SD21063 TEAN JIN HE Lab Report 2

April 12, 2023

1 BSD2513 ARTIFICIAL INTELLIGENCE

1.1 LAB REPORT 2

NAME : TEAN JIN HE

MATRIC ID : SD21063

SECTION : 02G

Questions 1 : General Knowledge Discuss three applications of genetic algorithms in real-world phenomena. Give references.

[6]:
```
#Three applications of genetic algorithms in real-world phenomena are:

#1.Engineering Design
#Engineering design is the process of creating and developing products or␣
↪systems that meet certain requirements and specifications. Genetic␣
↪algorithms can be applied to engineering design problems to find optimal or␣
↪near-optimal solutions that satisfy multiple objectives and constraints.
#Reference: Iowa State University Ames, Iowa 2007 Xiaopeng Fang, 2007.
#           https://dr.lib.iastate.edu/handle/20.500.12876/69627

#2.Robotics
#Genetic algorithms can be used to evolve the behavior and control of robots,␣
↪such as navigation, obstacle avoidance, coordination, and learning.
#Reference: Chris Messom Institute of Information and Mathematical Sciences,␣
↪Massey University, Albany Campus, Auckland, New Zealand
#           https://dr.lib.iastate.edu/entities/publication/
↪a0deb3ac-ac1f-4027-884a-a5da36a16ea7

#3.Medical Science
#Medical science is the science of diagnosing, treating, and preventing␣
↪diseases and disorders. Genetic algorithms can be applied to medical science␣
↪problems to find optimal or near-optimal solutions that improve the quality␣
↪and efficiency of health care.
```

```
#Reference: Ali Ghaheri, Saeed Shoar, Mohammad Naderan and Sayed Shahabuddin␣
↪Hoseini,Department of Management and Economy, Science and Research Branch,␣
↪Azad University, Tehran, Iran2Department of Surgery, Shariati Hospital,␣
↪Tehran University of Medical Sciences, Tehran, Iran3School of Medicine␣
↪Tehran University of Medical Sciences, Tehran, Iran4Hannover Medical School,␣
↪Germany
#          https://www.researchgate.net/publication/
↪283498449_The_Applications_of_Genetic_Algorithms_in_Medicine
```

Question 2 Python: Search Algorithms (Genetic Algorithm) Generate a bit pattern with predefined parameters from genetic algorithms. You are required to consider this condition as follows:

1. Population set up is 300.
2. The formula used in the preceding function reaches its maximum value when the number of one equal 50.
3. The length of all individuals is 80.
4. When the number of ones equals 50, the return value would be 80.
5. Number of generations is 50.

[4]:
```
pip install deap
```

```
Requirement already satisfied: deap in d:\anaconda\lib\site-packages (1.3.3)
Requirement already satisfied: numpy in d:\anaconda\lib\site-packages (from
deap) (1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

[5]:
```
import random

from deap import base, creator, tools

# Evaluation function
def eval_func(individual):
    target_sum = 50
    return len(individual) - abs(sum(individual) - target_sum),

# Create the toolbox with the right parameters
def create_toolbox(num_bits):
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMax)

    # Initialize the toolbox
    toolbox = base.Toolbox()

    # Generate attributes
    toolbox.register("attr_bool", random.randint, 0, 1)

    # Initialize structures
```

```python
    toolbox.register("individual", tools.initRepeat, creator.Individual,
        toolbox.attr_bool, num_bits)

    # Define the population to be a list of individuals
    toolbox.register("population", tools.initRepeat, list, toolbox.individual)

    # Register the evaluation operator
    toolbox.register("evaluate", eval_func)

    # Register the crossover operator
    toolbox.register("mate", tools.cxTwoPoint)

    # Register a mutation operator
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)

    # Operator for selecting individuals for breeding
    toolbox.register("select", tools.selTournament, tournsize=3)

    return toolbox

if __name__ == "__main__":
    # Define the number of bits
    num_bits = 80

    # Create a toolbox using the above parameter
    toolbox = create_toolbox(num_bits)

    # Seed the random number generator
    random.seed(7)

    # Create an initial population of 500 individuals
    population = toolbox.population(n=300)

    # Define probabilities of crossing and mutating
    probab_crossing, probab_mutating  = 0.5, 0.2

    # Define the number of generations
    num_generations = 50

    print('\nStarting the evolution process')

    # Evaluate the entire population
    fitnesses = list(map(toolbox.evaluate, population))
    for ind, fit in zip(population, fitnesses):
        ind.fitness.values = fit

    print('\nEvaluated', len(population), 'individuals')
```

```python
# Iterate through generations
for g in range(num_generations):
    print("\n===== Generation", g)

    # Select the next generation individuals
    offspring = toolbox.select(population, len(population))

    # Clone the selected individuals
    offspring = list(map(toolbox.clone, offspring))

    # Apply crossover and mutation on the offspring
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
        # Cross two individuals
        if random.random() < probab_crossing:
            toolbox.mate(child1, child2)

            # "Forget" the fitness values of the children
            del child1.fitness.values
            del child2.fitness.values

    # Apply mutation
    for mutant in offspring:
        # Mutate an individual
        if random.random() < probab_mutating:
            toolbox.mutate(mutant)
            del mutant.fitness.values

    # Evaluate the individuals with an invalid fitness
    invalid_ind = [ind for ind in offspring if not ind.fitness.valid]
    fitnesses = map(toolbox.evaluate, invalid_ind)
    for ind, fit in zip(invalid_ind, fitnesses):
        ind.fitness.values = fit

    print('Evaluated', len(invalid_ind), 'individuals')

    # The population is entirely replaced by the offspring
    population[:] = offspring

    # Gather all the fitnesses in one list and print the stats
    fits = [ind.fitness.values[0] for ind in population]

    length = len(population)
    mean = sum(fits) / length
    sum2 = sum(x*x for x in fits)
    std = abs(sum2 / length - mean**2)**0.5
```

```
        print('Min =', min(fits), ', Max =', max(fits))
        print('Average =', round(mean, 2), ', Standard deviation =',
                round(std, 2))

    print("\n==== End of evolution")

    best_ind = tools.selBest(population, 1)[0]
    print('\nBest individual:\n', best_ind)
    print('\nNumber of ones:', sum(best_ind))
```

```
Starting the evolution process

Evaluated 300 individuals

===== Generation 0
Evaluated 191 individuals
Min = 61.0 , Max = 80.0
Average = 73.63 , Standard deviation = 3.4

===== Generation 1
Evaluated 165 individuals
Min = 66.0 , Max = 80.0
Average = 76.21 , Standard deviation = 2.52

===== Generation 2
Evaluated 192 individuals
Min = 70.0 , Max = 80.0
Average = 77.63 , Standard deviation = 1.94

===== Generation 3
Evaluated 176 individuals
Min = 73.0 , Max = 80.0
Average = 78.6 , Standard deviation = 1.37

===== Generation 4
Evaluated 179 individuals
Min = 71.0 , Max = 80.0
Average = 78.53 , Standard deviation = 1.67

===== Generation 5
Evaluated 180 individuals
Min = 73.0 , Max = 80.0
Average = 78.73 , Standard deviation = 1.43

===== Generation 6
Evaluated 179 individuals
Min = 72.0 , Max = 80.0
```

```
Average = 78.72 , Standard deviation = 1.58

===== Generation 7
Evaluated 180 individuals
Min = 71.0 , Max = 80.0
Average = 78.8 , Standard deviation = 1.43

===== Generation 8
Evaluated 172 individuals
Min = 73.0 , Max = 80.0
Average = 78.86 , Standard deviation = 1.38

===== Generation 9
Evaluated 180 individuals
Min = 73.0 , Max = 80.0
Average = 78.74 , Standard deviation = 1.53

===== Generation 10
Evaluated 191 individuals
Min = 73.0 , Max = 80.0
Average = 78.96 , Standard deviation = 1.24

===== Generation 11
Evaluated 174 individuals
Min = 72.0 , Max = 80.0
Average = 78.89 , Standard deviation = 1.58

===== Generation 12
Evaluated 167 individuals
Min = 73.0 , Max = 80.0
Average = 79.06 , Standard deviation = 1.29

===== Generation 13
Evaluated 181 individuals
Min = 70.0 , Max = 80.0
Average = 78.8 , Standard deviation = 1.53

===== Generation 14
Evaluated 168 individuals
Min = 73.0 , Max = 80.0
Average = 79.02 , Standard deviation = 1.34

===== Generation 15
Evaluated 189 individuals
Min = 74.0 , Max = 80.0
Average = 78.92 , Standard deviation = 1.37

===== Generation 16
```

```
Evaluated 189 individuals
Min = 71.0 , Max = 80.0
Average = 78.84 , Standard deviation = 1.49

===== Generation 17
Evaluated 174 individuals
Min = 74.0 , Max = 80.0
Average = 78.93 , Standard deviation = 1.42

===== Generation 18
Evaluated 192 individuals
Min = 73.0 , Max = 80.0
Average = 78.69 , Standard deviation = 1.48

===== Generation 19
Evaluated 177 individuals
Min = 74.0 , Max = 80.0
Average = 78.87 , Standard deviation = 1.31

===== Generation 20
Evaluated 162 individuals
Min = 73.0 , Max = 80.0
Average = 79.04 , Standard deviation = 1.21

===== Generation 21
Evaluated 189 individuals
Min = 73.0 , Max = 80.0
Average = 78.98 , Standard deviation = 1.21

===== Generation 22
Evaluated 190 individuals
Min = 71.0 , Max = 80.0
Average = 78.8 , Standard deviation = 1.55

===== Generation 23
Evaluated 175 individuals
Min = 73.0 , Max = 80.0
Average = 78.68 , Standard deviation = 1.62

===== Generation 24
Evaluated 176 individuals
Min = 74.0 , Max = 80.0
Average = 78.88 , Standard deviation = 1.39

===== Generation 25
Evaluated 174 individuals
Min = 72.0 , Max = 80.0
Average = 78.88 , Standard deviation = 1.51
```

```
===== Generation 26
Evaluated 189 individuals
Min = 73.0 , Max = 80.0
Average = 78.85 , Standard deviation = 1.32

===== Generation 27
Evaluated 189 individuals
Min = 74.0 , Max = 80.0
Average = 78.81 , Standard deviation = 1.33

===== Generation 28
Evaluated 168 individuals
Min = 71.0 , Max = 80.0
Average = 78.97 , Standard deviation = 1.45

===== Generation 29
Evaluated 176 individuals
Min = 72.0 , Max = 80.0
Average = 79.05 , Standard deviation = 1.31

===== Generation 30
Evaluated 181 individuals
Min = 71.0 , Max = 80.0
Average = 78.86 , Standard deviation = 1.52

===== Generation 31
Evaluated 194 individuals
Min = 72.0 , Max = 80.0
Average = 78.84 , Standard deviation = 1.42

===== Generation 32
Evaluated 167 individuals
Min = 74.0 , Max = 80.0
Average = 79.0 , Standard deviation = 1.27

===== Generation 33
Evaluated 180 individuals
Min = 71.0 , Max = 80.0
Average = 78.74 , Standard deviation = 1.56

===== Generation 34
Evaluated 174 individuals
Min = 74.0 , Max = 80.0
Average = 78.99 , Standard deviation = 1.23

===== Generation 35
Evaluated 165 individuals
```

```
Min = 75.0 , Max = 80.0
Average = 79.15 , Standard deviation = 1.09

===== Generation 36
Evaluated 177 individuals
Min = 75.0 , Max = 80.0
Average = 79.02 , Standard deviation = 1.14

===== Generation 37
Evaluated 169 individuals
Min = 74.0 , Max = 80.0
Average = 78.94 , Standard deviation = 1.25

===== Generation 38
Evaluated 183 individuals
Min = 72.0 , Max = 80.0
Average = 78.77 , Standard deviation = 1.53

===== Generation 39
Evaluated 182 individuals
Min = 73.0 , Max = 80.0
Average = 78.8 , Standard deviation = 1.38

===== Generation 40
Evaluated 169 individuals
Min = 74.0 , Max = 80.0
Average = 79.08 , Standard deviation = 1.19

===== Generation 41
Evaluated 176 individuals
Min = 73.0 , Max = 80.0
Average = 78.91 , Standard deviation = 1.36

===== Generation 42
Evaluated 180 individuals
Min = 74.0 , Max = 80.0
Average = 78.93 , Standard deviation = 1.35

===== Generation 43
Evaluated 193 individuals
Min = 72.0 , Max = 80.0
Average = 78.79 , Standard deviation = 1.46

===== Generation 44
Evaluated 202 individuals
Min = 73.0 , Max = 80.0
Average = 78.83 , Standard deviation = 1.36
```

```
===== Generation 45
Evaluated 177 individuals
Min = 75.0 , Max = 80.0
Average = 79.01 , Standard deviation = 1.18

===== Generation 46
Evaluated 186 individuals
Min = 72.0 , Max = 80.0
Average = 78.87 , Standard deviation = 1.42

===== Generation 47
Evaluated 163 individuals
Min = 72.0 , Max = 80.0
Average = 79.05 , Standard deviation = 1.24

===== Generation 48
Evaluated 165 individuals
Min = 74.0 , Max = 80.0
Average = 79.06 , Standard deviation = 1.33

===== Generation 49
Evaluated 188 individuals
Min = 72.0 , Max = 80.0
Average = 78.81 , Standard deviation = 1.48

==== End of evolution

Best individual:
 [0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1,
1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1]

Number of ones: 50
```