Lecture4-1 有信息搜索与局部搜索

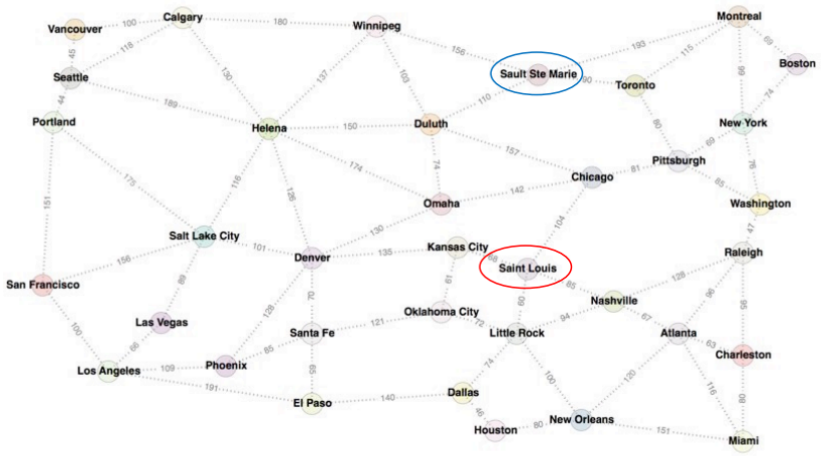
1. 有信息搜索 Informed Search

有信息搜索可以利用全局的知识（知道一些与目标相关的信息）

- 知道与目标的距离
- 使用启发式函数估计状态与目标的距离

策略

- 贪心算法
- A* 算法
- IDA*



Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156

- 有信息：知道每个城市到达 Sault Ste Marie 的直线距离

贪心算法

代价函数 $h(n)$，估计了节点 n 到目标的距离

- 例如：$h_{SLD}(n)$ = 节点 n 到 Sault Ste Marie 的直线距离

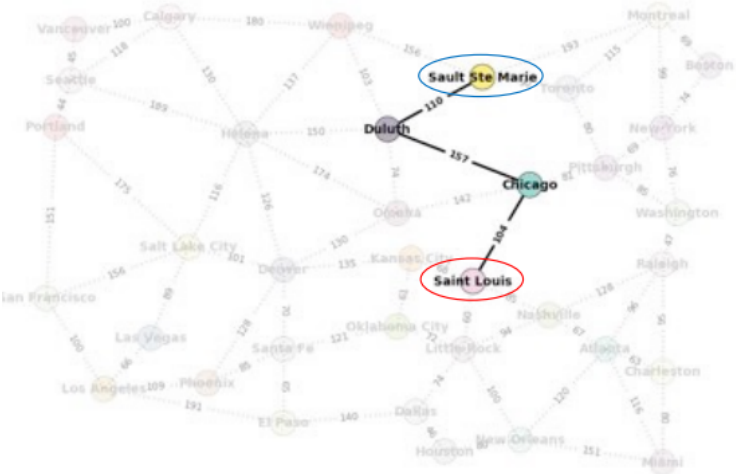
贪心算法每次**扩展看起来最近的节点**

伪代码

```
function GREEDY-BEST-FIRST-SEARCH(initialState, goalTest)
    return SUCCESS or FAILURE

    # cost f(n) = h(n)
    frontier = Heap.new(initialState)
```

```
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin() # 与 UCS 不同的是，这里的min是有信息的
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in forntier ∪ explored:
                frontier.insert(neighbour)
            else if neighbour in frontier:
                frontier.decreaseKey(neighbour)

    return FAILURE
```

贪心算法示例



A* 搜索

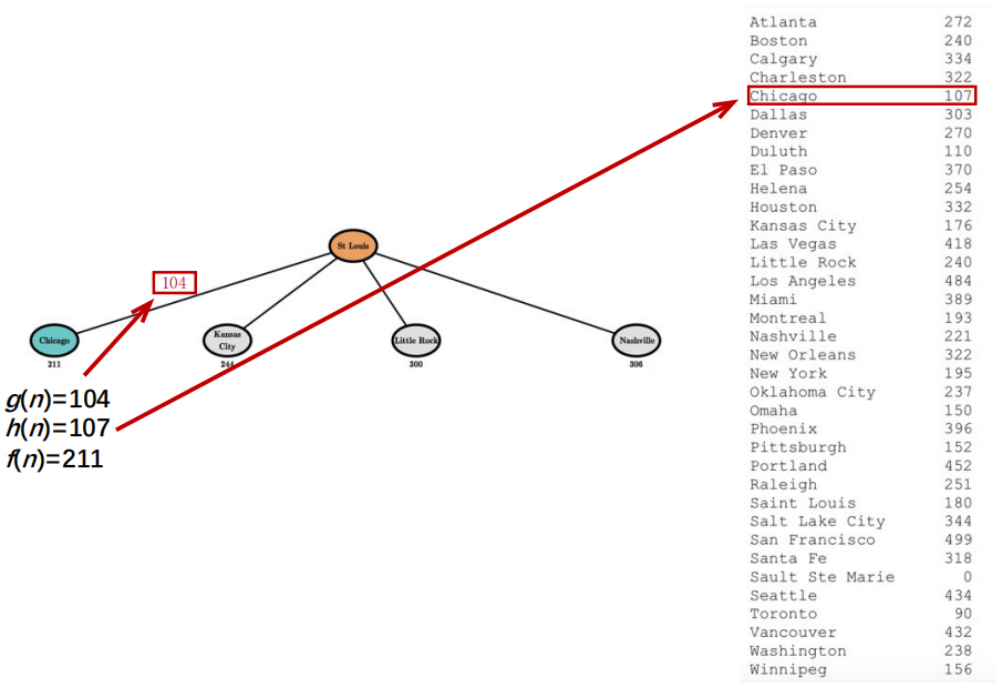
- 最小化估计的解总成本
- 代价函数 $f(n)$ 结合了
  - $g(n)$：展开路径开销
  - $h(n)$：节点 n 到目标的开销

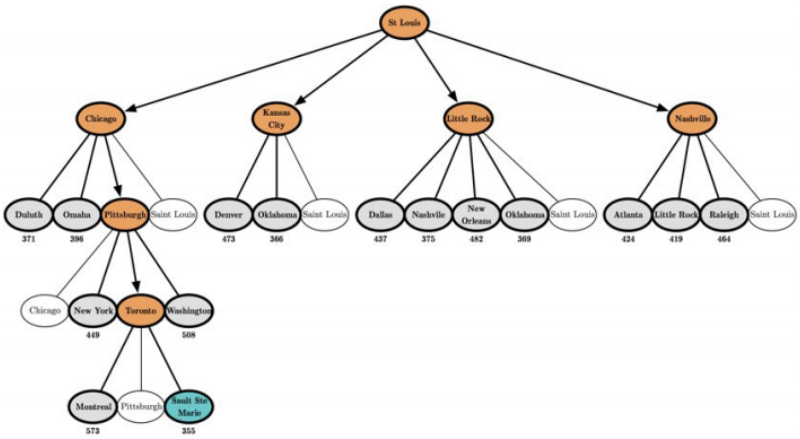
    - $f(n) = g(n) + h(n)$
- $f(n)$ 是估计的代价最便宜的解

伪代码

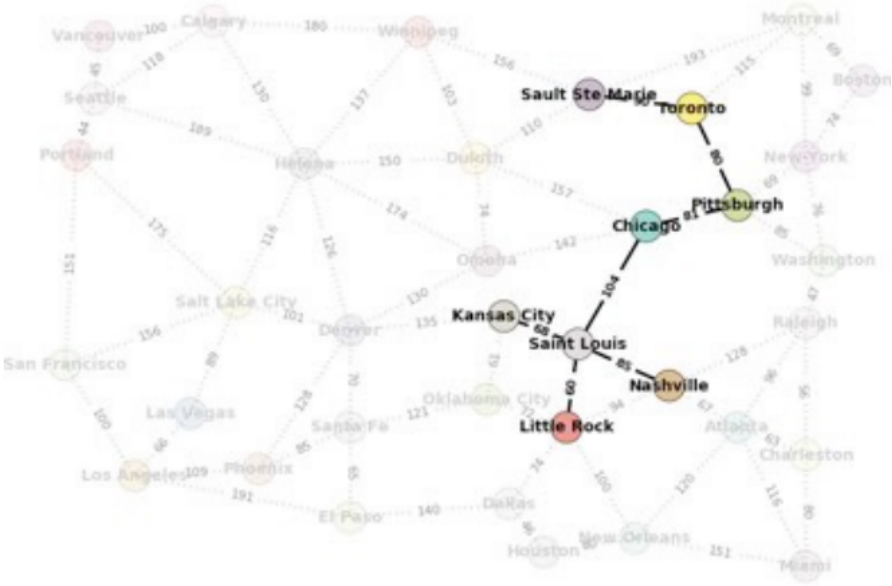
```
function A-STAR-SEARCH(initialState, goalTest)
    return SUCCESS or FAILURE

    # cost f(n) = g(n) + h(n)
    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in forntier U explored:
                frontier.insert(neighbour)
            else if neighbour in frontier:
                frontier.decreaseKey(neighbour)

    return FAILURE
```

A* 算法示例





Atlanta	272
Boston	240
Calgary	334
Charleston	322
Chicago	107
Dallas	303
Denver	270
Duluth	110
El Paso	370
Helena	254
Houston	332
Kansas City	176
Las Vegas	418
Little Rock	240
Los Angeles	484
Miami	389
Montreal	193
Nashville	221
New Orleans	322
New York	195
Oklahoma City	237
Omaha	150
Phoenix	396
Pittsburgh	152
Portland	452
Raleigh	251
Saint Louis	180
Salt Lake City	344
San Francisco	499
Santa Fe	318
Sault Ste Marie	0
Seattle	434
Toronto	90
Vancouver	432
Washington	238
Winnipeg	156



评价指标

- **完成性 Completeness**：是的
- **时间复杂度 Time Complexity**： 指数时间
- **空间复杂度 Space Complexity**：将每个节点都放在内存中
- **最佳性 Optimality**：是的

2. 局部搜索 Local Search

引入

- 到目前为止，搜索算法的设计都是为了**系统地探索搜索空间**，环境是**可观察的、确定性的、已知的**，解决方案是一系列的**动作**
- 现实世界的问题更为复杂，当一个目标被找到时，通往这个目标的道路就构成了问题的解决方案，但是，根据应用程序的不同，**路径可能重要，也可能无关紧要**， 如果路径无关紧要/系统搜索是不可能的，那么考虑另一类算法

- 在这种情况下，我们可以使用**迭代改进算法，局部搜索**，在纯优化问题中也很有用，其中目标是根据**优化函数**找到最佳状态
- 局部搜索的原因是全局搜索算法需要的时间复杂度和空间复杂度太大了，一般承受不了，所以采用了这种节省的搜索算法

示例 - 八皇后问题

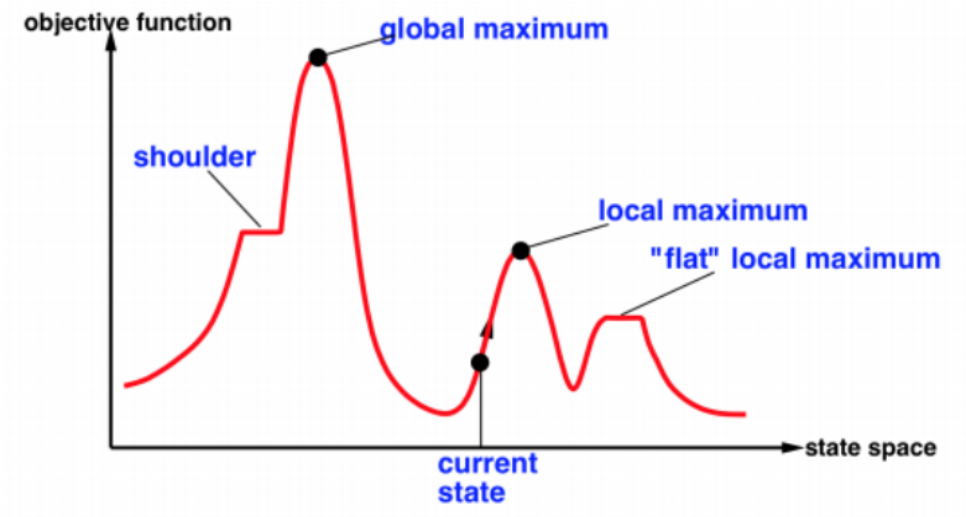
真正重要的是找到最终解，而不是到达它的中间步骤

想法：保持单一的“当前”状态，并尝试改进它，只移动皇后到它的邻居

优势

- 不需要维护搜索树
- 使用很少的内存
- 通常可以在连续或大的状态空间中找到足够好的解

策略



State space landscape

- **爬山 Hill Climbing**（最陡峭的上坡/下坡）
- **模拟退火 Simulated Annealing**：受统计物理启发
- **当地的定向搜索 Loacl Beam Search**
- **遗传算法 Genetic Algorithm**：受进化生物学启发

爬山算法 Hill Climbing

- 也被称为**贪心的局部搜索**
- 只关注眼前的好邻居，而不放眼更远
- 搜索上山：向海拔/值递增的方向移动，以找到山顶
- 当它达到峰值时终止，可以终止与局部最优解，全局最优解或可能卡住，没有进展
- 节点是一种状态和一种值

伪代码

```
function HILL-CLIMBING(initialState)
    return State that is a local maximum

    initialize current with initialState

    loop do
        neighbour = a highest-valued successor of current

        if neighbour.value ≤ current.value:
            return current.state

        current = neighbour
```

爬山算法的变种

- **横向移动 Sideways moves**：从最佳后继者与当前状态具有相同价值的平台状态中逃脱
- **随机重新开始 Random-restart**：克服局部最大值的爬山算法，继续尝试，要么找到一个目标，要么找到几个可能的解决方案，然后选择最大值
- **随机爬坡 Stochastic**：爬坡动作中随机选择

模拟退火 Simulated Annealing

伪代码

```
function Simulated-Annealing
    return a solution State

    for t = 1 to ∞ do
        T = schedule(t) # 初始温度，比如T0=100，每一代T都会下降
        if T = 0 then return current
        next = a randomely selected successor of current
        ΔE = value(current) - value(next)
        if ΔE < 0
            current = next # 邻居比我好，替换
        else
            # 邻居没我好，以一个概率替换，ΔE越大，替换的概率越小
            current = next only with probablity e^{-ΔE/T}
```

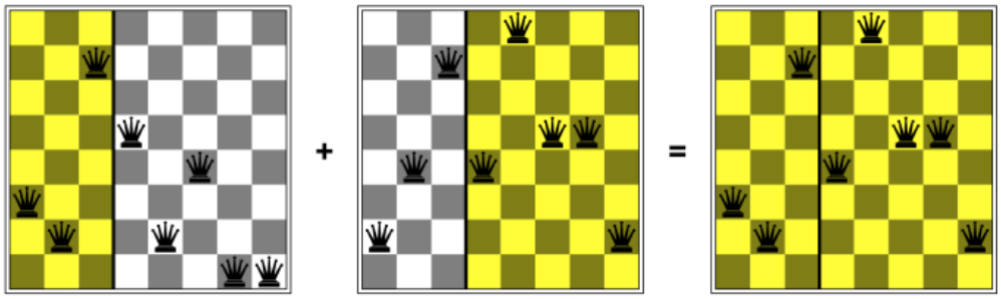
遗传算法 Genetic Algorithms

- 遗传算法 GA 是随机波束搜索的一种变体
- 继承状态是通过合并两个父状态而不是修改单个状态生成的
- 这个过程受到**自然选择**的启发
- 从 **k 个随机生成的状态**开始，称为**种群 population**，每个状态被称为一个**个体 individual**
- 一个个体通常由一串 0 和 1 或数字表示，是一个有限的集合
- 目标函数称为**适应度函数 fitness function**：较好的状态有较高的适应度函数值
- 成对的个体被随机选择以某些概率进行**繁殖 reproduction**
- 在字符串中随机选择一个**交叉点 crossover point**

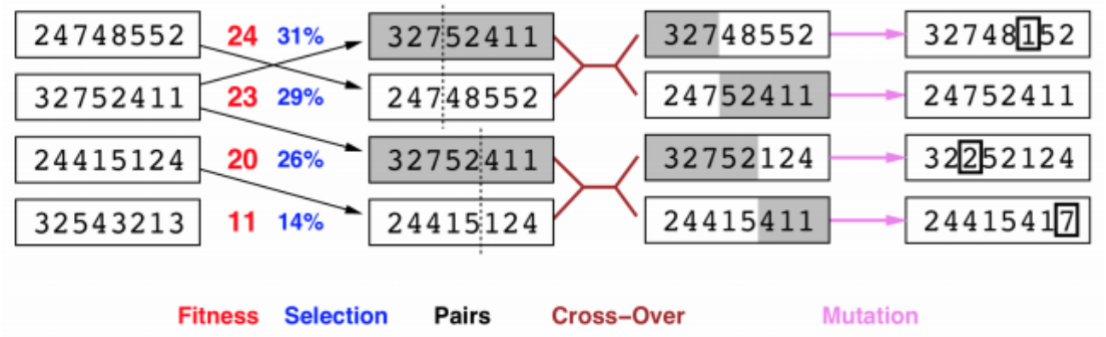
- **后代 offspring** 是通过在交叉点上与双亲杂交而产生的
- 字符串中的每个元素也会以很小的概率发生某种**变异 mutation**

示例 —— 八皇后问题

- 在八皇后问题中，一个皇后可以用 1 到 8 的字符串数字表示，表示 8 个皇后在 8 列中的位置
- 可能的适应度函数是非攻击皇后对的数量



- 繁殖八皇后问题的子代



伪代码

```
function GENETIC-ALGORITHM(population, fitness-function)
    return an individual

    repeat
        initialize new-population with Ω
        for i = 1 to size(population) do
            x = random-select(population, fitness-function)
            y = random-select(population, fitness-function)
            child = cross-over(x,y)
            mutate(child) with a small random probability
            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 w.r.t fitness-function
```