模拟退火应用举例

- 例题　已知背包的装载量为c=8，现有n=5个物品，它们的重量和价值分别是(2, 3, 5, 1, 4)和(2, 5, 8, 3, 6)。试使用模拟退火算法求解该背包问题，写出关键的步骤。

- 求解：假设问题的一个可行解用0和1的序列表示，例如i=(1010)表示选择第1和第3个物品，而不选择第2和第4个物品。用模拟退火算法求解关键过程如图所示：

模拟退火应用举例

已知：
物体个数：n=5
背包容量：c=8
重量 w = (2, 3, 5, 1, 4)
价值 v = (2, 5, 8, 3, 6)

第一步：初始化。假设初始解为i=(11001)，初始温度为T=10。计算$f(i)$=2+5+6=13，最优解s=i

第三步：降温，假设温度降为T=9。如果没有达到结束标准，则返回第二步继续执行

假设在继续运行的时候，从当前解i=(10110)得到一个新解j=(00111)，这时候的函数值为$f(j)$=8+3+6=17，这是一个全局最优解。可见上面过程中接受了劣解是有好处的。

第 二步：在T温度下局部搜索，直到“平衡”， 假设平衡条件为执行了3次内层循环。
（2-1）产生当前解i的一个邻域解j（如何构造邻域根据具体的问题而定，这里假设为随机改变某一位的0/1值或者交换某两位的0/1值），假设j=(11100)
要注意产生的新解的合法性，要舍弃那些总重量超过背包装载量的非法解

（2-2）$f(j)$ = 2+5+8=15 > 13=$f(i)$，所以接受新解j
i=j; $f(i)$=$f(j)$=15; 而且s=i;
要注意求解的是最大值，因此适应值越大越优

（2-3）返回（2-1）继续执行。
(a) 假设第二轮得到的新解j=(11010)，由于$f(j)$ = 2+5+3=10 < 15=$f(i)$，所以需要计算接受概率$P(T)$=exp(($f(j)$-$f(i)$)/T) = exp(-0.5) = 0.607，假设random(0,1)>$P(T)$，则不接受新解

(b) 假设第三轮得到的新解j=(10110)，由于$f(j)$ = 2+8+3=13 < 15=$f(i)$，所以需要计算接受概率$P(T)$=exp(($f(j)$-$f(i)$)/T) = exp(-0.3) = 0.741，假设random(0,1)<$P(T)$，则接受新解
按照一定的概率接受劣解，也是跳出局部最优的一种手段

（2-4）这时候，T温度下的“平衡”已达到（即已经完成了3次的邻域产生），结束内层循环

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**.

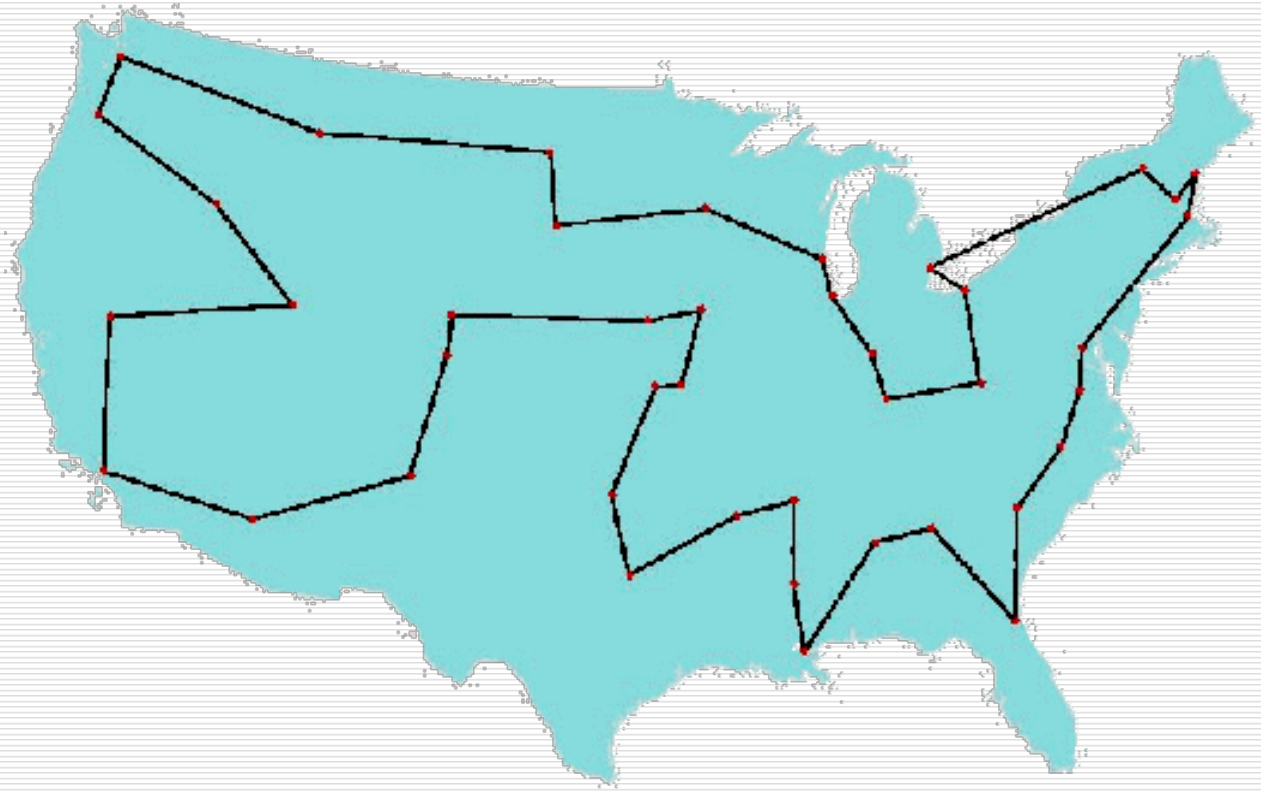


Fig. 1 George Dantzig, Ray Fulkerson, and Selmer Johnson (1954)
a description of a method for solving the TSP :49 cities

Traveling Salesman Problem

Existing Instances

49 city problem

- Dagtzig, G., D. Fulkerson and S. Johnson: “Solution of a large scale traveling salesman problems”, *Operations Research*, vol. 2, pp. 393-410, 1954.

120 city problem

- Grötschel, M.: “On the symmetric traveling salesman problem: solution of a 120 city problem”, *Mathematical Programming Studies*, vol. 12, pp. 61-77, 1980.

318 city problem

- Crowder, H. and M. Padberg: “Solving large scale symmetric traveling salesman problems to optimality”, *Management Science*, vol. 22, pp. 15-24, 1995.

532 city problem

- Padberg, M. and G. Rinaldi: “Optimization of 532 city symmetric traveling salesman problem by branch and cut”, *Operations Research Letters*, vol. 6, pp. 1-7, 1987.

Traveling Salesman Problem

- **666 city problem**
  - Grötschel, M. and O. Holland: "Solution of large scale symmetric traveling salesman problems", *Mathematical Programming Studies*, vol. 51, pp. 141-202, 1991.
- **2392 city problem**
  - Padberg, M. and G. Rinaldi: "A branch and cut algorithm for the resolution of large scale symmetric traveling salesman problem", *SIAM Review*, vol. 33, pp. 60-100, 1991.
- The earlier studies using the genetic algorithm to solve TSP
  - Grefenstette, J.: *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
- TSP has become a target for the genetic algorithm community.
  - Michalewicz, Z.: *Genetic Algorithm + Data structure = Evolution Programs*, 2nd ed., Springer-Verlag, New York, 1994.

Traveling Salesman Problem

- Notations
  - Indices

    i, j : the index of city, $i, j = 1, 2, \ldots, n$

  - Parameters

    n: the total number of cities

    d_{ij} : the distance city i to city j, *i.e.,* the distance of route (i,j); the distance matrix (d_{ij}) is symmetric.

  - Decision Variables

    x_{ij} : the 0,1 decision variable; 1, if route (i,j) is selected, and 0, otherwise.

Traveling Salesman Problem

Mathematical Model of TSP

$$\min \quad z = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(d_{ij}x_{ij}\right)$$

$$\text{s.t.} \quad \sum_{i=1}^{n} x_{ij} = 1, \qquad j = 1,2,3,\cdots,n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad i = 1,2,3,\cdots,n$$

$$x_{ij} = \begin{cases} 1, & \text{if route } (i,j) \text{ is selected} \\ 0, & \text{otherwise} \end{cases}$$

data set for non-directed graph

i	j	d_{ij}	i	j	d_{ij}
1	2	8	3	7	7
	3	5		8	12
	4	9		9	12
	5	12	4	5	3
	6	14		6	17
	7	12		7	10
	8	16		8	7
	9	17		9	15
2	3	9	5	6	8
	4	15		7	10
	5	17		8	6
	6	8		9	15
	7	11	6	7	9
	8	18		8	14
	9	14		9	8
3	4	7	7	8	8
	5	9		9	6
	6	11	8	9	11

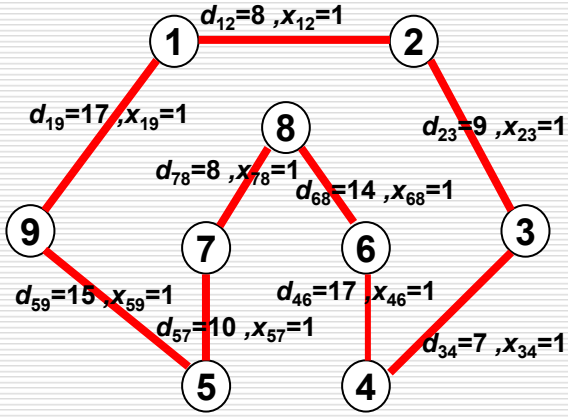
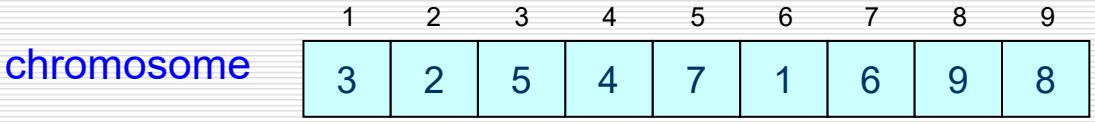


Fig. 2 Simple TSP Model

1 Representation

Permutation Representation

- This direct representation is perhaps the most natural representation of a TSP, where cities are listed in the order in which they are visited.



- This representation is also called a path representation or order representation.

procedure: Permutation Encoding
Input: city set, total number of cities *N*
output: chromosome *v*
begin
  for *j*=1 to *N*
    v(*j*)← *j*;
  for *i*=1 to $\lceil n/2 \rceil$
    repeat
      j←random[1, *N*];
      l←random[1, *N*];
    until *l*≠*j*
    swap(*v*(*j*), *v*(*l*));
  output chromosome *v*;
end

procedure: Permutation Decoding
Input: chromosome *v,*
  total number of cities *N*
output: tour list *L*
begin
  L← *φ*;
  for *i* =1 to *N*
    L ← *L* ∪*v*(*i*);
  output tour list *L*;
end

1 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.

	1	2	3	4	5	6	7	8	9
chromosome	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

2 Crossover Operators

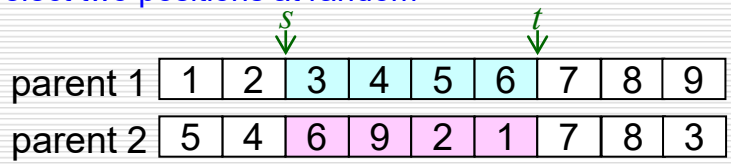
- During the past decade, several crossover operators have been proposed for permutation representation, such as **partial-mapped crossover** (PMX), **order crossover** (OX), **cycle crossover** (CX), **position-based crossover,** **order-based crossover**, **heuristic crossover**, and so on.
- These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

2 Crossover Operators

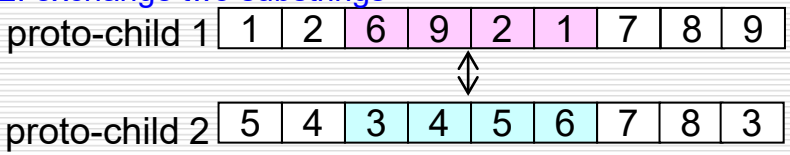
1. **Partial-Mapped Crossover** (PMX)

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

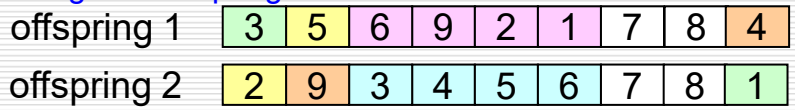


step 2: exchange two substrings



step 3 : determine the mapping relationship

step 4 : legalize offspring



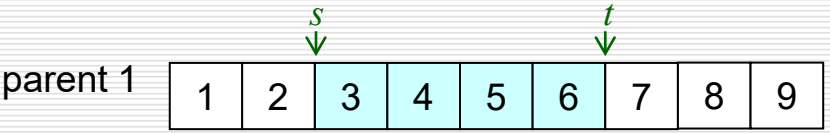
v_1 : parent chromosome 1 — v_2 : parent chromosome 2
l : length of chromosome — v_1' : offspring chromosome 1
v_2' : offspring chromosome 2 — R : relationships
s : start position of substring — t : end position of substring
relation(v_1,v_2) : searching relationship between v_1 and v_2
legalize(v_1,v_2,R)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

2 Crossover Operators

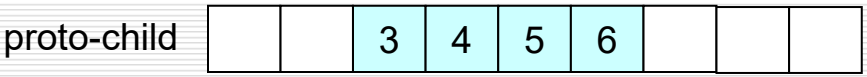
2. **OX crossover**

```
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;
```

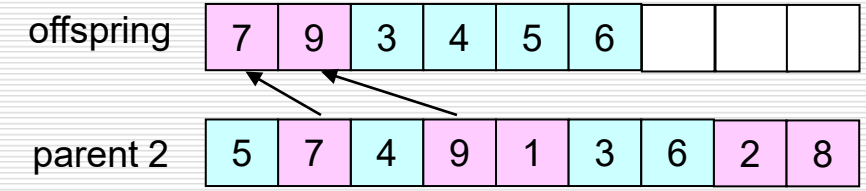
step 1 : select substring at random



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



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

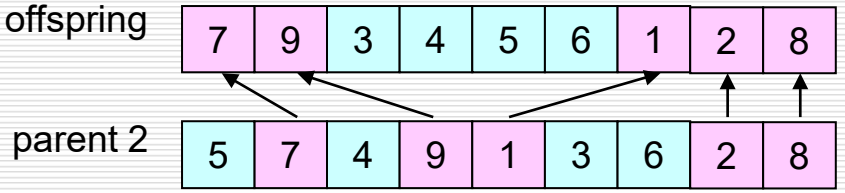


v_1 : parent chromosome 1	v_2 : parent chromosome 2
l : length of chromosome	v' : offspring chromosome
w: working data	fg : flag
s : start position of substring	t : end position of substring

2 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
            v'[i] ← v2[j] ;
            w ←j +1; break ;

output offspring v' ;
end;
```



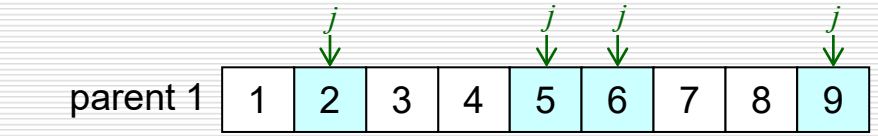
v_1 : parent chromosome 1	v_2 : parent chromosome 2
l : length of chromosome	v' : offspring chromosome
w: working data	fg : flag
s : start position of substring	t : end position of substring

2 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



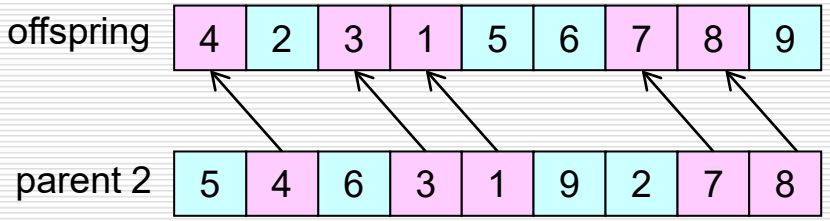
step 2 : produce proto-child by copying values of selected positions

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

2 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'[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

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

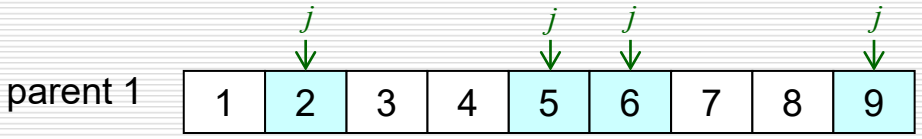
2 Crossover Operators

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

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

$S \leftarrow \emptyset$, $T \leftarrow \emptyset$, $w \leftarrow 1$;
// step 1: select a set of positions
from parent 1 at random
$N \leftarrow$ random[1:l] ;
for i=1 **to** N
$j \leftarrow$ random[1:l];
$S \leftarrow S \cup v_1[j]$;

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'

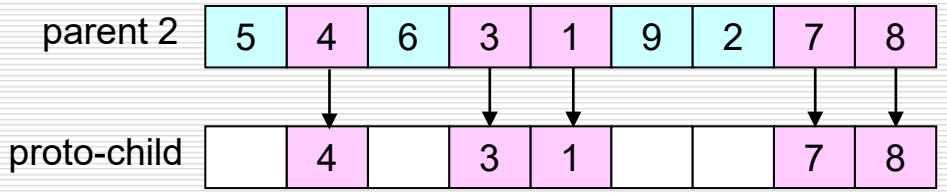
S ={$s[i]$}, i=1,2,…, N : genes value set of selected positions

fg_1 : flag 1 fg_2 : flag 2
w : working data

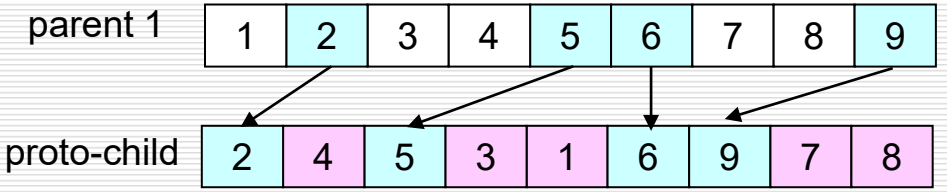
2 Crossover Operators

```
// 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 v';
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

$T=\{t[j]\}$, $j=1,2,\ldots, N$: positions set of assigned genes from v_2 to v'

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

fg_1 : flag 1 — fg_2 : flag 2

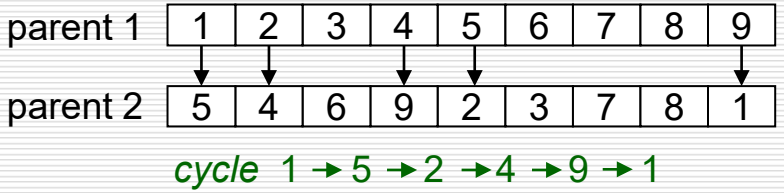
w : working data

2 Crossover Operators

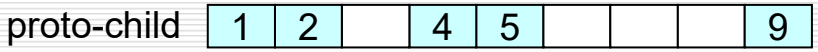
5. **Cycle Crossover** (CX)

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 ← $S \cup v_1[i]$;

step 1 : find the cycle between parents



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



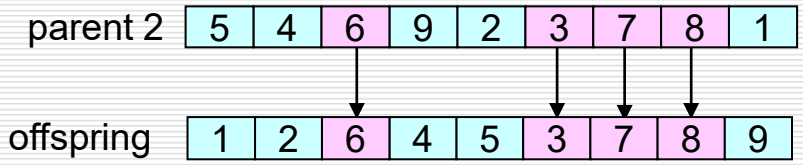
v_1 : parent chromosome 1 v_2 : parent chromosome 2
l : length of chromosome v' : offspring chromosome
N : total number of values in cycle

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

2 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 ;
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

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**

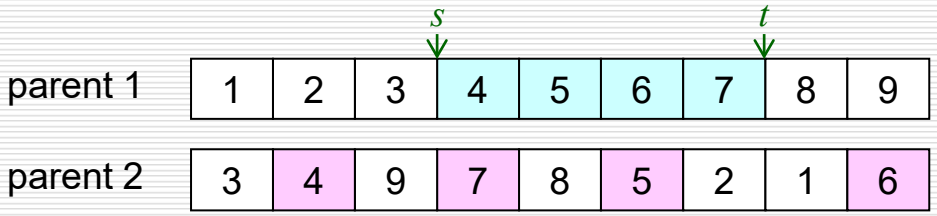
$S \leftarrow S \cup v_2[i]$;

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

2 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

3 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

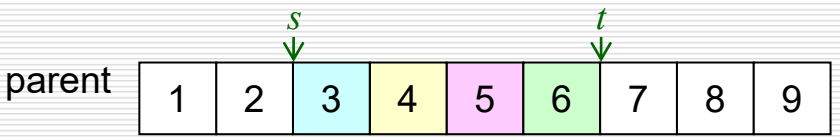
$S \leftarrow$ invert(v [s: t]) ;

$v' \leftarrow v$ [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

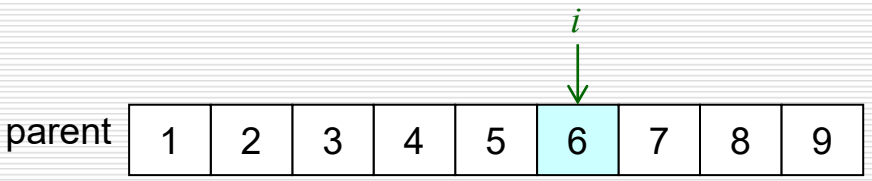
v : parent chromosome
v' : offspring chromosome
t : end position of substring
invert(*string*) : inversely changing order of *string*
l : length of chromosome
s : start position of substring
S : inverse string of substring

3 Mutation Operators

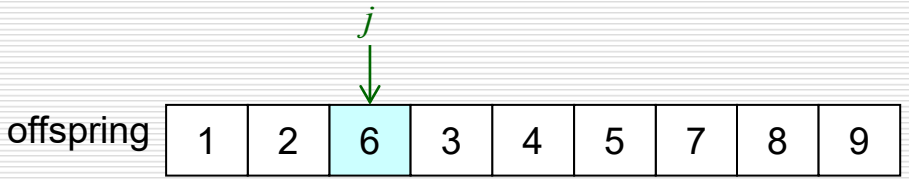
2. **Insertion Mutation**

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

3 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] ;

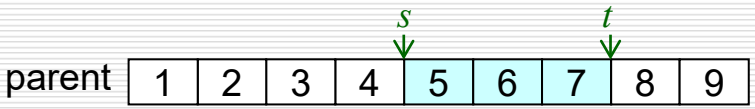
$W \leftarrow v$ [1:s-1] // v [t+1:l];

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

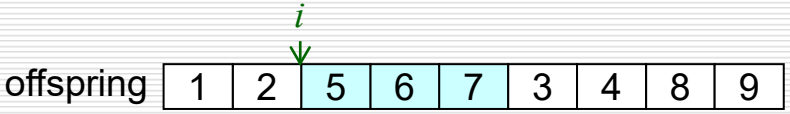
output offspring v';

end

step 1 : select subtour



step 2 : insert subtour in a random position



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

3 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

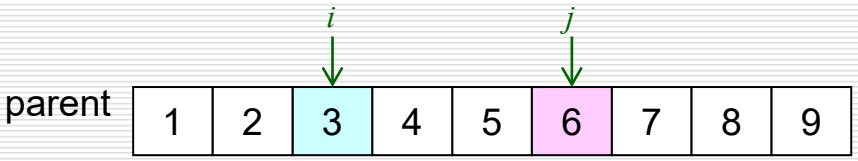
$v' \leftarrow v$ [1: i -1] // v [j] // v [i +1 : j -1]// v [i] // v [j +1 : l];

v'

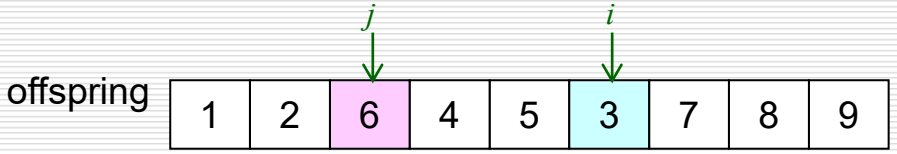
output offspring ;

end

step 1: select two position at random



step 2 : produce offspring by swapping selected positions



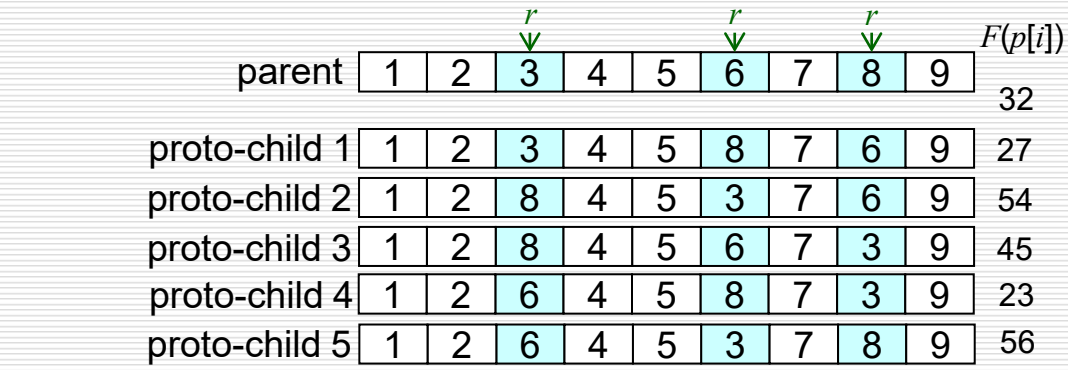
v : parent chromosome
v' : offspring chromosome
j : selected position
l : length of chromosome
i : selected position

3 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



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

4 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
    $t \leftarrow 0$;
    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 $C(t)$;
        $t \leftarrow t+1$;
    end
    output best tour route;
end