第二章　八皇后问题的局部搜索求解

2.1　教学目标

让学生能够应用基于局部搜索的技术来实现一个问题求解智能体。这一类问题关注目标本身，而不是达到目标的路径。学生能够通过本次实验学会应用局部搜索技术解决优化问题。

2.2　实验任务

如图2.1所示，要求在国际象棋棋盘中放置8个皇后，使得任何一个皇后都不能攻击其他任意一个皇后。国际象棋的规则为一个皇后可以攻击同一行、同一列、同一对角线上的棋子。2.1中的布置并不满足要求，因为左上角和右下角相互攻击。

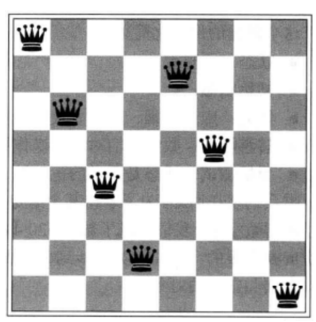

图 2.1: 八皇后问题

2.3　模型与方法

根据布局规则，每列只能放一个皇后，因此将问题状态定义为一个布局，用8元组$S=(s_1,s_2,\cdots,s_8)$表示，其中S_i为第i列的那个皇后所在的行数。例如图2.1的布局$S=(1,3,5,7,2,4,6,8)$。假设函数$A(S,i,j)$判断布局S的第i列和第j列的是否存在攻击行为，

$$A(S,i,j)=\begin{cases}1 & s_i=s_j\\ 1 & |s_i-s_j|=|i-j|\\ 0 & else\end{cases} \tag{2.1}$$

对每个布局S, 定义一个价值函数$V(S)$, 跟S中存在攻击的皇后对数有关，

$$V(S)=100-\sum_{i=1}^{8}\sum_{j=i+1}^{8}A(S,i,j) \tag{2.2}$$

实验的目标状态的价值到达最大值100，成为一个求最大值问题

2.3.1　爬山法

在解决优化问题时，爬山法只是一个简单的循环。以求最大值问题为例，爬山法不断从邻居(后继状态)中选择一个状态使得目标值增大，在到达一个峰顶时停止。算法只需记录当前状态和目标值，不需要维护搜索树。

```python
def GreedyHillClimbing(problem):
    current=Node(problem.InitialState)
    while True:
        neighbour=Node(current.HighestSucessor())
        if neighbour.Value<=current.Value:
            return current.State
        current=neighbour
```

其中问题模型为

```python
class Problem():
    def __init__(self,start):
    self.InitialState=start
```

结点定义为

图 2.2: 爬山法示意图

```
class Node():
    def __init__(self,state):
        self.State=state
        self.Value=Value(state)
    def HighestSucessors():
        return a successor state with higest Value
```

2.3.2 爬山法的变形

前述HillClimbing中寻找的最大价值邻居结点，故称为**贪婪爬山法**。贪婪爬山法能到达的峰值，只是周围的邻居都比它小，并不代表是一个全局范围内的最大值，所以称为局部最优状态。如图2.2左侧中，爬山法到达红点时会停止搜索。

HillClimbing中如果发现价值最大的邻结点的价值等于当前结点的价值，也会停止搜索。当这样的邻居很多时，形成一个高原（如图2.2 右侧的红色区域）。如果允许继续在高原上连续搜索多次，则称为**侧向移动爬山法**。侧向移动爬山法的价值有可能跳出高原区域继续上升。

如果并不选取最大价值邻居，而是在比当前更大的邻居中随机选取一个，则称为**随机爬山法**。随机爬山法有时候效果会更好。

爬山法能到达的局部最优点跟起点 InitialState有关系，当爬山法到达一个局部最优点时，重启算法随机换一个起始点重新搜索，有可能得到更好的解。随机重启多次的爬山法称为**随机重启爬山法**。

2.3.3 模拟退火方法

模拟退火方法可以克服爬山法停在局部最优状态，成为一种可能达

到“全局最优”的局部搜索方法。相对爬山法，其主要不同点在于不再限制于价值比当前点更高的邻居，而是以小概率允许邻居价值相对或者更小。例如在图2.2 左侧情形中，到达红点后，有一定概率往下，从而有可能到达第二个峰值，从而取得全局最优。

接受价值变小的的概率不是常数：如果接受价值更小的状态，则后续继续的概率会以指数级别减少。同时，该概率也会随着迭代的次数增加而减小。

```
Def SA(problem, schedule):#schedule 是一个的“温度”表格
    current=Node(problem.InitialState)
    for i in range(MAX):
        T=schedule[i]
        if T==0:
            return current.State
        neighbour= Node(current.RandSuccessor())
        dE=neighbour.Value-current.Value
        P=exp(dE/T)
        if dE>0 or rand()<P:
            Current=neighbour
```

2.4　实验步骤

1. 随机产生1000个起始点；
2. 分别实现贪婪爬山法、侧向移动爬山法、随机爬山法；
3. 对前三个算法实现随机重启；
4. 实现模拟退火方法；
5. 利用1000个起始点，对前述方法分别统计成功求解的次数，成功求解时的迭代步数，失败时的步数；

2.5　小结

按实验步骤完成实验后，提供数据和代码，并完成对实验结果的分析和总结，撰写实验报告。