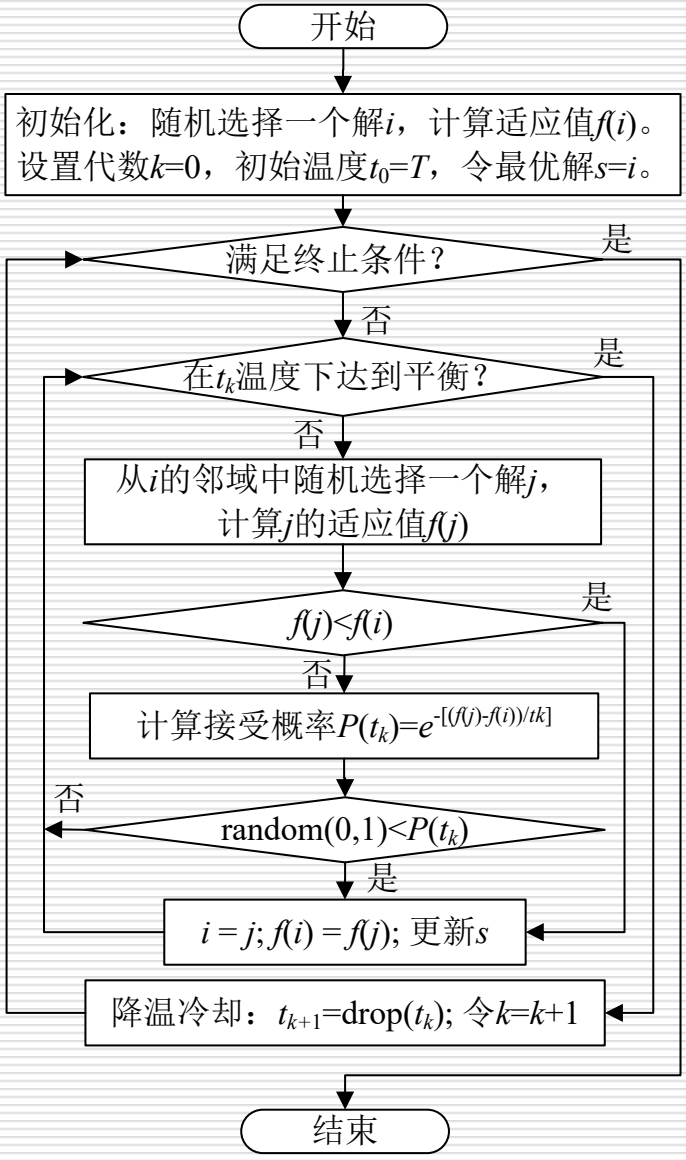
高级搜索

目录

1.1 模拟退火算法



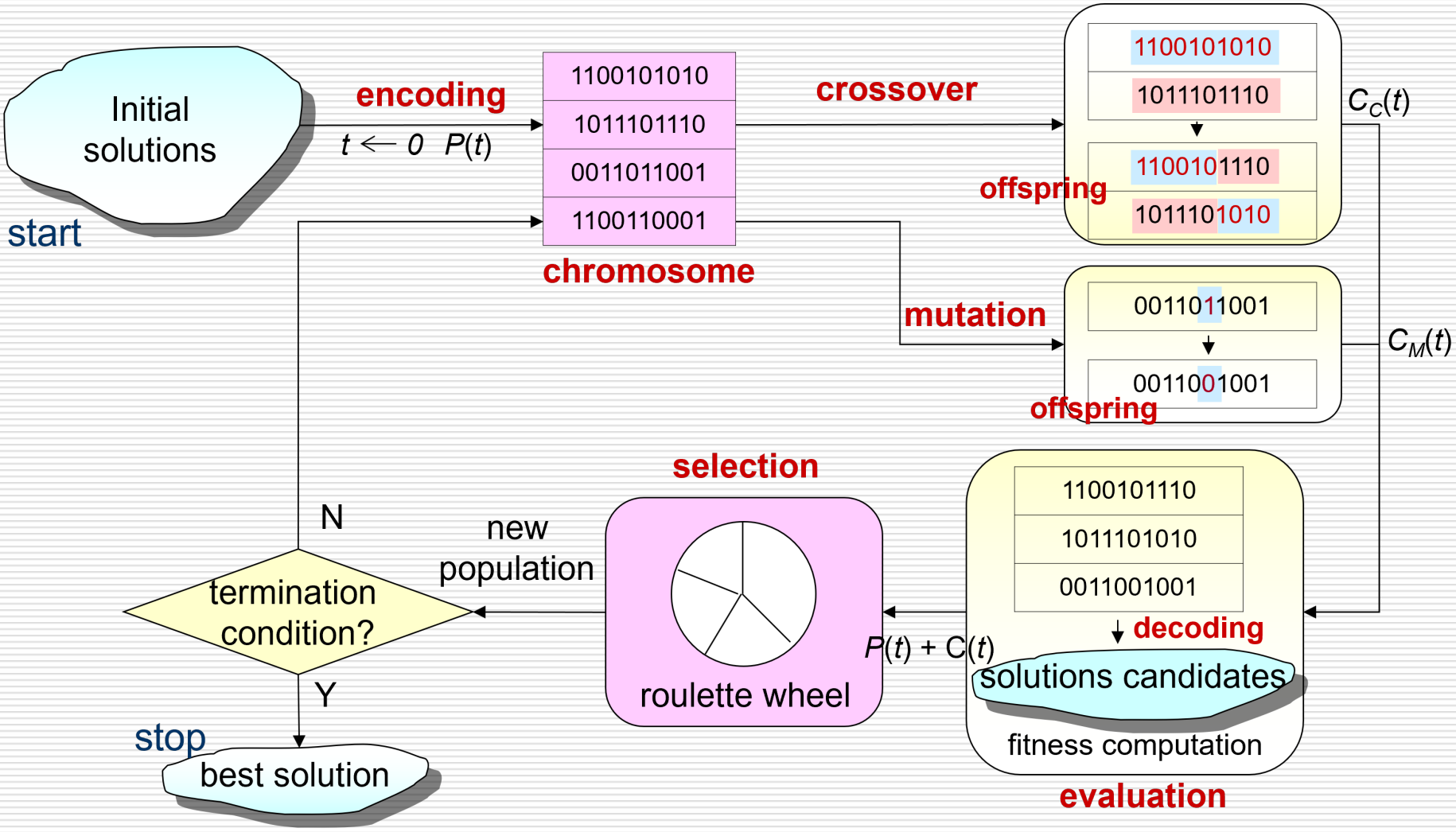

```
//功能：模拟退火算法伪代码
//说明：本例以求问题最小值为目标
//参数：T为初始温度; L为内层循环次数

procedure SA
  //Initialization
    Randomly generate a solution X0, and calculate its
    fitness value f(X0);
  Xbest=X0; k=0; tk=T;
  while not stop
    //The search loop under the temperature tk
    for i=1 to L  //The loop times
      Generate a new solution Xnew based on the current
      solution Xk, and calculate its fitness value f(Xnew).
      if f(Xnew) < f(Xk)
          Xk = Xnew;
          if f(Xk) < f(Xbest) Xbest=Xk;
          continues;
      end if
      Calculate P(tk)=e^-[(f(Xnew)-f(Xk))/tk];
       if random(0,1) < P
          Xk = Xnew;
      end if
    end for
    //Drop down the temperature
    tk+1=drop(tk); k=k+1;
  end while
  print Xbest
end procedure
```

1.1 模拟退火算法

功能意义	基本要素	设置方法
影响模拟退火算法全局搜索性能的重要因素之一。 实验表明，初温越大，获得高质量解的几率越大，但花费的计算时间将增加。	初始温度	1、均匀抽样一组状态，以各状态目标值的方差定初温 2、随机产生一组状态，以两两状态间最大差值定初温 3、利用经验公式给出初温
状态空间与状态产生函数。 邻域函数（状态产生函数）应尽可能保证产生的候选解遍布全部解空间。	邻域函数	候选解一般采用按照某一概率密度函数对解空间进行随机采样来获得。 概率分布可以是均匀分布、正态分布、指数分布等等
指从一个状态X_k(一个可行解)向另一个状态X_{new}(另一个可行解)的转移概率，通俗的理解是接受一个新解为当前解的概率	接受概率	一般采用Metropolis准则 $P_{ij}^{T}=\begin{cases}1, & \text{if } E(j)\le E(i)\\ e^{-\left(\frac{E(j)-E(i)}{KT}\right)}=e^{-\left(\frac{\Delta E}{KT}\right)}, & \text{otherwise}\end{cases}$
指从某一较高温状态t_0向较低温状态冷却时的降温管理表，或者说降温方式	冷却控制	1、经典模拟退火算法的降温方式 $t_k=\frac{t_0}{\lg(1+k)}$ 2、快速模拟退火算法的降温方式 $t_k=\frac{t_0}{1+k}$
内层平衡也称Metropolis抽样稳定准则，用于决定在各温度下产生候选解的数目	内层平衡	1、检验目标函数的均值是否稳定 2、连续若干步的目标值变化较小 3、预先设定的抽样数目，内循环代数
算法的终止条件	终止条件	1、设置终止温度的阈值 2、设置外循环迭代次数 3、算法搜索到的最优值连续若干步保持不变 4、检验系统熵是否稳定

1.2 遗传算法


Initial solutions
start
encoding
$t \leftarrow 0$ $P(t)$
1100101010
1011101110
0011011001
1100110001
chromosome
crossover
1100101010
1011101110
1100101110
1011101010
offspring
$C_C(t)$
mutation
0011011001
0011001001
offspring
$C_M(t)$
selection
1100101110
1011101010
0011001001
decoding
solutions candidates
fitness computation
evaluation
$P(t)$ + C(t)
roulette wheel
new population
N
termination condition?
Y
stop
best solution

1.2 遗传算法

```
procedure: Simple GA
input: GA parameters
output: best solution
begin
    t ← 0;                                  // t: generation number
    initialize P(t) by encoding routine;    // P(t): population of chromosomes
    fitness eval(P) by decoding routine;
    while (not termination condition) do
        crossover P(t) to yield C(t);       // C(t): offspring
        mutation P(t) to yield C(t);
        fitness eval(C) by decoding routine;
        select P(t+1) from P(t) and C(t);
        t ← t+1;
     end
     output  best solution;
end
```

1.3 Traveling Salesman Problem

- The **Traveling Salesman Problem** (TSP) is one of the most widely studied combinatorial optimization problems.
- Its statement is deceptively simple: A salesperson seeks the **shortest tour through *n* cities**.

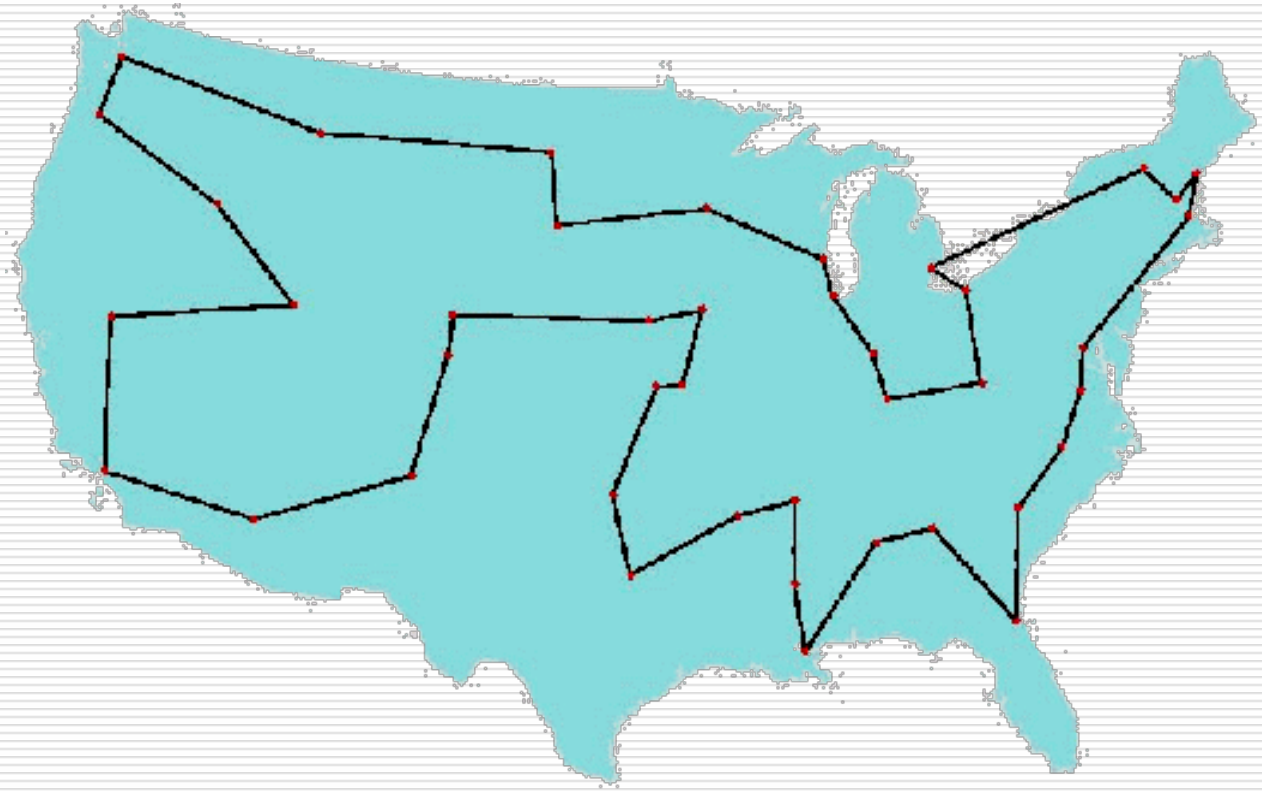


Fig. 3.4 George Dantzig, Ray Fulkerson, and Selmer Johnson (1954) a description of a method for solving the TSP :49 cities

Representation

Random Keys Representation

- This indirect representation encodes a solution with random numbers from (0,1).
- These values are used as sort keys to decode the solution.

chromosome

1	2	3	4	5	6	7	8	9
0.23	0.82	0.45	0.74	0.87	0.11	0.56	0.69	0.78

where position *i* in the list represents city *i*.

tour list 6 - 1 - 3 - 7 - 8 - 4 - 9 - 2 - 5

procedure: Random Keys Encoding
Input: city set,
total number of cities *N*
output: chromosome *v*
begin
for *i* =1 to *N*
v[*i*] ← random[0,1];
output chromosome *v*;
end

procedure: Random Keys Decoding
Input: chromosome *v,*
total number of cities *N*
output: tour list *L*
begin
L← ∅;
for *i* =1 to *N*
L ← *L* ∪*i*;
sort *L* by *v*[*i*];
output tour list *L*;
end

Crossover Operators

- During the past decade, several crossover operators have been proposed for permutation representation, such as **partial-mapped crossover** (PMX), **order crossover** (OX), **cycle crossover** (CX), **position-based crossover,**

  order-based crossover, **heuristic crossover**, and so on.

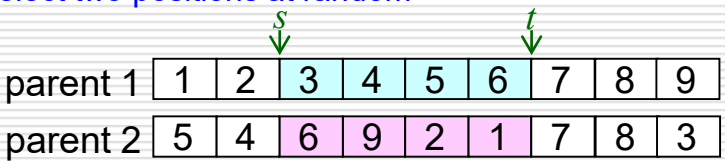
- These operators can be classified into two classes:
  - **Canonical approach**
    - The canonical approach can be viewed as an extension of two-point or multipoint crossover of binary strings to permutation representation.
  - **Heuristic approach**
    - The application of heuristics in crossover intends to generate an improved offspring.

Crossover Operators

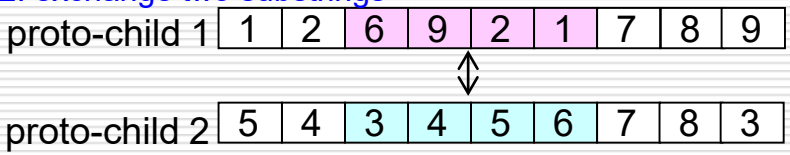
1. **Partial-Mapped Crossover** (PMX)

procedure : PMX crossover
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v_1' , v_2'
begin
$R \leftarrow \phi$;
// step 1: select two positions at random
$s \leftarrow$ random[1:l-1] ;
$t \leftarrow$ random[s+1:l] ;
// step 2: exchange two substrings
$\leftarrow v_1[1{:}s\text{-}1] \,//\, v_2[s{:}\,t] \,//\, v_1[t+1{:}l]$;
$v_1' \leftarrow v_2[1{:}s\text{-}1] \,//\, v_1[s{:}\,t] \,//\, v_2[t+1{:}l]$;
// step 3: determine the mapping relationship
$R \leftarrow$ relation($v_1[s{:}t]$, $v_2[s{:}t]$) ;
// step 4: legalize offspring
legalize(, , R);
output offspring , ;
end

step 1 : select two positions at random

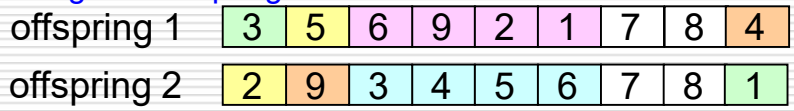


step 2: exchange two substrings



step 3 : determine the mapping relationship

step 4 : legalize offspring



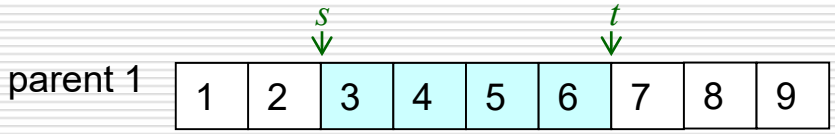
v_1 : parent chromosome 1 — v_2 : parent chromosome 2
l : length of chromosome — v_1' : offspring chromosome 1
v_2' : offspring chromosome 2 — R : relationships
s : start position of substring — t : end position of substring
relation(v_1,v_2) : searching relationship between v_1 and v_2
legalize(v_1,v_2,R) change genes value of v_1, v_2 based on relationship R

Crossover Operators

2. OX crossover

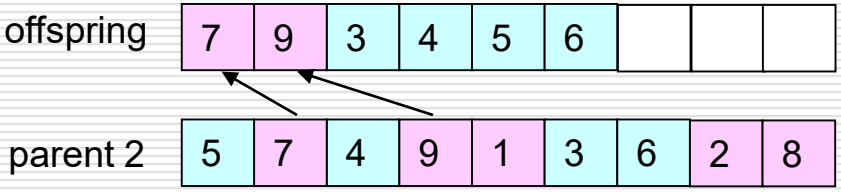
```
procedure : Order Crossover (OX)
input :chromosome  v1, v2,
       length of chromosome  l
output : offspring   v'
begin
      w ←1;
       // step 1: select substring at random
      s ← random[1: l-1] ;
      t ← random[s+1: l] ;
       // step 2: produce a proto-child by
                  copying the substring
      v' ← v1[s: t] ;
       // step 3: produce offspring by filing
                  unfixed positions of proto-
                  child from parent 2
     for i=1 to s-1
          for j=w to l
               fg ←0;
               for k=s to t
                    if v2[j] =v1[k] then
                         fg ← 1; break;
               if fg=0 then
                    v'[i] ← v2[j] ;
                    w ←j +1;  break;
```

step 1 : select substring at random



step 2 : produce a proto-child by copying the substring

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

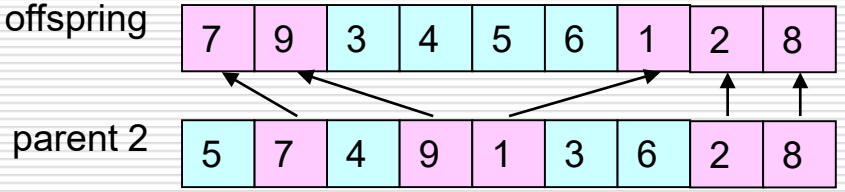


v_1 : parent chromosome 1	v_2 : parent chromosome 2
l : length of chromosome	v' : offspring chromosome
w: working data	fg : flag
s : start position of substring	t : end position of substring

Crossover Operators

```
for i=t+1 to l
    for j=w to l
        fg ←0;
        for k=s to t
            if v2[j] =v1[k] then
                fg ← 1; break;
        if fg=0 then
            [i] ← v2[j] ;
            w ←j +1; break ;

output offspring  v';
end;
```



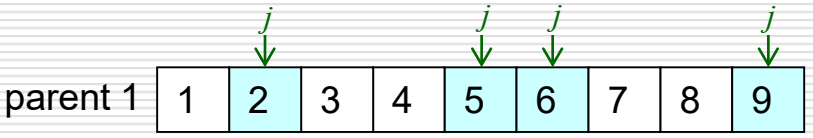
v_1 : parent chromosome 1	v_2 : parent chromosome 2
l : length of chromosome	v' : offspring chromosome
w: working data	fg : flag
s : start position of substring	t : end position of substring

Crossover Operators

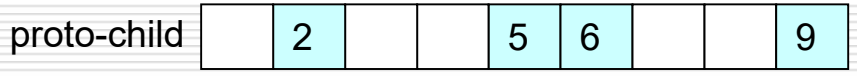
3. **Position-based Crossover** (PBX)

procedure : Position-based Crossover
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
$T \leftarrow \phi, S \leftarrow \phi, w \leftarrow 1$;
// step 1: select a set of positions from parent 1 at random
$N \leftarrow$ random[1:l] ;
// step 2: produce proto-child by copying values of selected positions
for i=1 **to** N
$j \leftarrow$ random[1:l];
$v'[j] \leftarrow v_1[j]$;
$T \leftarrow T \cup j$;
$S \leftarrow S \cup v_1[j]$;

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



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

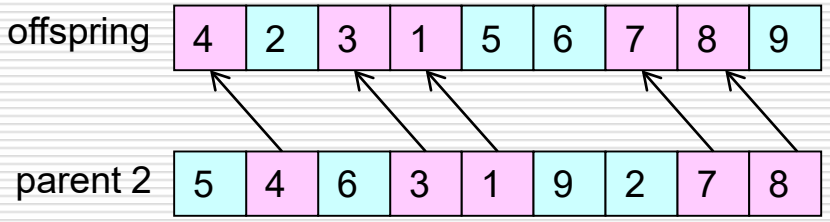


v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of selected positions
T={$t[j]$} , j=1,2,…, N : selected positions set
S ={$s[m]$}, m=1,2,…, N : genes value set of selected positions
fg_1 : flag 1 fg_2 : flag 2
w : working data

Crossover Operators

```
// 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'[t] ← 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

$T=\{t[j]\}$, $j=1,2,\ldots, N$: selected positions set

$S=\{s[m]\}$, $m=1,2,\ldots, N$: genes value set of selected positions

fg_1 : flag 1 fg_2 : flag 2

w : working data

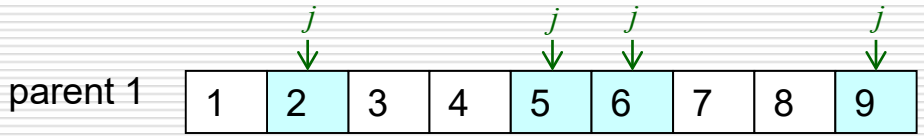
Crossover Operators

4. **Order-Based Crossover** (OBX)

procedure : Order-Based Crossover
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin

S ← ϕ, T ← ϕ, 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 ∪ $v_1[j]$;

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

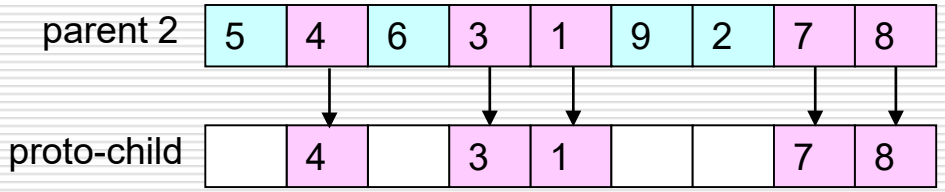


v_1 : parent chromosome 1
v_2 : parent chromosome 2
l : length of chromosome
v' : offspring chromosome
N : total number of selected positions
T={$t[j]$} , j=1,2,…, N : positions set of assigned genes from v_2 to v'
S ={$s[i]$}, i=1,2,…, N : genes value set of selected positions
fg_1 : flag 1
fg_2 : flag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

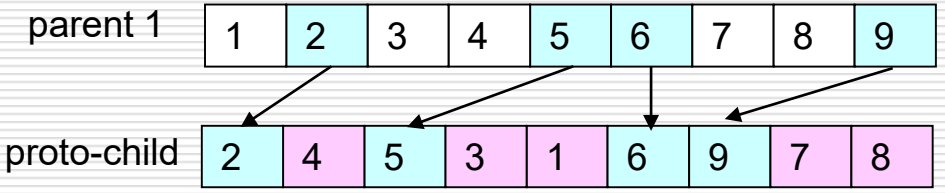
Crossover Operators

```
    // 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   ;y'
end
```

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



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



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

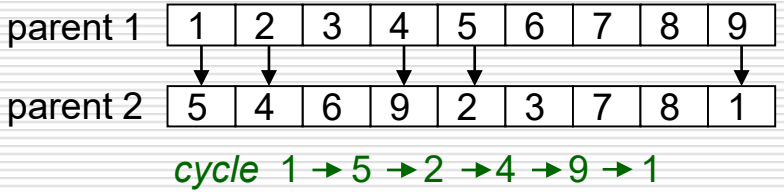
w : working data

Crossover Operators

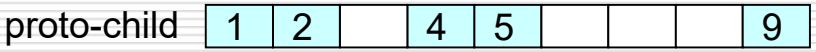
5. **Cycle Crossover** (CX)

procedure : CX crossover
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
S ← ϕ, T ← ϕ, w ← 1;
// step 1: find the cycle between parents
C ← cy (v_1, v_2);
// step 2: produce proto-child by copying
gene values in cycle from parent 1
for i=1 **to** l
for j=1 **to** N-1
if $v_1[i]$ =$c[j]$ **then**
$v'[i]$← $v_1[i]$;
T ← $T \cup i$;
S ← S ∪ $v_1[i]$;

step 1 : find the cycle between parents



step 2 : produce proto-child by copying gene values in cycle from parent 1



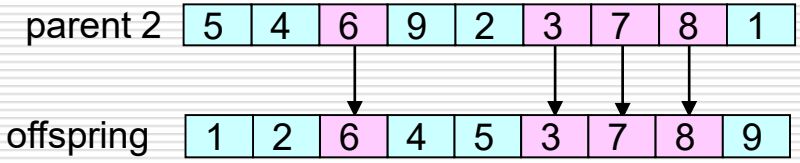
v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of values in cycle

T={$t[n]$} , n=1,2,…, N-1: positions set of S in proto-child
S ={$s[k]$}, k=1,2,…, N-1: proto-child genes value set in cycle
C={$c[j]$}, j=1,2,…, N-1: value set of cycle
fg_1 : flag 1 fg_2 : flag 2
w : working data
cy (v_1,v_2):finding the cycle which is defined by the corresponding positions between parents

Crossover Operators

```
// step 3: produce offspring by filling
           unfixed position from parent 2
    for i=1 to l
        fg1 ← 0;
        for n=1 to |T|
            if i=t[n] then fg1 ← 1;
        if fg1 =1 then continue;
        for j=w to l
            fg2 ← 0;
            for k=1 to |S|
                if v2 [j]=s[k] then
                    fg2 ← 1; break;
            if fg2 =0 then
                v'[i] ← v2[j ];
                w ← j +1 ; break;
    output offspring v' ;
end
```

step 3 : produce offspring by filling unfixed position from parent 2



v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of values in cycle

T={$t[n]$} , n=1,2,…, N-1: positions set of S in proto-child
S ={$s[k]$}, k=1,2,…, N-1: proto-child genes value set in cycle
C={$c[j]$}, j=1,2,…, N-1: value set of cycle
fg_1 : flag 1 fg_2 : flag 2
w : working data
cycle(v_1,v_2): searching cycle between v_1 and v_2

Crossover Operators

6. Subtour Exchange Crossover

procedure : Subtour Exchange Crossover

input : chromosome v_1, v_2,
length of chromosome l

output : offspring v_1' v_2'

begin

$S \leftarrow \phi$, $T \leftarrow \phi$;

// step 1: select subtours in parents

$s \leftarrow$ random[1:l-1] ;

$t \leftarrow$ random[s+1:l] ;

for i=1 **to** l

for j=s **to** t

if $v_2[i]$=$v_1[j]$ **then**

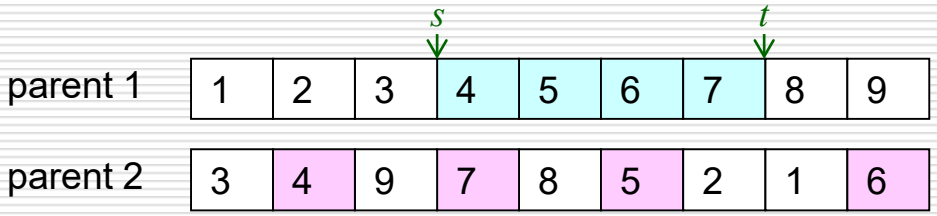
$S \leftarrow S \cup v_2[i]$;

else

$v_2'[i] \leftarrow v_2[i]$,

$T \leftarrow T \cup i$;

step 1 : select subtours in parents



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

Crossover Operators

```
// 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

Mutation Operators

1. Inversion Mutation

procedure : Inversion Mutation

input : chromosome v_1, v_2,
length of chromosome l

output : offspring v'

begin

// step 1: select subtour at random

$s \leftarrow$ random[1:l-1] ;

$t \leftarrow$ random[s+1:l] ;

// step 2: produce offspring by
copying inverse string of
substring

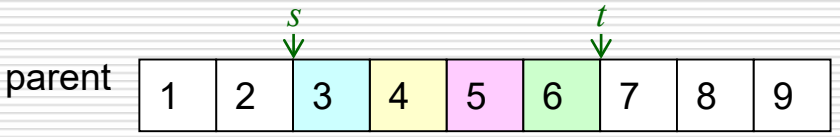
$S \leftarrow$ invert(v[s: t]) ;

$v' \leftarrow v$[1:s-1] // S // v[t+1:l] ;

output offspring v'

end

step 1: select subtour at random



step 2 : produce offspring by copying inverse string of substring

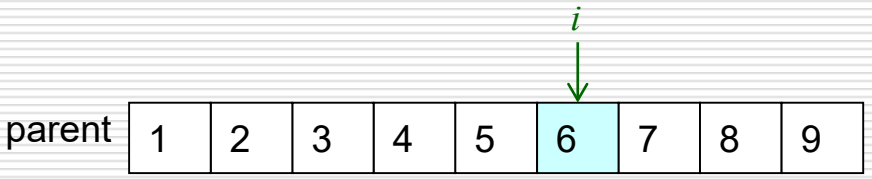
v : parent chromosome
v' : offspring chromosome
t : end position of substring
invert(*string*) : inversely changing order of *string*
l : length of chromosome
s : start position of substring
S : inverse string of substring

Mutation Operators

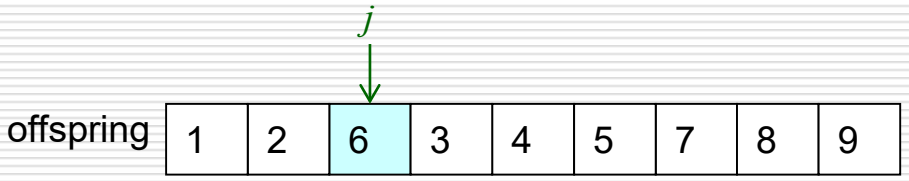
2. Insertion Mutation

procedure : Insertion Mutation
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
// step 1 : select a position in parent 1 at random
$i \leftarrow$ random[1:l] ;
// step 2: insert selected value in randomly selected position parent 2
$j \leftarrow$ random[1:l-1] ;
$W \leftarrow v$[1:i-1] // v[i+1:l] ;
$v' \leftarrow W$[1:j-1] // v[i] // W[j:l-1];
output offspring v'
end

step 1 : select a position in parent 1 at random



step 2: insert selected value in randomly selected position of parent 2



v : parent chromosome
v' : offspring chromosome
j : selected position in parent 2
l : length of chromosome
i : selected position in parent 1
W : working data set

Mutation Operators

3. Displacement Mutation

procedure : Displacement Mutation

input : chromosome v_1, v_2,
length of chromosome l

output : offspring v'

begin

// step 1: select subtour

$s \leftarrow$ random[1:l-1] ;

$t \leftarrow$ random[s+1:l] ;

// step 2: insert subtour in a
random position

$n \leftarrow t$-(s-1);

$i \leftarrow$ random[1:l-n] ;

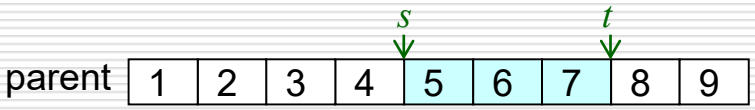
$W \leftarrow v$ [1:s-1] // v [t+1:l];

$v' \leftarrow W$ [1:i-1] // v [s : t] // W [i:l-n] ;

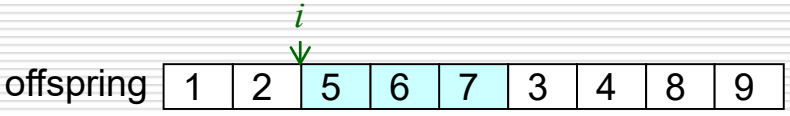
output offspring ;v'

end

step 1 : select subtour



step 2 : insert subtour in a random position



v : parent chromosome
v' : offspring chromosome
t : end position of substring
i : insert position
W : working data set
l : length of chromosome
s : start position of substring
n : length of subtour

Mutation Operators

4. Swap Mutation

procedure : Swap Mutation

input : chromosome v_1, v_2,
length of chromosome l

output : offspring v'

begin

// step 1: select two position at random

$i \leftarrow$ random[1:l -1] ;

$j \leftarrow$ random[i+1 :l] ;

// step 2: produce offspring by swapping

selected positions

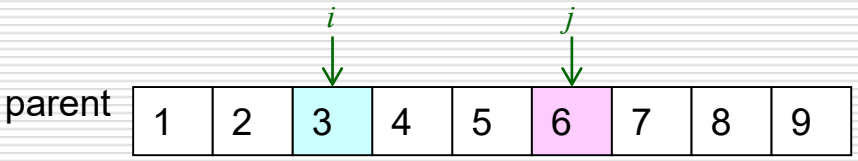
$v' \leftarrow v[1: i-1] // v[j] // v[i+1 : j-1] // v[i] // v[j+1 : l]$;

v'

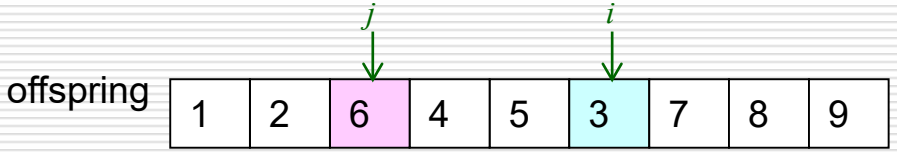
output offspring ;

end

step 1: select two position at random



step 2 : produce offspring by swapping selected positions



v : parent chromosome
v' : offspring chromosome
j : selected position
l : length of chromosome
i : selected position

Mutation Operators

5. Heuristic Mutation

procedure : Heuristic Mutation
input : chromosome v_1, v_2,
length of chromosome l
output : offspring v'
begin
$P \leftarrow \phi$;
// step 1: select positions and produce neighbors
for i=1 **to** m {
$r \leftarrow$ random[1:l] ;
$P \leftarrow P \cup$ nb(v[r]) ;
}
// step 2: produce offspring by evaluating neighbors
$w \leftarrow F(p[1])$;
for i=2 **to** $|P|$
if $w > F(p[i])$ **then**
$w \leftarrow F(p[i]), n \leftarrow i$;
$v' \leftarrow p[n]$;
output offspring v';
end;

step 1 : select positions and produce neighbors

			r			r		r		$F(p[i])$
parent	1	2	3	4	5	6	7	8	9	32
proto-child 1	1	2	3	4	5	8	7	6	9	27
proto-child 2	1	2	8	4	5	3	7	6	9	54
proto-child 3	1	2	8	4	5	6	7	3	9	45
proto-child 4	1	2	6	4	5	8	7	3	9	23
proto-child 5	1	2	6	4	5	3	7	8	9	56

step 2 : produce offspring by evaluating neighbors

offspring	1	2	6	4	5	8	7	3	9	

v : parent chromosome
v' : offspring chromosome
r : selected position
w :working data
n : position of chromosome with best fitness value in P
$P\{p[i]\}$, i=1,2,…,N : neighbor chromosome set
nb(v[r]) : searching neighbors of rth gene
$F(p[i])$: fitness value of $p[i]$
l : length of chromosome
m : total number of selected positions
N : total number of neighbor chromosomes

Overall Algorithm

- GA procedure for Traveling Salesperson Problem

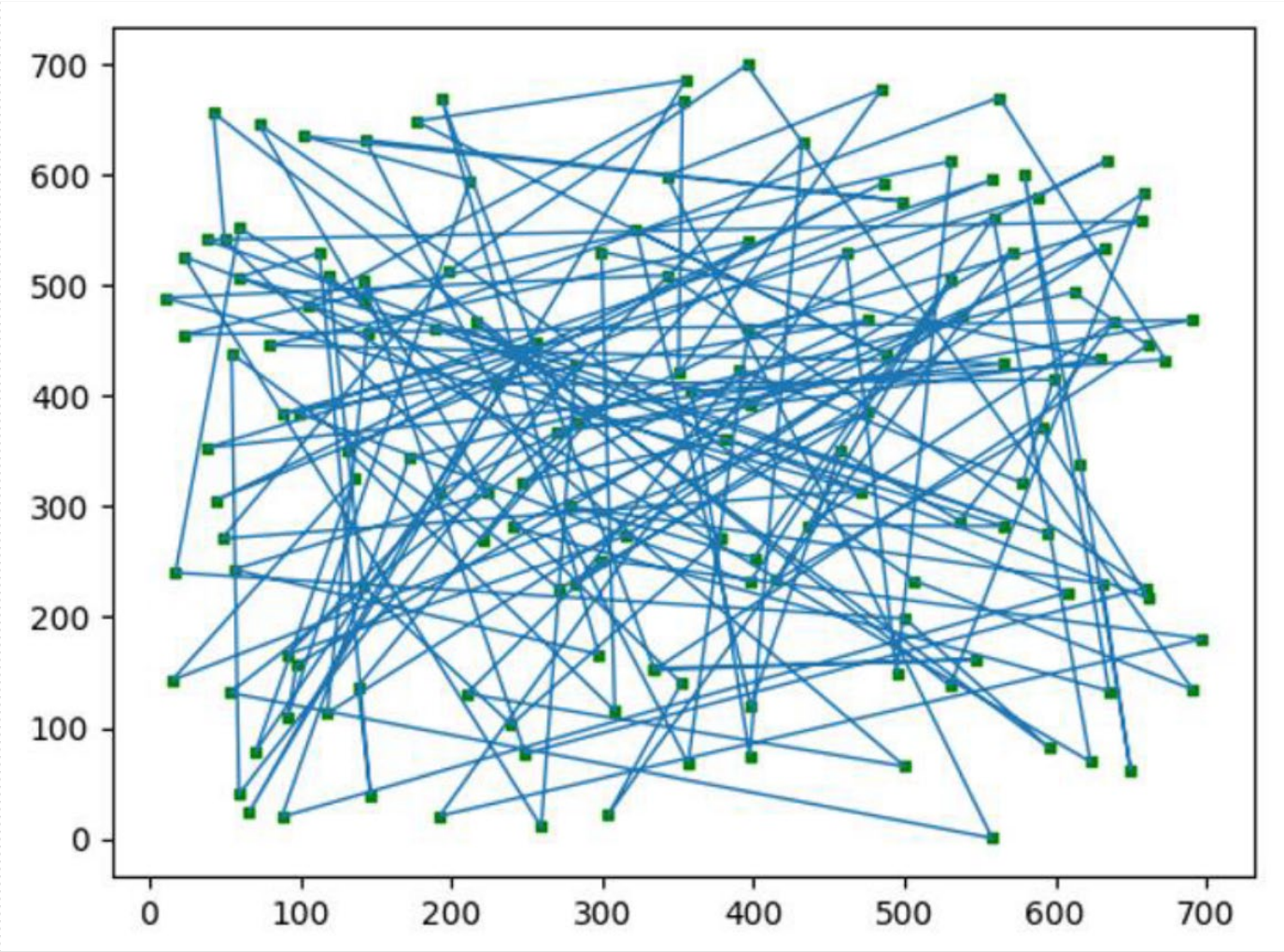
```
procedure: GA for Traveling Salesperson Problem (TSP)
Input: TSP data set, GA parameters
output: best tour route
begin
        t ←0;
        initialize P(t) by permutation encoding or random keys encoding;
        fitness eval(P) by permutation decoding or random keys decoding;
        while (not termination condition) do
                crossover P(t) to yield C(t) by partial-mapped crossover;
                mutation P(t) to yield C(t) by swap mutation;
                fitness eval(C) by permutation decoding or random keys decoding;
                select P(t+1) from P(t) and C(t);
                 t ← t+1;
        end
        output  best tour route;
end
```

2. 实验任务

- 在TSPLIB（http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/，多个地址有备份；其他网站还可以找到有趣的art TSP和national TSP）中选一个大于100个城市数的TSP问题，使用模拟退火和遗传算法求解。
- 模拟退火：
  - 采用多种邻域操作的局部搜索local search策略求解；
  - 在局部搜索策略的基础上，加入模拟退火simulated annealing策略，并比较两者的效果；
  - 要求求得的解不要超过最优值的10%，并能够提供可视化，观察路径的变化和交叉程度。
- 遗传算法：
  - 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）
  - 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
  - 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。

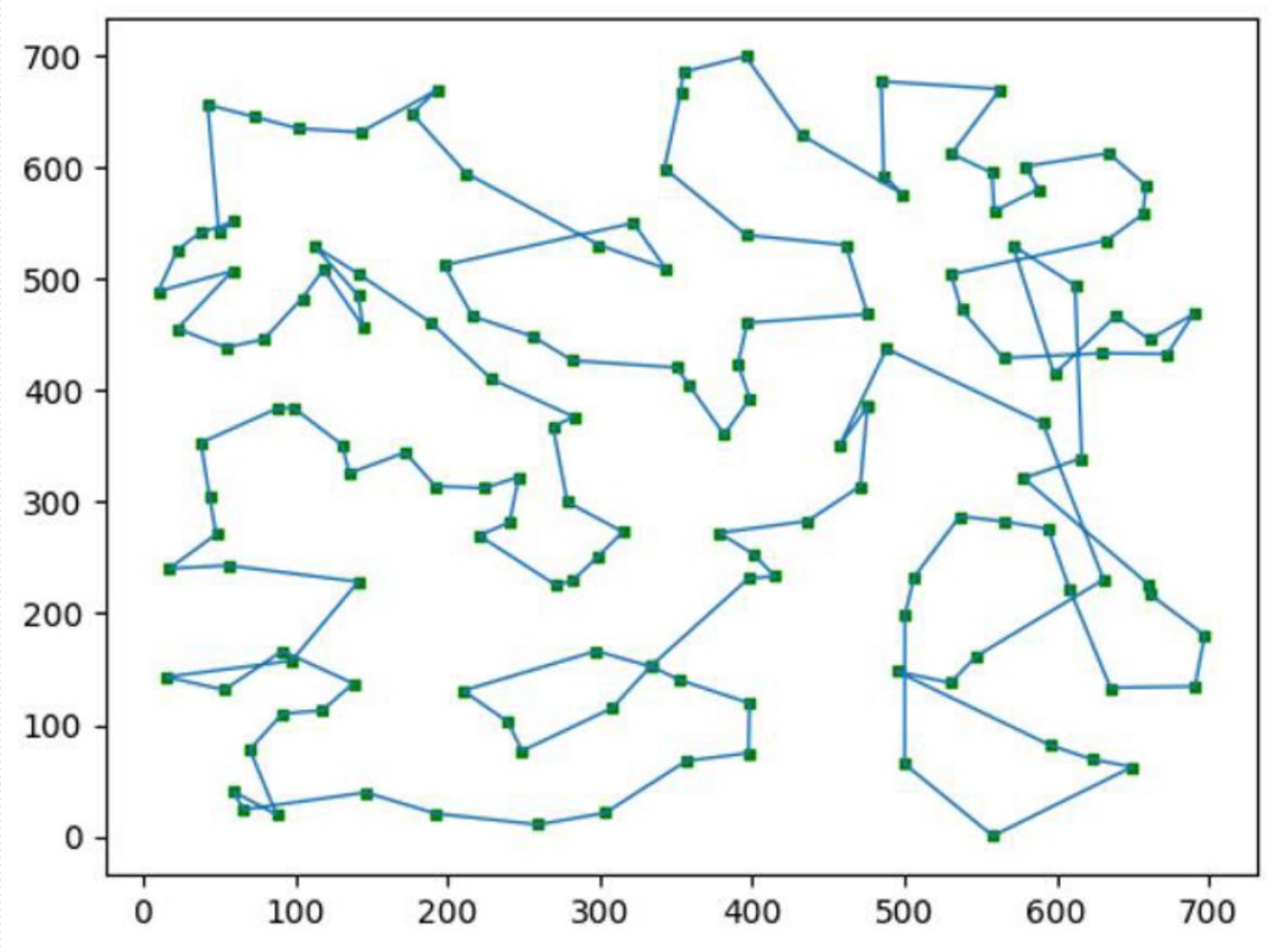
往届学生作品演示

- 路径可视化



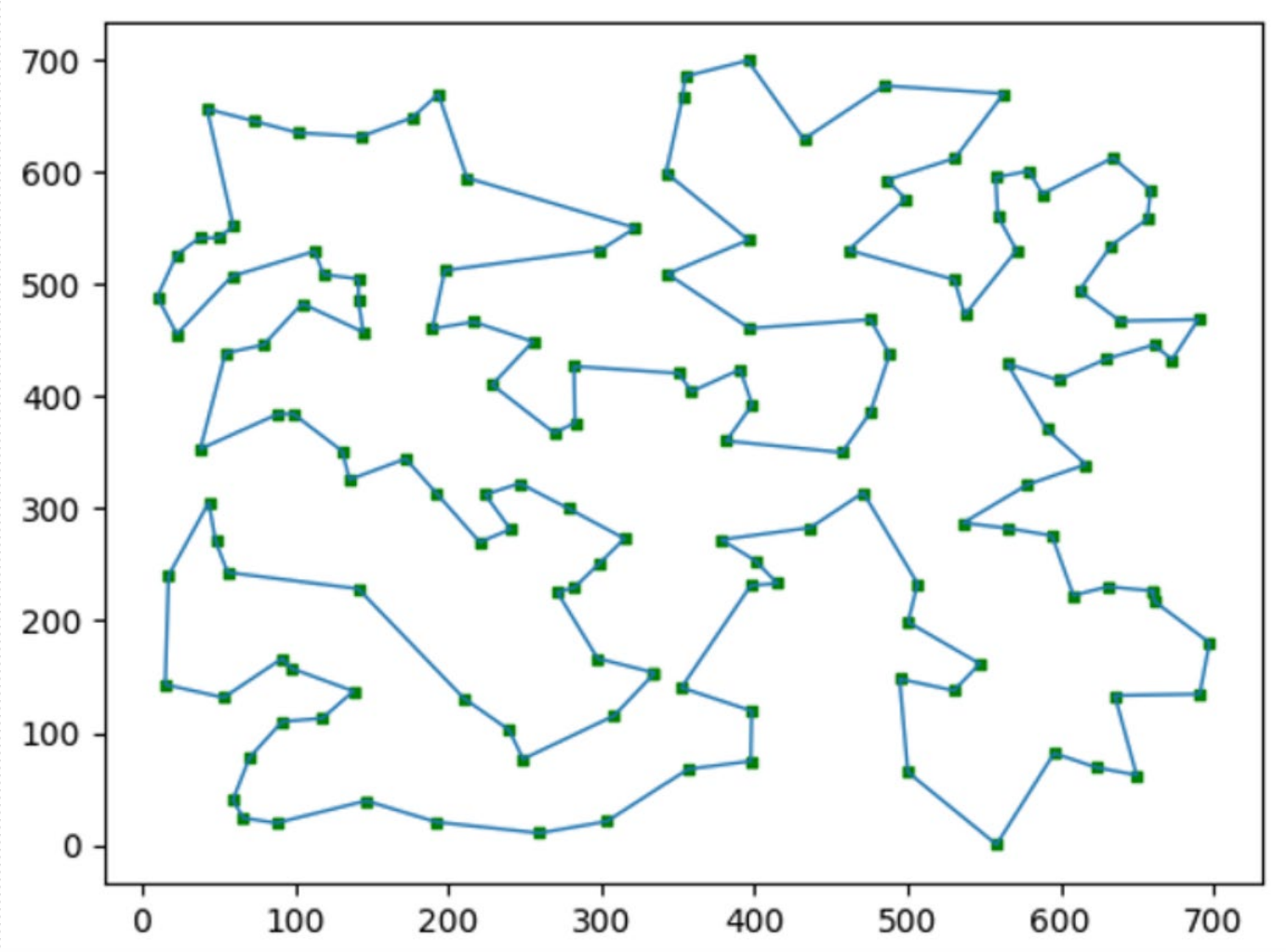
往届学生作品演示

- 路径可视化



往届学生作品演示

- 路径可视化



往届学生作品演示

- 模拟退火收敛曲线