# APPENDIX A:

## TSP SOLVER USING GENETIC ALGORITHM

# The outline of the GA for TSP

The algorithm consists of the following steps:

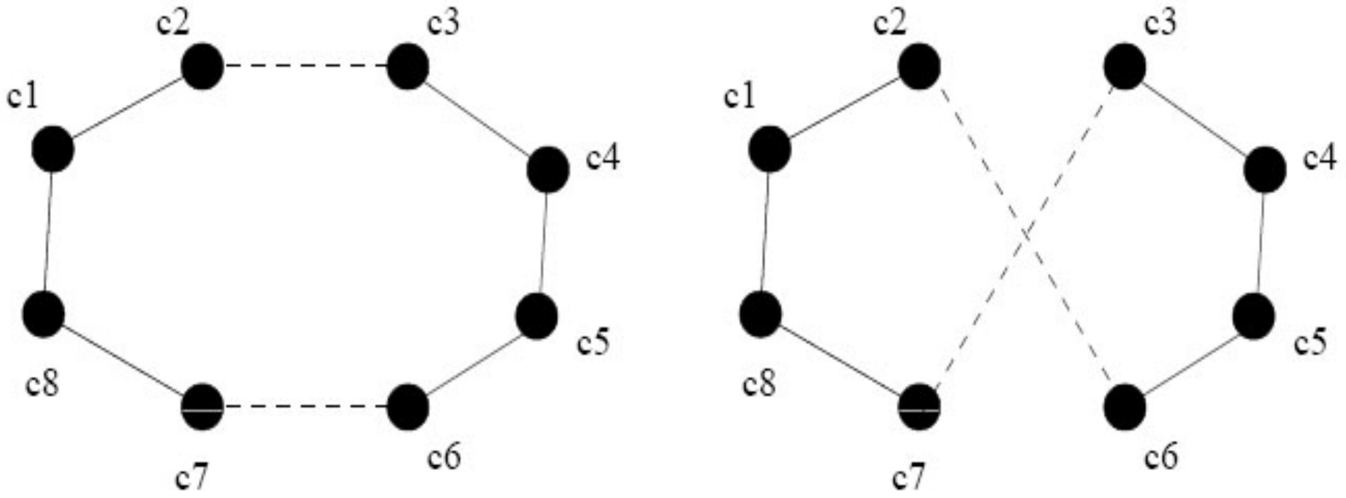
- **Initialization**: Generation of M individuals randomly.
- **Natural Selection**: Eliminate $p_1$% individuals. The population decreases by $M.p_1/100$.
- **Multiplication**: Choose $M.p_1/100$ pairs of individuals randomly and produce an offspring from each pair of individuals (by crossover). The population reverts to the initial population $M$.
- **Mutation by 2-opt**: choose $p_2$% of individuals randomly and improve them by the 2-opt method. The elite individual (the individual with the best fitness value in the population) is always chosen. If the individual is already improved, do nothing.

# Representation

- We use the *path representation* for solution coding.
- EX: the chromosome g = (D, H, B, A, C, F, G, E) means that the salesperson visits D, H, B, A,…E successively, and returns to town D.

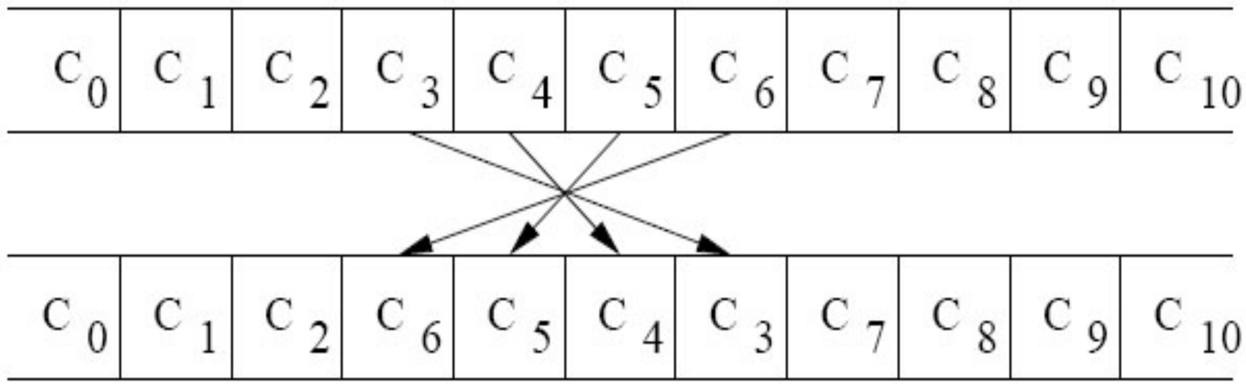
# Mutation by 2-opt

- The **2-opt** is one of the most well-known local search operator (move operator) among TSP solving algorithms.
- It improves the tour edge by edge and reverses the order of the subtour.
- When we apply the 2-opt mutation to a solution, the solution may fall into a local mimimum.




(a) The current tour.

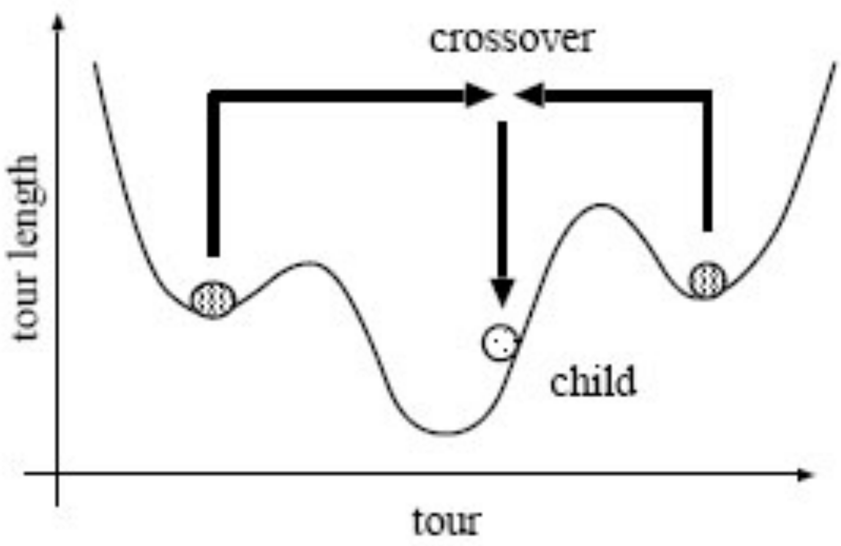
(b) The proposed tour by 2OPT.



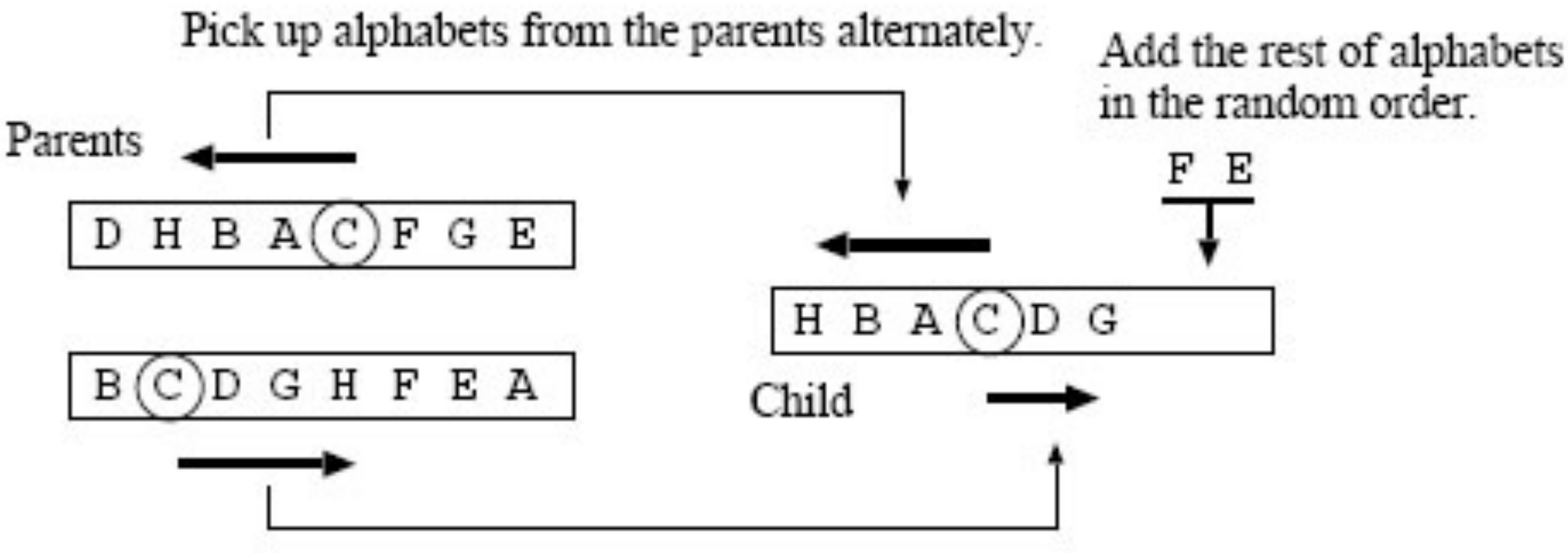

(c) The array structure.

# Crossover operator

- Greedy Subtour Crossover (GSX). It acquires the longest possible sequence of parents' subtours.
- Using GSX, the solution can pop up from local minima effectively.



# Greedy Subtour Crossover




**Inputs**: Chromosomes $g_a$ = (D, H, B, A, **C**, F, G, E) and

$g_b$ = (B, **C**, D, G, H, F, E, A).

**Outputs**: The offspring $g$ = (H, B, A, **C**, D, G, F, E)

**Algorithm**: Greedy Subtour Crossover
**Inputs**: Chromosomes $g_a = (a_0, a_1, \ldots, a_{n-1})$ and $g_b = (b_0, b_1, \ldots, b_{n-1})$.
**Outputs**: The offspring chromosome *g*.

```
procedure crossover(g_a, g_b){
    fa ← true
    fb ← true
  choose town t randomly
  choose x, where a_x = t
  choose y, where b_y = t
  g ← t
  do {
      x ← x -1 (mod n),
      y ← y + 1 (mod n).
      if f_a = true then {
          if a_x ∉ g then
                g ← a_x.g
          else  f_a ← false
      }
      if f_b = true then {
         if b_x ∉ g then
                g ← g.b_y
          else  f_b ← false
      }
  } while f_a = true or f_b = true
  if |g| < g_a| then {
     add the rest of towns to g in the random order
  }
  return g
}
```

# Example:

- Suppose that parent chromosomes $g_a$ = (D, H, B, A, C, F, G, E) and $g_b$ = (B, C, D, G, H, F, E, A).
- First, choose one town at random, say, town C is chosen. Then $x$ = 4 and $y$ =1. Now the child is (C).
- Next, pick up towns from the parents alternately. Begin with $a_3$ (A) and next is $b_2$ (D). The child becomes $g$ = (A, C, D).
- In the same way, add $a_2$ (B), $b_3$ (G), $a_1$ (H), and the child becomes $g$ = (H,B,A,C,D,G).
- Now the next town is $b_4$ = H and H has already appeared in the child, so we can't add any more towns from parent $g_b$.
- Now, we add towns from parent $g_a$. The next town is $a_0$ = D, but D has already used. Thus, we can't add towns from parent $g_a$, either.
- Then we add the rest of the towns, i.e., E and F, to the child in the random order. Finally the child is $g$ = (H,B,A,C,D,G,F,E).

# Survivor Selection

- Eliminate $R = M.p_1/100$. We eliminate similar individuals to maintain the diversity in order to avoid immature convergence.
- First, sort the individuals in fitness-value order. Compare the fitness value of adjoining individuals. If the difference is less than $\varepsilon$, eliminate preceding individual while the number of eliminated individuals ($r$) is less than $R$.
- Next, if $r < R$, eliminate $R - r$ individuals in the order of lowest fitness value.

# Other parameters

- Test data: TSPLIB
- gr96.data  (666 cities)
  - Population size: 200
  - maximum number of generations: 300
  - $p_1$ = 30%
  - $p_2$ = 20%
  - produces the minimum solution
- <u>Conclusion</u>: The solver brings out good solution very fast since GA here utilizes heuristic search (2-opt) in genetic operators. It is a hybrid algorithm.

# Reference

- H. Sengoku, I. Yoshihara, “A Fast TSP Solver Using GA on Java”, 1997.