0094_genetic_algorithm_1_code_only

September 22, 2019

[ ]:
```
import random
import numpy as np

def create_reference_solution(chromosome_length):

    number_of_ones = int(chromosome_length / 2)

    # Build an array with an equal mix of zero and ones
    reference = np.zeros(chromosome_length)
    reference[0: number_of_ones] = 1

    # Shuffle the array to mix the zeros and ones
    np.random.shuffle(reference)

    return reference


def create_starting_population(individuals, chromosome_length):
    # Set up an initial array of all zeros
    population = np.zeros((individuals, chromosome_length))
    # Loop through each row (individual)
    for i in range(individuals):
        # Choose a random number of ones to create
        ones = random.randint(0, chromosome_length)
        # Change the required number of zeros to ones
        population[i, 0:ones] = 1
        # Sfuffle row
        np.random.shuffle(population[i])

    return population


def calculate_fitness(reference, population):
    # Create an array of True/False compared to reference
    identical_to_reference = population == reference
    # Sum number of genes that are identical to the reference
```

```python
    fitness_scores = identical_to_reference.sum(axis=1)

    return fitness_scores


def select_individual_by_tournament(population, scores):
    # Get population size
    population_size = len(scores)

    # Pick individuals for tournament
    fighter_1 = random.randint(0, population_size-1)
    fighter_2 = random.randint(0, population_size-1)

    # Get fitness score for each
    fighter_1_fitness = scores[fighter_1]
    fighter_2_fitness = scores[fighter_2]

    # Identify undividual with highest fitness
    # Fighter 1 will win if score are equal
    if fighter_1_fitness >= fighter_2_fitness:
        winner = fighter_1
    else:
        winner = fighter_2

    # Return the chromsome of the winner
    return population[winner, :]


def breed_by_crossover(parent_1, parent_2):
    # Get length of chromosome
    chromosome_length = len(parent_1)

    # Pick crossover point, avoding ends of chromsome
    crossover_point = random.randint(1,chromosome_length-1)

    # Create children. np.hstack joins two arrays
    child_1 = np.hstack((parent_1[0:crossover_point],
                        parent_2[crossover_point:]))

    child_2 = np.hstack((parent_2[0:crossover_point],
                        parent_1[crossover_point:]))

    # Return children
    return child_1, child_2


def randomly_mutate_population(population, mutation_probability):
```

```python
    # Apply random mutation
        random_mutation_array = np.random.random(
            size=(population.shape))

        random_mutation_boolean = \
            random_mutation_array <= mutation_probability

        population[random_mutation_boolean] = \
        np.logical_not(population[random_mutation_boolean])

        # Return mutation population
        return population

# ***************************************
# ******** MAIN ALGORITHM CODE ********
# ***************************************

# Set general parameters
chromosome_length = 75
population_size = 500
maximum_generation = 200
best_score_progress = [] # Tracks progress

# Create reference solution
# (this is used just to illustrate GAs)
reference = create_reference_solution(chromosome_length)

# Create starting population
population = create_starting_population(population_size, chromosome_length)

# Display best score in starting population
scores = calculate_fitness(reference, population)
best_score = np.max(scores)/chromosome_length * 100
print ('Starting best score, % target: ',best_score)

# Add starting best score to progress tracker
best_score_progress.append(best_score)

# Now we'll go through the generations of genetic algorithm
for generation in range(maximum_generation):
    # Create an empty list for new population
    new_population = []

    # Create new popualtion generating two children at a time
    for i in range(int(population_size/2)):
        parent_1 = select_individual_by_tournament(population, scores)
```

```python
        parent_2 = select_individual_by_tournament(population, scores)
        child_1, child_2 = breed_by_crossover(parent_1, parent_2)
        new_population.append(child_1)
        new_population.append(child_2)

    # Replace the old population with the new one
    population = np.array(new_population)

    # Apply mutation
    mutation_rate = 0.002
    population = randomly_mutate_population(population, mutation_rate)

    # Score best solution, and add to tracker
    scores = calculate_fitness(reference, population)
    best_score = np.max(scores)/chromosome_length * 100
    best_score_progress.append(best_score)

# GA has completed required generation
print ('End best score, % target: ', best_score)

# Plot progress
%matplotlib inline # For Jypyter notebook only
import matplotlib.pyplot as plt
plt.plot(best_score_progress)
plt.xlabel('Generation')
plt.ylabel('Best score (% target)')
plt.show()
```