实验报告

————模拟退火算法

姓名：陈恩婷　　学号：19335015　　日期：2020/10/11

摘要：本次实验通过实现多种交叉与变异策略与遗传算法并进行可视化的 Matlab 程序来求解 tsp 问题，探究了遗传算法的性能。总的来说，遗传算法的性能不如模拟退火算法的最优性能但差距不大。

1．导言

(1) 问题描述：

用遗传算法求解 TSP 问题（问题规模等和模拟退火求解 TSP 实验同），要求：
1.设计较好的交叉操作，并且引入多种局部搜索操作（可替换通常遗传算法的变异操作）
2.和之前的模拟退火算法（采用相同的局部搜索操作）进行比较
3.得出设计高效遗传算法的一些经验，并比较单点搜索和多点搜索的优缺点。

(2) 问题及方法介绍：

旅行商问题（英语：Travelling salesman problem, TSP）是组合优化中的一个 NP 困难问题，在运筹学和理论计算机科学中非常重要。问题内容为“给定一系列城市和每对城市之间的距离，求解访问每一座城市一次并回到起始城市的最短回路。”

遗传算法（英语：Genetic Algorithm, GA）是计算数学中用于解决最优化的搜索算法，是进化算法的一种。进化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、自然选择以及杂交等等。

遗传算法通常实现方式为一种计算机模拟。对于一个最优化问题，一定数量的候选解（称为个体）可抽象表示为染色体，使种群向更好的解进化。进化从完全随机个体的种群开始，之后一代一代发生。在每一代中评价整个种群的适应度，从当前种群中随机地选择多个个体（基于它们的适应度），通过自然选择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。

2．实验过程

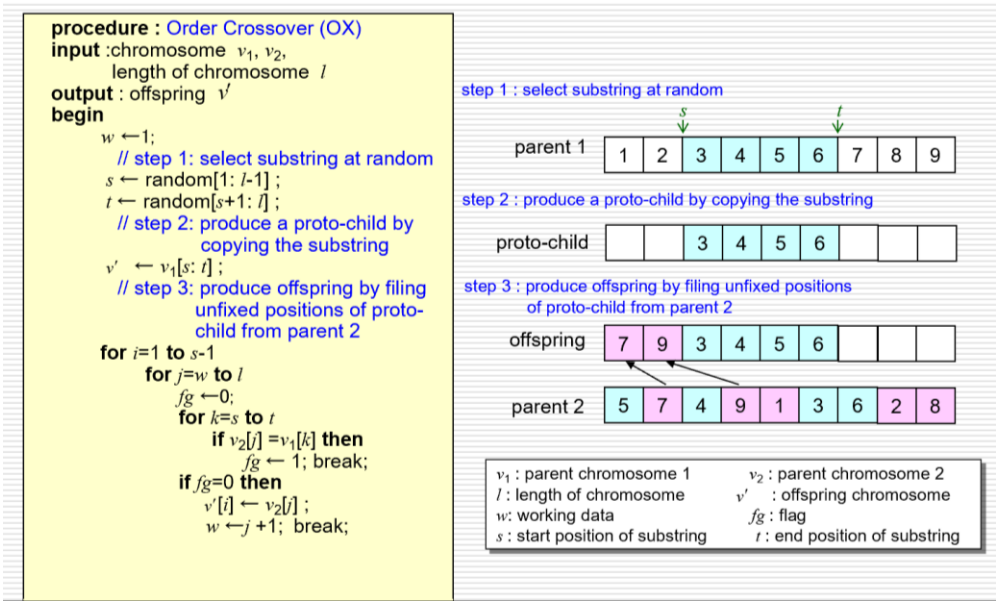
程序主要流程如下：

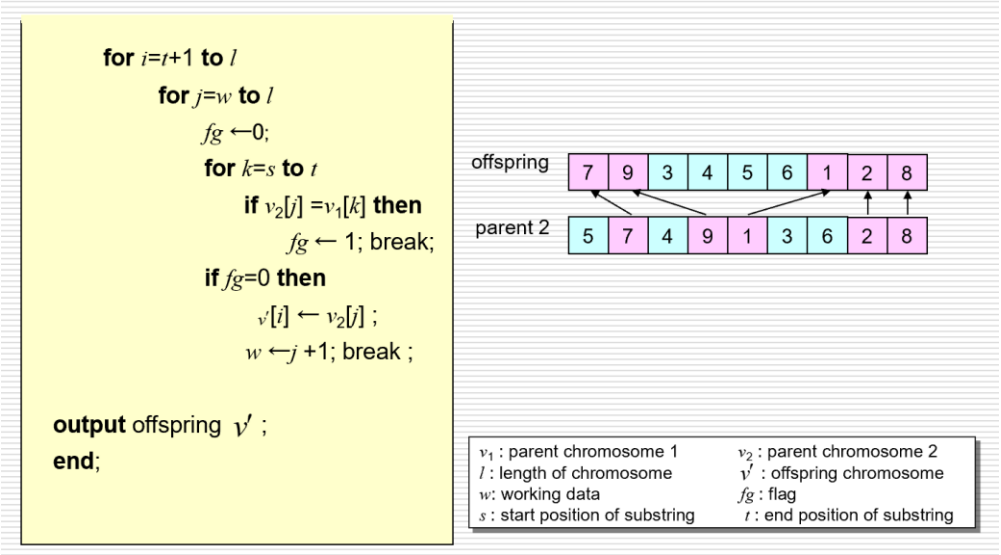
1. 从文件中读入 tsp 问题以及最优解；

2. 随机生成 100 个解作为初始种群；
3. 按照所有个体的路径长度分配一定概率（长度越小概率越大），反复随机选取亲本进行交叉或变异，每次都从所有的交叉和变异策略中随机选取策略，重复 100 次，生成一些新的近似解；
4. 在所有的个体中选出长度最小的 100 个解，作为下一代种群，并将其余个体淘汰掉；
5. 不断重复第 3、4 步，直到最优解的长度符合要求，或者种群的代数达到设定大小。

变异策略和模拟退火实验中的局部搜索策略相同，接下来主要具体解释各个交叉策略。

1. OX crossover

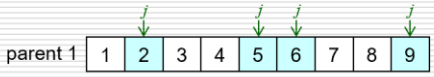




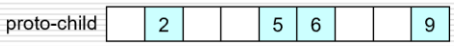
2. PBX

```
procedure : Position-based Crossover
input : chromosome v1, v2,
        length of chromosome l
output : offspring v'
begin
    T ← ∅, S ← ∅, w ← 1;
     // step 1: select a set of positions
                from parent 1 at random
    N ← random[1:l] ;
     // step 2: produce proto-child by
                copying values of
                selected positions
    for i=1 to N
        j ← random[1:l];
        v'[j] ← v1[j] ;
        T ← T∪j ;
        S ← S ∪ v1[j] ;
```

step 1 : select a set of positions from parent 1 at random



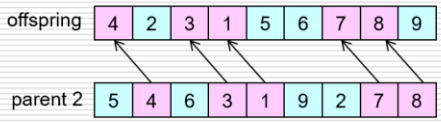
step 2 : produce proto-child by copying values of selected positions



v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of selected positions
$T=\{t[j]\}$, j=1,2,…, N : selected positions set
$S=\{s[m]\}$, m=1,2,…, N : genes value set of selected positions
fg_1 : flag 1 fg_2 : flag 2
w : working data

```
// step 3: produce offspring by filing unfixed
           positions of proto-child from parent 2
    for i=1 to l
        fg1 ← 0;
        for j =1 to N
            if i=t[j] then fg1 ← 1;
        if fg1 =1 then continue;
        for k=w to l
            fg2 ← 0;
            for m=1 to N
                if v2[k]=s[m] then
                    fg2 ← 1; break;
            if fg2 =0 then
                v'[i] ← v2[k ];
                w ← k +1 ; break ;
 output offspring v' ;
end
```

step 3 : produce offspring by filing unfixed positions of proto-child from parent 2

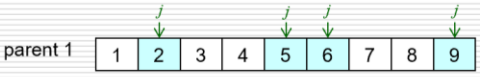


v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of selected positions
$T=\{t[j]\}$, j=1,2,…, N : selected positions set
$S=\{s[m]\}$, m=1,2,…, N : genes value set of selected positions
fg_1 : flag 1 fg_2 : flag 2
w : working data

3. OBX

```
procedure : Order-Based Crossover
input : chromosome v1, v2,
        length of chromosome l
output : offspring v'
begin
    S ← ∅, T← ∅, w ← 1;
     // step 1: select a set of positions
                from parent 1 at random
    N ← random[1:l] ;
    for i=1 to N
        j ← random[1:l];
        S ← S ∪ v1[j] ;
```

step 1 : select a set of positions from parent 1 at random



v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of selected positions
$T=\{t[j]\}$, j=1,2,…, N : positions set of assigned genes from v_2 to v'
$S=\{s[i]\}$, i=1,2,…, N : genes value set of selected positions
fg_1 : flag 1 fg_2 : flag 2
w : working data

```
    // step 2: produce proto-child by copying
            non-selected value from parent 2
    for i=1 to l
        fg1←0;
        for j=1 to N
          if v2[i] = s[j] then
              fg1←1; break;
        if fg1=0 then
          v'[i] ← v2[i],
          T← T ∪i;
// step 3: produce offspring by copying selected
            values from parent 1
    for i=1 to l
        fg2←0;
        for j=1 to N
          if i =t[j] then fg2←1 ;
        if fg2=1 then continue;
        v'[i] ← s[w];
        w ← w+1;
    output offspring v';
end
```

step 2: produce proto-child by copying non-selected value from parent 2

parent 2	5	4	6	3	1	9	2	7	8
proto-child		4		3	1			7	8

step 3 : produce offspring by copying selected values from parent 1

parent 1	1	2	3	4	5	6	7	8	9
proto-child	2	4	5	3	1	6	9	7	8

v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of selected positions
$T=\{t[j]\}$, j=1,2,…, N: positions set of assigned genes from v_2 to v'
$S=\{s[i]\}$, i=1,2,…, N : genes value set of selected positions
fg_1 : flag 1 fg_2 : flag 2
w : working data

4. Subtour Exchange Crossover

```
procedure : Subtour Exchange Crossover
    input : chromosome v1, v2,
            length of chromosome l
    output : offspring v1', v2'
    begin
        S ← ∅, T ← ∅;
        // step 1: select subtours in parents
        s ← random[1:l-1] ;
        t ← random[s+1:l] ;
        for i=1 to l
            for j=s to t
                if v2[i]=v1[j] then
                    S ← S ∪ v2[i];
                else
                    v2'[i] ← v2[i],
                    T ← T ∪i;
```

step 1 : select subtours in parents

			s				t		
parent 1	1	2	3	4	5	6	7	8	9
parent 2	3	4	9	7	8	5	2	1	6

v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v_1' : offspring chromosome 1
v_2' : offspring chromosome 2
N : total number of values in subtour
$S=\{s[i]\}$, i=1,2,…, N: value set of subtour
$T=\{t[j]\}$, j=1,2,…, N: positions set of S
s : stsrt position of substring in v_1 t : end position of substring v_1
fg : flag

```
    // step 2: exchange subtours
    v1' ← v1[1:s-1] // S // v1[ t+1:l] ;
    k ← s;
    for i=1 to l
        fg ← 0;
        for j=1 to |T|
            if i=t[j] then fg ← 1;break;
        if fg=1 then continue;
        v2'[i]← v1[k] ;
        k ← k+1;
    output offspring v1', v2';
end
```

step 2 : exchange subtours

offspring 1	1	2	3	4	7	5	6	8	9
offspring 2	3	4	9	5	8	6	2	1	7

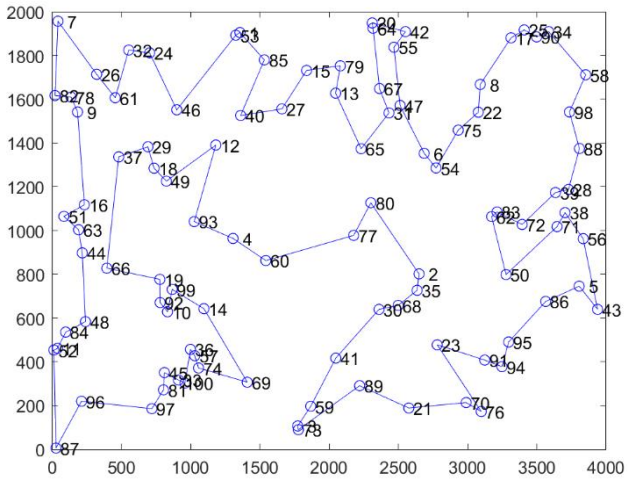
v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v_1' : offspring chromosome 1
v_2': offspring chromosome 2
N : total number of values in subtour
$S=\{s[i]\}$, i=1,2,…, N: value set of subtour
$T=\{t[j]\}$, j=1,2,…, N: positions set of S
s : start position of substring in v_1 t : end position of substring v_1
fg : flag

3．结果分析

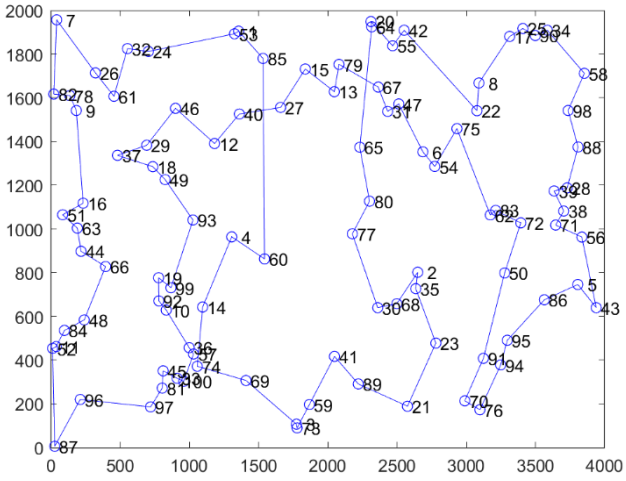
本实验在 Matlab R2020b 上实现并运行。种群的初始数量为 100，每进行 100 次交叉或变异就进行一次淘汰。

以下是其中两次运行的结果：

```
total iteration: 2906
ans =
     2.275220398939275e+04
```



```
total iteration: 3028
ans =
     2.267166663344660e+04
```



多次运行可以发现，遗传算法找到最优解的 1.1 倍以内的近似解大致需要迭代 3000 代，也就是大约进行 300000 次交叉与变异，这个结果和之前的模拟退火算法的最优性能相差不太大（229000 次局部搜索），这可能是因为模拟退火本身选择的就是最优的局部搜索策略（inversion mutation），另外，模拟退火在运行过程中，每生成一个新解就决定是否淘汰它，这样会比遗传算法在生成一批新个体后再进行淘汰效果更好。

4．结论

本次实验使用了多种交叉与变异策略，通过实现遗传算法并进行可视化，探究了遗传算法的性能，并与模拟退火算法进行了对比。总的来说，遗传算法性能不如模拟退火算法的最优性能但差距不大，这主要是因为模拟退火算法选用了最优的局部搜索策略，且遗传操作本身开销较大。本次实验还让我体会到了 Matlab 编程的很多好处，代码较简洁且 debug 很方便。

主要参考文献(三五个即可)

1. combOptGA-v1.00(1).ppt
2. 人工智能基础教程-朱福喜第二版