Beyond Classical Search

1 何谓局部搜索

2 局部搜索策略：(最陡)爬山法、随机爬山法、首选爬山法、随机重启爬山法；模拟退火算法；局部束搜索；遗传算法；连续空间怎么办？

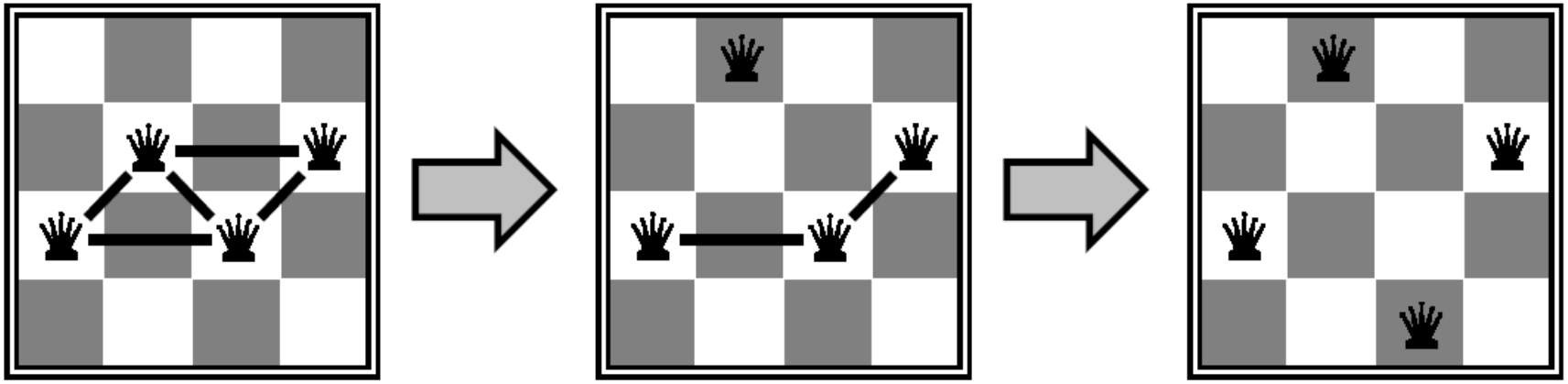
3 动作具有不确定性时、无传感器时、部分可观察时，怎么搜索？

4 联机搜索

何谓局部搜索

n-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal(对角线)
- Move a queen to reduce number of conflicts



n-queens

- Romania problem cares about the path which indicates how to get to Bucharest.
  - a search tree is kept in memory
- N-queens problem cares about whether there exists a specific state and what it is.
- In many optimization problems, e.g. n-queens,
  - the path to the goal is irrelevant; the goal state itself is the solution
  - State space = set of "complete" configurations
  - Find configuration satisfying constraints

Local search algorithms {4.1}

- In such cases, use local search algorithms
  - keep a single "current" state, try to improve it
  - The agent do not know what the previous state is. It only know the current state.

爬山法（最陡爬山法，随机爬山法，首选爬山法，随机重启爬山法）；模拟退火法；局部束搜索；遗传算法；连续空间局部搜索

局部搜索策略

Hill-climbing search {4.1.1}

- "Like climbing Everest in thick fog with amnesia（健忘症）"
- (steepest) Hill-Climbing:

```
function HILL-CLIMBING(problem) returns a state that is a local maximum

  current ← MAKE-NODE(problem.INITIAL-STATE)
  loop do
    neighbor ← a highest-valued successor of current
    if neighbor.VALUE ≤ current.VALUE then return current.STATE
    current ← neighbor
```

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h.

what would go wrong with the Hill Climbing Search?

Hill-climbing search {4.1.1}

- local maxima
- ridges
- plateaux

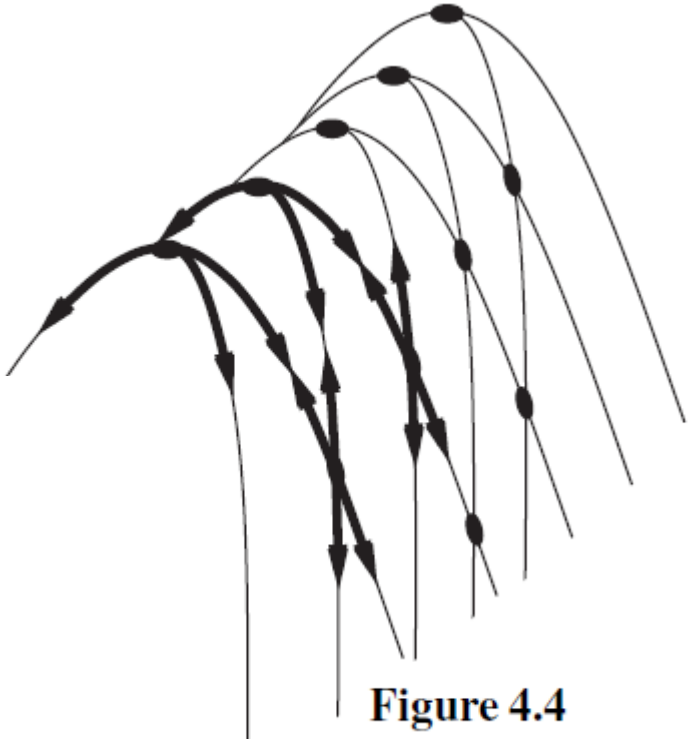


Figure 4.4

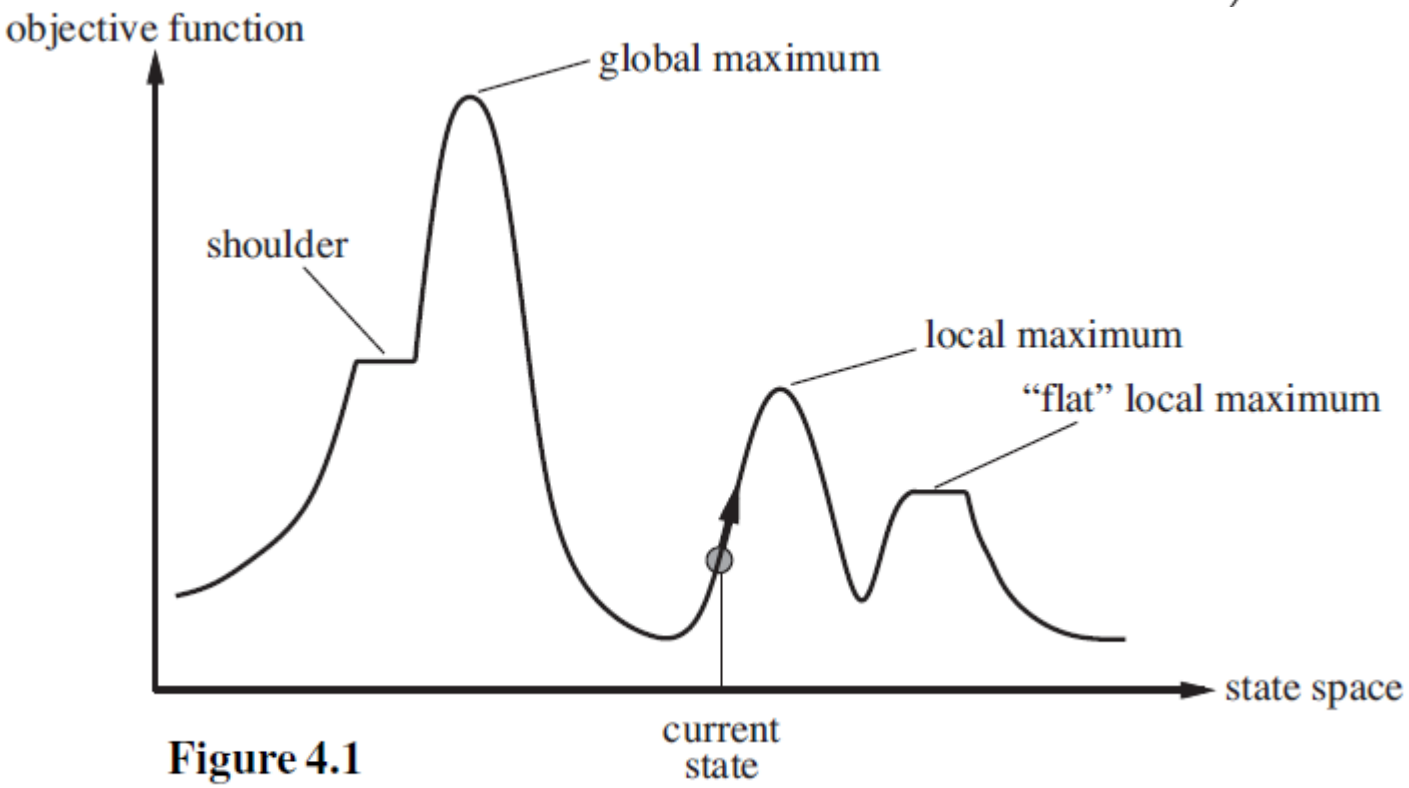


Figure 4.1

最陡爬山不一定最优！

后继结点过多怎么办？

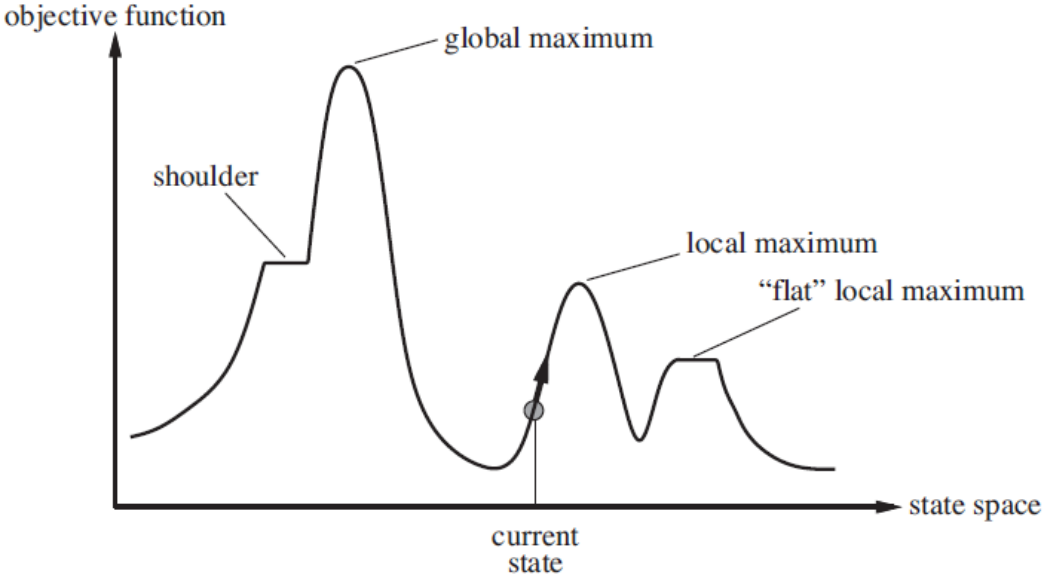
Hill-climbing search {4.1.1}

- stochastic hill climbing
  - chooses a random from among the uphill moves
  - 比最陡爬山法收敛慢，但在某些状态空间里能找到更好的解。
- first-choice hill climbing
  - generating successors randomly until one is generated that is better than the current state
  - 在后继结点很多时是个好策略。

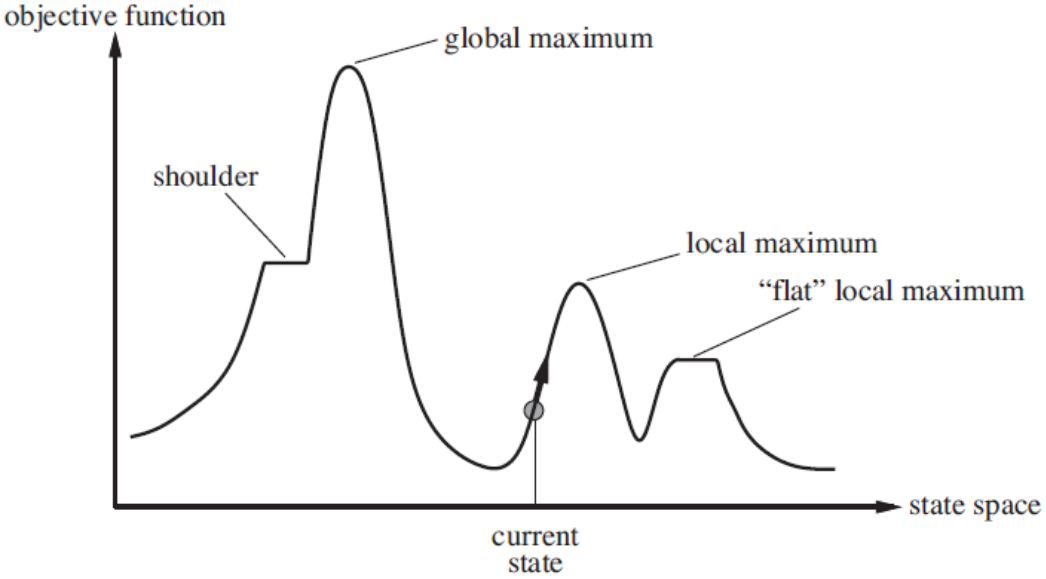
如果失败，怎么办？

Hill-climbing search {4.1.1}

- Random-restart hill climbing
  - "If at first you don't succeed, try, try again. "
  - **如果每次爬山法搜索成功的概率为*p*，那么需要重新开始搜索的期望次数为_____。**



How to escape local maximum?



How to escape local maximum?

多次随机重启（串行）；从多个随机初始状态出发同时爬山（随机重启的并行版本）；允许下山移动。

Simulated annealing search {4.1.2}

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

```
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)
```

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature T as a function of time.

Properties of simulated annealing search {4.1.2}

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc

多次随机重启（串行）或从多个随机初始状态出发爬山（随机重启的并行版本）是解决局部极值的有效途径。这种思路的一种变体是局部束搜索。

Local beam search {4.1.3}

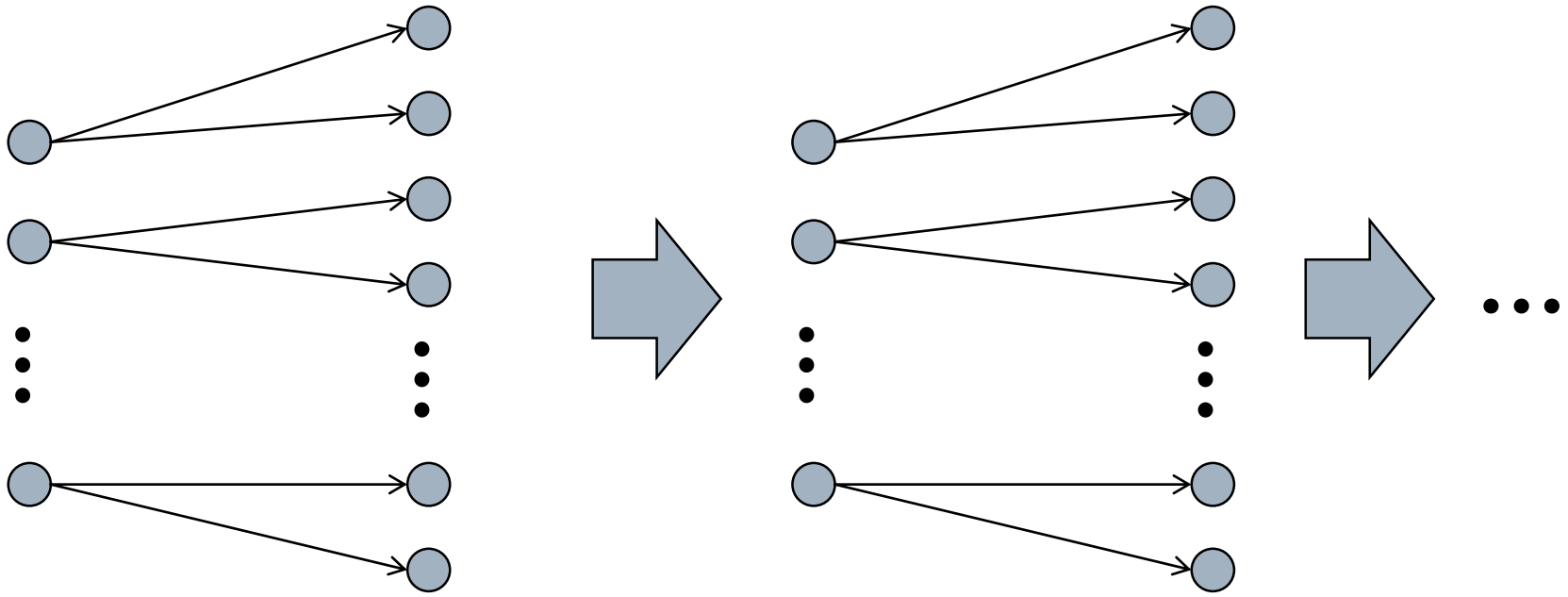
- Start with *k* randomly generated states
- At each iteration, all the successors of all *k* states are generated
- If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

这和k个随机重启有何区别？

Local beam search {4.1.3}

- 这似乎是并行地运行了k个随机重启？
- 随机重启的每个搜索过程是独立的！
- 局部束搜索中并行的搜索线程之间不是独立的。

Local beam search {4.1.3}



Start with *k* randomly generated states

all the successors of all *k* states are generated

select the *k* best successors from the complete list

Until any one in the list is a goal state.

局部束搜索中，每个后继状态都由单个父（前驱）状态产生。如果每个后继都从两个父（前驱）状态产生，则得到**遗传算法**。

Genetic algorithms {4.1.4}

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce next generation of states by **selection**(选择), **crossover**(杂交), and **mutation**(变异)

Genetic algorithms {4.3.4}

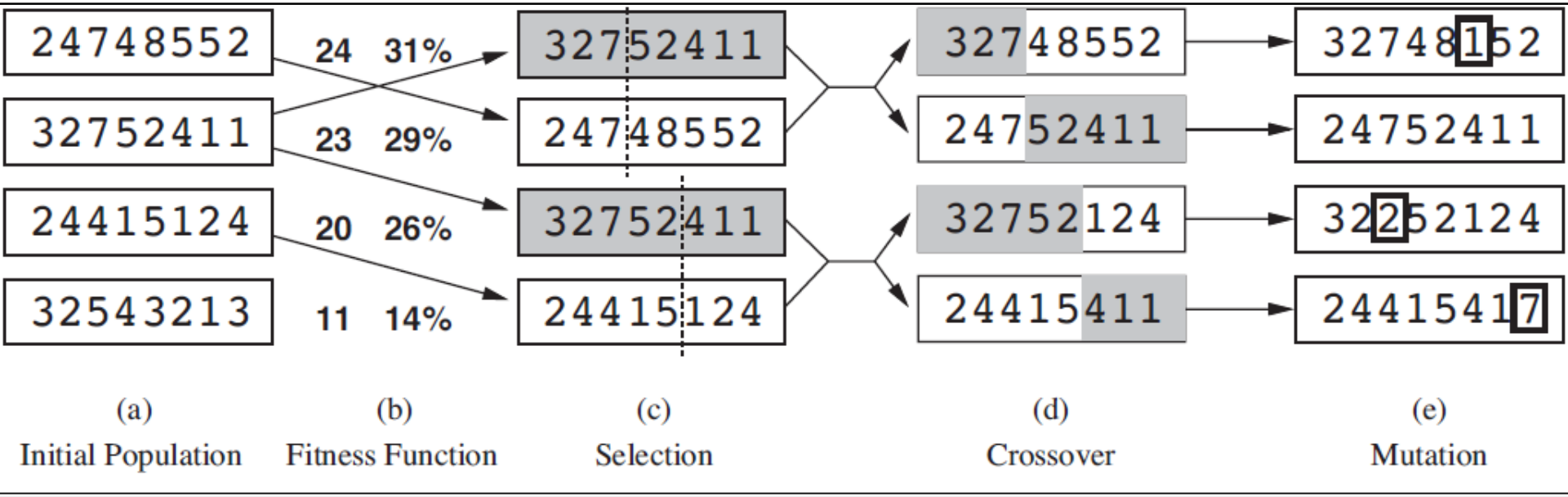


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc

Genetic algorithms {4.3.4}

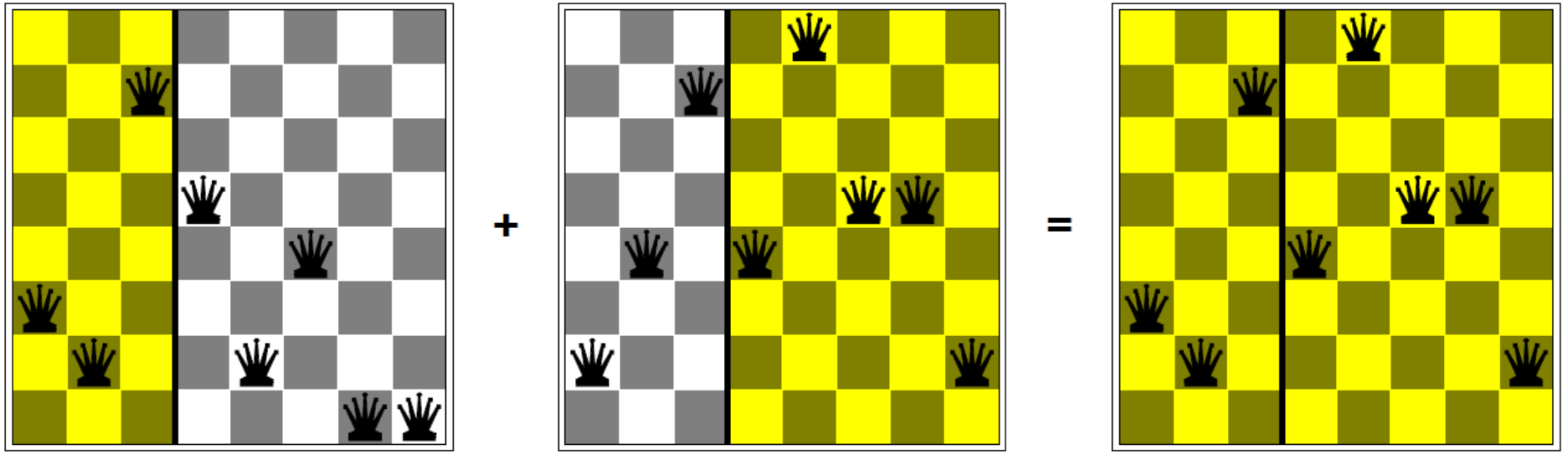


Figure 4.7 The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d). The shaded columns are lost in the crossover step and the unshaded columns are retained.

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
      new_population ← empty set
      for i = 1 to SIZE(population) do
          x ← RANDOM-SELECTION(population, FITNESS-FN)
          y ← RANDOM-SELECTION(population, FITNESS-FN)
          child ← REPRODUCE(x, y)
          if (small random probability) then child ← MUTATE(child)
          add child to new_population
      population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure 4.6, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

连续空间怎么搜索？例如求使得f(x,y)最大或最小的(x,y)值。

local search in continuous spaces {4.2}

- 我们讨论过的算法，除了首选爬山法和模拟退火，其他算法不能够处理连续的状态和动作空间

local search in continuous spaces {4.2}

- 假设在罗马尼亚建三个新机场，使地图上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： $\mathbf{x}=(x_1, y_1, x_2, y_2, x_3, y_3)$。假设用$C_i$表示当前状态下离机场$i$最近的城市集合，则有

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

local search in continuous spaces {4.2}

- 很多方法都试图利用地形图的**梯度**来找到最大值。

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

- 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：

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

local search in continuous spaces {4.2}

- 在连续状态空间同样存在局部极大值问题、山脊问题、高原问题。

local search in continuous spaces {4.2}

- **约束优化**：问题的解必须满足变量的某些严格约束，称此优化问题是受约束的。例如，在机场选址问题中，可以限定机场的位置在罗马尼亚境内并且是在陆地上（而不是在某个湖的中央）。

- 约束最优化问题的难点取决于约束和目标函数的性质。

local search in continuous spaces {4.2}

- 最著名的一类约束优化问题是**线性规划**问题，其约束是线性不等式并且能够组成一个**凸多边形**区域，目标函数也是线性的。线性规划问题的时间复杂度是关于变量数目的多项式时间函数。

	确定	不确定
完全可观察	ch03	4.3
完全不可观察	4.4.1	4.4.1
部分可观察	4.4.2	4.4.2

动作具有不确定性时、无传感器时、部分可观察时，怎么搜索？

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。

如果环境是不确定的或部分可观察（甚至无观察）的，则Agent无法断定自己处于什么状态。这时Agent怎么办？

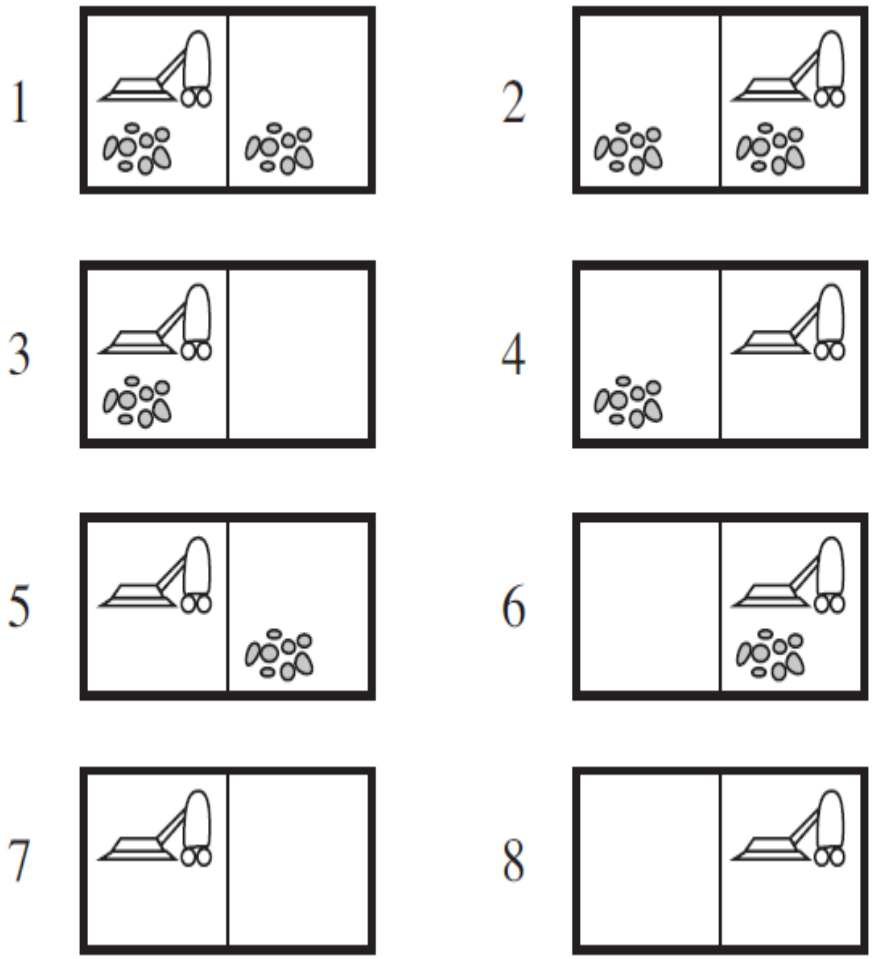
在第3章中，我们假设环境是完全可观察的和确定的，并且Agent了解每个行动的结果。所以，Agent可以准确地计算出经过任何行动序列之后能达到什么状态，Agent总是知道自己处于什么状态。

如果环境是不确定的或部分可观察（甚至无观察）的，则Agent无法断定自己处于什么状态。这时Agent怎么办？

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。
如果环境是部分可观察的，感知信息可缩小Agent可能的状态范围.

如果环境是不确定的或部分可观察（甚至无观察）的

- 这两种情况中，Agent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。



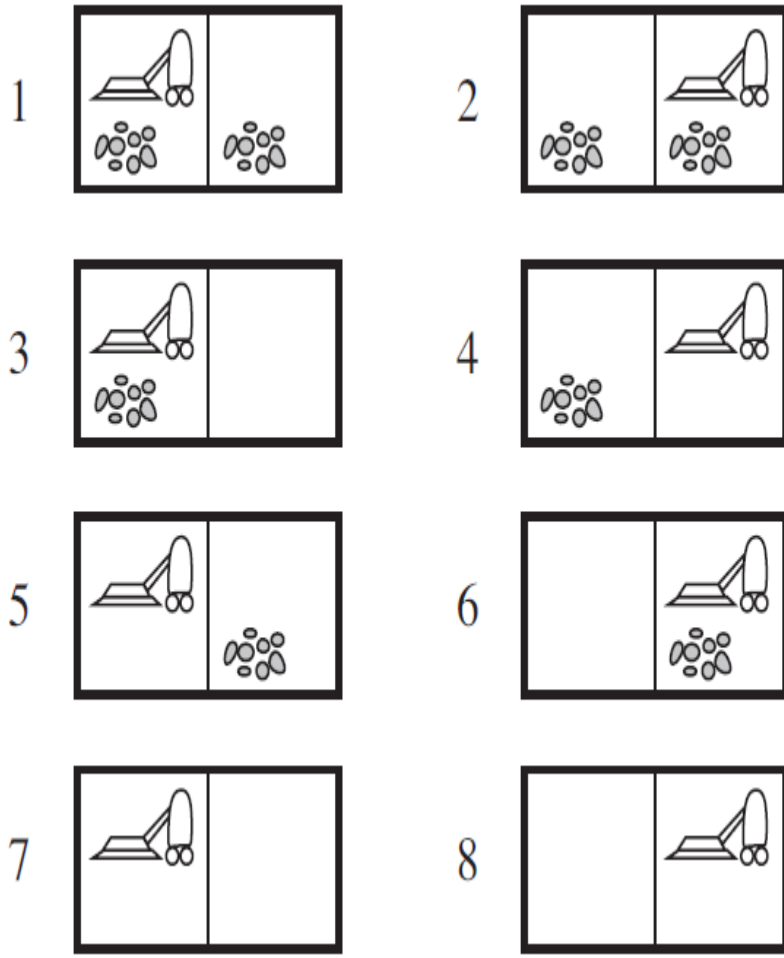
完全可观察+不确定性

	确定	不确定
完全可观察	ch03	**4.3**
完全不可观察	4.4.1	4.4.1
部分可观察	4.4.2	4.4.2

searching with nondeterministic actions {4.3}

完全可观察+不确定

- 现在假设我们引入某种不确定形式，吸尘器更强大但却是不稳定的。在不稳定的吸尘器世界中，*Suck*的特点是：
  - 在一块脏的区域中进行此动作可以使该区域变得干净，有时也会同时清洁邻近区域。
  - 如果是在干净区域进行此动作有可能使脏东西掉在地毯上。



RESULT(1,SUCK) = ?
如何描述问题的解？

searching with nondeterministic actions {4.3}

完全可观察+不确定

- **转移模型**：RESULT函数的返回不再是单个状态，而是一组可能的状态

  RESULT(1, suck) = {5,7}

- **解**：问题的解不再是一个动作序列，而是一个应急规划：

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

searching with nondeterministic actions {4.3}

完全可观察+不确定

- 此时，不确定性的问题的解是嵌套的**if-then-else**语句；这就意味着它们是树而不是序列。这就允许了在执行过程中根据发生的应急情况进行选择。

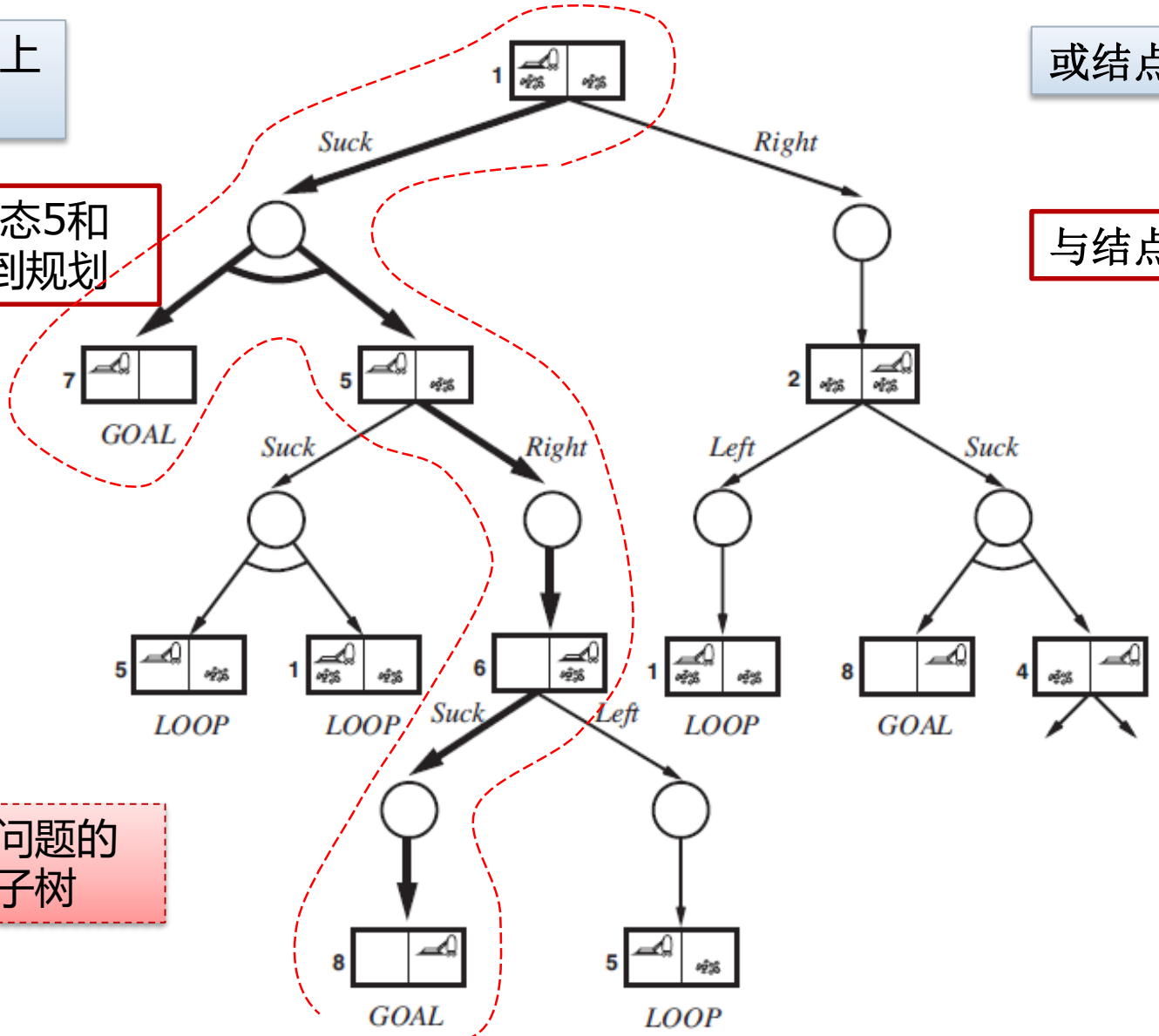
- 真实、物理的世界中的许多问题都是应急问题，不可能做出精确预测。正是由于这个，人们在走路或是驾车的时候一直是睁大眼睛留着神的。

searching with nondeterministic actions {4.3}

完全可观察+不确定

在或结点上选择动作

分别为状态5和状态7找到规划



或结点

与结点

与或搜索问题的解是一棵子树

完全不可观察

	确定	不确定
完全可观察	ch03	4.3
完全不可观察	**4.4.1**	**4.4.1**
部分可观察	4.4.2	4.4.2

searching with no observations {4.4.1}

- 如果Agent感知不到任何信息，我们称之为**无传感问题**。这时，Agent无法断定自己的物理状态。

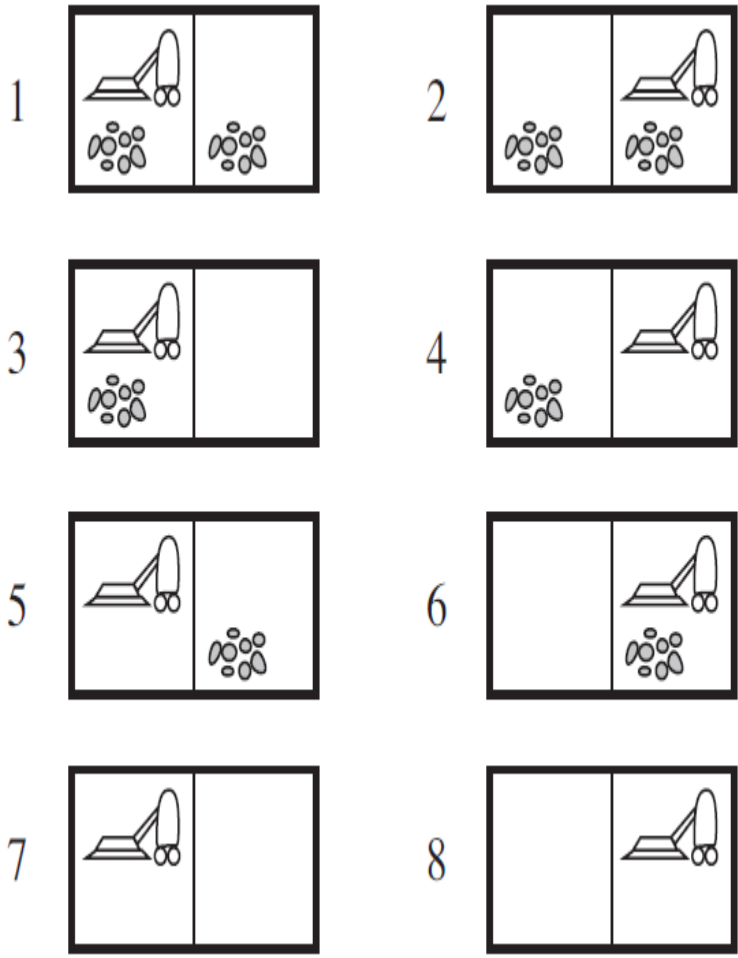
- 无传感器问题是否无解？

searching with no observations {4.4.1}

- 无传感器问题很多时候是可解的。
- 无传感Agent十分有用。医生通常会使用广谱抗生素，而不会使用应急规划——即做个昂贵的血液检查然后等待检查结果，然后再开出特定的抗生素，估计还得住院，因为感染可能更厉害了。

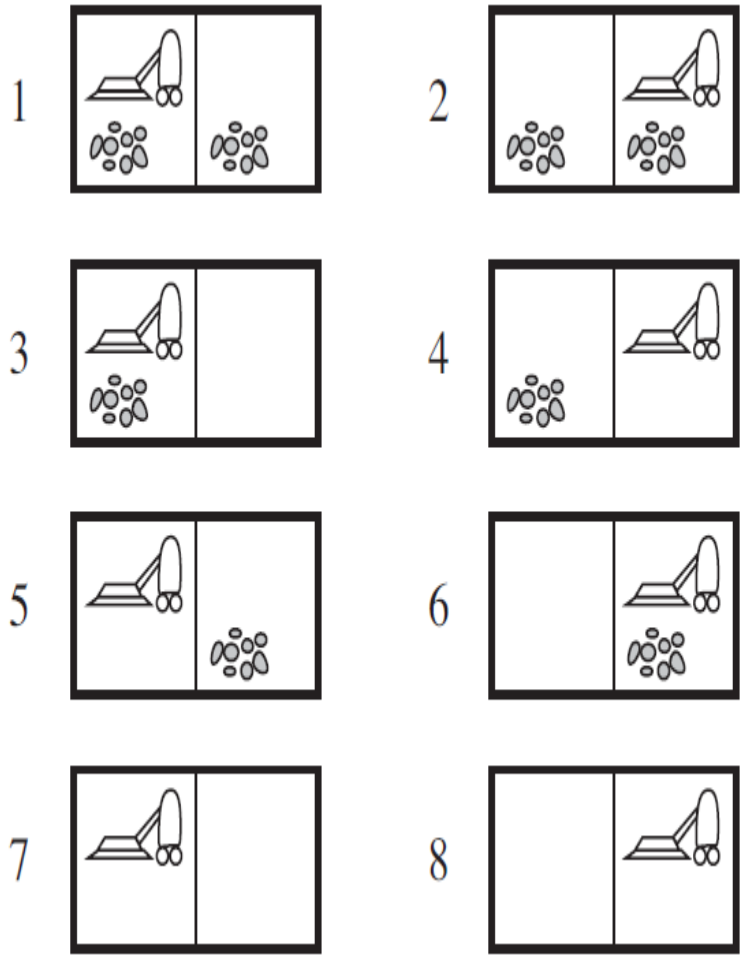
searching with no observations {4.4.1}

- 吸尘器世界的无传感版本（但动作具有确定性）。假设Agent知道世界的地理情况，但不知道自己的位置和地上垃圾的分布。



searching with no observations {4.4.1}

- 吸尘器世界的无传感版本（但动作具有确定性）。假设Agent知道世界的地理情况，但不知道自己的位置和地上垃圾的分布。
- 初始状态可能是集合{1,2,3,4,5,6,7,8}中的一个。
- 行动*Right*导致Agent在{2,4,6,8}的某个状态中
- 行动序列[*Right*, *Suck*]总会导致状态集{4,8}。
- 不管初始状态是什么，[*Right*, *Suck*, *Left*, *Suck*] 确保Agent到达目标状态7。



searching with no observations {4.4.1}

- 假设物理问题*P*由$ACTIONS_P$, $RESULT_P$, $GOAL\text{-}TEST_P$和$STEP\text{-}COST_P$定义。
- 不再搜索物理状态空间，转而搜索信念状态空间，我们可以如下定义对应的无传感问题：
- **信念状态**：整个信念状态空间包含物理状态的每个可能集合。如果*P*有*N*个状态，那么这个无传感问题有2^N个信念状态。
- **初始(信念)状态**：_____。

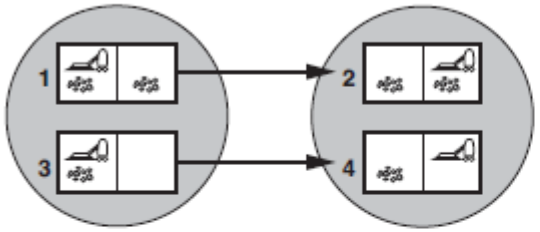
searching with no observations {4.4.1}

- 假设物理问题*P*由$ACTIONS_P$, $RESULT_P$, $GOAL\text{-}TEST_P$和$STEP\text{-}COST_P$定义。
- 不再搜索物理状态空间，转而搜索信念状态空间，我们可以如下定义对应的无传感问题：
- **信念状态**：整个信念状态空间包含物理状态的每个可能集合。如果*P*有*N*个状态，那么这个无传感问题有2^N个信念状态。
- **初始(信念)状态**：P中所有状态的集合。

searching with no observations {4.4.1}

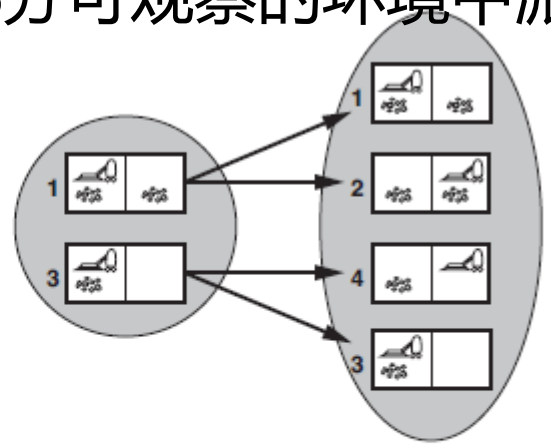
- **行动**ACTIONS(b)：假设Agent的信念状态b = $\{s_1, s_2\}$，但$\text{ACTIONS}_P(s_1) \neq \text{ACTIONS}_P(s_2)$；Agent就不确定哪个行动是合法的。
- 如果非法行动不影响环境，那么在当前信念状态b下的任一物理状态采取行动的*并集*是安全的。
- 如果非法行动导致世界末日，只允许交集可能更安全。
- 对吸尘器世界而言，每个状态有相同的合法行动，因此两种方法有同样的结果。

- **转移模型**RESULT(b,a)= b' ：
  - 对于确定性的行动，
    b' = {s' : s' = RESULT$_P$(s,a)且$s \in b$}，
    b' 决不会比b大。
  - 对于不确定性的行动，
    b' = {s' : $s' \in$ RESULT$_P$(s,a)且$s \in b$}，
    b' 可能会大于b。
  - 行动后生成新的信念状态的过程称为**预测**；
    b'=PREDICT$_P$(b,a)将在部分可观察的环境中派上用场



确定性动作

(a)



向左向右的动作有不确定性

(b)

Figure 4.13 (a) Predicting the next belief state for the sensorless vacuum world with a deterministic action, *Right*. (b) Prediction for the same belief state and action in the slippery version of the sensorless vacuum world.

searching with no observations {4.4.1}

- **目标测试**：Agent需要一个确保生效的规划，意味着一个信念状态满足目标仅当_____。Agent可能*不经意间*很早就到达了目标，但它并不*知道*它已经做到了。

searching with no observations {4.4.1}

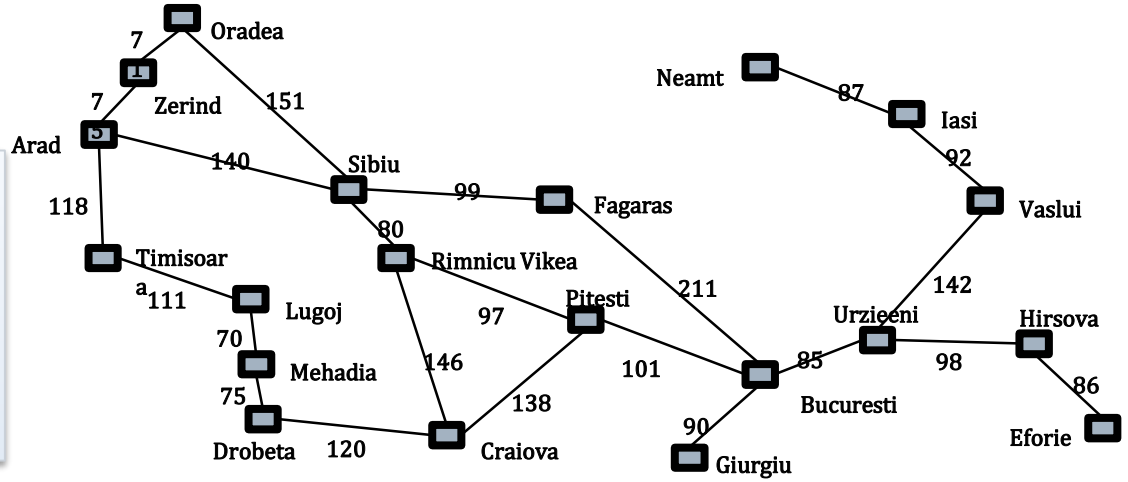
- **目标测试**：Agent需要一个确保生效的规划，意味着一个信念状态满足目标仅当其中所有的物理状态都满足GOAL - $TEST_P$。Agent可能*不经意间*很早就到达了目标，但它并不*知道*它已经做到了。

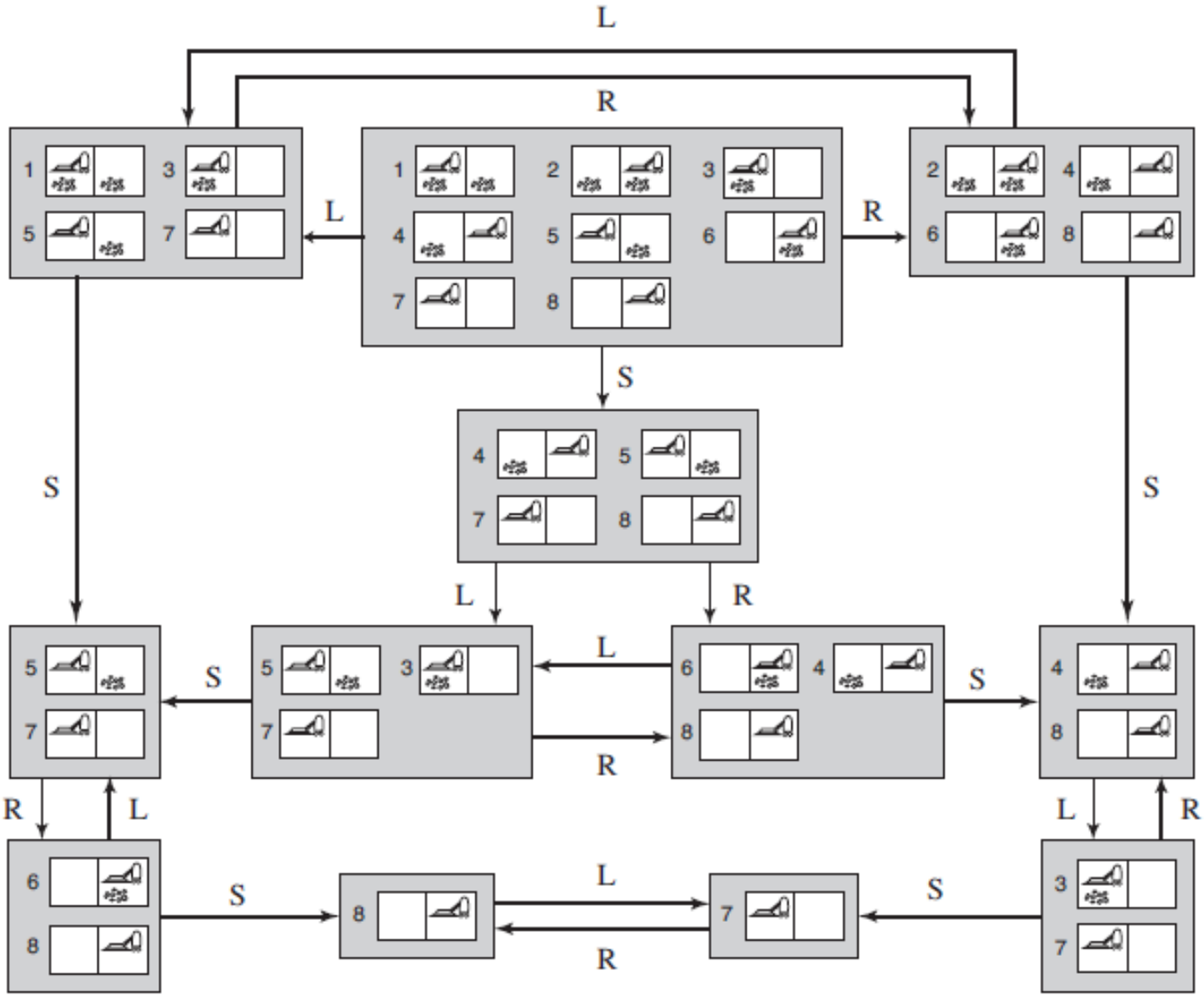
searching with no observations {4.4.1}

- **路径开销**：同样棘手。如果同一行动在不同状态下可能有不同的开销，那么在给定信念状态下采取这种行动会有几个可能的值。目前我们假设一个行动的开销在所有状态下是相同的。

In(Zerind)时，
Goto(Arad)的开销是7

In(Sibiu)时，
Goto(Arad)的开销是140





完全不可观察
+
确定性环境

Figure 4.14 The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each shaded box corresponds to a single belief state. At any given point, the agent is in a particular belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box. Actions are represented by labeled links. Self-loops are omitted for clarity.

部分可观察

	确定	不确定
完全可观察	ch03	4.3
完全不可观察	4.4.1	4.4.1
部分可观察	**4.4.2**	**4.4.2**

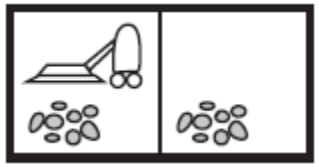
searching with partial observations {4.4.2}

- 对于部分可观察问题，规定环境如何生成Agent的感知信息。可定义吸尘器世界的局部感知为Agent*有位置传感器和局部垃圾传感器，它不能感知其他方格中的垃圾。*

searching with partial observations {4.4.2}

- 部分可观察问题的形式化包括PERCEPT(*s*)函数，返回给定状态的感知信息。如果感知本身是不确定的，则PERCEPT(*s*)函数返回所有可能感知信息的集合。
  - 对于局部感知的吸尘器世界，状态1的PERCEPT(*s*)函数值为[*A, Dirty*]。
  - 对于完全可观察的问题，PERCEPT(*s*)=*s*。
  - 对于无感知信息的问题，PERCEPT(*s*)=*null*。

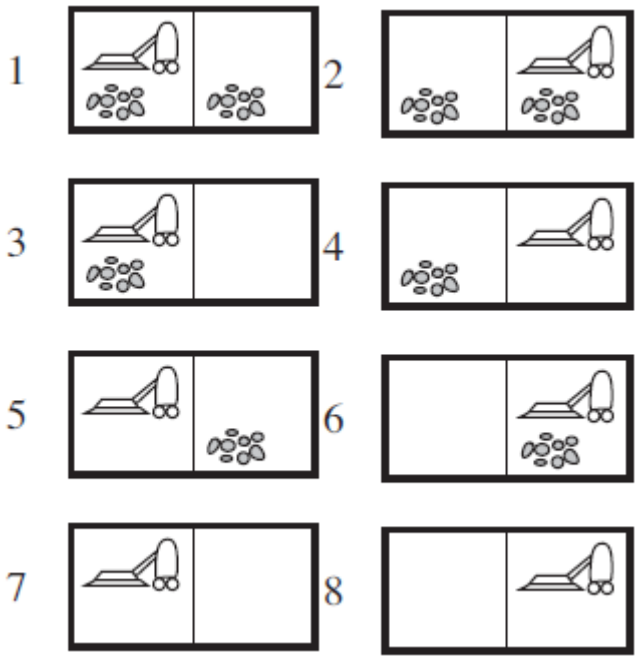
1



[*A, Dirty, unknown*]
[A, Dirty, Dirty]
[unknown, unknown, unknown]

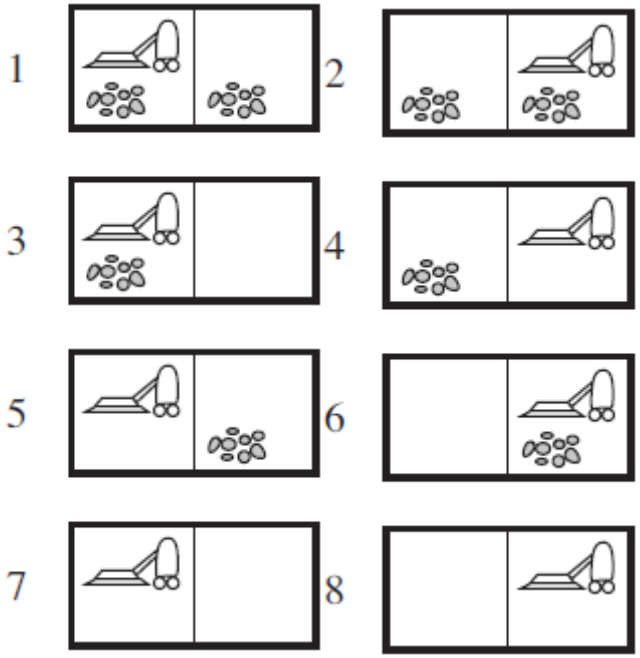
searching with partial observations {4.4.2}

- 当只有局部观察信息，很可能是几个状态都产生给定感知信息。例如，状态1和状态3都产生[*A, Dirty*]。假设这是初始感知信息，初始**信念状态**就是{1, 3}。



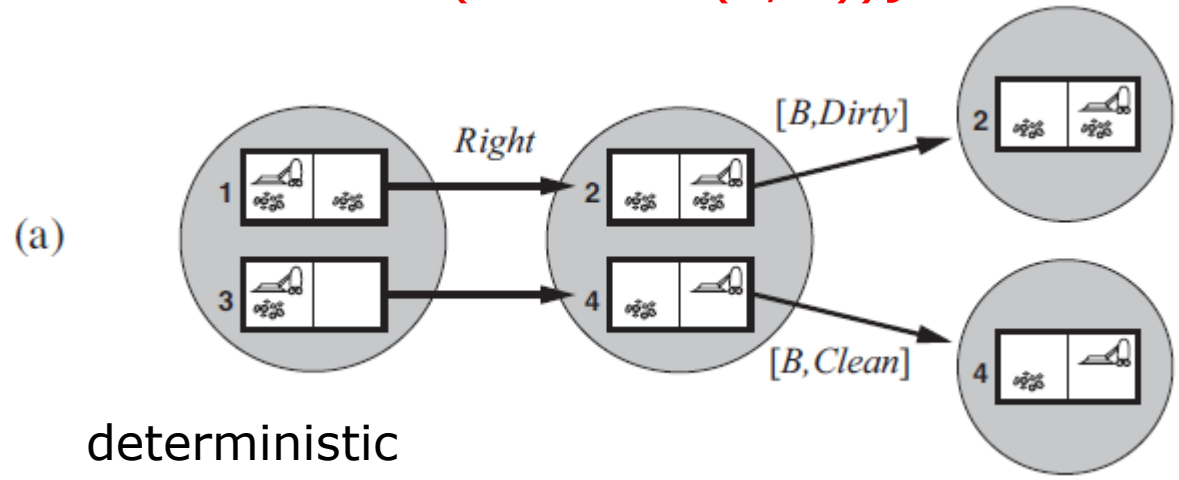
searching with partial observations {4.4.2}

- 就像无感知信息的问题一样，从底层的物理问题可以构造出**信念状态空间**的ACTIONS, STEP-COST, GOAL-TEST, RESULT。
- 对于转移模型，我们可以将导致从一个信念状态转移到另一信念状态的一个特定行动看做是分为三个阶段发生的。



RESULT(b,a)

- **预测**阶段与无感知信息问题相同：给定信念状态b的活动a，预测的信念状态为b'= PREDICT(b,a)。
- **观察预测**阶段确定预测信念状态里可观察到的感知o的集合：POSSIBLE-PERCEPTS(b') = {o: o = PERCEPTS(s) 且 $s \in b'$}。
- **更新**阶段对于每个可能的感知信息确定可能得到的信念状态。新的信念状态b_o是b'中可能产生该感知的状态的集合：b_o = UPDATE(b', o) = {s: o = PERCEPT(s) 且 $s \in b'$}
- 综合考虑这三个阶段，我们对给定行动和一系列可能感知得出可能的信念状态：RESULTS(b, a) = {b_o: b_o = UPDATE(PREDICT(b, a), o) 且 $o \in$POSSIBLE-PERCEPTS(PREDICT(b, a))}.



deterministic

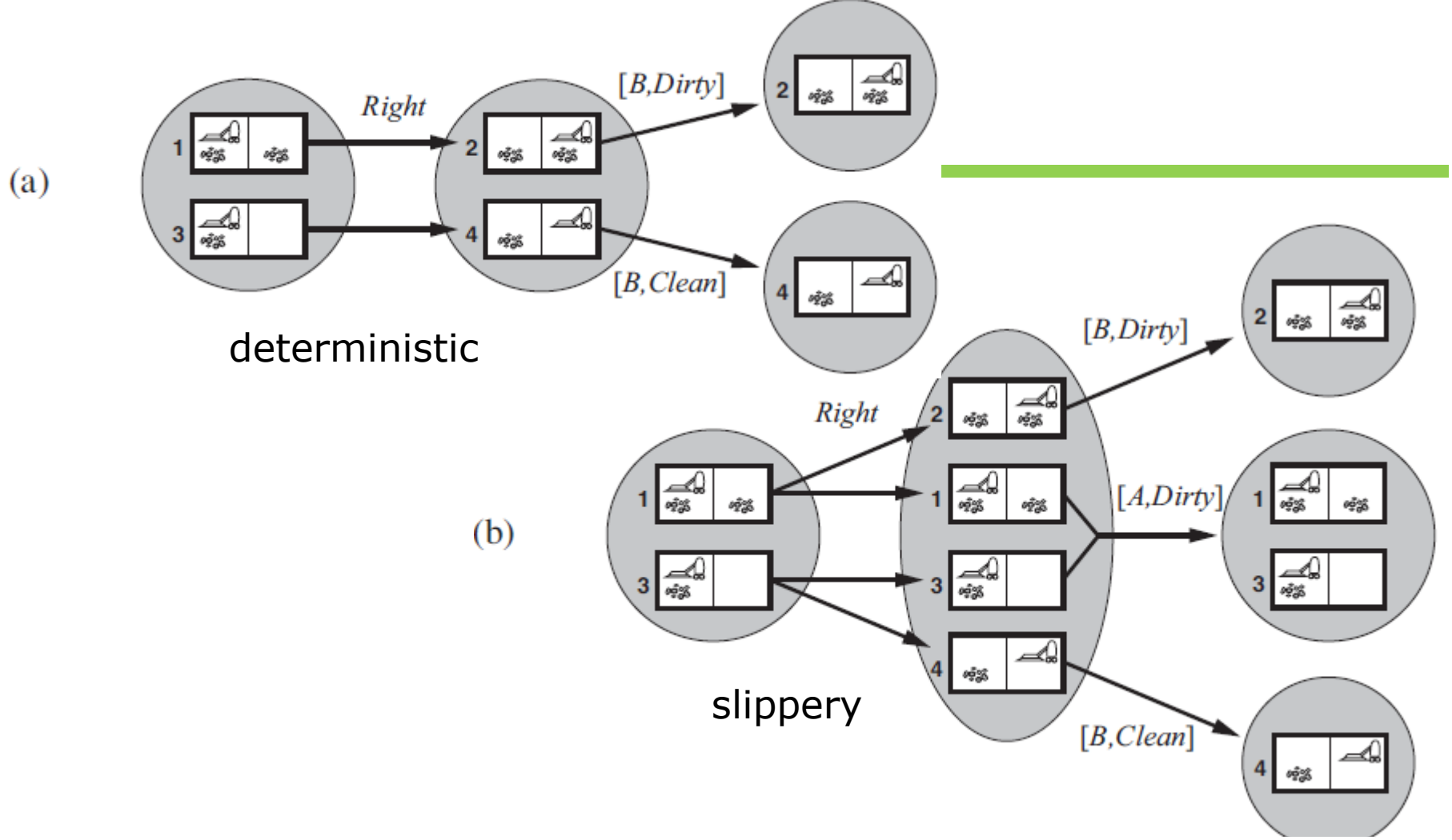


Figure 4.15 Two example of transitions in local-sensing vacuum worlds. (a) In the deterministic world, *Right* is applied in the initial belief state, resulting in a new belief state with two possible physical states; for those states, the possible percepts are [*B*, *Dirty*] and [*B*, *Clean*], leading to two belief states, each of which is a singleton. (b) In the slippery world, *Right* is applied in the initial belief state, giving a new belief state with four physical states; for those states, the possible percepts are [*A*, *Dirty*], [*B*, *Dirty*], and [*B*, *Clean*], leading to three belief states as shown.

test

是动作序列还是应急规划？

可观察性	确定性	解
完全	确定	?
完全	不确定	?
完全不	确定	?
完全不	不确定	?
部分	确定	?
部分	不确定	?

test

是动作序列还是应急规划？

可观察性	确定性	解
完全	确定	动作序列
完全	不确定	应急规划
完全不	确定	动作序列
完全不	不确定	动作序列
部分	确定	应急规划
部分	不确定	应急规划

联机搜索

联机搜索

- **脱机搜索**：计算出完整的方案（然后“闭着眼睛”执行该方案），脱机搜索只涉及计算过程，不涉及动作执行过程。
- **联机搜索**：计算和动作执行交替进行。先采取某个行动，然后观察环境变化并且计算出下一行动。
- Agent不清楚自身的状态和行动的结果，这时的Agent面临联机搜索问题。

联机搜索问题

- 联机搜索知道哪些信息？
  - ACTIONS(s)
  - c(s,a,s’)
  - GOAL-TEST(s)
  - 可能有一个启发式函数h(s)预测s到目标的距离
  - RESULT(s,a),

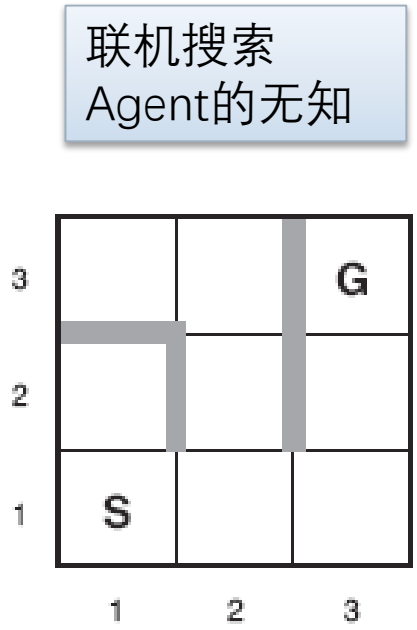
联机搜索问题

- 联机搜索Agent只知道以下信息：
  - ACTIONS(s)
  - c(s,a,s’)，只有当Agent知道行动的结果为s’时才能使用这个函数
  - GOAL-TEST(s)
  - 可能有一个启发式函数h(s)预测s到目标的距离

- 联机搜索Agent不知道：
  - RESULT(s,a), 除非在s执行了a，然后查看结果

联机搜索问题

- 在迷宫问题中，联机Agent并不知道从状态(1, 1)采取行动*Up*能到达状态(1, 2)；即使当完成Up后，Agent也不知道再执行*Down*能回到状态(1, 1),也不知道再执行Up能达到(1,3)
- 在某些应用中可以减少这种无知——例如，机器人探测器也许知道它是如何移动的，无知只是体现在对障碍物位置的认识上。

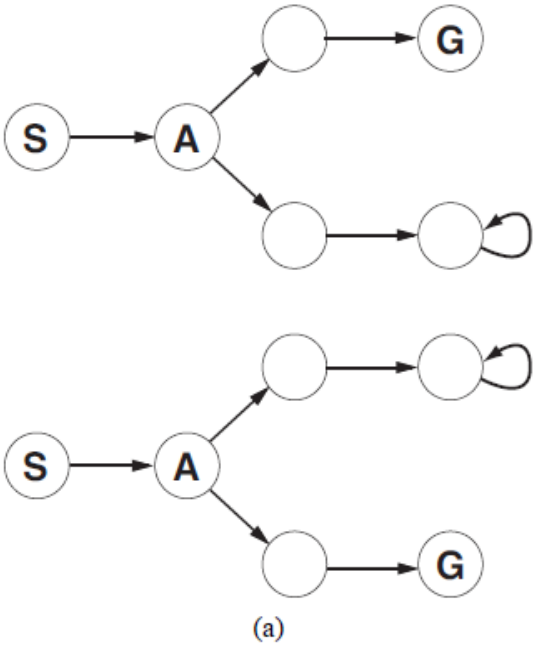


联机搜索问题

- Agent的目标是用最小的代价达到目标状态。（另一个可能的目标是探查整个环境。）代价是指Agent实际经过的路径开销。经常会把这个开销与Agent事先了解搜索空间时应该走的最短路径开销相比较。这被称为**竞争比**，这个值应尽可能小。

联机搜索问题

- 某些情况下能够取得的最好的竞争比也是无穷大。联机搜索有可能进入**死胡同**状态。
- 死胡同是机器人探索中的一个难点——楼梯，斜坡，悬崖和各种各样自然的地形都可能是不可逆行的。



(a)

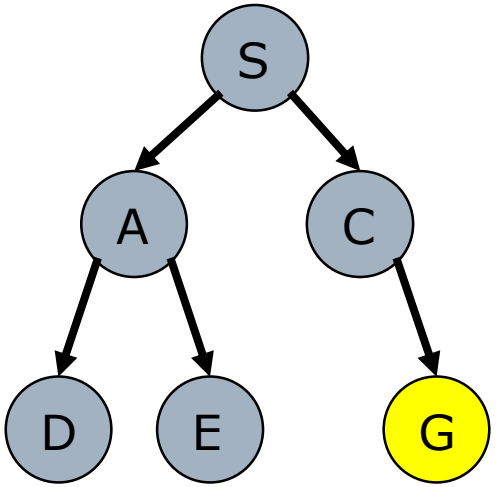
联机搜索Agent

- 脱机算法能在状态空间的一部分扩展一个结点后，马上在状态空间的另一部分扩展另一个结点。

- 联机算法只会扩展它实际占据的结点。为了避免遍历整个搜索树去扩展下一个结点，**按照*局部*顺序扩展结点**看来更好一些，深度优先搜索具有这个性质。————联机深度优先搜索。

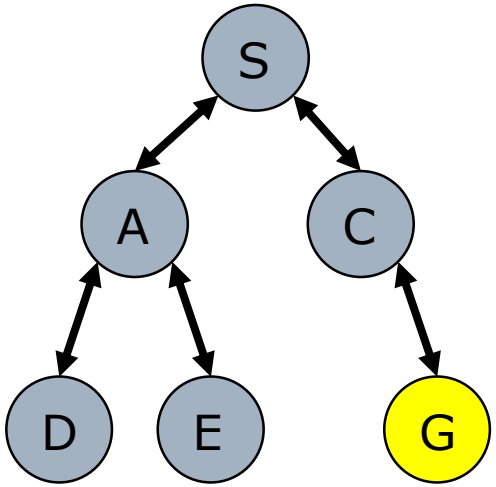
联机搜索Agent

- DFS vs. ONLINE-DFS



脱机深度优先考查顺序
SADECG

只是模拟，不实际经历
状态空间的行动可以不可逆



联机深度优先经历顺序
SADAEASCG

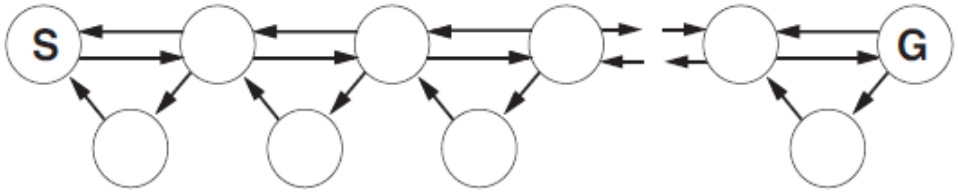
不是模拟，是实际经历
状态空间的行动必须可逆

联机局部搜索

- **爬山法**已经是联机搜索算法！但用处不大，因为Agent可以会陷在局部极大值上而无路可走。而且，还无法应用随机重启算法，因为Agent不能把自己传送到一个新状态。

联机局部搜索

- 可考虑用**随机行走**取代随机重启，以探索环境。随机行走是指在当前状态下随机选择可能的行动之一；选择的时候可以偏向选择那些尚未尝试过的行动。若状态空间有限，随机行走*最终*会找到目标或完成完整探索，但这个过程会很慢。



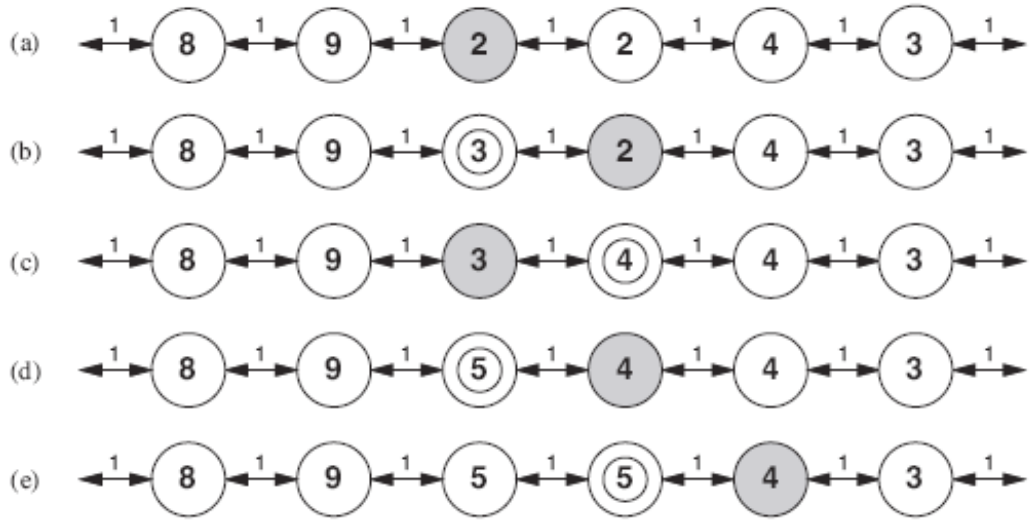
每一步往回走的过程都是向前走的两倍。许多现实世界的状态空间的拓扑结构都能形成此类随机行走的“陷阱”。

联机局部搜索

- 增加爬山法的*内存*而不是随机性是个更有效的方法。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Agent从状态空间中获得的经验再对*H*(*s*)进行更新。

联机局部搜索

- 经过邻居s‘到达目标的估计代价$c(s, a, s')$ + $H(s')$。在(a)，有两个行动选择，估计代价分别为1 + 9和1 + 2，向右移动。此时(a)的阴影状态的H需要更新，如(b)所示。重复这个过程，Agent来回反复两次，每次都更新H并“平滑”局部极小值，直到它向右逃离。



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”$H(s)$

联机搜索中的学习

- 联机搜索Agent初始对环境的无知提供了一些**学习**的机会。
- 首先，Agent通过记录经历来学习RESULT(s,a)。（要注意的是，我们假设环境是确定性的，即每个行动经历一次就足够了。）
- 其次，局部搜索Agent利用局部更新规则可以得到每个状态更精确的估计值。一旦得到了状态的精确值，最优决策就可以简单地选择值最高的后继来完成——纯粹的爬山法是一个最优策略。

联机搜索中的学习

- 观察ONLINE-DFS-AGENT，你会注意到：当它已经知道行动*Up*能从状态(1, 1)到状态(1, 2)时，它并不知道*Down*能回到(1, 1)，或者*Up*还能从(2, 1)到(2, 2)，从(2, 2)到(2, 3)，等等。一般来说，我们希望Agent学习到*Up*在不遇到墙的情况下能使*y*坐标值增加；*Down*则使*y*坐标值降低，等等。

联机搜索中的学习

- 要达到这些必须做两件事情。
- 首先，需要对这类一般规则做规范的、明确的可操作描述；到目前为止，我们把这些信息隐藏在称为RESULT函数的黑盒中。第III部分（知识表示）讨论这个问题。
- 其次，需要有算法能够根据Agent得到的观察信息来构造合适的一般化规则。第18章（样例学习）讨论这个问题。

联机搜索与脱机搜索有何不同？

哪些说法是正确的？

1 脱机：先计算出完整方案，然后执行该完整方案。

2 脱机：规划和动作执行没有交替

3 脱机：扩展一个结点后马上可以跳到另一个分支的结点进行扩展

4 联机：执行完一个动作后，观察环境变化，计算出下一个动作，再执行这个动作，如此循环

5 联机：规划和动作执行是交替的

6 联机：只能扩展实际占据的结点

超越经典搜索

1 何谓局部搜索

2 局部搜索策略：(最陡)爬山法、随机爬山法、首选爬山法、随机重启爬山法；模拟退火算法；局部束搜索；遗传算法；连续空间怎么办？

3 动作具有不确定性时、无传感器时、部分可观察时，怎么搜索？

4 联机搜索

summary {4.6}

- 本章介绍的搜索算法超越了在确定的、可观察的和离散环境下寻找目标路径的“经典”情况。
- *局部搜索*方法如**爬山法**适用于完整状态形式化，它在内存中只保留少量结点信息。随机算法正在研究中，包括**模拟退火**，当给定合适的冷却调度安排时能够返回最优解。
- 局部搜索方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**杂交**和**变异**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，无论执行过程中产生怎样的后果。
- 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。
- 标准的搜索算法可直接应用于信念状态空进行无感知问题求解，信息状态AND-OR搜索可以解决一般部分可观察问题。在信念状态空间中逐个状态构造解的增量算法通常效率更高。
- **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。

Thanks!

next:
Adversarial Search