Evolutionary Computation

Authors: Gadfrey Balacy

Authors: (Ken) Huy Do

CSC-180

Fall 2017

Project 4 - Evolutionary Computation

Introduction Page 2

Part 1A : Genetic Algorithms Page 2

Part 2B: Genetic Algorithms Page 3 - 6

Part 2C - Part 1: Genetic Programming Page 7-8

Part 2C - Part 2: Genetic Programming Page 8-9

Part 2D - Genetic Programming Page 9

**Introduction**

For this project we are learning about Evolutionary Computation. Evolutionary computation is algorithms that are inspired by biology. It use biological concept such as selection, crossover, mutation and fitness. Selection is the probability that any given “individual” in a population gets selected. The crossover is the parts in which genetic coding is exchange between the parents to create new “children”. Mutation is the probability that a change in the “dna” happens to a individual. Fitness is how well is the individual “genes” is fit for a certain setting.

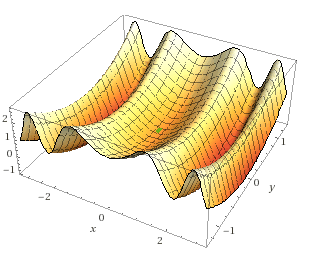
In the 1st part, the genetic algorithm project we are tasked to find the global minimum (or maximum). For a given function. That function must have at least have one or more local minimum.

In the 2nd part of the project we are tasked to find the genetic programming. We find the genetic programming using tools clojure, lein and fungp. We use this tools to generate a function based on inputs that we provided the program. This part of the project comes in two parts. Part 1 of the project it uses our function from the first part. To derived inputs for the programming language. We do that to see if the program generates a similar function based on the inputs. The second part of the project sees if we can generate another function but instead of the inputs being provided by another function. We are provided we a set of training data for the program.

**PART 1: Genetic Algorithms**

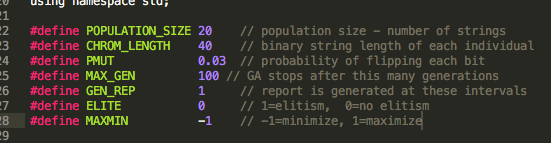
(A) Invent a function f(x,y) of your own design that has more than one  
 local minimum. Graph the function over some range of x and y values  
 that clearly show the local minima (or local maxima, your choice)



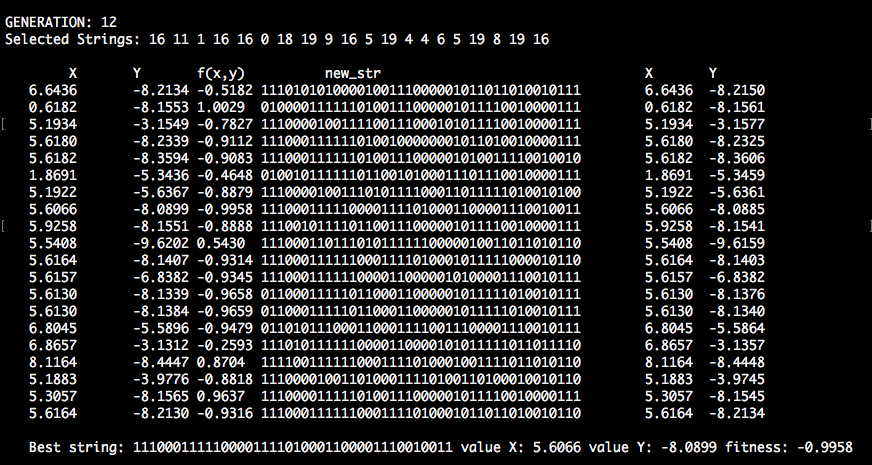


This function has two local mimas: 0, -1

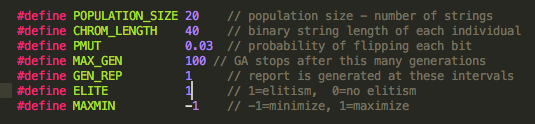
(B) Experiment with various parameters, and report on whether the  
genetic algorithm is able to find the solution, and if so, how quickly.



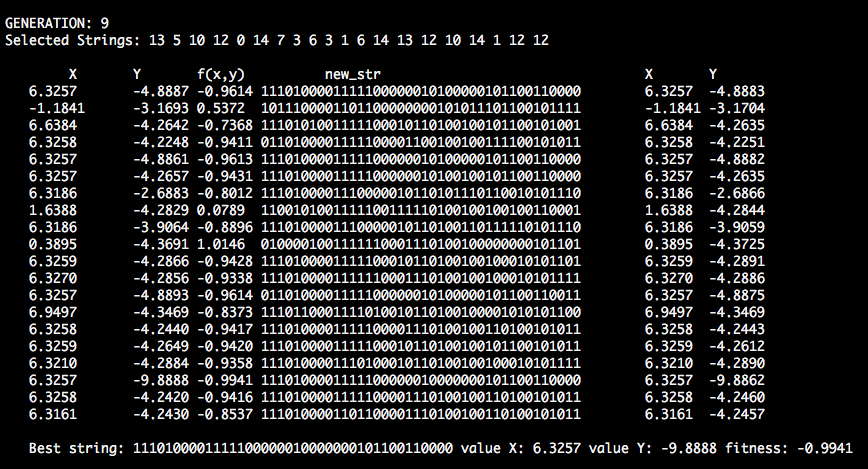
This is the default settings used. Population size of 20 strings. Length of 40. Probability of 3% mutation. Elitism turned off.



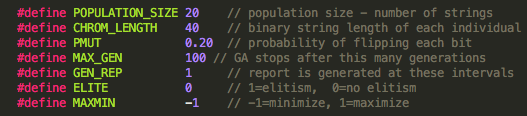
The criteria for fitness success is approximately 2 decimals places to our minima of -1. With the default setting, the SGA found the result around the 12th generation.



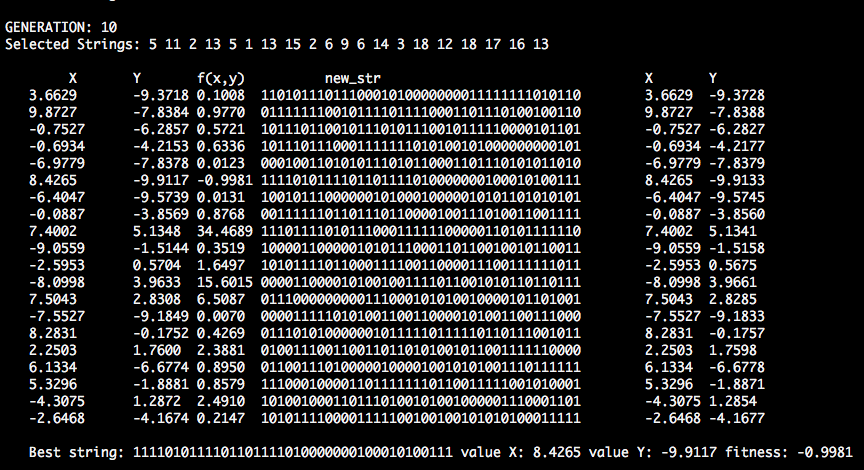
Elistism was turn on. We wanted the SGA to copy the best string.



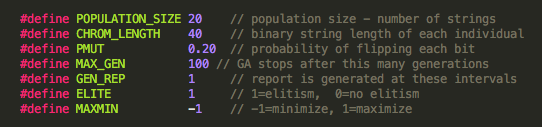
The fitness was found by 9th Generation. Elitism gave desired result quicker.



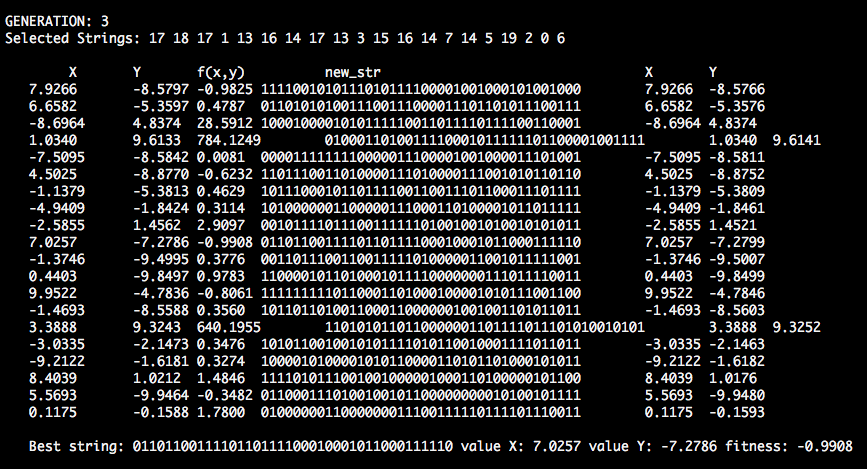
Mutation was changed from 3% to 20%..



With a high mutation, the fitness results was all over the place for each run. In this case fitness was found by 10th generation, but later generation didn’t stabilized well and jumped all over.



Here elitism was turned on and mutation probability changed from 3% to 20%.



With mutation set much higher & elitism turned on, the fitness result was found faster approximately by the 3rd generation.

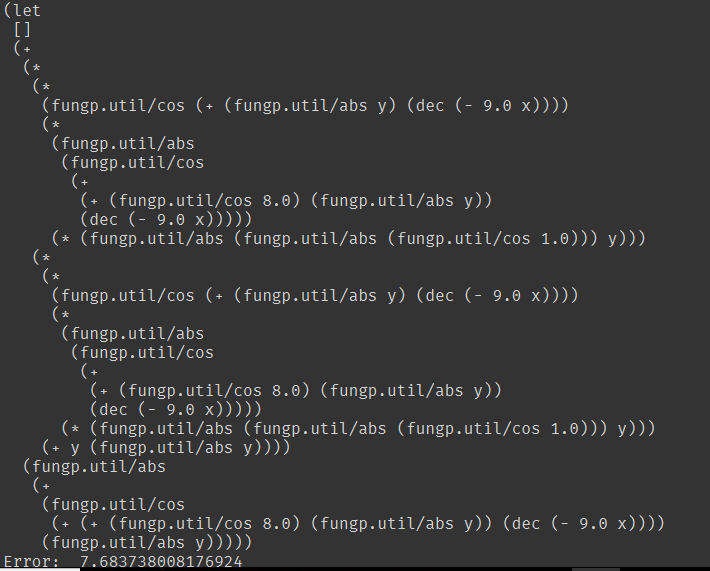
**PART 2: Genetic Programming**

**(C) See if you can get fungp to learn to mimic the following functions:**

1. **your f(x,y) function, above, and**
2. **the "beam" problem provided by your instructor**

The first run of the program had default values in the sample functions and test-regression function. We wanted a case to compare our later test of the program. The first run of the program did not provided a solution that was meaningful. It had high error rate and complicated functions. So for next test run of the program I decided to change the number of iterations, migrations, num-islands and mutation probability to see if that would provide a better solution. The program did provided a better solution but still it was not satisfactory and the solution function was still to complicated.

So we decided to add a sin function to the available sample functions that the program has access to. The reason we did that is because we knew our function that provided that inputs had a sin function. We made a assumption that by providing similar functions to the program it will provide better solutions. Fungp util library of sin, cos was added while exponent (exp) was defined. That solution is the one below.



This solution has lower error rate than my previous runs of the program. But the overall solutions is still too complicated.

So based on the assumption that providing the program similar functions to the one that provided inputs it would provided a better solution. We decided to create our own exponent function and add it to the sample-functions. This is the result



As you can see the function is very similar to the original function. It provides a better solution than the previous runs.

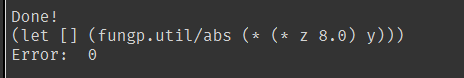
**Function**:

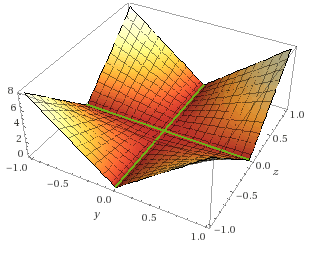
We did more runs of the program by changing the parameters and adding different sample functions but the program did not provide a better solution or the solution it provide was too complicated to use.

2. “Beam”

So similar to what we did in the previous problem we use inputs to have our program create. But instead using a known function to provide the inputs . We are provided a set of training data by our professor. We use that training data to train our program.

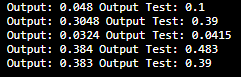
So I run the program with the previous settings from the previous program to find the solution for this problem. The program did find the solution of this program quickly but the solution function was too complicated. So I decided to remove some of the functions inside the sample functions. And run the program again. This is what the program provided.

****

****

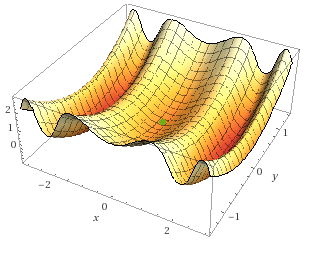
We did run the program multiple times after this solutions was provided. But the program would sometimes output solutions that where complicated or very similar to this function.

So we check to see how will the program generalize by using the test case provided by the professor. This is the output



The left side is the output generated by our function and the right side is the known output of the test case. As you can see it did not generalize will. It only has outputs that you would consider it being close to known output. The reason why we believe it did not generalize will is that our function only uses two of four variables.

(D) Using the same graphing calculator as you used in part 1, over the same range  
 of x and y values, graph the function found in part [C] above, produced by fungp.  
 How closely does your result match the original f(x,y) function?



Local minima = 0, sin(-1)

The graph of the genetically programmed closely match our original . It’s minima of 0 is same and it’s minima Sin(-1) is approximately -1.