UNIT 12 APPLICATIONS OF GENETIC ALGORITHM

Structure **Page No**

12.1 INTRODUCTION

In this Unit we present an example of genetic algorithms tailored to a benchmark problem for a graph based application for optimization in Sec. 12.2. The travelling salesman problem (TSP) is a special problem which we shall discuss in Sec. 12.3. There are many reasons why it is considered as the mother of all problems. First of all, the TSP is conceptually very simple: the travelling salesman must visit every city in his territory exactly once and then return to the starting point. The path the salesman takes is called a tour. Given the cost of travel between all cities, how should he plan his itinerary for minimum total cost of the entire tour? The search space for the TSP is a set of permutations of given n cities. Any single permutation of n cities is a potential solution (which is a complete tour of n cities). The optimal solution is a permutation which yields the minimum cost of the tour. The size of the search space is n!. The TSP is a relatively old problem: it was documented as early as 1759 by Euler (though not by that name), whose interest was in solving the knights' tour problem. A correct solution would have a knight visit each of the 64 squares of a chessboard exactly once in its tour.

Objectives

After studying this unit you will be able to:

- recall the simple genetic algorithm;
- apply genetic algorithm for travelling salesman problem;
- apply crossover for TSP to find the solution.

Let us begin this section by revisiting the simple genetic algorithm.

12.2 REVISITING SIMPLE GENETIC ALGORITHM (SGA)

ALGORITHM

```
   BEGIN
        Generate initial population;
        Compute fitness of each individual;
        REPEAT /* New generation /*
          FOR population_size / 2 DO
            Select two parents from old generation;
             /* biased to the fitter ones */
        Recombine parents for two offspring;
        Compute fitness of offspring;
        Insert offspring in new generation
      END FOR
    UNTIL population has converged
END
```

GA COMPONENTS

- **Generate initial population;**
- **Fitness Function;**
- **Production (Recombine) operators**
  - **Selection**
  - **Crossover**
  - **Mutation**
  - **Inversion**
- **Insert offspring in new generation**

Let us first discuss the following example.

Example 1: Maximize $f(x) = x^2$ subject to $0 \leq x \leq 31$

Solution: GA APPROACH

- Representation: binary code, e.g. 01101 = 13
- Population size: 4
- Random initialisation
- 1-point crossover, bitwise mutation
- Roulette wheel selection

PARAMETER SETTING

- Initialization:
  - Population size
  - Random
  - Dedicated greedy algorithm
- Reproduction:
  - Generational: as described before (insects)
  - Generational with elitism: fixed number of most fit individuals are copied unmodified into new generation
  - Steady state: two parents are selected to reproduce and two parents are selected to die; two offspring are immediately inserted in the pool (mammals)
- Stop criterion:
  - Number of new chromosomes
  - Number of new and unique chromosomes
  - Number of generations
- Measure:
  - Best of population
  - Average of population
- Duplicates
  - Accept all duplicates
  - Avoid too many duplicates, because that degenerates the population (inteelt)
  - No duplicates at all

We show one generational cycle done by hand

Table 1

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	058	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

Now try the following exercises

E1 Maximize f(x, y) = 4x + 3y
Subject to

$$2x + 3y \le 6$$
$$-3x + 2y \le 3$$
$$2x + y \le 4$$
$$0 \le x \le 2$$

using genetic algorithm.

E2) Minimize $f(x) = x^2$
Subject to $1 \le x \le 51$

So far we discussed the simple genetic algorithm with its application. In the following section we shall discuss another important application of genetic algorithm, known as traveling salesman problem (TSP). TSP is a special type of scheduling problem.

12.3 TRAVELLING SALESMAN PROBLEM

The term 'travelling salesman' was first used in a 1932 German book, *The travelling salesman, how and what he should do to get commissions and be successful in his business*, written by a veteran travelling salesman. The TSP was introduced by the RAND Corporation in 1948. The Corporation's reputation helped to make the TSP a well known and popular problem. The TSP also became popular at that time due to the new subject of linear programming and attempts to solve combinatorial problems. The Travelling Salesman Problem was proved to be NP-hard. It arises in numerous real life applications as found in the literature where the number of nodes (cities) to be connected are very large such as, Circuit board drilling applications with up to 17,000 cities, X-ray crystallography instances with up to 14,000 cities, VLSI fabrication have been reported with as many as 1.2 million cities. Therefore, hours on a multi-million dollar super computer (due to the large number of cities) for the best solution may not be cost-effective if it be possible to obtain a good (near-optimal) solution within a few seconds on a PC. In the TSP, the goal is to find the shortest distance between n different cities. Testing every possibility for an n city tour would be n! math additions. A 30 city tour would have to measure the total distance of be 2.65×10^{32} different tours. Assuming a comforting speed of trillion additions per second, this would take 252,333,390,232,297 years. Adding one more city would cause the time to increase by a factor of 31. Obviously, this is an impossible solution. This indicates the need for a heuristic approach/method to yield the acceptable solution in good time. A genetic algorithm can be used to find a solution in much less time. Although it might not find the best solution, it can find a near perfect solution for a 100 city tour in less than a minute.

During the last decades, several algorithms emerged to approximate the optimal solution: nearest neighbour, greedy algorithm, nearest insertion, farthest insertion, double minimum spanning tree, strip, space-filling curve, algorithms by Karp, Litke, Christofides, etc., some of these algorithms assume that the cities correspond to points in the plane under some standard metric. Another group of algorithms B-opt, 3-opt, etc., aim at a local optimization: an improvement of a tour by local perturbations. The TSP also became a target for the GA community: several genetic-based algorithms were reported for optimization problems. These algorithms aim at producing near-optimal solutions by maintaining a population of potential solutions which undergoes some unary and binary transformations ('mutations' and 'crossovers') under a selection scheme biased towards fit individuals. It is interesting to compare these approaches, paying particular attention to the representation and genetic operators used – this is what we intend to do in this Unit.

Evolution of genetic algorithms for TSP has been traced. The other objective of this Unit is to help you in coding (representing as chromosome) the solution. This involves determining the length of the vector/chromosome, determining significance of each bit position and thus representation of the solution itself. The Unit would also enable you to write programmes for various components of a genetic algorithm and thus to help you in writing genetic algorithms to solve problems.

To underline some important characteristics of the TSP, it is useful to consider the CNF-satisfiability problem first. A logical expression in conjunctive normal form (CNF) is a sequence of clauses separated by the Boolean operator $\wedge$ (and/conjunction); a clause is a sequence of literals separated by the Boolean operator $\vee$ (or/disjunction); a literal is a logical variable or its negation; a logical variable is a variable that may be assigned values TRUE or FALSE (1 or 0). For example, the following logical expression is in CNF:

$$(a \vee \sim b \vee c) \wedge (b \vee c \vee d \vee \sim e) \wedge (\sim a \vee c) \wedge (a \vee \sim c \vee \sim e),$$

where, a, b, c, d, and e are logical variables; ~a denotes the negation of variable a (~a has the value TRUE if and only if a has the value FALSE). The problem is to determine whether there exists a truth value assignment for the variables in the expression, so that the whole expression evaluates to TRUE. For the above CNF logical expression there exist several truth assignments, for which the whole expression evaluates to TRUE, e.g., any assignment with a = TRUE and c = TRUE irrespective of the truth values of b, d and e.

If we try to apply a genetic algorithm to the CNF-satisfiability problem, we notice that it is hard to imagine a problem with better suited representation: a binary vector of fixed length (the length of the vector corresponds to the number of variables) should do the job. Moreover, there are no dependencies between bits: any change would result in a meaningful vector representing a potential solution of the problem.

Thus we can apply mutations and crossovers without any need for decoders or repair algorithms. However, for a problem such as this one designing the evaluation function is the hardest task. Note that all logical expressions evaluate to TRUE or FALSE, and if a specific truth assignment evaluates the whole expression to TRUE, then the solution to the problem is found. The point is that during the search for a solution, all chromosomes (vectors) in a population would evaluate to FALSE (unless a solution is found), so it is impossible to distinguish between *good* and *bad* chromosomes. In short, the CNF-satisfiability problem has natural representation and operators, without any natural evaluation function.

On the other hand, the TSP has an extremely easy (and natural) evaluation function: for any potential solution (a permutation of cities), we can refer to the table providing distances between cities and (after n–1 addition operations) it is possible to obtain the total length of the tour. Thus, in a population of tours, it is easily possible to compare any two of them. However, there is no rule of thumb for the choice of the representation of a tour, choice of operators and for setting the parameters.

The basic steps to solving the traveling salesman problem using a GA are as below,

- First, create a group of many random tours in what is called a **population**. The algorithm uses a greedy initial population that gives preference to linking cities that are close to each other.
- Second, **select** 2 of the better (shorter) tours as **parents** in the population and combine (**crossover**) them to make 2 new **child** tours. Hopefully, these children tour will be better than either of the parents.
- A small percentage of the times, the child tours are **mutated**. This is done to prevent all tours in the population from looking identical.
- The new child tours are inserted into the population replacing two of the longer tours. The size of the population remains the same.
- New children tours are repeatedly created until the desired goal is reached.

As the name implies, Genetic Algorithms mimic nature and evolution using the principles of **Survival of the Fittest**. The two complex issues with using a Genetic Algorithm to solve the Traveling Salesman Problem are the **encoding** of the tour and the **crossover** algorithm that is used to combine the two parent tours to make the child tours.

Ordering or sequencing problems form a special type of problems whose solutions cannot be coded by a binary representation. TSP involves ordering of the cities so as to meet the objective of finding a route, in fact a shortest route. Here the task is to arrange the cities in a certain order where important thing is which elements occur next to each other (adjacency). Therefore a random list of cities need not be a valid tour. The list of connected cities may only provide a valid option for a solution. Therefore in such cases the solution is generally expressed as a permutation. If there are *n* cities then the representation is as a list of *n* cities, each of which occurs exactly once.

- Given n cities
- Find a complete tour with minimal distance or cost (as provided in a table)

Encoding:

- Label the cities 1, 2, … , *n*
- One complete tour is one permutation
- For n = 7

  Tours: [F, A, B, E, C, G, D], [D, E, A, C, G, B, F], etc.

The search space for this problem is BIG for a significant number of cities. For 30 cities there are $30! \approx 10^{32}$ possible tours.

Crossover as described earlier is performed by identifying a random point in the two parent sequences and switching every number in the sequence after that point.

Example 2: Consider a TSP involving 7 cities. The crossover point is between the 3rd and 4th cities in the list. To create the children, every item in the parent's sequence after the crossover point is swapped.

Parent 1 F A B | E C G D
Parent 2 D E A | C G B F
Child 1 F A B | C G B F
Child 1 D E A | E C G D

Another difficulty with the Traveling Salesman Problem is that every city can only be used once in a tour. If the letters in the above example represented cities, this child tours created by this crossover operation would be invalid. Child 1 goes to city F & B twice and never goes to cities D or E.

The encoding cannot simply be the list of cities in the order they are traveled. Other encoding methods have been created that solve the crossover problem. Although these methods will not create invalid tours, they do not take into account the fact that the tour "A B C D E F G" is the same as "G F E D C B A". To solve the problem properly the crossover algorithm will have to get much more complicated.

One way to deal with this issue could be to store the **links** in both directions for each tour. In the above tour example, Parent 1 would be stored as given in Table 2.

Table 2

City	First Connection	Second Connection
A	F	B
B	A	E
C	E	G
D	G	F
E	B	C
F	D	A
G	C	D

The crossover operation for TSP is more complicated than combining 2 strings. The crossover will take every link that exists in both parents and place those links in both children. Then, for Child 1 it alternates between taking links that appear in Parent 2 and then Parent 1. For Child 2, it alternates between Parent 2 and Parent 1 taking a different set of links. For either child, there is a chance that a link could create an invalid tour where instead of a single path in the tour there are several disconnected paths. These links must be rejected. To fill in the remaining missing links, cities are chosen at random. Since the crossover is not completely random, this is considered a greedy crossover.

Eventually, this GA would make every solution look identical. Once every tour in the population is identical, the GA will not be able to find a better solution. There are two ways to address this problem. The first is to use a very large initial population so that it takes the GA longer to make all of the solutions the same. The second method is **mutation**, where some child tours are randomly altered to produce a new unique tour.

Normal mutation operator for permutation may lead to inadmissible solutions. Consider a bit-wise mutation. Let in a permutation gene i have value j. Then let the value of gene i be changed to some other value k would mean that k occurred twice and j no longer occurred. Therefore mutation must change at least two values. Mutation parameter then reflects the probability that some operator is applied once to the whole string rather than individually to the genes in each position. Therefore mutation for permutation is of various types:

- Insert mutation
- Swap mutation
- Inversion mutation, and
- Scramble mutation.

Insertion mutation:

- Pick two allele values at random
- Move the second to follow the first, shifting the rest along to accommodate

A	B	C	D	E	F	G	H	I	→	A	B	C	D	E	F	G	H	I

- Note that this preserves most of the order and the adjacency information

This Genetic Algorithm also uses a greedy initial population. The city links in the initial tours are not completely random. The GA will prefer to make links between cities that are close to each other. This is not done 100% of the time, because that would cause every tour in the initial population to be very similar.

There are 6 parameters to control the operation of the Genetic Algorithm:

- **Population Size:** The population size is the initial number of random tours that are created when the algorithm starts. A large population takes longer to find a result. A smaller population increases the chance that every tour in the population will eventually look the same. This increases the chance that the best solution will not be found.
- **Neighborhood / Group Size**: Each generation, this number of tours is randomly chosen from the population. The best 2 tours are the parents. The worst 2 tours get replaced by the children. For group size, a high number will increase the likelihood that the really good tours will be selected as parents, but it will also cause many tours to never be used as parents. A large group size will cause the algorithm to run faster, but it might not find the best solution.
- **Mutation %**: The percentage that each child after crossover will undergo **mutation**. When a tour is mutated, one of the cities is randomly moved from one point in the tour to another.
- **# Nearby Cities:** As part of a greedy initial population, the GA will prefer to link cities that are close to each other to make the initial tours. When creating the initial population this is the number of cities that are considered to be close.
- **Nearby City Odds %**: This is the percent chance that any one link in a random tour in the initial population will prefer to use a nearby city instead of a completely random city. If the GA chooses to use a nearby city, then there is an equally random chance that it will be any one of the cities from the previous parameter.
- **Maximum Generations:** How many crossovers are run before the algorithm is terminated.

Example 3: The parameter setting is given as below.

Parameter	**Initial Value**
Population Size	**10,000**
Group Size	**5**
Mutation	**3 %**
# Nearby Cities	**5**

Nearby City Odds	90 %

Example 4: Let us suppose that order crossover #1 is to be carried out on the following two parents to create two children:

$$\begin{array}{lccc|cccc|cc} \text{Parent1} & A & B & C & D & E & F & G & H & I \\ \text{Parent2} & C & D & E & A & B & I & H & G & F \\ & & & * & & & & * & & \end{array}$$

The steps to be followed to get two children solutions from the above two parents are:

Step 1: Select two crossover sites at random, as shown above.

Step 2: To determine child 1, the elements – A , B, I , H (lying between the two crossover sites) are directly copied from parent 2, keeping their locations and order intact.

Step 3: Child 1 tour then begins from parent 1, starting from the first position after the second crossover site and searching towards the initial elements of parent 1.

Step 4: Cities (elements) H, I, A, B are already present in child 1 solution, which should not be considered again.

The H, I, A, B and C of child 1 are selected as follows:

$$\begin{array}{ccccc} \text{Child1}[H] & = & \text{Parent1}[C] & = & C \\ \text{Child1}[I] & = & \text{Parent1}[D] & = & D \\ \text{Child1}[A] & = & \text{Parent1}[E] & = & E \\ \text{Child1}[B] & = & \text{Parent1}[F] & = & F \\ \text{Child1}[C] & = & \text{Parent1}[G] & = & G \end{array}$$

Thus, the obtained child 1 will look as follows:

$$\text{Child 1}: \quad E \quad F \quad G \quad (A \quad B \quad I \quad H) \quad C \quad D$$

Similarly, child 2 can be determined and it will look like the following:

$$\text{Child 2}: \quad B \quad I \quad H \quad (D \quad E \quad F \quad G) \quad C \quad A$$

Example 5: Consider the following two parents participate in order crossover (#2).

$$\begin{array}{cccccccccc} \text{Parent 1} & A & B & C & D & E & F & G & H & I \\ & & * & & * & & & * & * & \\ \text{Parent 2} & C & D & E & A & B & I & H & G & F \end{array}$$

Let us first explain the principle of this operator below in steps.

Step 1: Selected the positions – 2, 4, 7, and 8 as the key positions, at random.

Step 2: The elements of parent 2 at the above mentioned key positions are D, A, H, G. To determine child 1, the ordering of these elements is 4, 1, 8, 7 in parent 2 is imposed on parent 1.

Step 3: In parent 1, the elements – D, A, H and G found in positions – 4th, 1st, 8th and 7th, respectively.

Step 4: In child 1, the elements in these positions (that is, 1st, 4th, 7th and 8th) are selected by matching the order of the elements D, A, H, G as found in parent 2, that is,

$$\text{Child } 1[\text{A}] = \text{D};$$
$$\text{Child } 1[\text{D}] = \text{A};$$
$$\text{Child } 1[\text{G}] = \text{H};$$
$$\text{Child } 1[\text{H}] = \text{G}.$$

Step 5: The remaining elements of child 1 are directly copied from parent 1.

Thus, child 1 will look like the following:

$$\text{Child 1:} \quad \text{D} \quad \text{B} \quad \text{C} \quad \text{A} \quad \text{E} \quad \text{F} \quad \text{H} \quad \text{G} \quad \text{I}$$

Similarly, child 2 can be determined, which will look as follows:

$$\text{Child 2:} \quad \text{C} \quad \text{B} \quad \text{E} \quad \text{D} \quad \text{A} \quad \text{I} \quad \text{G} \quad \text{H} \quad \text{F}$$

Now try the following exercise.

E3) Consider the following travelling salesman problem involving 9 cities.

Parent 1	F	I	G	E	D	C	A	H	B
Parent 2	C	B	G	I	H	F	D	E	A

Determine the children solution using:

i) order crossover #1, assuming 4th and 7th sites as the crossover sites.

ii) order crossover #2, assuming 3rd, 5th and 7th as the key positions.

Now let us solve the TSP using cycle crossover, position-based crossover and partial mapped crossovers in the following examples.

Example 6: Consider two parents, given below, which will participate in a cycle crossover to create two children solutions.

$$\begin{array}{cccccccccc} \text{Parent 1} & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} & \text{H} & \text{I} \\ & & & * & & & & & & \\ \text{Parent 2} & \text{C} & \text{D} & \text{E} & \text{A} & \text{B} & \text{I} & \text{H} & \text{G} & \text{F} \end{array}$$

The operator cycle crossover is explained in the following steps:

Step 1: To determine child 1, select a parent (say parent 1) and the starting position of the cycle (say Parent 1[C] = C), at random. Thus, the 3rd element of child 1 is nothing but element C, that is, child 1[C] = C.

Step 2: Parent 2 is then searched to check the presence of element C and it has been found in the 1st position. The first element of child 1 is selected from the first element of parent 1, that is, Child 1[A] = Parent 1[A] = A.

Step 3: Parent 2 is again searched for a presence of element A and it has occurred at the 4th position. Thus, the 4th element of parent 1 has been copied as the 4th element of child 1, that is, Child 1[D] = Parent 1[D] = D. Similarly, we determine

Child 1[B] = Parent 1[B] = 2,

Child 1[E] = Parent 1[E] = E.

This completes one cycle because element E is seen to be present at the 3rd position of parent 2 and the corresponding 3rd position element of parent 1 is element C, which has already been selected as the starting element of the cycle.

Step 4: The remaining elements of child 1 are selected directly from parent 2, as follows:

$$\begin{array}{ccccc} \text{Child}\,1[\text{F}] & = & \text{Parent}\,2[\text{F}] & = & \text{I}, \\ \text{Child}\,1[\text{G}] & = & \text{Parent}\,2[\text{G}] & = & \text{H}, \\ \text{Child}\,1[\text{H}] & = & \text{Parent}\,2[\text{H}] & = & \text{G}, \\ \text{Child}\,1[\text{I}] & = & \text{Parent}\,2[\text{I}] & = & \text{F}. \end{array}$$

Thus, child 1 is found to be like the following:

Child 1 A B C D E I H G F

Using the same procedure, child 2 can be determined as follows:

Child 2 C D E A B F G H I

Example 7: Consider the following two parents, which will participate in the position-based crossover:

Parent 1	A	B	C	D	E	F	G	H	I
	*			*			*		*
Parent 2	C	D	E	A	B	I	H	G	F

The principle of position-based crossover is discussed in the following steps:

Step 1: To determine child 1, choose a number of crossover points (say 1st, 4th, 7th, 9th) on a parent, say parent 1. The elements – A, D, G, I are directly copied from parent 1 (by keeping their position information intact) to child 1. Thus, we get

Child 1[A] = A,
Child 1[D] = D,
Child 1[G] = G,
Child 1[I] = I.

Step 2: The remaining elements of child 1 are selected from parent 2 as follows:
Child 1[B] = Parent 2[A] = C;
Child 1[C] = Parent 2[C] = E, as Parent 2[B] = D has already been included in child 1;
Child 1[E] = Parent 2[E] = B, as Parent 2[D] = A has already been selected as child 1[A] element;
Child 1[F] = Parent 2[G] = H, as Parent 2[F] = I has already been considered as child 1[I];
Child 1[H] = Parent 2[I] = F, as Parent 2[H] = G has already been occurred as child 1[G].

Thus, the resulting child 1 solution will look as follows:

Child 1   A   C   E   D   B   H   G   F   I

Similarly, child 2 can be determined, which will look like the following:

Child 2   C   B   D   A   E   G   H   I   F

Example 8: Consider the following two parents, which will participate in a Partially Mapped Crossover (PMX):

$$\begin{array}{lccc|cccc|cc} \text{Parent 1:} & A & B & C & D & E & F & G & H & I \\ \text{Parent 2:} & C & D & E & A & B & I & H & G & F \\ & & & * & & & & * & & \end{array}$$

The steps involved in the Partially Mapped Crossover (PMX) are:

Step 1: Select two crossover sites at random, as shown above.

Step 2: The elements – D, E, F, G of parent 1 (lying within the two crossover sites) are directly copied into child 1. Thus, we get

Child 1[D] = D,
Child 1[E] = E,
Child 1[F] = F,
and Child 1[G] = G.

Step 3: The remaining elements of child 1 are determined as follows: The search starts with the elements of parent 1 residing in between the two crossover sites. The element Parent 1[D] = D is located at the 2nd position of parent 2, that is, Parent 2 [B]. Thus, 2nd position of child 1 will be filled up by an element of Parent 2 located at 4th position, that is, city 1.

$$\text{Child } 1[\text{B}] = \text{Parent } 2[\text{D}] = \text{A}.$$

Similarly, the element Parent 1[E] = E is seen to be present at the 3rd position of parent 2, that is, Parent 2[C]. Thus, the 3rd position of child 1 will be occupied by an element of parent 2 located at the 5th position, that is, city 2.

$$\text{Child } 1[\text{C}] = \text{Parent } 2[\text{E}] = \text{B}.$$

Similarly, we can determine the following elements of child 1:

$$\text{Child } 1[\text{I}] = \text{Parent } 2[\text{F}] = \text{I},$$
$$\text{Child } 1[\text{H}] = \text{Parent } 2[\text{G}] = \text{G}.$$

Step 4: The remaining element of child 1 is directly copied from parent 2, that is, Child 1[A] = Parent 2[A] = C.

Thus, child 1 is obtained as follows:

Child 1:   C   A   B   D   E   F   G   H   I

Child 2 can be determined by following the similar procedure and is given by:

Child 2: D E C A B I H G F

It may be noted that partially mapped crossover may sometimes give rise to a child solution, in which a particular city may occur more than once and some other cities might also be missing from it.

Now, try the following exercises.

E4) Let us consider a TSP problem involving eight cities A, B, C, D, E, F, G, H. A scheduling GA with various types of crossover operator is to be used to solve the said optimization problem. Determine children solutions of the following two parents:

Parent 1: A B C D E F G H

Parent 2: C A D B F H E G

i) Order crossover #1, assuming 2-nd and 5-th sites as the crossover sites.

ii) Order crossover #2, considering 3-rd, 4-th, and 7-th as the key positions.

iii) Cycle crossover assuming 4-th as the starting position.

iv) Position-Based Crossover considering 2-nd, 4-th and 6-th as the crossover points.

E5) Consider the two parents which are participating in the Partially Mapped Crossover (PMX), as shown below:

```
Parent 1:  A  B | C  D  E  F | G  H  I
Parent 2:  C  D | E  A  B  I | H  G  F
                *            *
```

Using Partially Mapped Crossover (PMX) assuming 2nd and 6th sites as the crossover sites, find the children solution.

Now, let us summarize the unit.

12.4 SUMMARY

The content of this unit has been summarized as follows:

1. The mechanisms of different crossover and mutation operators used in the real-coded Gas by various investigators, have been explained with the help of suitable numerical examples. The working principle of a micro-GA has been discussed, in which elitism has been used.

2. In a scheduling problem, not only the position of different elements but also their adjacency and order are to be considered. A number of crossover operators have been proposed by various researchers to solve the said problem. The principles of some of these operators have been explained with the help of some appropriate examples.

12.5 SOLUTIONS/ANSWERS

E1) i) Generate initial population by using random number generation.

ii) Select any two parents using tournament selection.

iii) The off springs are generated using the following crossover operator:

$$x = a * x + (1 - a) * y$$

$$y = a * y + (1 - a) * x$$

where a is the random number.

iv) Apply this operation separately for each iteration to calculate the maximum fitness value.

E2) i) Generate the initial population.

ii) Find x value and fitness function $f(x) = x^2$. The iterations are given

Iteration 1 is:

S.No.	Population	X	F(X)
1	10101	21	441
2	11011	27	729
3	01011	11	121
4	10010	18	324
5	11110	30	900
	Sum = 2517	Average = 503	Minimum = 121

Iteration 2 is:

S.No.	Population	X	F(X)
6	10010	18	324
7	10110	22	484
8	11011	27	729
9	01111	15	225
10	01011	11	121
	Sum = 1886	Average = 377	Minimum = 121

Iteration 3 is:

S.No.	Population	X	F(X)
11	10111	23	529
12	10101	21	441
13	11001	25	625
14	10101	21	441
15	10100	20	400
	Sum = 2440	Average = 488	Minimum = 400

Iteration 4 is:

S.No.	Population	X	F(X)
16	10110	22	484
17	11111	31	961
18	01111	15	225
19	10111	23	529
20	10110	22	484
	Sum = 2685	Average = 537	Minimum = 225

Iteration 5 is:

S.No.	Population	X	F(X)
21	01001	9	81
22	01111	15	225
23	01011	11	121
24	01111	15	225
25	11100	28	784
	Sum = 1436	Average = 287	Minimum = 81

Iteration 6 is:

S.No.	Population	X	F(X)
26	00000	0	0
27	00100	4	16
28	01111	15	225
29	11101	29	841
30	11010	26	676
	Sum = 1760	Average = 352	Minimum = 0

Iteration 7 is:

S.No.	Population	X	F(X)
31	10000	16	256
32	00000	0	0
33	01011	11	121
34	01010	10	100
35	00001	1	1
	Sum = 481	Average = 96	Minimum = 0

After iteration 7, minimum value is 0.

E3) i) Using order crossover #1,

$$\begin{array}{lccccccccccc}
\text{Parent 1:} & F & I & G & E & | & D & C & A & | & H & B \\
\text{Parent 2:} & C & B & G & I & | & H & F & D & | & E & A \\
 & & & & & * & & & & * & &
\end{array}$$

$$\begin{array}{l}
\text{Child 1:}\quad G\quad E\quad C\quad A\quad (H\quad F\quad D)\quad B\quad I \\
\text{Child 2:}\quad G\quad I\quad H\quad F\quad (D\quad C\quad A)\quad E\quad B
\end{array}$$

ii) Using order crossover #2

$$\begin{array}{lccccccccc}
\text{Parent 1:} & F & I & G & E & D & C & A & H & B \\
 & & & * & & * & & * & & \\
\text{Parent 2:} & C & B & G & I & H & F & D & E & A
\end{array}$$

The key position is III, V and VII selected at random.

$$\begin{array}{lccccccccc}
\text{Child 1:} & F & I & G & E & H & C & A & D & B \\
\text{Child 2:} & C & B & G & I & H & F & D & E & A
\end{array}$$

E4) i)

$$\begin{array}{lcccccccc}
\text{Child 1:} & G & H & D & B & F & A & C & E \\
\text{Child 2:} & H & G & C & D & E & A & B & F
\end{array}$$

ii) Child 1: A D C B E F G H
Child 2: A B C D F H G E

iii) Child 1: A B C D F H G E
Child 2: C A D B E F G H

iv) Child 1: C B A D H F E G
Child 2: C A D B E H F G

E5) The following children solutions are obtained using the PMX:

Child 1: E A (C D E F) H G I
Child 2: D E (E A B I) G H F

In child 1, city 5 has occurred twice and city 2 is missing from it. The cities lying inside the first brackets are generally not disturbed. Thus, the first element of child 1, that is, city 5 is to be replaced by the missing city, that is, 2. The modified child 1 may be written as follows:

Child 2: B A C D E F H G I

Similarly, the modified child 2 is found to be like the following:

Child 2: D C E A B I G H F

It is important to mention that the performance of different crossover operators is problem-dependent [50].

12.6 PRACTICAL ASSIGNMENT

Session 11

1. Write a program in 'C' language to write the children solution of a TSP consisting of n cities using

   i) Crossover order (#1);

   ii) Crossover order (#2).

   Test your program for E4).

2. Write a program in 'C' language to write the children solution of a TSP consisting of n cities using

   i) Cyclic crossover;

   ii) Position-based crossover.

   Test your program for E4).

12.7 REFERENCES

1. S. Rajasekaran and G.A. Vijayalakshmi Pai, (2010), *Neural Netwoks: Fuzzy Logic, and Genetic Algorithms*.
2. D.K. Pratihar, (2008), *Soft Computing*.
3. J. –S. R. Jang, C. –T. Sun and E. Mizutani, (2004), *Neuro-Fuzzy and Soft Computing*.