计算机科学188：人工智能导论
2024年春季 Note 4

作者（其他所有注释）：尼基尔·夏尔马
作者（贝叶斯网络注释）：乔希·胡格和杰基·梁，由王蕾吉娜编辑
作者（逻辑注释）：亨利·朱，由考佩林编辑
学分（机器学习和逻辑注释）：部分章节改编自教材《人工智能：一种现代方法》。

上次更新时间：2023年8月26日

局部搜索

在上一篇笔记中，我们想要找到目标状态以及到达该状态的最优路径。但在某些问题中，我们只关心找到目标状态——重构路径可能很简单。例如，在数独中，最优配置就是目标。一旦你知道了它，就知道通过逐个填充方格来达到目标。

局部搜索算法使我们能够找到目标状态，而不必担心到达那里的路径。在局部搜索问题中，状态空间是"完整"解的集合。我们使用这些算法来尝试找到满足某些约束或优化某个目标函数的配置。

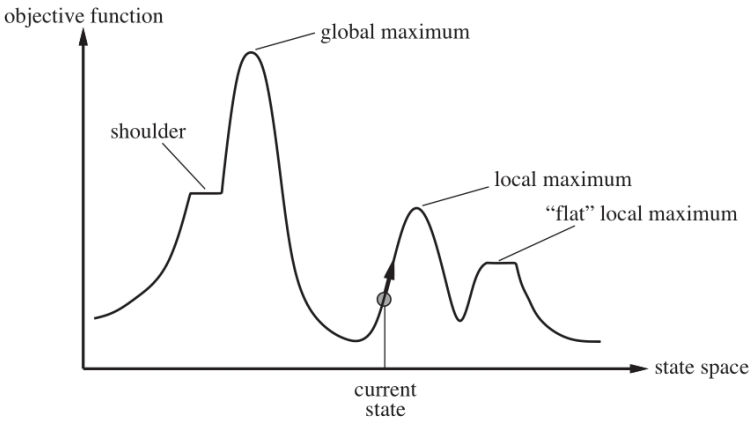


Figure 4.1 A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum. Hill-climbing search modifies the current state to try to improve it, as shown by the arrow. The various topographic features are defined in the text.

上图展示了状态空间上目标函数的一维图。

对于该函数，我们希望找到对应最高目标值的状态。局部搜索算法的基本思想是，从每个状态开始，它们在局部朝着具有更高目标值的状态移动，直到达到最大值（希望是全局最大值）。我们将介绍四种这样的算法：爬山法、模拟退火算法、局部束搜索算法和遗传算法。所有这些算法也用于优化任务，以最大化或最小化目标函数。

爬山搜索

爬山搜索算法（或最陡上升算法）从当前状态朝着使目标值增加最多的相邻状态移动。该算法不维护搜索树，只维护状态以及目标的相应值。爬山法的“贪婪性”使其容易陷入局部最大值（见图4.1），因为在局部这些点对算法来说就像是全局最大值，还容易陷入高原状态（见图4.1）。高原状态可分为“平坦”区域，在该区域没有方向能带来改进（“平坦局部最大值”），或者是进展可能缓慢的平坦区域（“肩部”）。
已经提出了爬山法的变体，比如随机爬山法，它会在可能的向上移动中随机选择一个动作。实践表明，随机爬山法虽然需要更多的迭代次数，但能收敛到更高的最大值。另一种变体是随机侧向移动，它允许进行并非严格增加目标值的移动，从而使算法能够逃离“肩部”区域。

```
function HILL-CLIMBING(problem) returns a state
  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
```

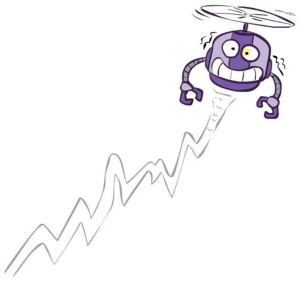
爬山法的伪代码如上文所示。顾名思义，该算法会迭代地移动到目标值更高的状态，直到无法再取得这样的进展为止。爬山法是不完备的。另一方面，随机重启爬山法会从随机选择的初始状态进行多次爬山搜索，由于在某个时刻随机选择的初始状态能够收敛到全局最大值，所以它显然是完备的。

需要注意的是，在本课程后面的内容中，你会遇到“梯度下降”这个术语。它与爬山法的思路完全相同，只是我们要最小化一个成本函数，而不是最大化一个目标函数。

模拟退火搜索

我们要介绍的第二种局部搜索算法是模拟退火算法。模拟退火算法旨在将随机游走（随机移动到附近状态）和爬山法相结合，以获得一种完整且高效的搜索算法。在模拟退火算法中，我们允许移动到目标值可能降低的状态。
该算法在每个时间步选择一个随机移动。如果该移动导致目标值更高，则总是被接受。如果它导致目标值更小，那么该移动以一定概率被接受。这个概率由温度参数决定，温度参数最初较高（允许更多“坏”移动），并根据某种“时间表”降低。从理论上讲，如果温度下降得足够慢，模拟退火算法将以概率趋近于1达到全局最大值。

```
function SIMULATED-ANNEALING(problem,schedule) returns a  state
current ← 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

局部束搜索是爬山搜索算法的另一种变体。两者的关键区别在于，局部束搜索在每次迭代时会跟踪 k 个状态（线程）。该算法从 k 个状态的随机初始化开始，并且在每次迭代时，它会像爬山算法那样采用 k 个新状态。这些并非只是常规爬山算法的 k 个副本。至关重要的是，该算法会从所有线程的后继状态完整列表中选择 k 个最佳后继状态。如果任何一个线程找到了最优值，算法就会停止。

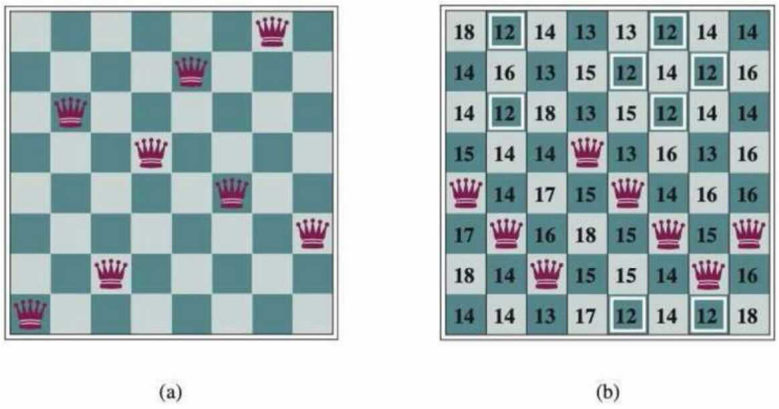
k 个线程可以相互共享信息，使得 “好的” 线程（目标值高的线程）也能够 “吸引” 该区域中的其他线程。
与爬山算法一样，局部束搜索也容易陷入 “平坦” 区域。类似于随机爬山算法，随机束搜索可以缓解这个问题。

遗传算法

最后，我们介绍遗传算法，它是局部束搜索的一种变体，也广泛应用于许多优化任务中。顾名思义，遗传算法的灵感来源于进化。遗传算法最初是从束搜索开始的，有 k 个随机初始化的状态，称为种群。状态（称为个体）表示为有限字母表上的字符串。

让我们回顾一下课堂上提出的八皇后问题。简要回顾一下，八皇后问题是一个约束满足问题，我们希望在一个8×8的棋盘上放置8个皇后。满足约束的解决方案中不会有任何相互攻击的皇后对，即处于同一行、同一列或同一对角线上的皇后。之前介绍的所有算法都是解决八皇后问题的可能方法。

Figure 4.3



(a) 八皇后问题：在棋盘上放置8个皇后，使得没有皇后能够攻击其他皇后。（皇后可以攻击同一行、同一列或对角线上的任何棋子。）除了第四列和第七列的两个皇后在对角线上相互攻击外，此位置几乎是一个解决方案。(b) 一个启发式成本估计为 $h = 17$ 的八皇后状态。棋盘展示了通过在列内移动皇后获得的每个可能后继状态的 h 值。有8步移动并列最佳，值为 $h = 12$ 。爬山算法将从这些移动中选择其一。

对于遗传算法，我们用1到8之间的数字表示八个皇后中的每一个，该数字代表每个皇后在其列中的位置（图4.6中的列 (a)）。使用评估函数（适应度函数）对每个个体进行评估，并根据该函数的值对它们进行排名。对于八皇后问题，这是不相互攻击（无冲突）的皇后对的数量。

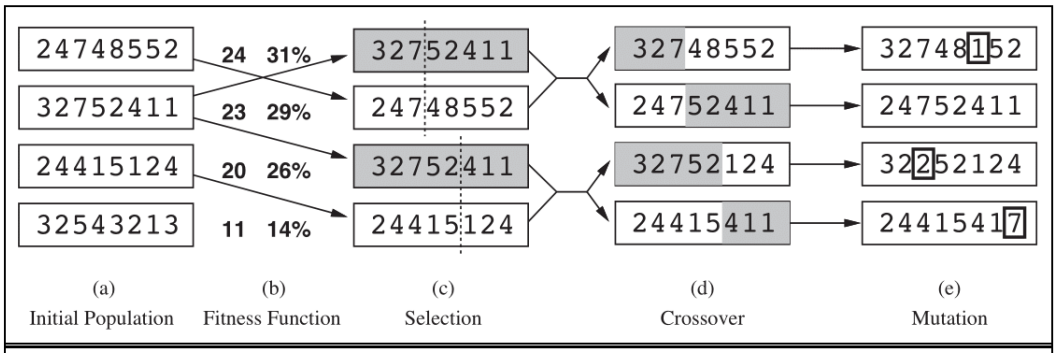


Figure 4.6 The genetic algorithm, illustrated for digit strings representing 8-queens states. The initial population in (a) is ranked by the fitness function in (b), resulting in pairs for mating in (c). They produce offspring in (d), which are subject to mutation in (e).

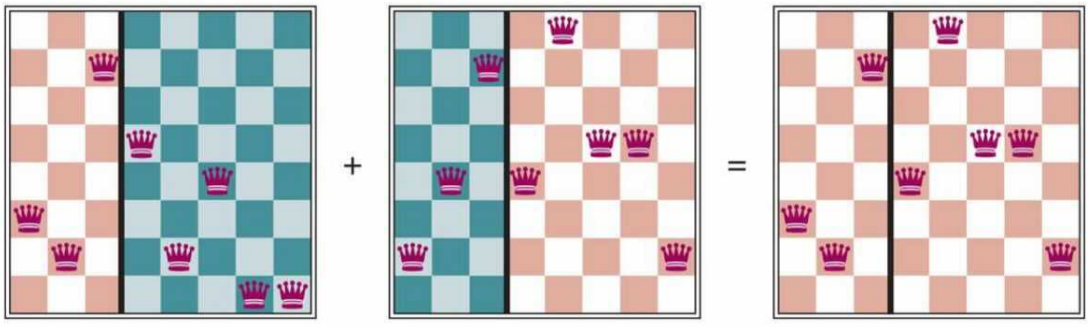
选择一个状态进行“复制”的概率与该状态的值成正比。我们通过从这些概率中进行采样来选择要复制的状态对（图4.6中的(c)列）。后代是通过在交叉点处交叉父代字符串而生成的。每个配对的交叉点是随机选择的。最后，每个后代都有独立概率发生一些随机变异。遗传算法的伪代码如下所示。

函数 遗传算法（种群，适应度函数）返回一个个体输入：种群，一组个体适应度函数，一个衡量个体适应度的函数重复新种群 为空集对于 从 1 到 种群大小 执行随机选择（种群，适应度函数）随机选择（种群，适应度函数）子代 为 繁殖（x，y）如果（小随机概率）则 子代 为 变异（子代）将子代添加到新种群种群 为 新种群直到某个个体足够适应，或者经过了足够长的时间根据适应度函数返回种群中最佳个体函数 繁殖（x，y）返回一个个体输入：，父代个体随机数 从 1 到 返回 拼接（子串（x，1，c），子串 ）</text>

图4.8 遗传算法。该算法与图4.6中所示的算法相同，只有一个变化：在这个更流行的版本中，两个亲本的每次交配只产生一个后代，而不是两个。

与随机束搜索类似，遗传算法在探索状态空间并在各线程之间交换信息时试图向上移动。它们的主要优势在于使用交叉操作——那些已经进化并导致高估值的大字母块可以与其他这样的块组合，并产生一个总得分很高的解决方案。

图4.7



与图4.6(c)中的前两个亲本 □ 以及图4.6(d)中的第一个后代 ′ 相对应的八皇后状态。绿色列在交叉步骤中丢失，红色列被保留。（为了解释图4.6 □ 中的数字：第1行是最下面一行，第8行是最上面一行。）

Summary

在本笔记中，我们讨论了局部搜索算法及其动机。当我们不关心通往某个目标状态的路径，而是希望满足约束条件或优化某个目标时，可以使用这些方法。局部搜索方法使我们在处理大型状态空间时能够节省空间并找到合适的解决方案！

我们介绍了一些相互依存的基础局部搜索方法：

- 爬山法
- 模拟退火
- 局部束搜索
- 遗传算法

优化函数的概念将在本课程后面再次出现，特别是在我们讲解神经网络的时候。