Practical - 4 A-58

Aim: Write a program in Python to implement genetic algorithm.

Theory:

Genetic Algorithm:
GA is a search based optimization technique based on 'survival of the fittest'. The algorithm reflects the ~~post~~ process of natural selection in order to produce offspring of the next generation.

GA are implemented as a simulation in which a population of abstract representations of candidate solutions to an optimization problem evolves towards better solution.

There are some key terms in GA:
Population - subset of all the possible solutions to the given problem.

Chromosome - A chromosome is one such solution to the given problem.

Gene - A gene is one element position of a chromosome.

Genotype - Particular set of genes in a genome

Phenotype - Physical characteristic of the genotype.

The evolution starts from a population of randomly generated individuals and happens in generations.

In each generation, the fitness of each individual is checked and multiple individuals are selected for crossover. The result of this crossover is then mutated to form a new population. This new population is used in the next iteration and GA terminates when maximum number of generations are reached or a satisfactory solution is reached.

Problem Description:
In the game of chess, the queen can attack across any number of unoccupied space on the board horizontally, vertically or diagonally.
The 8-queen puzzle involves putting 8-queens on a standard chess board such that none are under attack.

Code:

Code Link: https://onlinegdb.com/BJBLSdNwv

practical 4.py - C:\Users\shiva\Documents\GitHub\Artificial-Intelligence-Department-GHRCE\2nd Year\AI Knowledge Representation\practical 4.py (3.8.3)

File Edit Format Run Options Window Help

```python
# Shivam Tawari A-58
import random
from bisect import bisect_left
from enum import Enum
from math import exp

class Chromosome:
    def __init__(self, genes, fitness, strategy):
        self.Genes = genes
        self.Fitness = fitness
        self.Strategy = strategy
        self.Age = 0

class Strategies(Enum):
    Create = 0,
    Mutate = 1,
    Crossover = 2

def _generate_parent(length, geneSet, get_fitness):
    genes = []
    while len(genes) < length:
        sampleSize = min(length - len(genes), len(geneSet))
        genes.extend(random.sample(geneSet, sampleSize))
    fitness = get_fitness(genes)
    return Chromosome(genes, fitness, Strategies.Create)

def _mutate(parent, geneSet, get_fitness):
    childGenes = parent.Genes[:]
    index = random.randrange(0, len(parent.Genes))
    newGene, alternate = random.sample(geneSet, 2)
    childGenes[index] = alternate if newGene == childGenes[index] else newGene
    fitness = get_fitness(childGenes)
    return Chromosome(childGenes, fitness, Strategies.Mutate)

def _crossover(parentGenes, index, parents, get_fitness, crossover, mutate,
               generate_parent):
    donorIndex = random.randrange(0, len(parents))
    if donorIndex == index:
        donorIndex = (donorIndex + 1) % len(parents)
    childGenes = crossover(parentGenes, parents[donorIndex].Genes)
    if childGenes is None:
```

practical 4.py - C:\Users\shiva\Documents\GitHub\Artificial-Intelligence-Department-GHRCE\2nd Year\AI Knowledge Representation\practical 4.py (3.8.3)

File Edit Format Run Options Window Help

```
    if childGenes is None:
        parents[donorIndex] = generate_parent()
        return mutate(parents[index])
    fitness = get_fitness(childGenes)
    return Chromosome(childGenes, fitness, Strategies.Crossover)

def get_best(get_fitness, targetLen, optimalFitness, geneSet, display,
             maxAge=None, poolSize=1, crossover=None):
    def fnMutate(parent):
        return _mutate(parent, geneSet, get_fitness)

    def fnGenerateParent():
        return _generate_parent(targetLen, geneSet, get_fitness)

    strategyLookup = {
        Strategies.Create: lambda p, i, o: fnGenerateParent(),
        Strategies.Mutate: lambda p, i, o: fnMutate(p),
        Strategies.Crossover: lambda p, i, o:
        _crossover(p.Genes, i, o, get_fitness, crossover, fnMutate,
                   fnGenerateParent)}

    usedStrategies = [strategyLookup[Strategies.Mutate]]
    if crossover is not None:
        usedStrategies.append(strategyLookup[Strategies.Crossover])

        def fnNewChild(parent, index, parents):
            return random.choice(usedStrategies)(parent, index, parents)
    else:
        def fnNewChild(parent, index, parents):
            return fnMutate(parent)

    for improvement in _get_improvement(fnNewChild, fnGenerateParent,
                                        maxAge, poolSize):
        display(improvement)
        f = strategyLookup[improvement.Strategy]
        usedStrategies.append(f)
        if not optimalFitness > improvement.Fitness:
            return improvement

def _get_improvement(new_child, generate_parent, maxAge, poolSize):
    bestParent = generate_parent()
```

practical 4.py - C:\Users\shiva\Documents\GitHub\Artificial-Intelligence-Department-GHRCE\2nd Year\AI Knowledge Representation\practical 4.py (3.8.3)

File Edit Format Run Options Window Help

```
    bestParent = generate_parent()
    yield bestParent
    parents = [bestParent]
    historicalFitnesses = [bestParent.Fitness]
    for _ in range(poolSize - 1):
        parent = generate_parent()
        if parent.Fitness > bestParent.Fitness:
            yield parent
            bestParent = parent
            historicalFitnesses.append(parent.Fitness)
        parents.append(parent)
    lastParentIndex = poolSize - 1
    pindex = 1
    while True:
        pindex = pindex - 1 if pindex > 0 else lastParentIndex
        parent = parents[pindex]
        child = new_child(parent, pindex, parents)
        if parent.Fitness > child.Fitness:
            if maxAge is None:
                continue
            parent.Age += 1
            if maxAge > parent.Age:
                continue
            index = bisect_left(historicalFitnesses, child.Fitness, 0,
                                len(historicalFitnesses))
            proportionSimilar = index / len(historicalFitnesses)
            if random.random() < exp(-proportionSimilar):
                parents[pindex] = child
                continue
            bestParent.Age = 0
            parents[pindex] = bestParent
            continue
        if not child.Fitness > parent.Fitness:
            child.Age = parent.Age + 1
            parents[pindex] = child
            continue
        child.Age = 0
        parents[pindex] = child
        if child.Fitness > bestParent.Fitness:
            bestParent = child
            yield bestParent
```

practical 4.py - C:\Users\shiva\Documents\GitHub\Artificial-Intelligence-Department-GHRCE\2nd Year\AI Knowledge Representation\practical 4.py (3.8.3)

File Edit Format Run Options Window Help

```
            historicalFitnesses.append(bestParent.Fitness)


def get_fitness(genes, size):
    board = Board(genes, size)
    rowsWithQueens = set()
    colsWithQueens = set()
    northEastDiagonalsWithQueens = set()
    southEastDiagonalsWithQueens = set()
    for row in range(size):
        for col in range(size):
            if board.get(row, col) == 'Q':
                rowsWithQueens.add(row)
                colsWithQueens.add(col)
                northEastDiagonalsWithQueens.add(row + col)
                southEastDiagonalsWithQueens.add(size - 1 - row + col)
    total = size - len(rowsWithQueens) \
            + size - len(colsWithQueens) \
            + size - len(northEastDiagonalsWithQueens) \
            + size - len(southEastDiagonalsWithQueens)
    return Fitness(total)

class Board:
    def __init__(self, genes, size):
        board = [['.'] * size for _ in range(size)]
        for index in range(0, len(genes), 2):
            row = genes[index]
            column = genes[index + 1]
            board[column][row] = 'Q'
        self._board = board

    def get(self, row, column):
        return self._board[column][row]

    def print(self):
        for i in reversed(range(len(self._board))):
            print(' '.join(self._board[i]))

class Fitness:
    def __init__(self, total):
        self.Total = total
```

practical 4.py - C:\Users\shiva\Documents\GitHub\Artificial-Intelligence-Department-GHRCE\2nd Year\AI Knowledge Representation\practical 4.py (3.8.3)

File Edit Format Run Options Window Help

```
    def get(self, row, column):
        return self._board[column][row]

    def print(self):
        for i in reversed(range(len(self._board))):
            print(' '.join(self._board[i]))

class Fitness:
    def __init__(self, total):
        self.Total = total

    def __ge__(self, other):
        return self.Total >= other.Total

    def __gt__(self, other):
        return self.Total < other.Total

def display(candidate, size):
    board = Board(candidate.Genes, size)
    board.print()
    print("Number of attacking pairs: ", candidate.Fitness.Total)
    print("---------Gen End-----------")

class EightQueensTests():
    def test(self, size=8):
        geneset = [i for i in range(size)]

        def fnDisplay(candidate):
            display(candidate, size)

        def fnGetFitness(genes):
            return get_fitness(genes, size)

        optimalFitness = Fitness(0)
        best = get_best(fnGetFitness, 2 * size, optimalFitness,
                                geneset, fnDisplay)

if __name__ == '__main__':
    eightQueen = EightQueensTests()
    eightQueen.test()
```

Output:

```
MINGW64:/c/Users/shiva/Documents/GitHub/Artificial-Intelligence-Department-GHRCE/2nd Year/AI Knowledge Representation

shiva@DESKTOP-BVAG36M MINGW64 ~/Documents/GitHub/Artificial-Intelligence-Department-GHRCE/2nd Year/AI Knowledge Representation (master)
$ python 'practical 4.py'
. . . . . . . .
. . . . Q . . Q
Q . . . . . . .
. . . . . . . .
. . . . . . . Q
. . . . . Q . .
. . Q . Q . . .
. . . Q . . . .
Number of attacking pairs:  10
---------Gen End-----------
. . . . . . . .
. . . . Q . . Q
Q . . . . . . .
. . . . . . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . Q Q . . .
Number of attacking pairs:  9
---------Gen End-----------
. . . . . . . .
. . . . . . . Q
Q . . . Q . . .
. . . . . . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
. . . Q Q . . .
Number of attacking pairs:  8
---------Gen End-----------
. . . . . . . .
. . . . . . . Q
Q . . . Q . . .
. . . . . . . .
. . . . . Q . .
. . . . . Q . .
. . Q . . . . .
. . . Q Q . . .
Number of attacking pairs:  7
---------Gen End-----------
```

MINGW64:/c/Users/shiva/Documents/GitHub/Artificial-Intelligence-Department-GHRCE/2nd Year/AI Knowledge Representation

```
. . . . . . . .
. . . . . . . Q
Q . . . Q . . .
. . Q . . . . .
. . . . . Q . .
. . . . . Q . .
. . . . . . . .
. Q . Q . . . .
Number of attacking pairs:  5
---------Gen End-----------
. Q . . . . . .
. Q . . . . . .
. . . . Q . . .
Q . . . . . . .
. . . . . . . Q
. . . . . Q . .
. Q . . . . . .
. . . . . Q . .
Number of attacking pairs:  4
---------Gen End-----------
. . . . . . Q .
Q . . . . . . .
. . . . Q . . .
Q . . . . . . .
. . . . . . . Q
. . . Q . . . .
. Q . . . . . .
. . . Q . . . .
Number of attacking pairs:  3
---------Gen End-----------
. . . . . . Q .
Q . . . . . . .
. . . . Q . . .
. Q . . . . . .
. . . . . . . Q
. . . . . Q . .
. . . Q . . . .
. Q . . . . . .
Number of attacking pairs:  2
---------Gen End-----------
. . Q . . . . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
```

MINGW64:/c/Users/shiva/Documents/GitHub/Artificial-Intelligence-Department-GHRCE/2nd Year/AI Knowledge Representation

```
Q . . . . . . .
. . . . Q . . .
Q . . . . . . .
. . . . . . . Q
. . . Q . . . .
. Q . . . . . .
. . . Q . . . .
Number of attacking pairs:  3
---------Gen End-----------
. . . . . . Q .
Q . . . . . . .
. . . . Q . . .
. Q . . . . . .
. . . . . . . Q
. . . . . Q . .
. . . Q . . . .
. Q . . . . . .
Number of attacking pairs:  2
---------Gen End-----------
. . Q . . . . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . . . . . Q
. . . . . Q . .
. . . Q . . . .
Q . . . . . . .
Number of attacking pairs:  1
---------Gen End-----------
. Q . . . . . .
. . . . Q . . .
. . . . . . Q .
. . . Q . . . .
Q . . . . . . .
. . . . . . . Q
. . . . . Q . .
. . Q . . . . .
Number of attacking pairs:  0
---------Gen End-----------
```

Conclusion: Hence successfully implemented Genetics Algorithm on 8-queens problem. Following points were concluded / noted while implementing :

1. G.A. is based on Probabilistic Rules.
2. Mutation doesn't always give best results.
3. 12 unique solutions to 8-queens problem.