经 典 搜 索 Classic Search

- 目前我们所学到的搜索算法被设计成系统地探索问题的空间。

  在内存中保持一条或多条路径，并且在沿着该路径的每个点上记录哪些已经被探索过了，目标找到了，这个路径也就构成了问题的一个解。

- 很多问题中，到达目标的路径是无关紧要的。

优化和组合优化问题

- 很多问题属于优化问题，或者可以转化为优化问题
- 如TSP问题，皇后问题

优化问题的描述

设x是决策变量，D是x的定义域，f(x)是指标函数，g(x)是约束条件集合。则优化问题可以表示为，求解满足g(x)的f(x)最小值问题。

$$\min_{x \in D} \left(f(x) \mid g(x) \right)$$

如果在定义域D上，满足条件g(x)的解是有限的，则优化问题称为组合优化问题。

算法的时间复杂度

对于组合优化问题，由于其可能的解是有限的，当问题的规模比较小时，总可以通过枚举的方法获得问题的最优解，但当问题的规模比较大时，就难于求解了。

常用的算法复杂度函数

$$O(\log n),\ O(n),\ O(n\log n),\ O(n^2),\ O(2^n),\ O(n^{\log n}),\ O(n!),\ O(n^n)$$

时间复杂性函数比较（10亿次/秒）

输入量n / 复杂性函数	10	20	30	40	100
n	10ns	20ns	30ns	40ns	100ns
$n\log n$	10ns	26.0ns	44.3ns	64.1ns	200ns
n^2	100ns	400ns	900ns	1.6us	10us
2^n	1.0us	1.0ms	1.1s	18.3min	4.0世纪
$n!$	3.6ms	77.1年	8.4×10^{13}世纪	2.6×10^{29}世纪	3.0×10^{139}世纪

一些难的组合优化问题

- 旅行商问题
- 背包问题
- 装箱问题
- . . .

寻求在可以接受的时间内得到满意解的方法

邻域的概念

- 邻域，简单的说就是一个点附近的其他点的集合。

  在距离空间，邻域就是以某一点为中心的圆。

- 组合优化问题的定义：

  设D是问题的定义域，若存在一个映射N，使得：

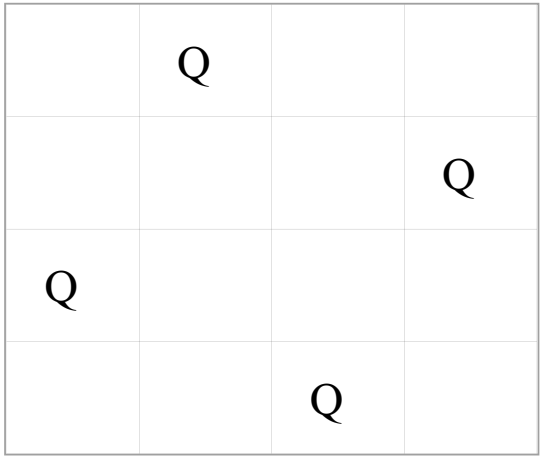
$N: x \in D \rightarrow N(x) \in 2^D$，

且$x \in N(x)$，称为一个邻域映射，其中2^D表示D的所有子集组成的集合。

$N(x)$称为x的邻域，$y \in N(x)$称为x的一个邻居。

皇后问题

- S={S_i}表示一个可能解，其中S_i表示在第i行，第S_i列有一个皇后。
  如四皇后问题的一个解：S=(2, 4, 1, 3)



- 定义映射N为棋盘上任意两个皇后的所在行或列进行交换，即S中任意两个元素交换位置。

- 例：当S = (2, 4, 1, 3)时，其邻域为：

N(S) = {(4, 2, 1, 3), (1, 4, 2, 3), (3, 4, 1, 2), (2, 1, 4, 3), (2, 3, 1, 4), (2, 4, 3, 1)}

旅行商问题

用一个城市的序列表示一个可能的解。

通过交换两个城市的位置获取S的邻居

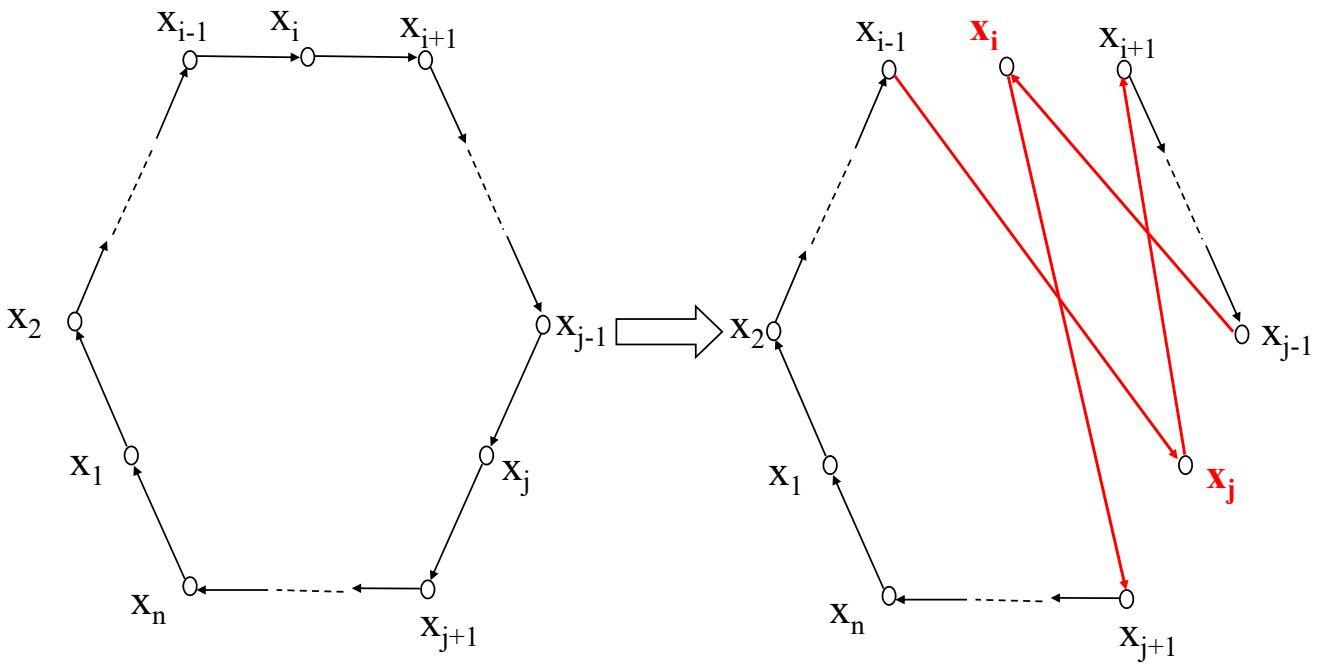
- 例：简单交换方法

设$S = (x_1, x_2, \ldots x_{i-1}, \mathbf{x_i}, x_{i+1}, \ldots, x_{j-1}, \mathbf{x_j}, x_{j+1}, \ldots, x_n)$

则通过交换x_i和x_j两个城市的位置可以得到S的一个邻居：

$S' = (x_1, x_2, \ldots x_{i-1}, \mathbf{x_j}, x_{i+1}, \ldots, x_{j-1}, \mathbf{x_i}, x_{j+1}, \ldots, x_n)$

旅行商问题



旅行商问题

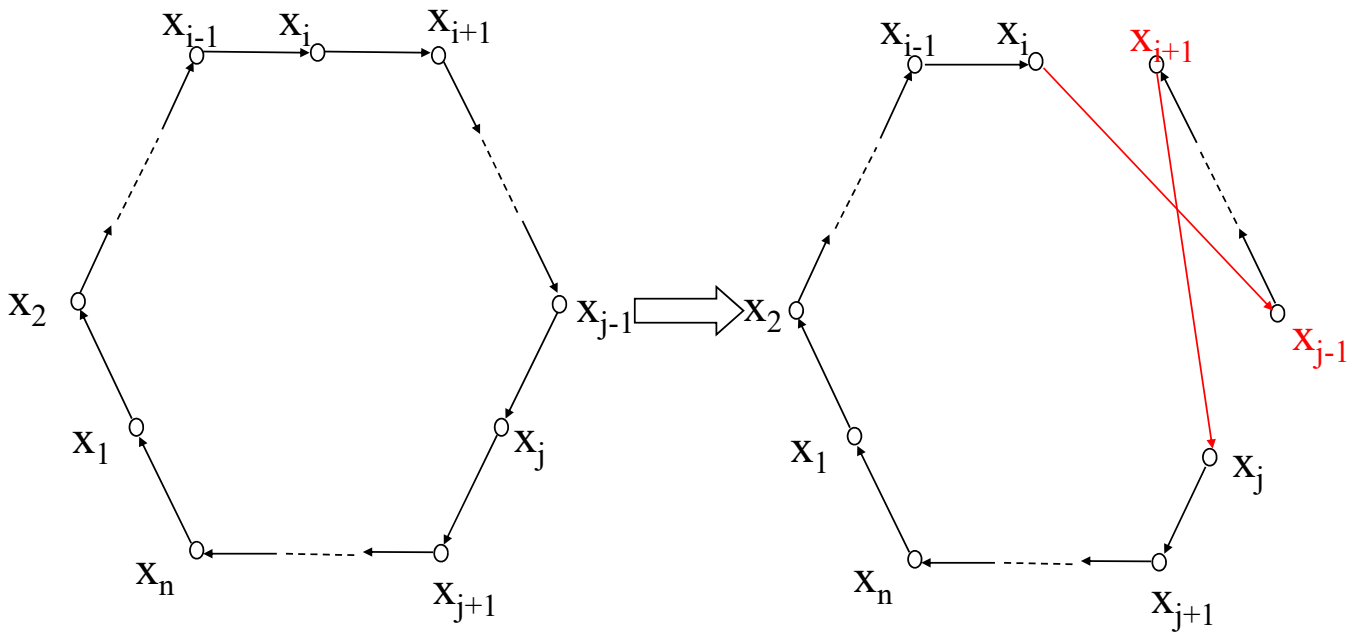
- 例：逆序交换方法

设x_i、x_j是选取的两个城市，所谓的逆序交换方式是指，通过逆转x_i、x_j两个城市之间的城市次序来得到S的邻居。

设：$S = (x_1, x_2, \ldots x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n)$

则：$S' = (x_1, x_2, \ldots x_{i-1}, x_i, x_{j-1}, x_{j-2} \ldots, x_{i+1}, x_j, x_{j+1}, \ldots, x_n)$

旅行商问题



局 部 搜 索 Local Search

- 局部搜索是一种不同类型的算法，它不关心路径。
- 从单个当前的节点（而不是多个条路径）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 的 优 点

- 使用很少的内存（通常是常数）
- 在大的无限空间状态空间中发现合理的解

局 部 搜 索 的 应 用

很多应用问题是与路径无关的，目标状态本身就是解。例如：

- integrated-circuit design 集成电路设计
- factory-floor layout 工厂车间布局
- job-shop scheduling 车间作业调度
- automatic programming 自动规划
- telecommunications 通讯
- network optimization 网络优化
- vehicle routing 车辆路由
- portfolio management 投资组合管理

爬山法 (Hill-climbing)

考虑能否利用反馈信息以帮助决定生成什么样的解，这种改进就是爬山法。

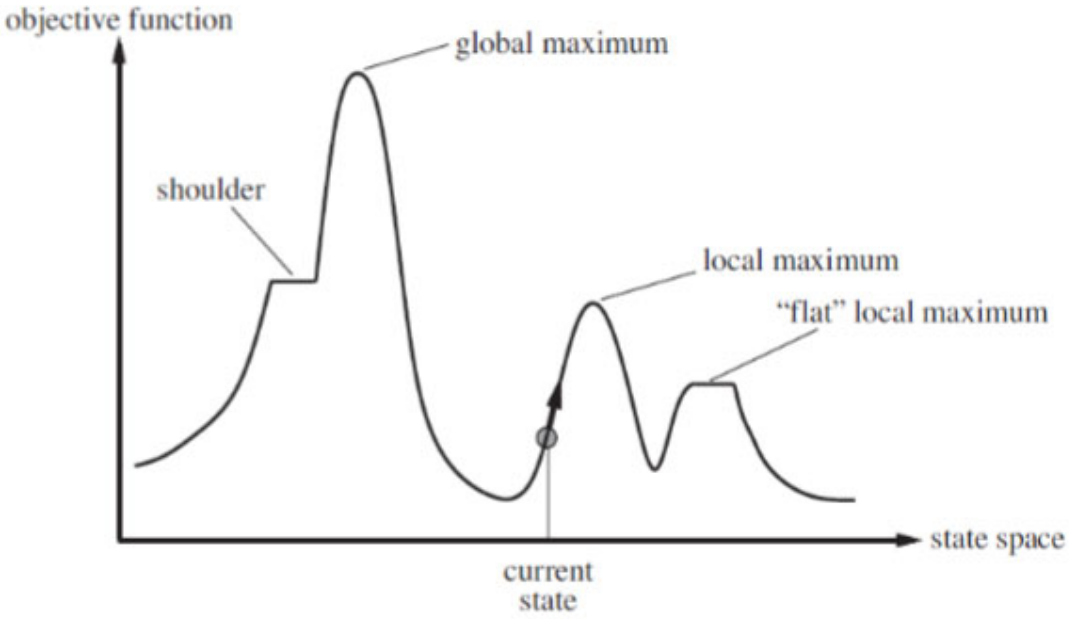
爬山法把出现在离目标更近的节点选作为它下一步的节点。此算法取名于模仿一个在黑暗中在半山腰迷路的徒步旅行者。假设他的营地在山顶，那末即使在黑暗中，徒步旅行者也知道每向上走一步就是向正确的目标前进了一步。

爬山法 (Hill-climbing)

- 是一种属于局部搜索家族的数学优化方法。
- 是一种迭代方法：开始时选择问题的一个任意解，然后递增的修改该解的一个元素，若得到一个更好的解，则将之修改为新的解，重复这个过程直到无法找到更好的改善。
- 大多数基本的局部搜索算法都不保持一颗搜索树。

状态空间地形图 (State-Space Landscape)

- It can be explored by one of local search algorithms.
  可通过局部搜索算法对其进行搜索。
- A complete local search algorithm always finds a goal if one exists.
  一个完备的局部搜索算法总能找到一个存在的目标。
- an optimal local search algorithm always finds a global minimum or maximum.
  一个最优的局部搜索算法总能找到一个全局的最小或最大值。



爬山搜索算法

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    persistent: current, a node
                neighbor, a node
    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
      neighbor ← a successor of current
      if neighbor.VALUE ≤ current.VALUE then return current.STATE
      current ← neighbor
```

Steepest-ascent version: at each step, current node is replaced by the best neighbor (the neighbor with highest value), else it reaches a "peak".

最陡爬坡版：当前节点每一步都用最佳邻接点（具有最高值的邻接点）替换，否则到达一个“峰值”。

爬山法的基本步骤

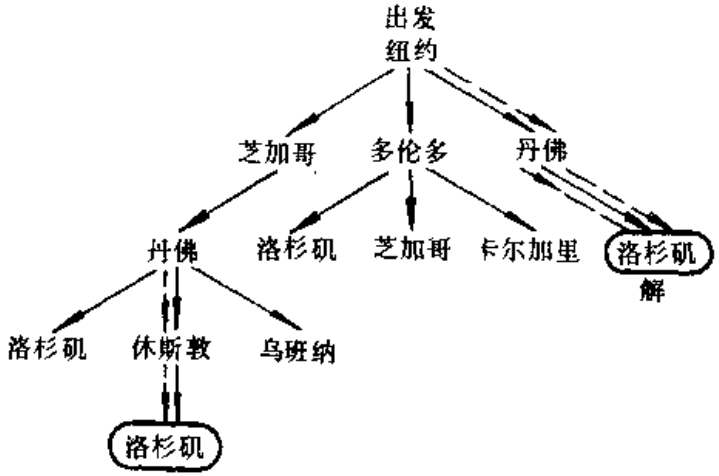
1 生成第一个可能的解。若是目标，则停止；否则转下一步。

2 从当前可能的解出发，生成新的可能解集。

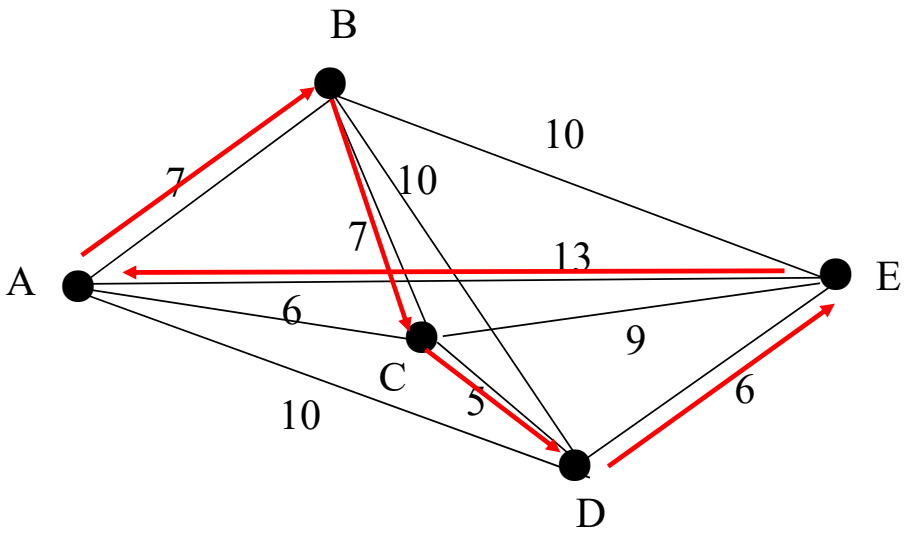
2.1 用测试函数测试新的可能解集中的元素，若是解，则停止；否则转2.2。

2.2若不是解，则将它与至今已测试过的“解”比较。若它最接近解，则保留作为最佳元素；若它不最接近解，则舍弃。

3 以当前最佳元素为起点，转（2）。



5城市旅行商问题



设初始的可能解：$x_0 = (a, b, c, d, e)$

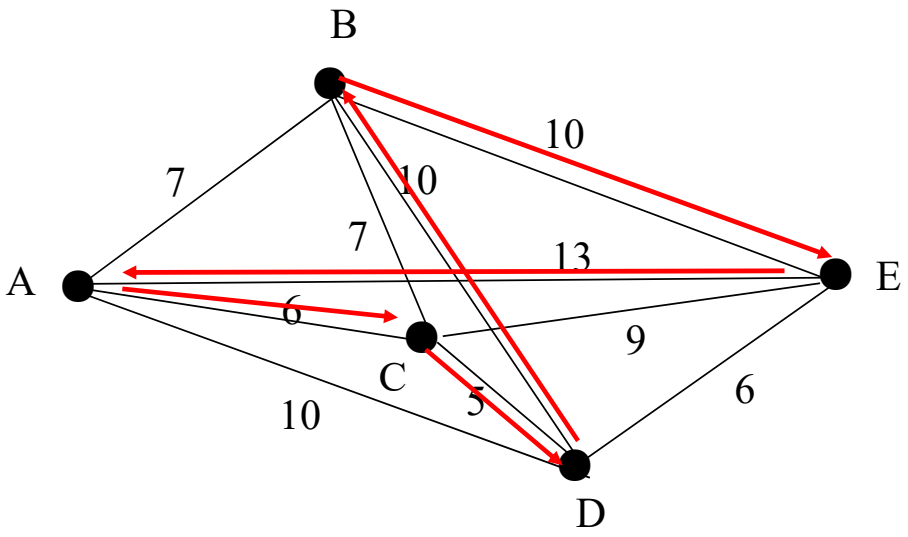
$f(x_b) = f(x_0) = 38$

通过交换两个城市获得领域:

P = {(a, c, b, d, e), (a, d, c, b, e), (a, e, c, d, b), (a, b, d, c, e), (a, b, e, d, c), (a, b, c, e, d)}

设每次随机从P中选择一个邻居。

第一次循环



从P中选择一个元素，

假设$x_n = (a, c, b, d, e)$, $f(x_n) = 42$,

此时：$f(x_n) > f(x_b)$,

$P = P - \{x_n\}$

$= \{(a, d, c, b, e), (a, e, c, d, b), (a, b, d, c, e),$

$(a, b, e, d, c), (a, b, c, e, d)\}$

第二次循环

从P中选择一个元素，

假设x_n = (a, d, c, b, e),

$f(x_n) = 45$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\}$

= {(a, e, c, d, b), (a, b, d, c, e),

(a, b, e, d, c), (a, b, c, e, d)}

第三次循环

从P中选择一个元素，

假设x_n = (a, e, c, d, b),

$f(x_n) = 44$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\}$

= {(a, b, d, c, e), (a, b, e, d, c),

(a, b, c, e, d)}

第四次循环

从P中选择一个元素，

假设x_n = (a, b, d, c, e),

$f(x_n) = 44$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\}$ = {(a, b, e, d, c),
(a, b, c, e, d)}

第五次循环

从P中选择一个元素，

假设x_n = (a, b, e, d, c),

$f(x_n) = 34$,

$f(x_n) < f(x_b)$,

x_b = (a, b, e, d, c),

P = {(a, e, b, d, c), (a, d, e, b, c),
(a, c, e, d, b), (a, b, d, e, c),
(a, b, c, d, e), (a, b, e, c, d)}

第六次循环

从P中选择一个元素，

假设x_n = (a, e, b, d, c),

$f(x_n) = 44$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\}$

= {(a, d, e, b, c), (a, c, e, d, b),
(a, b, d, e, c), (a, b, c, d, e),
(a, b, e, c, d)}

第七次循环

从P中选择一个元素，

假设x_n = (a, d, e, b, c),

$f(x_n) = 39$, $f(x_n) > f(x_b)$,

$P = P - \{x_n\}$

= {(a, c, e, d, b), (a, b, d, e, c),
(a, b, c, d, e), (a, b, e, c, d)}

第八次循环

从P中选择一个元素，

假设x_n = (a, c, e, d, b),

$f(x_n) = 38$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\}$

= {(a, b, d, e, c), (a, b, c, d, e),

(a, b, e, c, d)}

第九次循环

从P中选择一个元素，

假设x_n = (a, b, d, e, c),

$f(x_n) = 38$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\}$ = {(a, b, c, d, e),

(a, b, e, c, d)}

第十次循环

从P中选择一个元素，

假设x_n = (a, b, c, d, e),

$f(x_n) = 38$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\} = \{(a, b, e, c, d)\}$

第十一次循环

从P中选择一个元素，

假设x_n = (a, b, e, c, d),

$f(x_n) = 41$,

$f(x_n) > f(x_b)$,

$P = P - \{x_n\} = \{\}$

P等于空，算法结束，

得到结果为x_b = (a, b, e, d, c),

$f(x_b) = 34$。

八皇后问题

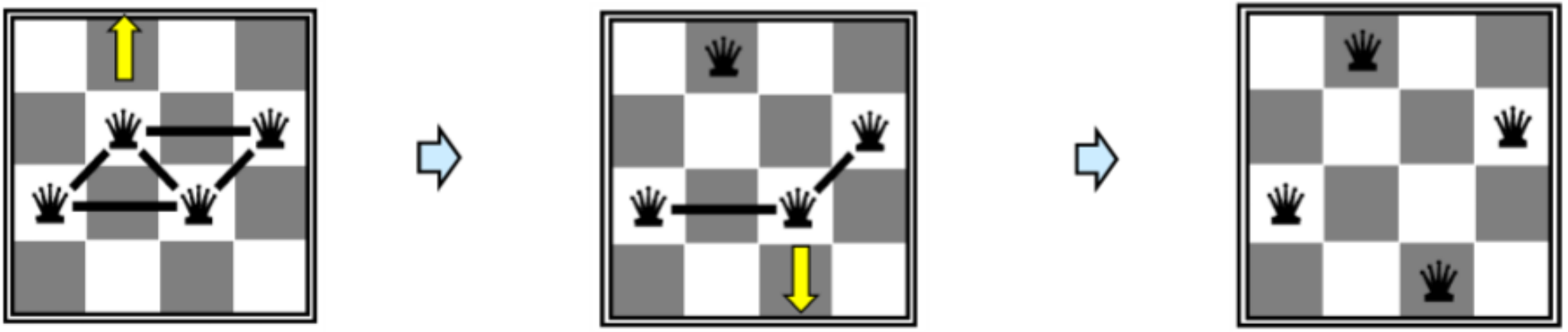
八皇后问题：和八数码问题不一样，八皇后问题目标就是找到最终状态，所有初始状态随机改变是可以允许的。

把每个状态都表示为在棋盘上放8个皇后，每列一个。

后继函数返回的是移动一个皇后和它同一列的另一个方格中的所有可能的状态（因此每个状态有8*7:56个后继）。

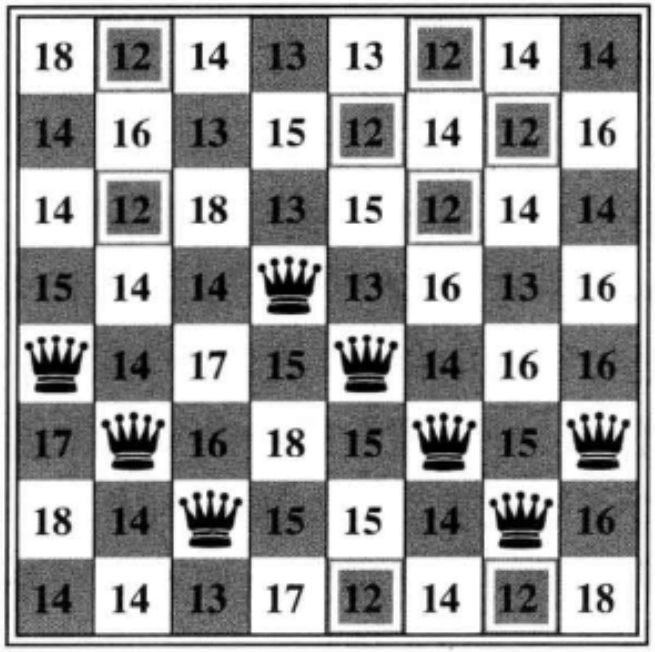
启发式函数h是可以彼此攻击的皇后对的数量。

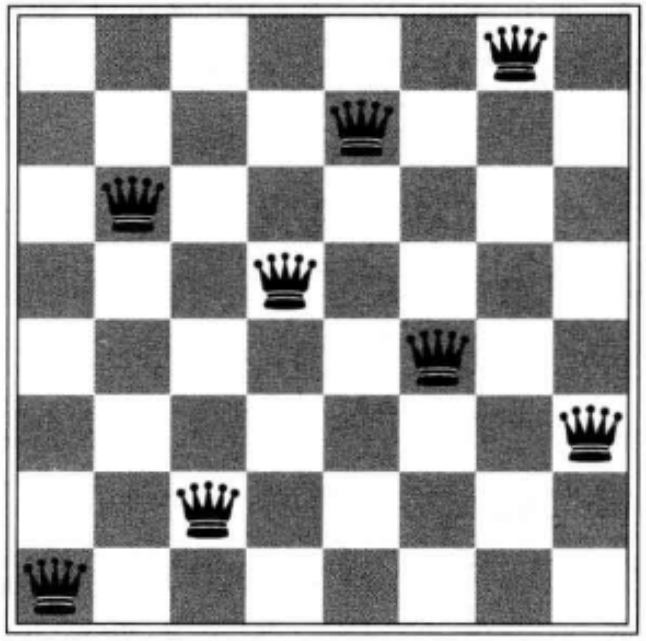
启发式函数图示



(a) $h = 5$　　(b) $h = 2$　　(c) $h = 0$

八皇后问题





h = 互相攻击到的皇后对数

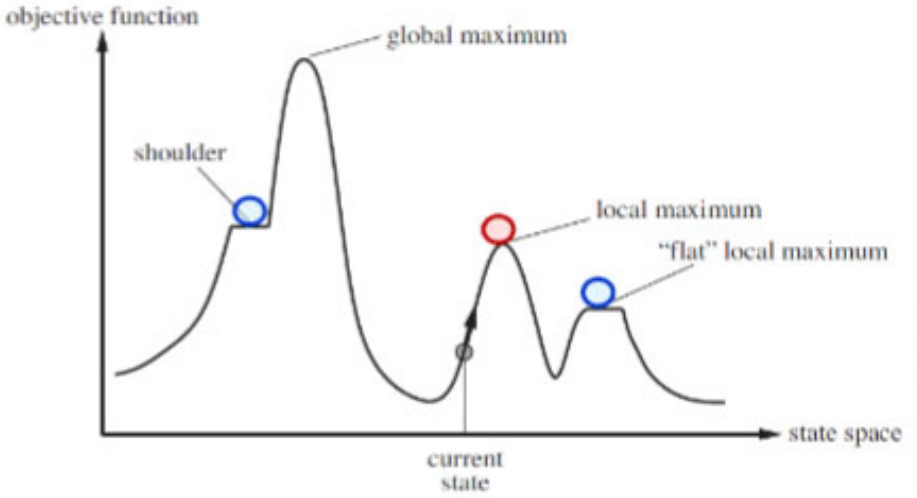
爬山法的弱点

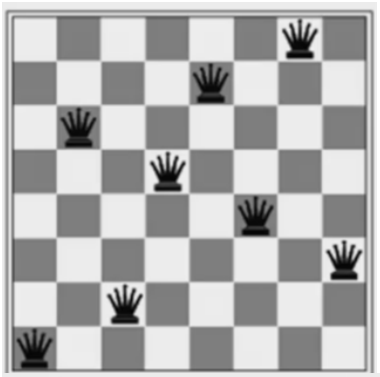
- 在下面三种情况下经常被困

  局部最大值：高于相邻结点但低于全局最大值。

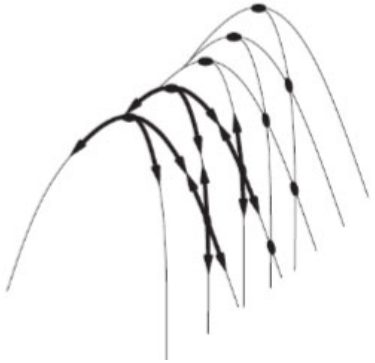
  高原：可能是一个平坦的局部最大值，或山肩。

  山脊：结果是一系列局部最大值，非常难爬行。





●一个局部最值点（h = 1）



爬山法的变形

1. 随机爬山法：在上山移动中随机地选择下一步；选择的概率随上山移动的陡峭程度而变化。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能找到更好的解。

2. 首选爬山法：它在实现随机爬山法的基础上，采用的方式是随机地生成后继结点直到产生一个优于当前结点的后继。这个算法在有很多后继结点的情况下有很好的效果。

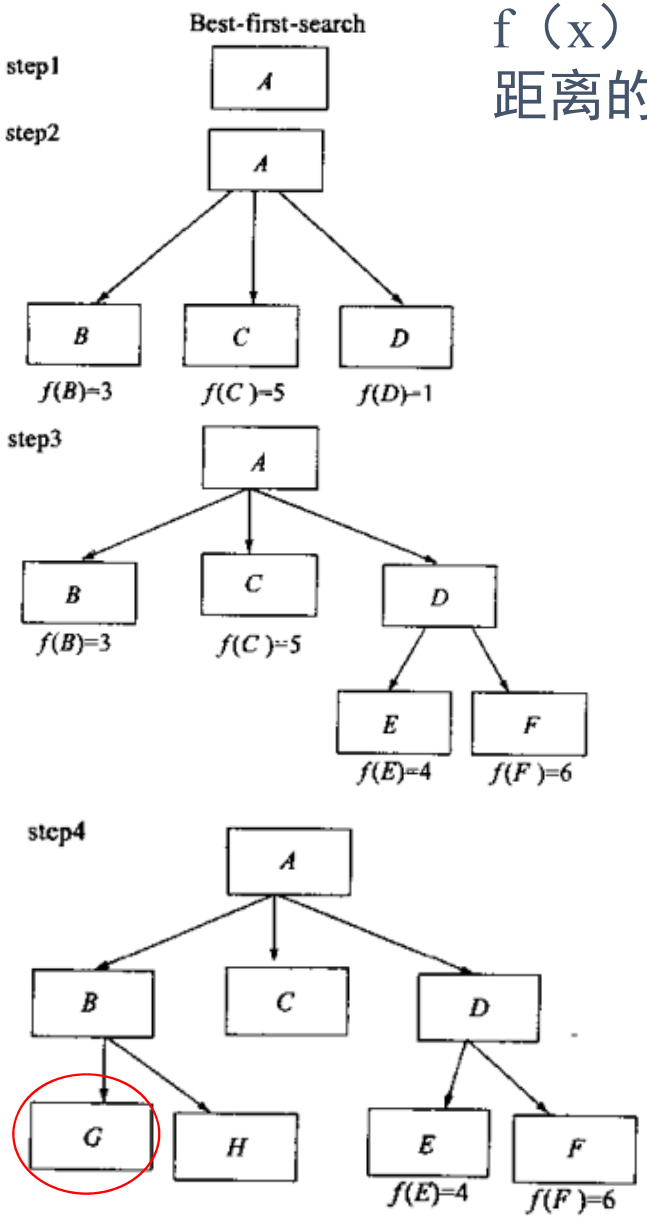
3. 随机重新开始的爬山法：求解过程中，当得到了局部极大值时，如果不是全局最优解，则随机生成初始状态，重新求解，直到得到全局最优解。

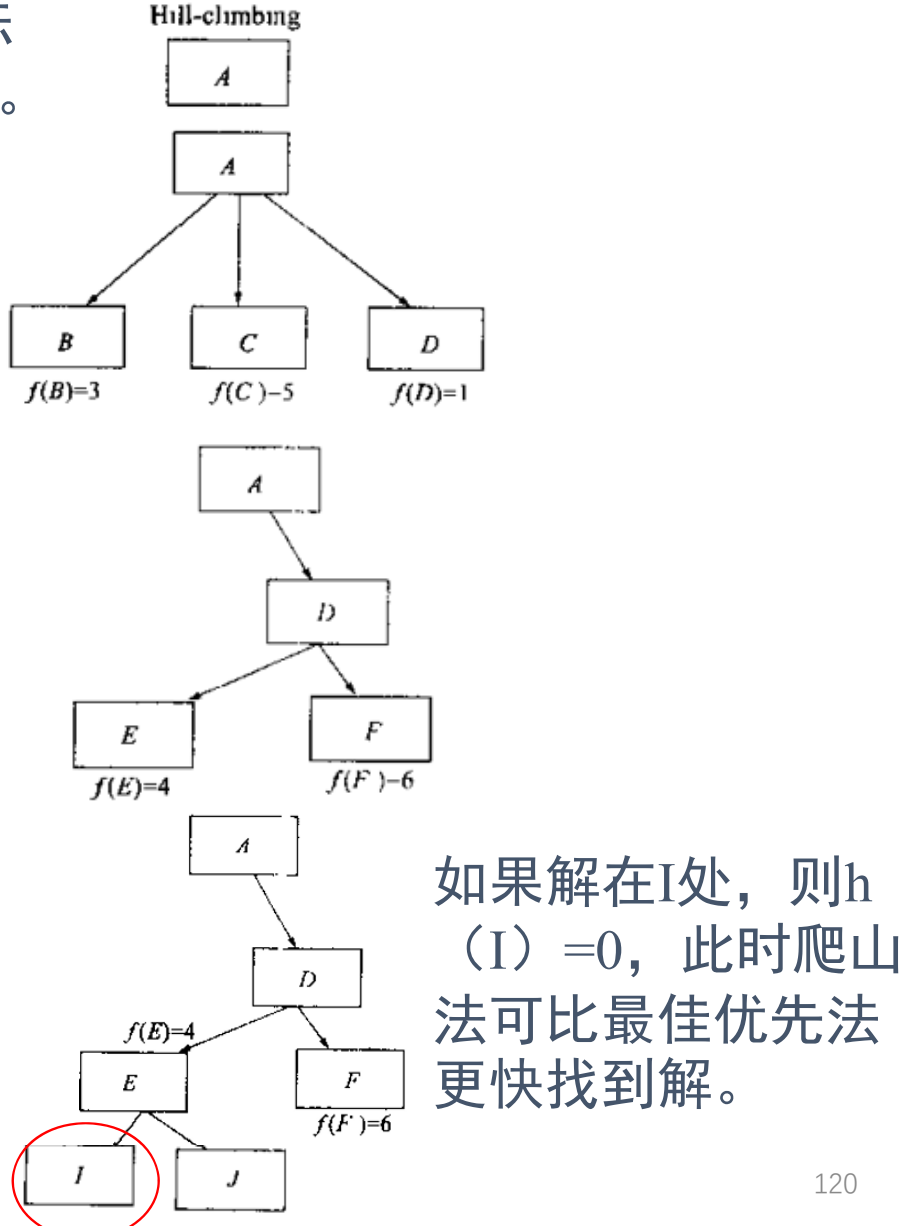
爬山法

爬山法算法成功与否在很大程度上取决于状态空间地形图的形状：如果在图中几乎没有局部极大值和高原，随机重新开始的爬山法将会很快地找到好的解。

另一方面，许多实际问题的地形图存在着大量的局部极值。NP问题通常有指数级数据量的局部极大值。尽管如此，经过少数随机重新开始的搜索之后还是能找到一个合理的较好的局部极大值的。

f（x）为x到目标距离的估计函数。





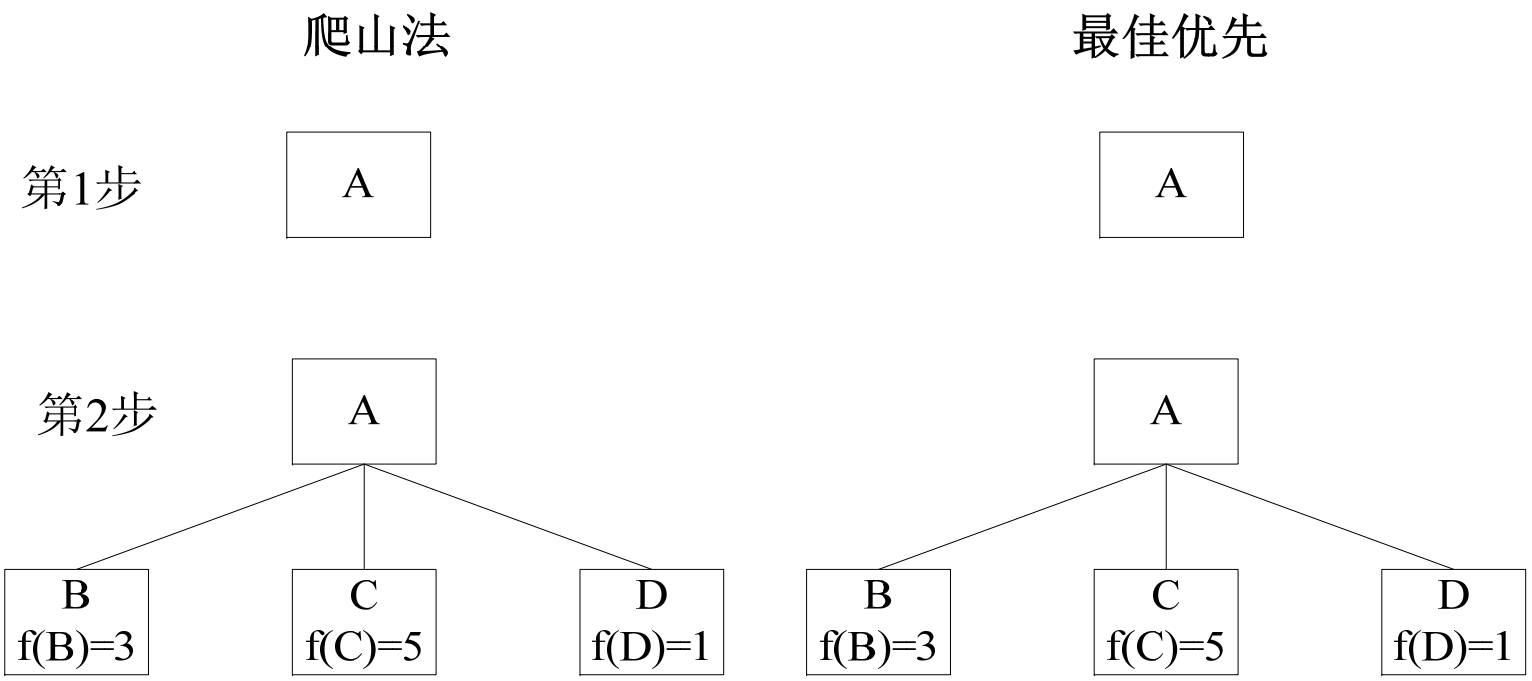
如果解在G处，则h（G）=0，爬山法没办法找到解。

如果解在I处，则h（I）=0，此时爬山法可比最佳优先法更快找到解。

爬山法和最佳优先搜索的比较

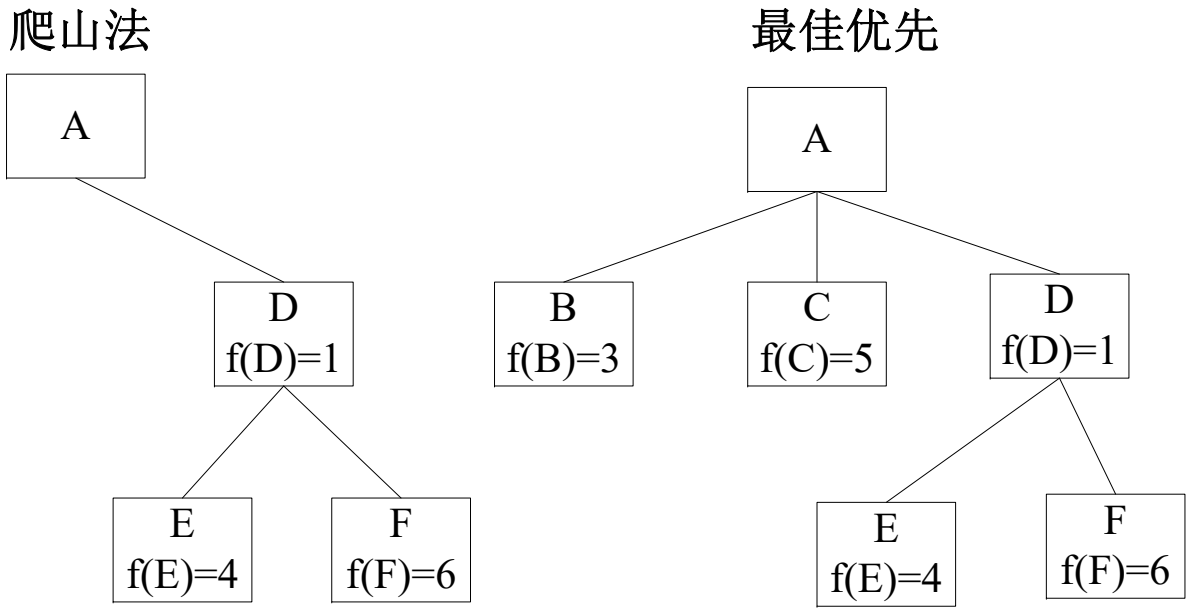
- 爬山法：对新的可能解与至今测试过的解进行比较，如最接近解，则保留为最佳解，否则舍弃。
- 最佳优先搜索：将新生成的解集加入到可能解集中，从所有解集中挑选最好的元素作为起点。
- 区别：前者从当前后继中找最好的，后者为全局最佳。

解题过程



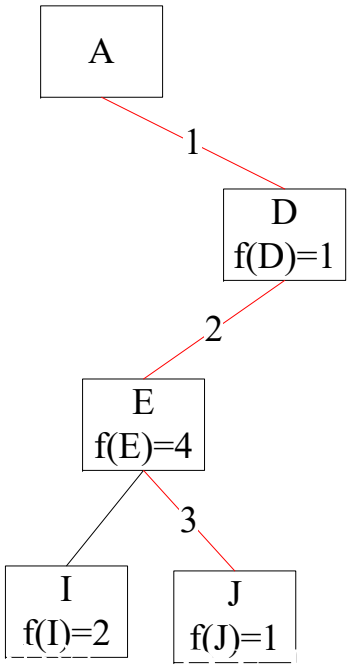
解题过程

第3步

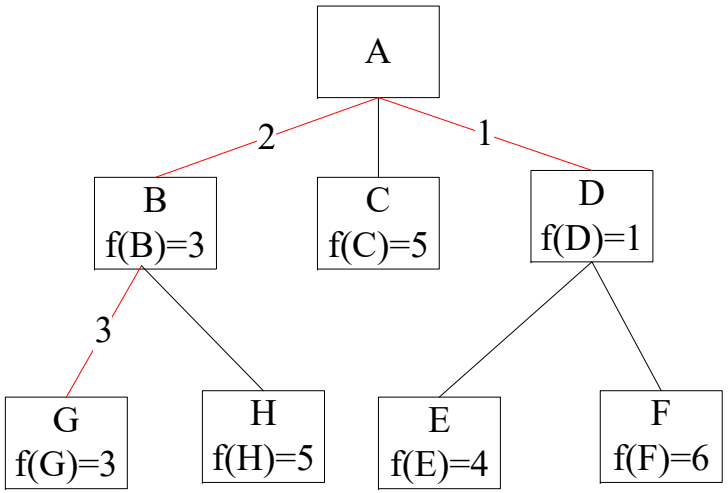


解题过程

爬山法



最佳优先



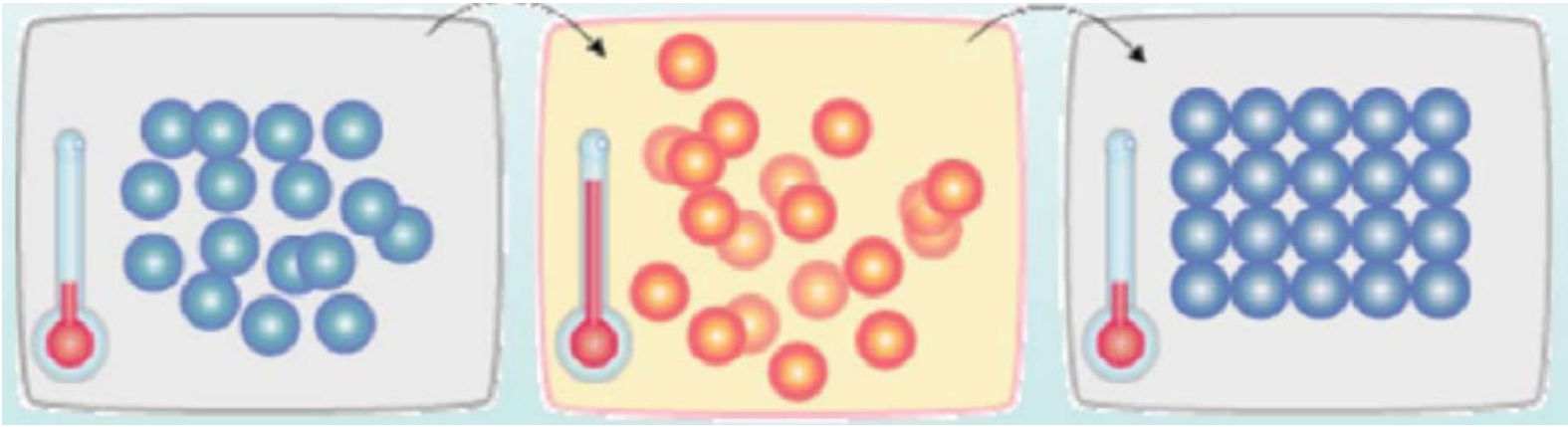
3.3 随 机 搜 索

当问题的空间很大，而可行解较多，并且对解的精度要求不高时，随机搜索是很有效的解决办法，因为其他的做法在这个时候的时空效率不能让人满意。

模拟退火法(simulated Annealing)

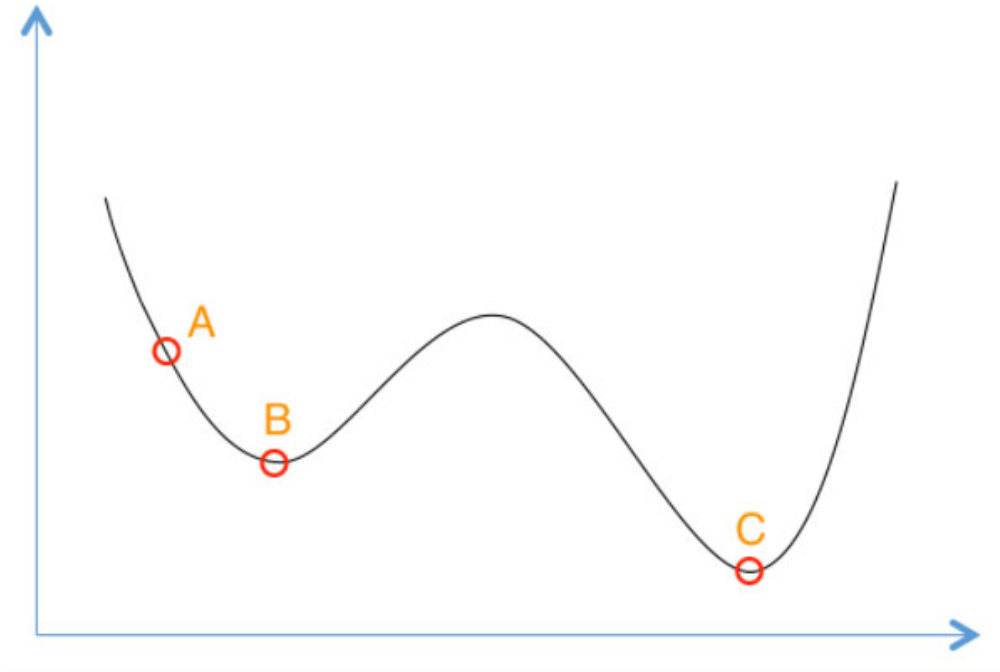
- 是局部搜索算法的一种扩展
- 最早由Metropolis在1953年提出，Kirkpatrick等人在1983年成功地将模拟退火算法用于求解组合优化问题。
- 基本思想是借用金属的退火过程改进局部搜索算法。

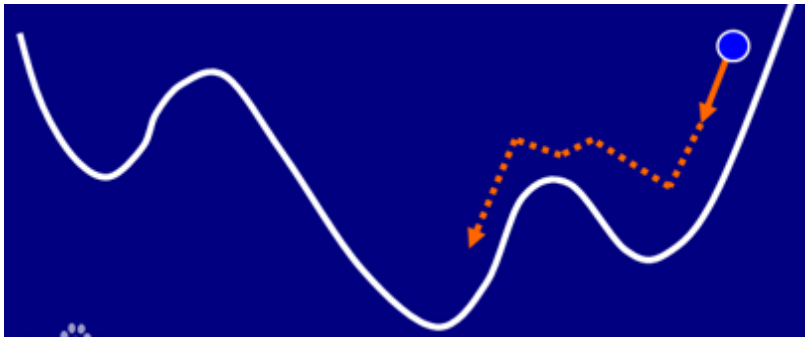
什么是退火



非晶体状态　　加温　　退火

模拟退火法(simulated Annealing)





考虑一个滚动的球，从一个高能量状态滚动到一个低谷，然后滚动到高一点的能量状态。

Metropolis准则

从状态i转换为状态j的准则：

- 如果E(j)≤E(i)，则状态转换被接受；
- 如果E(j)＞E(i)，则状态转移被接受的概率为：

$$e^{\frac{E(i)-E(j)}{KT}}$$

- 其中E(i)、E(j)分别表示在状态i、j下的能量，T是温度，K＞0是波尔兹曼常数。

模拟退火法(simulated Annealing)

将内能E模拟为目标函数值f，温度T演化成控制参数t，即得到解组合优化问题的模拟退火算法：由初始解i和控制参数初值t开始，对当前解重复“产生新解→计算目标函数差→接受或舍弃”的迭代，并逐步衰减t值，算法终止时的当前解即为所得近似最优解，这是基于蒙特卡罗迭代求解法的一种启发式随机搜索过程。

组合优化问题与退火过程的类比

固体退火过程	组合优化问题
物理系统中的一个状态	组合优化问题的解
状态的能量	解的指标函数
能量最低状态	最优解
温度	控制参数

模拟退火算法(simulated Annealing)

- **Initial Solution** 初始解

  Generated using an heuristic. Chosen at random.

  使用启发式方法生成。随机选择。

- **Neighborhood** 相邻节点

  Generated randomly. Mutating the current solution.

  随机生成。当前解的变异。

- **Acceptance** 接受条件

  Neighbor has lower cost value, higher cost value is accepted with the probability p.

  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。

- **Stopping Criteria** 停止判据

  Solution with a lower value than threshold. Maximum total number of iterations.解具有比阈值低的值。已达到迭代最大总次数。

模拟退火算法(simulated Annealing)

```
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)
```

- 退火过程由冷却进度表(Cooling Schedule)控制，包括控制参数的初值t及其衰减因子Δt、每个t值时的迭代次数L和停止条件S。

如果总存在一个比现在更好的下一个状态，并且该状态就是下一个要扩展的状态。则该算法和爬山算法类似。

如果不是这种情况，则会是后一句else，这一部分就是模拟退火部分逐一检测合理的后序状态，查看该状态出现的概率是否大于[0,1]之间的随机值。如果是则选择该状态，否则检测下一个可能的状态。我们希望的是至少有一个状态出现的概率大于随机生成的概率。

模拟退火法（simulated Annealing）

模拟退火的解不像局部搜索那样最后的结果依赖初始点。它引入了一个接受概率p。如果新的点更好，则p=1，表示选取新点；否则，接受概率p是当前点、新点以及另一个控制参数“温度”T的函数。也就是说，模拟退火没有像局部搜索那样每次都贪婪地寻找比现在好的点，差一些的点也有可能接受进来。随着算法的执行，系统温度T逐渐降低，最后终止于不再有可接受变化的低温。

算法中的问题

- 初始温度的选取
- 温度的下降方法
- 内循环的结束条件，即每个温度状态交换何时结束
- 外循环的结束条件，即温度下降到什么时候结束

起始温度的选取

一个合适的初始温度，应保证平稳分布中每一个状态的概率基本相等。

温度的下降方法

1. 等比例下降
2. 等值下降
3. 基于距离参数的下降方法

每一温度下的停止准则

1. 固定长度方法
2. 基于接受率的停止准则
3. 阈值

算法终止原则

1.零度法

2.循环总控制法

3.无变化控制法

应用举例——旅行商问题

- 设有n个城市，城市间的距离用矩阵D=[d_{ij}] (i,j=1,2,…,n)表示，其中d_{ij}表示城市i与城市j之间的距离。当问题对称时，有$d_{ij}=d_{ji}$。有一个旅行商从一个城市出发，每个城市访问一次，并且只能访问一次，最后再回到出发城市。问如何行走才能使得行走的路径长度最短。

解的表示

n个城市的任何一种排列均是问题的一个可能解，表示为：

$$(\pi_1,...,\pi_n)$$

- 其中 π_i=j表示第i个到达的城市是j，并且默认：$\pi_{n+1}=\pi_1$

指标函数

$$f(\pi_1,...,\pi_n)=\sum_{i=1}^{n} d_{\pi_i\pi_{i+1}}$$

$$\pi_{n+1}=\pi_1$$

新解的产生

采用两个城市间的**逆序交换**方式得到问题的一个新解。

- 设当前解是$(\pi_1,\ldots,\pi_n)$
- 被选中要逆序交换的城市是第u和第v个到访的城市，u<v。则逆序排列u和v之间的城市，得到问题的新解为：

$$(\pi_1,\ldots,\pi_u,\boxed{\pi_{v-1},\ldots,\pi_{u+1}},\pi_v,\pi_{v+1},\ldots,\pi_n)$$

指标函数差

两个路径的距离差为：

$$\Delta f=(d_{\pi_u\pi_{v-1}}+d_{\pi_{u+1}\pi_v})-(d_{\pi_u\pi_{u+1}}+d_{\pi_{v-1}\pi_v})$$

新解的接受准则

$$A_t=\begin{cases}1 & \text{如果}\Delta f<0\\ e^{-\frac{\Delta f}{t}} & \text{其他}\end{cases}$$

参数的确定

- 康立山等人的方法：
  - 初始温度t_0=280；
  - 在每个温度下采用固定的迭代次数，L_k=100n，n为城市数；
  - 温度的衰减系数＝0.92，即t_{k+1}=0.92×t_k；
  - 算法的停止准则为：当相邻两个温度得到的解无任何变化时算法停止。

- Nirwan Ansari和Edwin Hou的方法：
  - 初始温度t_0：从t_0=1出发，并以t_0=1.05×t_0对t_0进行更新，直到接受概率大于等于0.9时为止，此时得到的温度为初始温度；
  - 在每个温度下采用固定的迭代次数，L_k=10n，n为城市数；
  - 温度的衰减系数＝0.95，即t_{k+1}=0.95×t_k；
  - 算法的停止准则为：温度低于0.01，或者没有任何新解生成。

10城市旅行商问题求解结果

	路径长度	出现次数	平均转移次数	路径
最优	2.691	906	3952	BCADEFGHIJ
次优	2.752	46	4056	BCADEGFHIJ
第三	2.769	10	4053	DEFGHIJCBA
最差	2.898	5	4497	ABCDEFHIJG

20城市旅行商问题求解结果

	路径长度	出现次数	平均转移次数	路径
最优	24.38	792	8740	ACLBIQFTMEP RGSOJHDKN
次优	24.62	167	8638	ADCLBIQFTME PRGSOJHKN
第三	25.17	39	9902	ANKDHIOJSGR PEMTFQBLC
最差	25.50	1	5794	AQFTMEPRGSJ OIBLCDHKN

其他典型的随机搜索算法

随机搜索的目的往往是增加算法的灵活性和搜索过程扩展方式的多样性，使得算法避免陷入过早收敛的境地。

- 遗传算法
- 粒子群算法
- 蚁群算法
- 人工免疫算法