Farhad Mohammad Kazemi

I followed the algorithm configuration and parameter setting discussed in class as follows.

population size=1000

number of generations=900

Tournament size=2

Input range= [-4.0 4.0]

Number of examples=400

Number of Registers=6

Maximum program length=200

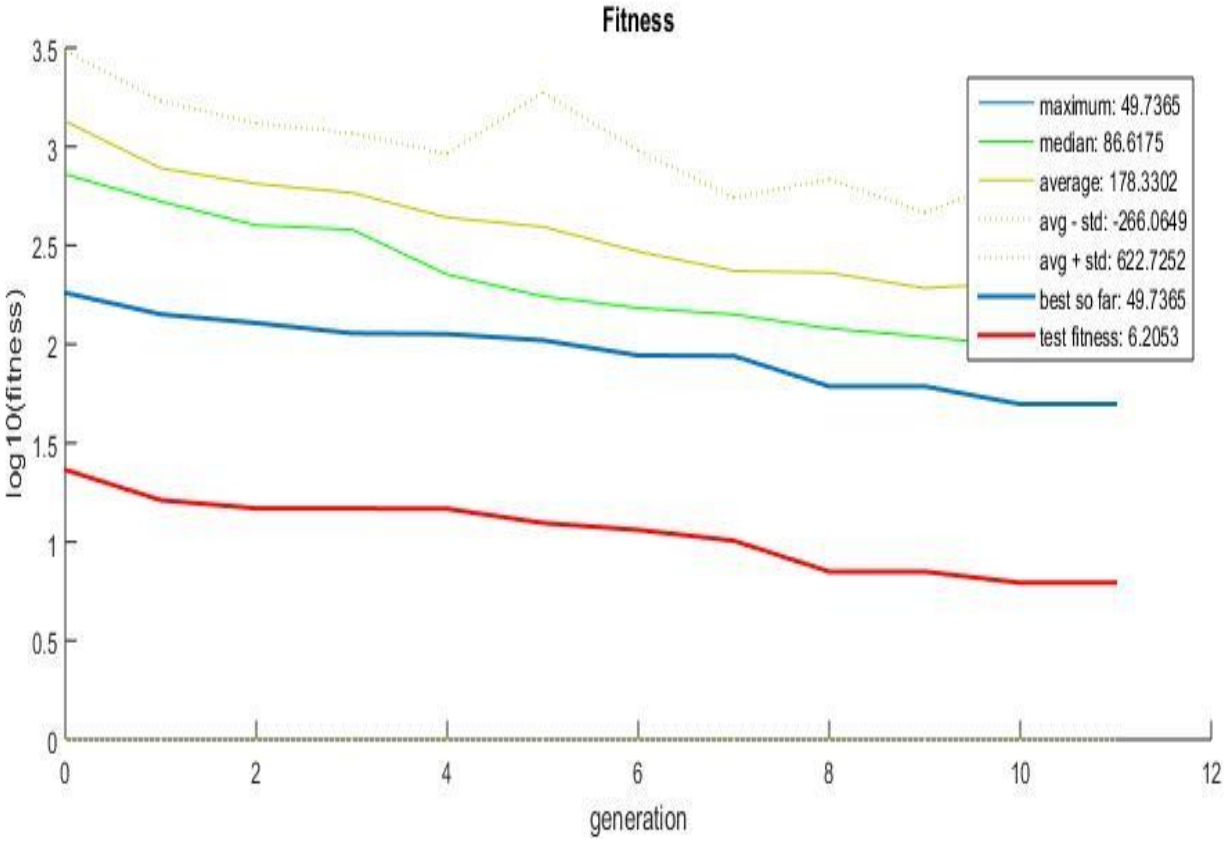
Initial program length=10

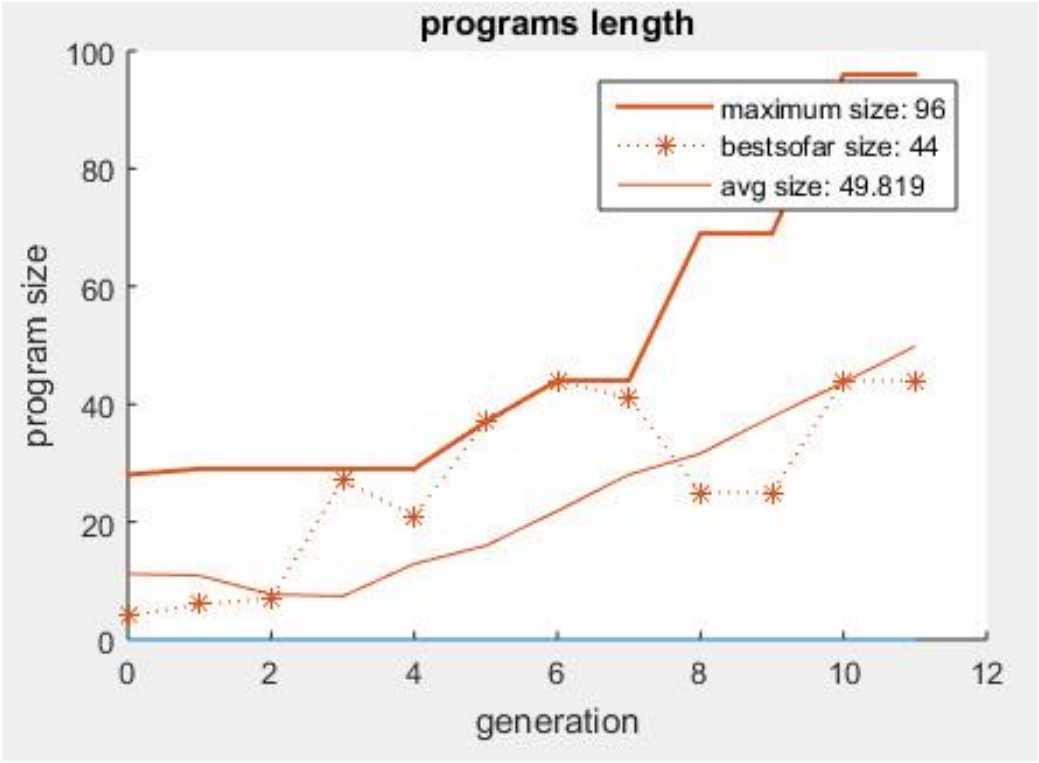
Macro variation rate=75%

Micro mutation rate= 25%

Reproduction rate=100%

a) In this section, for better viewing, I supposed population size=1000 and number of generations=11.





b)

Results with population size=1000 and number of generations=900

The fitness of the 10 best-of-run models and their mean.

Number of run	Fitness of Best Program	Average Fitness	Average Size
1	0.844471	1.51009e+025	88.195
2	0.821541	6.77389e+023	92.68
3	0.918075	2.94936e+023	90.444
4	0.917856	1.10371e+025	88.322
5	0.970921	2.93141e+082	98.271
6	0.829891	1.28053e+055	91.879
7	0.949995	8.11396e+026	100.419
8	0.845594	5.67515e+025	89.81
9	0.839878	1.2055e+025	87.506
10	0.845594	5.67515e+025	89.81

c)

Results with population size=1000 and number of generations=900

introns are indicated with //

For details of best programs for each generation, you can refer to log.txt file that I attached.

Also, for example, I attached produced programs in generation=900. (population900.txt).

################################ **900** ################################

Average Fitness: 1.28053e+055

Average Size: 91.879

Current Best Program:

```
// Fitness Information: 0.829891
template <class T>
void LGP::Program::Execute(std::vector<T>& r, const std::vector<T>& cf) {
	//r[2] = -0.279724 * r[5];
	//r[4] = r[2] - 0.929871;
	//r[5] = 0.626831 * 0.569702;
	//r[0] = cf[0] + r[0];
	//r[2] = r[5] + cf[0];
	//r[4] = cf[0] - 0.404724;
	//if(r[3] < cf[0])
	//if(r[2] < cf[0])
	//r[4] = cf[0] + -0.13739;
	//r[4] = cf[0] - r[1];
	//if(r[2] < r[4])
	//r[2] = cf[0] * r[5];
	//r[5] = cf[0] * cf[0];
	//r[4] = r[2] - 0.820618;
	//if(cf[0] < 0.569702)
	//r[0] = cf[0] - cf[0];
	//if(r[2] < r[3])
	//r[4] = cf[0] + r[0];
	//r[0] = cf[0] - -0.324463;
	//r[4] = 0.72168 + -0.13739;
	r[4] = cf[0] - r[1];
```

//r[0] = r[4] * -0.781494;

//if(0.808044 < r[1])

//r[1] = cf[0] / cf[0];

//r[5] = r[4] - cf[0];

//if(r[1] < 0.0737305)

//r[1] = 0.0165405 - 0.929871;

//r[0] = -0.55426 - 0.587219;

//r[2] = cf[0] + r[5];

//if(-0.947083 < r[0])

//if(cf[0] < -0.363953)

//r[2] = r[5] + cf[0];

//r[5] = -0.68512 - -0.930176;

//r[3] = r[2] + cf[0];

//r[2] = 0.99231 - r[2];

r[3] = -0.151367 / -0.640259;

//r[2] = cf[0] * r[4];

//r[2] = r[0] * 0.358704;

//if(r[3] < cf[0])

//r[2] = cf[0] * -0.812317;

r[5] = cf[0] * 0.569702;

r[4] = r[4] * r[5];

//r[2] = -0.680969 * cf[0];

//r[2] = r[2] - r[3];

//r[0] = -0.52533 * -0.0922852;

//r[1] = r[3] - 0.59613;

```
r[0] = 0.203735 * r[3];
r[2] = cf[0] * r[5];
r[3] = -0.151367 / -0.640259;
r[1] = r[2] - 0.934692;
if(r[1] < 0.0737305)
r[1] = r[1] / 0.270203;
if(0.585144 < r[1])
r[5] = cf[0] / r[1];
//r[2] = r[5] + cf[0];
//r[2] = r[0] - -0.803223;
r[2] = cf[0] * r[5];
if(r[2] < r[4])
r[1] = r[2] - 0.978455;
//if(r[1] < 0.0737305)
//r[4] = 0.99231 + r[3];
if(0.656006 < r[5])
if(0.808044 < r[1])
r[2] = r[5] + cf[0];
//if(cf[0] < r[1])
//r[4] = cf[0] + -0.13739;
//if(r[2] < cf[0])
//r[4] = cf[0] + r[0];
if(cf[0] < r[0])
if(r[5] < r[1])
r[5] = cf[0] * 0.550171;
```

```
        r[4] = r[2] - 0.820618;
        //r[2] = cf[0] / cf[0];
        r[2] = cf[0] * r[5];
        if(r[2] < r[4])
        r[1] = r[2] - 0.726074;
        if(r[1] < 0.0737305)
        r[0] = 0.99231 - r[2];
        if(r[3] < r[1])
        r[0] = -0.255493 * r[1];
        r[2] = r[5] + cf[0];
        //r[4] = cf[0] - 0.159119;
        r[5] = cf[0] - r[5];
        if(-0.38562 < -0.0687866)
        r[2] = r[5] + cf[0];
        //r[4] = cf[0] + -0.13739;
        //if(r[2] < cf[0])
        //r[4] = cf[0] * -0.13739;
        r[4] = r[5] - 0.159119;
        r[5] = cf[0] - r[3];
}
```

Source Code (ConsoleApplication4) (for details of another files Please refer to source code folder that I attached to email)

```
//        Farhad   Mohammad Kazemi

#include <ctime>
#include <cmath>
#include <fstream>
#include <iostream>

#include "Config.h"
#include "GenerationalTournamentPopulation.h"

#include "InstructionArgumentConstant.h"
#include "InstructionArgumentFeature.h"
#include "InstructionArgumentRegister.h"

#include "InstructionOperationPlus.h"
#include "InstructionOperationMinus.h"
#include "InstructionOperationMult.h"
#include "InstructionOperationDiv.h"
#include "InstructionOperationIflt.h"

#include "SymRegFitnessCase.h"

#include "SymRegProgram.h"

using namespace LGP;

double
GenerationalTournamentPopulation<SymRegProgram<double>,
double>::proportionElitism
= 0.1;
```

```
unsigned int
GenerationalTournamentPopulation<SymRegProgram<double>,
double>::tournamentSize
= 2;

int main(int argc, char** argv) {
     // Building the relevant config object - specify it all,
even though not all of it is needed:
     Config<double>* c;
     if (argc == 1) {
          c = new Config<double>();
     }
     else {
          c = new Config<double>(argc, argv, true);
     }

     c->numRegisters = 6;
     c->numFeatures = 1;
     c->epsilon = 0.1;

     c->argumentGenerators-
>AddElement(InstructionArgumentConstant<double>::Generate);
     c->argumentGenerators-
>AddElement(InstructionArgumentFeature<double>::Generate);
     c->argumentGenerators-
>AddElement(InstructionArgumentRegister<double>::Generate);

     c->instructionOperations-
>AddElement(InstructionOperationPlus<double>::Generate);
     c->instructionOperations-
>AddElement(InstructionOperationMinus<double>::Generate);
     c->instructionOperations-
>AddElement(InstructionOperationMult<double>::Generate);
     c->instructionOperations-
>AddElement(InstructionOperationDiv<double>::Generate);
     c->instructionOperations-
>AddElement(InstructionOperationIflt<double>::Generate);

     c->initialMinLength = 6;
```

```
    c->initialMaxLength = 10;
    c->maxLength = 200;
    c->populationSize = 1000;

    c->maxGenerations = 900;

    c->popLogInterval = 1;
    c->popLogFilePath = "population";
    c->statsLogFilePath = "log";

    c->runLogFilePath = "results.csv";

    // Initialise the RNG:
    if (c->seedSpecified) {
        Rand::Init(c->randSeed);
    }
    else {
        Rand::Init();
    }

    // Building the population object, randomising the
fitness of it all:
    IPopulation<SymRegProgram<double>, double>* pop =
        new
GenerationalTournamentPopulation<SymRegProgram<double>,
double>(c);

    // Build the fitness environment we will train:
    FitnessEnvironment<double> train(c);
    for (double i = -4; i <= 4; i += 0.02) {
        train.AddCase(new SymRegFitnessCase<double>(i, (1 -
((i*i) / 4) - ((i*i) / 4))*exp(-((i*i) / 8) - ((i*i) /
8))));
    }

    FitnessEnvironment<double> test(c);
    for (double i = -8; i <= 8; i += 0.04) {
```

```
                    test.AddCase(new
SymRegFitnessCase<double>(i, (1 - ((i*i) / 4) - ((i*i) /
4))*exp(-((i*i) / 8) - ((i*i) / 8))));
     }

     std::cout << "Loaded " << train.NumberOfCases() << "
training cases and "
          << test.NumberOfCases() << " test cases. Beginning
evolution." << std::endl;


     // Carry out the evolution:
     //clock();
     unsigned int generationsUsedOrNoSolution = pop-
>Evolve(train);
     int time = clock();

     if (generationsUsedOrNoSolution <= c->maxGenerations) {
          std::cout << "Solution found.\nSolution is:" <<
std::endl;
     }
     else {
          std::cout << "No solution found.\nBest program:" <<
std::endl;
     }

     IProgram<double>* best = pop->GetFittestProgram();
     std::cout << pop->GetFittestProgram()->ToString(true,
true) << std::endl;

     // Now calculate the statistics that aren't already
calculated:
     std::cout << "Writing run statistics to: " << c-
>runLogFilePath << std::endl;


     // Test fitness still needs to be calculated for the
best program.
```

```
	unsigned int generations =
std::min(generationsUsedOrNoSolution, c->maxGenerations);

	double bestTrainingFitness = best->Fitness();

	best->UpdateFitness(test);
	double bestTestFitness = best->Fitness();

	std::ofstream fout(c->runLogFilePath.c_str(),
std::ios::out | std::ios::app);
	fout << generations << "," << time << "," <<
bestTrainingFitness << "," << bestTestFitness
		<< std::endl;
	std::cout << generations << "," << time << "," <<
bestTrainingFitness << ","
		<< bestTestFitness << std::endl;
	std::cout << "\n\t(NB: Clock ticks per second: " <<
CLK_TCK << ")" << std::endl;
	fout.close();

	delete pop;
	delete c;
	return 0;
}
```