Artificial Intelligence

4.超越经典搜索

引入

- 上一章，我们讨论了一个单一类别的问题，其解决方案是具有如下特点的一系列动作：
  - 可观测
  - 确定性
  - 已知环境
- 关注解状态而不是路径代价，局部搜索4.1-4.2：
  - 模拟退火、遗传算法
- 部分可观察性4.3-4.4：
  - Agent需要跟踪可能的状态
- 完全未知的空间4.5
  - Agent需要从头开始搜索

4.1 局部搜索算法和最优化问题

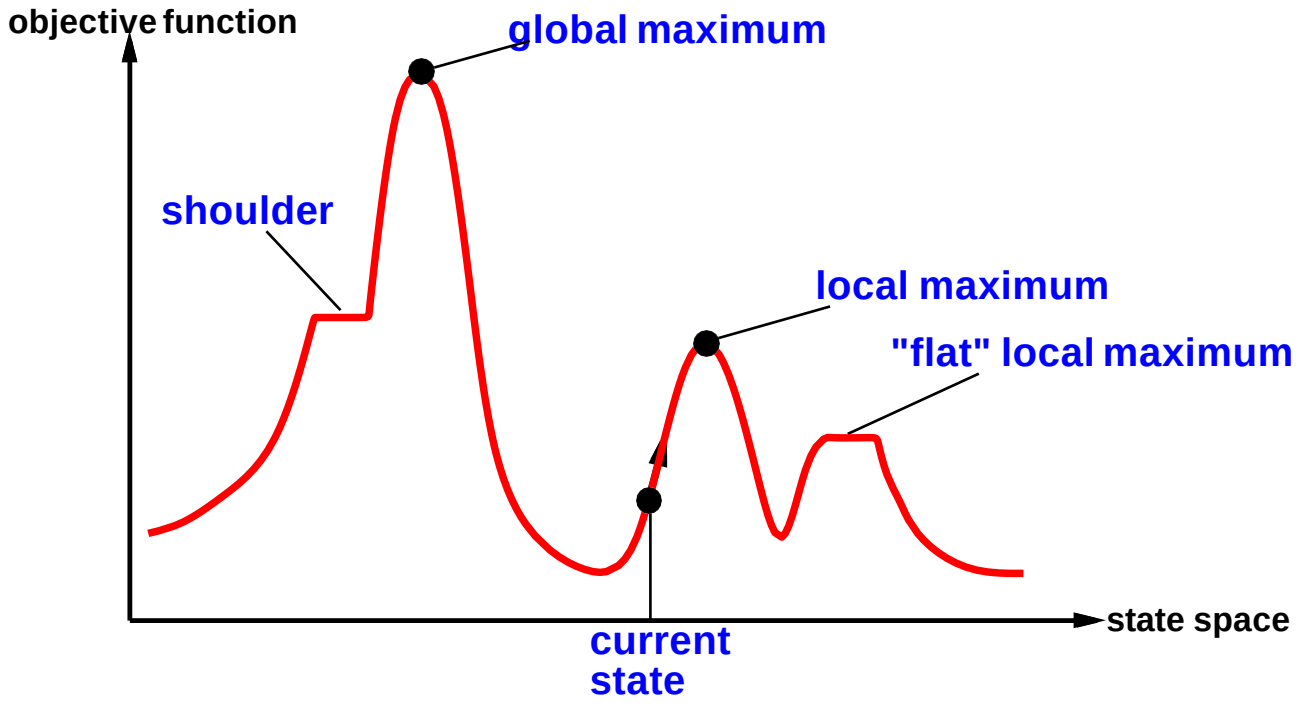
我们前面介绍过的搜索算法都系统地探索空间。这种系统化通过在内存中保留一条或多条路径和记录路径中的每个结点的选择。当找到目标时，到达此目标的路径就是这个问题的一个解。然而在许多问题中，到达目标的路径是不相关的。例如，在八皇后问题中（参见 3.2.1 节），重要的是最终皇后在棋盘上的布局，而不是皇后加入的先后次序。许多重要的应用都具有这样的性质，例如集成电路设计、工厂场地布局、作业车间调度、自动程序设计，电信网络优化、车辆寻径和文件夹管理。

如果到目标的路径是无关紧要的，我们可能考虑不同的算法，这类算法不关心路径。**局部搜索**算法从单个**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。虽然局部搜索算法不是系统化的，但是有两个关键的优点：（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。

除了找到目标，局部搜索算法对于解决纯粹的**最优化问题**十分有用，其目标是根据**目标函数**找到最佳状态。第 3 章中介绍的“标准的”搜索模型并不适用于很多最优化问题。例如，自然界提供了一个目标函数——繁殖适应性——达尔文的进化论可以被视为最优化的尝试，但是这个问题本身没有“目标测试”和“路径代价”。

4.1 局部搜索算法和最优化问题

为了理解局部搜索，我们借助于**状态空间地形图**（如图 4.1 所示）。地形图既有“坐标”（用状态定义）又有“标高”（由启发式式代价函数或目标函数定义）。如果标高对应于代价，那么目标就是找到最低谷——即**全局最小值**；如果标高对应于目标函数，那么目标就是找到最高峰——即**全局最大值**（可以通过插入一个负号使两者相互转换）。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。



局部搜索的应用

- 集成电路设计
- 工厂车间布局
- 车间作业调度
- 自动规划
- 通讯
- 网络优化
- 车辆路由
- 投资组合管理

4.1.1 爬山法

- 一种属于局部搜索家族的数学优化方法
- 一种迭代算法：
  - 开始时选择问题的一个任意解，然后递增地修改该解的一个元素，若得到一个更好的解，则将该修改作为新的解；
  - 重复直到无法找到进一步的改善。
- 大多数基本的局部搜索算法都不保持一棵搜索树。
- 爬山搜索算法是最基本的局部搜索方法。
- 它常常会朝着一个解快速地进展，因为通常很容易改善一个不良状态。
- 它往往被称为贪婪局部搜索，因为它只顾抓住一个好的邻接点的状态，而不提前思考下一步该 去哪儿。

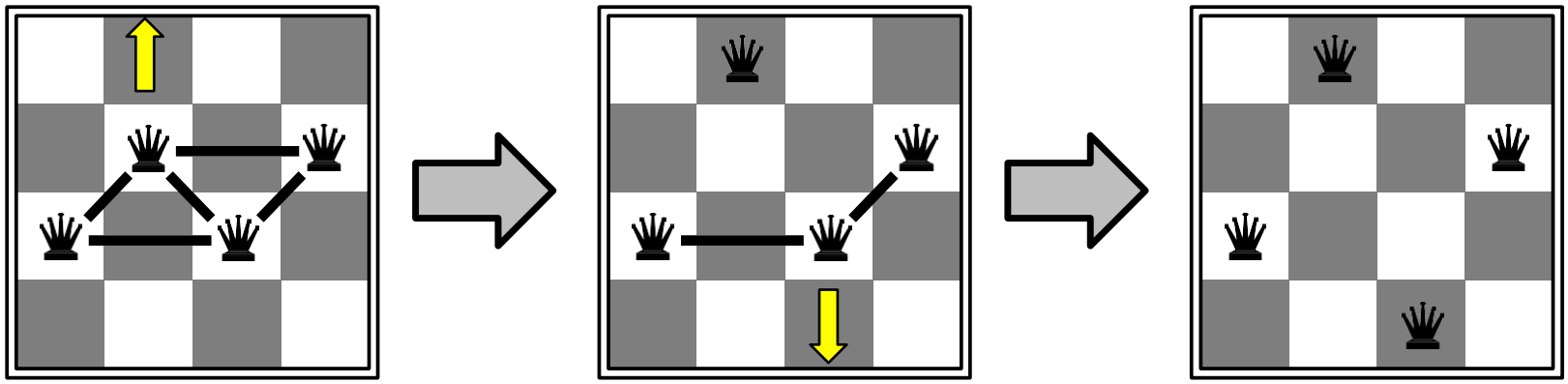
爬山搜索算法

```
function Hill-Climbing( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node

   current ← Make-Node(Initial-State[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if Value[neighbor] ≤ Value[current] then return State[current]
      current ← neighbor
   end
```

例子：n皇后问题

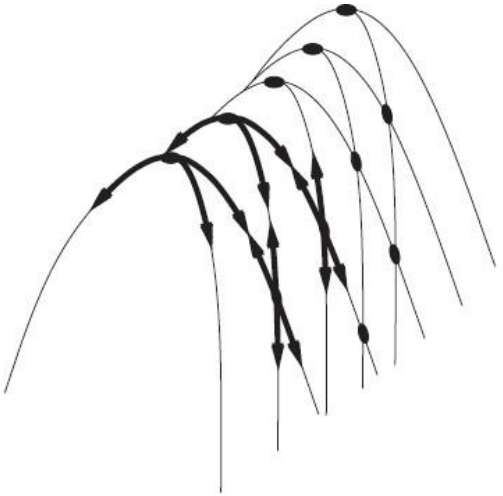
• 把n个皇后放在*n* x *n*的棋盘上。每次移动一个皇后来减少冲突数量，使得没有两个皇后在同一行、 同一列、或同一对角线上。

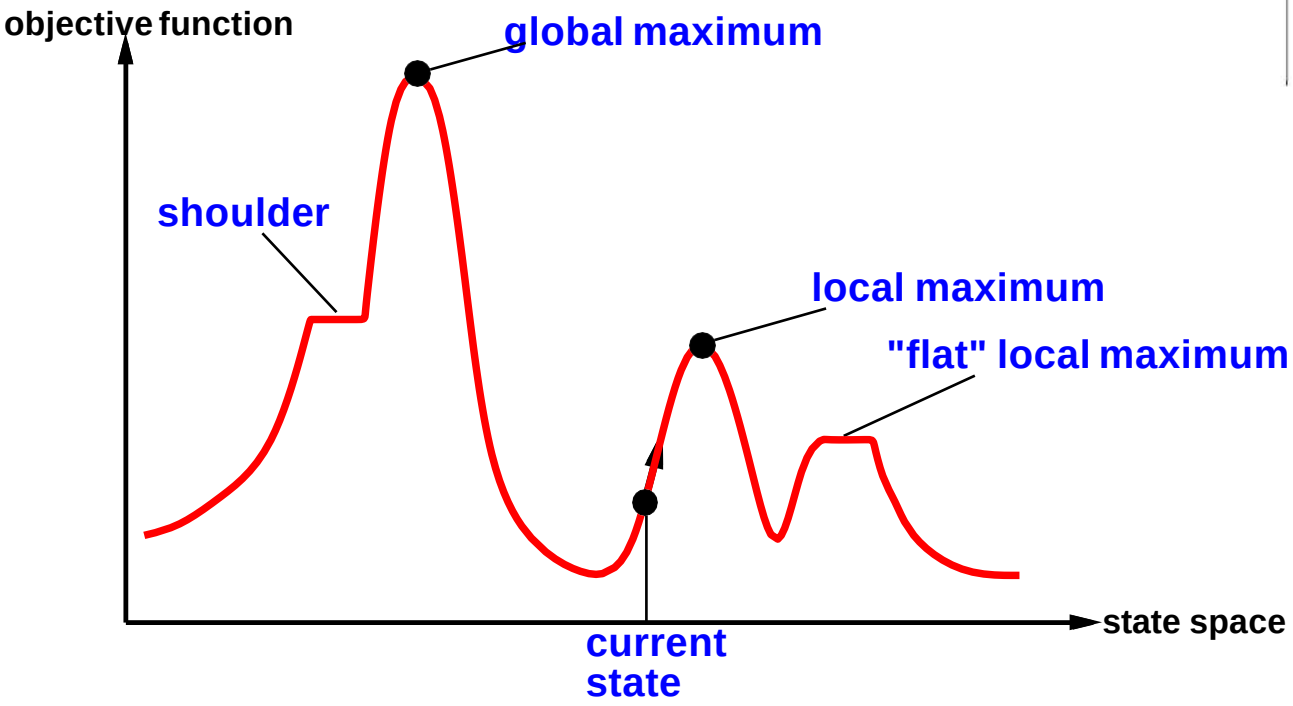


h = 5　　h = 2　　h = 0

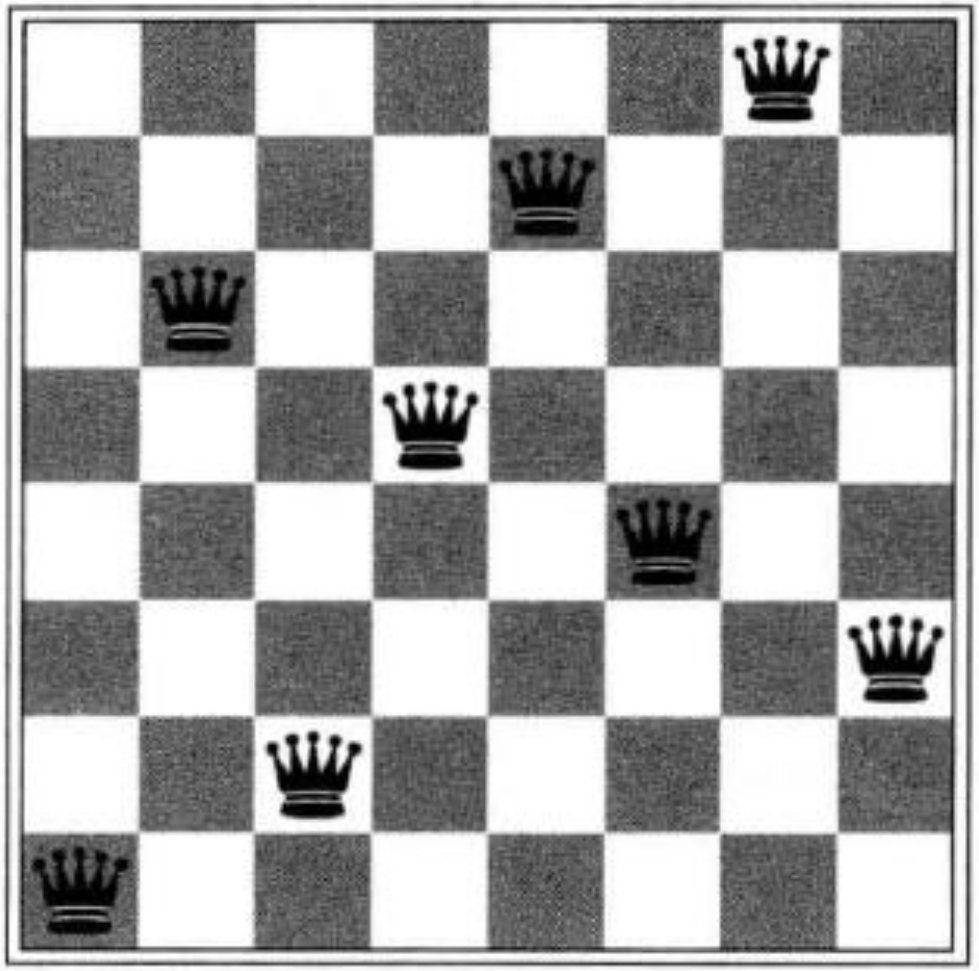
爬山法的弱点

- 它在如下三种情况下经常被困：
  - 局部最大值
  - 山脊：一系列局部最大值
  - 高原：一块评的局部最大值





八皇后问题的局部最大值



随机爬山法和随机重启爬山法

爬山法有许多变形。**随机爬山法**在上山移动中随机地选择下一步；被选中的概率可能随着上山移动的陡峭程度不同而不同。这种算法通常比最陡上升算法的收敛速度慢不少，但是在某些状态空间地形图上它能找到更好的解。**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优于当前结点的后继。这个算法在后继结点很多的时候（例如上千个）是个好策略。

到现在为止我们描述的爬山法是不完备的——它们经常会在目标存在的情况下因为被局部极大值卡住而找不到目标。**随机重启爬山法**（random restart hill climbing）吸纳了这种思想："如果一开始没有成功，那么尝试，再尝试（重新开始搜索）。"它通过随机生成初始状态[1]来导引爬山法搜索，直到找到目标。这个算法完备的概率接近于 1，理由是它最终会生成一个目标状态作为初始状态。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 $1/p$。对于不允许侧向移动的八皇后问题实例，$p \approx 0.14$，因此大概需要 7 次迭代找到目标（6 次失败 1 次成功）。所需步数为一次成功迭代的搜索步数加上失败的搜索步数与$(1-p)/p$ 的乘积，大约是 22 步。允许侧向移动时，平均需要迭代约 $1/0.94 \approx 1.06$ 次，平均的步数为$(1\times 21)+(0.06/0.94)\times 64 \approx 25$ 步。对于八皇后问题，随机重启爬山法实际上是有效的。即使有 300 万个皇后，这个方法找到解的时间不超过 1 分钟。[2]

4.1.2 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（或代价高的）方向搜索，它肯定是不完备的，理由是可能卡在局部极大值上。与之相反，纯粹的随机行走——就是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。因此，把爬山法和随机行走以某种方式结合，同时得到效率和完备性的想法是合理的。**模拟退火**就是这样的算法。在冶金中，**退火**是用于增强金属和玻璃的韧性或硬度而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。为了更好地理解模拟退火，我们把注意力从爬山法转向**梯度下降**（即，减小代价），想象在高低不平的平面上有个乒乓球想掉到最深的裂缝中。如果只允许乒乓球滚动，那么它会停留在局部极小点。如果晃动平面，我们可以使乒乓球弹出局部极小点。窍门是晃动幅度要足够大让乒乓球能从局部极小点弹出来，但又不能太大把它从全局最小点弹出来。模拟退火的解决方法就是开始使劲摇晃（也就是先高温加热）然后慢慢降低摇晃的强度（也就是逐渐降温）。

模拟退火

- 核心思想：通过允许一些“坏”的移动来避免局部最大值，但逐渐减少他们的规模和频率

- 初始解:使用启发式方法生成。随机选择。
- 相邻节点：随机生成。当前解的变异。
- 接受条件：相邻节点具有较低代价值，具有较高代价值的相邻节点则以概率P接受。
- 停止判据：有比阈值低的值。已达到迭代最大总次数。

模拟退火算法

```
function Simulated-Annealing(problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps
    current ← Make-Node(Initial-State[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current
        next ← a randomly selected successor of current
        ΔE ← Value[next] − Value[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(ΔE/T)
```

模拟退火算法的内层循环（图 4.5）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——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。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。

4.1.3 局部束搜索

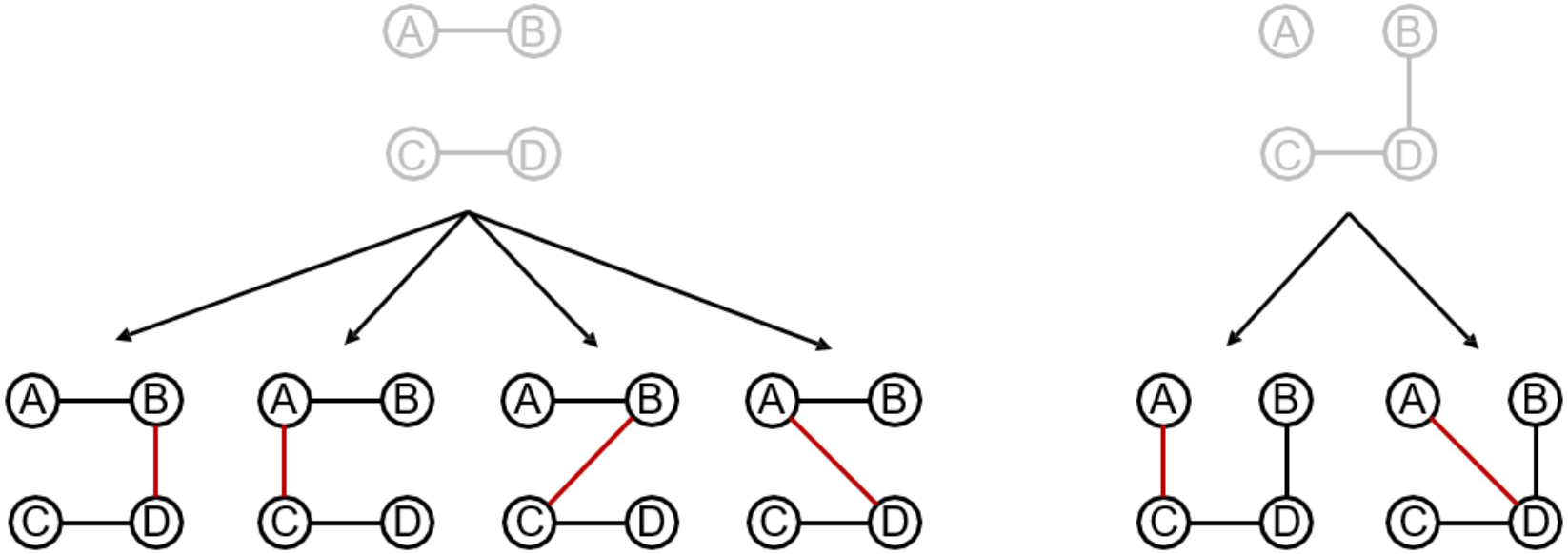
内存总是有限的，但在内存中只保存一个结点又有些极端。**局部束搜索**（local beam search）算法[1]记录 k 个状态而不是只记录一个。它从 k 个随机生成的状态开始。每一步全部 k 个状态的所有后继状态全部被生成。如果其中有一个是目标状态，则算法停止。否则，它从整个后继列表中选择 k 个最佳的后继，重复这个过程。

k 个状态的局部束搜索给人的第一印象是，并行而不是串行地运行 k 个随机重启搜索。实际上，这两个算法有很大不同。在随机重启搜索中，每个搜索的运行过程是独立的。而在局部束搜索中，有用的信息在并行的搜索线程之间传递。实际上，产生最好后继的状态会通知其他状态说："过来，这儿的草更绿！"算法很快放弃没有成果的搜索而把资源都用在取得最大进展的路径上。

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，使得搜索代价比高昂的爬山法版本还要多。**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，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。随机束搜索类似于自然选择，"状态"（生物体）根据"值"（适应度）产生它的"后继"（后代子孙）。

例子：旅行推销员问题

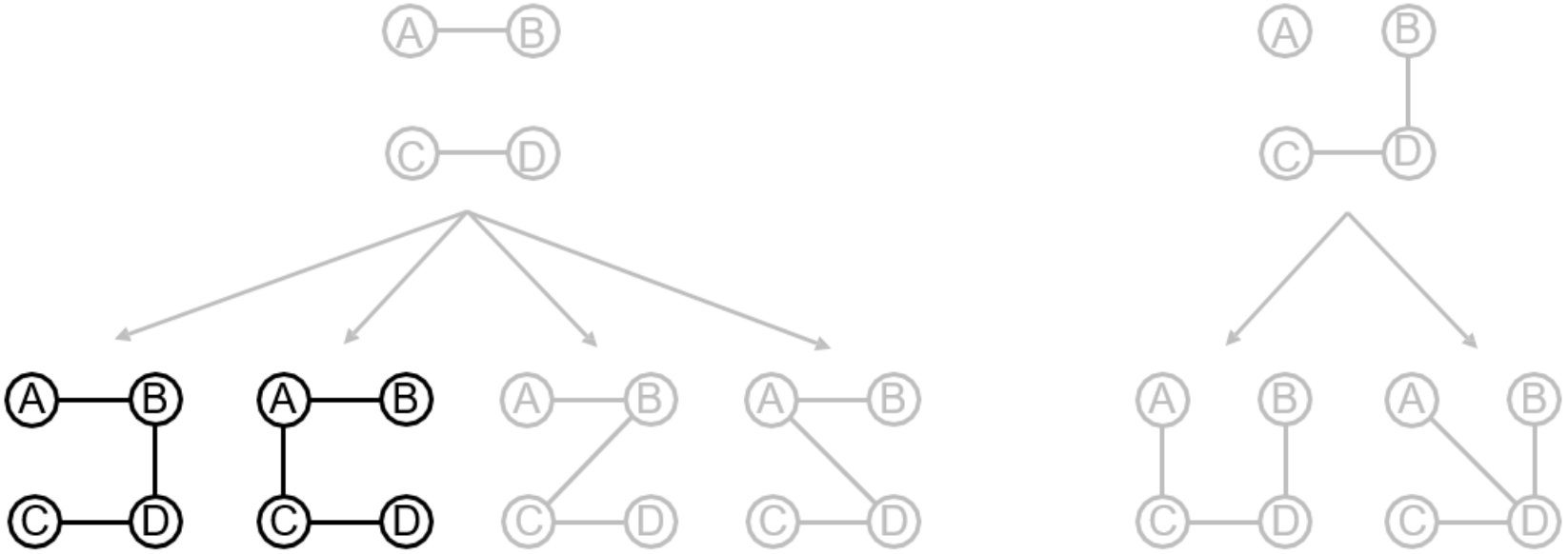
• 保持k个状态而不仅仅为1。从k个随机生成的状态开始。本例中k=2。



• 生成所有k个状态的全部后继节点。这些后继节点中没有目标状态，故继续下一步。

例子：旅行推销员问题

- 从完成表中选择最佳k个后继节点。重复上述过程，直到找到目标。



4.1.4 遗传算法

遗传算法（genetic algorithm，或 GA）是随机束搜索的一个变形，它通过把两个父状态结合来生成后继，而不是通过修改单一状态进行。这和随机剪枝搜索一样，与自然选择类似，除了我们现在处理的是有性繁殖而不是无性繁殖。

像束搜索一样，遗传算法也是从 k 个随机生成的状态开始，我们称之为**种群**。每个状态，或称**个体**，用一个有限长度的字符串表示——通常是 0、1 串。例如，八皇后问题的状态必须指明 8 个皇后的位置，每列有 8 个方格，所以需要 $8\times\log_2 8 = 24$ 比特来表示。换句话说，每个状态可以由 8 个数字表示，数字范围都是从 1 到 8（后面我们会看到这两种不同的编码形式表现是有差异的）。图 4.6（a）显示了 4 个表示 8 皇后状态的 8 位数字串组成的种群。

遗传算法

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*

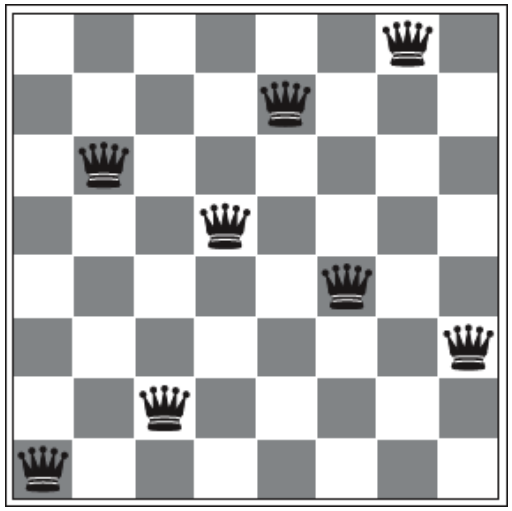
population ← *new_population*

until some individual is fit enough, or enough time has elapsed

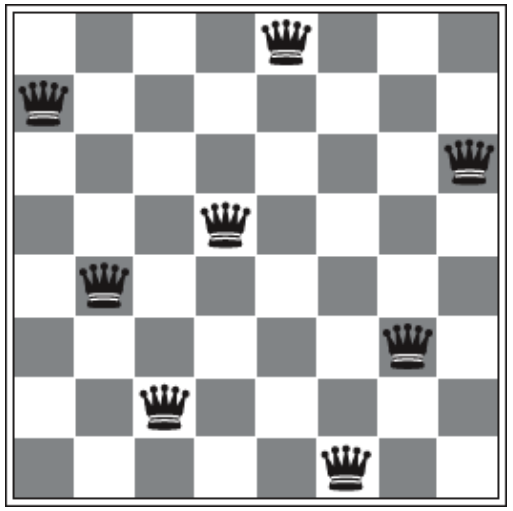
return the best individual in *population*, according to FITNESS-FN

例子：8皇后问题

- 某8皇后状态需要指明8个皇后的位置，每个位于8个方格的一列，其状态可用8个数字表示，每个位于1到8之间。

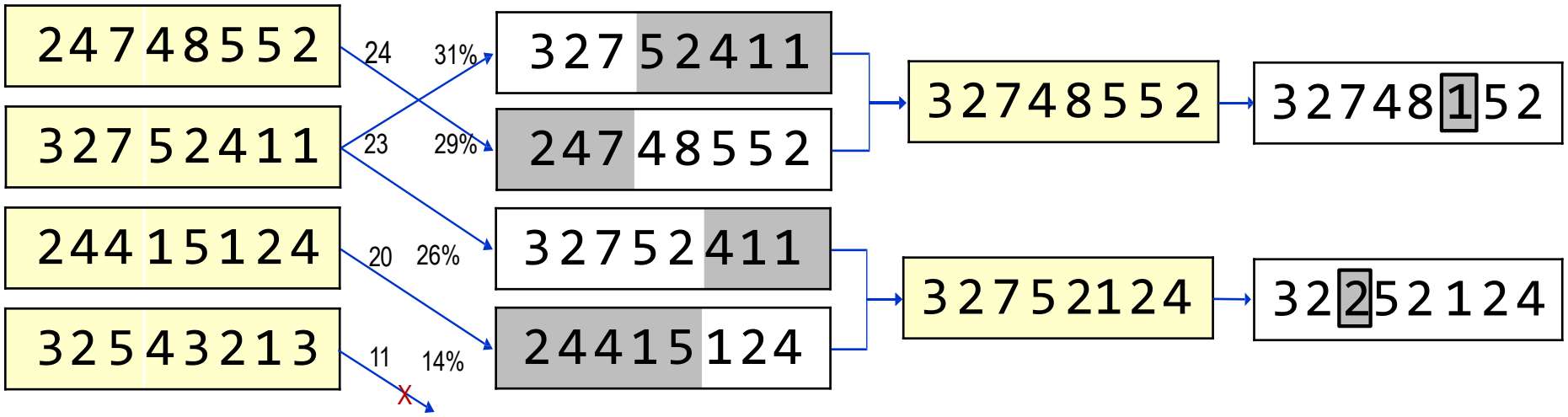


16257483



74258136

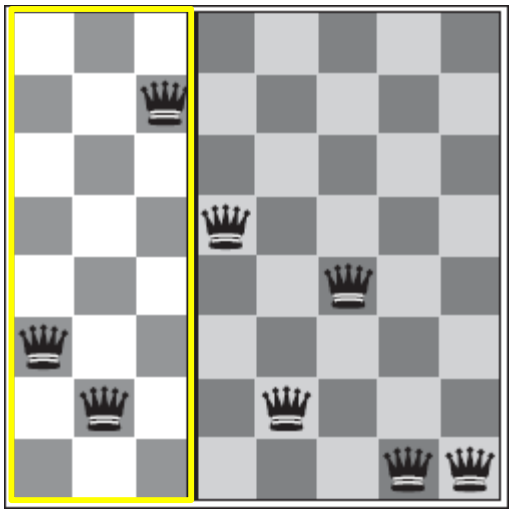
例子：8皇后问题



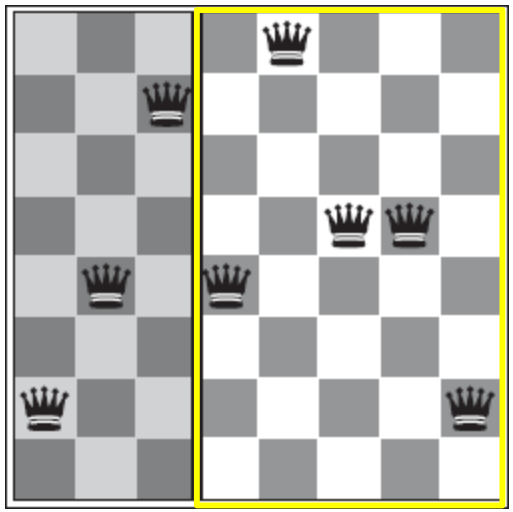
(a) Initial Population 初始种群　(b) Fitness 适应　(c) Selection 选择　(d) Crossover 杂交　(e) Mutation 变异

数字串表示8皇后的状态。(a)为初始种群，(b)通过适应函数进行分级，(c)导致交配对产生，(d)繁殖后代，(e)取决于突变。

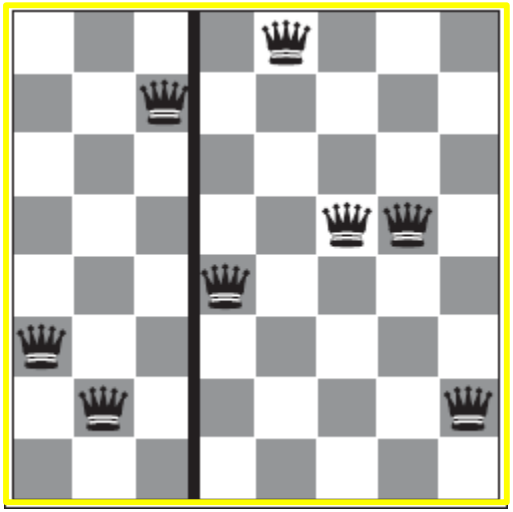
例子：8皇后问题



+



=



32752411

24748552

32748552

- 这三个8皇后的状态分别对应于“选择”中的两个父辈和“杂交”中它们的后代。
- 阴影的若干列在杂交步骤中被丢掉，而无阴影的若干列则被保留下来。

4.2 连续空间中的局部搜索

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：

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

这个表达式显然是局部的不是全局的，原因在于 C_i 集合是状态的非连续函数。

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。

梯度搜索

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：

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

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——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。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决定所谓的**经验梯度**。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。

凸优化

局部搜索算法在连续状态空间和离散状态空间一样受到局部极大值、山脊和高原的影响。可以使用随机重启和模拟退火算法，会比较有效。然而，高维的状态空间太大了，很容易使算法迷路找不到解。

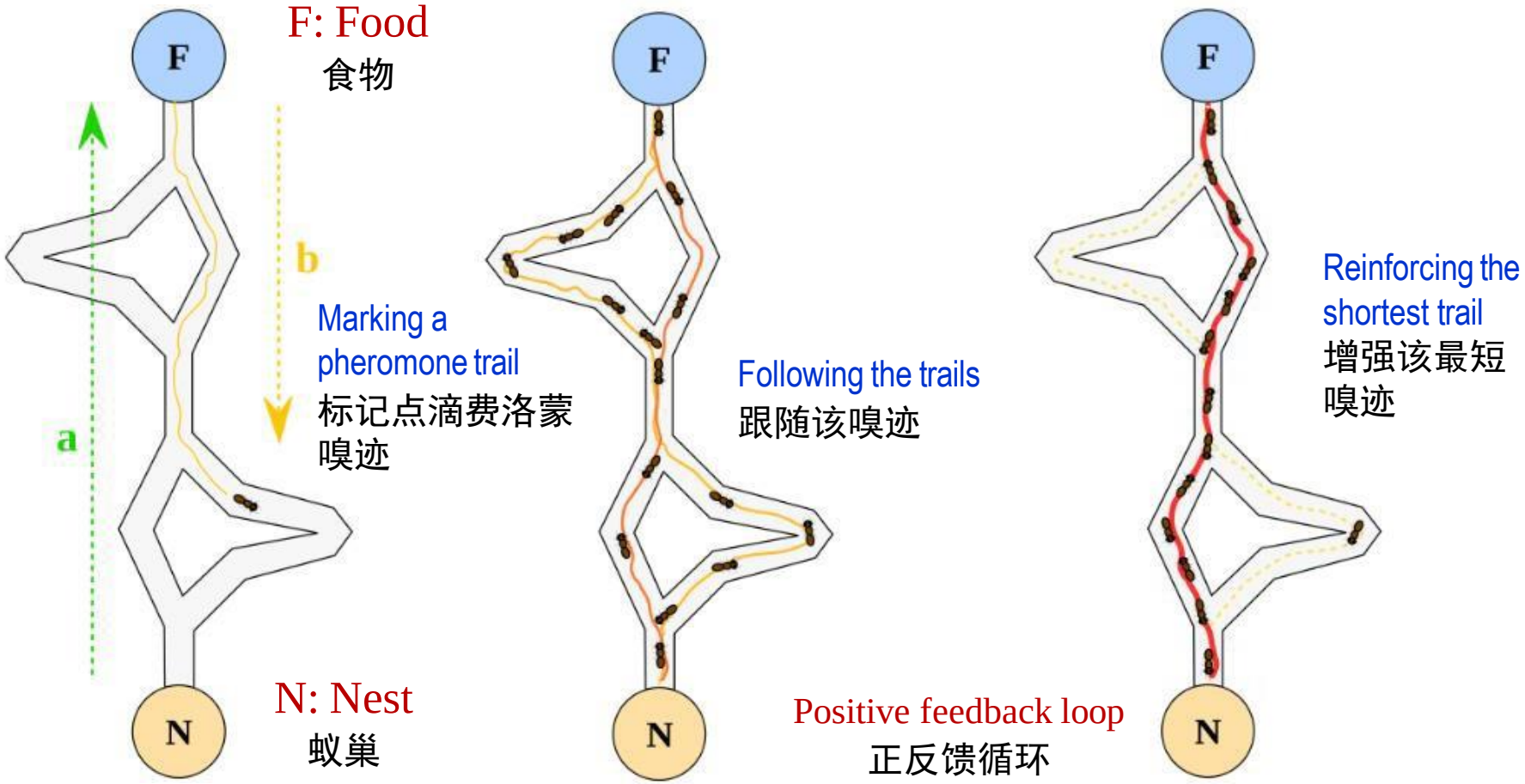
最后一个值得讨论的话题是**约束优化**。如果问题的解必须满足变量的某些严格约束，称此优化问题是受约束的。例如，在机场选址问题中，可以限定机场的位置在罗马尼亚境内并且是在陆地上（而不是在某个湖的中央）。约束最优化问题的难点取决于约束和目标函数的性质。最著名的一类问题是**线性规划**问题，其约束是线性不等式并且能够组成一个**凸多边形**[1]区域，目标函数也是线性的。线性规划问题的时间复杂度是关于变量数目的多项式时间函数。

线性规划被广泛研究，也是最有用的一类优化问题。它是凸优化问题的特殊情况，允许约束区域可以是任一凸区，目标可以是凸区的任何凸函数。在某些情况下，凸优化问题在多项式时间内是可解的，即使有上千个变量也是实际可行的。机器学习和控制论中的一些重要问题可以形式化成凸优化问题（详见第 20 章）。

4.3 蚁群优化

- 它是一种解决计算问题的概率技术，可以用于发现一个图上的最佳路径。最初是由Marco Dorigo于1992年在他的博士论文中提出的。
- 该算法是受蚂蚁在蚁巢和食物源之间寻找路径行为的启发而形成的。
- 蚂蚁从蚁巢到食物源之间盲目地游荡：
  - 最短路径是通过费洛蒙嗅迹发现的
  - 每个蚂蚁随机地移动
  - 费洛蒙就遗留在路径上
  - 蚂蚁察觉到前面蚂蚁的路径，跟随而去
  - 路径上更多的费洛蒙增加了跟随该路径的概率

蚁群优化的概念

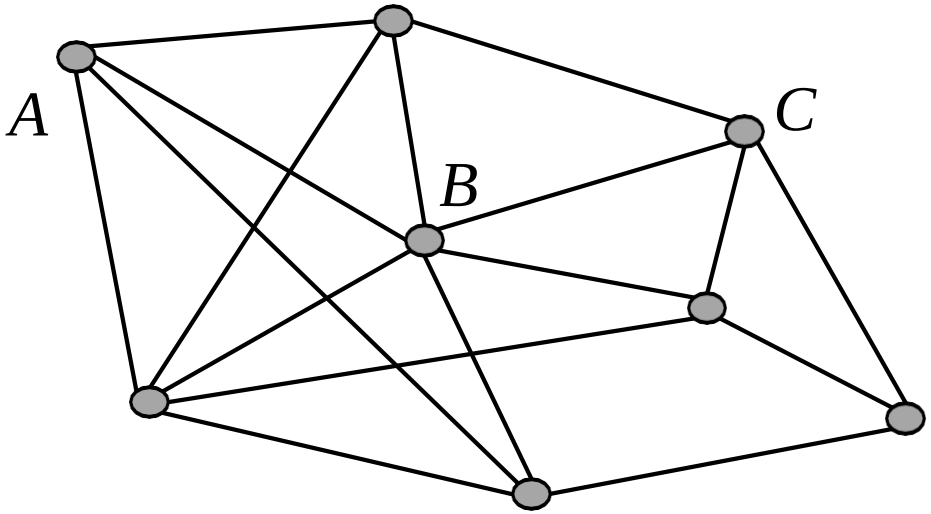

F
N
a
b
F: Food
食物
Marking a pheromone trail
标记点滴费洛蒙嗅迹
N: Nest
蚁巢
Following the trails
跟随该嗅迹
Positive feedback loop
正反馈循环
Reinforcing the shortest trail
增强该最短嗅迹

蚁群优化算法

- 在路径段上积累“虚拟”嗅迹
- 开始时随机选择某个节点
- 随机选择一条路径：
  - 基于从初始节点至合适路径上出现嗅迹的量；
  - 具有较多嗅迹的路径则具有较高的概率
- 蚂蚁到达下一个节点后，再选择下一个路径
- 重复直到更多的蚂蚁在每个循环中都选择同一个路径

例：旅行推销员问题TSP

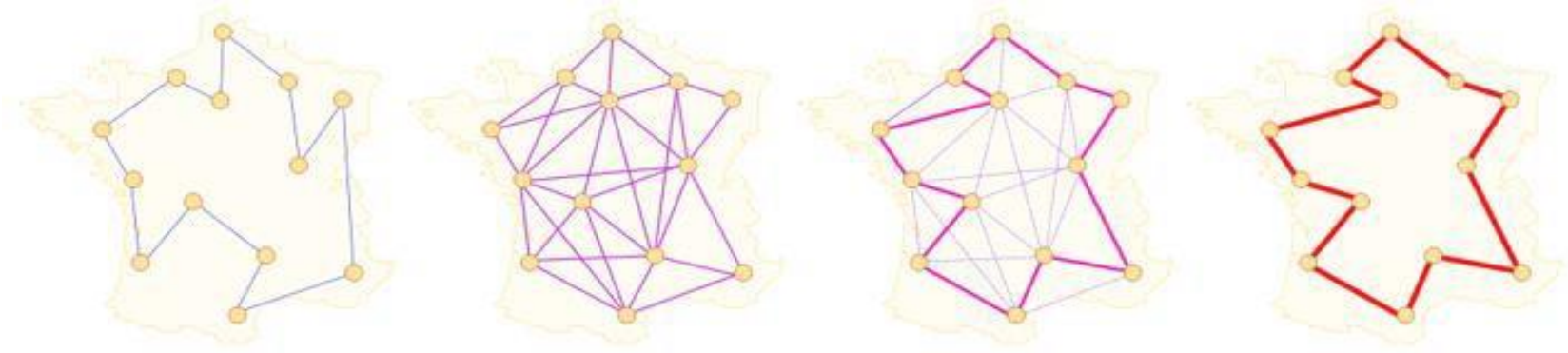
- 一个推销员花时间访问n个城市。
- 他每次仅访问一个城市，最后回到他出发的地方。
- 他应该按什么顺序访问这些城市才能使距离最短?



	A	B	C	…
A	0	12	34	…
B	12	0	76	…
C	34	76	0	…
…	…	…	…	…

例：旅行推销员问题TSP

- TSP的要点如下：
  - 不是一个状态空间问题
  - “状态” = 可能的旅行路线 = （n-1）!/2
- TSP是组合优化中的一个NP难问题，在运筹学和理论计算机科学中非常重要。
- 最早的蚁群优化算法旨在解决旅行推销员问题，其目标是找到连接所有城市的最短往返旅程。
- 一般的算法相对简单，基于一群蚂蚁，每个都能够沿着这些城市形成一个可能的往返旅程。



4.4 粒子群优化PSO

- 由詹姆斯·肯尼迪和拉塞尔·埃伯哈特于1995年提出。受鸟类和鱼类的社会行为的启发。
- 采用若干粒子构成一个围绕搜索空间移动的群体来寻找最优解。
- 搜索空间的每个粒子根据它自己的飞行经验和其它粒子的飞行经验调整它的“飞行”。

鸟群

- 一群鸟在一个区域随机地寻找食物，在该被搜索区域仅有一块地方有食物，所有的鸟都不知道食物在哪儿。但它们在经过每次环飞后知道食物有多远。
- 因此发现食物的最好策略是什么?
- 最有效的方法是跟随离食物最近的鸟。
- 仅需三个简单的规则:
  - 避免与相邻的鸟碰撞;
  - 保持与相邻的鸟相同的速度;
  - 靠近相邻的鸟。