Design and Implementation of ILP Based Selection Algorithm for Genetic Programming system (ILPSAGP)

Dr .Rabah Nory Farhan
College of Computers, University Of Anbar
Email:rabahalobaidy@yahoo.com

ABSTRACT

Genetic Programming is one of the evolutionary algorithms developed to solve wide area of industrial and scientific problems. Rather than dealing with population of string like Genetic Algorithm, Genetic Programming composes the first population from programs tree derived from the function set of the problem. In this paper, we extend the selection algorithm of the GP by using the Learning Classifier System, which build and derive the Hypothesis set from the population. The selection algorithm redesigned to enforce the selection been added from the Hypothesis domain. The proposed system called ILPSAGP *was built using C#.net 2008 and tested with traditional problem like Line Regression problem. The obtained results shows more accurate result than traditional Genetic Programming.*

Keywords: Genetic Programming, Inductive Logic Programming, Selection Algorithms, Crossover, Mutation, S-Expression.

1. Introduction

The term Genetic Programming1 (GP) [2] has two possible meanings. First, it is often used to subsume all evolutionary algorithms that have tree data structures as genotypes. Second, it can also be defined as the set of all evolutionary algorithms that breed programs, algorithms, and similar constructs. In. The conventional well-known input-processing-output model from computer science states that a running instance of a program uses its input information to compute and return output data. In Genetic Programming, usually some inputs or situations and corresponding output data samples are known or can be produced or simulated. The goal then is to find a program that connects them or that exhibits some kind of desired behavior according to the specified situations. GP uses the natural phenomena in the life in cell level like Crossover, Mutation and natural selection. In this paper we propose an excellent algorithm called ILPSAGP which emphases and extend the GP by Inductive Logic Programming (ILP).

2) Genetic Programming Overview:

2.1) Introduction:

In genetic programming, populations of hundreds or thousands of computer programs are genetically bred. This breeding is done using the Darwinian principle of

survival and reproduction of the fittest along with a genetic recombination (crossover) operation appropriate for mating computer programs. As will be seen, a computer program that solves (or approximately solves) a given problem may emerge from this combination of Darwinian natural selection and genetic operations. Genetic programming starts with an initial population of randomly generated computer programs composed of functions and terminals appropriate to the problem domain. The functions may be standard arithmetic operations, standard programming operations, standard mathematical functions, logical functions, or domain-specific functions.
In summary, the genetic programming paradigm breeds computer programs to solve problems by executing the following three steps[2,5]:

(1) Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
(2) Iteratively perform the following substeps until the termination criterion has been satisfied:

(a) Execute each program in the population and assign it a fitness value according to how well it solves the problem.

(b) Create a new population of computer programs by applying the following two primary operations. The operations are applied to computer program(s) in the population chosen with a probability based on fitness.

(i) Copy existing computer programs to the new population.

(ii) Create new computer programs by genetically recombining randomly chosen parts of two existing programs.

(3) The best computer program that appeared in any generation (i.e., the best-so-far individual) is designated as the result of genetic programming. This result may be a solution (or an approximate solution) to the problem.

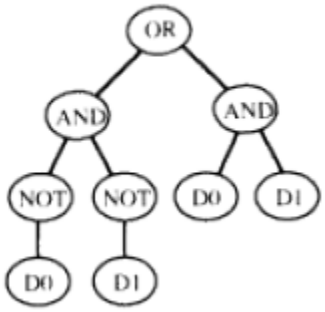
2.2) Initial Structure (Population):

In every adaptive system or learning system, at least one structure is undergoing adaptation. For the conventional genetic algorithm and genetic programming, the structures undergoing adaptation are a population of individual points from the search space, rather than a single point. Genetic methods differ from most other search techniques in that they simultaneously involve a parallel search involving hundreds or thousands of points in the search space[6].

The set of possible structures in genetic programming is the set of all possible compositions of functions that can be composed recursively from the set of Nfunc functions from F = {f1,f2,...,fNfunc} and the set of Nterm terminals from T = {a1,a2,...,aNterm}. Each particular function fi in the function set F takes a specified number z(fi) of arguments (z(f1), z(f2) ..., z(fNfunc)). That is, function fi has arity z(fi).
The functions in the function set may include

- arithmetic operations (+, -, *, etc.),
- mathematical functions (such as sin, cos, exp, and log),
- Boolean operations (such as AND, OR, NOT),
- conditional operators (such as If-Then-Else),
- functions causing iteration (such as Do-Until),
- functions causing recursion, and
- any other domain-specific functions that may be defined.
- As an example, consider the even-2-

- parity function (i.e., the not-exclusive-or function, the equivalence function) with two arguments. This function returns T (True) if an even number of its arguments (i.e., D0 and D1) are T; otherwise, this function returns NIL (False). This Boolean function can be expressed in disjunctive normal form (DNF). See figure (1).



Figure(1). Initial Population

2.3) GP Fitness Function:

Fitness is the driving force of Darwinian natural selection and, likewise, of both conventional genetic algorithms and genetic programming. In nature, the fitness of an individual is the probability that it survives to the age of reproduction and reproduces. This measure may be weighted to consider the number of offspring. There are many fitness functions for GP as most used is the Raw Fitness which is shown as bellow[2,8]:

$$R(i,t)=\sum_{J=1}^{N}|S(i,j)-C(j)| \quad \text{.....(1)}$$

where S(i, j) is the value returned by S-expression i for fitness case j (of N cases) and where C(j) is the correct value for fitness case j.

2.4) Reproduction:

The reproduction operation for genetic programming is the basic engine of Darwinian natural selection and survival of the fittest. The reproduction operation is asexual in that it operates on only one parental S-expression and produces only one offspring S-expression on each occasion when it is performed. The operation of reproduction consists of two steps. First, a single S-expression is selected from the population according to some selection method based on fitness. Second, the selected individual is copied, without alteration, from the current population into the new population (i. e., the new generation).

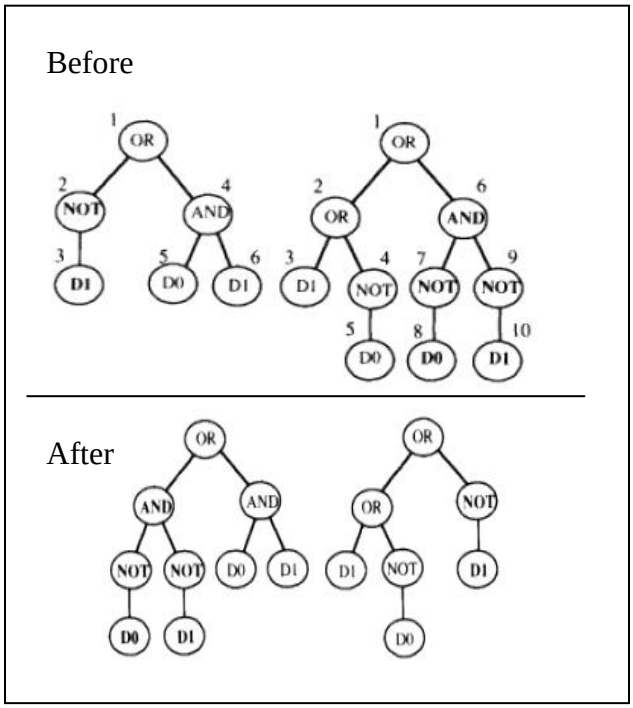
When the reproduction operation is performed by means of the fitness-proportionate selection method, it is called fitness-proportionate reproduction.

Among the alternative selection methods are tournament selection and rank selection[5,6]. In rank selection, selection is based on the rank (not the numerical value) of the fitness values of the individuals in the population. Rank selection reduces the potentially dominating effects of comparatively high-fitness individuals in the population by establishing a predictable, limited amount of selection pressure in favor of such individuals. At the same time, rank selection exaggerates the difference between closely clustered fitness values so that the better ones can be sampled more In tournament selection, a specified group of individuals (typically two) are chosen at random from the population and the one with the better fitness (i.e., the lower standardized fitness) is then selected. When two bulls fight over the right to mate with a given cow, tournament selection is occurring.

2.5) Crossover Operation

The crossover (sexual recombination) operation for genetic programming creates

variation in the population by producing new offspring that consist of parts taken from each parent. The crossover operation starts with two parental S-expressions and produces two offspring S-expressions. The first parent is chosen from the population by the same fitness-based selection method used for selection for the reproduction operation. The operation begins by independently selecting, using a uniform probability distribution, one random point in each parent to be the crossover point for that parent. Note that the two parents typically are of unequal size. The crossover fragment for a particular parent is the rooted subtree which has as its root the crossover point for that parent and which consists of the entire subtree lying below the crossover point. Figure (2) shows the Crossover operation[2].

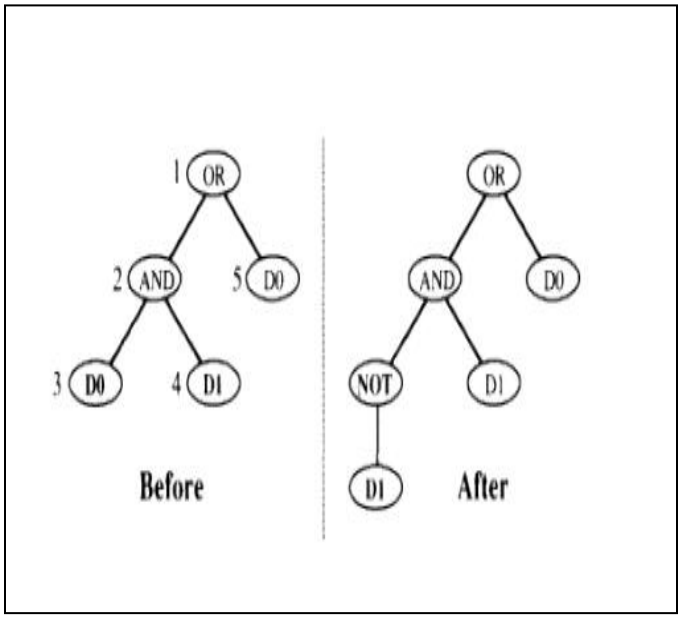


Figure(2). Crossover Operation

2.6) Mutation Operation

The mutation operation begins by selecting a point at random within the S-expression. This mutation point can be an internal (i.e., function) point or an external (i.e., terminal) point of the tree. The mutation operation then removes whatever is currently at the selected point and whatever is below the selected point and inserts a randomly generated subtree at that point. This operation is controlled by a parameter that specifies the maximum size (measured by depth) for the newly created subtree that is to be inserted. This parameter typically has the same value as the parameter for the maximum initial size of S-expressions in the initial random population.

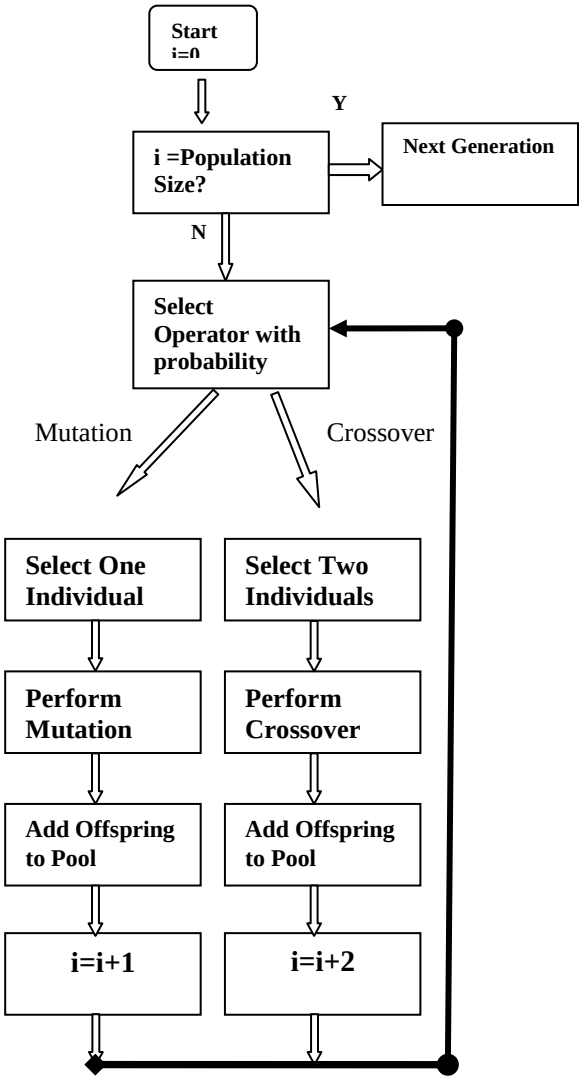
A special case of the mutation operation involves inserting a single terminal at a randomly selected point of the tree. This point mutation occurs occasionally in the crossover operation when the two selected crossover points are both terminals. See figure (3) for details.



Figure(3). Mutation Operation

3.7) GP architecture

Figure (4) shows the detailed flowchart for the genetic programming algorithm. The index `i` refers to an individual in the population of size *M*. The variable `GEN` is the number of the current generation. The box labeled "Evaluate fitness of each individual in the population" in this flowchart is computed using Raw Fitness as discussed earlier. This flow chart is often embedded within an outer loop for controlling multiple independent runs[2,5,8].



Figure(4). Detailed GP architecture

3)Inductive Logic Programming(ILP):

Inductive Logic Programming is a research area formed by the intersection of machine learning and logic programming. An ILP problem consists of a set of given positive and negative examples (E^+ , E^-), the background knowledge (B) and the set of possible programs(P) to learn. ILP aims to learn new Hypothesis (H) that justify the positive Examples. It is defined as[3,7] :

Fillset() procedure accomplished by initiating the set S to be filled for each example. the bottom clause (line 2) generated. Then, using the bottom clause, we generate all valid clauses4 (line 4), normalize them (line 5), and insert them in the set (line 6).The insertUpdateInSet() procedure works as follows. If the example class is positive, the clause is inserted into S and the positive counter is updated. If the example class is negative, the clause is not added to S, only the negative counter of the clause is updated. This means that the clauses generated from the negative examples that are not in S are discarded.

prune(), is invoked to remove useless clauses from S (e.g., clauses with coverage lower than some predefined minimum number of examples). Next, all negative examples are also processed and then the set is pruned again.

Each individual in the population of the GP can be regarded as an example for the ILP. The fitness function used for achieving a classification tool to split the Examples in GP population into Positive Examples and Negative Examples.

The covering of the Hypothesis (H) occur if (H) cover all the positive examples and none of the negative examples. The details of the ILP are shown in the algorithm below[4,11]:

ILP_Generate_H(*B*,*E*+,*E*−,*C*):
Given: background knowledge *B*, finite training set $E = E+ \cup E-$, constraints *C*.
Return: the *best* hypothesis that explains some of the *E*+ and satisfies *C*.
1. $S = \varnothing$
2. **foreach** $e \in E+$ **do**
3. *fillSet*(*S*,*B*, *e*,*C*)
4. **endforeach**
5. *S* = *prune*(*S*,*C*)
6. **foreach** $e \in E-$ **do**
7. *fillSet*(*S*,*B*, *e*,*C*)
8. **endforeach**
9. *S* = *prune*(*S*,*C*)
10. **return** *bestClause* (*S*,*C*)
****
fillSet(S, B, e, C):
Given: decorated set S, background knowledge B, example e, constraints C.
1. class = getExampleClass(e)
2. bottom = saturate(e,B,C)
3. do
4. clause = findValidClause(bottom,C)
5. clause = normalise(clause)
6. insertUpdateInSet(clause,S, class)
7. while clause ! = ∅
8.end

4) the proposed system ILPSAGP

The proposed system developed in this research extend the functionality of the Selection algorithm to take into consideration the Covering Test (CT) of the individuals(Programs). Before we can use the covering Test, the Individuals must be transformed into binary representation in the form of Logic Rules. The transformation set predefined template for each rule.

4.1) Individuals Transformation

The main idea behind the scene is accomplished via the process called Individual Transformation, this process take a program represents the individual in the population, and convert is into the norm of rule. the produced rule consists of two sections, first the condition part and the second the conclusion part.

If (conditions) then (Conclusion)

the number of symbols and their properties must be specified as well as the possible actions. Each symbol identifies an integer variable which is either read-only or read-write. a program can be provided with some general-purpose variables (a and b in the example). Additional symbols with special meanings can be introduced. an input symbol IN where incoming messages will occur and a variable OUT from which outgoing messages can be transmitted from could be added. An action set containing mathematical operations like addition, subtraction, value assignment, and an equivalent to logical negation58 is sufficient for many problems but may be extended arbitrarily. Before transform the individual, we must set the encoding table for the Variable – Length Individual.

To transform an individual, we follow the Encoding Table shown in table(2). For the Program bellow:

- Program Factorial(a)
  - p=1
  - b=a
  - while(b>0)
  - {
  - p=p*b
  - b=b-1
  - }

We should take into consideration that this program may be constructed using ordinary GP operation but for simplicity we omit the Tree representation for it.

Table(1) shows the process of (IT) for this program and the binary representation of the rules obtained from this process.

Table (1). The IT transformation of the Program.

Program (individual)	Transformed Program	Binarizaion
b=a	IF (start=1) AND true THEN b= a	0010 010 0001 0 110 0110 01 1010
p=1	(start)=1 AND true THEN p=1	...

while(b>0)
{
p=p*b
	IF (b>0) AND true
	THEN
	p=p*b
		...

b=b-1 }	IF (b>0) OR false THEN b=b-1	...

Table(1). Encoding of the IT Rules

1-Symbols	Encoding
0	0000
1	0001
Start	0010
id	0011
IN	0101
OUT	0110
a	1001
b	1010
2-Comp.	
>	000
>=	001
=	010
<=	011
<	100
!=	101
true	110
false	111
3-Concat.	
AND	0
OR	1
4-ACTION	
=X+Y	00
=X-Y	10
=X	01
=1-X	11

0010	010	0001	0	110	0110	01	1010
Start	=	1	AND	True	OUT	= X	b

This rule incoded using the IT table.

The following algorithm represents the overall design of the ILPSAGP system

****
ILPSAGP_Selection (POP[MAX])
****

- For i=1 to MAX do
  - IT_transfrom(POP[i])
  - Set_H(POP[i], *B,E+,E−,C,* Threshold)
  - *ILP_Generate_H*(*B,E+,E−, C*):
  -
- Normalize_H(POP)
- SL=Set_Tournament(H,POP)
- Return (SL)
- END

This extended selection algorithm uses Set_H() function to classify the population into Positive examples and negative examples depending on the Threshold value with the constraint C.
Then the ILP_Generate_H() algorithm described earlier, used for successive covering for best Hypothesis explain the set of { *B,E+,E−,C,*}. The Tournament selection algorithm used for the best individual cover the hypothesis.

5) Experiment results on ILPSAGP system

The proposed system called ILPSAGP was written in C#.net 2008. and has the Graphical User Interface shown in figure(5). Figure (5) shows the system initiated with ILPSAGP choice with (Population size =100, Generation =1000) for the Line Regression problem. The Error shows the difference between the optimal and the actual Fitness, it is for ILPSAGP , Error=6.838. and for ordinary GP with the same setting the Error obtained is (Error=12.48). this result shows that the ILPSAGP is more accurate than the GP. The sample table of line Coordinate points is shown in the table (3).

table (3). Line Points for experiment

X-Cord.	Y-Cord.
1	0.8
1.5	1
2.5	2
3.5	2.8
5	14
8.5	6.3
12	8
8	12

The line regression try to do the curve fitting for these points and returen the S-Expresion that is for these X-points as Input give us the Y-points as output with Error registered in the system.

6) Conclusion and future Work

In this paper, a new paradigm in Evolutionary algorithm was accomplished by extended the Genetic Programming by the fruitful technique called Inductive Logic Programming that operate with Logic Programming. The implemented system called ILPSAGP was built and tested with the Line Regression Problem. The experimental results shows that the proposed system has less Error than traditional GP algorithm due to use of Hypothesis discovery technique of ILP. The proposed system could be developed and extended with other selection algorithm rather than Tournament algorithm and could be turned from Binary rules into symbolic Rule-based GP.

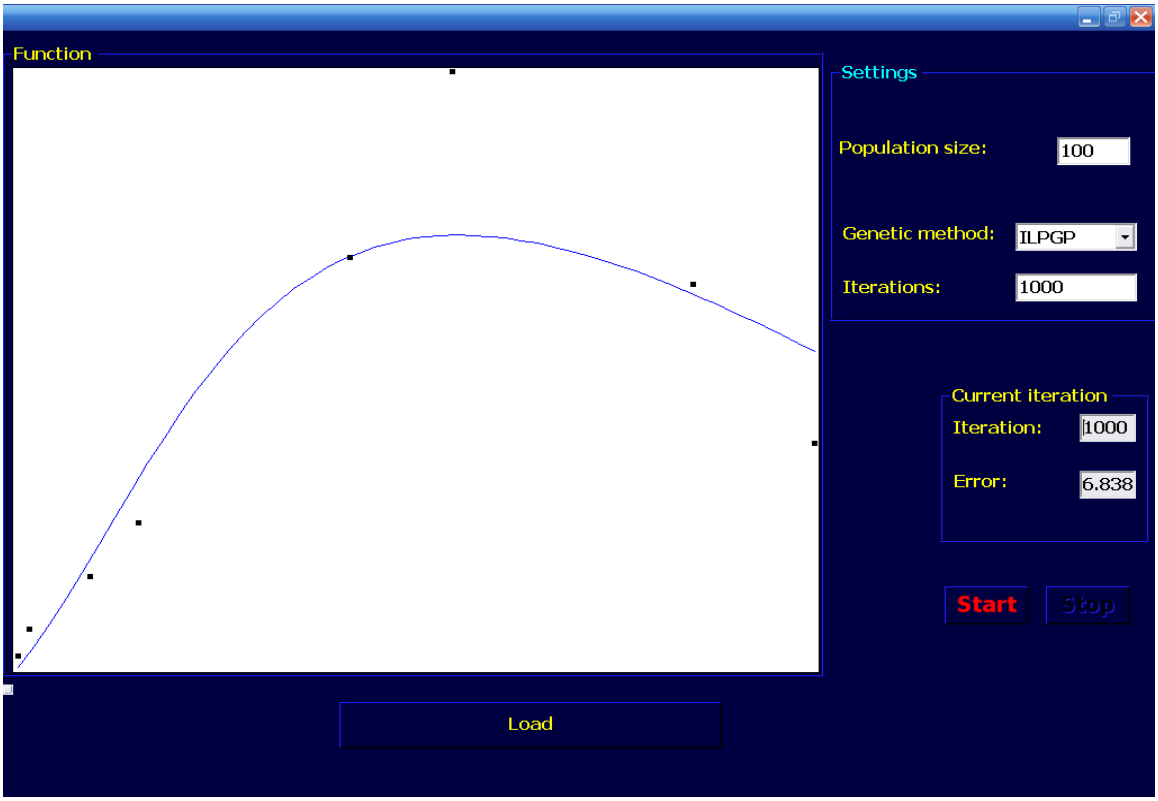


Figure (5). ILPGP initiated with Extended ILP.

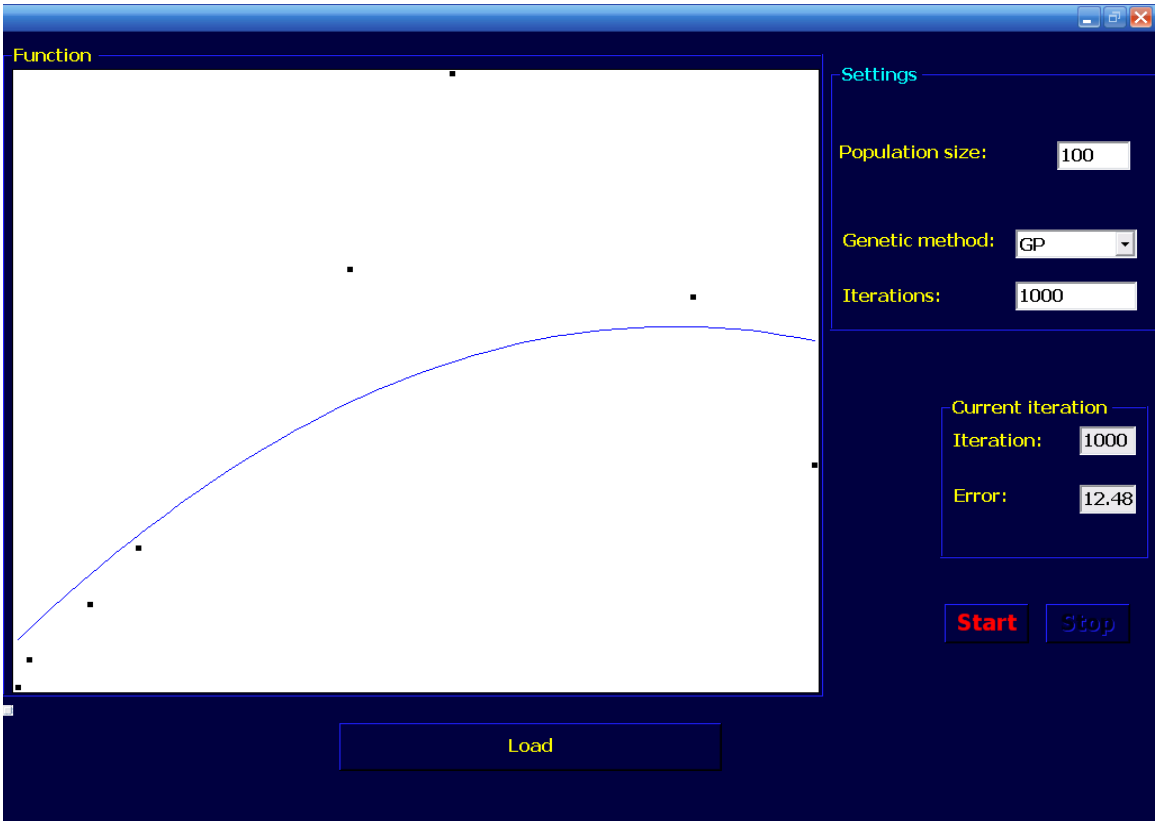


Figure (6). ILPGP initiated with traditional GP setting.

7) References:

1. Hendrik Blockeel Jan Ramon, "Inductive Logic Programming",17th International Conference, ILP, Corvallis, OR, USA, June 19-21, 2007.

2. Jian Huang and Adrian R. ,"Toward Inductive Logic Programming for Collaborative Problem Solving", NICTA Victoria Research Laboratory, University of Melbourne, Victoria, Australia, 2002.

3. John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89, pages 768–774, 1989. In proceedings [1953]. online available at http://dli.iiit.ac.in/ijcai/IJCAI-89-VOL1/PDF/123.pdf

4. John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. Genetic Programming III: Darwinian Invention and Problem Solving. Morgan Kaufmann, first edition, May 1999. ISBN: 978-1-55860-543-5.

5. Katsumi Inoue, "Circumscription Policies for Induction", Inductive Logic Programming,14th International Conf., Porto, Portugal, September 6-8, 2004.

6. Kristian Kersting, "An Inductive Logic Programming Approach to Statistical Relational Learning", IOS Press, 2006.

7. Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In Proceedings of the 1st International Conference on Genetic Algorithms and their Applications, pages 183–187, 1985. In proceedings [855]. Online available at http://www.sover.net/~nichael/nlc-publications/icga85/index.

8. StefanoFerilli,."Automatic Induction of First-Order Logic Descriptors Type Domains from Observations", Inductive Logic Programming,14th International Conf., Porto, Portugal, September 6-8, 2004.

9. S. H. Muggleton. Stochastic logic programs. In L. De Raedt, editor, Advances in Inductive Logic Programming, pages 254–264. IOS Press, 1996.

10. S. H. Muggleton. "Learning Structure and Parameters of Stochastic Logic Programs", In Proceedings of the Twelfth International Conference on Inductive Logic Programming (ILP-02), volume 2583 of LNCS, Sydney, Australia, July 9–11 2002. Springer.

11. Thomas Weise, "Global Optimization Algorithms Theory and Application",2end edition, preprinted copy,2009.