Dynamic Programming Inspired Genetic Programming to Solve Regression Problems

Asim Darwaish
National University of Computer and Emerging Sciences FAST
Islamabad, Pakistan

Hammad Majeed
NUCES-FAST
Islamabad, Pakistan

M. Quamber Ali and Abdul Rafay
NUCES-FAST
Islamabad, Pakistan

***Abstract*—The candidate solution in traditional Genetic Programing is evolved through prescribed number of generations using fitness measure. It has been observed that, improvement of GP on different problems is insignificant at later generations. Furthermore, GP struggles to evolve on some symbolic regression problems due to high selective pressure, where input range is very small, and few generations are allowed. In such scenarios stagnation of GP occurs and GP cannot evolve a desired solution. Recent works address these issues by using single run to reduce residual error which is based on semantic concept. A new approach is proposed called Dynamic Decomposition of Genetic Programming (DDGP) inspired by dynamic programing. DDGP decomposes a problem into sub problems and initiates sub runs in order to find sub solutions. The algebraic sum of all the sub solutions merge into an overall solution, which provides the desired solution. Experiments conducted on well known benchmarks with varying complexities, validates the proposed approach, as the empirical results of DDGP are far superior to the standard GP. Moreover, statistical analysis has been conducted using T test, which depicted significant difference on eight datasets. Symbolic regression problems where other variants of GP stagnates and cannot evolve the required solution, DDGP is highly recommended for such symbolic regression problems.**

***Keywords*—Genetic Programming; Evolutionary Computing; Machine Learning; Fitness Landscape; Semantic GP; Symbolic Regression and Dynamic Decomposition of GP**

I. Introduction

Since the inception of Genetic Programming by Koza [1], it is being used in various domains of Medical, Engineering and Computer science. GP is inspired by human biological evolution process. The leverage of Genetic Programming over some other inductive logic programming techniques is as it does not require any human interaction and domain specific knowledge. Comparatively GP is still a young field in research and has attracted substantial research community. Varieties of problems are being solved by genetic programming. For example: designing the robot controllers, discovering new quantum algorithms and continuous optimization problems etc. One of the important strength of Genetic Programming is its ability to find the solution of problem which human would probably never consider. On the other hand GP still persist some problems for instance scalability issues, GP Bloat are the most commonly concerned for research community.

It has been observed that GP struggles to evolve on different symbolic regression problems and stagnation occurs. The reason of stagnation and insignificant improvement is high selective pressure. In some problems where input ranges are very small and fewer generations are allowed the stagnation occurs and GP can not evolve the required solution. The primary goal of this research work is to overcome these aforesaid problems for example GP stagnation etc. This research work also diverts the focus of GP in order to overcome GP struggle and stagnation. Furthermore, it has been also observed when stagnation occurs in GP, the size of tree is continuously increasing with less or no performance improvement with respect to fitness. The same problem associated with GP is also reported by Maarten Keijzer and had mentioned that due to very small range of inputs the traditional GP struggles to evolve or cannot evolve a model for some symbolic regression problems [2]. It has also been reported in literature that the curve of performance improvement of GP is much higher at early generations as compared to later generations. As the number of generation increases there is minor improvement in performance with respect to fitness [3], [4]. The said scenario is termed as GP bloat which is defined as the growth of GP tree increases without improvement in fitness or with insignificant improvement in fitness.

Substantial portion of research has been conducted in Genetic Programming to overcome aforesaid problems like semantic GP by Moraglio [6] and Sequential Symbolic Regression SSR [7]. In order to tackle state of the art problems, a novel approach has been proposed named Dynamic Decomposition of Genetic Programming (DDGP). DDGP has some similarities with SSR and semantic GP. However, in order to find a particular solution, proposed approach decomposes the GP run into number of dynamic runs instead of using single run. The cumulative sum of these sub-runs are then merged to get the final solution of a problem. Keeping in view, the aforesaid problems, proposed approach got an inspiration form dynamic programming. In which previously found sub-solutions helps in finding the final solution. Empirical results mentioned in section V depicted that DDGP is much better than SGP in term of fitness improvement. It is highly recommended to use DDGP in the problems where standard GP struggles to evolve and stagnation occurs. DDGP is the main contribution of this research work, which is comprised of small runs instead of single independent run of GP. In DDGP the succession of these sub-runs participates in finding the final solution. DDGP simply add these sub-runs and reaches upto desired solution. DDGP incorporated the dynamic decomposition with the help of special parameter named error change parameter. The paper is organized as follow: section II comprises of literature review and study of previous approaches to dynamic fitness landscape in GP section III describes Dynamic Decomposition of GP;

; section IV comprises of experimental setup and symbolic regression problems suited for this approach. Section V ; summarizes the results and significance of DDGP; in last, section VI concludes the achievements of DDGP and highlights the future work directions.

II. Literature Review

The first and most popular approach for creating sub modules was coined by Koza named Automatically Defined Function in (1994). Koza proposed ADF for exploring the regularities and modularities of the search space when dealing with complex problems in context of GP. The proposed architecture decomposed the problem into simpler sub problems. Koza has implemented these steps within the run of GP. The architecture proposed by koza exploits the problem regularities through modularization. Problem decomposition is being done manually such as number of ADFs, number of arguments for each ADF. Moreover, interactions of ADFs were restricted by user defined value. In addition to this all these should be specified prior to the run of GP. Another limitation of this approach is that there is single decomposition step. It may be possible more complex problem with respect to GP require many decomposition steps.

Ahmed Kattan and Alexandar et al [5] performed unsupervised problem decomposition using genetic programming at two levels. At top level GP evolves way of splitting the fitness cases in each subset. At lower level GP evolves the program that solves the fitness cases in each subset. The objective of their contribution was to reduce complexity and to discover regularities in problem space. Two main steps of their work was training and testing. In training the system learn to divide the training cases into different group based on similarity. Training is further split into re-sampling and solving phase. Re-sampling tries to discover best decomposition of problem and solving phase tries to solve the problem by solving sub problem independently discovered in re-sampling phase.

Otero and Johnson [9] proposed a sequential covering strategy for problem decomposition specific to Boolean domain. Prior to Otero and Johnson, focus of previous research was on discovery of modules rather than the use of modules in problem decomposition [9]. They had introduced three distinct steps as like ADF. First one is decomposition of the problem, second is searching for a sub problem solution and third one is combination of sub solutions into a complete solution. SCGP is started with an empty solution by considering all the input cases and evolves the partial solution. SCGP add this partial solution to a solution tree. Next step is the removal of input cases for which SCGP predicts accurately and repetition of process until all input cases are removed. The removal of input cases, after each iteration changes the search space effectively for next iteration and allows GP to evolve different parts of the problem. They have used mask selector based on semantic crossover to combine the partial solutions from solution tree to final solution. Experiments were conducted on two Boolean benchmark problems namely even parity and multiplexer. Their work is only limited to Boolean domain and does not deal in real domain. Moreover, they gave no idea of generalization for unseen input cases.

Nabi and et al [8] have proposed automatic problem decomposition for increasingly complex problems called differential grouping. It is based on divide and conquers approach. According to Nabi the growth in dimensions impacted the performance of evolutionary algorithms adversely. The differential grouping uncovers the underlying interaction of decision variable and form subcomponent with minimum number of interdependencies. Nabi highlighted the drawback of previous approaches of unequal distribution of computational budget among subcomponents. They have allocated the computational budget according to the contribution of subcomponents [8]. This work also showed that near optimal decomposition is beneficial and along with contribution based approach can improve the performance in large scale optimization with up to 1000 decision variables. They have used additively separable function [8] for differential grouping based on two stages namely grouping stage and optimization stage. As mentioned earlier in grouping stage the interaction of underlying variable structure is identified and in optimization stage the subcomponent discovered in grouping stage are optimized in round robin fashion. The experiments conducted on IEEE CEC 2010 benchmark and used 20 benchmarks functions in order to evaluate the performance and compared with CCVIL. The results gave 100 percent accuracy on 13 benchmarks out of 20 benchmarks.

Luiz Otavio V.B. Oliveira et al [7] proposed a framework to deal with complex problems using GP named sequential symbolic. Their main contribution was transformation of original problem into potentially simpler problems based on semantic distance and semantic crossover. Luiz Otavio work is inspired from sequential covering strategy SCR, same to one proposed by Otero and Johnson [9]. The difference between SSR and SCR is of transformation and reduction, In SSR the problem is transformed into simpler problem, while in SCR the problem is reduced, after each iteration. In SCR the training cases covered by an iteration are removed which results in reducing the size of problem. Iterative solution in SSR allows GP to focus on different aspect of original problem and combines the individual solution (sub problems) using GSC. After generating a sub optimal function the residual is approximated by another function [7].

The concept of semantic Genetic operator SGP is coined by Moraglio [6]. As Moraglio work combines individual at random therefore, exponential growth is reported in SGP. SSR has overcome this flaw by finding the individual with minimum error on desired output vector. SSR does not need to keep all the solution in memory. Experiments of SSR are conducted using 8 univariate polynomial function of degree 3 to 10 with real valued coefficient [-1,1] same as Moraglio et al [6]. The results of SSR are same as GP but better than SGP along with better generalization than SGP. However, the critical part of SSR is the setting of different parameter and it does not reduce the fitness budget effectively. Moreover, SSR has not been validated in more complex Symbolic regression problems for example the GP benchmarks (white et al, 2013) yet.

Tomasz P. Pawlak and Krzysztof et al [10] proposed semantic backpropagation for designing search operators in Genetic Programming. According to them inversion of program execution can generate the subtasks from the original task. These sub tasks can be solved using exhaustive search in constraint set of programs [10]. For the sake of desired intermediate output, their algorithm can heuristically inversed

the execution of evolving program. With the help of inverse operator they can get desired output of any sub sequent node. The proposed algorithm randomly selects any node and gets desired output. After obtaining desired output the algorithm searches for a program with a match or very close match in the repository. Tomasz P et al have introduced two kinds of operators for this purpose namely Random Desired Operator (RDO) and Approximately Geometric Semantic Crossover Operator (AGX). RDO is useful for Boolean and regression problems and for unknown target output due to some confidential reasons, AGX operator is used. The empirical results showed that their works helps the evolution at identifying the desired intermediate states and also improve the search process and make it more efficient [10]. Moreover, they have introduced the program semantic in order to analyze the program behavior. Their results showed that the inversion of program can be a feasible for automatic programming algorithm, including GP, with property of problem decomposition [10].

Amin Lamine et al [11] conducted a study on finding solution to constrained optimization problems (COPs). COPs are generally considered as NP-Hard problems. The proposed technique is S&D (solve and decompose). It uses depth-first iteration technique based on decomposition of problem. The proposed strategy uses the solution of COP which is feasible, that is found by any exact technique, which is further decomposed into smaller sub problems. To strengthen the cost based filtering it uses values of feasible solution as the bound to be added to sub problems. Exploitation of feasible solution (promising region of search tree) is done for finding sub problems that show more promise in finding good solutions. Heuristic is needed that may be adaptive for comparison of two sub problems for finding better one to improve whole solution because, the objective function is linear .Depth first approach is used for their exploration. This whole process is continued in proposed work until some criteria is reached where further decomposition is stopped. After this Branch and Bound type of exact methods are used to solve these sub problems in optimal way. Experiment results for the S&D approach show that improvements in order of three magnitudes were achieved in comparison with Branch and Bound methods, with respect to their runtime.

David Medernach et al [20] proposed a novel approach named Wave GP. Wave is the form of semantic genetic programming which works on periods. Periods are short genetic programing runs. This work shares some similarities with methods such as Sequential Symbolic Regression. The main idea of this work is to run succession GP periods and produce cumulative solution by training on the basis of previous residual errors. The algebraic sum of best evolved period is called final solution. New periods are started when the rate fitness gain slows down. Residual errors are optimized for each successive period on the basis of previous Heterogeneous configuration is applied across the periods, which results in different generation span for different periods on the basis of their progression. Partial population is renewed with 80% new individuals at the beginning of every new successful period. Wave performs equal or better than standard GP with or without linear scaling. It performs significantly better than GSGP.

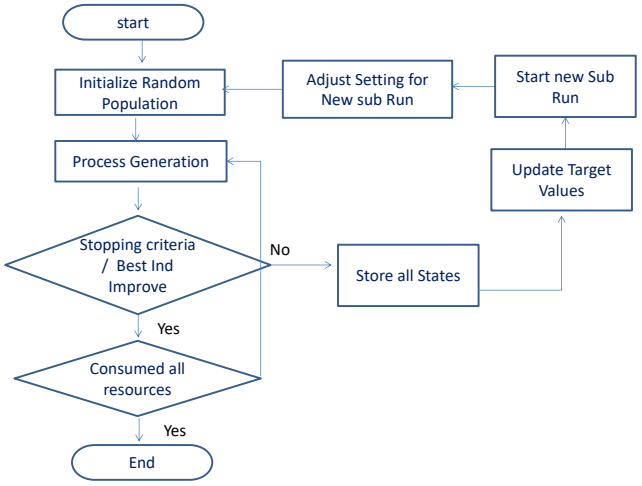


Fig. 1. Flow chart of Proposed Approach DDGP is depicted.

III. Dynamic Decomposition of GP

Dynamic decomposition of genetic programming is an idea inspired from dynamic programming. The symbolic regression problems where standard GP struggles and stagnation occurs, DDGP is quiet successful in such problems. Moreover, DDGP is also beneficial to avoid GP bloat. The DDGP decomposes the original GP run into number of sub-runs (problem dependent) in order to find the final solution, which cannot be found by traditional GP. Proposed approach incorporates divide and conquer strategy, which is quiet helpful in those areas where standard GP struggles. DDGP starts like a traditional GP and tries to find the solution with the help of fitness function. DDGP used a special parameter named error change parameter. The purpose of error change parameter is to monitor the best individual after each generation. If the best individual does not improve in term of fitness for x number of consecutive generations, then decomposition comes into an action. The proposed methodology stores all the states of current run and initiates a new run. The new run requires some parameter adjustments for example modification of target values and number of generations. The recursive process of this mechanism either yields required final solution or end up with termination criteria. At the end of each sub-run, DDGP stores the best individual. On the basis of this, target values are updated which lead to change in fitness landscape. This mechanism changes the target value for the subsequent run. In the end, the final solution is the cumulative sum of all the linear sub runs. The flow chart of proposed DDGP is shown in figure 1.

Pertaining to the proposed idea of DDGP, it is deemed appropriate to mention a mechanism which decides, when decomposition will incorporate. For the sake of this, error change parameter has been used as mentioned above. This special parameter works on the basis of best individual of every generation. For intelligent stopping mechanism the error change parameter monitors the best individual of each generation. If the performance of best individual does not improve or remain same for X number of consecutive generation, then current run is halted and all states are stored.

The argument is supported by stasis introduced by Blackburn (1995). The stasis is the time period in human biological evolution process, when no change occurs for some time period in human biological evolution. Similarly to this when best individual does not improve, current run is halted and incorporates the decomposition by introducing new sub runs. For above mentioned stopping mechanism some threshold could have been used but DDGP did not use this option. After intelligent stopping and saving the states, new run is initiated. The proposed approach repopulates and uses the same desired parameter setting except number of generations. For number of generations, DDGP subtracts the number of generations consumed by previous run from total allocated generations as given below: $M_f = M_1 + M_2.....M_n$

$$G_{M_i} = G_t - G_{M_i-1} \tag{1}$$

Where M_1 to M_n are the number of decomposed models or sub models which are dynamic in DDGP and depends upon the nature of problem while M_f is the final model. G_t is the total number of allocated generations and G_{M_i} is the number of generations for current sub run or sub model and G_{M_i-1} is the number of generations consumed by previous sub run or sub model.

When the best individuals does not improve or remain same for x number of consecutive generations, the intelligent stopping criteria stops the current run and stores all the states. For new sub-run target value is modified by subtracting the obtained output of a previous sub run from the desired target value. This lead to change in fitness landscape for subsequent runs. The procedure for updating the target value is given as under:

$$O_{M_i} = O_t - O_{M_i-1} \tag{2}$$

Where O_{M_i} is the target output for current run and O_t is the desired output against the input cases and O_{M_i-1} is the output model evolved by the previous sub-run. the final solution will be the algebraic sum of all the sub runs or sub models i-e.

$$Final(Solution) = \sum_{i=1}^{n} M_i \tag{3}$$

The proposed ideal help us in finding the target solution using algebraic sum of all the runs. Moreover, it also overcomes the GP stagnation and bloat problem because of the mechanism, which is monitoring the performance of best individual after each generation for fitness improvement. Due to which DDGP eradicates the chance of stagnation and GP bloat. Moreover, DDGP also improves the speed and accuracy of standard GP, the empirical results mentioned in section IV are evidence to this argument.

A. DDGP Algorithm

Algorithm 1 presents the high level pseudo-code fo DDGP. Fitness cases comprised of inputs / outputs usually provided to algorithm 1. It starts with empty solution S and constructs the required solution iteratively just like traditional GP. If the required solution is found at k-th generation, the required solution is added to S. Otherwise proposed approach monitors the best individual after each generation using Errorchange($Best_{Individual}, Gen$). If fitness does not improve for x number of consecutive generations, Algorithm 1 stops the current run and updates the target vector. The partial solution evolved by recently stopped run is subtracted from desired output and new target vector is updated for subsequent run. Moreover, the partially evolved solution is added to S. Now algorithm 1 runs on original inputs and updated target vectors and process is repeated until termination criteria is met. Finally the algebraic sum of all the partially evolve solutions are merged and final solution can be achieved, which was difficult to obtain using single run or using standard GP.

Input: fitness cases (T), DDGP Parameters, Stopping Criteria
$Input \leftarrow (t_1, t_2, ..., t_n for t_k \in T)$;
$Output \leftarrow (O(t_1), O(t_2), ..., O(t_n) for O(t_k) \in T)$;
`/* Construction of Required Solution Iteratively */`
$S \leftarrow 0$;
while *While stopping criteria does not reach* **do**
  $M_i \leftarrow$ `RunGp` (*input, output*) ;
  if (`MSError`$(M, output) \leq 0.001$) **then**
    $S \leftarrow$ `AddSolution`(M_i);
  else
    `ErrorChange` $\leftarrow$ `FitnessImprovement`$(Best_{Individual}, Gen)$;
    Return `ErrorChange`;
  end
  `/* Set Threshold according to nature of problem and resources */`
  $\epsilon \leftarrow value$;
  if `ErrorChange` $\geq \epsilon$ **then**
    `StopCurrentRun`;
    $Run_i \leftarrow$ `StoreSt`(`UpTarget`$(output, Parameters)$;
    $S \leftarrow$ `AddPartialSol`(Run_i);
    $M_i \leftarrow$ `RunGp`$(input, updatedoutput)$;
    Go to Step 6 and Repeat the process ;
  else
    Go to Step 5;
  end
end

Algorithm 1: Algorithm for Dynamic Decomposition of GP

B. Running Example

Lets take a look at example of divide and conquer approach used by DDGP in table I for solving equation 4. The first columns contains the input value shown as x ranging from [-1,1]. The second column contains output value shown as y. It can be seen from the table that the difference between input and desired output values are very large. In these cases where this difference between desired output and input is very large, Standard GP struggles and stagnates. Stagnation can be referred as condition where GP is unable to improve the fitness and gets stuck at some value despite further evolution occur over the generations. In table I third column shows the evolved model M1. It can be seen from table I that GP was suppose to evolve output y of 499 against input value -1 of x, but it was unable to achieve it, instead it stagnated at the output

value of 202.1. In standard GP there is no way to overcome stagnation in problems where there is huge difference between input and desired output values along with small generation sizes. In proposed approach when stagnation happens divide and conquer strategy is used. As shown in table I in first row third column M1 is 202.1 this is the evolved model until stagnation happened. At this point in DDGP new desired output y' is evolved by subtracting M1 from original desired output y. Column four show the new desired output value y' of 296.9. DDGP starts the sub run and the new value of y' is evolved as it is still large as compared to input x, it results in stagnation and DDGP stops after evolving M2 model with value of 150.1 . At this stage new desired value output y" is evolved using model M2. Again DDGP is started with input x = -1 and desired output y" value of 146.8. Again same process is repeated and new modal is evolved using divide and conquer strategy until desired outcome or termination criteria is achieved. In table I value of y" is 48.7, and at this stage DDGP evolved model M4 with value 48.69 which is equal to the desired value resulting in termination of DDGP. At this stage all the partially evolved models are combined to form complete solution to the problem. In the example cumulative result is formed by adding partial solutions of M1, M2, M3, M4 as shown in 5 . By using this approach DDGP overcomes the problem of stagnation which standard GP is unable to solve.

$$Y = 500 + x \tag{4}$$

$$Y = M1 + M2 + M3 + M4 \tag{5}$$

TABLE I. EXAMPLE OF DIVIDE AND CONQUER STRATEGY USED IN DDGP

X	Y= f(x)	M1	Y= Y-M1	M2	Y=Y-M2	M3	Y= Y-M3	M4
-1	499	202.1	296.9	150.1	146.8	98.1	48.7	48.69
-0.99	499.1	202	297.1	150.2	146.9	98.2	48.7	48.7
-0.98	499.2	201.9	297.3	150.3	147	98.3	48.7	48.7
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
1	501	201	300	152	148	100	48	48

IV. EXPERIMENTS

A. Dataset

This paper targets the symbolic regression problems to perform variety of experiments. The symbolic regression tasks are of varying difficulties in order to test the proposed GP variant named DDGP and methods over wide range of dataset with varying complexities.

B. Symbolic Regression

An opposed to traditional regression, symbolic regression does not make any assumption regarding underlying relationship between dependent and independent variable [7].Symbolic regression try to find a model which completely satisfy all the inputs and outputs. The model in symbolic regression is composed of mathematical expression which describes the relationship among one dependent and multiple / single interdependent variables. The purpose of traditional regression techniques is to seek optimization of parameters for already specified model. While in symbolic regression goal is to find the model which is a mathematical expression, in short symbolic regression find both model structure and model parameters.

C. Experimental Setup

The experimental setup for DDGP is Symbolic regression problems. Symbolic regression is usually solved by genetic programming and is supervised learning. In GP, the system is presented with data points from which GP construct a mathematical modal which fit all the inputs and outputs points in the dataset. Usually the fitness function use for obtaining this model accurately is Means Squared error [12].

Selection of good set of problems for conducting experiments on symbolic regression is difficult task. Because no well-formed established benchmarks has been formulated. This paper takes most of the problems from past papers that perform improvement on symbolic regression. Keeping in view the stagnation problem associated with GP, following symbolic regression are used for experimental setup. First three symbolic regressions problems are chosen from Maarten Keijzer's benchmark and other 8 symbolic regression are comprises of 8 univariate polynomial function of degree 2 to 9 .

$$f(x) = 0.3xsin(2x) \tag{6}$$

$$f(x) = x^3exp - xcos(x)sin(x)(sin2(x)cos(x)1) \tag{7}$$

$$f(x) = Sqrt(x) \tag{8}$$

$$f(x) = x^2 + 500 \tag{9}$$

$$f(x) = 200 + x^2 \tag{10}$$

$$f(x) = x^4 + 4x + 700 \tag{11}$$

$$f(x) = f(x) = x^5 + x^2 + sin(x) + 400 \tag{12}$$

$$f(x) = x^6 + 7x^3 + 2x^2 + cos(x) + 300 \tag{13}$$

$$f(x) = x^7 + 5x^4 + 3x^3 + cos(x) + 600 \tag{14}$$

$$f(x) = x^8 + 3x^6 + 2x^4 + sin(x) + 900 \tag{15}$$

$$f(x) = x^9 + 9x^7 + 5x^5 + cos(x^3) + 450 \tag{16}$$

The common parameter setting for experimental setup involves the population size which is set to 5000 for all symbolic regression problems in suited experiment setup for DDGP. Different generation size is set for different problems as specified in Maarten keijzer paper for fair comparative analysis. The input range of all problems is from [-1,1] to support keijzer argument. According to keijzer when input range is very small and few generations are allowed, selection pressure for selecting the right range is so high on GP. Due to this GP spends most of the time for finding the particular value and if found in some cases, the diversity has dropped substantially. The functional set comprises of arithmetic and logarithmic functions. Crossover probability is set to 0.7, mutation is set to 0.2 and 0.1 is set for reproduction. Minimum depth of tree allowed is 2 and maximum allowed depth is set to 16. According to DDGP the error change parameter is set to five for intelligent stopping criteria, which means if the best individual does not improve or remain same for five

consecutive generations. The algorithm stops the current run and initiates a new run. The question is how much runs would be there for any problems? In case of DDGP it is dynamic; depend upon the nature of problem and termination criteria. The detailed configuration for all the symbolic regression problems suited for DDGP's experimental setup is given in table II.

GP System used for this research work is LIL GP developed by Dr Bill punch and Douglas Zongker at Michigan State University. It is written in C language for the sake of high execution speed, modularity, ease of use and support number of other options. Lil GP code is modified according to proposed methodology. Necessary changes and modifications are made in standard LIL GP code in order to examine DDGP. The naming convention used in this paper are given as; the "Pop" means the population size; "Gen No" means the total number of generations allowed for a problem at start of run. Input ranges are used from [-1,1] and two hundred fitness cases have been used for testing each regression problem. ECP stands for error change parameter which is used to monitor the stopping criteria of intermediate run as stated above.

TABLE II. GP: PARAMETER SETTINGS

Problem Name	Pop Size	Gen No's	ECP	Range	Fitness cases
Equation No 6	5000	30	5	[-1,1]	200
Equation No 7	5000	30	5	[-1,1]	200
Equation No 8	5000	25	5	[-1,1]	200
Equation No 9	5000	50	5	[-1,1]	200
Equation No 7	5000	30	5	[-1,1]	200
Equation No 11	5000	30	5	[-1,1]	200
Equation No 12	5000	35	5	[-1,1]	200
Equation No 13	5000	35	5	[-1,1]	200
Equation No 14	5000	35	5	[-1,1]	200
Equation No 15	5000	35	5	[-1,1]	200
Equation No 16	5000	35	5	[-1,1]	200

V. RESULTS AND DISCUSSION

In experimental phase, the proposed approach (DDGP) is tested on suited symbolic regression problems from Maarten keijzer benchmarks and on eight univariate polynomials from 2 to 9. It is imperative to mention that both SSR and Moraglio [6] have the same benchmark for their experimental setup (Polynomials from 2 to 9). DDGP is close to the work performed, by Luiz and Otero in SSR. They have performed transformation of original problem into simpler problem, while this paper performed dynamic decomposition. The problem of stagnation and GP struggles on different problems was also highlighted by Maarten Keijzer. However, his work was related to linear scaling. The results of experiments depicted that DDGP outperforms the standard GP.

For monitoring stagnation and incorporating decomposition the value of error change parameter can be x number of generations. In all the experiments, performed on Maartin Keijzer's benchmarks and eight univariate problems the values of x is set to five. However, error change parameters is also tested on the values 3,4,5,6. The behavior of DDGP was relatively better on using x = 5 for error change parameter. Due to this x was set to five during evaluation of DDGP in context of error change parameter.

A. Suited Benchmarks

In experimental setup the first three mathematical equations are taken form keijzer paper. Same parameter settings are used as mentioned by keijzer, for fair and transparent comparative analysis. The mathematical expressions were $f(x) = 0.3xsin(2x)$, $f(x) = x^3exp - xcos(x)sin(x)(sin2(x)cos(x)1)$ and $f(x) = Sqrt(x)$. The input range for all these equations was between [-1, 1] as specified by keijzer. The error change parameter in DDGP was set to five. Two hundred fitness cases are chosen for each equation that are generated at regular intervals between the range [-1,1]. Thirty independent runs of SGP are conducted on 30 different seeds for each equation of keijzer. Same seeds are used for 30 runs of DDGP on above mentioned three problems. The results produced by DDGP were many times better than SGP. The figure 2a showed the average fitness of SGP and DDGP for 30 runs. From the figure 2a it is clearly obvious that DDGP beats the SGP and the average maximum fitness of DDGP was 0.188797 while in case of SGP it was 0.071755133 for equation 6 of Keizjer. Moreover, it can be noted from the figure 2a that in case of SGP after second generation the best individual is no more improving upto 30th generation and stagnation occurred. Due to stagnation The SGP is not able to solve the problem. While DDGP helps to remove the stagnation of GP and it is improving in fitness graph as shown in figure 2a.

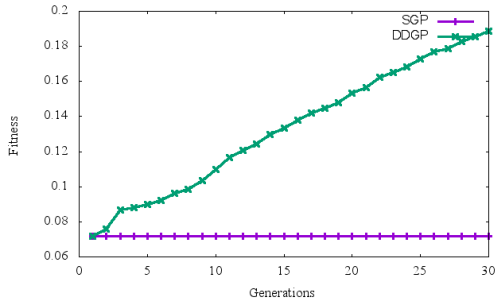
For second and third equations of keijzer the experiment comprises of 30 independent runs for both SGP and DDGP on different seeds as like for equation (6) . The results in a figure 3a and 4a showed that DDGP outperforms the Standard GP. Moreover, the scenarios where the input range is so small and few generations are allowed, the standard GP cannot find the solution due to stagnation and high selective pressure. While in those scenarios the DDGP performs far better than SGP. Furthermore, speed and performance of DDGP is better than standard GP and also overcomes the problem of GP bloat.

B. 8 univariate Polynomials

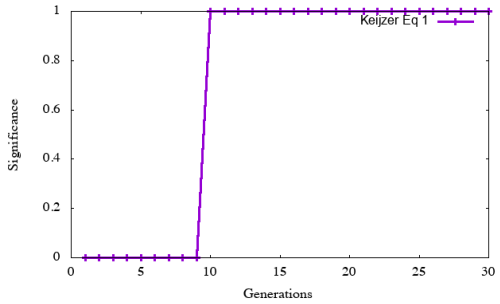
In this phase of experiments evaluation of proposed approach DDGP is done on eight univariate polynomials [7] and Moraglio [6] from degree 2 to 9. Thirty independent runs have been conducted for each polynomial (from 2 to 9) using both Standard GP and DDGP. Each polynomial consists of 200 input fitness cases between the range of [-1, 1]. Other parameter settings are specified in table II. The error change parameter is used to monitor the best individual after each generation and for intelligent stopping of current run, when stagnation occurred. The decompositions of models were dynamic. The results of each polynomial 2 to 9 showed that DDGP performs better than Standard GP. Moreover, using proposed approach GP bloat problem can be minimized and it has been observed from empirical results that, performance in term of fitness and speed of DDGP is much better than Standard GP. The figure 5a, figure 6a, figure 7a, figure 8a, figure 9a, figure 10a, figure 11a and figure 12a are evidence that the performance of DDGP is better than Standard GP.

C. Statistical Analysis

Student T Test has been conducted on all the datasets (polynomial 2 to 9) and (suited Keizjer's benchmark equa-



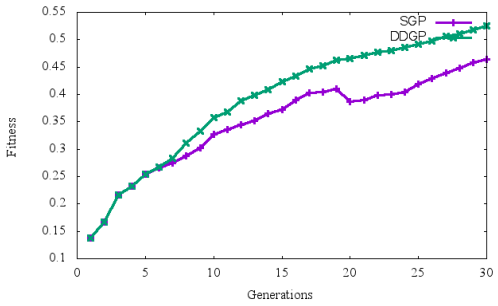
(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = 0.3xsin(2x)$ between range -1 to 1. 1



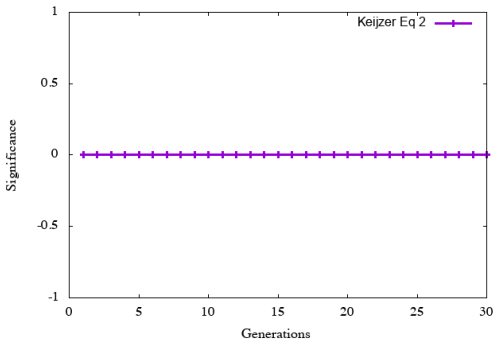
(b) T Test of Keizjer's benchmark equation 1 $f(x) = 0.3xsin(2x)$ after each generation between DDGP and SGP

Fig. 2. DDGP and SSGP Comparison with respect to fitness and T Test for Kezijer's benchmark equation 1

tions). The purpose of student T test is to determine the significant difference between the results obtained by DDGP and SGP. The T test has been performed on the best individual obtained after each generation by DDGP and SGP. As in experimental setup 30 independent runs are conducted for each dataset. The T test examine the significant difference between the best individual obtained through DDGP and SGP after each generations of all 30 runs on all datasets. The T Test shows that the fitness improvement on seven datasets (2b,4b,6b,5b,8b,9b,10b) are significant improvement with 95% confidence and more while, on four datasets (3b,7b,11b,12b) the improvement is not quiet significant. However,results are better than SGP.

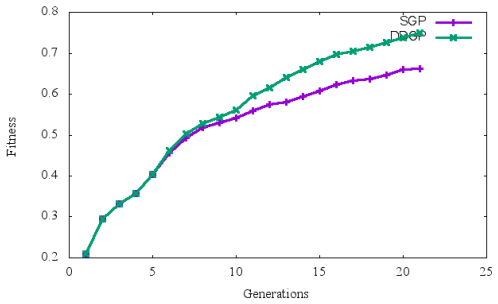


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = x^3expxcos(x)sin(x)(sin2(x)cos(x)1)$ between range -1 to 1 1

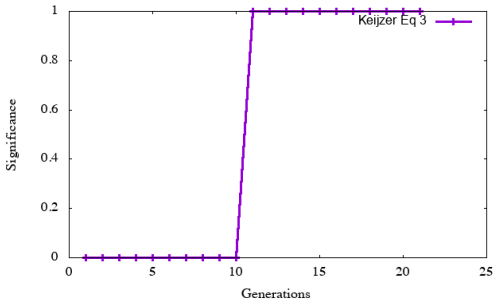


(b) T Test of Keizjer's benchmark equation 2 $f(x) = x^3exp - xcos(x)sin(x)(sin2(x)cos(x) - 1)$ after each generation between DDGP and SGP

Fig. 3. DDGP and SSGP Comparison with respect to fitness and T Test for Kezijer's benchmark equation 2

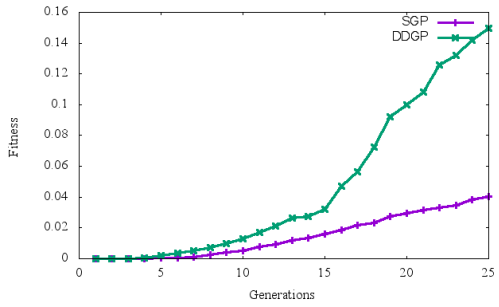


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = Sqrt(x)$ between range -1 to 1

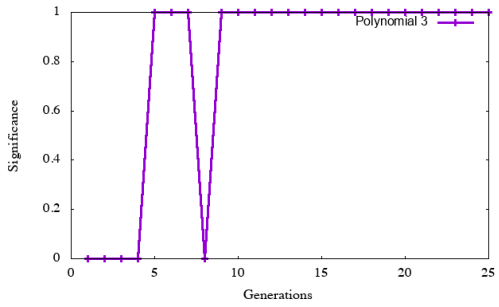


(b) T Test of Keizjer's benchmark equation 3 for dataset $f(x) = Sqrt(x)$ after each generation between DDGP and SGP

Fig. 4. DDGP and SSGP Comparison with respect to fitness and T Test for Kezijer's benchmark equation 3

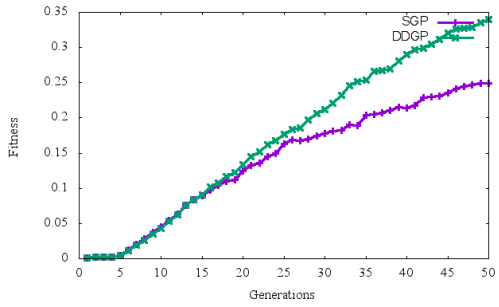


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $Y = x^3 + 500$ between range -1 to 1

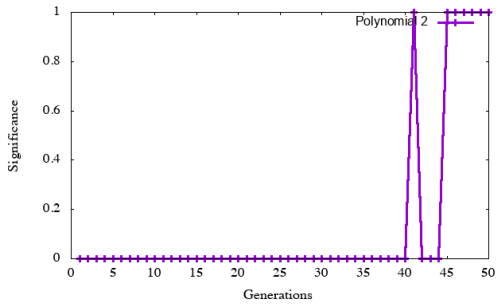


(b) T Test on Polynomial 3 $Y = x^3 + 500$ after each generation between DDGP and SGP

Fig. 5. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 3

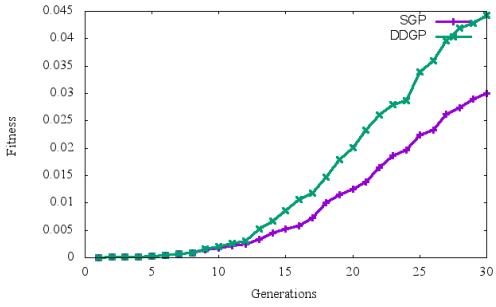


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $Y = x^2 + 200$ between range -1 to 1

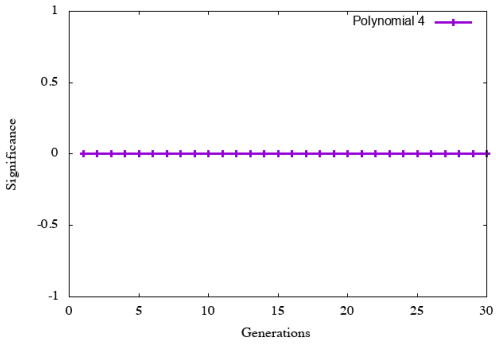


(b) T Test on Polynomial 3 $Y = x^2 + 200$ after each generation between DDGP and SGP

Fig. 6. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 2

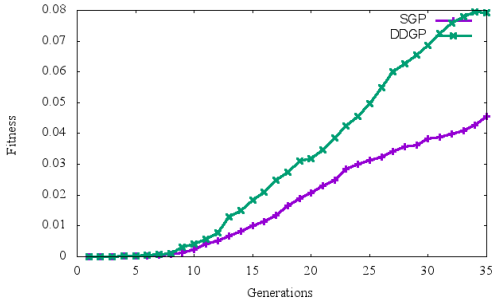


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = x^4 + 4x + 700$ between range -1 to 1

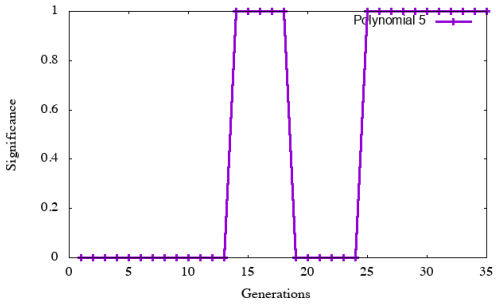


(b) T Test on Polynomial 4 $f(x) = x^4 + 4x + 700$ after each generation between DDGP and SGP

Fig. 7. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 4

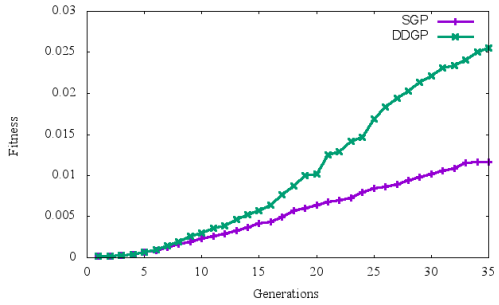


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = x^5 + x^2 + sin(x) + 400$ between range -1 to 1

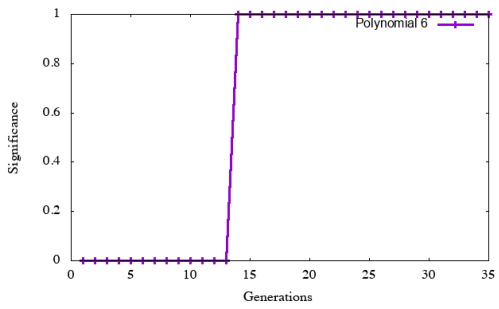


(b) T Test on Polynomial 4 $f(x) = x^5 + x^2 + sin(x) + 400$ after each generation between DDGP and SGP

Fig. 8. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 5

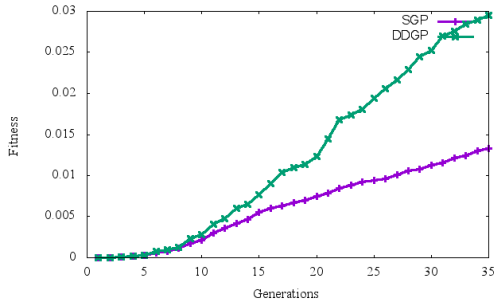


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = x^6 + 7x^3 + 2x^2 + cos(x) + 300$ between range -1 to 1

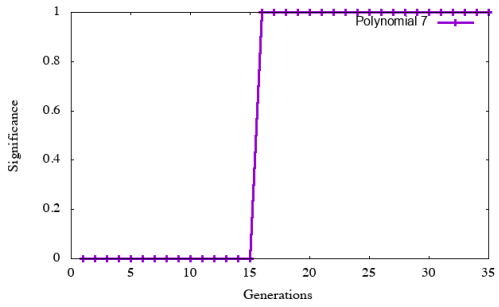


(b) T Test on Polynomial 4 $f(x) = x^6 + 7x^3 + 2x^2 + cos(x) + 300$ after each generation between DDGP and SGP

Fig. 9. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 6

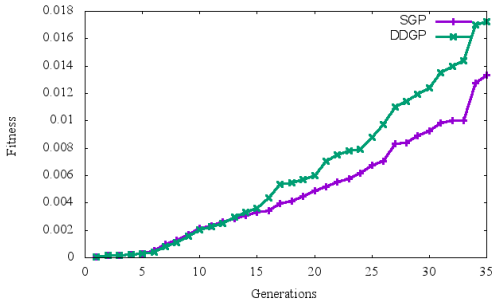


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = x^7 + 5x^4 + 3x^3 + cos(x) + 600$ between range -1 to 1

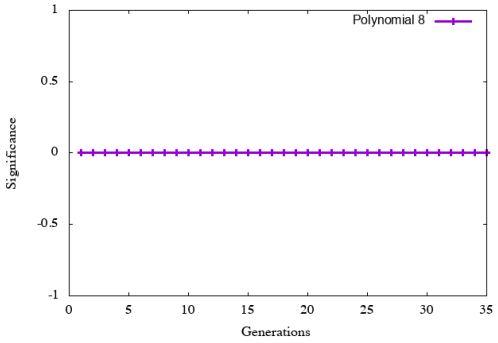


(b) T Test on Polynomial 4 $f(x) = x^7 + 5x^4 + 3x^3 + cos(x) + 600$ after each generation between DDGP and SGP

Fig. 10. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 7

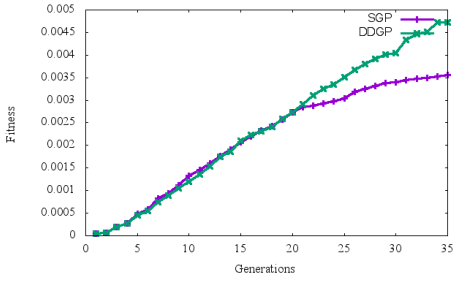


(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = x^8 + 3x^6 + 2x^4 + sin(x) + 900$ between range -1 to 1

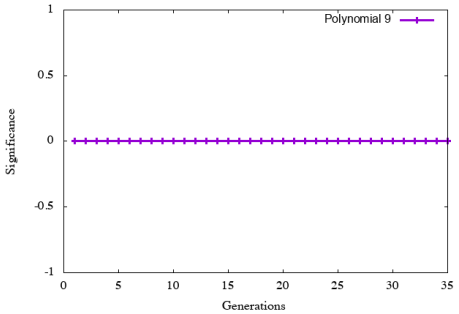


(b) T Test on Polynomial 4 $f(x) = x^8 + 3x^6 + 2x^4 + sin(x) + 900$ after each generation between DDGP and SGP

Fig. 11. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 8



(a) Average fitness of DDGP and SGP over 30 independent runs for dataset $f(x) = x^9 + 9x^7 + 5x^5 + cos(x^3) + 450$ between range -1 to 1



(b) T Test on Polynomial 4 $f(x) = x^9 + 9x^7 + 5x^5 + cos(x^3) + 450$ after each generation between DDGP and SGP

Fig. 12. DDGP and SSGP Comparison with respect to fitness and T Test for Polynomial 9

VI. Conclusion

This paper has proposed a novel approach called Dynamic Decomposition of Genetic Programming DDGP. The inspiration for this idea was dynamic programming. It has been reported several time in literature that prolong evaluation of GP cause the GP bloat or extra growth without significant improvement in fitness. Moreover, it has been also reported, when input ranges are very small and few generations are allowed. Traditional GP cannot evolve the required solution and stagnation occurs. Keeping in view, the aforementioned problems a new system is developed named DDGP to address state of the art problems.

DDGP involves the decomposition of original GP by introducing the special parameter which is known as Error change parameter. This parameter decides the stopping criteria for a sub run or sub model. By using this parameter the fitness of best individual is monitored at each generation. If fitness of best the individual does not change for x number of consecutive generations. DDGP saves all the states of currently halted run and initiate a new run. Before initializing the new run the target values are updated for new run by subtracting the values obtained by currently halted run from the desired target values. In this way, the final solution is the algebraic sum of all the successive runs. Some parameter setting is also performed when new sub runs come into an action. DDGP is tested on Maarten Keijzer's benchmarks (2003) and on eight univariate polynomials 2 to polynomial 9. The empirical results showed that DDGP outperforms the standard GP. Moreover, DDGP helps in reducing the GP bloat problem by introducing the small sub runs as decomposition. The speed and performance of DDGP is much better than standard GP.

A. Future Work and Directions

Although DDGP has been tested on numerous symbolic regression problems and all of these aforesaid problems involve single variable. DDGP has not been tested on multivariate symbolic regression problems. In future DDGP is require to evaluate on multivariate SR problems. Moreover, error change parameter is used as stopping criteria for monitoring the best individual for x consecutive generations. A certain threshold can be used as stopping criteria for instance, if the fitness of an individual is not improving from specified threshold for specific number of generations. This mechanism of using some threshold for stopping criteria is not tested and left for future work. Moreover, how DDGP will behave, when specific threshold will be used as a stopping criteria.

References

[1] Koza, John R. *Genetic programming: on the programming of computers by means of natural selection.* volume-1, publisher MIT press, 1992.

[2] Maarten Keijzer. *Improving symbolic regression with interval arithmetic and linear scaling* European Conference on Genetic Programming, 70–82, 2003

[3] Arnaldo, Ignacio and Krawiec, Krzysztof and O'Reilly, Una-May. *Multiple regression genetic programming.* Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, pages, 879–886, Publisher ACM, 2014.

[4] Langdon, William B and Poli, Riccardo. *Fitness causes bloat.* Soft Computing in Engineering Design and Manufacturing, pages 13–22, publisher Springer, 1998.

[5] Ahmed Kattan, A. and R. Poli. *Unsupervised problem decomposition using genetic programming.* European Conference on Genetic Programming,122-133, 2010.

[6] Moraglio, Alberto and Krawiec, Krzysztof and Johnson, Colin G. *Geometric semantic genetic programming* International Conference on Parallel Problem Solving from Nature,21-31, Springer, 2012.

[7] Luiz Otavio V.B, Oliveira, F. E. O. G. L. P. and J. Albinati . *Sequential symbolic regression with genetic programming.* Genetic Programming Theory and Practice XII, 73–90, Springer, 2015.

[8] Mohammad Nabi Omidvar, X. L. and Y. Meili. *Cooperative co-evolution with differential grouping for large scale optimization.* IEEE Transactions on Evolutionary Computation, 378–393, 2014.

[9] Otero, F. E. B. and C. G. Johnsoni . *Automated problem decomposition for the boolean domain with genetic programming.* 16th European Conference, EuroGP 2013, Vienna, Austria on Genetic Programming, Vol-7831, pp 169–180. Springer, 2013.

[10] Tomasz P. Pawlak, Bartosz Wieloch, K. K. *Semantic backpropagation for designing search operators in genetic programming.* IEEE Transactions on Evolutionary Computation, Vol-19, pp, 326–340, 2015.

[11] Amine, Mahdi, Brahim Hnich, Habib. *Solving constrained optimization problems by solution-based decomposition search.* Journal of Combinatorial Optimization,pp, 1–24, Springer, 2015.

[12] Garg, A and Sriram, S and Tai, K. *Empirical analysis of model selection criteria for genetic programming in modeling of time series system.* IEEE conference on computational intelligence for financial engineering & economics (CIFEr) pages, 90–94, 2013.

[13] Robyn F francon, Marc Schoenauer. *Memetic Semantic Genetic Programming* Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation GECO -2015, pp, 1023–1030, 2015.

[14] EKLUND, S. E. *A massively parallel GP engine in VLSI* Proceedings of the 2002 Congress on Evolutionary Computation CEC2002. IEEE Press ,pp, 629–633, 2002.

[15] BANZHAF, W,S. Forrest, Ed., Morgan Kaufman. *Genetic programming for pedestrians.* Proceedings ofthe5th International Conference on Genetic Algorithms,ICGA-93 (University of Illinois at Urbana-Champaign) pp, 628, 1993.

[16] POLI. R, D.B.Fogel, M. A.El-Sharkawi, X. Yao, G. Greenwood. *Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming.* University of Birmingham, School of Computer Science, Presented at 3rd International Conference on Artificial Neural Networks and Genetic Algorithms,Aug. 1996.

[17] BISHOP, C. M, D.B.Fogel, M. A.El-Sharkawi, X. Yao, G. Greenwood. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Published in Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006.

[18] CHEROWITZO, B. Lecture notes. *Pattern Recognition and Machine Learning (Information Science and Statistics).* http://www-math.cudenver. edu/wcherowi/courses/m5410/exeucalg.html

[19] CHITTY, D. M. *A data parallelapproachto geneticprogramming using programmable graphics hardware.* GECCO 07: Proceedings of he 9th annual conference on Genetic and evolutionary computation (New York, NY, USA, 2007) pp.156–1573, 2007.

[20] Medernach, David and Fitzgerald, Jeannie and Azad, R and Ryan, Conor *A New Wave: A Dynamic Approach to Genetic Programming* Proceedings of the 2016 on Genetic and Evolutionary Computation Conference,757–764, 2016. ACM

[21] LANGDON, W. B., AND POLI, R. *Foundations of Genetic Programming.* Published in Springer-Verlag, 2002.

[22] DAIDA, J. M., BERTRAM, R. R., POLITO, J. A. *Analysis of single-node (building) blocks in genetic programming.* Advances in genetic programming Journal, Published by MIT press. Vol- 3 pp. 217–241, 1999.

[23] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.