|  |  |  |
| --- | --- | --- |
| **Module 3** | **Evolutionary Computation & Intelligent Agents** | **15** |
|  | * Simulate genetic algorithm with suitable example using Python / R or any other platform. * Design intelligent agent using any AI algorithm. (e.g. design tic-tac-toe game, design expert tutoring system) |  |

|  |  |  |
| --- | --- | --- |
| **Module 4** | **Knowledge Representation and Language Decoding** | **15** |
|  | * Design an application to simulate language parser. * Develop the semantic net using python. |  |

Practical 1

**Aim:**

To create a Python program that functions as a basic math tutor, capable of explaining and performing basic arithmetic operations (addition, subtraction, multiplication, and division). The program should help users understand the operations and perform calculations based on user input.

class MathTutor:

def \_\_init\_\_(self):

self.operations = {

'+': lambda a, b: a + b,

'-': lambda a, b: a - b,

'\*': lambda a, b: a \* b,

'/': lambda a, b: a / b,

}

def explain\_operation(self, operator):

explanation = {

'+': "Addition adds two numbers together.",

'-': "Subtraction subtracts the second number from the first.",

'\*': "Multiplication gives the product of two numbers.",

'/': "Division divides the first number by the second.",

}

return explanation.get(operator, "Invalid operation.")

def perform\_operation(self, operator, a, b):

if operator in self.operations:

return self.operations[operator](a, b)

else:

return None

if \_\_name\_\_ == "\_\_main\_\_":

tutor = MathTutor()

# Example usage:

operator = '+'

a, b = 10, 5

print(tutor.explain\_operation(operator))

result = tutor.perform\_operation(operator, a, b)

print(f"Result of {a} {operator} {b} = {result}")

output

Program : Genetic Algorithm to Solve a Simple String Matching Problem

This example demonstrates a genetic algorithm that evolves a population of strings to match a target string.

import random

import string

# Genetic Algorithm parameters

target\_string = "HELLO"

population\_size = 50 # Increased population size

mutation\_rate = 0.01

generations = 200 # Increased generations for more evolution

# Fitness function: number of characters matching the target

def fitness(individual):

return sum(1 for a, b in zip(individual, target\_string) if a == b)

# Create initial population (random strings)

def create\_population(size):

return [''.join(random.choices(string.ascii\_uppercase, k=len(target\_string))) for \_ in range(size)]

# Select parents (tournament selection)

def select\_parents(population):

tournament = random.sample(population, 5) # Select 5 individuals instead of 3 for better diversity

return max(tournament, key=fitness)

# Crossover (single-point crossover)

def crossover(parent1, parent2):

crossover\_point = random.randint(1, len(parent1) - 1)

return parent1[:crossover\_point] + parent2[crossover\_point:]

# Mutation (random character mutation)

def mutate(individual):

individual = list(individual)

for i in range(len(individual)):

if random.random() < mutation\_rate:

individual[i] = random.choice(string.ascii\_uppercase)

return ''.join(individual)

# Main genetic algorithm loop

population = create\_population(population\_size)

for generation in range(generations):

best\_individual = max(population, key=fitness)

print(f"Generation {generation}: Best individual: {best\_individual}, Fitness: {fitness(best\_individual)}")

if fitness(best\_individual) == len(target\_string): # Stop early if the optimal solution is found

break

# Create new generation

new\_population = []

for \_ in range(population\_size):

parent1 = select\_parents(population)

parent2 = select\_parents(population)

child = crossover(parent1, parent2)

child = mutate(child)

new\_population.append(child)

population = new\_population

# Best individual in the final population

best\_individual = max(population, key=fitness)

print(f"Best individual: {best\_individual}, Fitness: {fitness(best\_individual)}")

output

practical 3

To design and implement a simple language parser that can evaluate basic arithmetic expressions involving addition, subtraction, multiplication, and division.

Refer python file

Practical 4

Refer python file