIONS(s)，返回状态s下可能进行的行动列表；
  - 单步耗散函数$c(s,a,s')$，执行一个动作的代价
  - GOAL-TEST(s)。
  - 智能体不知道Result(s,a)！



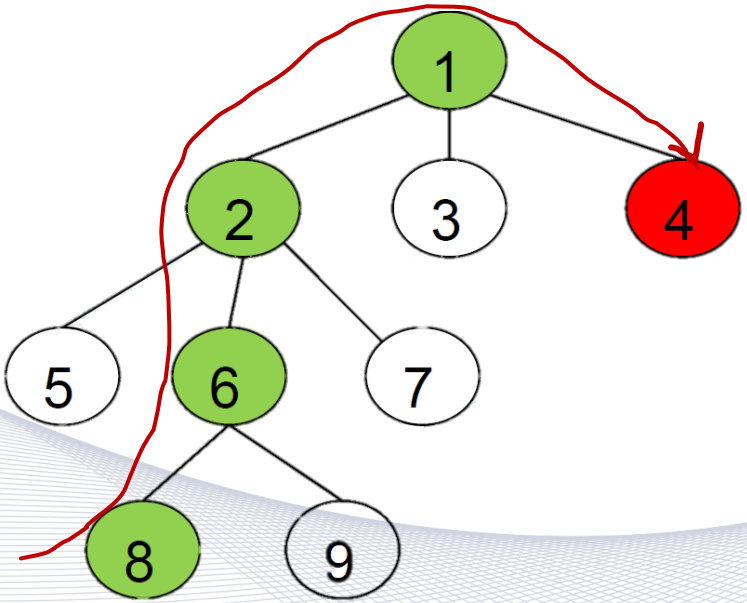
智能体不能知道执行up会到（1,2）

注意：智能体不能知道一个动作的结果，除非它实际尝试了该状态下的**行动**。

4.5.1 联机搜索智能体

联机搜索智能体的特点

- 联机搜索算法：规划和行动交叉进行。
- 脱机搜索算法，如A*：可以在状态空间的一部分扩展一个结点，然后马上又在空间的另一部分扩展另一个结点。
  - 因为脱机算法结点扩展涉及的是**模拟**的而不是实际的**行动**：
  - **在现实世界中，智能体无法回溯**！（没法传送的之前任意一个位置）



4.5.1 联机搜索智能体

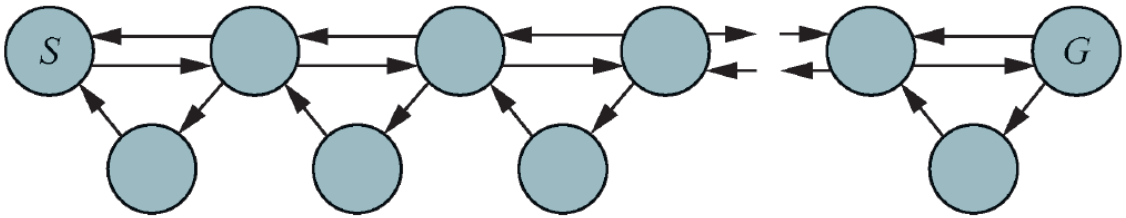
- 使用深度优先探索的在线搜索智能体。智能体只有在每个动作都可以被其他动作“撤消”（原路返回）的状态空间中才能安全地探索，然而现实世界中，智能体可能会卡在某个状态：

```
function ONLINE-DFS-AGENT(problem, s′) returns 一个动作
  persistent: s，a，之前的状态和动作，初始为空
              result，将(s, a)映射到s′的表，初始为空
              untried，将s映射到未探索动作列表的表
              unbacktracked，将s映射到未回溯状态队列的表
  if problem.IS-GOAL(s′) then return stop
  if s′是一个（不在untried中的）新状态 then untried[s′] ← problem.ACTIONS(s′)
  if s不空 then
      result[s, a] ← s′
      将s添加到unbacktracked[s′]的队尾
  if untried[s′]为空 then
      if unbacktracked[s′]为空 then return stop
      s′, a ← null, 使得result[s′, b] = POP(unbacktracked[s′])的动作b
  else a ←POP(untried[s′])
  s ← s′
  return a
```

4.5.2 联机局部搜索

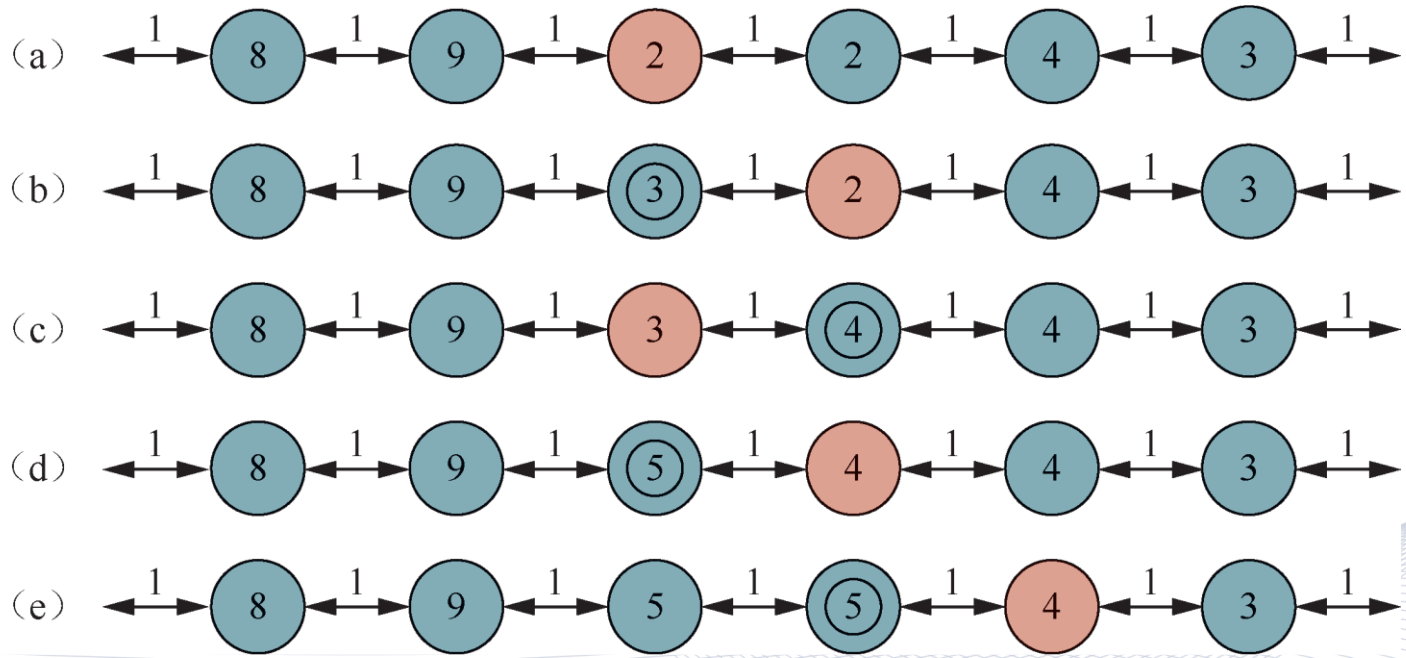
联机搜索方法

- 爬山法搜索在内存中只存放一个当前状态：
  - 因此，它是一个**联机搜索算法**；
  - 会使智能体呆在局部极大值上而无处可去。
- 改进方法：
  - 随机重新开始是不可能的，因为智能体不能把自己传送到一个新的状态；
  - 不妨考虑随机游走，（但随机游走通常需要耗费指数级的步骤来寻找目标）。

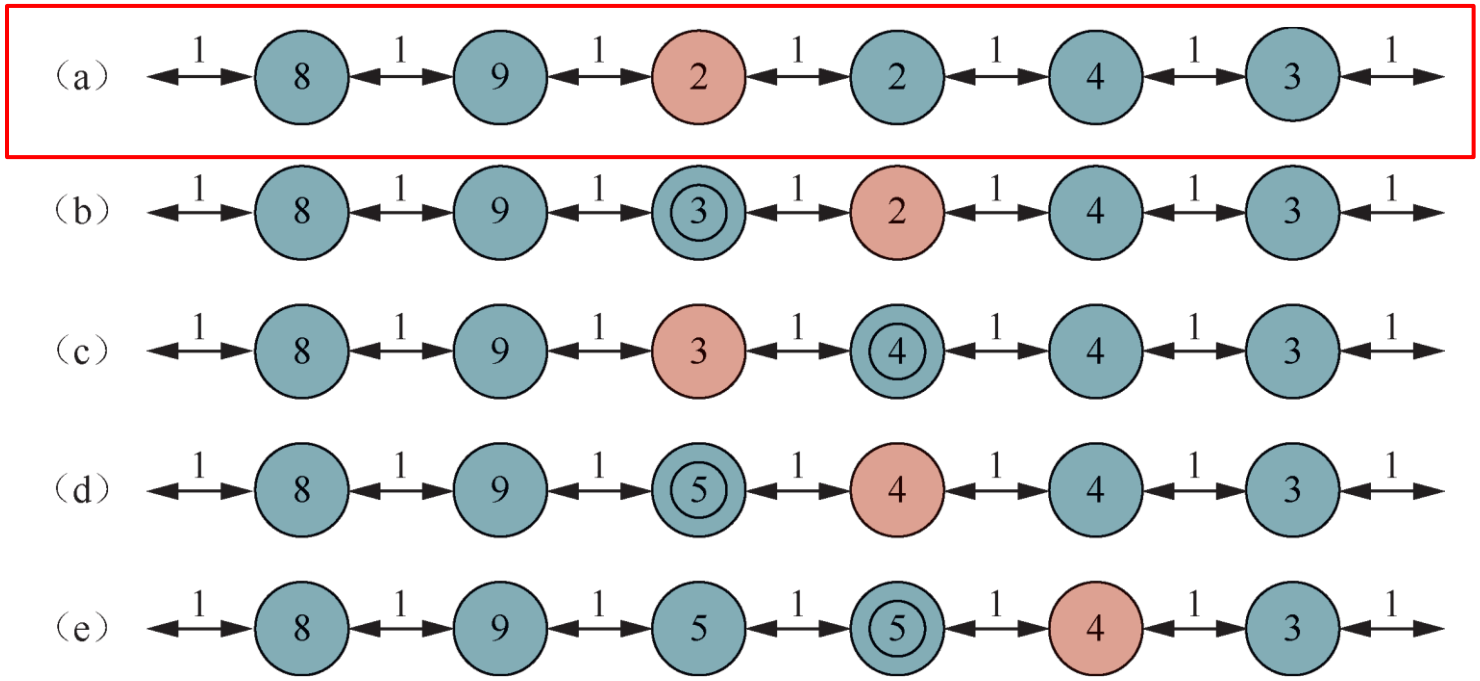


4.5.2 联机局部搜索 – 在线学习A*

- 存储从每个已经访问的状态s达到目标的代价的“当前最佳估计”H(s)。
- H(s)的初始值可以是启发式h(s)，**根据智能体探索状态空间中获得的经验再对H(s)进行更新。**
- 每个状态都标有H(s)，即到达目标的当前代价估计值，连接的动作代价为1。红色表示智能体的位置，双圈表示每次迭代所更新的代价估计值

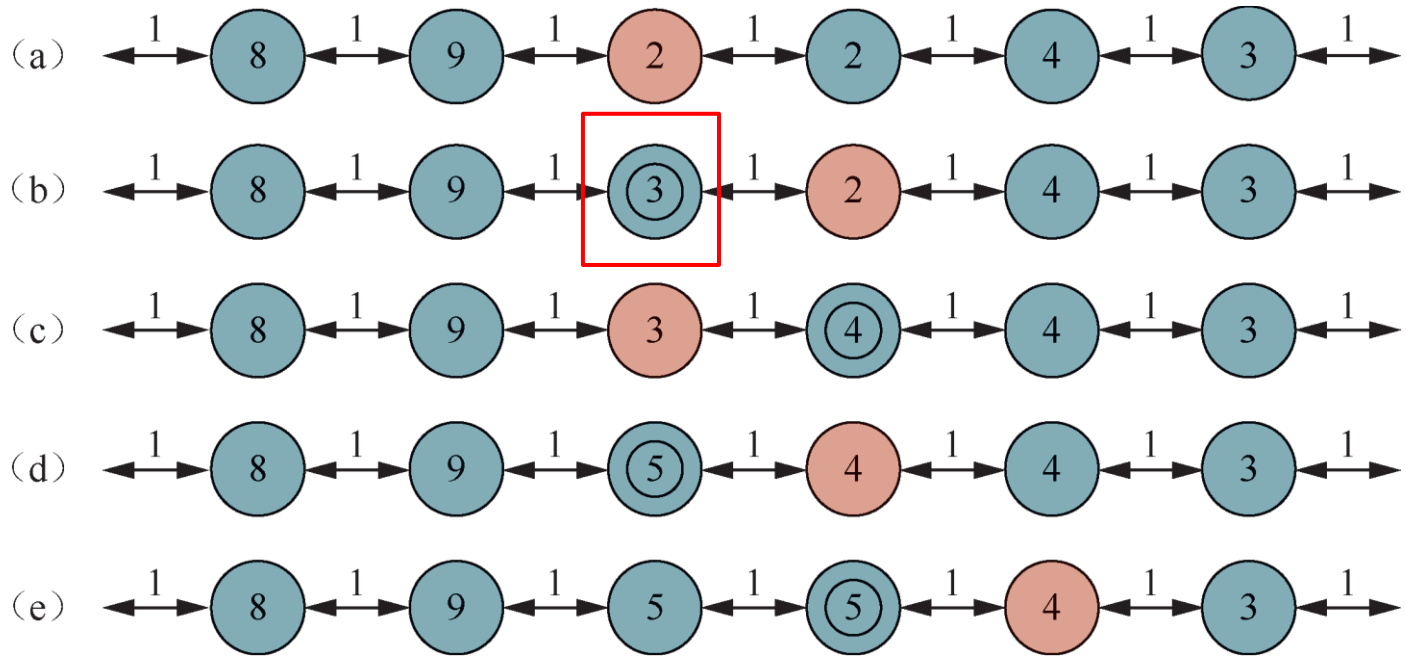


4.5.2 联机局部搜索 -提高爬山法算法的内存利用率



- （a）中，智能体陷入了位于红色状态的局部极小值。智能体不应该停留在原地，而应该根据其邻居节点的当前代价估计值选择到达目标的最优路径。
- 两个选择：经由邻居节点s ‘到达目标的估计代价等于到达s’ 的代价加上从s ‘到达目标的估计代价，即c(s, a, s’) + H(s’)。在这个示例中，有2个动作，估计代价分别为向左1 + 9，向右1 + 2，因此最好向右移动（鼓励探索）。

4.5.2 联机局部搜索 -提高爬山法算法的内存利用率



- 更新离开节点的代价：因为最佳移动的代价为1，而且其结果状态离目标状态至少还有2 步，所以红色状态离目标一定至少还有3 步，所以应该相应地更新红色状态的H。
- 继续上述过程，智能体将再来回移动两次，每次都会更新H 并“拉平”局部极小值，直到它逃逸到右侧。

4.5.2 联机局部搜索 -实时学习A*

- 上述方案的智能体称为**实时学习A***（learning real-time A*）智能体。
  - learning real-time A*, LRTA*;
  - 首先更新刚刚离开的状态的代价估计值;
  - 然后根据当前的代价估计值选择“显然最佳”移动;
  - 在状态s下**尚未尝试的动作**总是被假定为以最少的可能代价，即h(s)直接到达目标。**鼓励智能体去探索新的、可能更有希望的路径。**

- 局部搜索算法和最优化问题
- 连续空间中的局部搜索
- 使用不确定动作的搜索
- 使用部分可观察信息的搜索
- 联机搜索Agent和未知环境
- 本章小结

4.6 本章小结

- 局部搜索算法如爬山法适用于完整状态形式化。
- 局部搜索算法也可以应用于连续空间上的问题求解。
- 遗传算法是维护大量状态种群的随机爬山搜索。新的状态通过变异和杂交产生，杂交把来自种群的状态对结合在一起。
- 在不确定的环境中，Agent可以应用AND-OR搜索来生成应急规划达成目标。
- 当环境是部分可观察时，用信念状态表示Agent可能在的状态集合。

4.6 本章小结

- 标准的搜索算法可以直接应用于信念状态空间进行无感知问题求解，信念状态AND-OR搜索可以解决一般部分可观察问题；
- 探索问题发生在Agent对环境的状态和行动一无所知时。对于可安全探索的环境，联机搜索Agent能够建造地图并且在有解时能够找到目标。根据经验不断修正启发式估计，是一种避免局部极小值的有效方法。

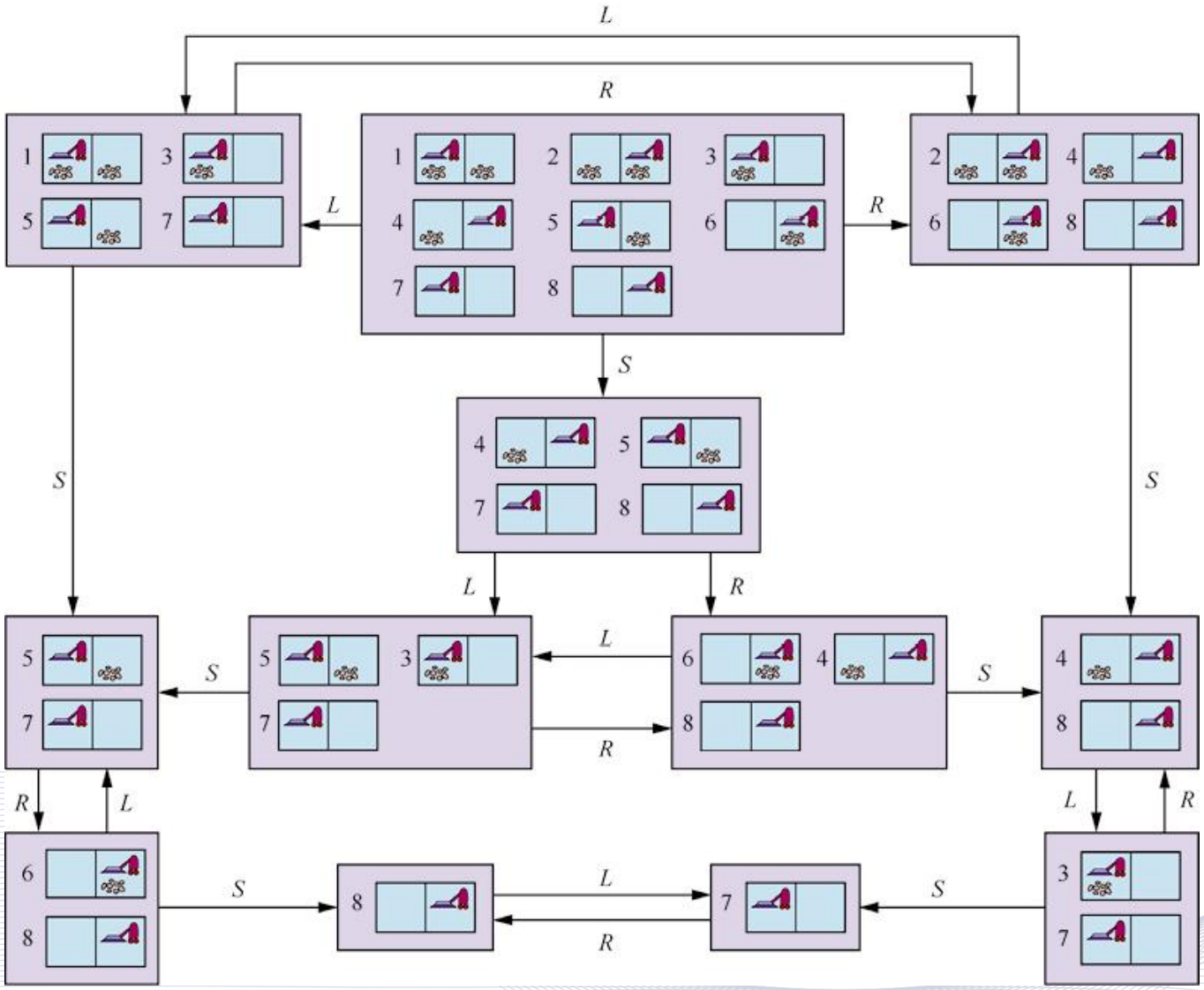
4.6 本章小结 –课程作业

- 生成大量的八数码问题和八皇后问题并用以下算法分别求解（如果可能的话）：爬山法（最陡上升和首选爬山法），模拟退火算法。计算搜索耗散和问题的解决率，并用图对比它们和最优解代价的曲线，对结果进行评估。

4.6 本章小结 –课程作业

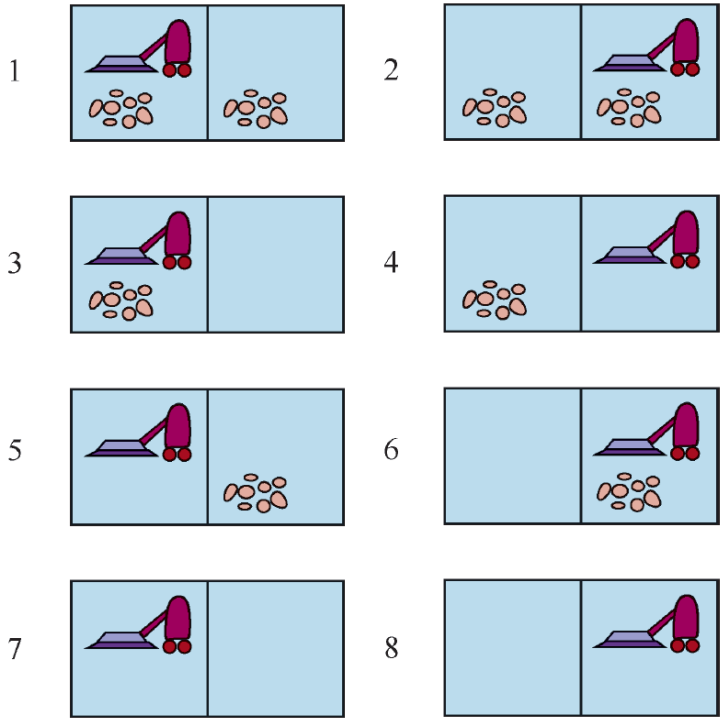
- 引入信念状态来求解无传感信息的搜索问题。如果一个行动序列能够将信念状态b的每一个物理状态都映射到目标状态，那么它就是此无传感信息的问题的解。假设Agent知道b中所有s的h*(s)，即知道完全可观察环境中物理状态s的最优代价。用h*(s)给出无传感信息问题的可采纳的h(b)，并证明它是可采纳的。以下图中的无传感信息的吸尘器问题为例，对该启发式的精确性进行评价。如果使用A*算法，会如何选择扩展结点?

4.6 本章小结 –课程作业


L
R
1 3 5 7
L
1 2 3 4 5 6 7 8
R
2 4 6 8
S
4 5 7 8
S
S
L
R
5 7
S
5 3 7
L
R
6 4 8
S
4 8
R
L
L
R
6 8
S
8
L
R
7
S
3 7

4.6 本章小结 –课程作业

- 考虑不稳定的吸尘器世界的无传感器版本。请给出从初始信念状态{1，2，3，4，5，6，7，8}出发可达的信念状态空间，并说明为什么此题是无解的。



  - 在一块脏的区域中进行“吸尘”可以使该区域变得干净，有时也会同时清洁邻近区域；
  - 如果是干净区域进行此动作有可能使脏东西掉在地毯上。

THANKS