Genetic Folding MATLAB Toolbox: Solving Santa Fe Trail Problem

Mohammad A. Mezher, Maysam F. Abbod

Abstract— This paper is a primarily attempt to design a toolbox for Genetic Folding algorithm using MATLAB. The toolbox was designed for training ACO in solving Santa Fe Trail problem. However, GF algorithm can encode and decode any type of problem into a linearly folding scheme. For advance or even simple type of problems, a string scheme is encoding to represent a set of operators. GF is a novel algorithm in solving optimization problems as shown in the experiments. We will also illustrate the benefits of GF algorithm conducted into ACO in order finding the best feeding function in comparison with the literature.

Keywords-Genetic Folding; Genetic Algorithm; Genetic Programming; Ant Colony Optimization; Evolutionary Algorithms; Santa Fe Trail Problem

I. Introduction

Genetic folding algorithm (GF) is an evolutionary computation algorithm to find computer programs that can perform a task [1]. Programs (chromosomes) generated by GF are usually represented by floating numbers separated by dots. Each cell in a chromosome holds floating numbers represent a function call and a cell holds less than one is a value to be plugged in functions.

GF has been shown to be an effective strategy in problems of classifications, regressions and recently in ant colony optimization problems. For example, classification [2] and regression [3] have demonstrated how genetic optimization methods can be used to derive superior results. The [4] have presented GF using Ant Colony Optimization to solve Santa Fe Trail problem, whilst others have demonstrated how a combination of GF and GP implemented in analogue circuit design [7].

Unlike older AI systems, GF is better than conventional AI in that it is more genetic generation diversity [4] and simple to implement as shown here. Furthermore, GF has shown significant benefits over typical optimization techniques in searching a large, multi-class, or n-dimensional dataset.

Mohammad A. Mezher is with the College of Computing, Fahad Bin Sultan University, Tabuk, KSA (e-mail: mmezher@fbsu.sa.edu)

Maysam F. Abbod is with the Electronic and Computer Engineering, Brunel University London, Uxbridge, UK (e-mail: maysam.abbod@brunel.ac.uk)

The structure of this paper, section II introduces the Santa Fe Trail problem. In Section III we discuss the overview of GF algorithm. We then explain and provide examples of MATLAB code of our new GF schema. We then summarize experiments results of the previous literature in Section IV. Section V states the conclusion and future directions.

II. Ant Colony Optimisation

Ant Colony Optimization (ACO) is a technique for optimization that was initially introduced in the early of 1990's and it is one of the most successful techniques of the field of swam intelligence. The general idea is relatively simple and based on a set of ants, each making one of possible round-paths looking for grain [8]. Figure 1 shows the ACO algorithm pseudo code.

Artificial ant problem however, has been solved using GA [11], GP [10] to different problems such as travelling salesmen problem [9]. The general pseudo code of the ACO algorithm has been shown in Figure 1.

```
ACO Algorithm:
Initialize the pheromone trails and parameters;
   while(not_termination)
      for each ant complete a tour
         LocalPhermoneUpdate();
      End for
      Update_Best();
      GlobalPheromoneUpdate();
     end while
end procedure
```

Figure 1. An ACO Algorithm Pseudo-Code

III. Genetic Folding Algorithm

Genetic Folding algorithm has been inspired by RNA/DNA folding mechanism as in [2]. GF algorithm is a proofed to be a comparable algorithm in predicting mathematical equations such as kernel function [4] for either classification [5,6] and regression [3] problems. GF simulates the survival of the fittest individuals among all the individuals in the population over successive generations for solving a problem. Each generation consists of a population of linear floating numbers that represents a program. GF algorithm has a flexible representation of GF program; Prefix, Suffix and Infix formats [2].

A simple MATLAB GF algorithm code is shown in Figure 2. GF firstly operates on an initial population of potential solutions producing the successively fittest solution. At each generation, the solutions (chromosomes) generated randomly using a set of predefined functions and terminals [12]. All chromosomes are created along with their level of fitness. Every chromosome then undergoes to one of the reproducing operators. In this paper, refolding operator will be implemented for the production process.

```
MAXGEN = 50; % No. of generations
Oplist =
{'antprogn3','antprogn2','antif','antmove','a
ntright','antleft'};% set of operators
Oparity ={3,2,2,0,0,0};% arity of each
operator
% Initialise population
Chrom = GFEncoding(Oplist, Oparity);
Chrom= GFOperator(‘GFDecoding’, Chrom);
%Create GF trees to evaluate it by GPLAB
GFPOP = fun(antEval(Chrom)); % Evaluate GF
Chromosomes
Gen = 0; % Counter
% Start GF generational loop
while Gen < MAXGEN
% Visualization using GPLab toolbox
plotgraphics
%Encode a GF chromosome using the defined set
Chrom = GFEncoding(Oplist, Oparity);
% Add chromosome into population
[GFPOP]=Combine (Chrom, GFSelected);
% use a sus selection method
GFSelected = selectMethod('sus',GFPOP);
% Refold individuals (GF Genetic Operator)
GFSelected = GFOperator('Refolding',
GFSelected);
% Create A GF Tree to be evaluated in GPLab
GFSelected = GFOperator(‘GFDecoding’,
GFSelected);
% Evaluate offspring using antEval function
GFSelected = fun(antEval(GFSelected));
% Add offspring into population
[GFPOP]=Combine (Chrom, GFSelected);
% Increment counter
Gen = Gen+1;
end
```

Figure 2. MATLAB GF Algorithm Code

The chromosomes in the population are then made to go through a process of evolution. GF is based on the genetic structure and the refolding operator. Genes from a good individual propagate throughout refolding each chromosome so that a good chromosome will sometimes produce offspring that is better than the old chromosome. Thus, each successive generation will carry the fittest refolded chromosomes for all successive population. The basic approach for implementing GF is by using one genetic operator; refolding operator as discussed in GF literature [12].

IV. GF Toolbox Structure

To our knowledge, this is the first study to deal with GF algorithm using MATLAB code. The MATLAB Genetic Folding Toolbox aims to make GF accessible to the scientist and engineer for further improvements. This allows the user also to make direct comparisons between GF methods and other evolutionary algorithms specifically Genetic Programming.

GF MATLAB Toolbox essentially supports structures data type and complex of numeric elements. The main data type in GF is a complex structure of integer values separated by dots and stored as a string type. The GF Toolbox produced phenotypes and genotype after finding the fitness function values. The chromosome structure stores an entire population in a single structure, where GF structure consists of the left and right numbers of individuals along with GF index which represents the type of the chromosome operator. Phenotypes are stored in a structured of Helix-Language [12] scheme. Finally, the fitness values are stored in a cell inside the GF chromosome structure. The GF toolbox uses GPLab [13] for the evaluation, graphics and selection methods in order not reinventing the wheel.

A. *Chromosome Architecture*

Each GF chromosome composed of a number of predefined genes. Each gene consists of a linear number of units that are allocated/reallocated to represent various number of computer programs. Another significant aspect of GF chromosome is being able to work on two types of spaces alternatively, a computation space (genotype) and a solution space (phenotype).

Typically, the floating-numbers inside GF chromosomes are used to represent solutions, where genes hold a float number are functions (antif, antprogn2 and antprogn3) and genes hold values less than zero are terminal (antright, antleft and antmove). Assume the following is GF chromosome to represent a computer program of

antprogn3(antprogn2(antmove,antleft),antright,antmove):

1	**2**	**3**	**4**	**5**	**6**
Antprogn3	Antprogn2	Antmove	antleft	Antright	Antmove
2.6.5	4.3	3	4	5	6

B. *Fitness Values*

Fitness values computed by an objective function shown in (1) specific to the Santa Fe Trail problem. This case study confirms the importance of GF algorithm in solving Santa Fe Trail problem in a comparison with Genetic Programming [8]. The Santa Fe Trail problem however, has 32×32 fields with 89 pieces of foods that are located on the discrete trail. As the generations pass, the members of the population should get fitter and fitter. The raw fitness value is calculated according to the following function:

$$\text{Fitness\_value} = 89 - \text{pieces\_eaten} \qquad (1)$$

where pieces_eaten is the number of eaten foods by an ant from the overall pieces of 89 and the best Fitness_value is to reach a cost equals to zero.

C. *Encoding/Genotype process*

In GF algorithm, an encoding function is use to represent mapping of the object variables to a string code based on the parity number of each gene. The encodings in Strings of the GF chromosomes are chosen according to the following facts:

1. Allows genetic operators to propagate building blocks from parents' genotype to offspring genotype.
2. Allows a tractable mapping to the phenotype by storing the number of folded RC, LC and IC in each gene of a chromosome.
3. Represents the fundamental building blocks that are important for the problem type in hand.

```
function[fresult]= GFEncoding(GFChr,arity)

sGF = size(GFChr,2); % size of GF
j=1; % start at index 1
GFTemp = GFChr; % A temp copy of GFChr
% Creating the LC and RC of each gene
while sGF > 1
  for a=1:arity % no of arity of each gene
    randIndex= intrand(1,sGF);
    GFendoded{j,a}=GFChr(randIndex);
    GFChr(randIndex) = [];%Remove the selected
                          %item
  End % End for-loop
  fresult{1,j} = strcat(
       num2str(GFendoded{j,1}),'.',
       num2str(GFendoded{j,2}),'.',
       num2str(GFendoded{j,3})
      ); %A Decoded of a 3-arity GF chromosome
  GFTemp{1}=[];%Remove an item from the GFTemp
  GFTemp=GFTemp(~cellfun('isempty',GFTemp));
  sGF = size(GFTemp,2);
  j = j+1;
end % End while-loop
```

Figure 3. MATLAB Encoding Code of GF Chromosome

D. *Decoding/Phenotype process*

To decode a chromosome, Figure 4 uses number of parity to initialize GF tree structures from the encoded GF chromosome which will be as a result implemented to the GPLab [13] toolbox. The mapping of string code to its tree structure is achieved through decoding function as shown in Figure 4. This MATLAB code converts all functions and terminals symbols to the proper GF tree structure. A GF tree Structure is computed due to two facts; one is to be implemented into the GPLab toolbox and secondly; due to the comparative purposes.

```
function[tree, thisDepth, GFLevel,
noGFnodes] = GFDecoding(GFChr, nextNode,
thisDepth, noGFnodes, GFLevel)
tree.op = GFparent{newNode};% the operator
thisDepth = thisDepth + 1; % Tree Depth
noGFnodes= noGFnodes +1; % no of nodes
tree.nodeid=noGFnodes; %the ID value of a
node
tree.maxid=tree.nodeid;%nodeid value=the
maxid
GFLevel = GFLevel + 1; % no of folded times

for i=1:arity % no of arity
   nextNode=depthGF(newNode,i);%next node
func
   [t, thisDepth, GFLevel, noGFnodes] =
GFDecoding(GFChr, nextNode, thisDepth,
noGFnodes, GFLevel);
   tree.kids{i}=t;
end% End for loop
tree.nodes=(tree.maxid+1)-tree.nodeid;
%update % the number of tree's nodes
tree.maxid = tree.kids{i}.maxid;
```

Figure 4. A MATLAB Encoding Code of GF Chromosome

E. *Refolding Operators*

During refolding operator [12] calling to reallocate a chromosome's cells, a lot of various kind of computer programs produced. In general, GF tries every time to fold each gene with its complementary gene within same chromosome. This operator leads to the evolution of populations of H-Language that are better suited to their environment than the H-Language from which they were created, just as in natural RNA folding. In Figure 5 a MATLAB code is shown to demonstrate simplicity of refolding operator, which are randomly generated using a *randperm* function. The function will always generate computer programs free of pathologies. i.e. all functioned generated by the operator have the right number of parities of both terminal and nonterminal functions. In this case no attention was paid on the function except saving the final result of code generated by the GFEncoding(). Based on its results, new fitness values were computed for each refolded chromosome.

```
function[GF,fresult]= Refolding(GFChr)

sGF = size(GFChr,2);
GFChr=randperm(sGF);%Random indices
arity = 3; %maximum number of arities
%Reallocating LC and RC of each gene
[GF,fresult]= GFEncoding(GFChr,arity);
```

Figure 5. Refolding Code of GF Chromosome

V. Implementation and Analysis

Comparisons with GP in various types of problems confirm that GF algorithm is a promising area of research for optimization programming. On the artificial ant problem (Table I), the GF algorithm using a refolding operator was an effective way to improve the looking for extreme of functions in a search space. The fittest chromosomes using GF algorithm were recovered at early generations in a comparison with GP as shown in the last column of Table I. In fact, GF found optimum chromosomes with a longest path of calling (depth) and number of folding processes (nodes). To aloow for more results in a comparison with GP, we integrated GF MATLAB code into GPLab [13] toolbox to obtain on results shown in Appendix A

TABLE I. List of all applications tested on GF algorithm

EA Type	Population	Generation	Fitness	Depth	Nodes	Fittest
GF	20	10	33/89	8	23	5
GP	20	10	24/89	3	6	2
GF	50	20	40/89	7	25	5
GP	50	20	33/89	6	12	18
GF	200	100	58/89	5	16	61
GP	200	100	77/89	6	18	96

The overall papers were conducted using GF algorithm are summarized in Table II. The results thus obtained were comparable with other evolutionary algorithms. Refolding operator rotates genes for a chromosome to create a new offspring. The simplest way to do this was to permutate randomly all genes' compartments before the parent chromosome transferred to the second child.

TABLE II. List of all applications tested on GF algorithm

Range of Applications	**Evolutionary Algorithms**				
	GEP	***DAG***	***GF***	***KGP***	***KGEP***
Multi-Classification Dataset	**[6]**	**[6]**	**[4,6]**	**[6]**	**[6]**
	GA	***GP***	***GF***	**-**	**-**
Binary Classification Dataset	**[2,5]**	**[2,5]**	**[2,5]**	**-**	**-**
Regression Dataset		**[3]**	**[3]**	**-**	**-**
Santa Fe Trail Problem			**[12]**	**-**	**-**
Analogue Circuit Design	**-**	**[7]**	**[7]**	**-**	**-**

VI. Conclusion and Future Work

The purpose of the paper is to introduce a MATLAB toolbox of Genetic Folding Algorithm. As GF being a member of evolutionary algorithms, GF is applicable for general optimization techniques or any specific NP-problems. GF algorithm is able to handle any problem based on the number of folding the problem might be embodied with. This means, the number of folding is a repeatable process based upon the type of operators and the number of parities the computer programs takes to be represented. The GF MATLAB toolbox shown here is meant to assist researchers making new comparisons with GF algorithms with any other members of evolutionary algorithms' family. Finally, some of GF comparisons were made with wide range of algorithms and problems using the introduced MATLAB simple code.

References

[1] J. Koza, Genetic Programming: on the programming of computers by means of natural selection, MIT press, pp. 66–551, 1992.

[2] M. Mezher, M Abbod, Genetic Folding: A New Class of Evolutionary Algorithm. AI-2010 Thirtieth SGAI International Conference on Artificial Intelligence Cambridge, England 14-16 2010

[3] M. Mezher, M Abbod, Genetic Folding: A New Genetic Folding Algorithm for Regression Problems. UKSim.2012, Cambridge, UK. 2012

[4] M. Mezher, M Abbod, Genetic Folding: Analyzing the Mercer's Kernels Effect in Support Vector Machine Using Genetic Folding

[5] M. Mezher, M Abbod, Genetic Folding: A New Class of Evolutionary Algorithms for Binary SVM Problem.

[6] M. Mezher, M Abbod , "Genetic Folding for Solving Multiclass SVM Problems" Elsevier Editorial System for Applied Soft Computing

[7] O. Ushie, M. Abbod, B. Usibe, Genetic Folding/Programming Toolbox: Analogue Circuit Design Case Study. Journal of Automation and Systems Engineering, PP. 40-64, 2016

[8] J. Koza, F. Bennet, D. Andre, M. Keane, Genetic Programming III Morgan Kaufnamm Pub. ISBN 1-55860-543-6, 1999

[9] J. Tavares, F. Pereira, Evolving strategies for updating pheromone trails: A case study with the tsp. In: PPSN XI Proceedings. Volume 6239 of Lecture Notes in Computer Science., Springer (2010) 523–532

[10] J. Tavares, F. Pereira, Designing pheromone update strategies with strongly typed genetic programming. In: EuroGP 2011 Proceedings. Lecture Notes in Computer Science, Springer (2011)

[11] T. White, B. Pagurek, F. Oppacher: ASGA: Improving the ant system by integration with genetic algorithms. In: Proceedings of the 3rd Genetic Programming Conference, Morgan Kaufmann (1998) 610–617

[12] M. Mezher, M Abbod, Novel Genetic Operator for Genetic Folding Algorithm: A Refolding Operator and A New Genotype, submitted at International Journal of Intelligence Science. November 2016

[13] S. Silva and J. Almeida, GPLAB-a genetic programming toolbox for MATLAB, in Proceedings of the Nordic MATLAB conference, 2003, pp. 273–278

Appendix A

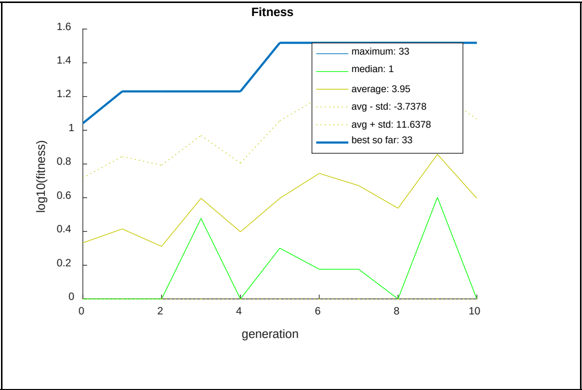
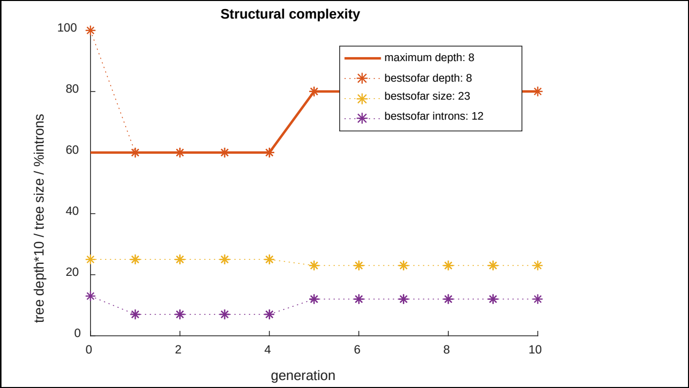
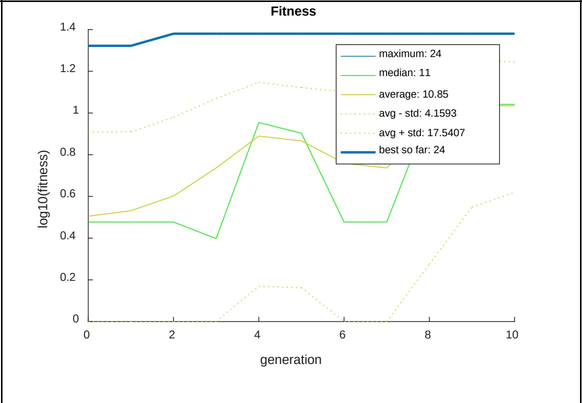
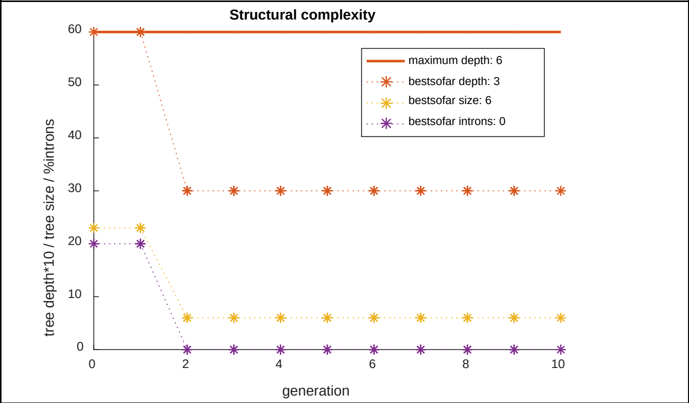
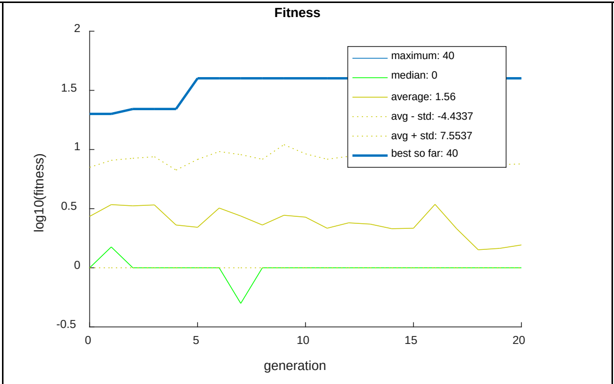
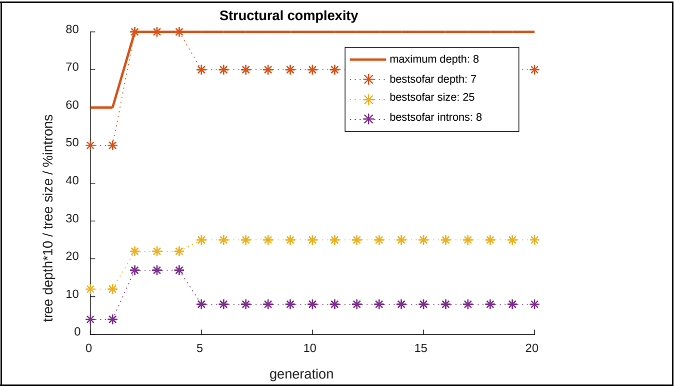
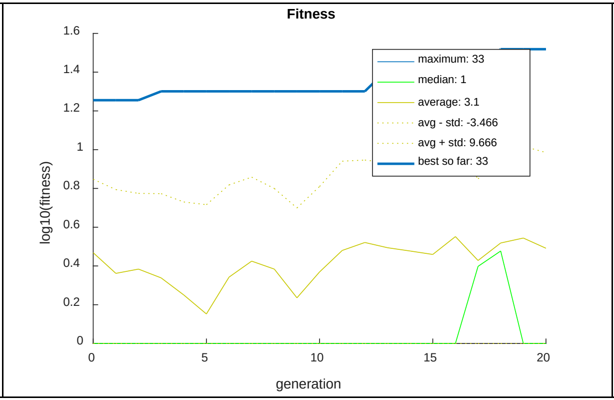
GF		
GP		

Figure 6. GF vs. GP at population size 20 and generations 10

GF		

GP



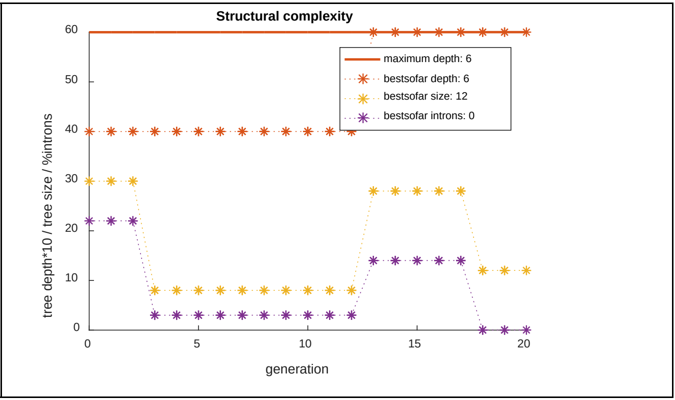
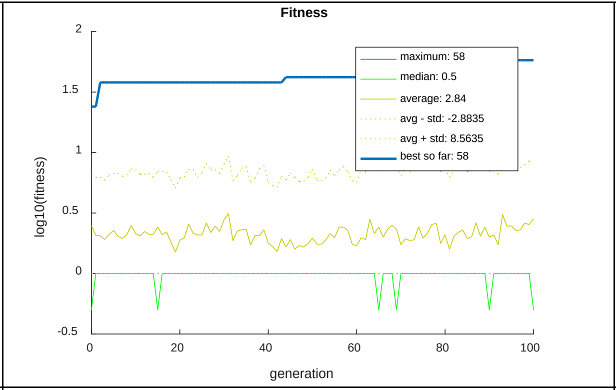
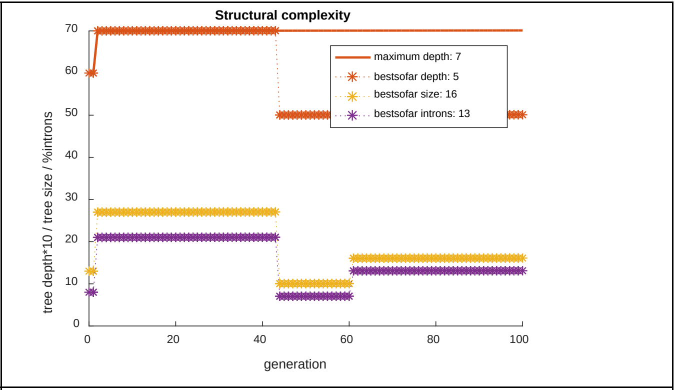


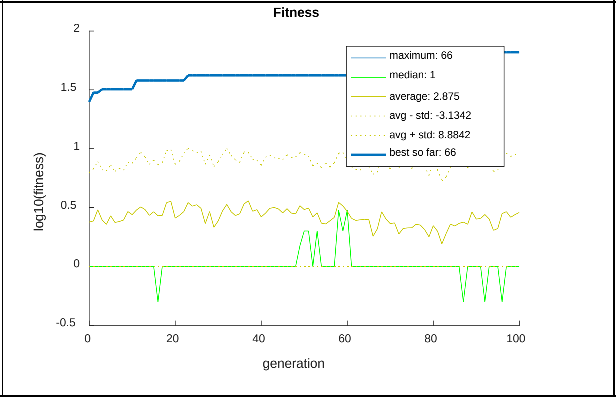
Figure 7. GFGF vs. GP at population size 50 and generations 20

GF





GP



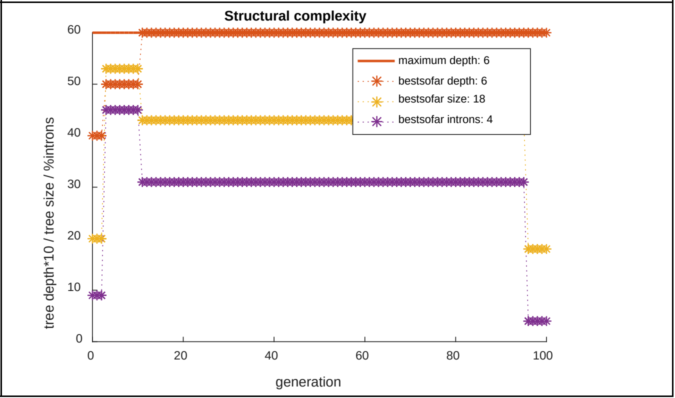


Figure 8. GF vs. GP at population size 200 and generations 100