Experiment 10

NAME	**Shreya Shetty**
UID	**2019140059**
CLASS	**TE IT**
BATCH	**B**
ACADEMIC YEAR	**2021-22**
SUBJECT	**SC (Soft Computing)**
COURSE CODE	**IT312**
EXPERIMENT NO.	**10**

Aim:

To apply genetic algorithms to a given problem

Theory:

Genetic Algorithms

GAs are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and generics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, GAs are by no means random; instead, they exploit historical information to direct the search into the region of better performance within the search space. The basic techniques of the GAs are designed to simulate processes in natural systems necessary for evolution, especially those that follow the principles first laid down by Charles Darwin, "survival of the fittest," because in nature, competition among individuals for seamy resources results in the fittest individuals dominating over the weaker ones.

Simple GA Procedure

1. **Selection**: The first step consists in selecting individuals for reproduction. This selection is done randomly with a probability depending on the relative fitness of the individuals so that the best ones are often chosen for reproduction rather than the poor ones.
2. **Reproduction**: In the second step, offspring are bred by selected individuals. For generating new chromosomes, the algorithm can use both recombination and mutation.
3. **Evaluation**: Then the fitness of the new chromosomes is evaluated.
4. **Replacement**: During the last step, individuals from the old population are killed and replaced by the new ones.
   The algorithm is stopped when the population converges toward the optimal solution.
5. BEGIN/* genetic algorithm"/
   a. Generate initial population;
   b. Compare the fitness of each individual;

c. WHILE NOT finished DO LOOP BEGIN
    - Select individuals from old generations for mating;
    - Create offspring by applying recombination and/or mutation for the selected individuals;
    - Compute fitness of the new individuals;
    - Kill old individuals to make room for new chromosomes and insert offspring in the new generalization;
    - IF the Population has converged
      THEN finishes: =TRUE;
        END

END

__Flow Chart__

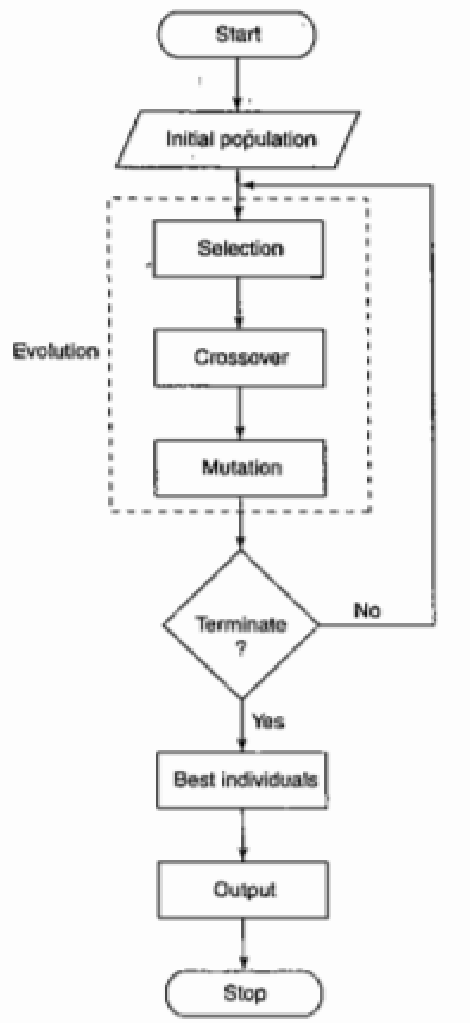


Figure 15-13 Flowchart for genetic algorithm.

Solved Example

Shreya Shetty TE IT 2019140059

① Experiment 10

Book: Principles of Soft Computing Pg: 418

Q Consider a problem of maximizing the function, $f(x) = x^2$, where x is permitted to vary between 0 & 31.

Sol^n: Objective function $f(x) = x^2$ is to be maximized.

Following is the initial selection of population at random.

String no.	Initial population (randomly selected)	x value	Fitness $f(x) = x^2$	Prob;	Expected Count	Actual Count
1	01100	12	144	0.1247	0.499	1
2	11001	25	625	0.5411	2.1645	2
3	00101	5	25	0.0216	0.086	0
4	10011	19	361	0.3126	1.2502	1
Sum			1155	1	4	4
Avg.			288.75	0.25	1	1
Max			625	0.5411	2.1645	2

Step 1: Code decision variable 'x' into finite length string.
Here, initial population of size 4 is chosen.

Step 2: Obtain decoded x values for initial population generated. Consider string 1:

$01100 = 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 + 0$
$= 12$

$\|^{ly}$ finding decoded values for all strings.

Step 3: Calculating fitness or objective function. $f(x) = x^2$

at $x = 12$, $f(12) = 12^2 = 144$

$f(25) = 25^2 = 625$

$f(5) = 5^2 = 25$

$f(19) = 19^2 = 361$

Step 4: Compute prob. of selection.

$$Prob_i = \frac{f(x)_i}{\sum_{i=1}^{n} f(x)_i}, \quad n \text{ is no. of populations.}$$

$$\Sigma f(x) = 144 + 625 + 25 + 361 = 1155$$

for string 1, $P_1 = \frac{144}{1155} = 0.1247$

% prob P_i obtained of $0.1247 \times 100 = 12.47\%$

for string 2, $P_2 = 625/1155 = 0.5411$

for string 3, $P_3 = 25/1155 = 0.0216$

for string 4, $P_4 = 361/1155 = 0.3126$

Step 5: Calculating expected count.

$$\text{expected count} = \frac{f(x)_i}{[\text{Avg } f(x)]_i}$$

$$[\text{Avg}(f(x))]_i = \frac{\sum_{i=1}^{n} f(x)_i}{n} = \frac{1155}{4} = 288.75$$

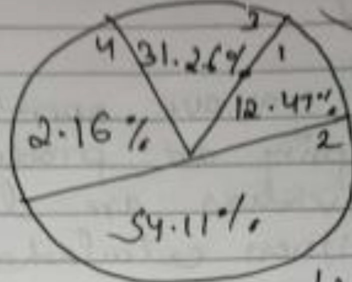
for string 1, expected count $= \frac{\text{fitness}}{\text{avg}} = \frac{144}{288.75} = 0.498$

for string 2, expected count $= \frac{625}{288.75} = 2.1645$

for string 3, expected count $= \frac{25}{288.75} = 0.0866$

for string 4, expected count $= \frac{361}{288.75} = 1.2501$

Step 6: finding actual count: Selection using Roulette wheel:



String 1 occupies 12.47%, hence, its chance is atleast 1.
With string 2 54.11%, count is considered 2.
String 3 has least prob, so chance is poor, count is 0.
string 4 with 31.26% has atleast 1 chance so actual count is 1.

Step 7: Now, write mating pool based on actual count
String 1 occurs once, string 2 occurs 2 times, string 3 occurs 0 times & string 4 occurs once in mating pool.

Step 8: Perform crossover operation to produce offspring
Crossover point is specified & based on that crossover is performed, single pt crossover is performed

Parent 1 0 1 1 0 0
Parent 2 1 1 0 0 1

offspring 1 01101

offspring 2 11000

Step 9: After cross over operation, new offspring are produced & 'x' values are decoded & fitness calculated.

Step 10: Mutation operation is performed to produce new offspring after cross over operation. Once, offspring are obtained, after mutation, they are decoded to x value & fitness values are computed.

Cross over:

String No	Mating Pool	Cross over pt.	Offspring	x value	fitness value
1	01100	4	01101	13	169
2	11001	4	11000	24	576
3	11001	2	11011	27	729
4	10011	2	10001	17	289

Mutation:

String No.	~~...~~ Offspring	Mutation chromosomes for flipping	offspring after mutation	x value	fitness $f(x)=x^2$
1	01101	10000	11101	29	841
2	11000	00000	11000	24	576
3	11011	00000	11011	27	729
4	10001	00100	10100	20	400

Code:

```python
# Importing libraries
import numpy as np
from random import randint
from prettytable import PrettyTable

# Defining function to maximize, F(x) = x^2
def fx(x):
  return x**2

# Crossover Function
def crossover(iter, mat_pool):
  init_op = [4,4,2,2]
  cross = []
  c_pts = []
  i = 0
  while i<len(mat_pool):
    p1 = mat_pool[i]
    p2= mat_pool[i+1]
    if iter == 0:
      cp = init_op[i]
    else:
      cp = randint(0,4)
    c_pts.append(cp)
    c_pts.append(cp)
    cross.append(p1[:cp]+p2[cp:])
    cross.append(p2[:cp]+p1[cp:])
    i+=2
  return cross, c_pts


# Mutation Function
def mutation(iter, cross):
  init_mut_chrom= [1,0,0,4]
  mutation_chrom = []
  if iter == 0:
    rn = init_mut_chrom
  else:
    rn = [randint(0,5) for i in range(len(pop))]

  for i in range(len(pop)):
    temp = ""
    for k in range(5):
      if k == rn[i]-1:
        temp+="1"
      else:
        temp += "0"
    mutation_chrom.append(temp)

  mutation = []
  for i in range(len(cross)):
    temp = ""
```

```
    for j in range(len(mutation_chrom[i])):
      if mutation_chrom[i][j] == "1":
        if cross[i][j] == "0":
          temp+="1"
        else:
          temp+="0"
      else:
        temp += cross[i][j]
    mutation.append(temp)
  return mutation, mutation_chrom


# Genetic Algorithm Steps
# Pg 418, Principles of Soft COmputing
# Defining initial poplation
initial_population = ['01100', '11001', '00101','10011']
pop = initial_population
iter = 0
while "11111" not in pop:
  x = PrettyTable()
  y = PrettyTable()
  z = PrettyTable()
  print("Iteraton Number :", iter)
  x.add_column('String No.', [1,2,3,4])

  # Adding population strings in table
  x.add_column('Population', pop)

  # Finding x vales
  x_val = [int(i,2) for i in pop]
  x.add_column('x value', x_val)

  # Finding fitness values
  fitness = [fx(i) for i in x_val]
  x.add_column('Fitness F(x)=x^2', fitness)

  # Findin sum, average and max of fitness values of all 4 strings
  fx_sum = sum(fitness)
  fx_avg = sum(fitness)/len(fitness)
  fx_max = max(fitness)

  # Calculating probability
  prob = [i/fx_sum for i in fitness]
  x.add_column('Probability', [round(p,4) for p in prob])
  x.add_column('Percentage Probability', [round(p*100,2) for p in prob])

  # Calculating expected count
  exp_count = [round(i/fx_avg,4) for i in fitness]
  x.add_column('Expected Count', exp_count)

  # Finding actual count
  act_count = [round(i+0.1) for i in exp_count]
  x.add_column('Actual Count',act_count)
```

```
print(x, end='\n')
print('Sum: {} Avg: {} Max: {}'.format(fx_sum, fx_avg, fx_max))

y.add_column('String No.', [1,2,3,4])
mat_pool = []
if iter == 0:
  for i in range(len(pop)):
    mat_pool.extend([pop[i]]*act_count[i])
else:
  mat_pool = np.random.choice(pop,len(pop), p=prob)
y.add_column("Mating Pool", mat_pool)

#generating cross over offspring
cross, c_pts = crossover(iter, mat_pool)
y.add_column("Crossover Points ", c_pts)
y.add_column("Offspring after Crossover", cross)

# Calculating x valuesof offsprings
x_val = [int(i,2) for i in cross]
y.add_column('x value', x_val)

# generating fitness values for crossover
fitness = [fx(i) for i in x_val]
y.add_column('Fitness F(x)=x^2', fitness)

print(y, end="\n")

# generating mutation offsprings
z.add_column('String No.', [1,2,3,4])
mutat, mutat_chrom = mutation(iter, cross)
z.add_column("Offspring after Crossover", cross)
z.add_column("Mutation Chromosomes", mutat_chrom)
z.add_column("Offspring after Mutation", mutat)

# generating x-val for mutation offsprings
x_val = [int(i,2) for i in mutat]
z.add_column("x value", x_val)

# generating fitness values for mutation offsprings
fitness = [fx(i) for i in x_val]
z.add_column("Fitness F(x)=x^2", fitness)

print(z, end="\n")
# assigning new mutation to population
pop = mutat
print()

iter +=1
```

Output:

Iteraton Number : 0

String No.	Population	x value	Fitness F(x)=x^2	Probability	Percentage Probability	Expected Count	Actual Count
1	01100	12	144	0.1247	12.47	0.4987	1
2	11001	25	625	0.5411	54.11	2.1645	2
3	00101	5	25	0.0216	2.16	0.0866	0
4	10011	19	361	0.3126	31.26	1.2502	1

Sum: 1155 Avg: 288.75 Max: 625

String No.	Mating Pool	Crossover Points	Offspring after Crossover	x value	Fitness F(x)=x^2
1	01100	4	01101	13	169
2	11001	4	11000	24	576
3	11001	2	11011	27	729
4	10011	2	10001	17	289

String No.	Offspring after Crossover	Mutation Chromosomes	Offspring after Mutation	x value	Fitness F(x)=x^2
1	01101	10000	11101	29	841
2	11000	00000	11000	24	576
3	11011	00000	11011	27	729
4	10001	00010	10011	19	361

Iteraton Number : 1

String No.	Population	x value	Fitness F(x)=x^2	Probability	Percentage Probability	Expected Count	Actual Count
1	11101	29	841	0.3355	33.55	1.3418	1
2	11000	24	576	0.2298	22.98	0.919	1
3	11011	27	729	0.2908	29.08	1.1631	1
4	10011	19	361	0.144	14.4	0.576	1

Sum: 2507 Avg: 626.75 Max: 841

String No.	Mating Pool	Crossover Points	Offspring after Crossover	x value	Fitness F(x)=x^2
1	11101	3	11100	28	784
2	11000	3	11001	25	625
3	11011	0	11011	27	729
4	11011	0	11011	27	729

String No.	Offspring after Crossover	Mutation Chromosomes	Offspring after Mutation	x value	Fitness F(x)=x^2
1	11100	00100	11000	24	576
2	11001	00001	11000	24	576
3	11011	00001	11010	26	676
4	11011	10000	01011	11	121

Iteraton Number : 2

String No.	Population	x value	Fitness F(x)=x^2	Probability	Percentage Probability	Expected Count	Actual Count
1	11000	24	576	0.2955	29.55	1.1821	1
2	11000	24	576	0.2955	29.55	1.1821	1
3	11010	26	676	0.3468	34.68	1.3874	1
4	01011	11	121	0.0621	6.21	0.2483	0

Sum: 1949 Avg: 487.25 Max: 676

String No.	Mating Pool	Crossover Points	Offspring after Crossover	x value	Fitness F(x)=x^2
1	11000	4	11000	24	576
2	11000	4	11000	24	576
3	11010	4	11010	26	676
4	11000	4	11000	24	576

String No.	Offspring after Crossover	Mutation Chromosomes	Offspring after Mutation	x value	Fitness F(x)=x^2

1	11000	00100	11100	28	784
2	11000	00010	11010	26	676
3	11010	00010	11000	24	576
4	11000	00100	11100	28	784

Iteraton Number : 3

String No.	Population	x value	Fitness F(x)=x^2	Probability	Percentage Probability	Expected Count	Actual Count
1	11100	28	784	0.278	27.8	1.1121	1
2	11010	26	676	0.2397	23.97	0.9589	1
3	11000	24	576	0.2043	20.43	0.817	1
4	11100	28	784	0.278	27.8	1.1121	1

Sum: 2820 Avg: 705.0 Max: 784

String No.	Mating Pool	Crossover Points	Offspring after Crossover	x value	Fitness F(x)=x^2
1	11000	2	11100	28	784
2	11100	2	11000	24	576
3	11100	4	11100	28	784
4	11000	4	11000	24	576

String No.	Offspring after Crossover	Mutation Chromosomes	Offspring after Mutation	x value	Fitness F(x)=x^2
1	11100	00010	11110	30	900
2	11000	00010	11010	26	676
3	11100	00010	11110	30	900
4	11000	00100	11100	28	784

Iteraton Number : 4

String No.	Population	x value	Fitness F(x)=x^2	Probability	Percentage Probability	Expected Count	Actual Count
1	11110	30	900	0.2761	27.61	1.1043	1
2	11010	26	676	0.2074	20.74	0.8294	1
3	11110	30	900	0.2761	27.61	1.1043	1
4	11100	28	784	0.2405	24.05	0.962	1

Sum: 3260 Avg: 815.0 Max: 900

String No.	Mating Pool	Crossover Points	Offspring after Crossover	x value	Fitness F(x)=x^2
1	11110	0	11110	30	900
2	11110	0	11110	30	900
3	11110	0	11010	26	676
4	11010	0	11110	30	900

String No.	Offspring after Crossover	Mutation Chromosomes	Offspring after Mutation	x value	Fitness F(x)=x^2
1	11110	01000	10110	22	484
2	11110	00001	11111	31	961
3	11010	00010	11000	24	576
4	11110	00001	11111	31	961

Conclusion:

In this experiment, I have implemented a Genetic algorithm, which is a family of heuristics, in python and solved on paper.