启发式算法简介及其在数学建模中的应用

周吕文

2014 年 11 月 20 日

引言

启发式算法 (Heuristic Algorithm) 兴起于上世纪 80 年代, 启发式算法是一种基于直观或经验的局部优化算法, 人们常常把从大自然的运行规律或者面向具体问题的经验和规则中启发出来的方法称之为启发式算法. 现在的启发式算法也不是全部来自然的规律, 也有来自人类积累的工作经验. 这些算法包括禁忌搜索 (tabu search)[1], 模拟退火 (simulated annealing)[2], 遗传算法 (genetic algorithms)[3], 人工神经网络 (artificial neural networks)[4], 蚁群算法 (Ant Algorithm)[5] 等, 下面给出其中几种启发式算法的简单定义:

- 模拟退火: 通过模拟物理退火过程搜索最优解的方法.
- 遗传算法: 通过模拟自然进化过程搜索最优解的方法.
- 神经网络: 模仿动物神经网络行为特征, 进行分布式并行信息处理的算法数学模型.
- 蚁群算法: 模仿蚂蚁在寻找食物过程中发现路径的行为来寻找优化路径的机率型算法.

启发式算法主要用于解决大量的实际应用问题. 目前, 这些算法在理论和实际应用方面得到了较大的发展. 无论这些算法是怎样产生的, 它们有一个共同的目标－求 NP-难组合优化问题的全局最优解. 虽然有这些目标, 但 NP-难理论限制它们只能以启发式的算法去求解实际问题. 而启发式算法使得能在可接受的计算费用内去寻找尽可能好的解, 但不一定能保证所得解的可行性和最优性, 甚至在多数情况下, 无法描述所得解与最优解的近似程度. 启发式算法包含的算法很多, 例如解决复杂优化问题的蚁群算法 (Ant Colony Algorithms). 有些启发式算法是根据实际问题而产生的, 如解空间分解, 解空间的限制等; 另一类算法是集成算法, 这些算法是诸多启发式算法的合成. 现代优化算法解决组合优化问题, 如 TSP(Traveling Salesman Problem) 问题, QAP(Quadratic Assignment Problem) 问题, JSP(Job-shop Scheduling Problem) 问题等效果很好 [6].

启发式算法是数学建模竞赛中最常用的算法之一. 在近十年来的美国大学生数学建模竞赛 (MCM) 和中国大学生数学建模竞赛 (CUMCM) 中, 几乎每两年就会出现适合或需要应用启发式算法的赛题. 表1为近十来年美国大学生数学建模竞赛中用到启发式算法的特等奖论文不完全统计.

本节我们只介绍模拟退火算法和遗传算法, 并以 TSP 问题为例介绍两种算法具体的应用方法. 在介绍模拟退火算法和遗传算法之前, 在1.1中先介绍一下旅行商问题.

表 1: 近十几年 MCM 比赛中用到启发式算法的特等奖论文统计

年份题号	题目	特等奖论文数
2003MCM-B	Gamma 刀治疗方案	2
2006MCM-A	灌溉洒水器的安置和移动	2
2006MCM-B	通过机场的轮椅	1
2007MCM-B	飞机座位方案	1
2008MCM-B	建立数独拼图游戏	1
2009MCM-A	交通环岛的设计	1
2011MCM-A	滑雪赛道优化设计	1
2014MCM-B	最佳大学体育教练	1

旅行商问题

旅行商问题, 即 TSP 问题（Travelling Salesman Problem）又译为旅行推销员问题, 货郎担问题, 是数学领域中著名问题之一. 假设有一个旅行商人要拜访 n 个城市, 他必须选择所要走的路径, 路径的限制是每个城市只能拜访一次, 而且最后要回到原来出发的城市. 路径的选择目标是要求得的路径路程为所有路径之中的最小值. 旅行商问题也是数图论中最著名的问题之一, 即“已给一个 n 个点的完全图, 每条边都有一个长度, 求总长度最短的经过每个顶点正好一次的封闭回路”. 该问题可以被证明具有 NP 计算复杂性, 迄今为止, 这类问题中没有一个找到有效算法. 如果用枚举的算法 [1], 并假设计算机枚举含 24 个城市的 TSP 问题需要 1 秒, 则对于不同城市数量的 TSP 问题的求解时间见表2. 虽然枚举的算法能给最全局最优解, 但随着问题规模的

表 2: TSP 问题枚举的算法所需时间

城市数量	24	25	26	27	28	29	30	34
计算时间	1s	24s	10m	4.3h	4.9d	136.5d	10.8y	∞

增大, TSP 问题的枚举算法计算开销也急剧增加, 甚至已经让人无法忍受. 表3给出了 34 个省会/直辖市的经纬度, 图1是相应的中国地图及模拟退火算法给出的结果, 若以中国 34 个省会作

表 3: 中国 34 个省会城市/直辖市的经纬度

编号	1	2	3	4	5	6	7	8	9
经度	39.54	31.14	39.09	29.32	45.45	43.52	41.50	40.49	38.02
纬度	116.28	121.29	117.11	106.32	126.41	125.19	123.24	111.48	114.28
编号	10	11	12	13	14	15	16	17	18
经度	37.52	36.38	34.48	34.16	36.03	38.20	36.38	43.48	31.51
纬度	112.34	117.00	113.42	108.54	103.49	106.16	101.45	87.36	117.18
编号	19	20	21	22	23	24	25	26	27
经度	32.02	30.14	28.11	28.41	30.37	30.39	26.35	26.05	23.08
纬度	118.50	120.09	113.00	115.52	114.21	104.05	106.42	119.18	113.15
编号	28	29	30	31	32	33	34		
经度	20.02	22.48	25.00	29.39	22.18	22.14	25.03		
纬度	110.20	108.20	102.41	90.08	114.10	113.35	121.31		

为旅行商问题, 枚举算法几乎已经无法计算. 在2.2和3.2中, 将以中国 34 个省会作为旅行商问题,

[1] TSP 问题枚举的算法复杂度

- 以第一个城市为始终点, 计算任意一条路径 $[1, i_2, \cdots, i_n, 1]$ 的长度的基本运算为两两城市间距离求和, 基本操作次数为 n. 路径的条数为 $(n-1)!$. 求和运算的总次数为 $(n-1)! \times n = n!$.
- 比较所有路径以得到最短路径, 需要比较的次数为 $(n-1)$.

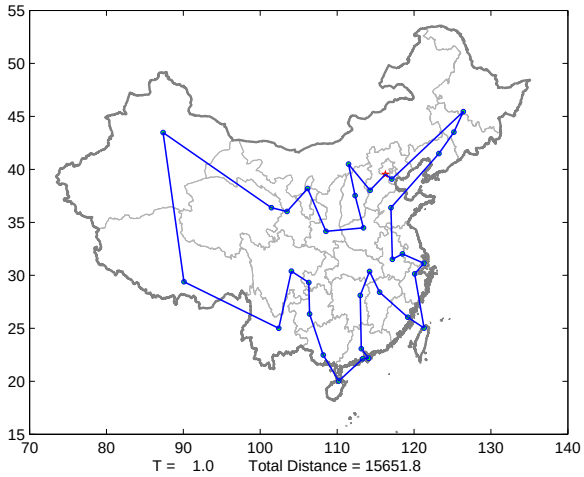

图 1: 中国 34 个省会城市

分别介绍模拟退火算法和遗传算法的 matlab 程序实现.

模拟退火算法

算法介绍

模拟退火是一种通用概率算法, 用来在固定时间内寻求在一个大的搜寻空间内找到的最优解. 模拟退火是 S. Kirkpatrick, C. D. Gelatt 和 M. P. Vecchi 在 1983 年所发明. 而 V. Cerny 在 1985 年也独立发明此算法 [2]. 模拟退火来自冶金学的专有名词退火. 统计力学表明材料中粒子的不同结构对应于粒子的不同能量水平. 在高温条件下, 粒子的能量较高, 可以自由运动和重新排列. 在低温条件下, 粒子能量较低.

退火是将金属材料加热后再经特定速率缓慢地冷却 (这个过程被称为退火), 粒子就可以在每个温度下达到热平衡, 目的是增大晶粒的体积, 并且减少晶格中的缺陷. 金属材料中的原子原来会停留在使内能有局部最小值的位置, 加热使能量变大, 原子会离开原来位置, 而随机在其他位置中移动. 退火冷却时速度较慢, 使得原子有较多可能可以找到内能比原先更低的位置, 最终形成处于低能状态的晶体. 图2是物理退火示意图.

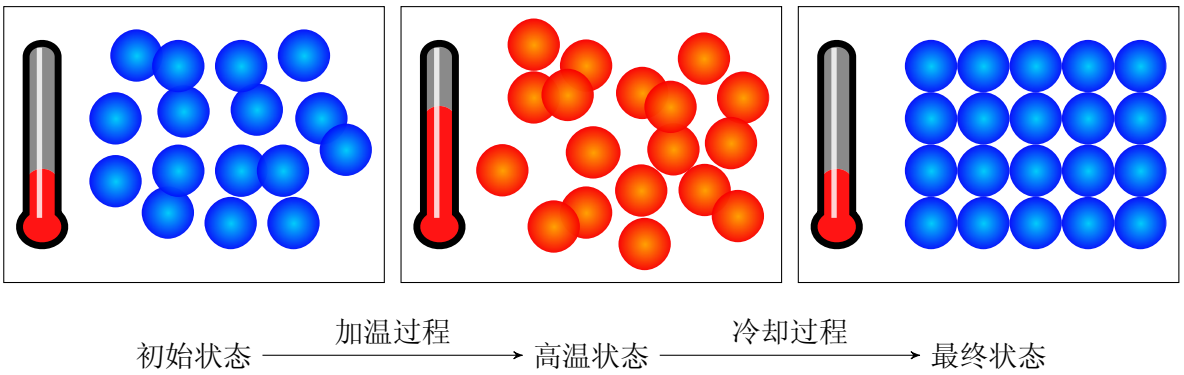

图 2: 物理退火过程

如果用粒子的能量定义金属材料的状态, Metropolis 算法用一个简单的数学模型描述了退火过程. 假设材料处于状态 s_i 时的能量为 $E(s_i)$, 那么材料在温度 T 下从状态进 s_i 转变为状态 s_j

概率为:

$$p(s_i \to s_j) = \begin{cases} 1 & E(s_i) \le E(s_j) \\ \exp\left\{\dfrac{E(s_i) - E(s_j)}{K_B T}\right\} & E(s_i) > E(s_j) \end{cases} \tag{1}$$

其中 K_B 为波尔兹曼常数. 式 (1) 表明如果 $E(s_i) \le E(s_j)$, 则接受状态的转变, 否则以概率 $\mathrm{e}^{[E(s_i)-E(s_j)]/(K_B T)}$ 接受状态的转变. 可以证明当温度降至很低时, 材料会以很大概率进入最小能量状态 [6].

假定我们要解决的问题是一个寻找最小值的优化问题. 考虑这样一个组合优化问题: 优化函数为 $F : x \to R^+$, 其中 $x \in S$, 它表示优化问题的一个可行解, $R^+ = \{y | y \in R, y > 0\}$, S 表示函数的定义域. $N(x) \subseteq S$ 表示的一个邻域集合. 将物理学中退火的思想应用于优化问题就可以得到模拟退火寻优方法. 表4为模拟退火算法与物理退火过程的对应关系. 首先给定一个初始温

表 4: 模拟退火算法与物理退火过程的对应

模拟退火	物理退火
解	粒子状态
目标函数	能量
最优解	能量最低态
设定初温	加温过程
扰动	热涨落
Metropolis 采样过程	热平衡, 粒子状态满足波尔兹曼分布
控制参数的下降	冷却

度 T_0 和该优化问题的一个初始解 x_0, 并由生成下一个解 $x' \in N(x_0)$, 是否接受 x' 作为一个新解依赖于下面概率:

$$p(x_0 \to x') = \begin{cases} 1 & f(x') \le f(x_0) \\ \exp\left\{\dfrac{f(x') - f(x_0)}{T_0}\right\} & f(x') > f(x_0) \end{cases} \tag{2}$$

换句话说, 如果生成的解 x' 的函数值比前一个解的函数值更小, 则接受 x' 作为一个新解. 否则以概率接受 x' 作为一个新解. 泛泛地说, 对于某一个温度 T_i 和该优化问题的一个解 x_k, 可以生成 x'. 接受 x' 作为下一个新解的概率为:

$$p(x_{k+1} \to x') = \begin{cases} 1 & f(x') \le f(x_k) \\ \exp\left\{\dfrac{f(x') - f(x_k)}{T_0}\right\} & f(x') > f(x_k) \end{cases} \tag{3}$$

在温度 T_i 下, 经过很多次的转移之后, 降低温度 T_i, 得到 $T_{i+1} < T_i$. 在 T_{i+1} 下重复上述过程. 因此整个优化过程就是不断寻找新解和缓慢降温的交替过程. 最终的解是对该问题寻优的结果. 我们注意到, 在每个 T_i 下, 所得到的一个新状态 x_{k+1} 完全依赖于前一个状态 x_k, 可以和前面的状态无关, 因此这是一个马尔可夫过程. 使用马尔可夫过程对上述模拟退火的步骤进行分析, 结果表明: 从任何一个状态 x_k 生成 x' 的概率, 在 $N(x_k)$ 中是均匀分布的, 且新状态被接受的概率满足式 (5), 那么经过有限次的转换, 在温度 T_i 下的平衡态 x_k 的分布由下式给出:

$$p_k(T_i) = \frac{\mathrm{e}^{-f(x_i)/T}}{\sum_{j \in S} \mathrm{e}^{-f(x_i)/T_i}} \tag{4}$$

当温度降为 0 时, x_k 的分布为:

$$p_k(0) = \begin{cases} 1/|S_{\min}| & x_k \in S_{\min} \\ 0 & \text{其它} \end{cases} \tag{5}$$

并且 $\sum_{x_k \in S_{\min}} P_k(0) = 1$. 这说明如果温度下降十分缓慢, 而在每个温度都有足够多次的状态转移, 使之在每一个温度下达到热平衡, 则全局最优解将以概率 1 被找到. 因此模拟退火算法所得解依概率收敛到全局最优解.

模拟退火算法新解的产生和接受可分为以下步骤:

- 由一个产生函数从当前解产生一个位于解空间的新解; 为便于后续的计算和接受, 减少算法耗时, 通常选择由当前新解经过简单地变换即可产生新解的方法, 如对构成新解的全部或部分元素进行置换, 互换等. 注意到产生新解的变换方法决定了当前新解的邻域结构, 因而对冷却进度表的选取有一定的影响.

- 计算与新解所对应的目标函数差. 因为目标函数差仅由变换部分产生, 所以目标函数差的计算最好按增量计算. 事实表明, 对大多数应用而言, 这是计算目标函数差的最快方法.

- 判断新解是否被接受, 判断的依据是一个接受准则, 最常用的接受准则是 Metropolis 准则: 若 $\Delta t < 0$ 则接受 S 作为新的当前解 S, 否则以概率 $\exp(-\Delta t / T)$ 接受 S 作为新的当前解 S.

- 当新解被确定接受时, 用新解代替当前解, 这只需将当前解中对应于产生新解时的变换部分予以实现, 同时修正目标函数值即可. 此时, 当前解实现了一次迭代. 可在此基础上开始下一轮试验. 而当新解被判定为舍弃时, 则在原当前解的基础上继续下一轮试验.

模拟退火算法与初始值无关, 算法求得的解与初始解状态 S（是算法迭代的起点）无关；模拟退火算法具有渐近收敛性, 已在理论上被证明是一种以概率 1 收敛于全局最优解的全局优化算法；模拟退火算法具有并行性.

对于特定的问题, 在应用和设计模拟退火算法过程中应注意以下问题

- 初始解的生成: 通常是以一个随机解作为初始解. 并保证理论上能够生成解空间中任意的解. 也可以是一个经挑选过的较好的解, 这种情况下, 初始温度应当设置的较低. 初始解不宜 “太好”, 否则很难从这个解的邻域跳出.

- 邻解生成函数: 邻解生成函数应尽可能保证产生的侯选解能够遍布解空间. 邻域应尽可能的小, 能够在少量循环步中允分探测, 但每次的改变不应该引起太大的变化.

- 确定初始温度: 初始温度应该设置的尽可能的高, 以确保最终解不受初始解影响. 但过高又会增加计算时间. 在正式开始退火算法前, 可进行一个升温过程确定初始温度: 逐渐增加温度, 直到所有的尝试运动都被接受, 将此时的温度设置为初始温度.

- 确定等温步数: 等温步数也称 Metropolis 抽样稳定准则, 用于决定在各温度下产生候选解的数目. 通常取决于解空间和邻域的大小. 等温过程是为了让系统达到平衡, 因此可通过检验目标函数的均值是否稳定 (或连续若干步的目标值变化较小) 来确定等温步数. 等温步数受温度的影响. 高温时, 等温步数可以较小, 温度较小时, 等温步数要大. 随着温度的降低, 增加等温步数. 有时为了考虑方便, 也可直接按一定的步数抽样.

- 确定降温方式: 理论上, 降温过程要足够缓慢, 要使得在每一温度下达到热平衡. 但在计算机实现中, 如果降温速度过缓, 所得到的解的性能会较为令人满意, 但是算法会太慢, 相对于简单的搜索算法不具有明显优势. 如果降温速度过快, 很可能最终得不到全局最优解. 因此使用时要综合考虑解的性能和算法速度, 在两者之间采取一种折衷.

程序实现

在2.1中已经简单的介绍了模拟退火算法主要思想. 在本节中以1.1中的 TSP 问题为例, 详细分析其 MatLab 程序实现. 求解 TSP 问题的模拟退火算法描述如下:

- 解空间: 如果我们按照表3用整数 1-34 对每个城市进行编号, 解空间 S 表示为 $\{1,2,\cdots,34\}$ 的所有固定起点和终点的循环排列集合. 任何一种 $\{1,2,\cdots,34\}$ 的循环排列

$$X=\{(x_1,x_2,\cdots,x_{34})\mid x_i\in[1,2,\cdots,34],\ x_i\neq x_j\}$$

  都是本问题的一个解. 其中 x_i 表示第 i 次访问的城市编号.

- 目标函数: TSP 问题的目标函数 (或称代价函数) 为路径长度, TSP 问题的目标为最小化路径长度, 即

$$\min d(x_1,x_{34})+\sum_{i=1}^{33}d(x_i,x_{i+1})$$

  其中 $d(x_i,x_{i+1})$ 表示编号为 x_i 和编号为 x_{i+1} 的两城市间的距离.

- 生成邻解: 从 $[1,2,\cdots,34]$ 中随机抽取两个数 i, j, 且有 $i<j$, 将两个城市间的子路径逆向排序顺序生成新解

$$X'\leftarrow(x_1,x_2,\cdots,x_j,x_{j-1},\cdots,x_{i+1},x_i,\cdots,x_{34})$$

  在程序中还给出了生成邻解另一种方式 '对调'.

- 降温方式: 利用选定的降温系数 α 进行降温即, 每执行 100 次循环降温一次: $T\leftarrow\alpha T$ 得到新的温度,

- 结束条件: 用选定的终止温度 1.0, 若 $T<1.0$ 计算终止, 输出当前状态.

基于以上描述, 可用 MatLab 实现 TSP 问题的模拟退火算法, 模拟退火算法结果如图1所示. 对于 34 个城市, 可以定义一个结构数组city, 结构数组长度为 34, 对于任何一个城市i, 可以用city(i).lat和city(i).long分别访问其经纬度. 比如对于北京 (见表3标号为 1 的城市) 有city(1).lat = 39.54, city(1).long= 116.28. 下面给出主程序的主要代码:

```
numberofcities = length(city);          % 程序的数量
dis = distancematrix(city);             % 距离矩阵 d_ij = dis(i,j)
temperature = 1000;                     % 初始温度
cooling_rate = 0.94;                    % 降温速度

route = randperm(numberofcities);       % 初始化路径
previous_distance = totaldistance(route,dis); % 计算路径的长度
temperature_iterations = 1;             % 用于计算恒温步数

while 1.0 < temperature
    temp_route = perturb(route,'reverse');      % 生成新解(邻解)
    current_distance = totaldistance(temp_route, dis); % 计算新解的距离
    diff = current_distance - previous_distance;       % 代价函数差

    % Metropolis 准则
    if (diff < 0) || (rand < exp(-diff/(temperature))) % 接受新解的条件
        route = temp_route;
        previous_distance = current_distance;

        temperature_iterations = temperature_iterations + 1;
    end

    % 每100步降一次温
    if temperature_iterations ≥ 100
       temperature = cooling_rate*temperature; % 降温方式: T = α·T
       temperature_iterations = 0;
```

```
    end

end
```

以上程序中调用了函数distancematrix, totaldistance, perturb, 下面我们逐一介绍.

函数distancematrix用来求得任意两城市间的球面距离矩阵dis. 由于城市的坐标由经度, 纬度表示, 两城市间的距离需要用球面距离来表示, 而 MatLab 中自带的函数 distance 可用来求取球面上两点的球面距离. 函数distancematrix的代码如下:

```
function dis = distancematrix(city)
numberofcities = length(city);
R = 6378.137; % 地球半径
for i = 1:numberofcities
    for j = i+1:numberofcities
        dis(i,j) = distance(city(i).lat, city(i).long, ...
                            city(j).lat, city(j).long, R);
        dis(j,i) = dis(i,j);
    end
end
```

函数distancematrix返回距离矩阵dis, 其中dis(i,j)表示第 i 个城市到第 j 个城市的距离.

函数totaldistance用来计算一条路径的总长度. 如果用一个向量route来表示一条路径 (问题的解), 向量route是 $\{1,2,\cdots,34\}$ 的某种循环排列. 可以定义以下函数totaldistance来非方便和快速的求取一条路径的总长度.

```
function d = totaldistance(route, dis)
d = dis(route(end),route(1));        % 形成圈
for k = 1:length(route)-1
    i = route(k);
    j = route(k+1);
    d = d + dis(i,j);                % d = dis(1,34) + sum_{i=1}^{34} dis(i,i+1)
end
```

其中距离矩阵dis是由函数distancematrix求得的. 函数totaldistance返回路径route的总长度 d, TSP 问题的目标就是 $\min d$.

函数perturb用来从当前解产生一个位于解空间的新解. 函数perturb的代码如下:

```
function route = perturb(route, method)
numbercities = length(route);        % 城市数量
i = randsample(numbercities,1);      % 随机整数满足 i ∈ [1,2,...,numbercities]
j = randsample(numbercities,1);      % 随机整数满足 j ∈ [1,2,...,numbercities]
switch method
    case 'reverse'                   % 逆序
        citymin = min(i,j);
        citymax = max(i,j);
        route(citymin:citymax) = route(citymax:-1:citymin);
    case 'swap'                      % 对调
        route([i, j]) = route([j, i]);
end
```

函数perturb定义了两种生成邻解的方式, 一种是 '逆序', 另一种为 '对调'. 当参数method为'reverse'时, 生成邻解的方式为 '逆序', 随机选择路径中的两个城市, 并将两个城市间的子路径逆向排序顺序生成新解; 当参数method为'swap'时, 生成邻解的方式为 '对调', 随机选择路径中的两个城市, 并将两个城市间的子路径逆向排序顺序生成新解.

遗传算法

算法介绍

遗传算法（Genetic Algorithm）是由美国的 J. Holland 教授于 1975 年首先提出的, 是计算数学中用于解决最优化的搜索算法. 遗传算法最初是借鉴了进化生物学中的一些现象而发展起来的, 这些现象包括遗传, 突变, 自然选择 (适者生存，优胜劣汰遗传机制) 以及杂交等. 在遗传算法中, 问题域中的可能解被看作是群体的个体. 对于一个最优化问题, 一定数量的候选解 (称为个体) 的抽象表示 (称为染色体) 的种群向更好的解进化. 传统上, 解用二进制将个体编码成符号串形式 (即 0 和 1 的串), 但也可以用其他表示方法. 进化从完全随机个体的种群开始, 之后一代一代发生. 在每一代中, 整个种群的适应度被评价, 基于它们的适应度, 从当前种群中随机地择优选择多个个体. 通过杂交和突变产生新的生命种群, 该种群在算法的下一次迭代中成为当前种群. 从而不断得到更优的群体, 同时搜索优化群体中的最优个体, 求得满足要求的最优解.

遗传算法可分为以下基本步骤:

1. 初始化: 初始化进化代数计数器 $t \leftarrow 0$, 最大进化代数 T. 随机生成 M 个个体作为初始体 $P(t)$. 始群体 P
2. 个体评价: 计算 $P(t)$ 中各个个体的适应度值.
3. 选择运算: 将选择算子作用于群体.
4. 交叉运算: 将交叉算子作用于群体.
5. 变异运算: 将变异算子作用于群体, 并通过以上运算得到下一代群体 $P(t+1)$.
6. 终止条件: 如果 $t \leq T$, 则 $t \leftarrow t+1$ 并跳转到第2步; 否则输出 $P(t)$ 中的最优解.

基本遗传算法的五个组成部分:

- 编码: 正如研究生物遗传是从染色体着手, 而染色体则是由基因排成的串, 遗传算法中, 首要问题就是如何通过某种编码机制把对象抽象为由特定符号按一定顺序排成的串 (解的形式). 编码影响到交叉, 变异等运算, 很大程度上决定了遗传进化的效率. 在基本遗传算法 (SGA) 使用, 二进制串进行编码, 每个基因值为符号 0 和 1 所组成的二值制数. 针对不同的问题, 可以适当选择其它编码形式, 如格雷编码, 实数编码, 符号编码.
- 适应度函数: 适应度函数也称评价函数, 是根据目标函数确定的用于区分群体中个体好坏的标准. 适应度函数是遗传算法进化过程的驱动力, 也是对个体的优胜劣汰的唯一依据. 它的设计应结合求解问题本身的要求而定. 一般情况下适应度是非负的, 并且总是希望适应度越大越好 (适应度值与解的优劣成反比例). 通常适应度函数可以由目标函数直接或间接改造得到. 比如, 目标函数, 或目标函数的倒数/相反数经常被直接用作适应度函数. 适应度函数不应过于复杂, 以便于计算机的快速计算.
- 选择算子: 选择运算的使用是对个体进行优胜劣汰: 从父代群体中选取一些适应度高个体, 遗传到下一代群体. 适应度高的个体被遗传到下一代群体中的概率大; 适应度低的个体, 被遗传到下一代群体中的概率小. 基本遗传算法中选择算子采用轮盘赌选择方法, 图3为轮盘赌示意图. 轮盘赌又称比例选择算子, 个体 i 被选中的概率 p_i 与其适应度成正比: $p_i = f_i / \sum_{j=1}^{N} f_j$. 个体的适应度值越大，被选中的概率就越高，直接体现了 “适者生存” 这一自然选择原理. 遗传算法中也常用其它选择算子, 如两两竞争 (从父代中随机地选取两个个体, 比较适应值, 保存优秀个体, 淘汰较差的个体) 等.
- 交叉算子: 交叉运算是指对两个相互配对的染色体依据交叉概率按某种方式相互交换其部分基因, 从而形成两个新的个体. 交叉运算是遗传算法区别于其他进化算法的重要特征, 它在遗传算法中起关键作用, 是产生新个体的主要方法. 遗传算法中, 交叉算子可以是单点交叉, 也可以是多点交叉, 如图4所示. 基本遗传算法中交叉算子采用单点交叉算子.

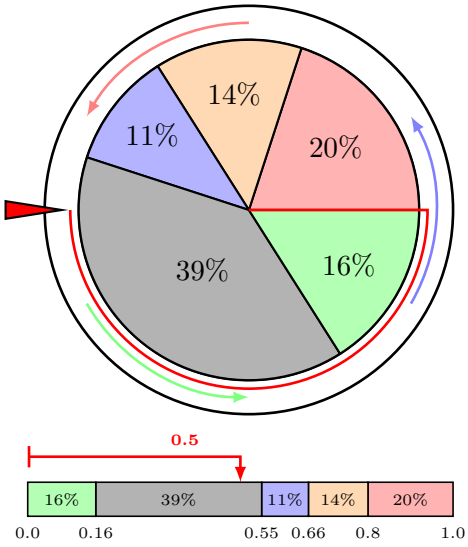

图 3: 轮盘赌示意图

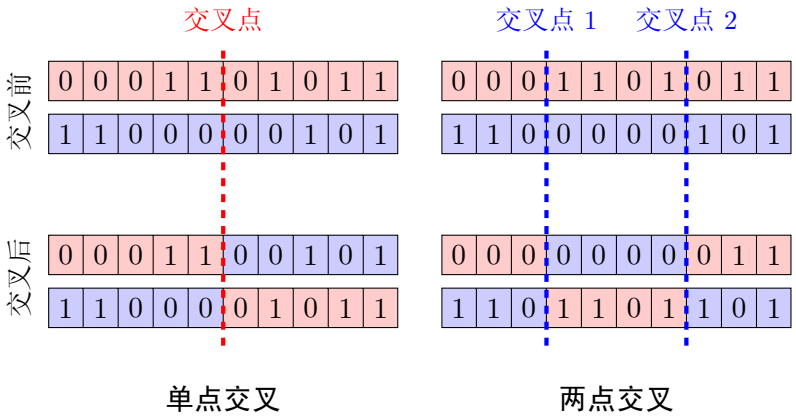

图 4: 单点交叉和两点交叉算子示意图

- 变异算子: 变异操作对群体中的个体的某些基因座上的基因值作变动, 模拟生物在繁殖过程, 新产生的染色体中的基因会以一定的概率出错. 变异运算是产生新个体的辅助方法, 决定遗传算法的局部搜索能力, 保持种群多样性. 交叉运算和变异运算的相互配合, 共同完成对搜索空间的全局搜索和局部搜索. 遗传算法中, 变异算子可以是基本位变异, 也可以是换位变异, 如图5所示. 基本遗传算法中变异算子采用基本位变异算子. 基本位变异算

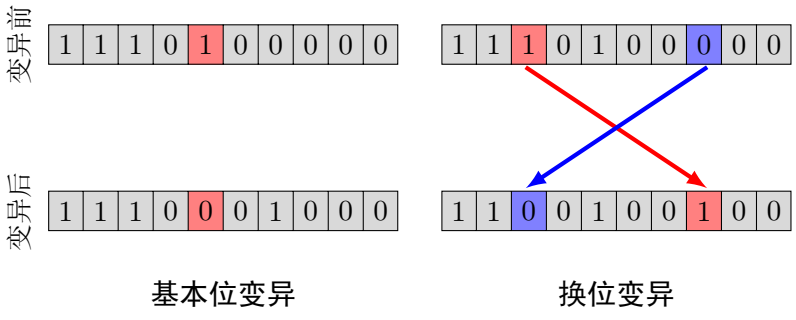

图 5: 轮盘赌示意图

子是指对个体编码串随机指定的某一位或某几位基因作变异运算. 对于二进制编码符号串

所表示的个体, 若需要进行变异操作的某一基因座上的原有基因值为 0, 则将其变为 1; 反之, 若原有基因值为 1, 则将其变为 0.

程序实现

在3.1中已经简单的介绍了模拟退火算法主要思想. 在本节中以1.1中的 TSP 问题为例, 详细分析其 MatLab 程序实现. 求解 TSP 问题的遗传算法描述如下:

- 编码: 如果我们按照表3用整数 1-34 对每个城市进行编号, 则任何一种 $\{1,2,\cdots,34\}$ 的循环排列都是本问题的一个解. 对于 TSP 问题, 选用符号编码, 直接用 $\{1,2,\cdots,34\}$ 的循环排列作为解的符号串形式.
- 适应度函数: TSP 问题的目标函数为路径长度 d, 考虑到适应度函数的特点, 可将路径长度的倒数 $1/d$ 作为适应度函数.
- 选择算子: 轮盘赌.
- 交叉算子: 采用两点交差, 采用符号编码的解形式在交叉后的必需要保证生成的新解也是一个合法的 $\{1,2,\cdots,34\}$ 的循环排列, 不能出现重复的城市编号. 因此在随机选取两个交叉点 i 和 j 后 $(i<j)$ 并进行交叉 (交换双亲第 i 个到第 j 个符号串) 后, 还需要一个额外的操作以消除重复的城市编号. 如果交换后, 后代 1 到 i 中的编号 $x_k, 1<k<i$ 与子串 i 到 j 中的编号 $x_m, i\le m\le j$ 重复，则用交差前的 x_m 来替换到 x_k, 按照此法对 i 到 j 中的编号依次作这样的操作, 直到新解中不出现编号的重复为止. 图6是这种交叉运算的结果示意图.

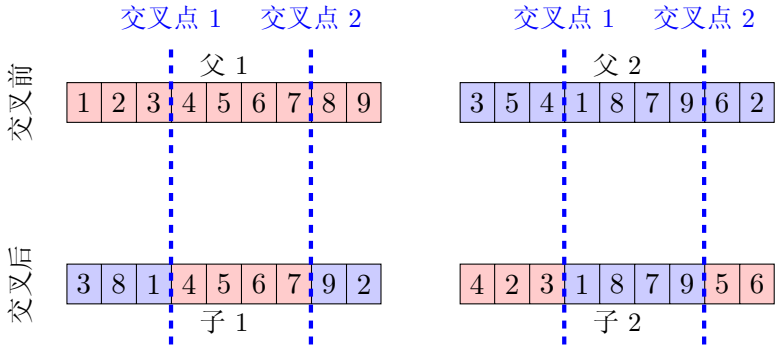

图 6: TSP 问题的两点交叉算子

- 变异算子: 在 TSP 问题中, 单点基本位变异会造成新解中出现编号的重复. 因此这里的变异算子采用对调, 滑移和逆序三种不会引起编号重复的变异算子.

基于以上描述, 可用 MatLab 实现 TSP 问题的遗传算法, 图7为 MatLab 实现的中国 34 个省会城市 TSP 问题的遗传算法求解结果. 在遗传算法的 MatLab 程序中, 解的形式, 任意两城市间的球面距离矩阵和路径的总长度的计算都与2.2节中的退火方法一致, 相同的内容这里不再复述. 下面给出主程序的主要代码:

```
popSize = 50;              % 种群规模
max_generation = 3000; % 初始化最大种群代数
Pmutation = 0.16;        % 变异概率
for i = 1:popSize        % 初始化种群
    pop(i,:) = randperm(numberofcities);
end
for generation = 1:max_generation
    fitness = 1/totaldistance(pop,dis); % 计算距离(适应度)
    [maxfit, bestID] = max(fitness);
    bestPop = pop(bestID, :);              % 找出精英
    pop = select(pop,fitness);             % 选择操作
    pop = crossover(pop);                  % 交叉操作
```

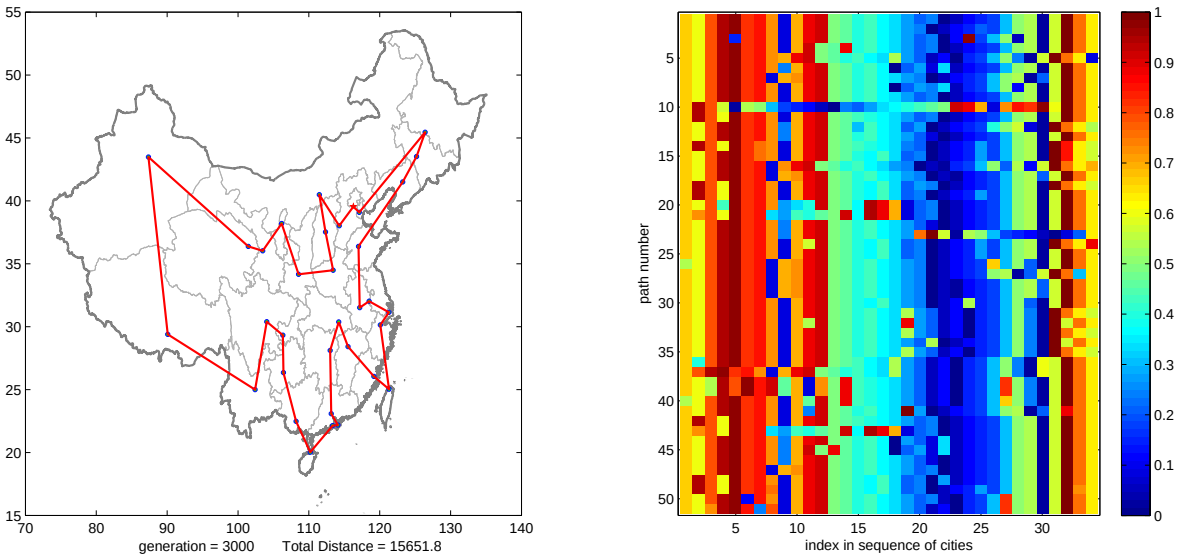

图 7: 中国 34 个省会城市 TSP 问题的遗传算法解

```
    pop = mutate(pop,Pmutation);          % 变异操作
    pop = [bestPop; pop];                 % 精英保护
end

popDist = total_distance(pop,dis);        %计算距离(适应度)
[minDist, index] = min(popDist);
optRoute = pop(index,:);                  % 找出最短距离
```

以上程序中第10找出当前代中最好的解, 并通过14行将这个最好的解不作任何改变操作直接放到下一代的种群中, 这样做是为了使得下一代的种群的最优解至少不会比上一代差. 此外, 上述主程序中还调用了函数select, crossover和mutate, 下面我们逐一介绍.

函数select是用来进行选择操作, 在函数select中, 我们提供两种选择算子: 轮盘赌和两两竞争. 函数select的代码如下:

```
function popselected = select(pop, fitness, nselected, method)
popSize = size(pop,1);

switch method
    case 'roulette'                       % 轮盘赌
        p=fitness/sum(fitness);           % 选中概率
        cump=cumsum(p);                   % 概率累加
        % 利用插值: yi = 线性插值(x, y, xi)
        I = interp1([0 ...
            cump],1:(popSize+1),rand(1,nselected),'linear');
        I = floor(I);

    case 'competition' % 两两竞争
        i1 = randsample(popSize, nselected);
        i1 = randsample(popSize, nselected);
        I = i1.*( fitness(i1)≥fitness(i2) ) + ...
            i2.*( fitness(i1)< fitness(i2) );
end

popselected=pop(I,:);
```

以上代码中第6行得到规一化的适应度. 在轮盘赌中, 各染色体规一化的适应度即为其被选中的概率. 第7是对选中概率进行累加. 如果种群的规模为 5, 种群中各染色体规一化的适应度为 $p = [0.16, 0.39, 0.11, 0.14, 0.20]$, 则经过累加得到 $cump = [0.16, 0.55, 0.66, 0.8, 1.0]$. 则可以通过生成随机来决定选择哪些染色体, 若生成的随机 $rand \in [0, 0.16)$, 则第一个染色体被选中; 若生成的随机 $rand \in [0.66, 0.8)$, 则第四个染色体被选中. 第9行利用插值运算实现轮盘赌. 以上程序中还提供了两两竞争的选择算子, 这里不再详述.

函数crossover是用来进行交叉操作, 由于需要消除交叉中出现的城市编号重得, 交叉操作的代码稍微有点复杂. 函数crossover的代码如下:

```
1  function children = crossover(parents)
2  [popSize, numberofcities] = size(parents);
3  children = parents;                          % 初始化子代
4
5  for i = 1:2:popSize
6      parent1 = parents(i+0,:);  child1 = parent1;
7      parent2 = parents(i+1,:);  child2 = parent2;
8      InsertPoints = sort(randsample(numberofcities, 2)); % 交叉点
9      for j = InsertPoints(1):InsertPoints(2)
10         if parent1(j) ≠ parent2(j)      % 如果对应位置不重复
11             child1(child1==parent2(j)) = child1(j);
12             child1(j) = parent2(j);
13
14             child2(child2==parent1(j)) = child2(j);
15             child2(j) = parent1(j);
16         end
17     end
18     children(i+0,:)=child1;      children(i+1,:)=child2;
19 end
```

以上程序中第9行中生成两个随机的交叉点. 第10至17行的循环对两交叉点间的城市编号依次进行互换和消重. 对于子代 1, 由第11行消重, 第12行互换以达到交叉的目的. 第11行通过child1==parent2(j)找出交差后会出现重复的位置, 并替换为child1(j), 那么第12行交叉后child1就不会出现重复的城市编码. 对于子代 2 类似.

函数mutation是用来进行变异操作, 在函数crossover提供了三种变异算子:“对调”, “滑移” 和“逆序”. 每一次变异都从三种变异算子中选择一种, 并且主要以 “逆序” 为主. 函数crossover的代码如下:

```
1  function children = mutation(parents, probmutation)
2  [popSize, numberofcities] = size(parents);
3  children = parents;                          % 初始化子代
4  for k=1:popSize
5      if rand < probmutation                   % 以一定概率变异
6          InsertPoints = randsample(numberofcities, 2);
7          I = min(InsertPoints);  J = max(InsertPoints); % 两交叉点: I<J
8          switch randsample(6, 1)              % 通过随机数判断使用哪种算子
9              case 1    % 对调
10               children(k,[I J]) = parents(k,[J I]);
11             case 2    % 滑移
12               children(k,[I:J]) = parents(k,[I+1:J I]);
13             otherwise % 逆序
14               children(k,[I:J]) = parents(k,[J:-1:I]);
15         end
16     end
17 end
```

以上程序通过switch语句来选择不同的变异算子. 第8行中randsample(6, 1)从 $[1, 2, \cdots, 6]$ 中随机

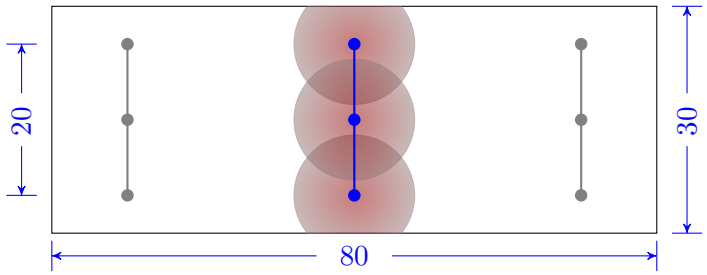

图 8: 灌溉喷洒系统的布置与移动问题

抽取出的一个数字为 1 则采用 “对调” 算子; 如果随机抽取出的数字为 2 则采用 “滑移” 算子; 否则当随机抽取出的数字在 $[3,4,5,6]$ 中, 则采用 “逆序” 算子.

应用举例

在数学建模中, 很多实际问题的解决最终归结为目标函数的优化. 通常情况下, 在有限的时间内求解这些实际问题的最优解并不容易, 甚至无法实现. 这类问题的求解通常有两种方式: 一种是转变为几种现行备选方案的比较, 另一种则是用启发式方法得到近似最优解. 在本节中, 我们列举一个退火算法在数学建模竞赛中的应用实例: “灌溉喷洒系统的布置与移动” 问题的求解.

灌溉喷洒系统的布置与移动 (MCM2006A)

灌溉喷洒系统的布置与移动 [7] 是 2006 年美国大学生数学建模竞赛 A 题, 原题翻译如下:

> 目前有很多种田间灌溉的技术. 从先进的滴灌系统到周期性的漫灌等各种技术. 用于较小农场的技术之一就是使用 “手动” 灌溉系统. 带有喷头的轻质铝管放置在田间, 定时用手移动它们以确保所有农田都能够得到充足的水. 这种灌溉系统比其他系统更加便宜, 更加容易管理、维护. 它们的使用非常灵活，可用于各种农田和农作物的灌溉. 其缺点是，每过一段时间, 就要花费很多时间和精力来移动和安装设备.
>
> 考虑到要使用这种灌溉系统, 怎样安装才能用最少的时间去灌溉一片 80m × 30m 米的农田? 为完成这项任务, 请求你们去寻求一种确定怎样灌溉这块矩形农田使得农场主管理、维护该灌溉系统所需要的时间最少. 这块农田上将使用一套管组. 你们需要确定喷头的数量以及喷头之间的距离, 同时还要给出一个移动管道, 包括需要把管道移动到什么位置的工作进度表.
>
> 一套管组由若干互相连接成直线形的管子组成. 每根管子的内壁直径为 10cm, 并带有一个内壁直径 0.6cm 的可旋转喷嘴. 把管子连接在一起, 其总长为 20m 长. 水源处的压力为 420KPa, 流量为每分钟 150L/min. 农田任何部分接受的水量不得超过每小时 0.75cm, 同时农田的每个部分每 4 天至少要接受 2cm 的水量. 尽可能均匀地使用洒水的总量.

该题与 03 年 gamma 刀问题 [8] 非常类似, 都可以转化为覆盖 (填充) 区域问题. 该题中要求制定一个移动管组的工作进度表, 这个进度表既包括移动到哪里, 也包括多长时间间隔移动一次. 对于前者, 我们只需假想一个具有多套 20m 管组的固定灌溉系统. 如果系统能很好的满足作物的需求, 即具有较高的分布均匀度, 那么这个问题中的管子移动方式则转变为将管子从固定灌溉系统中的一个管子的位置移到另一个管子的位置. 因此, 我们通过铺设一个具有多条 20m 管子的灌溉系统来确定管子的移动位置, 然后确定每个位置一次需要浇灌多长时间以及需要浇灌几轮. 本节仅详细讨论有关模拟退火算法求解该问题的部分, 讨论和求解过程主要参考了一篇特等奖参赛论文 [9].

该问题需要优化的目标主要是农田灌溉的均匀度. 灌溉喷洒系统移动的总次数和农田灌溉的均匀度都依赖于农田中每个位置的获水率. 若用 $S(\vec{x})$ 表示 $\vec{x}$ 位置的获水率, $\varphi(|\vec{x}-\vec{x_i}|)$ 表示位于 $\vec{x_i}$ 位置的喷头对 $\vec{x}$ 位置的获水率的贡献, 则 $S(\vec{x})$ 可表示为所有喷头对 $\vec{x}$ 位置获水率贡献的

叠加:

$$S(\vec{x}) = \sum_k \varphi(|\vec{x} - \vec{x_k}|) \tag{6}$$

显然农田中各区域的获水率 $S(\vec{x})$ 并不相同, 因此需要定义一个均匀度的概念来描述灌溉的效果. 这里采用 Christiansen 均匀系数:

$$CU = 100\left(1 - \frac{\sigma_S}{\langle S\rangle}\right) \tag{7}$$

其中 σ_S 和 $\langle S\rangle$ 分别表示农田中各区域的获水率 S 的标准差和均值. 对于获水率贡献的叠加及叠加后的均匀系数的求解, 详见代码3. 代码3定义了函数watersum, 能够根据管道的位置和角度直接计算出各区域的获水率S和相应的均匀度CU.

对于各位置获水率及相应的均匀度的计算, 都依赖于单个喷头的降水分布函数 $\varphi(|\vec{x} - \vec{x_i}|)$. 为了计算喷头喷洒范围, 我们需要计算喷头喷射出的水流速度 v_o. 如果水管上安置 n 个喷头, 根据 Bernoulli 方程及水的连续和不可压性可以推导 (具体推导过程见论文 [9]) 得

$$v_o = \sqrt{\left(\frac{J}{An}\right) + \frac{2(P_\mathrm{w} - P_\mathrm{a})}{\rho}} \tag{8}$$

其中 $J = 150\,\mathrm{L/min}$ 为水源处流量; $A = \pi(d/2)^2$, 其中 $d = 0.6\,\mathrm{cm}$ 为喷嘴内径; $P_\mathrm{w} = 420\,\mathrm{kPa}$ 为水压, $P_\mathrm{a} = 101\,\mathrm{kPa}$ 为外界大气压; ρ 为水的密度. 由水滴的运动方程可得喷管的喷射半径

$$R = \frac{1}{k}\ln\left(ktv_o\cos\theta + 1\right) \tag{9}$$

其中 $k = 0.2\,\mathrm{m}^{-1}$ 为空气阻力系数[2]; $t \approx 2v_o\sin\theta/g$ 为液滴在空中飞行的时间; $\theta = \pi/4$ 是喷嘴喷水的仰角. 假设单个喷头的降水分布函数是线性的, 则有

$$\varphi(r) = \begin{cases} h(1 - r/R) & r < R \\ 0 & r \geq R \end{cases} \tag{10}$$

其中 $h = 3J/(\pi nR^2)$ 由体积守恒确定. 以上通过求得喷嘴喷射初速度, 进而求得喷洒半径并最终确定线性降水分布函数的过程, 见代码2. 代码2定义的函数 sprinklerprofile 还给出另一种可供选择的指数型降水分布函数.

该问题的另一个优化目标是灌溉喷洒系统移动的总次数, 为了节约劳动力, 移动灌溉喷洒系统的总次数越小越好. 灌溉喷洒系统移动的总次数 = 灌溉喷洒系统布置的位置数 × 四天里灌溉的轮数. 令 S_min, S_max 分别为农田区域中最大和最小获水率. 由于该问题限制条件为: 农田任何部分接受的水量不得超过每小时 0.75cm, 同时农田的每个部分每 4 天至少要接受 2cm 的水量. 因此根据题义有

$$S_\mathrm{min}t \times \text{灌溉的轮数} = 2\ \mathrm{cm}, \qquad S_\mathrm{max}t = 0.75\ \mathrm{cm}$$

其中 t 为一次灌溉的时长. 由此可得灌溉的轮数

$$\text{灌溉的轮数} = \frac{2}{0.75}\frac{S_\mathrm{max}}{S_\mathrm{min}} \tag{11}$$

因此, 在已知各区域的获水率S的情况下, 容易求得灌溉的轮数和灌溉喷洒系统移动的总次数. 代码5定义了函数totalmoves来根据各区域获水率S计算灌溉喷洒系统移动的总次数n.

模拟退火算法中一个较为关键的步骤是在当前解的基础上生成邻解. 对于确定的喷头数量和管子位置数, 则该问题的解为管子的位置和角度. 如图9所示, 管子上的两个喷头位置 (x_1, y_1) 和 (x_2, y_2) 可由管道的位置 (x_0, y_0) 和角度 θ 确定:

$$(x_1, y_1) = \left(x_0 + \frac{\ell}{2}\cos\theta,\, y_0 + \frac{\ell}{2}\sin\theta\right), \quad (x_2, y_2) = \left(x_0 - \frac{\ell}{2}\cos\theta,\, y_0 - \frac{\ell}{2}\sin\theta\right) \tag{12}$$

[2]论文 [9] 中 $k = 0.15\,\mathrm{m}^{-1}$, 这里调整为 $0.2\,\mathrm{m}^{-1}$

通过调整当前管道的位置和角度可以很容易地生成邻解. 例如图9经过适当的扰动, 生成了新的管道位置 (x_0', y_0') 和角度 θ':

$$x_0' = x_0 + S_p \cdot \zeta_x, \quad y_0' = y_0 + S_p \cdot \zeta_y \tag{13}$$

$$\theta' = \theta + S_a \cdot \zeta_a \tag{14}$$

其中, S_p 和 S_a 分别为位置和角度扰动的幅度, 且均为模拟退火算法中温度的减函数. ζ_x, ζ_y, ζ_a 为均匀分布于区间的 $[-0.5, 0.5]$ 随机数. 代码4定义了函数perturbe来实现邻解的生成. 为了保证

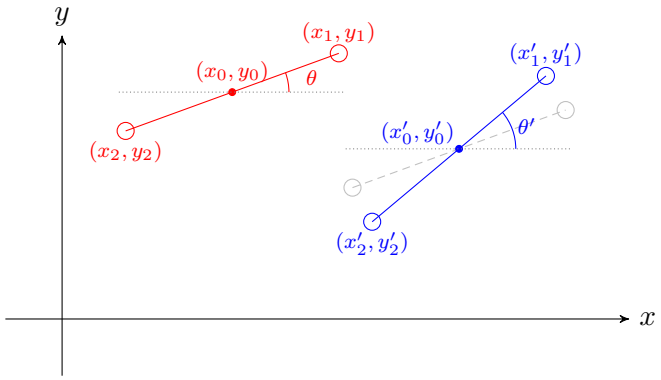

图 9: 求解灌溉问题的模拟退火算法扰动示意图

随机扰动后管道上的喷头不超出农田边界, 函数perturbe中对管道的位置和角度扰动后还进行了喷头位置的检查, 若扰动后有喷头位于农田之外, 则不接受扰动.

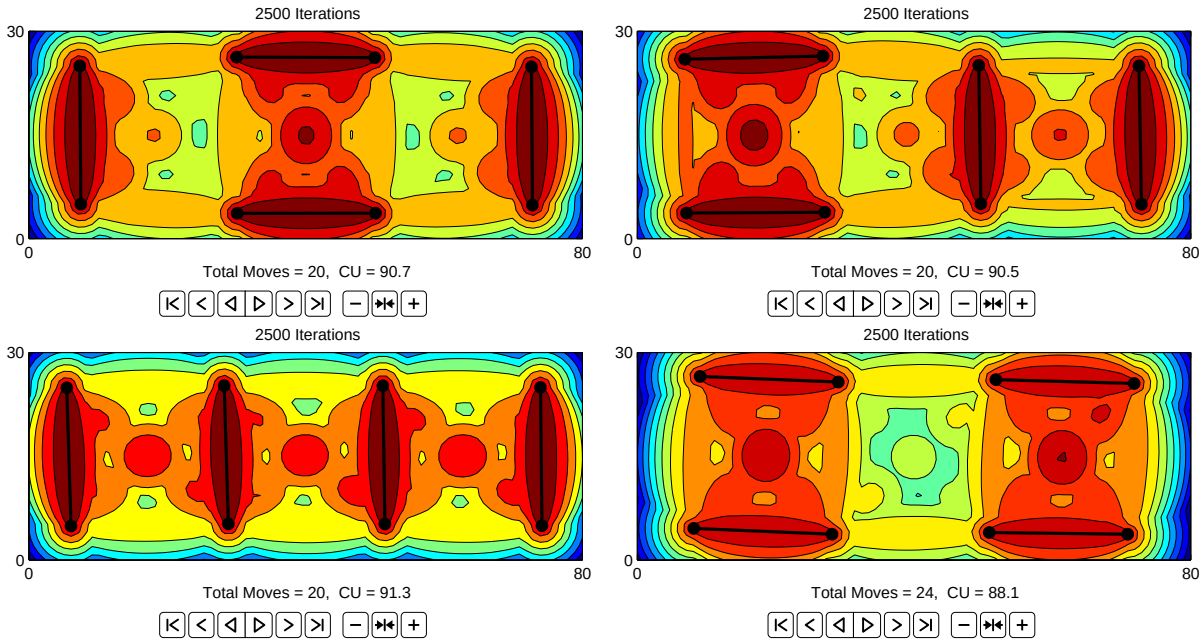

图 10: 灌溉喷洒系统的布置与移动问题的模拟退火算法解

代码 1: 模拟退火算法求解灌溉问题主程序 SAIrrigation

```
function [E, NumMoves] = SAIrrigation(NumHeads, NumSteps)

if nargin==0;
    NumHeads = 2;              % 管道上的喷头数量
    NumSteps = 4;              % 每轮移动次数(管道数量)
```

```
end

FieldLen = 80;                  % 农田的长度      [m      ]
FieldWid = 30;                  % 农田的宽度      [m      ]
PipeLen  = 20;                  % 管道的长度

Tmax = 1e-1;                    % 模拟退火初始温度
Tmin = 1e-5;                    % 模拟退火最终温度

kmax = 3000;                    % 降温的步数

% 初始化农田网络
nx = FieldLen;                  % X 方向格子数(分辨率)
ny = FieldWid;                  % Y 方向格子数(分辨率)
[field.x,field.y] = meshgrid(linspace(0,80,nx),linspace(0,30,ny));

%% 随机生成管道位置和角度 (初始解), 并等间距生成喷头在管道上的相对位置
for i = 1:NumSteps
   pipe(i).pos = PipeLen/2 + rand(1,2).*(field.size-PipeLen);
   pipe(i).ang = rand*2*pi;
end
HeadPos = linspace(-PipeLen/2, PipeLen/2, NumHeads);

%% 生成初始解的获水分布及相应的均匀度 (目标函数)
[S, E] = watersum(field, NumHeads, pipe, HeadPos);

% 计算移动总次数
NumMoves = totalmoves(S, NumSteps);

% 画出初始解的获水分布及管道和喷头方位
wateringplot(field, pipe, HeadPos, S, E, NumMoves, 0);


for k = 1:kmax
    %% 降低系统温度
    T = Tmax + (Tmin - Tmax)*k/kmax;

    %% 随机扰动并生成邻解
    [pipeNew] = perturb(pipe, HeadPos, T);

    %% 计算新解的获水分布及相应的均匀度 (目标函数)
    [Snew, Enew] = watersum(field, NumHeads, pipeNew, HeadPos);

    %% Metropolis 准则
    if Enew > E | rand < exp((Enew - E)/T)
        pipe = pipeNew;         % 接受新解
        E = Enew; S = Snew;     % 更新均匀度和获水分布

        % 计算移动总次数
        NumMoves = totalmoves(S, NumSteps);

        % 画出当前解的获水分布及管道和喷头方位
        wateringplot(field, pipe, HeadPos, S, E, NumMoves, k);
    end
end
```

代码 2: 单个喷嘴分布函数 (获水率) sprinklerprofile

```m
function p = sprinklerprofile(n, r, method)

if nargin == 2; method = 'linear'; end

g = 9.8;                        % 重力加速度      [m/s^2 ]
J = 150e-3/60;                  % 水源处流量      [m^3/s ]
d = 0.6e-2;                     % 喷嘴内径        [m     ]
A = pi * (d/2)^2;               % 水管截面积      [m^2   ]
Pw = 420e3;                     % 水源处压强      [Pa    ]
Pa = 101e3;                     % 外界大气压      [Pa    ]
rho = 10^3;                     % 水的密度        [kg/m^2]
k = 0.2;                        % 阻力系数        [m^-1  ]
theta = pi/4;                   % 喷嘴仰角

v = sqrt((J/(A*n))^2 + 2*(Pw-Pa)/rho); % 喷嘴出流速度    [m/s   ]
t = 2 * v*sin(theta) / g;              % 液滴飞行时间    [s     ]
R = (1/k)*log((k*t*v*cos(theta))+1);   % 喷嘴喷洒半径    [m     ]

switch method                   %% 两种分布函数: 线性和指数
    case 'linear'
        h = 3*J/(pi*n*R^2);     % 分布函数Y截距   [m/s   ]
        p = h*(1-r/R).*(r<R);   % 线性分布函数    [m/s   ]
    case 'exp'
        an = 4*J/(n*pi*R^2);    % 归一化因子      [m/s   ]
        p = an * exp(-r./R);    % 指数分布函数    [m/s   ]
end

p = p * 100 * 3600;             % 获水率          [cm/hr ]
```

代码 3: 多个喷嘴获水率叠加函数 watersum

```m
function [S, CU] = watersum(field, NumHeads, pipe, HeadPos)

%% 初始化各位置获水率为 0
S = zeros(size(field.x));

%% 叠加每根管上每个喷头对获水率的贡献
for i = 1:length(pipe)
    for j = 1:NumHeads
        % 计算第 i 个管上第 j 个喷头的位置
        x = pipe(i).pos(1) + cos(pipe(i).ang)*HeadPos(j);
        y = pipe(i).pos(2) + sin(pipe(i).ang)*HeadPos(j);

        % 计算任意位置到该喷头的距离
        r = sqrt( (field.x-x).^2 + (field.y-y).^2 );

        % 叠加该喷头的分布函数
        S = S + sprinklerprofile(NumHeads, r);
    end
end

%% 匀均度 = 100 * (1 - 标准差/均值)
CU = 100*( 1 -(std(S(:))/mean(S(:))) );
```

代码 4: 扰动函数 perturb

```m
function pipeNew = perturb(pipe, HeadPos, T)

```

```
scalePos = 60*T;                % 位置扰动范围
scaleAng = pi*T;                % 角度扰动范围
pipeNew = pipe;                 % 初始化扰动后的管道

for i = 1:length(pipe)
    % 扰动管道中心位置和方位角
    pipeNew(i).pos = pipe(i).pos + scalePos * [0.5-rand(1,2)];
    pipeNew(i).ang = pipe(i).ang + scaleAng * [0.5-rand(1,1)];

    % 计算喷头位置
    x = pipeNew(i).pos(1) + cos(pipeNew(i).ang) * HeadPos;
    y = pipeNew(i).pos(2) + sin(pipeNew(i).ang) * HeadPos;
    % 检查是否有喷头超出边界: 如果有, 则不接受扰动
    if any(x<0|x>80) | any(y<0|y>30)
        pipeNew(i) = pipe(i);
    end
end
```

代码 5: 总移动次数的计算函数 totalmoves

```
function n = totalmoves(S, NumSteps)

Smin = min(S(:));               % 最大获水量
Smax = max(S(:));               % 最小获水量

%% 需要灌溉的轮数 NumPasses 和总移动次数 n
NumPasses = 2*Smax/(Smin*0.75);
n= ceil(NumPasses)*NumSteps;
```

代码 6: 画图函数 wateringplot

```
function wateringplot(field, pipe, HeadPos, S, E, NumMoves, k)

%% 画出获水量云图
contourf(field.x,field.y,S);
hold on

%% 计算各管道上喷头的位置, 并画出含喷头的管道
for i = 1:length(pipe)
    x = pipe(i).pos(1) + cos(pipe(i).ang)*HeadPos;
    y = pipe(i).pos(2) + sin(pipe(i).ang)*HeadPos;
    plot(x,y,'-k.', 'LineWidth',2, 'MarkerSize', 25);
end
set(gca,'DataAspectRatio',[1,1,1],'XTick',[0,80],'YTick',[0,30]);

%% 输出移动总次数和灌溉均匀度
xlabel(sprintf('Total Moves = %d,  CU = %.1f',NumMoves, E));

%% 输出当前计算循环步数
title(sprintf('%5d Iterations',k));

drawnow
hold off
```

参考文献

[1] Tabu search, December 2014. Available at http://en.wikipedia.org/wiki/Tabu_search[Accessed December 20, 2014].

[2] Simulated annealing, December 2014. Available at http://en.wikipedia.org/wiki/Simulated_annealing[Accessed December 20, 2014].

[3] Genetic algorithm, December 2014. Available at http://en.wikipedia.org/wiki/Genetic_algorithm[Accessed December 20, 2014].

[4] Artificial neural network, December 2014. Available at http://en.wikipedia.org/wiki/Artificial_neural_network[Accessed December 20, 2014].

[5] Ant colony optimization algorithms, December 2014. Available at http://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms[Accessed December 20, 2014].

[6] 司守奎, 孙玺菁. 数学建模算法与应用. 国防工业出版社, 北京, 第 1 版 edition, August 2011.

[7] Comap. 2006 MCM/ICM Problems. http://www.comap.com/undergraduate/contests/mcm/contests/2006/problems/.

[8] Comap. 2003 MCM/ICM Problems. http://www.comap.com/undergraduate/contests/mcm/contests/2003/problems/.

[9] Brian Camley, Bradley Klingenberg, and Pascal Getreuer. Sprinkle, sprinkle, little yard. *UMAPJournal*, page 295, 2006.