中国科学院
自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

超越经典搜索

张文生

中国科学院自动化研究所
中科院大学人工智能学院

2020年10月23日

讲课内容

一、局部搜索算法

二、连续空间局部搜索

三、不确定动作搜索

四、部分可观察搜索

五、联机搜索与学习

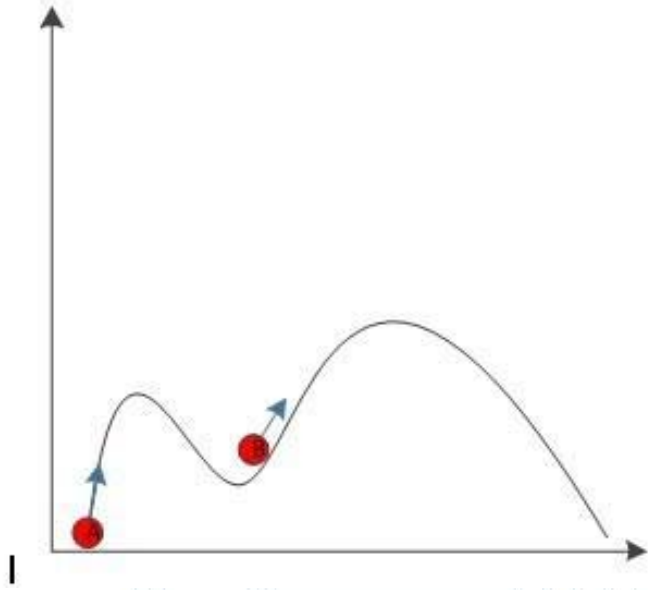
局部搜索算法和最优化问题

- 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节点的选择。当找到目标时，到达此目标的路径就是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。从单个当前节点（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

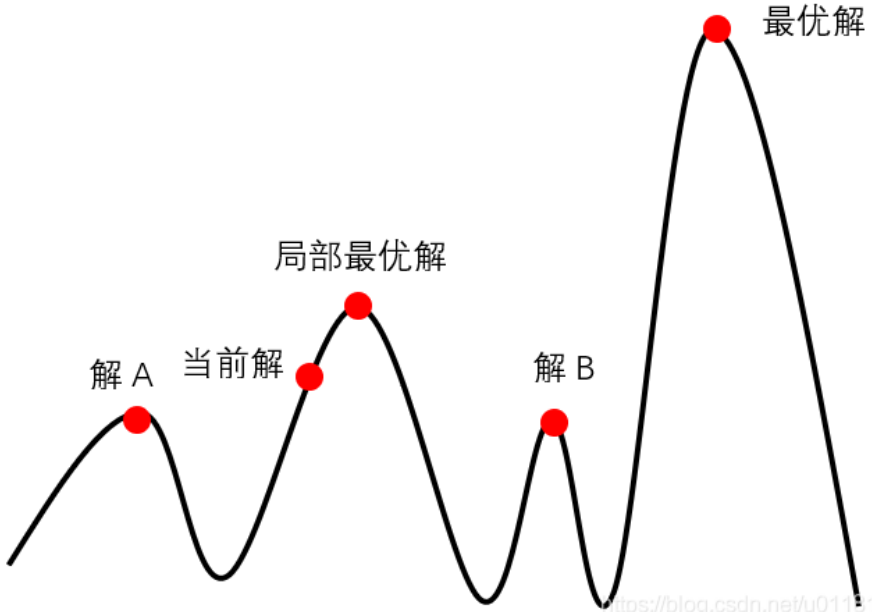
局部搜索



- 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

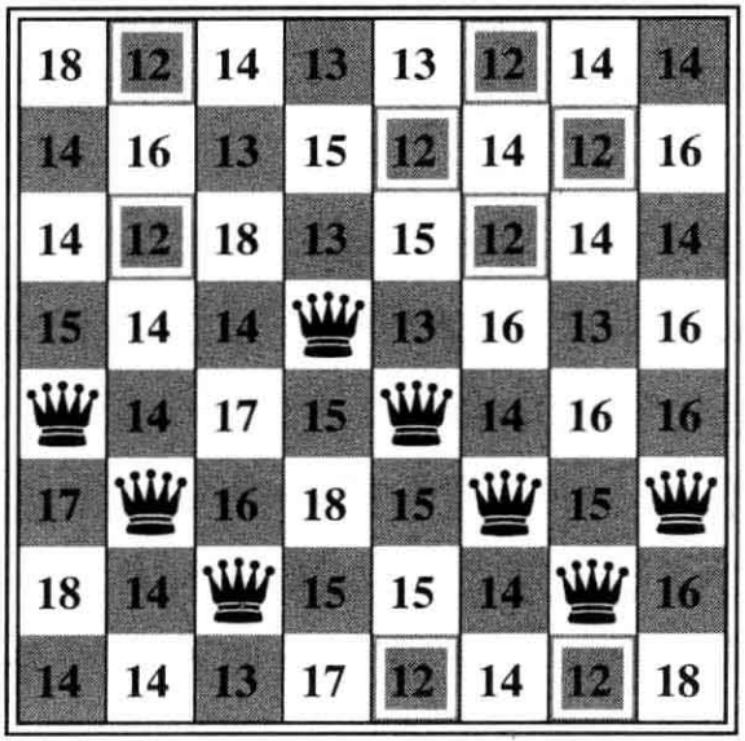
爬山法

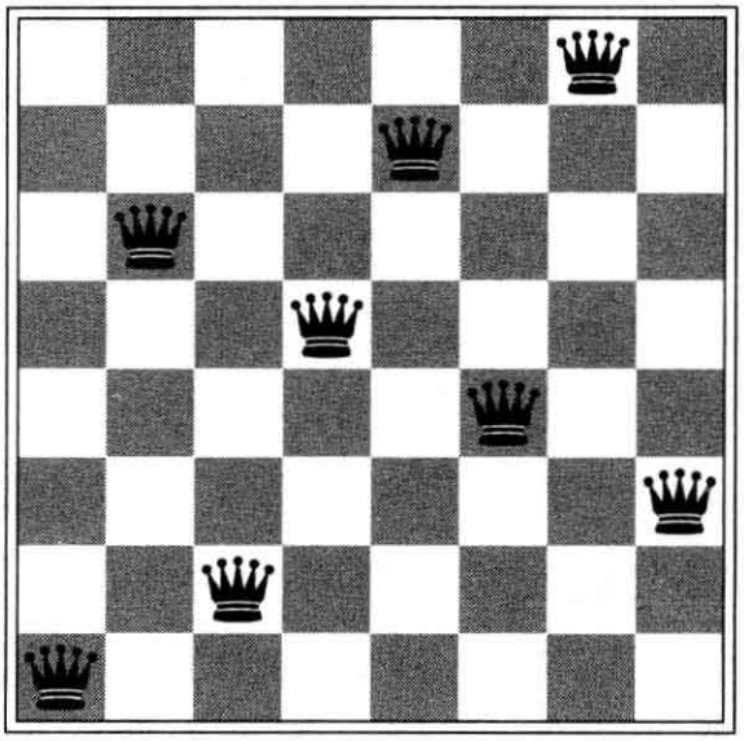
- 特点：
  - ✓ 算法在到达一个“峰顶”时终止，邻接状态中没有比它的值更高的
  - ✓ 算法不维护搜索树，当前结点的数据结构只需要记录当前状态和目标函数值
  - ✓ 爬山法不会考虑与当前状态不相邻的状态



八皇后问题

- 图1只需5步就能到达图2的状态，此时h=1已经很接近解了





- 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：实现了随机爬山法，随机地生成后继结点直到生成一个由于当前结点的后继
  - ✓ 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：实现通过随机生成初试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

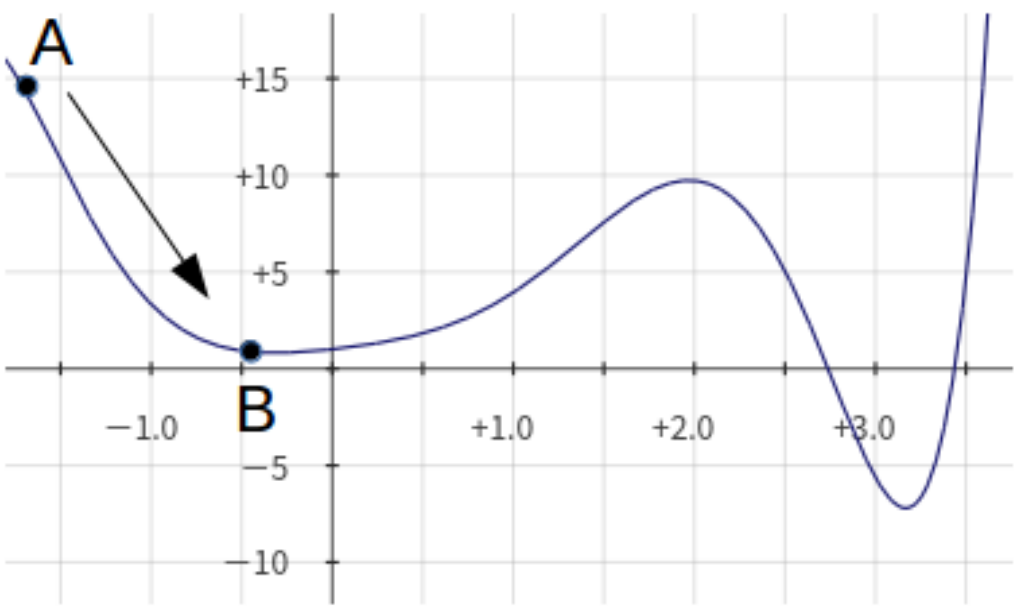
模拟退火搜索

- 简述：

✓ 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）



- 算法流程：

① 如果该移动使情况改善，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△E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

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

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，每一步全部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，他从整个后继列表中选择k个最佳的后继，重复这个过程

- 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

遗传算法

- 遗传算法(generic algorithm/GA)是随机剪枝的变种——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

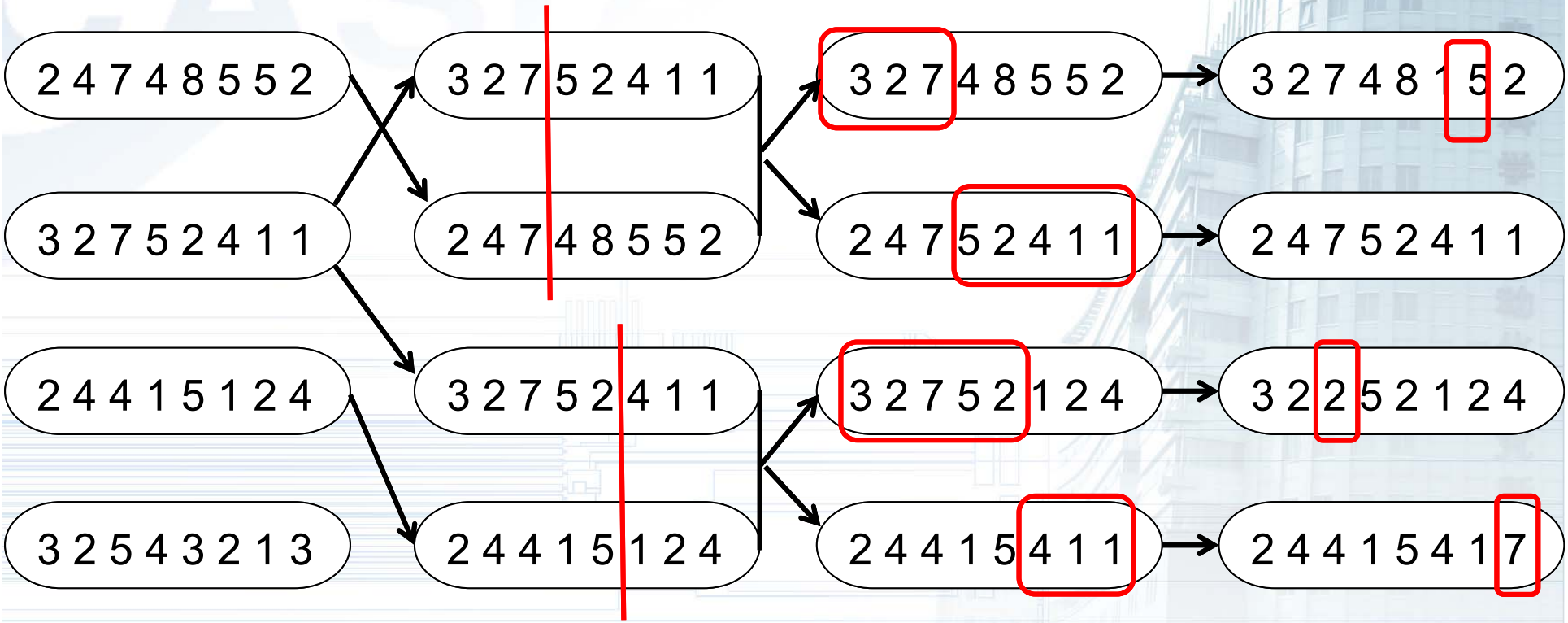
  - ✓ 与剪枝搜索一样，遗传算法也是从k个随机状态并始——这k个状态称为种群，每个状态称为个体。
  - ✓ 个体用有限长的字符串(通常为0/1串)表示
  - ✓ 个状态用其评价函数(适应度函数)给出评价值(适应值)
  - ✓ 随后的操作包括—选择/杂交/变异

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—交交点是在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


2 4 7 4 8 5 5 2
3 2 7 5 2 4 1 1
2 4 4 1 5 1 2 4
3 2 5 4 3 2 1 3
3 2 7 5 2 4 1 1
2 4 7 4 8 5 5 2
3 2 7 5 2 4 1 1
2 4 4 1 5 1 2 4
3 2 7 4 8 5 5 2
2 4 7 5 2 4 1 1
3 2 7 5 2 1 2 4
2 4 4 1 5 4 1 1
3 2 7 4 8 1 5 2
2 4 7 5 2 4 1 1
3 2 2 5 2 1 2 4
2 4 4 1 5 4 1 7
Initial Population
Selection
Crossover
Mutation

遗传算法描述

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

遗传算法的模式

- 遗传算法上述特点可以用模式(schema)来解释—模式是某些位置上的数字尚未确定的一个

  - ✓ 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

讲课内容

一、局部搜索算法

二、连续空间局部搜索

三、不确定动作搜索

四、部分可观察搜索

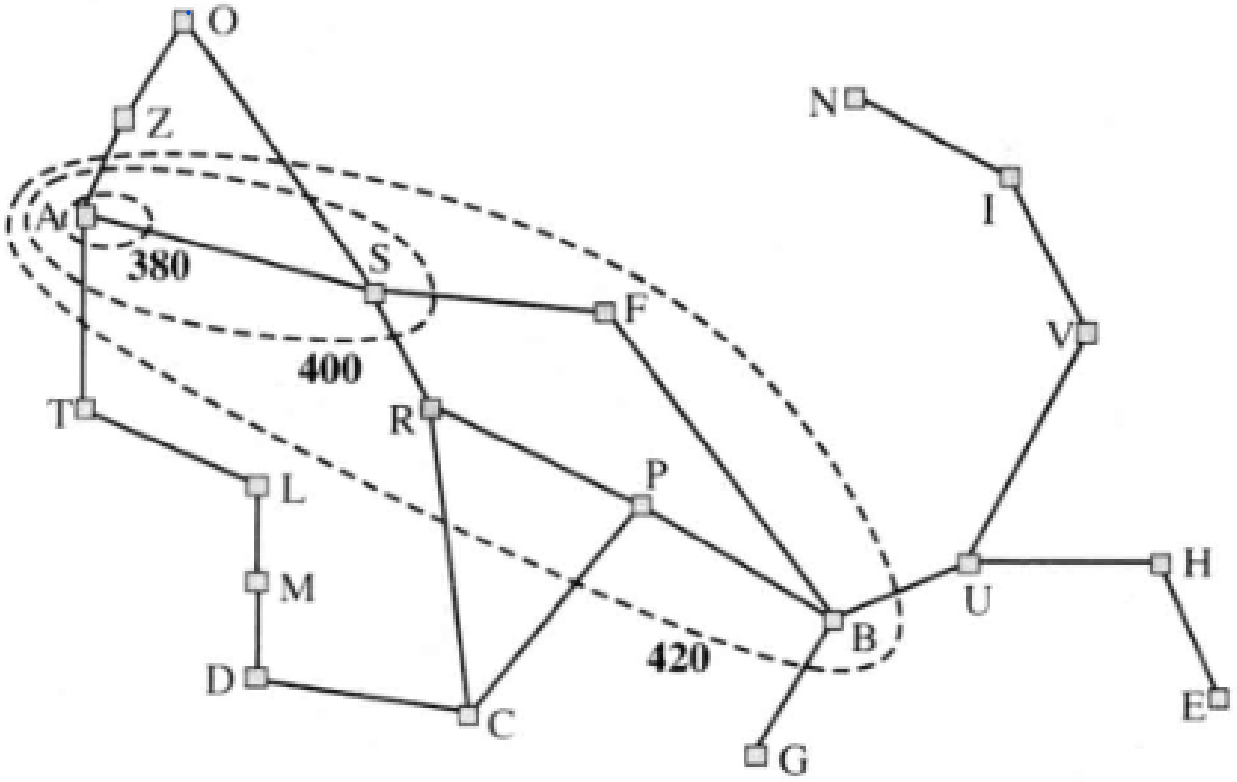
五、联机搜索与学习

连续空间中的局部搜索问题

- 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：罗马尼亚旅行问题

考虑一个实例。假设我们想在罗马尼亚建三个新机场，使地图上（图 3.2）每个城市到离它最近的机场的距离平方和最小。那么问题的状态空间通过机场的坐标来定义：(x_1, y_1)、(x_2, y_2)和(x_3, y_3)。这是个六维空间；也可以说状态空间由六个**变量**定义（一般地，状态是由 n 维向量 $\mathbf{x}$ 来定义的）。在此状态空间中移动对应于在地图上改变一个或多个机场的位置。对于某特定状态一旦计算出最近城市，目标函数 $f(x_1, y_1, x_2, y_2, x_3, y_3)$就很容易计算出来了。假设用 C_i 表示（当前状态下）离机场 i 最近的城市集合。那么，当前状态的邻接状态中，各 C_i 保持常量，我们有：

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} (x_i - x_c)^2 + (y_i - y_c)^2 \qquad (4.1)$$



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。如果不对空间进行离散化，可以直接应用随机爬山法和模拟退火。这些算法随机选择后继，通过随机生成长度为 δ 的向量来完成。

局部搜索问题求解过程

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 $\nabla f = 0$ 找到最大值。（这是可以做到的，例如，如果我们只建一个飞机场：解就是所有城市坐标的算术平均。）然而，在很多情况下，该等式不存在闭合式解。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。这意味着我们只能局部地计算梯度（而不是全局地计算）；例如，

$$\frac{\partial f}{\partial x_1} = 2 \sum_{c \in C_1} (x_i - x_c) \qquad (4.2)$$

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

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

步长

讲课内容

一、局部搜索算法

二、连续空间局部搜索

三、不确定动作搜索

四、部分可观察搜索

五、联机搜索与学习

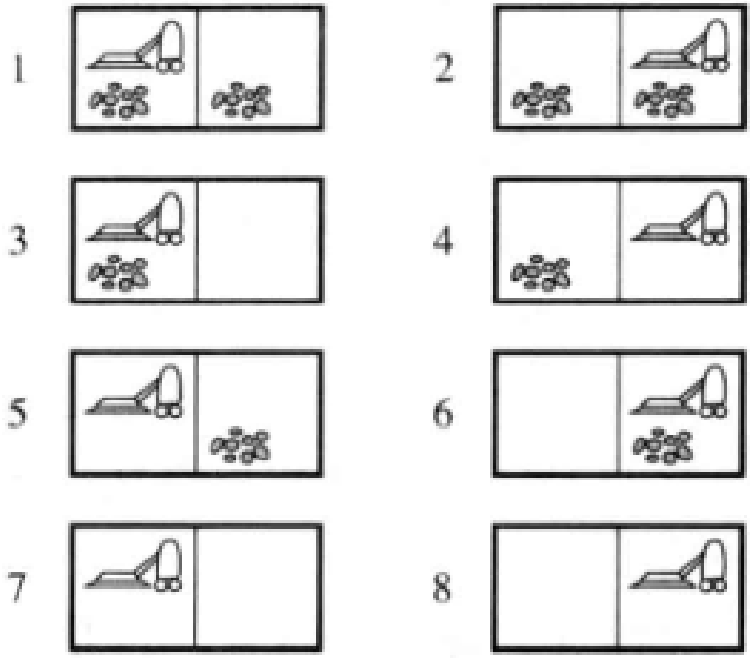
在第 3 章中，我们假设环境是完全可观察的和确定的，并且 Agent 了解每个行动的结果。所以，Agent 可以准确地计算出经过任何行动序列之后能达到什么状态，Agent 总是知道自己处于什么状态。它的传感器在一开始告知 Agent 初始状态，而在行动之后无需提供新的信息。

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。在部分可观察环境中，每个感知信息都可能缩小 Agent 可能的状态范围，这样也就使得 Agent 更容易到达目标。如果环境是不确定的，感知信息告知 Agent 某一行动的结果到底是什么。在这两种情况中，无法预知未来感知信息，Agent 的未来行动依赖于未来感知信息。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。

不稳定的吸尘器世界

例如：在状态1下实施Suck结果为状态集{5,7}，没有一个序列可以用来求解问题，我们需要如下所示的应急规划

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

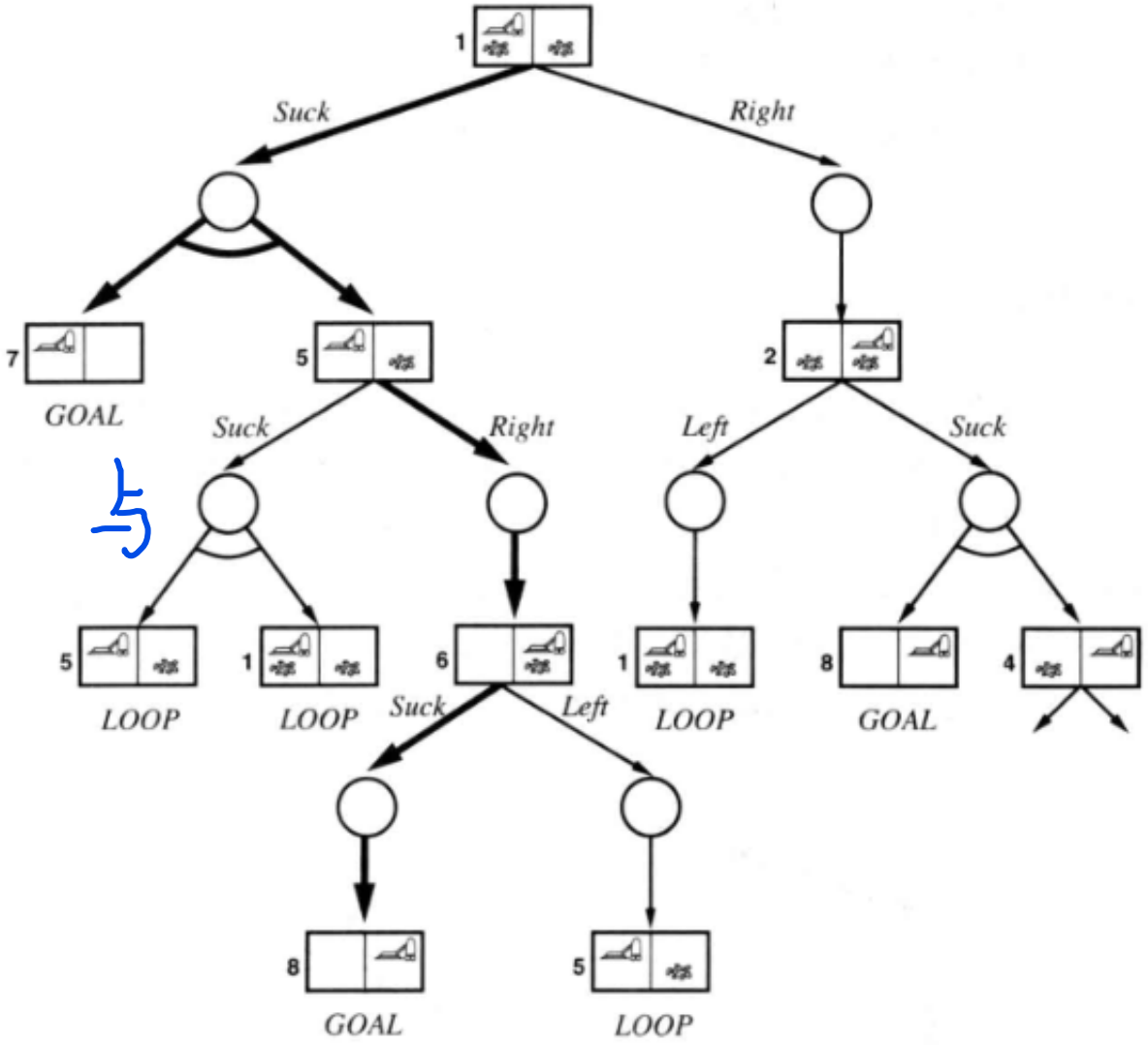


与或搜索树

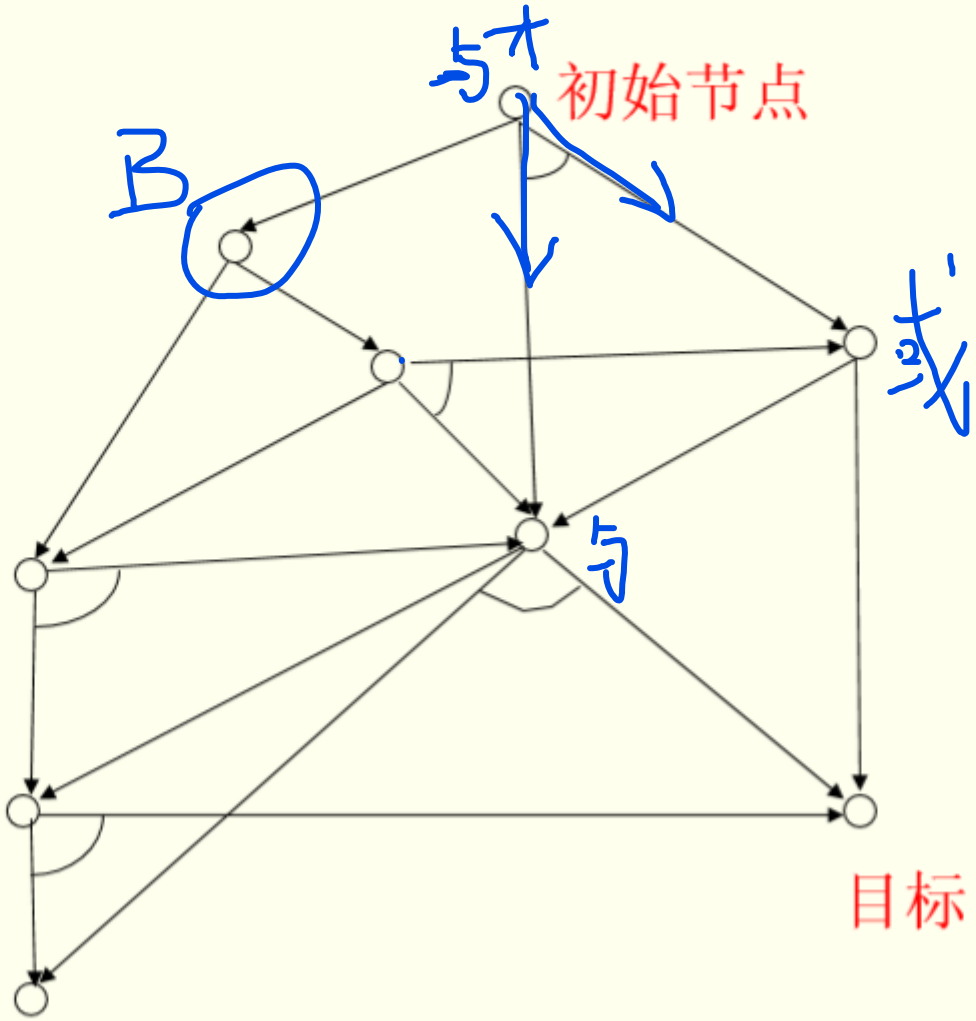
- 与或搜索问题的解是一棵子树：
  - ✓ 每个叶子上都有目标节点
  - ✓ 在或节点上规划一个活动
  - ✓ 在与节点上包含所有可能后果

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

---

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state is on path then return failure
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then return [action | plan]
  return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, 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]
```

与或搜索树


1
Suck
Right
7
GOAL
5
2
Suck
Right
Left
Suck
与
5
LOOP
1
LOOP
6
Suck
Left
1
LOOP
8
GOAL
4
8
GOAL
5
LOOP

AND/OR图搜索



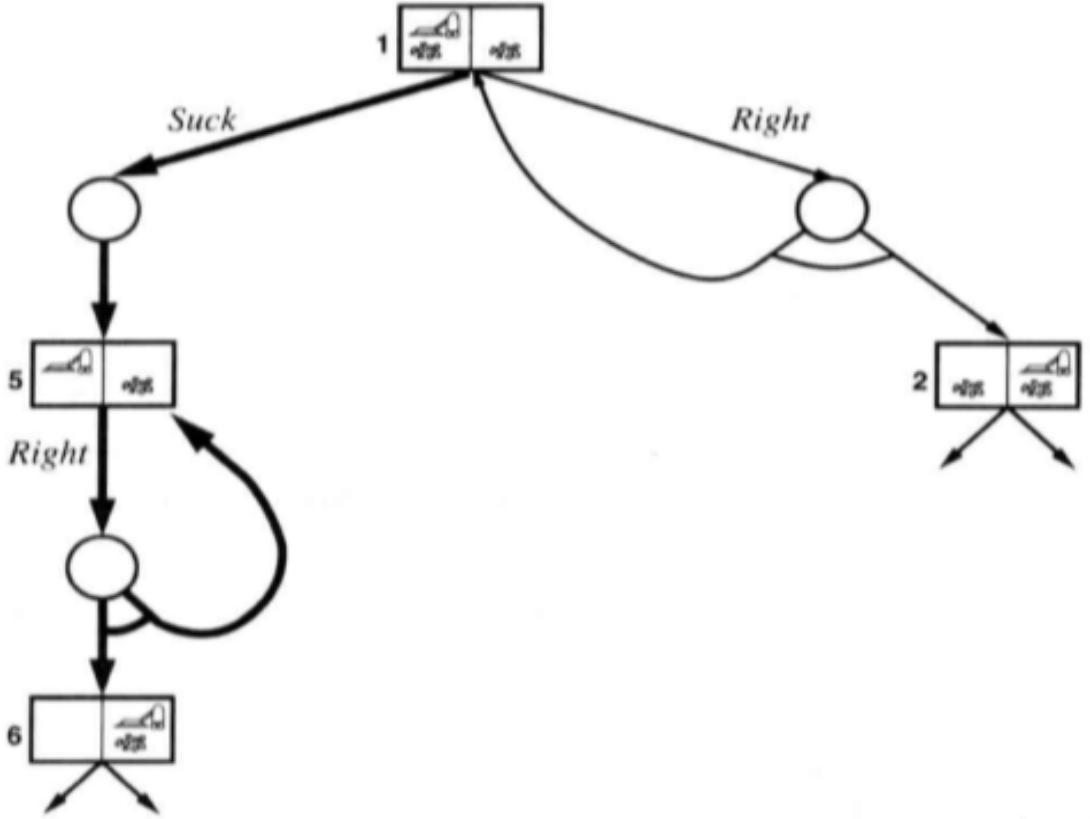
A→B

B open表

不断尝试

考虑不稳定的吸尘器世界，其他地方都与稳定的吸尘器相同，除了有些移动动作会失败，Agent 在原地不动。例如，在状态 1 实施 *Right* 会导致状态集{1, 2}。图 4.12 给出了部分搜索图；显然，从状态 1 出发不再有非循环解，与或图搜索会返回失败。存在循环解，一直尝试 *Right* 动作直到生效。我们可以这样表达，在规划中添加一个标签，之后就可以用这个标签而不用重复整个规划。这样，循环解为：

[*Suck*, L_1: *Right*, **if** *State* = 5 **then** L_1 **else** *Suck*]



讲课内容

一、局部搜索算法

二、连续空间局部搜索

三、不确定动作搜索

四、部分可观察搜索

五、联机搜索与学习

无观察信息的搜索

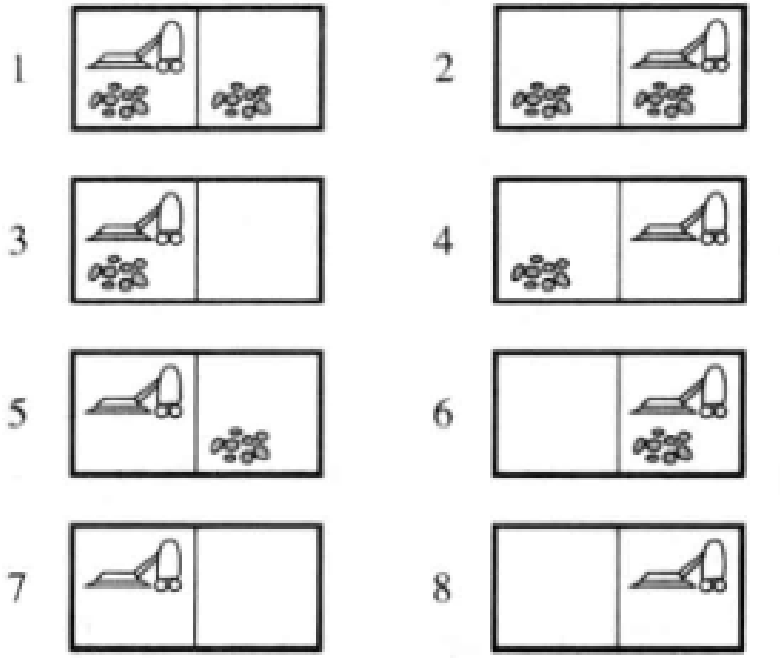
- 假设，Agent知道世界的地理情况，但并不知道自己的位置和地上垃圾的分布：

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

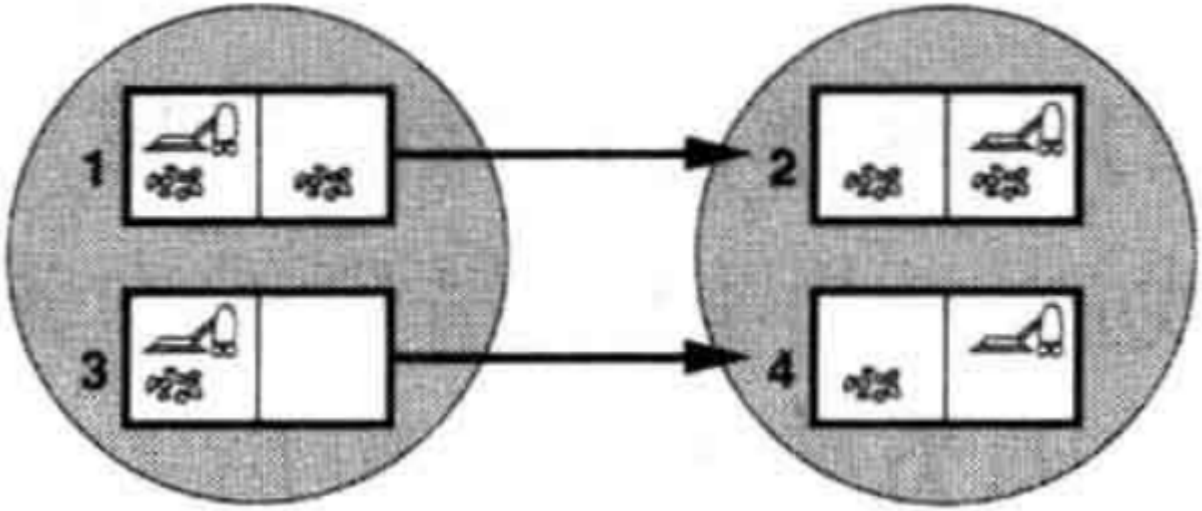


无观察信息的搜索

- 可以如下定义无传感器问题

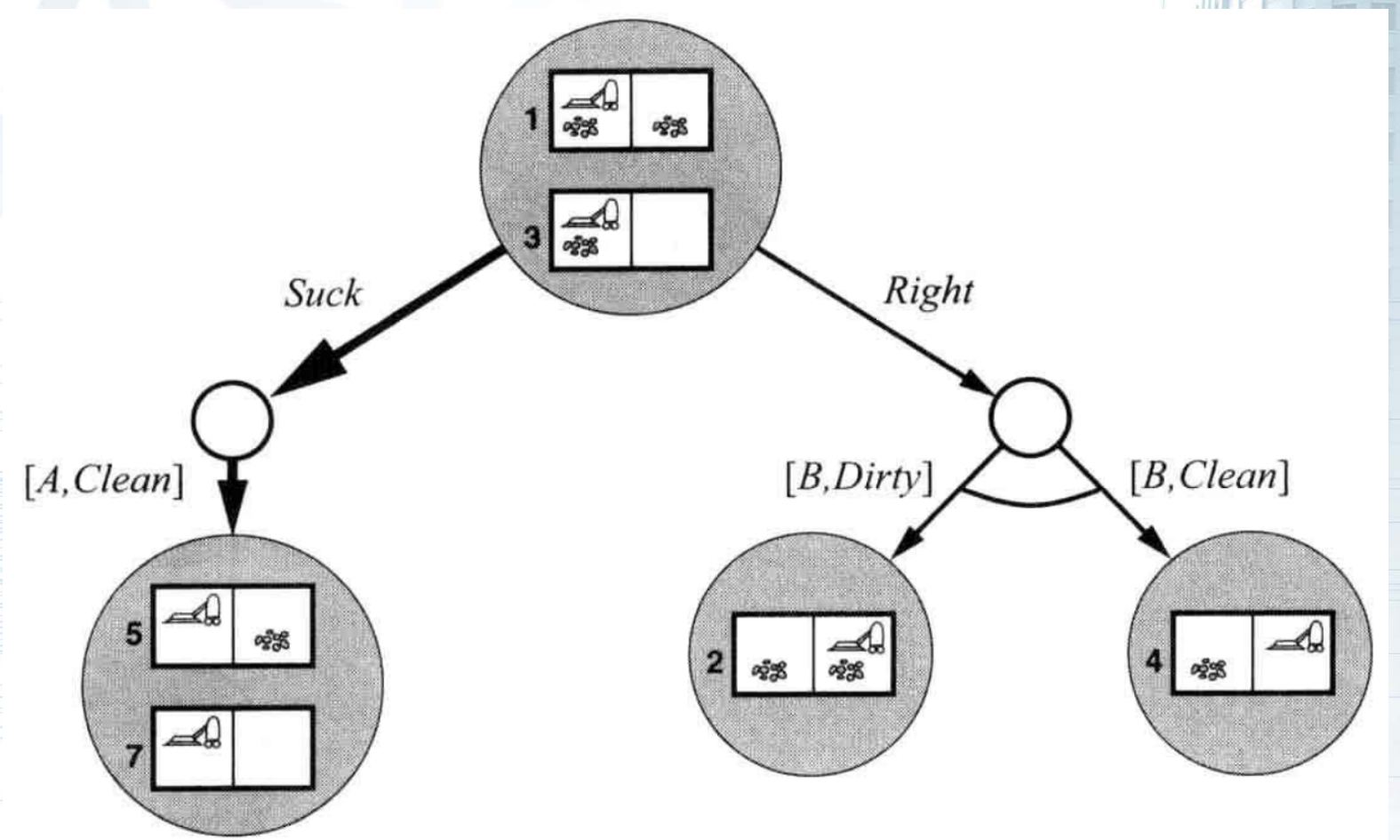
① 信念状态 ②初始状态 ③行动 ④转移模型 ⑤路径开销

- 例：在确定的无感知信息吸尘器世界中，观测Right行动后的下一个信念状态。



求解部分可观察环境中的问题

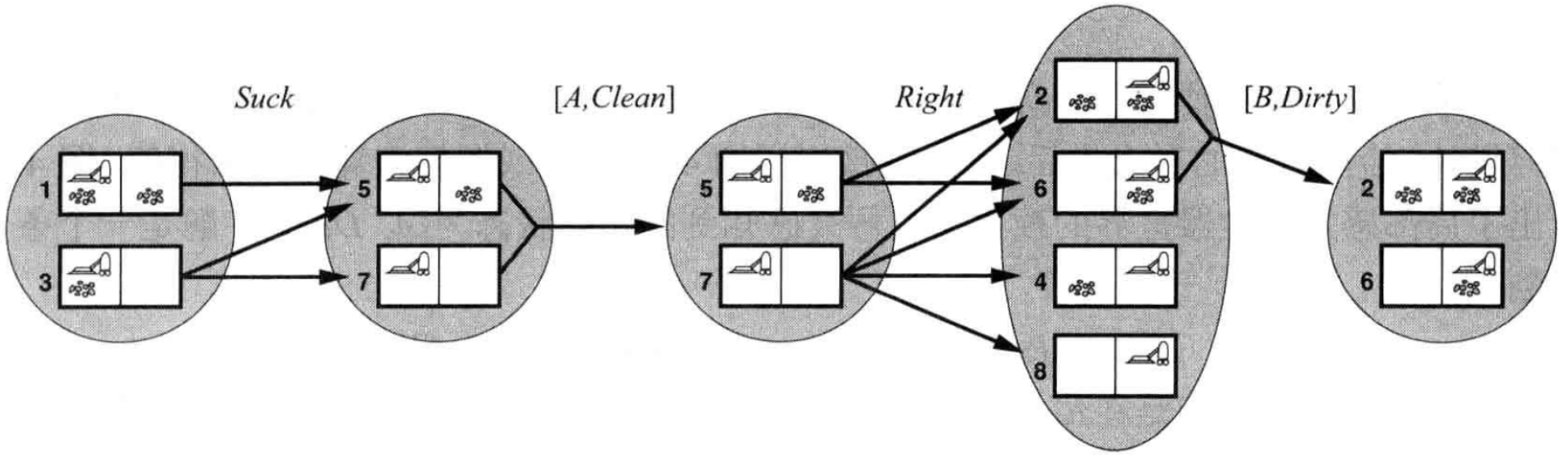
- 例：局部感知器世界的搜索树。



与或树搜索

部分可观察环境中的智能体

- 例：局部感知的学前班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



讲课内容

一、局部搜索算法

二、连续空间局部搜索

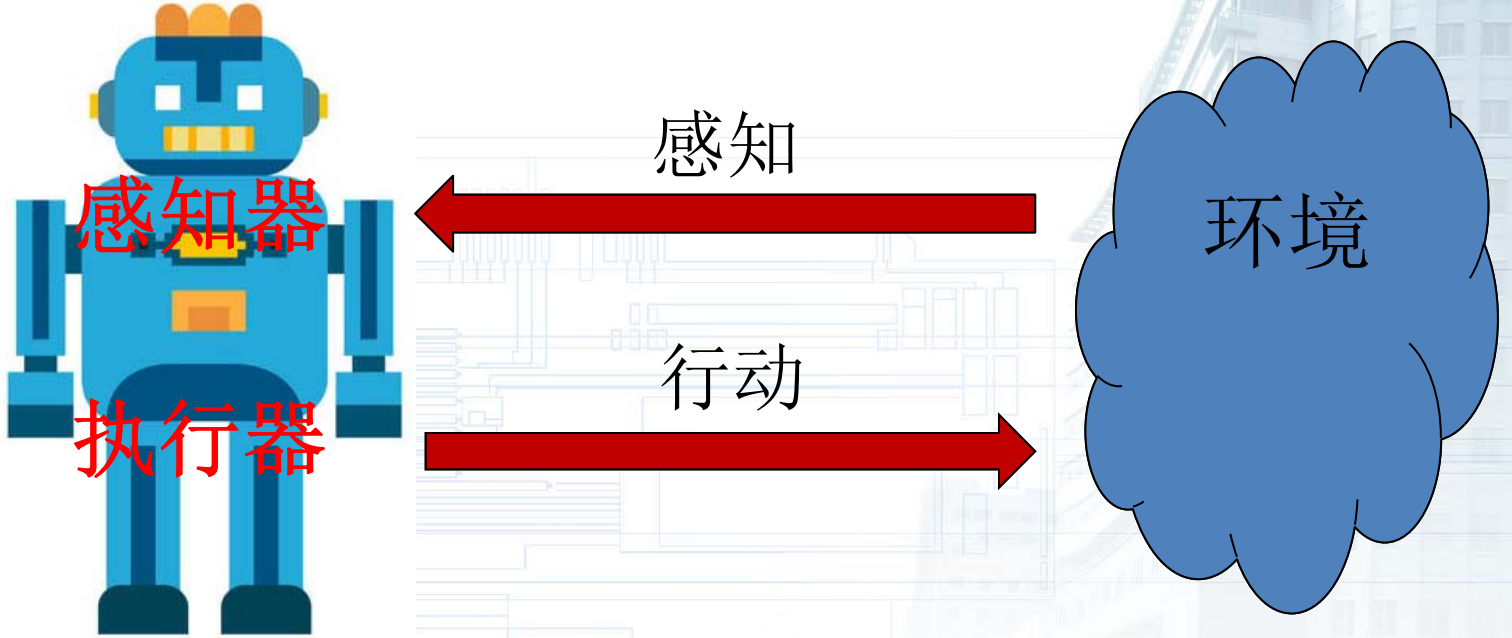
三、不确定动作搜索

四、部分可观察搜索

五、联机搜索与学习

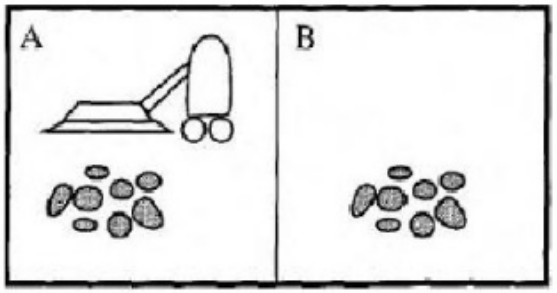
智能体和环境

- Agent可以被视为通过传感器感知环境并通过执行器对该环境产生作用的物体



真空吸尘器世界

- 只有两个地点的真空吸尘器世界



Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
……	……
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck

联机搜索问题

- 脱机搜索：先对实际问题计算出解决方案，然后再涉足现实世界执行解决方案。
- 联机搜索Agent：通过计算和行动交叉完成。

  ① Agent首先采取一个行动；

  ② 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。

- 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
  - ✓ 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Agent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Agent执行行动解决，而不是纯粹的计算过程。

- 假定Agent仅知道如下信息：

① ACTIONS(s)，返回状态s下可能进行的行动列表；

② 单步耗散函数c(s,a,s')--注意Agent必须在知道行动的结果为s'时才能用；

③ GOAL-TEST(s)。

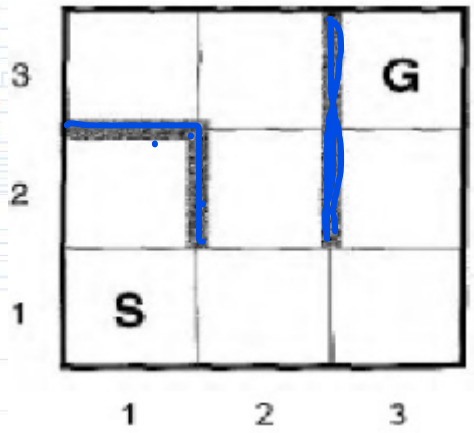
- 注意：Agent不能访问一个状态的后继，除非它实际尝试了该状态下的所有行动

- 假设：

① Agent总能认出它以前到达过的状态，并且它的动作是确定性的；

② Agent将使用一个能够估计从当前状态到目标状态的距离的可采纳启发函数h(s)。

✓ 例如，对于迷宫问题，Agent知道目标的位置并且可以使用曼哈顿距离启发式

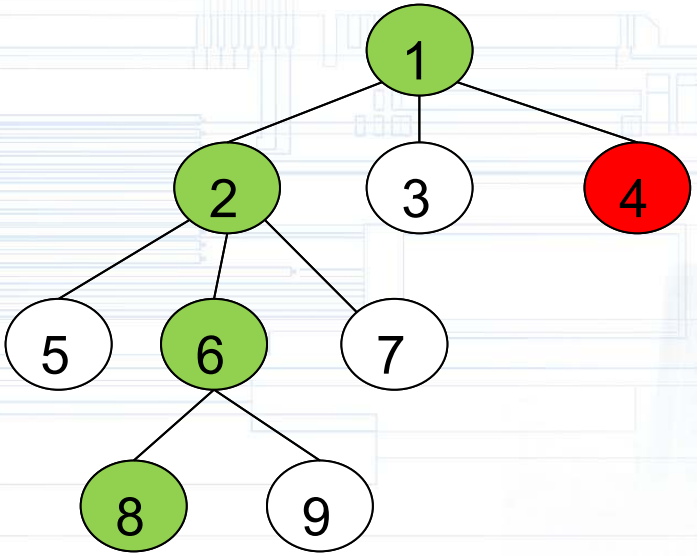


$$\text{竞争率} = \frac{\text{Agent实际旅行经过的路径总耗散}}{\text{实际最短的路径（如果事先了解搜索空间）}}$$

- 理想的竞争率为1

联机搜索

- 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节点，然后马上又在空间的另一部分扩展另一个节点，
  - ✓ 因为脱机算法节点扩展涉及的是模拟的而不是实际的行动。



联机深度优先搜索算法

```
function Online-DFS-Agent(s') returns an action
    inputs: s', a percept that identifies the current state
1.  if Goal-Test(s') then return stop
2.  if s' is a new state then unexplored[s'] ← Action(s')
3.  if s is not null then do ； s是前一个状态，初值为空
4.      result[a, s] ← s' ； a是前一个行动，即action
5.      add s to the front of unbacktracked[s']
6.  if unexplored[s'] is empty then
7.      if unbacktracked[s'] is empty then return stop
8.      else a ← an action b such that
                    result[b, s']=Pop(unbacktracked[s'])
9.  else a ← Pop(unexplored[s'])
10. s ← s'
11. return a
```

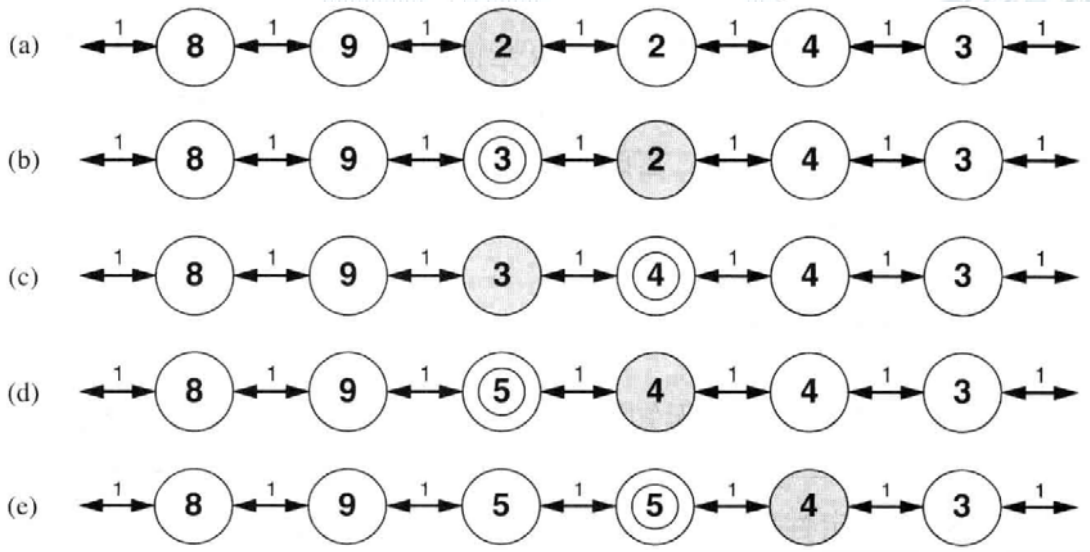
联机局部搜索

- 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Agent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Agent不能把自己传送到一个新的状态。
  - ✓ 不妨考虑随机游走。

提高爬山法算法的内存利用率

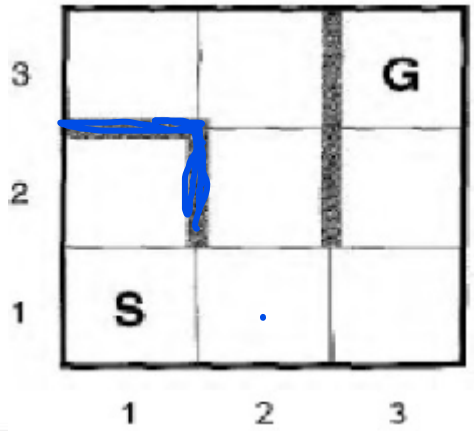
- 例：(a)Agent已经进入局部最优
  - ✓ Agent根据对邻居状态的当前耗散估计来选择到达目标的最佳路径，而不是停下来
  - ➢ 经过邻居s’到达目标的估计耗散为：c(s, a, s’)+H(s’)
  - ✓ (b)两个邻接的耗散分别为（1+9）和（1+2），选择向右移动。此时，原来的状态到达目标状态至少需要（1+2）步，因此它的H需要更新。



实时学习A*（LRTA*）算法

```
function LRTA*-Agent(s') returns an action
  inputs: s', a percept that identifies the current state
1.   if Goal-Test(s') then return stop
2.   if s‘ is a new state (not in H) then H[s’] ← h(s')
3.   unless s is null ；s是前一个状态，初值为空
4.      result[a, s] ← s' ；a是前一个行动，即action
5.      H[s] ← min {LRTA*-Cost(s, b, result[b,s], H) | b∈Actions(s)}
6.   a ← an action b in Actions(S') that
7.       minimizes LRTA*-Cost(s', b, result[b, s'], H)
8.   s ← s'
9.   return a

function LRTA*-Cost(s, a, s', H) returns a cost estimate
1.   if s' is undefined then return h(s)
2.   else return c(s, a, s')+H[s']
```

联机搜索中的学习



- 当Agent已经到达状态（1,2）时，不知道行动down能回到状态（1,1）
- Agent不知道行动up还能从状态（2,1）到状态（2,2），从状态（2,2）到状态（2,3）等
- 通常，希望Agent能学习到up能使y坐标值增长，除非遇到墙；down能使y坐标值降低等

- 希望Agent能学习到up能使y坐标值增长，除非遇到墙；down能使y坐标值降低
- 必须满足如下两个要求：

① 需要一个对这类一般规则的形式化的和明确的可操作描述

✓ 到目前为止，这些信息隐藏在称为后继函数的黑盒子里

✓ 涉及：“知识表示和推理”

② 需要有算法能够根据Agent得到的特定的观察资料来构造合适的一般规则

本次课程作业

- 教科书 P134—136
  - 4.4
  - 4.12

中国科学院
自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

感谢同学们听课

欢迎讨论与交流