启发式算法简介及其在数学模型中的应用

大学生数学建模基础课程

周吕文

极值学院

Notes

启发式算法的定义

启发式算法 (Heuristic Algorithm) 是一种基于直观或经验的局部优化算法. 启发式算法的定义:

- 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.
- 在可接受的花费 (指计算时间和空间) 下给出待解决组合优化问题每一个实例的一个可行解, 该可行解与最优解的偏离程度不一定事先可以预计.
- 启发式算法是一种技术, 该技术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.

Notes

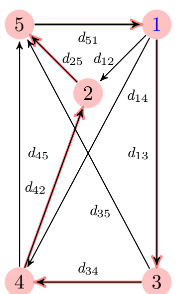
几种启发式算法

- 禁忌搜索 (Tabu Search): 它是对局部领域搜索的一种扩展, 是一种全局逐步寻优算法, 是对人类智力过程的一种模拟.
- 模拟退火 (Simulated Annealing): 它是一种通过模拟物理退火过程搜索最优解的方法.
- 遗传算法 (Genetic Algorithms): 它是一种通过模拟自然进化过程搜索最优解的方法.
- 神经网络 (Neural Networks): 它是一种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 (Ant Algorith): 它是一种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.

Notes

经典问题: 旅行商问题

旅行商问题 (TSP): 假设有一个旅行商人要拜访 n 个城市, d_{ij} 表示两城市间距离. x_{ij} 为 0,1 变量, 表示拜访路径是否包函路径 d_{ij}.



限制: 每个城市必需且只能拜访一次, 最后要回到原来出发的城市, 即

$$\sum_{i=1}^{n} x_{ij} = \sum_{j=1}^{n} x_{ij} = 1$$

目标: 路径的选择目标是要求所得路径路程为所有路径之中的最小值, 即

$$\min \sum_{i\neq j} d_{ij} x_{ij}$$

Notes

经典问题: 旅行商问题

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 $(n-1)$.

TSP 问题枚举的计算开销

如果计算机每秒能完成 24 个城市的枚举, 则

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

Notes

启发式算法与特等奖

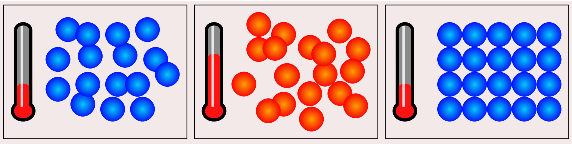
近十二年 MCM 比赛中用到启发式算法的特等奖论文统计

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

Notes

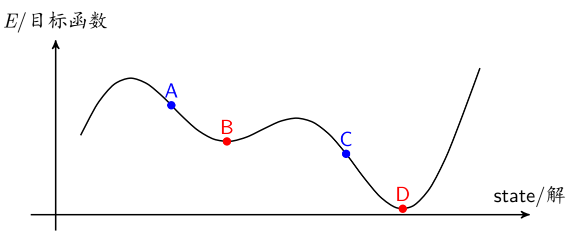
物理退火

物理退火过程



Notes

模拟退火算法与 Metropolis 准则



$$P_{\text{accept}}(E_{i+1} \le E_i) = 1, \quad P_{\text{accept}}(E_{i+1} > E_i) = e^{-(E_{i+1}-E_i)/KT}$$

Notes

模拟退火算法与物理退火过程的相似关系

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

Notes

模拟退火算法基本思想

模拟退火算法伪代码

```
construct initial solution x_0, and x^current = x_0
set initial temperature T = T_0
while continuing criterion do
    for i = 1 to T_L do
        generate randomly a neighbouring solution x' ∈ N(x^current)
        compute change of cost ΔC = C(x') − C(x^current)
        if ΔC ≤ 0 or random(0,1) < exp(−ΔC/kT) then
            x^current = x' {accept new state}
        end if
    end for
    set new temperature T = decrease(T) {decrease temperature}
end while
return solution corresponding to the minimum cost function
```


Notes

模拟退火算法设计要素

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.

Notes

模拟退火算法设计要素

初始温度如何确定?

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.

均匀抽样一组状态，以各状态目标值的方差为初温.

如果能确定邻解间目标函数 (COST 函数) 的最大差值，就可以确定出初始温度 T_0，以使初始接受概率 $P = e^{-|\Delta C|_{\max}/T}$ 足够大. $|\Delta C|_{\max}$ 可由随机产生一组状态的最大目标值差来替代.

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，直到所有的尝试尝试运动都被接受，将此时的温度设置为初始温度.

由经验给出，或通过尝试找到较好的初始温度.

Notes

模拟退火算法设计要素

等温步数如何确定?

- 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.

Notes

模拟退火算法设计要素

如何降温?

- 经典模拟退火算法的降温方式

$$T(t) = \frac{T_0}{\log(1+t)}$$

- 快速模拟退火算法的降温方式

$$T(t) = \frac{T_0}{1+t}$$

- 常用的模拟退火算法的降温方式还有 (通常 $0.8 \le \alpha \le 0.99$)

$$T(t+\Delta t) = \alpha T(t)$$

Notes

模拟退火算法设计要素

花费函数 COST

- 应该能被快速的计算, 花费函数的计算是程序的可能瓶颈.
- 花费函数 COST 一般由目标函数来构造. 目标函数, 或目标函数的倒数/相反数经常直接作为花费函数.

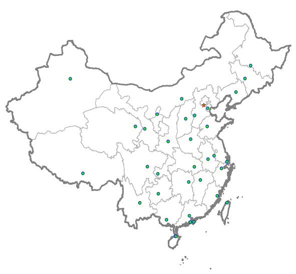
终止条件

- 理论上温度要降为 0 才终止退火算法. 但实际上温度较低时, 尝试的接受概率就几乎为 0 了.
- 设置终止温度的阈值, 或设置外循环迭代次数.
- 算法搜索到的最优值连续若干步保持不变.

Notes

TSP 问题的模拟退火求解: 问题

已知中国 34 个省会城市 (包括直辖市) 的经纬度, 要求从北京出发, 游遍 34 个城市, 最后回到北京. 用模拟退火算法求最短路径.



- 如何设置初始解 S_0?
  $[1, \cdots, i, \cdots, j, \cdots, n, 1]$
- 如何产生邻解 S'?
  $[1, \cdots, j, \cdots, i, \cdots, n, 1]$
- 如何定义 COST 函数?
  $\sum_{i=1}^{n} \mathsf{dist}(S_{(i)}, S_{(i+1)})$
- 如何设置温度, 降温?
  $T_0 = 1000$, $T_{\tau+d\tau} = \alpha T_\tau$

Notes

TSP 问题的模拟退火求解：程序

主程序 MatLab 代码

```matlab
route = randperm(numberofcities); %路径格式:[1,2,...,n]
temperature = 1000; cooling_rate = 0.95;   %初始化温度
Titerations = 1;                  %用来控制降温的循环
previous_distance = totaldistance(route);  %计算路径总长
while temperature > 1.0           %循环继续条件
   temp_route = perturb(route, 'reverse'); %扰动产生邻解
   current_distance = totaldistance(temp_route);%路长
   diff = current_distance - previous_distance;
   if (diff<0)||(rand < exp(-diff/(temperature)))
      route = temp_route;         %接受当前解
      previous_distance = current_distance;
      Titerations = Titerations + 1;
   end
   if Titerations >= 10          %每10步降温(等温步数为10)
      temperature = cooling_rate*temperature;
      Titerations = 0;
   end
end
```

Notes

TSP 问题的模拟退火求解：程序

距离矩阵函数 distancematrix

```matlab
function dis = distancematrix(city)
numberofcities = length(city);
R = 6378.137; %地球半径, 用于求两个城市的球面距离
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
```

路径距离计算函数 totaldistance

```matlab
function d = totaldistance(dis, route)
d = dis(route(end),route(1));
for k = 1:length(route)-1
    i = route(k); j = route(k+1);
    d = d + dis(i,j);
end
```

Notes

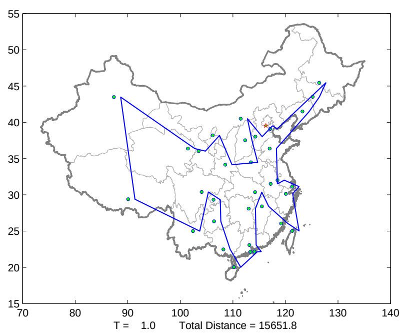
TSP 问题的模拟退火求解：程序

产生邻解的函数 perturb

```matlab
function route = perturb(route_old,method)
route = route_old;
numbercities = length(route);
city1 = ceil(numbercities*rand);    %  [1, 2, ..., n-1, n]
city2 = ceil(numbercities*rand);    %  1<=city1, city2<=n
switch method
    case 'reverse'        %[1 2 3 4 5 6] -> [1 5 4 3 2 6]
        cmin = min(city1,city2);
        cmax = max(city1,city2);
        route(cmin:cmax) = route(cmax:-1:cmin);
    case 'swap'           %[1 2 3 4 5 6] -> [1 5 3 4 2 6]
        route([city1, city2]) = route([city2, city1]);
end
```

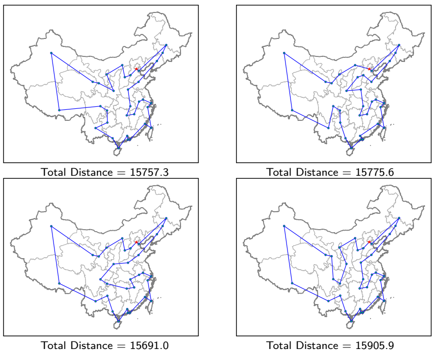

Notes

TSP 问题的模拟退火求解：结果



Notes

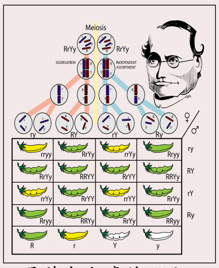
TSP 问题的模拟退火求解：结果



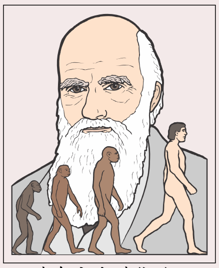
Notes

遗传算法的启源

生物中的遗传与进化理论



孟德尔和遗传理论



达尔文和进化论

Notes

基本概念

种群

染色体 1 ⟨ 1 2 3 4 5 ⟩

染色体 2 ⟨ 2 4 3 8 9 ⟩

⋮

染色体 n ⟨ 7 8 3 6 0 ⟩

基因

基因 (Gene)
染色体上的一个单元，解中的一个参数.

染色体 (Chromosome)
由一组基因构成，问题可能的一个解.

种群 (Population)
由一系列染色体组成的一个集合.

Notes

遗传算法基本思想

遗传算法伪代码

```
set initial generation k = 0
probability of mutation = α
probability of performing crossover = β
construct a population of n randomly-generated individuals P_k;
while not termination do
    evaluate: compute fitness(i) for each individuals in P_k
    select: select m members of P_k insert into P_{k+1}.
    crossover: produce αm children by crossover and insert into P_{k+1}
    mutate: produce βm children by mutate and insert into P_{k+1}
    update generation: k = k + 1
end while
reture the fittest individual from P_last
```


Notes

编码

遗传算法中，首要问题就是如何对解进行编码(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.

十进制	二进制	格雷码
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

二进制编码: 每个基因值为符号 0 和 1 所组成的二值制数.

格雷编码: 与二进制编码类似，连续两个整数所对应编码仅一码之差.

实数编码: 每个基因值用某一范围内的一个实数来表示.

符号编码: 染色体编码串中的基因值取自一个无数值含义，而只有代码含义的符号集.

Notes

适应度函数

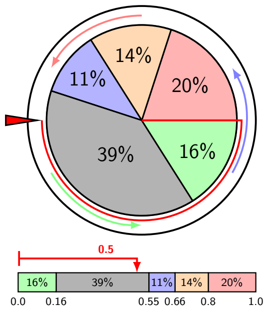
适应度函数也称评价函数，是根据目标函数确定的用于区分群体中个体好坏的标准. 适应度函数值的大小是对个体的优胜劣汰的依据.

- 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.
- 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).
- 比较好的适应度函数应: 单值，连续，非负，最大化.
- 适应度函数不应过于复杂，越简单越好，以便于计算机的快速计算.

Notes

选择

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.



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.

$$p_i = f_i / \sum_{j=1}^{N} f_j$$

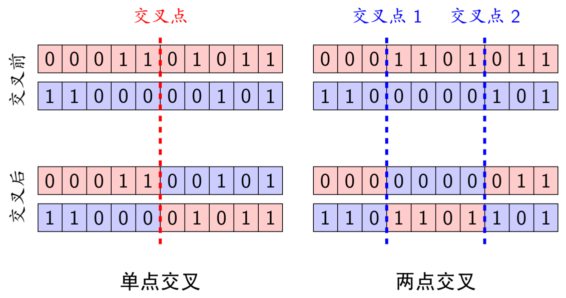
两两竞争: 从父代中随机地选取两个个体，比较适应值，保存优秀个体，淘汰较差的个体.

排序选择: 根据各个体的适应度大小进行排序，然后基于所排序号进行选择.

Notes

交叉

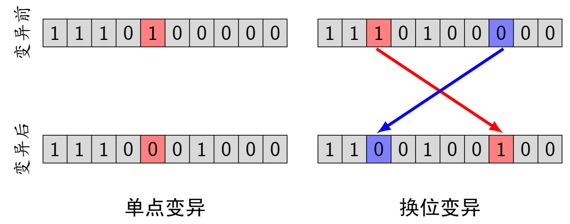
交叉运算，是指对两个相互配对的染色体依据交叉概率按某种方式相互交换其部分基因，从而形成两个新的个体.



Notes

变异

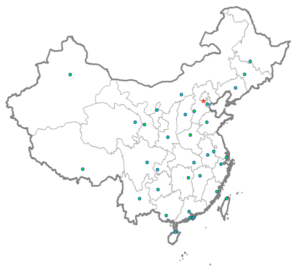
变异操作对群体中的个体的某些基因座上的基因值作变动，模拟生物在繁殖过程，新产生的染色体中的基因会以一定的概率出错.



Notes

TSP 问题的遗传算法求解

已知中国 34 个省会城市 (包括直辖市) 的经纬度，要求从北京出发，游遍 34 个城市，最后回到北京．用遗传算法求最短路径.



如何对问题的解编码?

$[1, \cdots, i, \cdots, j, \cdots, n, 1]$

如何构造适应度函数?

$1/\sum_{i=1}^{n} \mathsf{dist}(S_{(i)}, S_{(i+1)})$

如何设计选择运算?

两两竞争，轮盘赌...

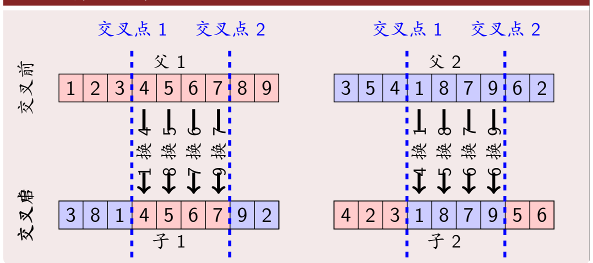
如何设计交叉运算?

如何设计变异运算?

Notes

TSP 问题的遗传算法求解

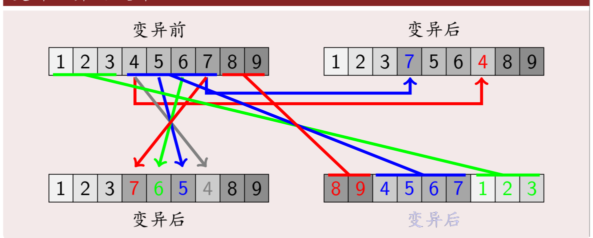
交叉运算的设计



Notes

TSP 问题的遗传算法求解

变异运算的设计



Notes

TSP 问题的遗传算法求解

主程序 MatLab 代码

```
popSize = 100;                      % 种群规模
max_generation = 1000;              % 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 % 主循环开始
    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,popSize,'competition');% 选择
    pop = crossover(pop);           % 交叉
    pop = mutate(pop,Pmutation);    % 变异
    pop = [bestPop; pop];           % 精英保护
end                                 % 主循环开始
popDist = total_distance(pop,dis);% 计算距离(适应度)
[minDist, index] = min(popDist);    % 找出最短距离
optRoute = pop(index,:);            % 找出最短距离对就的路径
```

Notes

TSP 问题的遗传算法求解

选择操作函数 select

```
function popselect = select(pop, fitness, nselect, method)
popSize = size(pop,1);
switch method
    case 'roulette'                 % 轮盘赌
        p=fitness/sum(fitness);     % 选中概率 [0.2 0.3 0.5]
        cump=cumsum(p);             % 概率累加 [0.2 0.5 1.0]
        % 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 = ceil( popSize*rand(1,nselected) );
        i2 = ceil( popSize*rand(1,nselected) );
        I = i1.*( fitness(i1)>=fitness(i2) ) + ...
            i2.*( fitness(i1)< fitness(i2) );
end
popselect = pop(I);
```

Notes

TSP 问题的遗传算法求解

交叉操作函数 crossover

```
function children = crossover(parents)
[popSize, numberofcities] = size(parents);
children = parents;                 % 初始化子代
for i = 1:2:popSize                 % 交叉开始
    parent1 = parents(i+0,:);  child1 = parent1;
    parent2 = parents(i+1,:);  child2 = parent2;
    InsertPoints = ceil(numberofcities*rand(1,2));% 交叉点
    for j = min(InsertPoints):max(InsertPoints)
        if parent1(j)~=parent2(j) % 如果对应位置不重复
            child1(child1==parent2(j)) = child1(j);
            child1(j) = child2(j);
            child2(child2==parent1(j)) = child2(j);
            child2(j) = child1(j);
        end
    end
    children(i+0,:) = child1; children(i+1,:) = child2;
end                                 % 交叉结束
```

Notes

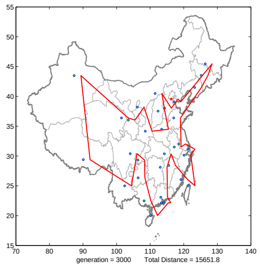
TSP 问题的遗传算法求解

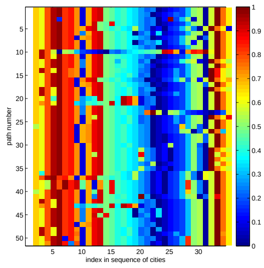
变异操作函数 mutate

```
function children = mutation(parents, probmutation)
[popSize, numberofcities] = size(parents);
children = parents;                 % 初始化子代
for k=1:popSize                     % 变异开始
    if rand < probmutation          % 以一定概率变异
       InsertPoints = ceil(numberofcities*rand(1,2));
       I = min(InsertPoints); J = max(InsertPoints)l
       switch ceil(rand*4)          % swap, slide, flip
           case 1    % [1 2 3 4 5 6 7] -> [1 5 3 4 2 6 7]
            children(k,[I J]) = parents(k,[J I]);
           case 2    % [1 2 3 4 5 6 7] -> [1 3 4 5 2 6 7]
            children(k,I:J) = parents(k,[I+1:J I]);
           otherwise % [1 2 3 4 5 6 7] -> [1 5 4 3 2 6 7]
            children(k,I:J) = parents(k,J:-1:I);
       end
    end
end                                 % 变异结束
```

Notes

TSP 问题的遗传算法结果





Notes

Thank You!!!

Notes

Notes

Notes