CS3642-W02 Programming Assignment #2 (Spring 2024)

Total Points: 100

Due: February 25, 2024 (11:59 PM, EST)

[**100 points**] Implement Genetic Algorithm (GA) to find the minimal value of $f(x) = x^2, where\ 0 \leq x \leq 63$ and x is an integer.

1. [**20 points**] Implement a random vector generation function to generate the population. The state should be stored in a vector with 6 binary digits. Implement a function named generate_population(n_pop) to generate the initial solutions randomly, where n_pop is the population size.

   Instruction:

   Implement a function generate_population(n_pop) that generates a random population of size n_pop. Each individual in the population should be represented as a binary vector of length 6, representing integers in the range of [0,63]. The desired output of generate_population(n_pop) is like:

   [[0,0,0,0,0,1], [1,0,1,0,0,0]]

   when n_pop is set as 2. It returns a list of lists, where each element represents the initial number represented in binary form. In the above case, [0, 0, 0, 0, 0, 1] represents the binary form of the integer 1, and [1, 0, 0, 0, 0, 0] represents the binary form of the integer 40.

2. [**60 points**] Implement these 5 functions:

- fitness(c): c is a vector. It converts the vector from binary array into integer and calculate $f(x) = x^2$.
- crossover(c1, c2): c1 and c2 are two solutions. Generate a random index to perform single point crossover
- mutation(c, p_m): p_m is the probability for mutation
- selection(population): population is a list of solutions. Use Elite selection (pages 26, 37, and 38 on M4_BeyondClassicalSearch_GA.pptx) to perform reproduction.
- evolution(population): perform GA, it may use the above three functions. *Print the optimal solution* when iteration stops.
- check(population): check if there is an optimal solution.

Instruction:

1) crossover(c1, c2): Generate a random index to perform a single-point crossover. Combine the binary vectors at the chosen index. Return the two offspring created through crossover. Suppose two solutions are [0,0,0,1,0,1] and [0,0,1,1,1,1], the randomly generated index is 4. Then the results of single-point crossover, i.e. two generated offspring, are shown below. The desired return values are shown in 'combine' column.

Original c1 and c2	'cut'	'combine'
[0,0,0,1,0,1]	[0,0,0,1,0,1]	[0,0,0,1,1,1]
[0,0,1,1,1,1]	[0,0,1,1,1,1]	[0,0,1,1,0,1]

2) mutation(c, p_m): For each bit in the binary vector, apply mutation with a probability of p_m. If mutation occurs, flip the bit. Return the mutated binary vector.
3) selection(population) that takes a list of binary vectors population as input. To implement Elite selection, we need to rank order the individuals based on their fitness, discard the bottom half, and double the remaining. Then, return the selected parents for reproduction.
4) evolution(population) performs the GA operations: selection, crossover, and mutation. The pseudo code for evolution is shown below:

```
n_generation = 0
while(check(population)==False):

    # selection
    population = selection(population)
    # crossover
    offspring_list = []
    for c1, c2 in population:
        offspring = crossover(c1, c2)
        offspring_list.append(offspring)
    population = population + offspring_list
    # mutation
    new_population = []
    for c in population:
        new_population.append(mutate(c, p_m))
    population = new_population
    n_generation += 1

return n_generation
```

5) check(population) iterates through each individual in the population. It uses the fitness function to evaluate the fitness of each individual. If an individual reaches the optimal fitness (in this case, the minimum value of $f(x)$ is 0 when $x = 0$), return True if at least one individual is found to be 0; otherwise return False.

3. [**20 points**] run GA for 50 times under different settings and provide a table of statistics below.

n_pop	p_m	Average # of generations	Average running time (milliseconds)
10	0.05		
100	0.05		
10	0.2		
100	0.2		

** Average # of generations: the number of average iterations for evolution.

** Note that the average running time should be reported in milliseconds rather than seconds.

You may write your code in a contemporary language of your choice; typical languages would include C/C++, Python, Java, Ada, Pascal, Smalltalk, Lisp, and Prolog.

Submission requirement:

1. Submit *a **PDF file*** of your well-commented source program, your design, and your printed outputs (**screen shots**). **Please include your codes in your PDF file.** It is plagiarism to take any codes from the website or others. Try to understand the algorithm and implement the algorithm by your own. You must have the following 2 sections in your PDF file.
2. Please submit ***your project in a zipped file*** with an organized structure.
3. Please upload the PDF file and project separately to D2L.

Adding the following 2 sections (I and II) at the beginning of your PDF including your code and outputs.

I. Your Information:

// Course:	________________
// Student name:	________________
// Student ID:	________________
// Assignment #:	________________
// Due Date:	________________
// Signature:	________________(Your signature assures that everything is your own work. Required.)
// Score:	________________(Note: Score will be posted on D2L)

II. A brief design of your algorithm, including the description, source code, and screenshots.

1. [**100 points**] GA

   [**20 points**] Population generation function

   [**60 points**] Implementation of these 6 functions:

- fitness(c):
- crossover(c1, c2):
- mutation(c, p_m):
- selection(population):.
- evolution(population):
- check(population):

[**20 points**] a table of statistics below. (one screenshot to show the program is runnable)

n_pop	p_m	Average # of generations	Average running time (milliseconds)
10	0.05		
100	0.05		
10	0.2		
100	0.2		