Genetic Algorithm & Fuzzy Logic

Semester-5

Practical-3

Name: Shivam Tawari

Roll no: A-58

Aim: Implement Multipoint crossover for given f(x) optimization (maximization) using GA.

$f(x) = w1x1 + w2x2 + w3x3 + w4x4 + w5x5 + w6x6$, where; $x1 = 2$, $x2 = 4$, $x3 = 6$, $x4 = 8$, $x5 = 10$, $x6 = 12$

Theory:

Multi Point Crossover:

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.

In a binary coded GA, model parameters representing a solution to the optimization problem are encoded by binary strings of 0's and 1's referred to as a chromosome. The algorithm starts with a population consisting of a set of chromosomes randomly selected within the search space. The population undergoes genetic operations, i.e. selection, cross-over and mutation, leading to a new population that would be better than the old population.

For Example, Suppose two chromosomes:

*Chromosome 1: < **0 0 0 1 0 0** > and*

Chromosome 2: < 1 0 1 1 1 1 >

For Crossover at points 2 & 6, perform single-point crossover at

crossover points 2 & 6 sequentially on the parents to form

children.

Therefore,

*Offspring 1: < **0 0** 1 1 1 **0** > and*

*Offspring 2: < 1 0 **0 1 0** 1 >*

Code:

```
[2] # Shivam Tawari
    # A-58
```

```
[3] import numpy as np

    # Given Problem
    # f(x) = w1x1 + w2x2 + ... + w6x6
    # Here, x1=2, x2=4, x3=6, x4=8, x5=10, x6=12
    # Multipoint Crossover for Binary Coded GA

    # Suppose we have selected 2 chromosomes from parents for crossover:
    select = 2
    chromosome_length = 6                        # 6 weight values in given problem
    chromosomes_selected = np.random.randint(0, 2, size=(select, chromosome_length))
```

```
[4] def single_pt_crossover(A, B, k):
        """
            Function for Single Point Crossover
            Here used as a building block for multi point crossover
            Inputs:
                - A: Chromosome 1
                - B: Chromosome 2
                - k: Index for crossover (k value)
            Outputs:
                - A_new: Crossover-ed Chromosome 1
                         An array of dimension like A (length of chromosome,)
                - B_new: Crossover-ed Chromosome 2
                         An array of dimension like B (length of chromosome,)
        """
        A_new = np.append(A[:k], B[k:])
        B_new = np.append(B[:k], A[k:])
        return A_new, B_new
```

```python
def multi_pt_crossover(A, B, K):
    """
        Function for Multi Point Crossover
        Inputs:
            - A: Chromosome 1
            - B: Chromosome 2
            - K: Index for crossovers (multiple k values)
        Outputs:
            - A_new: Crossover-ed Chromosome 1
                     An array of dimension like A (length of chromosome,)
            - B_new: Crossover-ed Chromosome 2
                     An array of dimension like B (length of chromosome,)
    """
    for k in K:
        A, B = single_pt_crossover(A, B, k)
    return A, B
```

```python
[5] multi_points = 3                                # Total k values

    # Define k_1, k_2, k_3, ..., k_multiple_points (Crossover Points)
    # Python indexing starts from 0,
    # However complete swapping of genes is same as no swapping
    # Therefore, starting from index 1 to chromosome_length (not included)
    K = np.random.choice(range(1, chromosome_length), multi_points, replace=False)
    K = np.sort(K)

    print(f'Chromosome 1: {chromosomes_selected[0]}')
    print(f'Chromosome 2: {chromosomes_selected[1]}')
    print(f'\nCrossover points: {K}')

    cross_1, cross_2 = multi_pt_crossover(chromosomes_selected[0], chromosomes_selected[1],

    print(f'\nCrossovered Chromosome 1: {cross_1}')
    print(f'Crossovered Chromosome 2: {cross_2}')
```

```
Chromosome 1: [1 1 0 0 0 0]
Chromosome 2: [1 1 0 1 0 1]

Crossover points: [1 3 5]

Crossovered Chromosome 1: [1 1 0 0 0 1]
Crossovered Chromosome 2: [1 1 0 1 0 0]
```

Conclusion: Hence, multipoint crossover for given f(x) has been implemented successfully.