Roll no - 41239,41232
Class :BE Comp

LP4 SCOA: Mini project No. 1

Title: Apply the Genetic Algorithm for optimization on n queens problem.

Software Requirements:

-Anaconda with Python 3.7/ Colab

Hardware Requirement:

-PIV, 2GB RAM, 500 GB HDD, Lenovo A13-4089Model.

Learning Objectives:

-Learn How to Apply Genetic Algorithm for given Iris Dataset. The main objective of this assignment is to implement Iris Flower Dataset or any other dataset into a data frame using python .

Outcomes:

-After completion of this assignment students are able Implement code for the Iris Dataset with plotting diagram.

Theory Concepts:

Genetic Algoritm :
1. Genetic Algorithms (GAs) are search based algorithms based on the concepts of natural selection and genetics. GAs are a subset of a much larger branch of computation known as **Evolutionary Computation**.

2.GA works on a population consisting of some solutions where the population size (popsize) is the number of solutions. Each solution is called individual

3.Also, each individual has a fitness value. To select the best individuals, a fitness function is used. The result of the fitness function is the fitness value representing the quality of the solution. The higher the fitness value the higher the quality the solution. .

4.The individuals in the mating pool are called parents. Every two parents selected from the

mating pool will generate two offspring (children).

A general outline how Genetic Algorithm (GA) works is given below:

1. A random population of candidate solutions is created and the fitness scores of the individuals are calculated and the chromosomes are sorted in the population and ranked according to the fitness values.

2. Certain number of chromosomes will pass onto the next generation depending on a selection operator, favoring the better individuals based on their ranking in the population.

3. Selected chromosomes acting as parents will take part in crossover operation to create children whose fitness values are to be calculated simultaneously. Crossover probability is generally kept high, because it is seen to give better children in terms of fitness values.

4. Then based on a mutation probability, a mutation operator is applied on new individuals which randomly changes few chromosomes. Mutation probability is generally kept low.

5. Evaluated off-springs, together with their parents form the population for the next generation.

Advantages of GAs

GAs have various advantages which have made them immensely popular. These include −

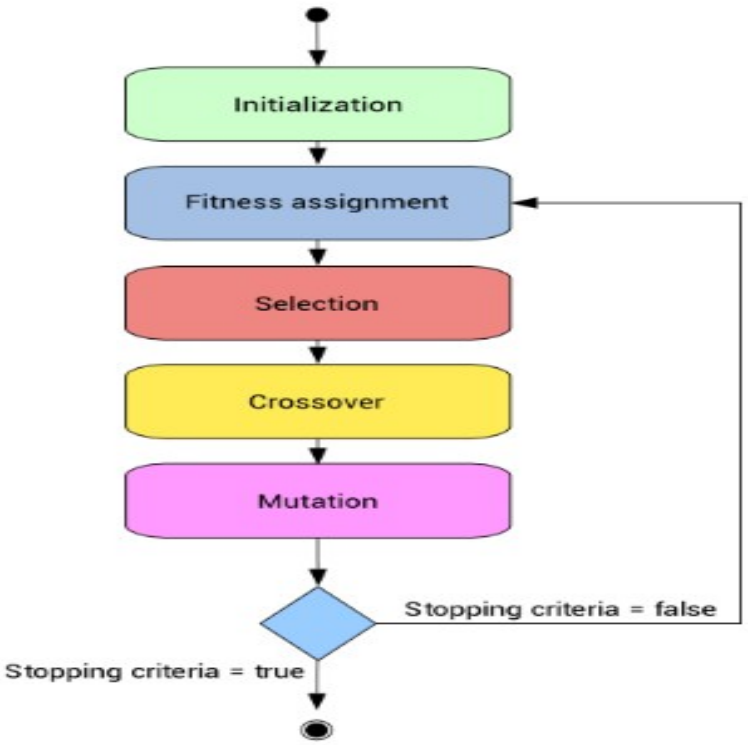
- Does not require any derivative information (which may not be available for many real-world problems).
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of “good” solutions and not just a single solution.

- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

Limitations of GAs

Like any technique, GAs also suffer from a few limitations. These include −

- GAs are not suited for all problems, especially problems which are simple and for which derivative information is available.
- Fitness value is calculated repeatedly which might be computationally expensive for some problems.
- Being stochastic, there are no guarantees on the optimality or the quality of the solution.
- If not implemented properly, the GA may not converge to the optimal solution.



Mutation: Mutation is very important in genetic algorithm for not to stuck the process in local optimum.

Crossover : Order 1 Crossover is one sort of very simple permutation crossover in which 2 random points are selected from parent-1 and the alleles between these points are carried over to the child and the other alleles from parent-2 which are absent in child are carried and placed in the child in the order which they appear in parent-2.

Conclusion

We implemented genetic algorithm for N-Queens.

Code :

```
import random

def random_chromosome(size): #making random chromosomes
return [ random.randint(1, nq) for _ in range(nq) ]

def fitness(chromosome):
horizontal_collisions = sum([chromosome.count(queen)-1 for queen in
chromosome])/2
diagonal_collisions = 0

n = len(chromosome)
left_diagonal = [0] * 2*n
right_diagonal = [0] * 2*n
for i in range(n):
left_diagonal[i + chromosome[i] - 1] += 1
right_diagonal[len(chromosome) - i + chromosome[i] - 2] += 1

diagonal_collisions = 0
for i in range(2*n-1):
counter = 0
if left_diagonal[i] > 1:
counter += left_diagonal[i]-1
if right_diagonal[i] > 1:
counter += right_diagonal[i]-1
```

```
diagonal_collisions += counter / (n-abs(i-n+1))
return int(maxFitness - (horizontal_collisions + diagonal_collisions))
#28-(2+3)=23

def probability(chromosome, fitness):
return fitness(chromosome) / maxFitness

def random_pick(population, probabilities):
populationWithProbabilty = zip(population, probabilities)
total = sum(w for c, w in populationWithProbabilty)
r = random.uniform(0, total)
upto = 0
for c, w in zip(population, probabilities):
if upto + w >= r:
return c
upto += w
assert False, "Shouldn't get here"
def reproduce(x, y): #doing cross_over between two chromosomes
n = len(x)
c = random.randint(0, n - 1)
return x[0:c] + y[c:n]

def mutate(x): #randomly changing the value of a random index of a
chromosome
n = len(x)
c = random.randint(0, n - 1)
m = random.randint(1, n)
x[c] = m
return x

def genetic_queen(population, fitness):
mutation_probability = 0.03
new_population = []
probabilities = [probability(n, fitness) for n in population]
for i in range(len(population)):
x = random_pick(population, probabilities) #best chromosome 1
y = random_pick(population, probabilities) #best chromosome 2
child = reproduce(x, y) #creating two new chromosomes from the best 2
chromosomes
if random.random() < mutation_probability:
child = mutate(child)
print_chromosome(child)
new_population.append(child)
if fitness(child) == maxFitness: break
return new_population
```

```
def print_chromosome(chrom):
print("Chromosome = {}, Fitness = {}"
.format(str(chrom), fitness(chrom)))

if __name__ == "__main__":
nq = int(input("Enter Number of Queens: ")) #say N = 8
maxFitness = (nq*(nq-1))/2 # 8*7/2 = 28
population = [random_chromosome(nq) for _ in range(100)]
generation = 1

while not maxFitness in [fitness(chrom) for chrom in population]:
print("=== Generation {} ===".format(generation))
population = genetic_queen(population, fitness)
print("")
print("Maximum Fitness = {}".format(max([fitness(n) for n in
population])))
generation += 1
chrom_out = []
print("Solved in Generation {}!".format(generation-1))
for chrom in population:
if fitness(chrom) == maxFitness:
print("");
print("One of the solutions: ")
chrom_out = chrom
print_chromosome(chrom)
board = []

for x in range(nq):
board.append(["x"] * nq)
for i in range(nq):
board[nq-chrom_out[i]][i]="Q"

def print_board(board):
for row in board:
print (" ".join(row))
print()
print_board(board)
```

Output -

…
….

```
Maximum Fitness = 14
=== Generation 98 ===
Chromosome = [4, 3, 5, 5, 6, 4],  Fitness = 12
Chromosome = [4, 3, 3, 2, 6, 3],  Fitness = 11
Chromosome = [6, 3, 5, 5, 1, 4],  Fitness = 13
Chromosome = [6, 3, 5, 5, 1, 1],  Fitness = 12
Chromosome = [1, 3, 5, 2, 6, 4],  Fitness = 14
Chromosome = [4, 6, 5, 5, 6, 4],  Fitness = 11
Chromosome = [4, 6, 2, 1, 1, 4],  Fitness = 12
Chromosome = [4, 3, 5, 2, 1, 4],  Fitness = 13
Chromosome = [4, 2, 5, 2, 1, 1],  Fitness = 12
Chromosome = [4, 3, 2, 1, 1, 5],  Fitness = 13
Chromosome = [4, 3, 5, 2, 1, 4],  Fitness = 13
Chromosome = [6, 3, 5, 5, 1, 4],  Fitness = 13
Chromosome = [6, 3, 2, 1, 1, 4],  Fitness = 13
Chromosome = [1, 3, 2, 2, 2, 4],  Fitness = 11
Chromosome = [6, 3, 2, 1, 6, 1],  Fitness = 12
Chromosome = [4, 3, 5, 5, 2, 4],  Fitness = 12
Chromosome = [4, 3, 5, 5, 6, 1],  Fitness = 13
Chromosome = [4, 6, 2, 1, 2, 4],  Fitness = 12
Chromosome = [4, 3, 5, 1, 1, 4],  Fitness = 12
Chromosome = [4, 3, 5, 5, 1, 4],  Fitness = 12
Chromosome = [6, 3, 5, 5, 6, 1],  Fitness = 12
Chromosome = [4, 2, 5, 5, 1, 1],  Fitness = 13
Chromosome = [1, 3, 5, 5, 1, 4],  Fitness = 12
Chromosome = [4, 3, 5, 5, 1, 4],  Fitness = 12
Chromosome = [4, 1, 5, 2, 6, 3],  Fitness = 15

Maximum Fitness = 15
Solved in Generation 98!

One of the solutions:
Chromosome = [4, 1, 5, 2, 6, 3],  Fitness = 15

x x x x Q x
x x Q x x x
Q x x x x x
x x x x x Q
x x x Q x x
x Q x x x x
```