PseudoCode

Crossover

1. Partial mapped Crossover (PMX)

```
// i1, i2 cut locations
// m[] partial map, m[i] => the mapping target of i
PMX (p1, p2, c1, c2)
i1←Random(n); i2←Random(n)
if i1 > i2 swap i1 and i2
m[i] ←-1, for all i, i = 0, 1, …, n-1
for i ←  i1 to i2-1
  if p1[i] = p2[i] then next i // no mapping
  if m[p1[i]] = -1 and m[p2[i]] = -1
    m[p1[i]] ←  p2[i]; m[p2[i]] ←  p1[i]
  else if m[p1[i]] = -1
    m[p1[i]] ←  m[p2[i]]; m[m[p2[i]]] ←  p1[i]; m[p2[i]] ←  -2
  else if m[p2[i]] = -1
    m[p2[i]] ←  m[p1[i]]; m[m[p1[i]]] ←  p2[i]; m[p1[i]] ←  -2
  else
    m[m[p2[i]]] ←m[p1[i]]; m[m[p1[i]]] ←  m[p2[i]]
    m[p1[i]] ←  -3; m[p2[i]] ←  -3
for i ←  0 to n-1
  if i1 <= i and i < i2
      c1[i] ←  p2[i]; c2[i] ←  p1[i]
  else
    if m[p1[i]] < 0 then c1[i] ←  p1[i]
    else c1[i] ←  m[p1[i]]
    if m[p2[i]] < 0 then c2[i] ←  p2[i]
    else c2[i] ←  m[p2[i]]
```

2. Order Crossover (OX)

```
// i1, i2 cut locations
// List<int>F, List<int>M are empty list
OX (p1, p2, c1, c2)
Clear F; Clear M
i1←Random(n); i2←Random(n)
if i1 > i2 swap i1 and i2
c1[i] ←-1, for all i, i = 0, 1, …, n-1
c2[i] ←-1, for all i, i = 0, 1, …, n-1
for i ←  0 to n-1
  if i1 <= i and i < i2
      c1[i] ←  p1[i]; c2[i] ←  p2[i]
      F.append(p1[i]); M.append(p2[i])
for i ←  0 to n-1
  if c1[i] = -1
    for j ←  0 to n-1
        if p2[j] is in F then continue // next j
        else c1[i] ←  p2[j]; F.append(p2[j]) then break //next i
  if c2[i] = -1
    for j ←  0 to n-1
        if p1[j] is in M then continue // next j
        else c2[i] ←  p1[j]; M.append(p1[j]) then break //next i
```

3. Position based Crossover

```
// List<int>F, List<int>M, List<int>temp are empty list
// p[] positions, p[i] => the random position of crossover of i
PBX (p1, p2, c1, c2)
Clear F; Clear M; Clear temp
NumOfCrossover ←Random(1, NumOfGenes)
// 1 <= NumOfCrossover < NumOfGenes
c1[i] ←-1, for all i, i = 0, 1, …, n-1
c2[i] ←-1, for all i, i = 0, 1, …, n-1
for i ←  0 to NumOfCrossover-1
    p[i]  ←  Random(0, NumOfGenes)
    while (p[i] is in temp) then p[i]  ←  Random(0, NumOfGenes);
    temp.append(p[i])
p.Sorted;
for i ← 0 to NumOfCrossover-1
    c1[p[i]] ←  p1[p[i]]; c2[p[i]] ←  p2[p[i]]
    F.append(p1[p[i]]); M.append(p2[p[i]])
for i ←  0 to n-1
  if c1[i] = -1
    for j ←  0 to n-1
        if p2[j] is in F then continue // next j
        else c1[i] ←  p2[j]; F.append(p2[j]) then break //next i
  if c2[i] = -1
    for j ←  0 to n-1
        if p1[j] is in M then continue // next j
        else c2[i] ←  p1[j]; M.append(p1[j]) then break //next i
```

4. Order based Crossover

```
// List<int>temp are empty list
// gv[] gene value, gv[i] => the gene value of i
OBX (p1, p2, c1, c2)
Clear temp;
NumOfCrossover ←Random(1, NumOfGenes)
// 1 <= NumOfCrossover < NumOfGenes
c1[i] ←-1, for all i, i = 0, 1, …, n-1
c2[i] ←-1, for all i, i = 0, 1, …, n-1
for i ←  0 to NumOfCrossover-1
    gv[i]  ←  Random(0, NumOfGenes)
    while (gv[i] is in temp) then gv[i]  ←  Random(0, NumOfGenes);
    temp.append(gv[i])
for i ← 0 to NumOfGenes -1
    if p2[i] is not in temp then c1[i] ←p2[i]; continue // next i
    for j ← 0 to NumOfGenes -1
      for k ← 0 to NumOfCrossover -1
        if p1[j] = gv[k] then c1[i] ←  gv[k] then break//
      break; // next i
for i ← 0 to NumOfGenes -1
    if p1[i] is not in temp then c2[i] ←p1[i]; continue // next i
    for j ← 0 to NumOfGenes -1
      for k ← 0 to NumOfCrossover -1
        if p2[j] = gv[k] then c2[i] ←  gv[k] then break//
      break; // next i
```

5. Cycle Crossover (CX)

```
// List<int>temp are empty list
CX(p1, p2, c1, c2)
Clear temp;
position ← Random(0, NumOfGenes)
change_position = position
temp.append(p1[position])
do
    if temp does not contain p2[position]
        temp.append(p2[change_position])
        for i←0 to n
          if p1[i] = p2[change_position]
            change_position ←i;
            break;
while p2[change_position] != p1[position]
for i ← 0 to NumOfGenes - 1
  if p1[i] is in temp
    child1[i] ← p1[i]
    child2[i] ← p2[i]
  else
    child1[i] ← p2[i]
    child2[i] ← p1[i]
```

6. Subtour Exchange Crossover

```
// i1 is start position, i2 is end position
// List<int>temp are empty list
SEX (p1, p2, c1, c2)
Clear temp;
NumOfCrossover ← 4 // define subtour length
i1 ← NumOfCrossover – 1; i2 ← i1 + 4
c1[i] ←-1, for all i, i = 0, 1, …, n-1
c2[i] ←-1, for all i, i = 0, 1, …, n-1
for i ← i1 to i2-1
    temp.append(p1[i])
for i ← 0 to NumOfGenes -1
    if p1[i] is not in temp then c1[i] ←p1[i]; continue // next i
    for j ← 0 to NumOfGenes -1
      for k ← 0 to NumOfCrossover -1
        if p2[j] is in temp then c1[i] ← p2[j] then break//
    break; // next i
for i ← 0 to NumOfGenes -1
    if p2[i] is not in temp then c2[i] ←p2[i]; continue // next i
    for j ← 0 to NumOfGenes -1
      for k ← 0 to NumOfCrossover -1
        if p1[j] is in temp then c2[i] ← p1[j] then break//
    break; // next i
```

Mutate

1. Inversion Mutation

```
// i1 is start position, i2 is end position
// List<int>temp are empty list
Inversion(p1, c1)
i1 ← Random(n)
i2 ← Random(n)
if i1 > i2 swap i1 and i2
for i ← i1 to i2-1
    temp.append(p1[i])
for i ← 0 to n-1
    if i < i1 or i >= i2
        c1[i] ← p1[i]
    else
        c1[i] ←p1[i2-1]; i2 = i2-1
```

2. Insertion Mutation

```
// i1, selected position
// i2, insert position
Insertion (p1, c1)
i1 ←Random(n); i2 ←Random(n)
if i1 = i2 then i2 ←  Random(n)
for i ←0 to n-1
  if i2 > i1
    if i >= i1 and i < i2 then c1[i] ←  p1[i+1]
    else if i = i2 then c1[i] ←  p1[i1]
    else c1[i] ← p1[i]
  else if i1 > i2
    if i = i2 then c1[i] ←  p1[i1]
    else if i > i2 and i <= i1 then c1[i] ←  p1[i-1]
    elsec1[i] ←  p1[i]
```

3. Reciprocal Exchange Mutation

```
// i1, i2 mutated positions
ReciprocalExchange(p1, c1)
i1 ← Random(n); i2 ← Random(n)
if i1 = i2 then i2 ← Random(n)
for i ←0 to n-1
  if i =i1 then c1[i]←p1[i2]
  else if i =i2 then c1[i]←p1[i1]
  else c1[i] ← p1[i]
```

4. Displacement Mutation

```
// i1 is start position, i2 is end position
// List<int>temp are empty list
Displacement(p1, c1)
Clear temp
i1 ← Random(n)
i2 ← Random(n)
if i1 > i2 swap i1 and i2
for i ← 0 to i2-i1
    add p1 [i1+i] to temp
position ← Random(n) until != i1
for i ← 0 to n
  if i1 < position
    if i < i1 or i >= position + (i2-i1)
        c1[i] ← p1[i]
    else if i >= i1 and i<position
        c1[i] ← p1[i+(i2-i1)]
    else
        c1[i] ← temp[i-position]
  else if i1>position
    if 1>i2 or 1<position
        c1[i] ← p1[i]
    else if i>=position+(i2-i1) and i<i2
        c1[i] = p1[i-(i2-i1)]
    else
        c1[i] ← temp[i-position]
```