Assignment 2

Developing a Basic Genetic Optimization Algorithm in C

Pedram Pasandide

Due Date: 13 November 2023

1 Introduction

In this assignment, we will develop an optimization algorithm. Before we start talking about what a mathematical optimization is, take a look at A Few Optimization Problems to see where we use them.

Optimization problems involve the process of finding the best solution from a set of possible solutions. To understand how we formulate an optimization problem you can take a look at Appendix.

Genetic Algorithm (GA) is one of the well-known optimization algorithms developed many years ago. In a genetic algorithm, a population of individuals evolves over time. The stronger individuals reproduce, and the weaker ones die. The surviving ones can also mutate. With each individual, there is a fitness number that indicates how strong that individual is. For more details, see Genetic Algorithm. A genetic algorithm terminates when a given number of iterations is reached or when a solution is found, that is, an individual with certain fitness. The goal of this assignment is to **minimize** Ackley function using basic genetic algorithm approach. You can download all the files you need from my GitHub or from Avenue to Learn.

Take a look at the Python code `Opt_Ackley.ipynb` which I used to optimize the Ackley function using optimization packages available on Python. If you don't have Python on your OS, you can run your Python codes on Google Colab.

In this code, first I did some visualization for Ackley function to have a better understanding about the objective function. Ackley function has two variables x_1 and x_2 and by figure we can already see when $x_1 = 0$ and $x_2 = 0$, the minimum is happening. In this case we know what is the optimized solution by visualization but it is not always like this. We chose Ackley function a problem because it is easy do understand and suitable enough to find out about the efficiency of the GA program you develop. We will discuss `Opt_Ackley.ipynb` during our lectures this week.

GA is a popular optimization and search technique inspired by the process of natural selection and evolution. It's used to find approximate solutions to complex optimization and search problems. GAs are particularly effective when dealing with large solution spaces, combinatorial

optimization problems, and cases where the objective function is non-linear, discontinuous, or has multiple local optima.

Mutation and crossover functions are crucial components of Genetic Algorithms:

Mutation: Mutation is a random alteration of an individual's genetic code. It introduces diversity by making small changes in the solution space. For example, in a binary representation, mutation might flip a 0 to a 1 or vice versa. The mutation rate controls the probability of mutation for each gene.

Crossover (Recombination): Crossover involves the exchange of genetic material between two parents to create offspring. The way this exchange occurs depends on the specific crossover method used. For instance, in one-point crossover, a random point is selected in the genetic code, and the segments of the parents' genetic code are swapped at that point to create two offspring. In two-point crossover, two random points are selected for the exchange.

Important: Read a numerical example applying both crossover and mutation function. Also the Python code `mutation_cross.ipynb` that I provided applies the logic of a crossover and mutation function over some data for only one iteration. So, basically what you need to do is translate this code into C language. We discuss `mutation_cross.ipynb` during our lecture session this week.

2 Programming Genetic Algorithm

2.1 Implementation (12 points)

Files you have downloaded are:

- `OF.c` (Do not change anything in it)
- `functions.h` (Do not change anything in it)
- `GA.c`
- `functions.c`

- `OF.c`: The file `OF.c` has the implementation of the objective function which in this case it is Ackley function. **Do NOT change anything** in this file because you will not submit this file. This is how I defined Ackley function:

```
#include <math.h>
```

```
#include "functions.h"

double Objective_function(int NUM_VARIABLES, double x[
    NUM_VARIABLES])
{
 double sum1 = 0.0, sum2 = 0.0;
 for (int i = 0; i < NUM_VARIABLES; i++)
 {
  sum1 += x[i] * x[i];
  sum2 += cos(2.0 * M_PI * x[i]);
 }
 return -20.0 * exp(-0.2 * sqrt(sum1 / NUM_VARIABLES)) - exp
    (sum2 / NUM_VARIABLES) + 20.0 + M_E;
}
```

- `functions.h`: The file `functions.h` includes the declarations of all functions. **Do NOT change anything** in this file because you will not submit this file. This is what you see in `functions.h`:

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

double Objective_function(int NUM_VARIABLES, double x[
    NUM_VARIABLES]);

double generate_random(double min, double max);

unsigned int generate_int();

void generate_population(int POPULATION_SIZE, int
    NUM_VARIABLES, double population[POPULATION_SIZE][
    NUM_VARIABLES], double Lbound[NUM_VARIABLES], double
    Ubound[NUM_VARIABLES]);

void compute_objective_function(int POPULATION_SIZE, int
    NUM_VARIABLES, double population[POPULATION_SIZE][
    NUM_VARIABLES], double fitness[POPULATION_SIZE]);

void crossover(int POPULATION_SIZE, int NUM_VARIABLES,
    double fitness[POPULATION_SIZE], double new_population[
```

```
    POPULATION_SIZE][NUM_VARIABLES], double population[
    POPULATION_SIZE][NUM_VARIABLES], double crossover_rate);

void mutate(int POPULATION_SIZE, int NUM_VARIABLES, double
    new_population[POPULATION_SIZE][NUM_VARIABLES], double
    population[POPULATION_SIZE][NUM_VARIABLES], double Lbound
    [NUM_VARIABLES], double Ubound[NUM_VARIABLES], double
    mutate_rate);

#endif
```

- `GA.c`: The file `GA.c`, is the main code you have and based on what you need, you can all the function in `int main(int argc, char *argv[])`. The specifications of the algorithm is given by the user. Let's say if the executable file created after compiling the code is named `GA`, I must be able to run the code by the following command in the terminal:

  `./GA <POPULATION_SIZE> <MAX_GENERATIONS> <crossover_rate> <mutate_rate> <stop_criteria>`

  If you have followed the notes so far, you must know what are the input parameters, except `stop_criteria`. The early stopping criteria in a GA is a mechanism designed to terminate the algorithm prematurely if certain conditions are met before the maximum number of generations (MAX_GENERATIONS) is reached. One common early stopping criterion is based on the difference between the current best fitness and the previous best fitness. This is often referred to as a "convergence criteria" or "stagnation criteria." So I must be able to run you code by:

  `./GA 1000 10000 0.5 0.1 1e-16`

  where 1000 is population size, 10000 is maximum generation, 0.5 is crossover rate, the mutation rate is 0.1, and `1e-16` is the stopping criteria.

- `functions.c`: The real implementation of all functions are in `functions.c`. This is where you develop are the functions you need in `GA.c`. Follow the comments inside `functions.c` and complete the functions.

In general, for a maximum number of iterations, `MAX_GENERATIONS` given by user, the algorithm in the main loop will:

1. Compute the fitness values using `compute_objective_function` function.
2. Then it compute the probability of each set of decision variables in `population`.

3. Picking the stronger set of decision variables to reproduce in `crossover` function based on `crossover_rate`.

4. Applying mutation on the population based on `mutate_rate`.

5. Until the `MAX_GENERATIONS` is reached or stopping criteria is met.

The number of decision variables in the optimization problem is set to 2, with upper and lower bounds for both decision variables set to +5 and -5, respectively:

```
int NUM_VARIABLES = 2;
double Lbound[] = {-5.0, -5.0};
double Ubound[] = {5.0, 5.0};
```

Every time you execute the code, at the beginning you must print out some specification about you program. Let's say I execute the program by:

`./GA 100 10000 0.5 0.1 1e-16`

This is what I see in the terminal:

```
Genetic Algorithm is initiated.
-----------------------------------------------
The number of variables: 2
Lower bounds: [-5.000000, -5.000000]
Upper bounds: [5.000000, 5.000000]

Population Size:   100
Max Generations:   10000
Crossover Rate:    0.500000
Mutation Rate:     0.100000
Stopping criteria: 0.0000000000000001

Results
-----------------------------------------------
CPU time: 1.143935 seconds
Best solution found: (-0.0001323239496775, -0.0000225231135396)
Best fitness: 0.0003801313729501
```

And after the GA has obtained the best solution, you must print out the CPU time, the best solution, and the fitness value (value of Ackley function) of the best solution. In this example, after 10000 after about 1 second, for decision variables:

x1 = -0.0001323239496775

x2 = -0.0000225231135396,

the most minimum value the GA optimizer could get by minimizing the Ackley function is 0.0003801313729501. Fill out the following tables, tell us in your report what you see!

Table 1: Results with Crossover Rate = 0.5 and Mutation Rate = 0.05

Pop Size	Max Gen	Best Solution			CPU time (Sec)
		x_1	x_2	Fitness	
10	100				
100	100				
1000	100				
10000	100				
1000	1000				
1000	10000				
1000	100000				
1000	1000000				

Table 2: Results with Crossover Rate = 0.5 and Mutation Rate = 0.2

Pop Size	Max Gen	Best Solution			CPU time (Sec)
		x_1	x_2	Fitness	
10	100				
100	100				
1000	100				
10000	100				
1000	1000				
1000	10000				
1000	100000				
1000	1000000				

2.2 Report and `Makefile` (3 points)

Provide a `Makefile` to compile your code, and in the report with details tells us how your `Makefile` works. Your report **must** be in `README.tex` file. Other formats will not be accepted. Create the `README.pdf` file from `README.tex` and submit both formats. You can use LaTex file format that I have sent you.

2.3 Improving the Performance - Bonus (+1 points)

Why the fitness value could not reach to the minimum equal to zero? Based on what we saw on Python code, Ackley function has a minimum equal to zero when x1=0 and x2=0. If you try to

increase the number of population size (`POPULATION_SIZE`) and maximum number of generation (`MAX_GENERATIONS`) you might get better results computationally costs more time.

Improving GA means:

1. Higher accuracy
2. Less computationally expensive (faster)

Use `gprof` by adding the flag `-pg` during compiling your program. Run your code let's say with

```
./GA 1000 10000 0.5 0.3 1e-16
```

Then, execute the command line `gprof GA gmon.out > analysis.txt` in your terminal to create a `analysis.txt` file reporting the time taken by different functions in your code. In my computer the file `analysis.txt` is something like:

```
Each sample counts as 0.01 seconds.
%   cumulative   self              self     total
time   seconds   seconds    calls  ms/call  ms/call  name
92.27    13.49    13.49    10000     1.35     1.38  crossover
3.11     13.95     0.46    10000     0.05     0.07  mutate
2.74     14.35     0.40 21002000     0.00     0.00  generate_random
0.79     14.46     0.12 22490560     0.00     0.00  generate_int
0.65     14.55     0.10 10000000     0.00     0.00  Objective_function
0.21     14.59     0.03                             main
0.14     14.61     0.02    10000     0.00     0.01  compute_objective_function
0.10     14.62     0.01        1    15.00    15.04  generate_population
```

This means over 92% of time taken to execute the code was taken by the function `crossover`. There are more useful in formation in this report. This means if I want to improve my program to work faster I have to focus on `crossover` function. In terms of increasing the accuracy it could be anything. I tired many things to increase the accuracy so I know right know the problem is `crossover`. If you want you can ask me what did I try to find out `crossover` is stopping us to have better results.

There are several crossover methods in GAs, each with its own characteristics and advantages, and in previous section we tried only one method called **Single-Point Crossover**. In single-point crossover, a random crossover point is chosen, and the genes beyond that point in both parents are swapped to create two children.

The mutation function can also help us get a higher accuracy. Right now the method we used is called **Uniform Mutation**. For real-valued chromosomes, this method changes the value of a gene to a random value within its allowable range. Gaussian Mutation, Creep Mutation, Self-Adaptive Mutation, and many more are mutation methods in GAs, each with its own characteristics and purposes.

In your report file, `README.tex`, tell us what you improved in crossover and mutation function. And produce the following table with your improved program:

Table 3: Results with **Crossover Rate = ?** and **Mutation Rate = ?**

Pop Size	Max Gen	Best Solution			CPU time (Sec)
		x_1	x_2	Fitness	
100	100				
1000	100				
10000	100				
1000	1000				
1000	10000				
1000	100000				

In this case, you have to find the most optimum values for Crossover Rate and Mutation Rate.

2.4 Only the fastest program! - Bonus (+3 points)

If you haven't completed the prior section, progressing to this part won't serve much purpose. In addition to speed, your code must also attain a satisfactory level of accuracy.

In the world of job applications, hundreds of individuals may apply, but only a single candidate secures the position. This situation mirrors the workings of a genetic algorithm, where only the fittest survive. Similarly, in our context, resembling a Genetic Algorithm, the student who produces the swiftest code (at least 20% faster than their peers) will be awarded a +3 bonus.

3 Submission

Submit On GitLab following files:

- `GA.c`, `function.c`, and `Makefile`
- `README.tex` and `README.pdf`

To set up your GitLab you can read the instruction from `Setup Guide 1.docx` file. Your directory must be like `MT2MP3>A2>all_the_files_here`

Make sure to add, commit and push your files before the due date. You can verify your submission via web browser by inspecting the contents of your submission repo. If you have any question about the procedure of submitting in this way please feel free to email Alireza Daeijavad (daeijava@mcmaster.ca) with the subject "2MP3 Assignment 2".

4 Appendix

4.1 A Few Optimization Problems

Here are a few examples of optimization problems from different fields. The part **Optimization Method** for each example is just based on my general knowledge and they might not necessary work always!

1. **Chemical Engineering - Refinery Processes:**
   - **Objective:** Optimize the production of Liquefied Petroleum Gas (LPG), Aromatics, or Hydrogen while minimizing operating costs.
   - **Decision Variables:** Flow rates, temperatures, pressures, and compositions of various process streams, as well as catalyst selection.
   - **Constraints:** Temperature and pressure limits within specific equipment, material and energy balances, and purity requirements for the products.
   - **Optimization Method:** Nonlinear programming techniques, such as Sequential Quadratic Programming (SQP) or Genetic Algorithms, can be used to solve complex refinery optimization problems.
2. **Aerospace:**
   - **Objective:** Aircraft Wing Design for Maximum Efficiency
   - **Decision Variables:** Wing shape parameters (e.g., wing sweep, aspect ratio, airfoil shape), materials, and structural design.
   - **Constraints:** Weight constraints, stability, and strength requirements, as well as aerodynamic performance specifications.
   - **Optimization Method:** Computational fluid dynamics (CFD) simulations combined with multi-objective optimization algorithms, like multi-objective genetic algorithms, to find the trade-off between lift, drag, and structural weight.
3. **Civil Engineering - Traffic Signal Timing:**

- **Objective:** Optimize traffic signal timings to minimize congestion and reduce travel time.
- **Decision Variables:** Signal phase durations, cycle lengths, and offset timings for a network of traffic signals.
- **Constraints:** Traffic flow capacity, safety constraints, and legal requirements.
- **Optimization Method:** Dynamic programming or heuristic algorithms like traffic signal control optimization to minimize traffic congestion.

4. **Mechanical Engineering - HVAC System Design:**

- **Objective:** Optimize the design of a heating, ventilation, and air conditioning (HVAC) system to minimize energy consumption while maintaining indoor comfort.
- **Decision Variables:** Equipment selection, duct sizing, control strategies, and insulation materials.
- **Constraints:** Indoor temperature and humidity requirements, building occupancy, and budget constraints.
- **Optimization Method:** Genetic algorithms or simulated annealing for multi-objective HVAC system design.

5. **Environmental Engineering - Water Treatment Plant Operation:**

- **Objective:** Optimize the operation of a water treatment plant to minimize chemical usage and energy consumption.
- **Decision Variables:** Flow rates, chemical dosages, treatment processes, and equipment settings.
- **Constraints:** Water quality standards, equipment capacity, and safety regulations.
- **Optimization Method:** Linear programming or mixed-integer programming for efficient operation of water treatment plants.

6. **Electrical Engineering - Power Grid Optimization:**

- **Objective:** Optimize the operation of an electrical power grid to ensure reliable and efficient electricity distribution while minimizing operational costs.
- **Decision Variables:** Power generation levels from various sources (e.g., coal, natural gas, renewables), routing of power through transmission lines, and voltage control settings.
- **Constraints:** Power demand at different locations, line capacity limits, voltage constraints, and environmental regulations.

- **Optimization Method:** Linear programming or mixed-integer linear programming can be used to optimize power generation and distribution in real-time, taking into account changing demand and constraints. Advanced methods may involve optimal power flow (OPF) and security-constrained optimal power flow (SCOPF) models for large-scale power grids.

These examples demonstrate the diversity of optimization problems in various engineering fields, as well as the wide range of decision variables and constraints involved in real-world applications. In each case, the objective is to find the best possible configuration or solution while considering the specific requirements and limitations of the domain.

4.2 What an Optimization Algorithm Is!

Optimization problems involve the process of finding the best solution from a set of possible solutions. These problems can be broadly described as follows:

Objective Function: Optimization problems typically begin with the definition of an objective function, which is a mathematical expression or a simulation that quantifies what you want to maximize (e.g., profit, efficiency) or minimize (e.g., cost, error). This function depends on one or more decision variables.

Decision Variables: Decision variables are the parameters or variables that you can adjust or control in the problem. The objective function is often expressed in terms of these variables. The goal is to find the values of these variables that optimize the objective function. Let's say if $x1$ and $x2$ are decision variables, the objective function OF can be defined as **an example** like:

$$OF = \frac{exp(x_1/x_2) + sin(x_2)}{x_1^2 + \sqrt{x_2}}$$

Constraints: Many real-world optimization problems come with constraints, which are conditions or limitations that the solution must satisfy. Constraints can be of various types, such as equality constraints (e.g., $x1 + x2 = 10$) and inequality constraints (e.g., $x1 >= 0, x2 <= 5$). The solution must adhere to these constraints while optimizing the objective function.

Optimization Direction: Depending on the problem, you may be seeking to either maximize or minimize the objective function. These are known as maximization and minimization problems, respectively.

Feasible Region: The feasible region is the set of all possible solutions that satisfy the given constraints. It represents the space in which you can search for the optimal solution.

Optimal Solution: The optimal solution is the set of values for the decision variables that both

satisfy the constraints and either maximize or minimize the objective function. In maximization problems, this is the highest attainable value; in minimization problems, it is the lowest attainable value.

Search and Optimization Methods: Solving optimization problems often involves using various mathematical and computational methods. These can include linear programming, nonlinear programming, dynamic programming, genetic algorithms, gradient descent, and many others. The choice of method depends on the problem's characteristics, such as linearity, convexity, and the number of variables and constraints.

Sensitivity Analysis: In some cases, it's essential to understand how sensitive the optimal solution is to changes in the problem's parameters. Sensitivity analysis helps assess how robust or fragile the solution is to variations in input data.

Multi-Objective Optimization: In some scenarios, there may be multiple conflicting objectives that need to be considered simultaneously. Multi-objective optimization aims to find a set of solutions that represent trade-offs between these objectives, known as the Pareto front.

Optimization problems are ubiquitous in various fields, including engineering, economics, operations research, logistics, machine learning, and more. They play a crucial role in decision-making, resource allocation, and improving system performance. The choice of method and the complexity of solving an optimization problem depend on the specific problem's characteristics and the available computational resources.