# Introduction

As I was completing this last project for the class, I could not help but reflect on the new set of tools and skills that I have acquired which fulfills my quest for knowledge and will benefit my future research. I am pleased, and somewhat amazed, at the simplicity of the concept of genetic algorithms and the value that it provides in solving problems incalculable with traditional, brute force programming magic. I also welcome the generality of GA where a simple programming structure can easily be adapted to panoply of problems with few modifications to the fitness function, the structure of a genotype and possibly few numerical parameters.

Extending the GA to a multi-objective function proved to be a simple extension to the code set of the previous project. Most of the effort was devoted to the implementation of the ranking algorithm classifying individuals according to their dominance relation to the rest of the population. We implemented both single and double point cross over, and experimented with different values of parameters to examine the effect on convergence and solution space.

# GA Implementation

The algorithm used for this project is a standard GA with a constant population size of 100 using both single and double point cross over as a recombination operator and bit-flip for a mutation operator. We ran the algorithm for 200 generations and plotted the Pareto optimal front for generations 1, 100 and 200 using multiple values of parameters. Given the choice for a genotype, we opted for an 8x32-bit binary string (an integer) for the sake of computational simplicity and performance[[1]](#footnote-2). Floating point arithmetic is used for all further calculations.

This bi-objective optimization problem call for minimizing the following two functions simultaneously using the Pareto optimal algorithm:

*where*: n = 8 and *xi* ϵ [-2,2].

We used the following function to map the 32-bit genotype to calculate *x­­i* while spanning the range of [-2, 2]:

*where:* represents the 32-bit binary genotype.

It is clear from the specification of *f1* and *f2* that the two objectives are in conflict when we attempt to minimize both at the same time. Assuming a value of *xi* = for each *x* component would bring *f­1*to zero, an absolute minimum, but would catapult *f2* to 0.98 which is not a minimum. Similarly, the opposite is true for *f2* with a value of *xi* = . Also notice that all values of the 32-bit binary string always produce *feasible* solutions since the entire range of the 32-bit integer is mapped to the range [-2, 2] as specified by the above equation. This makes the implementation of crossover and mutation simple where no further *correction* or *penalty* terms are necessary.

The following parameters were used for implementation:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| ***Number of runs*** | 1 |
| ***Population size*** | 100 |
| ***¿*** | 0.1, 0.5 and 2 |
| ***Pm*** | 0.05 |
| ***Pr*** | 0.45 and 0.7 |
| ***Mutation*** | Bit flip with Pm probability |
| ***Recombination*** | One and Two Point Crossover on 32-bit boundary |
| ***Selection*** | Roulette Wheel with Rank based selection and dominance based ranking |

# The Algorithm

**Program Initialize:**

1. Pm:= 0.05;

Pr := {0.45, 0.7}; // tried two values

¿ := {0.1, 0.5, 2}; // tried three values

P(0) := {p1 , p2, … p**100**};

g := 0;

**while** (g < 200)

2. (g) = **RankUsingDominance**(P, 1); // initial rank = 1

**CalculateProbability**( // rank proportional

3. **for** i:=1 to 100

P1,P2 = **SelectWithRoulette**();

**if** (randn < Pr)

Oi(g) = **Recombine (**P1, P2); // 2-point or 1-point xover

**else**

Oi(g) = **MinRank(**P1, P2); // Least dominated of the two; random selection if equal rank.

**end if**

**Mutate(**Oi(g));

**end for**

4. P(g+1) = O(g);

g :**=** g + 1;

**end while**

**Function RankUsingDominance**(P(g)

5. **if** (P(g) empty) **return**; // All members have been ranked

6. **CalculateDominance**(P(g));

7. **for** i:=1 to 100

**if** (**!** **Dominated**(Pi))

P*i*.Rank = *rank;*

P(g) = P(g) - P*i* ;

**end if**

**end for**

8. **RankUsingDominance**(P(g), *rank + 1*); // recursive call with increased rank

# Observation and Analysis:

Multiple runs of the solution for this MOP was run with a variety of parameters as shown in Figures 1-4. The graphs show the Pareto optimal solution frontier for generations 1 (diamonds), 100 (squares) and 200 (triangles). For all choices of parameters, the algorithm initially discovered solutions where both functions were close to their maxima. As of generation 100, all variants gravitated toward a set of solutions (the Pareto frontier) exhibiting an minimal equilibrium for both functions. The frontier covered pairs where *f1* was much closer to a minimum which brought *f2* to away from its minimum, pairs on the other extreme and pairs in between. As generation 200 was reached, the algorithm could not produce better solutions than it found in generation 100, but was able, in some situations, to concentrate the solutions in a tighter range than what was found in generation 100.

The choice of a crossover operator hada visible impact on the results as shown in the figures below. The first figure shows a one-point crossover and the remaining plots use a two-point crossover. Using a two-point crossover always resulted in solutions that are more concentrated around the same region than using a single-point crossover.

Figure One point Crossover, Pc=0.45, Pm=0.05, ¿=0.5

Figure 2 Two point Crossover, Pc=0.7, Pm=0.05, ¿=0.5

The choice of ¿ had an observable impact on the results as shown in the two figures below. A choice of ¿= 0.1, resulted in a scattering effect and degradation of the solution as generation 200 was reached. Increasing ¿ to 2.0 resulted in the best solution out of all the runs where the results are concentrated around the frontier, and the crescent shape of the frontier at generations 100 and 200 better exhibits the tug of war between the two functions. This shape brings to mind the construction of an ellipse where a string is pinned at the two radii and a pencil pulling the string tight to the perimeter of the ellipse. As the pencil moves, the distance to one of the radii increases at the expense of the other. The crescent shape of the last figure depicts such a construction.

Figure 3 Two point Crossover, Pc=0.7, Pm=0.05, ¿=0.1

Figure 4 Two point Crossover, Pc=0.7, Pm=0.05, ¿=2.0

Solutions from the 200 generation of the plot shown in Figure 4:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Solution 1** |  | **Solution 2** |  | **Solution 3** |
| ***x1*** | 0x8F4BCFD1 |  | 0x82444A67 |  | 0x81EAF8E7 |
| ***x2*** | 0x841F3988 |  | 0x81EA9788 |  | 0x806ADF98 |
| ***x3*** | 0x7D5F2E3F |  | 0x756D7977 |  | 0x657F3A77 |
| ***x4*** | 0x7F408701 |  | 0x73D301A4 |  | 0x719D15C5 |
| ***x5*** | 0x7D2AF3BE |  | 0x7A9179EB |  | 0x72BB16C8 |
| ***x6*** | 0x7F7840CC |  | 0x7F68C098 |  | 0x73141076 |
| ***x7*** | 0x7EAE08B1 |  | 0x79A8B4BF |  | 0x74B8D0BE |
| ***x8*** | 0x8193DAB7 |  | 0x81B3F037 |  | 0x80F76060 |
| ***f1()*** | 0.833387 |  | 0.628141 |  | 0.569944 |
| ***f2()*** | 0.473523 |  | 0.684117 |  | 0.86926 |

1. Intel Architecture provides better memory access on a 32-bit boundary and compilers favor such a configuration. [↑](#footnote-ref-2)