第2章 问题求解与搜索2

南京信息工程大学
计算机与软件学院

应龙

2024年秋季

主要内容

5. 启发式函数 (Heuristic Functions)

6. Local Search and Optimization Problems

7. Evolutionary Algorithm

8. Search with Nondeterministic Actions or in Partially Observable Environments *

9. Online Search Agents and Unknown Environments *

Bibliography:

- ✓ Stuart J. Russel, Peter Norvig, “Artificial Intelligence: A Modern Approach” (4th Ed. 2020); 中译版 “人工智能 现代方法” (4rd Ed., 2022), 人民邮电出版社. Ch 3.6, Ch 4
- ✓ 王万良 编著, “人工智能导论” (第5版), 高等教育出版社, 2020. Ch 6

主要内容

5. 启发式函数 (Heuristic Functions)

6. Local Search and Optimization Problems

7. Evolutionary Algorithm

8. Search with Nondeterministic Actions or in Partially Observable Environments *

9. Online Search Agents and Unknown Environments *

Bibliography:

- ✓ Stuart J. Russel, Peter Norvig, “Artificial Intelligence: A Modern Approach” (4th Ed. 2020); 中译版“人工智能 一种现代方法” (4rd Ed., 2022), 人民邮电出版社. Ch 3.6.

Heuristic Functions

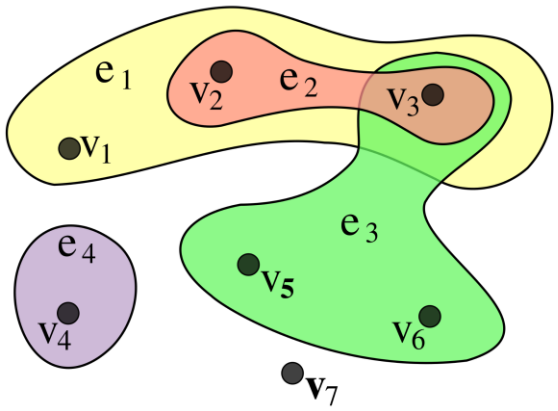
- 从松弛问题 (relaxed problems) 设计可容许的启发式 (admissible heuristics)

  ① 对任意结点n，有$h(n) \le h^*(n)$；

  ② 如果n是目标结点，则有$h(n) = 0$。

减少了动作的限制条件(restrictions)的问题称为松弛问题(relaxed problem)。松弛问题的状态空间图是原有状态空间的超图(supergraph)，原因是减少限制导致图中边的增加。

由于松弛问题增加了状态空间的边，原有问题中的任一最优解同样是松弛问题的解；但是松弛问题可能存在更好的解，理由是增加的边可能导致捷径。所以，一个松弛问题的最优解代价是原问题的可采纳的启发式。更进一步，由于得出的启发式是松弛问题的确切代价，那么它一定遵守三角不等式，因而是一致的。

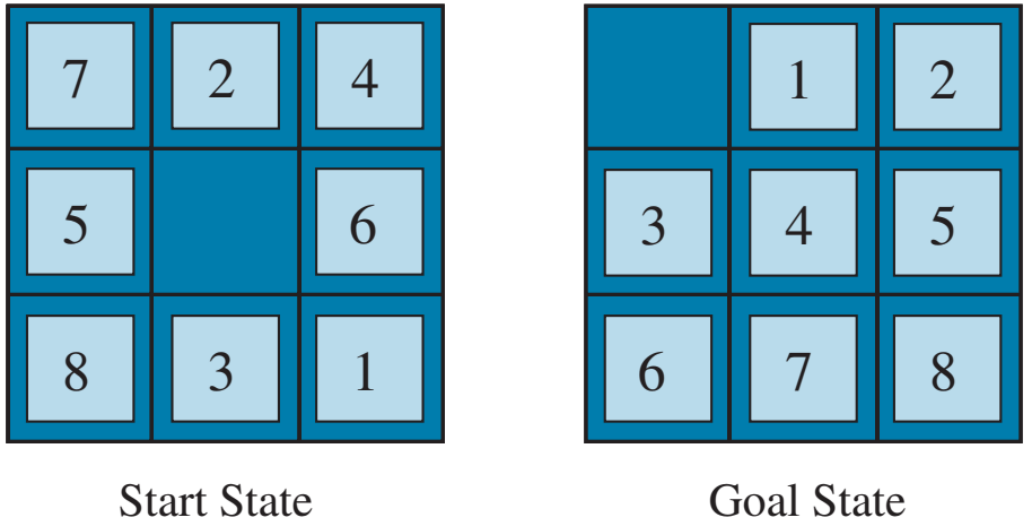


原有问题宽松的界限

启发式函数

- 从松弛问题 (relaxed problems) 设计可容许的启发式 (admissible heuristics)

如果问题定义是用形式语言描述的，那么有可能来自动构造它的松弛问题。



Start State　　Goal State

约束：棋子可以从方格A移动到方格B，如果A与B水平或竖直相邻而且B是空的。

可以去掉其中一个或者两个条件，生成三个松弛问题：

(a) 棋子可以从方格A移动到方格B,如果A和B相邻。

(b) 棋子可以从方格A移动到方格B,如果B是空的。

(c) 棋子可以从方格A移动到方格B。

(a) ⇒ h_2 曼哈顿距离。

(b) ⇒ Gaschnig 启发式

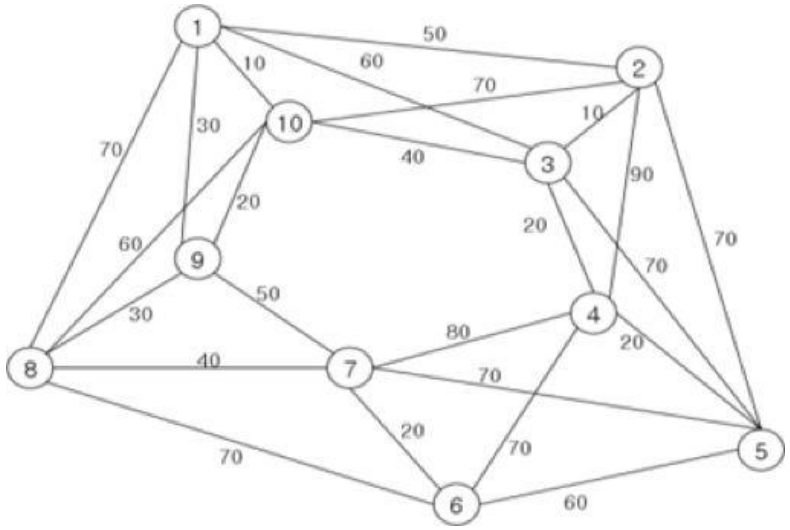
(c) ⇒ h_1 不在位的棋子数。

Ensemble:

$$h(n) = \max\{h_1(n), \ldots, h_m(n)\}$$

启发式函数

旅行商问题



$$C^* = \min \sum_{(i,j)\in A} c_{ij} \cdot x_{ij},$$

(1) 代价函数

s.t.

$$\sum_{(i,j)\in A} x_{ij} = 1,\ i = 1, \cdots, n,$$

(2) 结点的出度

$$\sum_{(i,j)\in A} x_{ij} = 1,\ j = 1, \cdots, n,$$

(3) 结点的入度

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leqslant |S| - 1,$$

$$S \subset \{1, 2, \cdots, n\},\ 2 \leqslant |S| \leqslant n - 2,$$

(4)子回路消除约束

$$x_{ij} \in \{0, 1\},\ i, j \in A,$$

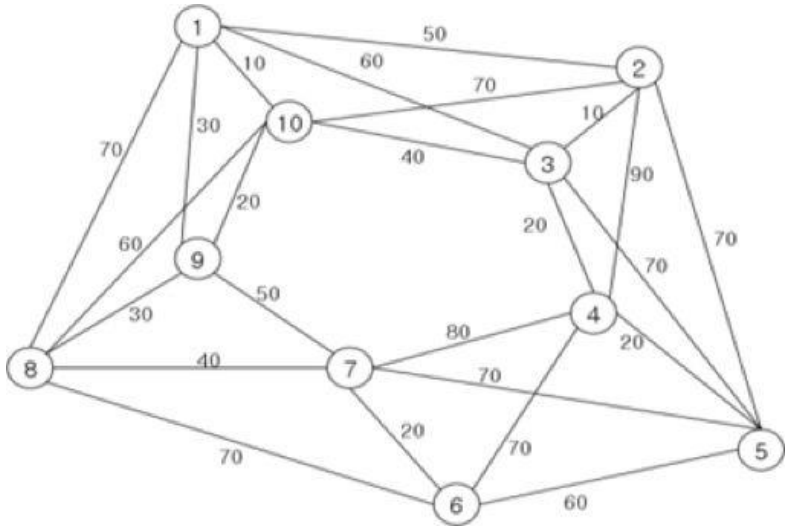
(5)边的取值

分支定界法的上界一般是通过 search tree 或 heuristic，记录下并更新 global 可行解来得到的。

分支定界法的下界，是以原问题为基础，减少部分 constraints 然后求解简化版原问题的解决方法，如减少0-1 constraint，把原问题变成LP问题，减少每个点的度都必须为2的constraint把原问题变成最小权值1-Tree问题。两种解下界的方法：简化为最小权值1-Tree问题，简化为分配问题

启发式函数

旅行商问题



$$C^* = \min \sum_{(i,j)\in A} c_{ij} \cdot x_{ij},$$

s.t.

$$\sum_{(i,j)\in A} x_{ij} = 1,\ i = 1, \cdots, n,$$

(2) 结点的出度

$$\sum_{(i,j)\in A} x_{ij} = 1,\ j = 1, \cdots, n,$$

(3) 结点的入度

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leqslant |S| - 1,$$

$$S \subset \{1, 2, \cdots, n\},\ 2 \leqslant |S| \leqslant n - 2,$$

(4)子回路消除约束

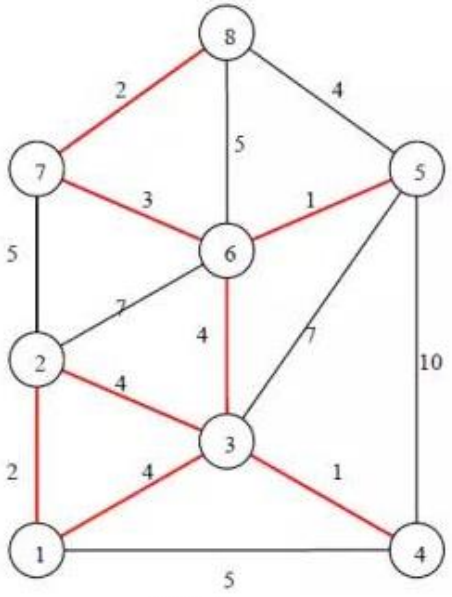
$$x_{ij} \in \{0, 1\},\ i, j \in A,$$

(5)边的取值

(1) 代价函数

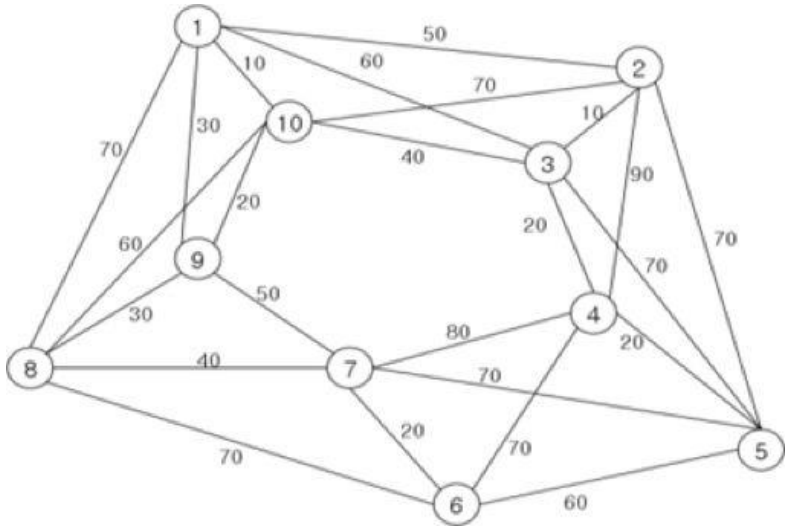
在一个图G(V,E)中，节点集合V = {1...n}，我们定义{2...n}节点组成的子图的生成树以及两条与1节点的边组成的新图为1-tree。

TSP的可行解是1-tree的一种。一棵1-tree是一个TSP的可行解的充要条件是1-tree中所有节点的度(degree)均为2。



启发式函数

旅行商问题



$$C^* = \min \sum_{(i,j)\in A} c_{ij} \cdot x_{ij},$$

(1) 代价函数

s.t.

$$\sum_{(i,j)\in A} x_{ij} = 1,\ i = 1, \cdots, n,$$

(2) 结点的出度

$$\sum_{(i,j)\in A} x_{ij} = 1,\ j = 1, \cdots, n,$$

(3) 结点的入度

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \leqslant |S| - 1,$$

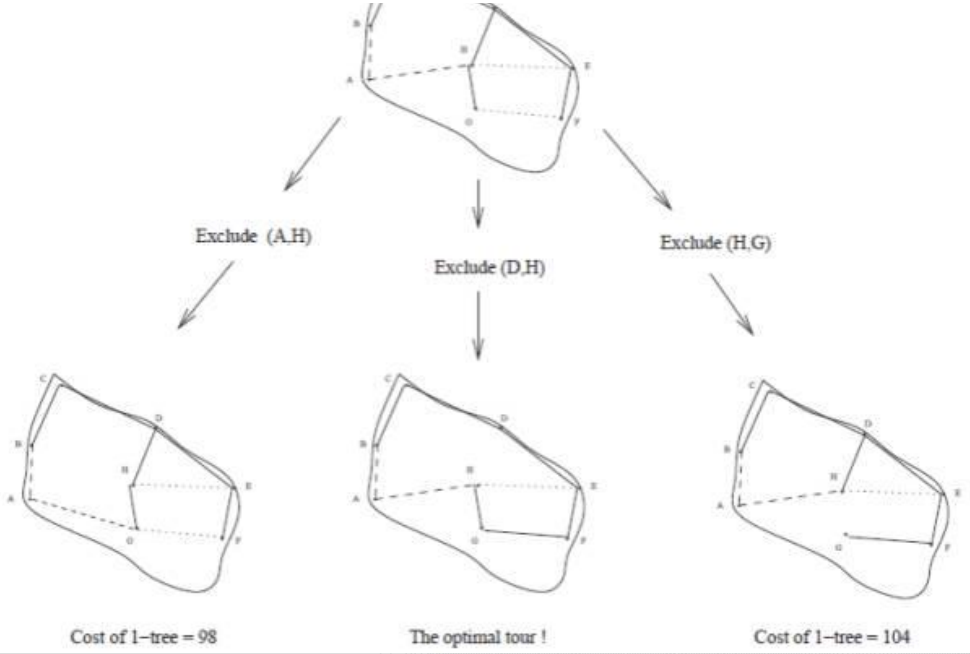
$$S \subset \{1, 2, \cdots, n\},\ 2 \leqslant |S| \leqslant n-2,$$

(4)子回路消除约束

$$x_{ij} \in \{0, 1\},\ i, j \in A,$$

(5)边的取值

分枝方法：寻找1-tree中所有度大于等于3的节点，枚举并依次删除这个节点所有的边，依次求解最小权值1-tree，直到找到可行的TSP解。



启发式函数

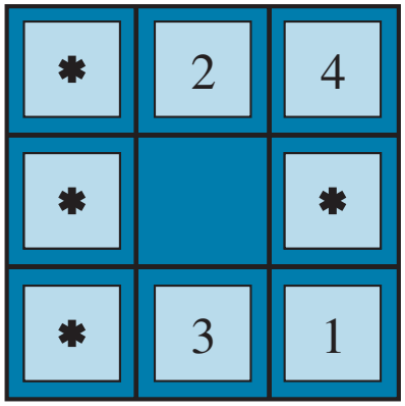
- 从子问题设计可容许的启发式：模式数据库

Admissible heuristics can also be derived from the solution cost of a subproblem of a given problem.

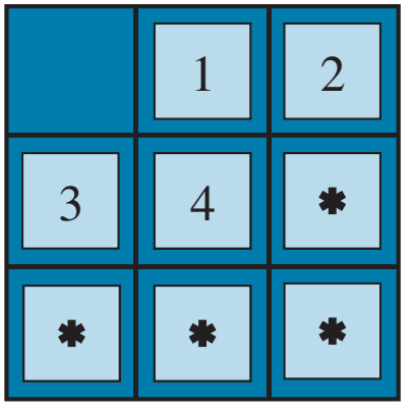
模式数据库 (pattern databases) 的思想是为每个可能的子问题实例存储解代价。通过在数据库里查找出相应的子问题，对搜索中遇到的每个状态计算一个可容许的启发式函数 h_{DB}。

数据库本身是从目标状态反向搜索并记录所遇到的每个新模式的代价来构建的；这一搜索的开销分摊到后续问题的实例中。

不相交模式数据库 (disjoint pattern database)：对于1-2-3-4 数据库和 5-6-7-8 数据库，不记录求解 1-2-3-4 子问题的总代价，只记录与1-2-3-4 有关的操作数。这两个代价的和仍然是求解完整问题代价的一个下界。



Start State



Goal State

定义的子问题

启发式函数

- 从经验中学习启发式 (Learning heuristics from experience)

An alternative is to learn from experience. “Experience” here means solving lots of 8-puzzles, for instance.

Each optimal solution to an 8-puzzle problem provides an example $(\text{goal}, \text{path})$ pair. From these examples, a learning algorithm can be used to construct a function h that can (with luck) approximate the true path cost for other states that arise during search.

Most of these approaches learn an imperfect approximation to the heuristic function, and thus risk inadmissibility.

如果在原始的状态描述外还能提供与预测状态启发值相关的特征，一些机器学习技术工作的更好。

- $x_1(n)$ 不在位滑块数
- $x_2(n)$ 在当前状态相邻而目标状态不相邻的滑块数量

设计的特征 (hand-crafted feature) → 特征学习 (feature learning)

启发式函数

◆ Learning to search better

一个智能体 (Agent) 可以学习如何更好地进行搜索吗？

Metalevel state space: each state in a metalevel state space captures the internal (computational) state of a program that is searching in an ordinary state space.

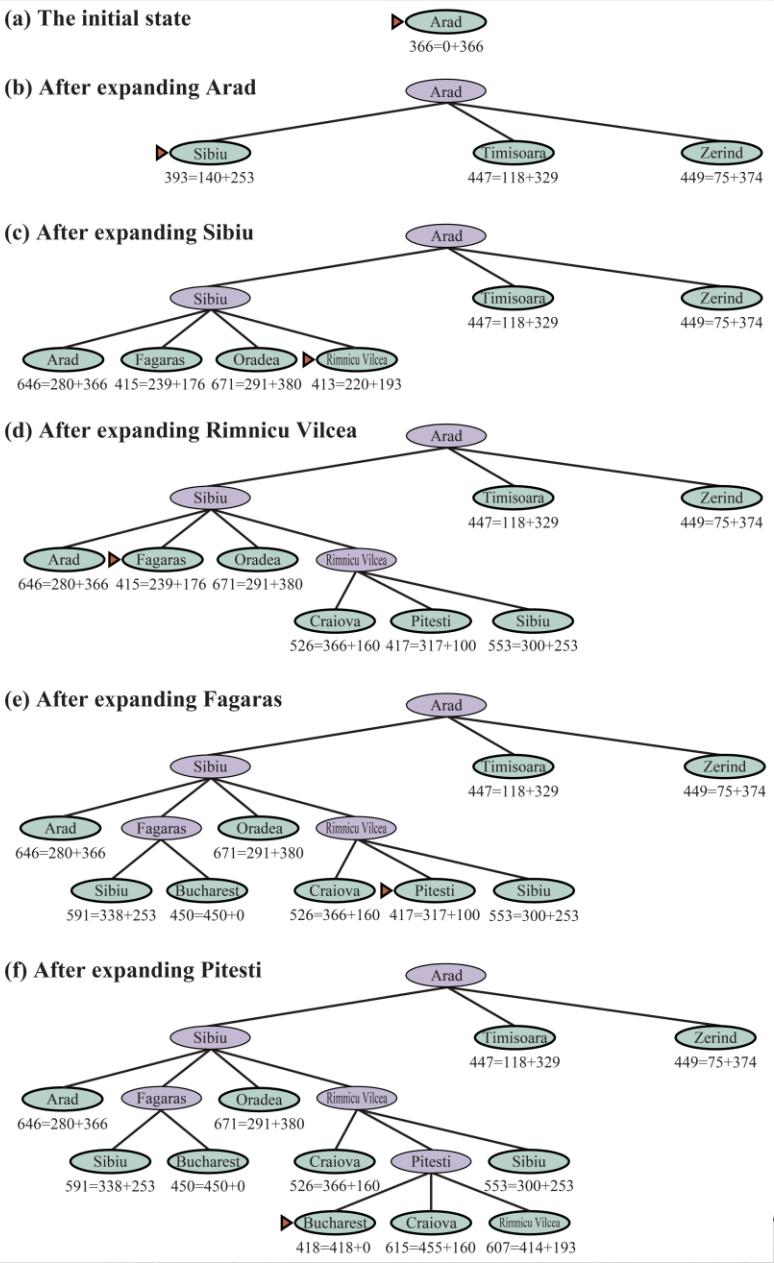
Each action in the metalevel state space is a computation step that alters the internal state.

A metalevel learning algorithm can learn from these experiences to avoid exploring unpromising subtrees.

The goal of learning is to minimize the total cost of problem solving, trading off computational expense and path cost.

Bibliography:

✓ Stuart J. Russel, Peter Norvig, “Artificial Intelligence: A Modern Approach (4th Ed. 2020)”; Chapter 3.6.5



主要内容

5. 启发式函数 (Heuristic Functions)

6. Local Search and Optimization Problems

7. Evolutionary Algorithm

8. Search with Nondeterministic Actions or in Partially Observable Environments *

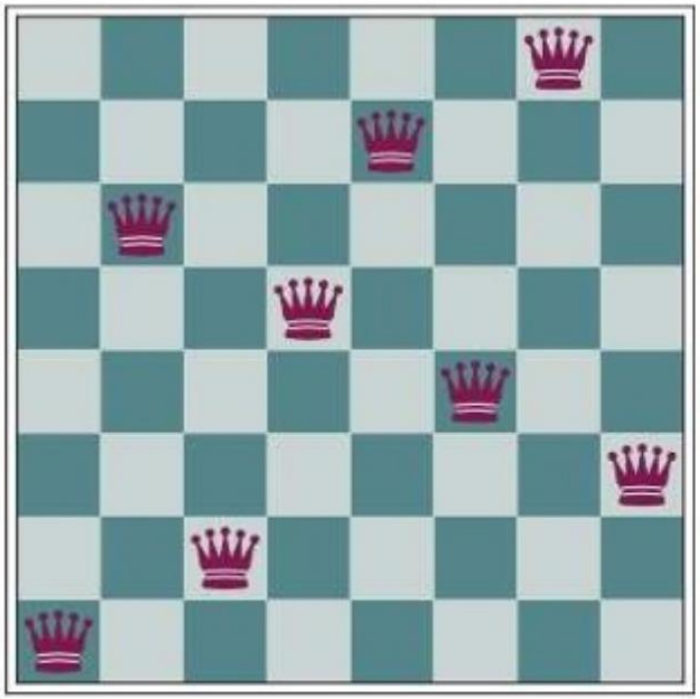
9. Online Search Agents and Unknown Environments *

Bibliography:

- ✓ Stuart J. Russel, Peter Norvig, “Artificial Intelligence: A Modern Approach” (4th Ed. 2020); 中译版“人工智能 一种现代方法”(4rd Ed., 2022), 人民邮电出版社. Ch 4.1, Ch4.2

局部搜索和最优化问题

前面介绍过的搜索算法都系统地探索空间。这种系统化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，到达此目标的路径就是这个问题的一个解。然而在许多问题中，只关心最终的状态，不关心到达的路径。



八皇后问题 (The 8-queens problem)
集成电路设计、工厂场地布局、作业车间调度、自动程序设计、电信网络优化、农作物规划和投资组合管理 (portfolio management)。

局部搜索 (Local search) 算法从一个起始（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。 局部搜索算法不是系统化的 (systematic)。They might never explore a portion of the search space where a solution actually resides.

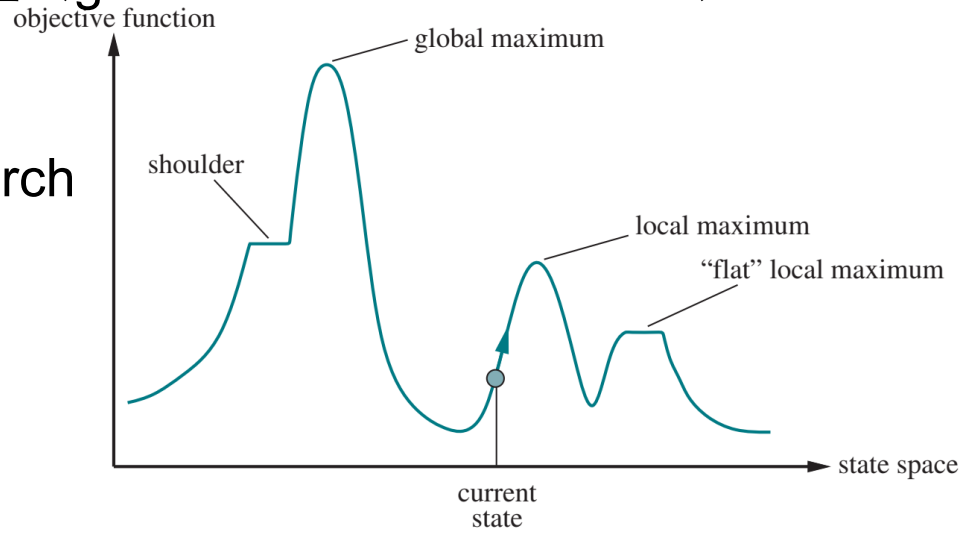
两个关键的优点：
1）它们只用很少的内存（通常是常数）；
2）它们经常能在系统化算法不适用的很大或无限的（连续的） 状态空间中找到合理的解。

Hill-climbing search

状态空间地形图（state-space landscape）：状态（state）+ 标高（elevation, 由目标函数 objective function 的值定义）。
寻找全局最小值 或 全局最大值

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（global minimum/maximum）。

爬山法 Hill-climbing search
Steepest ascent
Gradient descent



function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum
  current ← *problem*.INITIAL
  while *true* **do**
    neighbor ← a highest-valued successor state of *current*
    if VALUE(*neighbor*) ≤ VALUE(*current*) **then return** *current*
    current ← *neighbor*

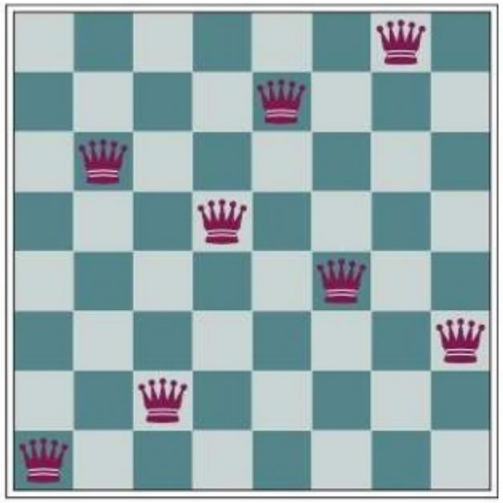
Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor.

Hill-climbing search

八皇后问题：complete-state formulation

The initial state is chosen at random, and the successors of a state are all possible states generated by moving a single queen to another square in the same column.

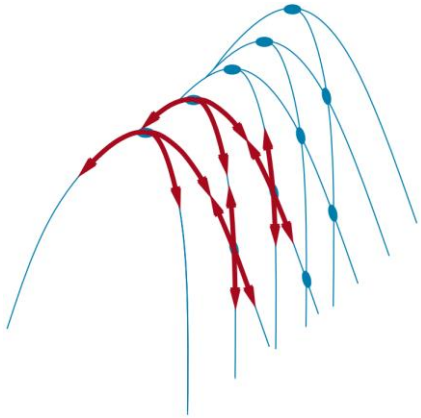
启发式代价函数 (The heuristic cost function) h 是形成相互攻击的皇后对的数量，该函数的全局最小值是0，仅在找到解时才会是这个值。



爬山法有时被称为贪婪局部搜索（greedy local search）

局部极大值 (Local maxima)，山脊 (Ridges)，高原 (Plateaus)

- Stochastic hill climbing
- First-choice hill climbing
- Random-restart hill climbing



The grid of states (dark circles)

模拟退火 Simulated annealing

爬山法 (Steepest ascent) 搜索不会向值比当前结点低的 （或代价高的）方向搜索， 是不完备的。纯粹的随机游走 (random walk) ——从后继集合中完全等概率的随机选取后继——是完备的， 但是效率极低。

在冶金中， 退火(annealing) 是用于增强金属和玻璃的韧性或硬度。先把它们加热到高温再让它们逐渐冷却的过程， 这样能使材料到达低能量的结晶态。

模拟退火 (Simulated annealing) 算法的内层循环与爬山法类似。只是它没有选择最佳移动， 选择的是随机移动。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。如果移动导致状态“变坏”， 概率则成指数级下降。 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。如果调度程序 (schedule) 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
  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 $e^{-\Delta E/T}$

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. The schedule input determines the value of the “temperature” T as a function of time.

Local Beam Search

局部束搜索 (local beam search) 算法记录k个状态而不是只记录一个。 它从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， 它从整个后继列表中选择k个最佳的后继，重复这个过程。

如果是最简单形式的局部束搜索，那么由于这k个状态缺乏多样性——它们很快会聚集到状态空间中的一小块区域内。

随机束搜索 (stochastic beam search) 为解决此问题的一种变形，它与随机爬山法相类似。随机束搜索并不是从候选后继集合中选择最好的k个后继状态，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。

Local Search in Continuous Spaces

设$f(\mathbf{x})$为目标函数或评估函数

$$\nabla f = 0 \qquad \mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x})$$

其中 α 是很小的常数，称为步长。一些情况下，目标函数可能无法用微分形式表示。可以通过评估每个坐标上小的增减带来的影响来决定所谓的经验梯度。

对于许多问题，最有效的算法是 Newton-Raphson Method。这是找到函数的根的一般方法，使用函数$g(x)$的泰勒级数的前2项求解 $g(x) = 0$ 的根x 。

$$0 = g(x) = g(x_n) + \nabla g(x_n)(x - x_n) + r((x - x_n)^2)$$

迭代公式为：

$$x \leftarrow x - (\nabla g)^{-1}(x) \cdot g(x) \quad \text{令 } g(x) = \nabla f(\boldsymbol{x}) \Rightarrow \mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\boldsymbol{x})$$

其中$\mathbf{H}_f(\mathbf{x})$ 是二阶导数的 Hessian 矩阵， 矩阵中的元素H_{ij}的值由 $\frac{\partial^2 f}{\partial x_i \partial x_j}$ 给出。

Constrained optimization:

The difficulty of constrained optimization problems depends on the nature of the constraints and the objective function.

Convex optimization, which allows the constraint region to be any convex region and the objective to be any function that is convex within the constraint region.

主要内容

5. 启发式函数 (Heuristic Functions)

6. Local Search and Optimization Problems

7. Evolutionary Algorithm

8. Search with Nondeterministic Actions or in Partially Observable Environments *

9. Online Search Agents and Unknown Environments *

◆ 遗传算法 (genetical algorithm, GA)

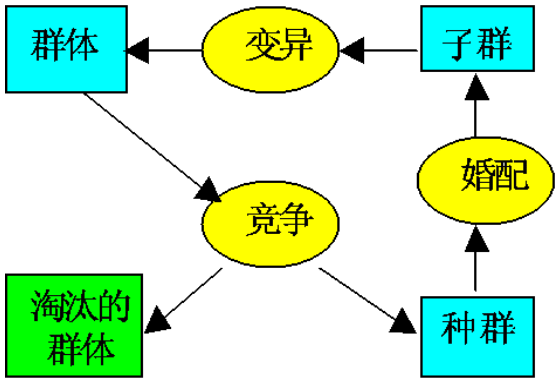
Bibliography:

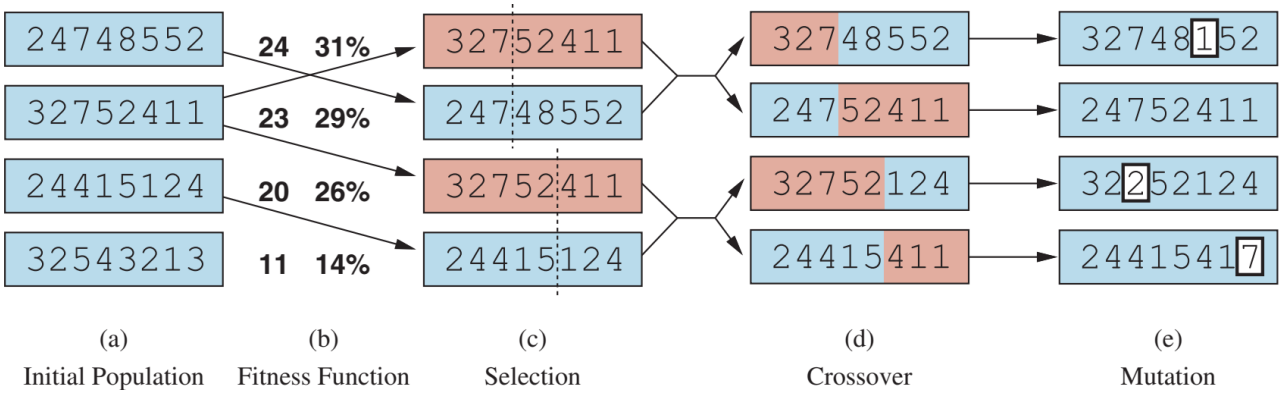
✓ 王万良 编著, “人工智能导论” (第5版), 高等教育出版社, 2020. Ch 6

✓ Stuart J. Russel, Peter Norvig, “Artificial Intelligence: A Modern Approach” (4th Ed. 2020); 中译版“人工智能 一种现代方法”(4rd Ed., 2022), 人民邮电出版社. Ch 3.6, Ch 4.1.4

Genetic algorithm

遗传算法 (genetical algorithm, GA) 是随机束搜索的一个变体，直接受到生物学中自然选择的启发：在由个体（状态）组成的种群中，适应度高（评价函数值高）的个体产生下一代（后继状态），这个过程称为重组 (recombination)。

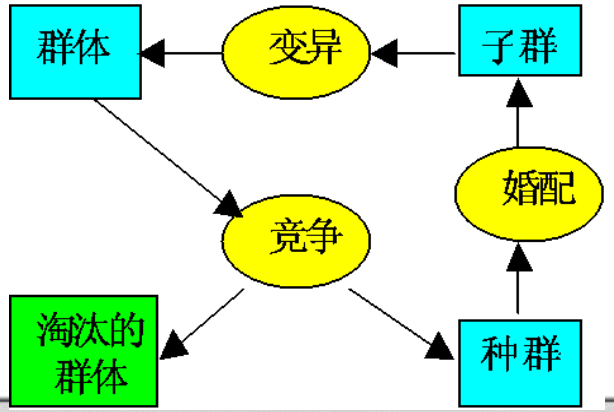




- 遗传算法从k个随机生成的状态开始，称为种群，像束搜索一样。每个状态，或称为个体，用一个有限长度的编码表示。
- 每个状态都由它的目标函数或（用遗传算法术语）适应度 (fitness) 函数给出评估值。对于好的状态，适应度函数应返回较高的值。
- 按照一定规则选择若干个体进行繁殖。并根据定义的方法进行交叉 (crossover)，或称为重组 (recombination)，产生新的个体表示。
- 个体表示每个位置可能按照某个小的独立概率随机变异。
- 产生下一代种群。

Genetic algorithm

生物遗传的概念	遗传算法中的应用
适者生存	目标值比较大的解被选择的可能性大
个体 (Individual)	状态或解
染色体 (Chromosome)	状态或解的编码（字符串、向量等）
基因 (Gene)	状态或解编码中的分量
适应性 (Fitness)	适应度函数值
群体 (Population)	根据适应度值选定的一组状态（个数为群体的规模）
婚配 (Marry)	选择两个染色体进行重组
繁殖 (Reproduction)	两个染色体重组（交叉）产生一组新的染色体的过程
变异 (Mutation)	编码的某一分量发生变化的过程



Genetic algorithm

```
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))
```

Figure 4.8 A genetic algorithm. Within the function, *population* is an ordered list of individuals, *weights* is a list of corresponding fitness values for each individual, and *fitness* is a function to compute these values.

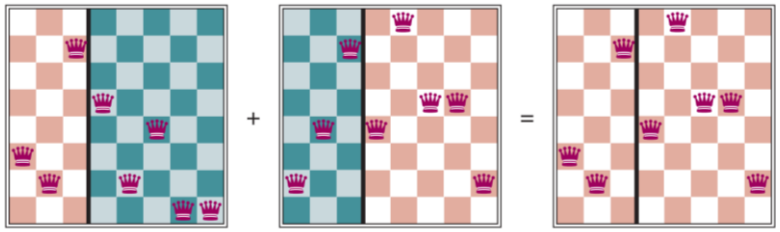


Figure 4.7 The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d). The green columns are lost in the crossover step and the red columns are retained. (To interpret the numbers in Figure 4.6: row 1 is the bottom row, and 8 is the top row.)

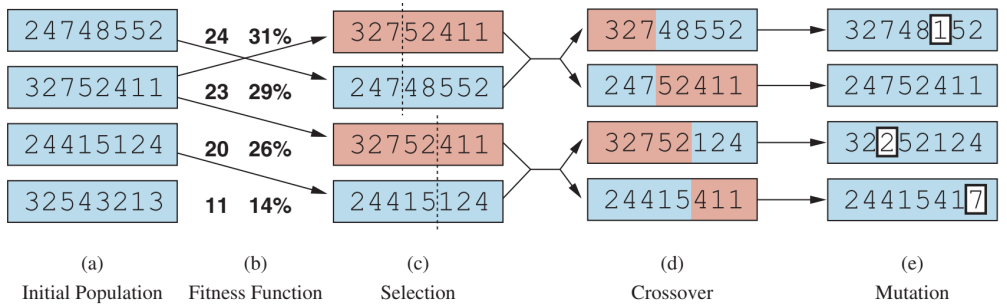


Figure 4.6 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic algorithm

- 个体的表示 (The representation of each individual) 编码 (encoding)

在遗传算法中，每个个体通常为有限字母表上的字符串（通常是布尔字符串）。在演化策略 (evolution strategies) 中，个体是实数序列，而在遗传编程 (genetic programming) 中，个体是计算机程序。

- 1. 位串编码

将问题空间的参数编码为一维排列的染色体的方法，称为一维染色体编码方法。

- 二进制编码 (binary encoding)

  ① 相邻整数的二进制编码可能具有较大的Hamming距离，降低了遗传算子的搜索效率。 ② 要先给出求解的精度。③ 求解高维优化问题的二进制编码串长，算法的搜索效率低。

- 格雷编码 (Gray encoding) 将二进制编码通过变换得到的编码

- 2. 实数编码 (real encoding)

用若干实数表示一个个体，然后在实数空间上进行遗传操作。
可直接在状态或解的表现型上进行遗传操作，可引入启发式信息。

- 3. 多参数级联编码　应用于多参数优化问题

基本思想：把每个参数先进行二进制编码得到子串，再把这些子串连成一个完整的染色体。多参数映射编码中的每个子串对应各自的编码参数，所以，可以有不同的串长度和参数的取值范围。

Genetic algorithm

种群 (population) 设定

- 初始种群的产生
  - 随机产生群体规模数目的个体作为初始群体。
  - 随机产生一定数目的个体，从中挑选最好的个体加到初始群体中。这种过程不断迭代，直到初始群体中个体数目达到了预先确定的规模。
  - 根据问题固有知识，把握最优解所占空间在整个问题空间中的分布范围，然后，在此分布范围内设定初始群体。

- 种群规模 (The size of the population) 的确定
  - 种群规模影响遗传优化的结果和效率。
  - 群体规模太小，遗传算法的优化性能不太好，易陷入局部最优解。群体规模太大，计算复杂。
  - 模式定理表明：若群体规模为M，则遗传操作可从这M个个体中生成和检测M^3个模式，并在此基础上能够不断形成和优化积木块，直到找到最优解。
  - 一般取为20-100

Genetic algorithm

适应度函数 (fitness function)

- 将目标函数映射为适应度函数

若目标函数为最大化问题，则 $Fit(f(x)) = f(x)$

若目标函数为最小化问题，则 $Fit(f(x)) = 1/f(x)$

将目标函数转换为求最大值的形式,且保证函数值非负！

若目标函数为最大化问题，则

$$Fit(f(x)) = \begin{cases} f(x) - C_{\min} & f(x) > C_{\min} \\ 0 & \text{其他情况} \end{cases}$$

若目标函数为最小化问题，则

$$Fit(f(x)) = \begin{cases} C_{\max} - f(x) & f(x) < C_{\max} \\ 0 & \text{其他情况} \end{cases}$$

- 适应度函数的尺度变换
  - 在遗传算法中，将所有妨碍适应度值高的个体产生，从而影响遗传算法正常工作的问题统称为欺骗问题 (deceptive problem)。
  - 过早收敛：出现大大超过群体平均适应度的超级个体。应缩小这些个体的适应度，以降低这些超级个体的竞争力。
  - 停滞现象：搜索后期，群体的平均适应值可能会接近群体的最优适应度值，实际已不存在竞争。改变原始适应值的比例关系，提高个体之间的竞争力。
  - 尺度变换 (fitness scaling) 或定标：对适应度函数值域的某种映射变换。

Genetic algorithm

- 适应度函数 (fitness function)
- 适应度函数的尺度变换
  - ①线性变换　　$f' = af + b$ 满足

    $f'_{avg} = f_{avg}$　变换后适应度的平均值

    $f'_{max} = C_{mult} \cdot f_{avg}$　变换后适应度函数的最大值，控制适应度最大的个体在下一代中的复制数

$$a = \frac{(C_{mult} - 1) f_{avg}}{f_{\max} - f_{avg}}$$

$$b = \frac{(f_{\max} - C_{mult} f_{avg}) f_{avg}}{f_{\max} - f_{avg}}$$

满足最小适应度值非负

$$a = \frac{f_{avg}}{f_{avg} - f_{\min}}$$

$$b = \frac{-f_{\min} f_{avg}}{f_{avg} - f_{\min}}$$

  - ②非性变换

    幂函数变换　　$f' = f^K$

    指数函数变换　　$f' = e^{-\alpha f}$　　基本思想来源于模拟退火过程，式中的系数α决定了复制的强制性，其值越小复制的强制性就越趋向那些具有最大适应度的个体

Genetic algorithm

- 重组 (recombination)
  - 基本的交叉算子
    - 一点交叉 (single-point crossover)：在个体串中随机设定一个交叉点，实行交叉时，该点前或后的两个个体的部分结构进行互换，并生成两个新的个体。
    - 两点交叉 (two-point crossover)：随机设置两个交叉点，将两个交叉点之间的码串相互交换。
  - 修正的交叉方法
    - 由于交叉，可能出现不满足约束条件的非法染色体。 ①构造惩罚函数。②对遗传操作进行适当的修正，使其满足优化问题的约束条件。
    - 旅行商 (TSP) 问题中采用部分匹配交叉 (partially matched crossover, PMX)，顺序交叉 (order crossover, OX) 和循环交叉 (cycle crossover, CX) 等。

部分匹配交叉 (PMX) Goldberg D. E. 和R. Lingle(1985)

$$A = 9\ \ 8\ \ 4\ \ |\ \ 5\ \ 6\ \ 7\ \ |\ \ 1\ \ 3\ \ 2 \qquad A' = 9\ \ 8\ \ 4\ \ |\ \ 2\ \ 3\ \ 9\ \ |\ \ 1\ \ 3\ \ 2$$

$$B = 8\ \ 7\ \ 1\ \ |\ \ 2\ \ 3\ \ 9\ \ |\ \ 5\ \ 4\ \ 6 \qquad B' = 8\ \ 7\ \ 1\ \ |\ \ 5\ \ 6\ \ 7\ \ |\ \ 5\ \ 4\ \ 6$$

$$A'' = 7\ \ 8\ \ 4\ \ |\ \ 2\ \ 3\ \ 9\ \ |\ \ 1\ \ 6\ \ 5$$

$$B'' = 8\ \ 9\ \ 1\ \ |\ \ 5\ \ 6\ \ 7\ \ |\ \ 2\ \ 4\ \ 3$$

- 交叉概率P_c是用来确定两个染色体进行局部互换已产生两个新的子代的概率

Genetic algorithm

- 选择 (selection process)

从当前群体中按照一定概率选出优良的个体，使它们有机会作为亲代繁殖下一代孩子。判断个体优良与否的准则是各个个体的适应度值：个体适应度越高，其被选择的机会就越多。

- 个体选择概率分配方法
  - 适应度比例方法 (fitness proportional model): 个体被选择的概率和其适应度值成比例。个体i被选择的概率为： $p_{si} = \frac{f_i}{\sum_{i=1}^{M} f_i}$
  - 排序方法 (rank-based model): 计算每个个体的适应度后，根据适应度大小顺序对群体中的个体进行排序，然后把事先设计好的概率按排序分配给个体作为各自的选择概率。
    ①线性排序
    ②非线性排序

Genetic algorithm

- 选择 (selection process)
  - 选择个体的方法
    - 轮盘赌选择 (fitness proportional model): 按个体的选择概率产生一个轮盘，轮盘每个区的角度与个体的选择概率成比例。 产生一个随机数，它落入转盘的哪个区域就选择相应的个体交叉。
    - 锦标赛选择方法 (tournament selection model): 从群体中随机选择k个个体，将其中适应度最高的个体保存到下一代。这一过程反复执行，直到保存到下一代的个体数达到预先设定的数量为止。
      随机竞争方法 (stochastic tournament): 每次按赌轮选择方法选取一对个体，然后让这两个个体进行竞争，适应度高者获胜。如此反复，直到选满为止。
    - 最佳个体 (elitist model) 保存方法: 把群体中适应度最高的个体不进行交叉而直接复制到下一代中，保证遗传算法终止时得到的最后结果一定是历代出现过的最高适应度的个体

Genetic algorithm

- 突变(mutation) 操作和突变率

  遗传算法中的变异操作增加了算法的局部随机搜索能力，从而可以维持种群的多样性，为选择、交叉过程中可能丢失的某些遗传基因进行补充。
  突变率 (mutation rate)决定了后代对其表示的随机突变的频率。

  - 位点变异：群体中的个体码串，随机挑选一个或多个位点，并对这些位点的基因值以变异概率P_m作变动。考虑消除非法性。
  - 逆转变异：在个体码串中随机选择两点（逆转点），然后将两点之间的基因值以逆向排序插入到原位置中。
  - 插入变异：在个体码串中随机选择一个码，然后将此码插入随机选择的插入点中间。
  - 互换变异：随机选取染色体的两个基因进行简单互换。
  - 移动变异：随机选取一个基因，向左或者向右移动一个随机位数。

- 下一代种群的产生 (The makeup of the next generation)。这可能只是新形成的后代，也可能包括上一代中一些得分最高的父母。通常在选择过程一起处理。

Genetic algorithm

```
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))
```

Figure 4.8 A genetic algorithm. Within the function, *population* is an ordered list of individuals, *weights* is a list of corresponding fitness values for each individual, and *fitness* is a function to compute these values.

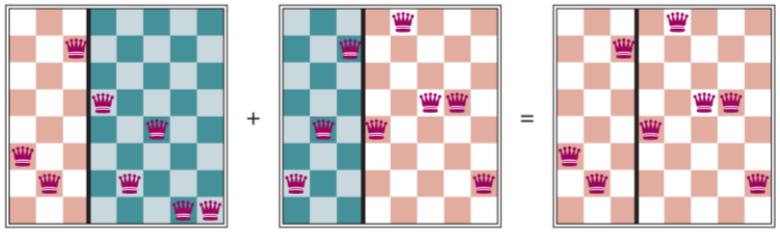


Figure 4.7 The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d). The green columns are lost in the crossover step and the red columns are retained. (To interpret the numbers in Figure 4.6: row 1 is the bottom row, and 8 is the top row.)

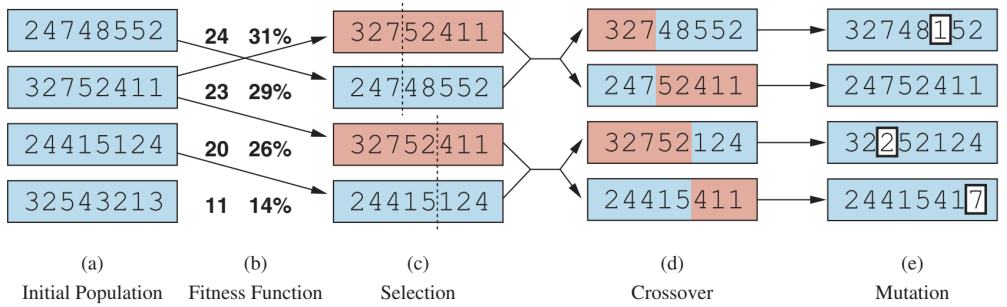


Figure 4.6 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic algorithm

如果两个父母串差异很大，那么交换 (crossover) 产生的状态和每个父母状态都相差很远。通常的情况是早期的种群是多样化的，因此交换 （类似于模拟退火）在搜索过程的早期阶段在状态空间中采用较大的步调，而在后来当大多数个体都很相似的时候采用较小的步调。

遗传算法结合了上山趋势、随机探索和在并行搜索线程之间交换信息。遗传算法最主要的优点来自于交换 (crossover) 操作。交换 (crossover) 的优势在于它能够将独立发展出来的能执行有用功能的字符区域结合起来，因此提高了搜索的粒度。

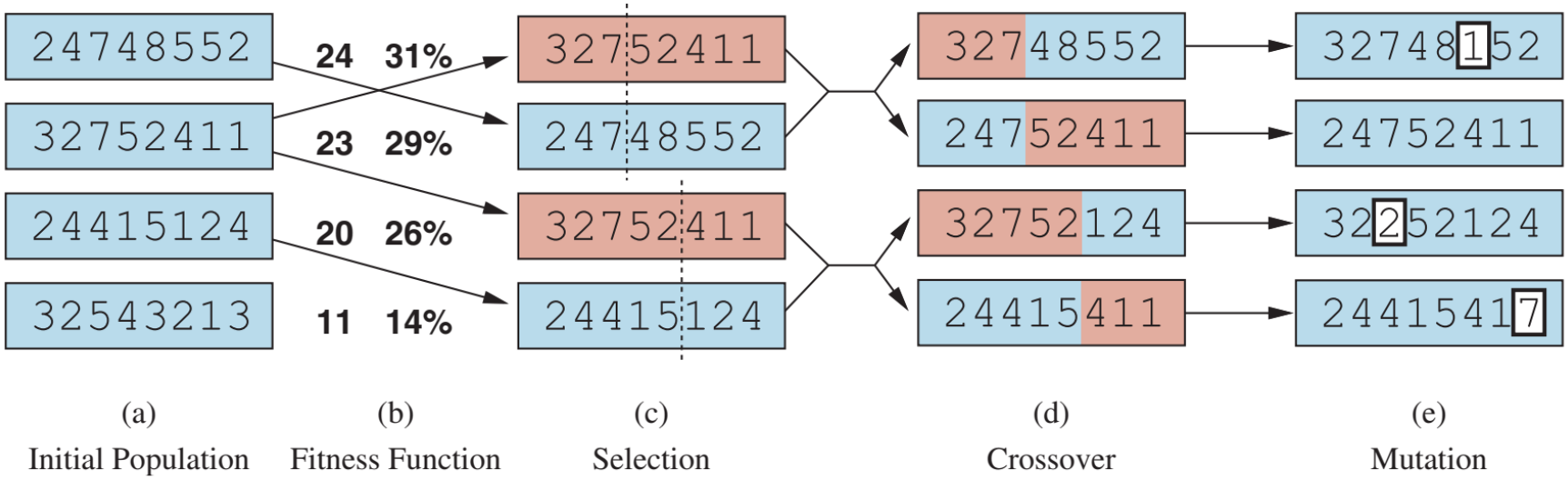


Figure 4.6 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic algorithm

遗传算法理论用模式 (schema) 思想来解释运转过程，模式是指其中某些位未确定的子串。例如，模式 246***** 描述了所有前三个皇后的位置分别是2,4,6的状态。能匹配这个模式的字符串 (例如24613578) 称作该模式的实例。可以证明，如果某模式的平均适应度超过均值， 那么种群内这个模式的实例数量就会随时间增长。

显然，如果模式实例邻近位互不相关，效果就没有那么显著，因为这样就会有很少的连续块可以提供一致的好处。 当模式 (schema) 对应于解决方案的有意义的组件时，遗传算法效果最好。

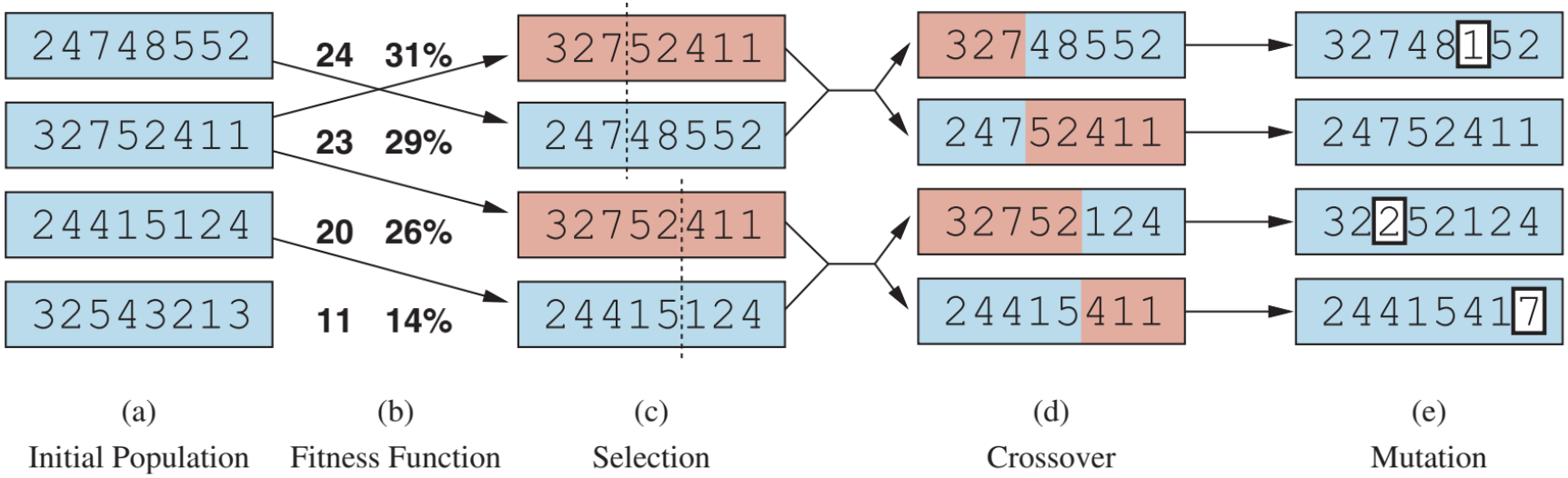


Figure 4.6 A genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by a fitness function in (b) resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

Genetic algorithm

例 已知函数 $y = f(x_1, x_2, x_3, x_4) = \frac{1}{x_1^2+x_2^2+x_3^2+x_4^2+8}$，其中 $-5 \leq x_1, x_2, x_3, x_4 \leq 5$，用遗传算法求解 y 的最大值及对应变量 x_1, x_2, x_3, x_4 的取值。

- ✓ Zhang, Yanbo, Benedikt Hartl, Hananel Hazan and Michael Levin. “Diffusion Models are Evolutionary Algorithms.” ArXiv abs/2410.02543 (2024): n. pag.

主要内容

5. 启发式函数 (Heuristic Functions)

6. Local Search and Optimization Problems

7. Evolutionary Algorithm

8. Search with Nondeterministic Actions or in Partially Observable Environments *

9. Online Search Agents and Unknown Environments *

Bibliography:

- ✓ Stuart J. Russel, Peter Norvig, “Artificial Intelligence: A Modern Approach” (4th Ed. 2020)”; 中译版“人工智能 现代方法”(4rd Ed., 2022), 人民邮电出版社. Ch 4.3, Ch 4.4

非确定动作的搜索

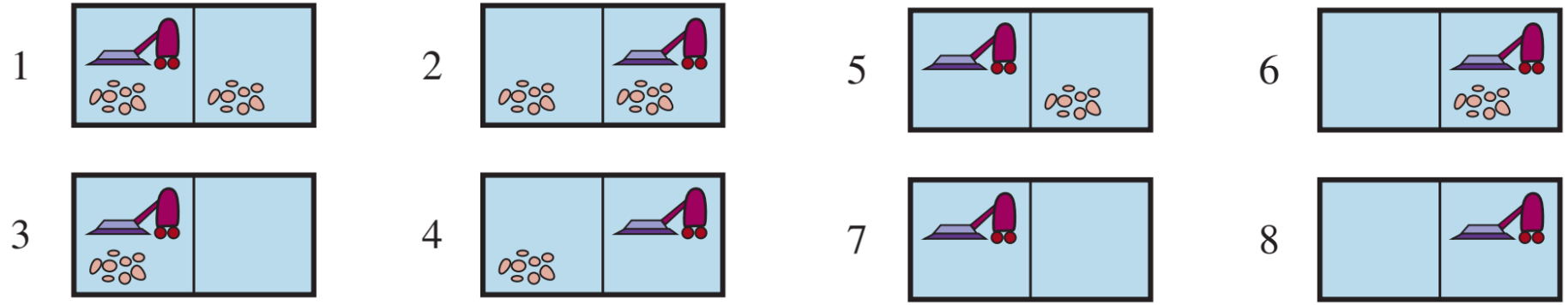
当环境是部分可观测 (partially observable) 的，Agent 不能确定它所处的状态，当环境是非确定的 (nondeterministic)，Agent 不知道在执行某个动作后，环境将转变到什么状态。Agent 认为它可能所处的物理状态集合称为信念状态 (belief state)。

在部分可观测环境和非确定性环境，问题的解不再是一个序列，而是一个条件规划 (conditional plan)，也称为 contingency plan 或 strategy。条件规划根据智能体在执行规划时接收到的感知决定行动。

不稳定吸尘器的世界

假设引入某种非确定形式，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 动作如下进行：

- 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。



为了更准确地形式化这个问题，需要讨论推广转移模型 (transition model) 概念。不用返回单个状态的RESULT函数定义转移模型，新的RESULT函数返回的是一组可能的状态。

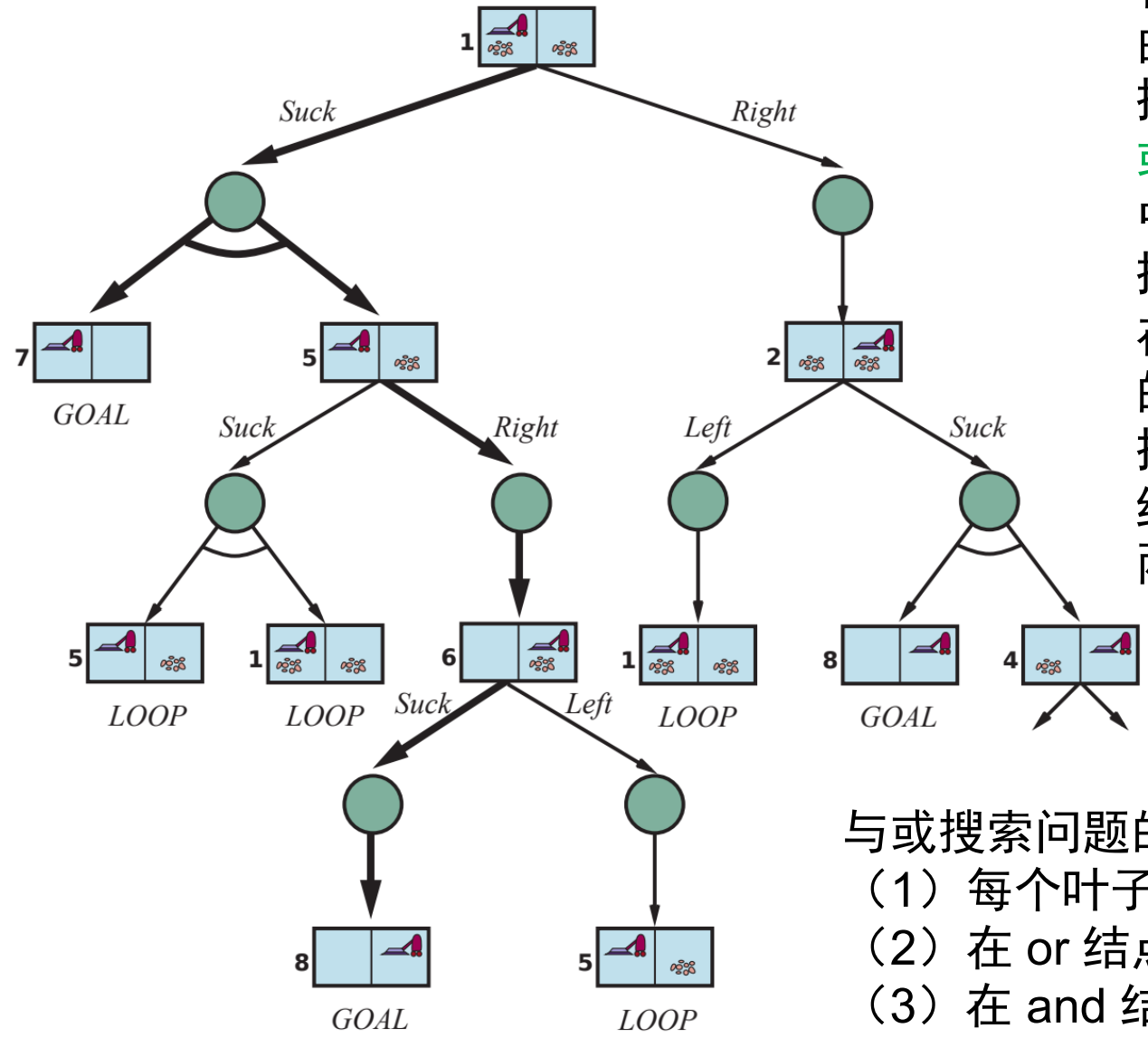
如果从状态1开始，没有一个序列可以求解问题。 转而，我们需要一个如下所示的条件规划：

Test state or percept

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

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。

AND-OR search trees



在确定性环境中，分支是由Agent在每个状态下的选择形成的。 称这些结点为或结点。 如在吸尘器世界中，Agent 在或结点上选择 Left 或 Right 或Suck。在非确定性环境中，分支的形成可能是由于环境选择每个行动的后果。这些结点则称为与结点。由这两种结点就构成了与或树。

与或搜索问题的解是一棵子树：

（1）每个叶子上都有目标结点，

（2）在 or 结点上指定一个行动，

（3）在 and 结点上包含所有可能后果。

Search in Partially Observable Environments

部分可观测问题中，agent 的感知 (percepts) 不足够弄清楚准确的环境。这意味着 agent 的一些行动的目标是降低当前状态的不确定性。

- 信念状态空间 (the belief-state space) 搜索

在信念状态空间中，问题是完全可观测的（fully observable）因为Agent总是知道自己的信念状态。

将底层的物理问题转换为信念状态问题，在信念状态而不是物理状态上进行搜索。原始问题P有$Actions_P$, $Result_P$, $Goal\_test_P$ 和 $Step\_cost_P$ 等组成部分。信念状态问题（the belief-state problem）有如下组成部分：

- 状态（States）：信念状态空间包含物理状态的每个可能的集和。如果P有N个状态，那么信念状态问题有2^N个状态，尽管有很多状态是不可达的。
- 初始状态（Initial state）：通常是P中所有状态的集合，尽管有些情况 agent 具有更多的知识。
- Actions：假设 agent 的信念状态$b = \{s_1, s_2\}$, $ACTIONS_P(s_1) \neq ACTIONS_P(s_2)$; agent 不确定哪个行动是合法的。如果假定非法行动对环境没有影响，那么在当前信念状态b中的任一物理状态所有行动的并集（union）是安全的：

$$ACTIONS(b) = \bigcup_{s \in b} ACTIONS_p(s)$$

另一方面，如果一个非法行动可能导致灾难，只允许交集（intersection）更安全。即所有状态中都合法的行动集合。

Search in Partially Observable Environments

- 信念状态空间 (the belief-state space) 搜索

- 转移模型（Transition model）：对于确定性（deterministic）行动，新的信念状态对于每个当前可能的状态有一个结果状态：

$$b' = RESULT(b,a) = \{s' : s' = RESULT_P(s,a) \wedge s \in b\}$$

考虑非确定性，新的信念状态包括当前信念状态中的任意状态应用行动后所有可能的结果：

$$b' = RESULT(b,a) = \{s' : s' = RESULT_P(s,a) \wedge s \in b\} = \bigcup_{s \in b} RESULT_P(s,a)$$

- 目标测试（Goal test）：如果信念状态中某一状态 s 满足底层问题的 $GOAL\_TEST_p(s)$，agent 可能达到目标。当每个状态满足 $GOAL\_TEST_p(s)$，agent 必然（necessarily）达到目标。
- 行动代价（Action cost）：这个同样棘手。如果相同的行动在不同状态下可能有不同的代价，那么在给定信念状态下采取这个行动会有几个值。目前我们假设在所有状态下相同的行动开销是相同的，因此可以从底层物理问题中直接转换。

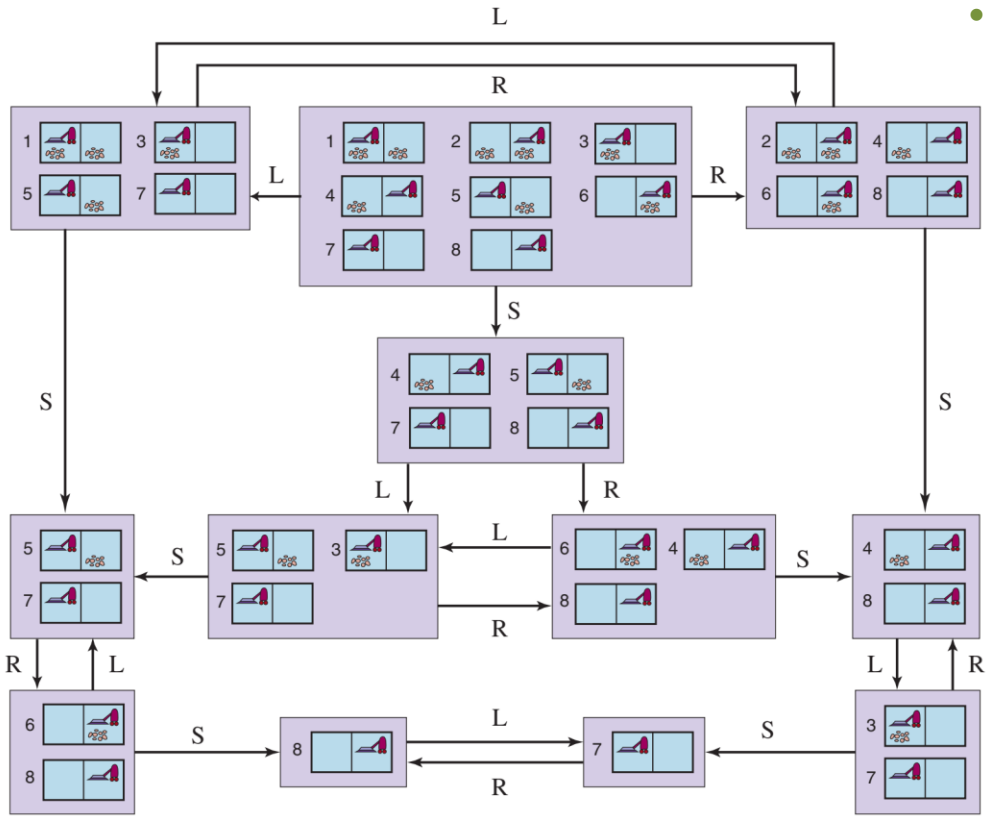
上述的定义确保可以从底层物理问题自动构建信念状态问题的形式化。一旦完成形式化，就可以应用搜索算法求解。

Search in Partially Observable Environments

在通常的图搜索中，要检测新生成的状态是否与已有状态相同。这也适用于信念状态。We can discard (prune) any such superset belief state.

信念空间非常巨大，如果物理空间的尺度为N，信念空间的尺度就为2^N。信念状态本身是集合，最多包括N个物理状态，存储空间开销大。

- 一种解决方法是用更紧凑的方式来表示信念状态。



- 另一种方法是不使用将信念状态和其它问题状态一样作为黑箱的标准搜索算法。观察信念状态内部，设计增量式信念状态搜索（incremental belief-state）算法，通过每次处理一个物理状态来求解。

Just as an AND node in AND–OR search, this algorithm has to find a solution for every state in the belief state; the difference is that AND–OR search can find a different solution for each branch, whereas an incremental belief-state search has to find one solution that works for all the states.

Search in Partially Observable Environments

The problem specification will specify a $\text{PERCEPT}(s)$ function that returns the percept received by the agent in a given state.

当只有部分观察信息的时候，通常是一些状态产生了相同的感知信息。给定初始的感知，可以得到初始的信念状态。信念状态之间的转移模型包括3个阶段：

- 预测阶段：计算行动产生的信念状态 $\hat{b} = PREDICT(b,a) = RESULT(b,a)$.
- 可能感知阶段：计算预测得到信念状态下可能观测到的感知集合

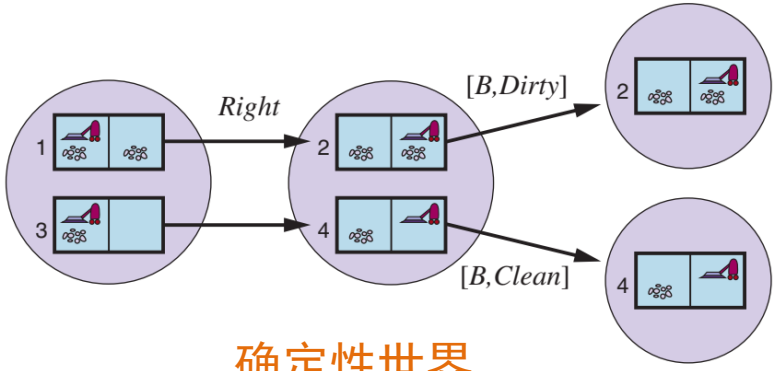
$$POSSIBLE\_PERCEPTS\left(\hat{b}\right) = \{o: o = PERCEPT(s) \wedge s \in \hat{b}\}$$

- 更新阶段：为每个可能感知计算其可能的信念状态。更新后的信念状态b_0是可能产生感知的包含在预测得到的信念状态$\hat{b}$中状态的集合

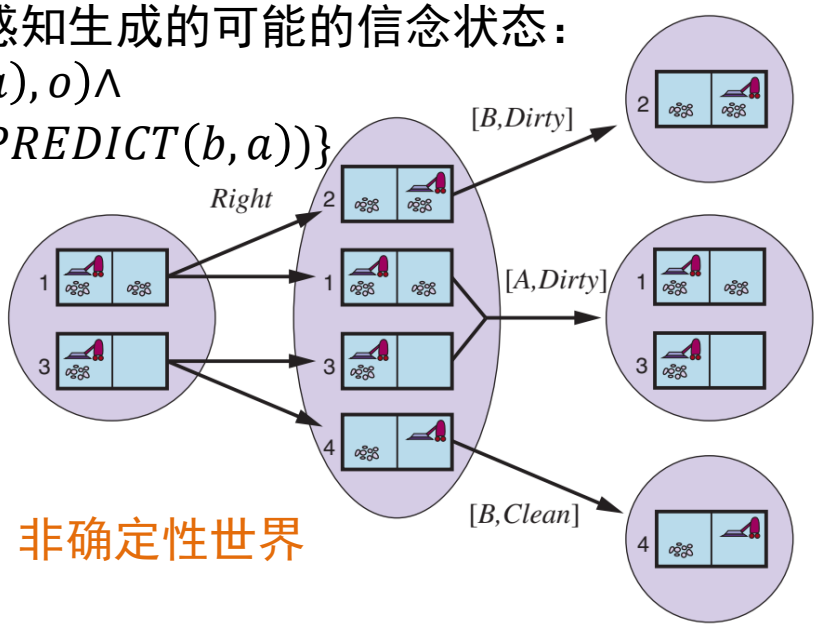
$$b_o = UPDATE\left(\hat{b}, o\right) = \{s: o = PERCEPT(s) \wedge s \in \hat{b}\}$$

综合考虑这3个阶段，一个给定行动和后续可能的感知生成的可能的信念状态：

$$RESULTS(b,a) = \{b_o: b_o = UPDATE(PREDICT(b,a), o) \wedge o \in POSSIBLE\_PERCEPTS(PREDICT(b,a))\}$$



确定性世界

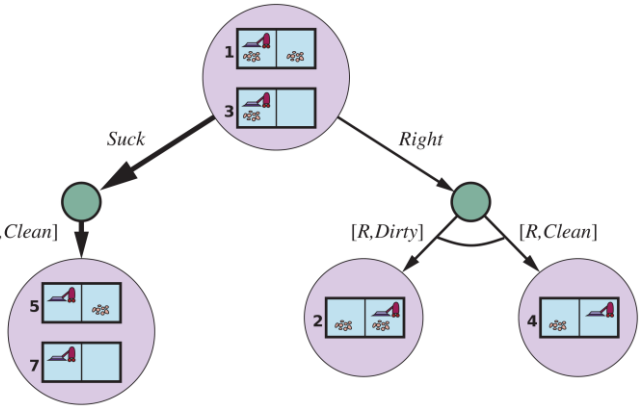


非确定性世界

Search in Partially Observable Environments

部分可观测环境中的Agent先对问题进行形式化，调用搜索算法（如 AND-OR-Search）求解，然后执行解步骤。

Because a belief-state problem is supplied to the AND–OR search algorithm, it returned a conditional plan that tests the belief state rather than the actual state. One can improve on this by checking for previously generated belief states that are subsets or supersets of the current state. One can also derive incremental search Algorithms.

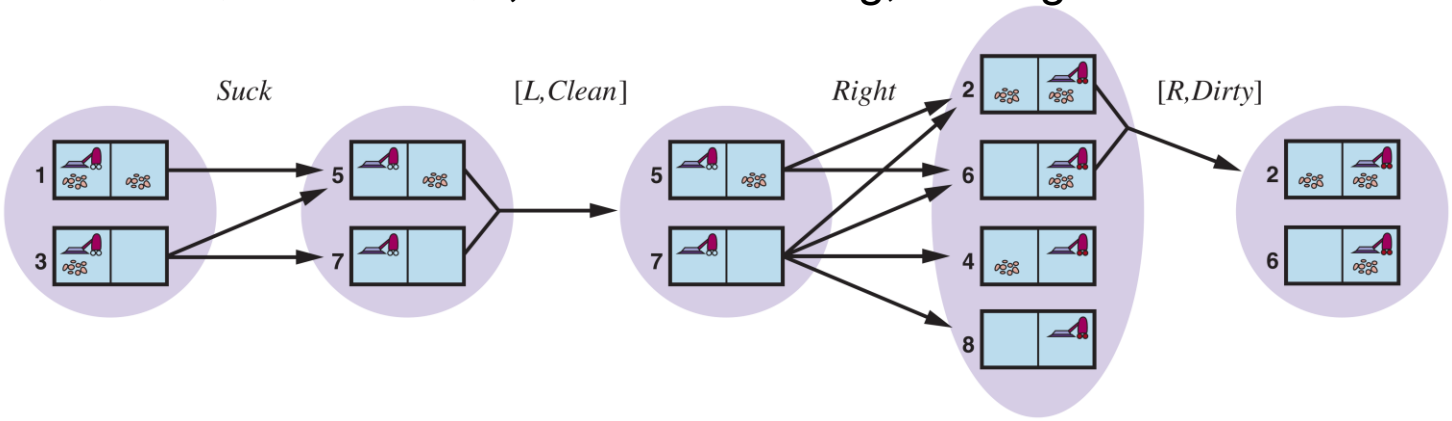


(1) 问题的解将是条件规划而不是一个序列。

(2) Agent 需要在完成行动和接收感知信息时维护自身的信念状态。

与信念状态转移模型的 prediction–observation–update 过程相比，感知信息由环境给出而不是 Agent 自己计算: $b' = UPDATE(PREDICT(b, a), o)$

在部分可观察环境中，维护自身的信念状态 belief state 成为任何智能系统的核心函数。这个函数可能有多个不同名字，包括 monitoring, filtering 和 state estimation。

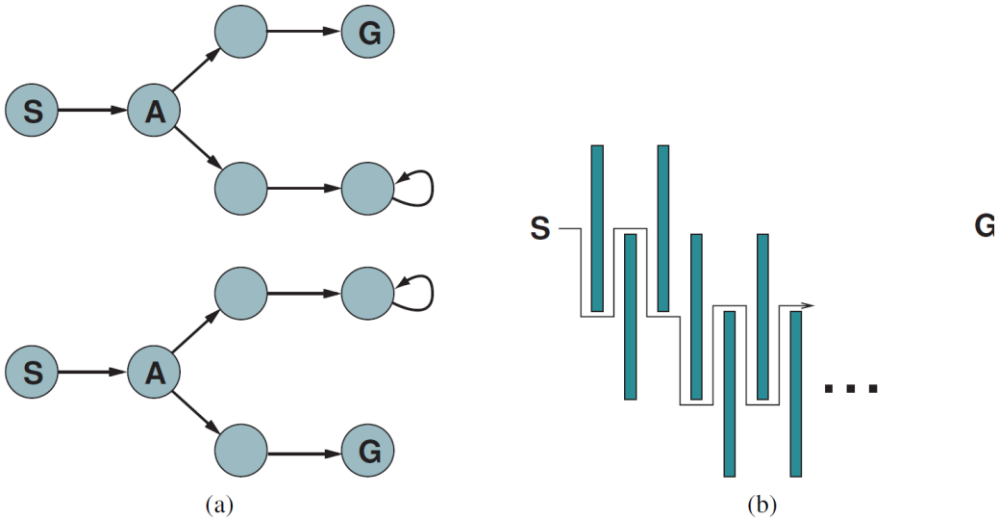


Online Search Agents and Unknown Environments

An online search problem is solved by interleaving computation, sensing, and acting.

- ACTIONS(s), the legal actions in state s;
- $c(s, a, s')$, the cost of applying action a in state s to arrive at state s'. Note that this cannot be used until the agent knows that s' is the outcome.
- IS-GOAL(s), the goal test.

Typically, the agent's objective is to reach a goal state while minimizing cost. It is common to compare this cost with the path cost the agent would incur if it knew the search space in advance—that is, the optimal path in the known environment. In the language of online algorithms, this comparison is called the competitive ratio.



Dead ends are a real difficulty for robot exploration—staircases, ramps, cliffs, one-way streets, and even natural terrain all present states from which some actions are irreversible—there is no way to return to the previous state.

Online Search Agents and Unknown Environments

function ONLINE-DFS-AGENT(*problem*, s') **returns** an action
  s, a, the previous state and action, initially null
  result, a table mapping $(s,\ a)$ to s', initially empty
  untried, a table mapping s to a list of untried actions
  unbacktracked, a table mapping s to a list of states never backtracked to

  if *problem*.IS-GOAL(s') **then return** *stop*
  if s' is a new state (not in *untried*) **then** *untried*[s'] $\leftarrow$ *problem*.ACTIONS(s')
  if s is not null **then**
    result[s, a] $\leftarrow s'$
    add s to the front of *unbacktracked*[s']
  if *untried*[s'] is empty **then**
    if *unbacktracked*[s'] is empty **then return** *stop*
    $a \leftarrow$ an action b such that *result*[s', b] = POP(*unbacktracked*[s'])$s' \leftarrow$ null
  else $a \leftarrow$ POP(*untried*[s'])
  $s \leftarrow s'$
  return a