ISSN: 2229-6956(ONLINE)
DOI: 10.21917/ijsc.2013.0070

A NEW MUTATION OPERATOR IN GENETIC PROGRAMMING

Anuradha Purohit[1], Narendra S. Choudhari[2] and Aruna Tiwari[3]
[1]*Department of Computer Technology and Applications, Shri Govindram Seksaria Institute of Technology and Science, India*
E-mail: anuradhapurohit@rediffmail.com
[2, 3]*Department of Computer Engineering, Indian Institute of Technology Indore, India*
E-mail: [2]nsc183@gmail.com and [3]aruna_tiwari@rediffmail.com

Abstract
This paper proposes a new type of mutation operator, FEDS (Fitness, Elitism, Depth, and Size) mutation in genetic programming. The concept behind the new mutation operator is inspired from already introduced FEDS crossover operator to handle the problem of code bloating. FEDS mutation operates by using local elitism replacement in combination with depth limit and size of the trees to reduce bloat with a subsequent improvement in the performance of trees (program structures). We have designed a multiclass classifier for some benchmark datasets to test the performance of proposed mutation. The results show that when the initial run uses FEDS crossover and the concluding run uses FEDS mutation, then not only is the final result significantly improved but there is reduction in bloat also.

Keywords:
Bloat, Crossover, Elitism, Fitness, Mutation, Reproduction

1. INTRODUCTION

Genetic Programming (GP) [1] is an evolutionary technique used for generating computer programs based on a high level description of the problem to be solved. This innovative flexible and interesting technique has been applied to solve numerous interesting problems. Classification is one of the ways to model the problems of face recognition, speech recognition, fraud detection and knowledge extraction from databases. GP has emerged as a powerful tool for classifier evolution. Classification is a common real world activity. It is used to put entities or patterns into predefined classes. To date, many variations of GP have been introduced to handle the classification, this includes Linear GP, Grammar based GP, Graph based GP and Tree based GP. These variations differ in representations of solutions.

GP works by evolving a population of randomly created initial programs/chromosomes using a fitness measure. It selects fitter ones to take part in the evolution to efficiently search for desired efficient solution. The basic GP algorithm is similar to any evolutionary algorithms. GP chromosomes are usually trees which are manipulated by using some specific genetic operators. These are reproduction, crossover and mutation. Crossover and mutation are considered to be the main GP operators [2].

A lot of discussion has been done in GP about its operators. Some researches debate the usefulness of the crossover operator, and importance of mutation operator has been suggested. In general, GP systems use a high level of crossover, and lower levels of mutation and reproduction operators to get new solution programs of next generation. Each operator has its own importance in finding solutions for a problem using GP. These solutions or program structures are in the form of variable length strings called trees.

During the evolution of solutions/trees using GP operators, there is generally an increase in average tree size without a corresponding increase in fitness. This phenomenon is commonly referred to as bloat and hampers the performance of trees [12]. It is the uncontrolled growth of program size that may occur in GP when relying on a variable length representation. This has been identified as a key problem in GP for which there have been several empirical studies.

Mutation is an important operator for genetic programming that introduces diversity in the building blocks created during evolution and is also among the factors causing bloat in GP. Therefore it is important to study the effects of mutation on the evolutionary process. Various authors have worked on the GP operators (crossover and mutation) to handle the problem of code bloating and improving the performance of classifiers designed for different applications [3], [4], [5]-[9].

In this paper we present a special mutation operator called FEDS mutation to reduce the problem of bloat in GP and to improve the performance of program structures obtained after. In FEDS mutation, we are applying the fitness, elitism, depth limit and size on every criteria on every individual generated during mutation operation and checked whether it is capable of going to the next generation or not. We are also applying the FEDS crossover operation previously suggested to control bloat.

The paper is structured as follows: Section 2 describes the background of work already done in the field of proposed work, section 3 describes the theoretical concept of proposed FEDS mutation, and its algorithm, section 4 contains experimental results defining the datasets used, values taken for various GP parameters and results obtained by designing classifiers using proposed mutation.

2. BACKGROUND

Mutation is a mechanism to inject new genetic material into a population of solutions. It promotes diversity and improve the algorithm's ability to exploit different regions of the search space. It is applied probabilistically to the offspring generated during the crossover operation or randomly selected from the population. Mutation introduces diversity in the building blocks created during evolution by replacing subtree of an individual by an entirely new one. There are three mutational probabilities to consider when evolving decision trees using GP: firstly, the probability that a tree will be selected for mutation, secondly, the probability that a specific mutational operator will be applied to the selected tree and lastly, the probability for each node in the tree to mutate [13].

Mutation used in GP is of three types and each type is selected according to the requirement. These are as follows,

1) Point Mutation: a single node in parent tree is selected and replaced with a random node of same type. E.g. a function node is replaced by a function node of same arity and a terminal node is replaced by a randomly selected terminal node.
2) Shrink Mutation: selects a node randomly and the subtree rooted at that node is replaced by a single terminal node.
3) Grow Mutation: selects a random node and a randomly generated subtree replaces the subtree rooted at that node. Also called as Gaussian mutation or subtree mutation.

Mutation plays a very important role in getting diverse solutions for various applications. A less amount of work has been done as compared to crossover operator to apply changes in standard mutation operator to get improved mutation operator. Majeed and Ryan [9] introduced a new type of mutation, Context-Aware Mutation, which is inspired by their context-aware crossover. Context-Aware mutation operates by replacing existing sub-trees with modules from a previously constructed repository of possibly useful subtrees.

Muntean, Diosan, and [6] investigated a new variant where the best subtree is chosen to provide the solution of the problem. The other nodes (not belonging to the best subtree) are deleted. This will reduce the size of the chromosome in those cases where its best subtree is different from the entire tree. They have tested this strategy on a wide range of regression and classification problems.

R. Poli and N. F. McPheea [14] presented a new general GP schema theory for headless chicken crossover and subtree mutation. The theory gives an exact formulation for the expected number of instances of a schema at the next generation either in terms of microscopic quantities or in terms of macroscopic ones. The paper gives examples which show how the theory can be specialised to specific operators.

Alan and Terence [7], in their paper studied three structure altering mutation techniques using parametric analysis on a problem with scalable complexity. They highlighted through parameter analysis that two of the three mutation types tested exhibit nonlinear behaviour. Higher mutation rates cause a larger degree of nonlinear behaviour as measured by fitness and computational effort. Characterization of the mutation techniques using parametric analysis confirms the nonlinear behaviour. In addition, they proposed an extension to the existing parameter setting taxonomy to include commonly used structure altering mutation attributes. They showed that the proportion of mutations applied to internal nodes, instead of leaf nodes, has a significant effect on the performance.

Badran and Rockett [8] observed that genetic programming populations can collapse to all single node trees when a parsimony measure (tree node count) is used in a multiobjective setting. They investigated the circumstances under which this can occur for both the 6-parity boolean learning task and a range of benchmark machine learning problems. They concluded that mutation is an important operator and believed in a hitherto unrecognized factor in preventing population collapse in multiobjective genetic programming; without mutation any one can routinely observe population collapse. From systematic variation of the mutation operator, they concluded that a necessary condition to avoid collapse is that mutation produces, on average, an increase in tree sizes (bloating) at each generation which is then counterbalanced by the parsimony pressure applied during selection. The use of a genotype diversity preserving mechanism is ineffective at preventing population collapse.

Muni, Pal, and Das [10] proposed a new approach for designing classifiers for a c-class problem using genetic programming (GP). The proposed approach takes an integrated view of all classes when the GP evolves. A multitree representation of chromosomes is used. In this context, they proposed a modified crossover operation and a new mutation operation that reduces the destructive nature of conventional genetic operations. They used a new concept of unfitness of a tree to select trees for genetic operations. This gives more opportunity to unfit trees to become fit.

3. PROPOSED WORK

The conventional GP mutation produces the variation and diversity in tree sizes because in conventional mutation we replace the subtree of selected parent with the randomly generated subtree. Removing a smaller subtree from a tree and adding a larger subtree during mutation may create a tree of larger size, depth and having less fitness. In this way the mutated tree increases the average program size and leads to code bloat. Thus, the only way the average program size can increase during a GP run is if larger offspring are preferentially chosen during selection. Thus, the problem of bloat basically occurs due to the crossover and mutation operations. Point mutation can be used to control growth of programs during mutation but to get diversity; subtree mutation is used in GP.

We have proposed a new mutation operation which selects an individual from the population; a new subtree is generated and placed at four different positions in the selected individual. In this way, four individuals are generated from a single individual. Then we calculate the fitness, elitism, depth limit and size of the generated trees and the one having best results is transferred to the next generation. If the new trees do not have better fitness than the parent tree, then the parent/child tree will be retained to the next generation with 0.5 probability. Hence, we also give chance to the individuals which have lower fitness in mutation operation. So, by applying the proposed modified mutation, we can check the average tree size by applying all the four parameters [FEDS] on the generated individuals which helps in reducing the bloat and improving the performance of the classifier designed. Initially we have used conventional mutation operation to get variety and randomness in programs and then after some generation we have used FEDS mutation.

The Fig.1 shows the operation of new mutation. Shaded nodes from 1-8 are the possible nodes where newly generated subtree can be placed. Four new individuals are created using new mutation and the one having higher fitness, lower depth and minimum size is selected for new generation. Here in example, Tree1, Tree2, Tree3 and Tree4 are created using new mutation and Tree3 having good performance in terms of fitness, size and depth is taken to new generation.

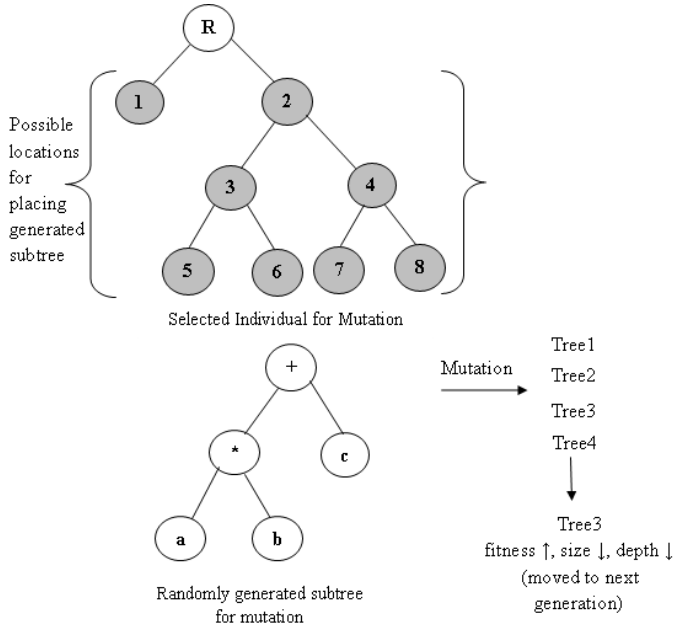

Fig.1. Proposed Mutation

Along with new mutation we have also utilized FEDS crossover [4] to improve the performance of classifier designed and to reduce the problem of code bloating that occurs during crossover and mutation operations. The algorithm of proposed mutation and the main GP algorithm utilizing proposed mutation are discussed in next section.

3.1 ALGORITHM FOR FEDS MUTATION

The steps involved in using the proposed mutation are described as follows.

Algorithm:

1) Randomly select individual from the population for mutation operation.
2) Randomly generate the subtree and place it at four different positions in the selected individual. So the total number of generated children is four.
3) Check the FEDS (Fitness, Elitism, Depth limit, Size) of all the children and the one with the greater FEDS is transferred to the next generation population.
4) If the FEDS of the generated children is less than the parent, than with the probability of 0.5 a parent or child can be retained to the next generation.

3.2 GP ALGORITHM WITH FEDS MUTATION

The complete GP algorithm with proposed mutation for designing any GP based classifier is described as follows.

Algorithm:

1) GP begins with a randomly generated population of solutions of size N.
2) A fitness value is assigned to each solution of the population.
3) A genetic operator is selected probabilistically.
   (i) If it the reproduction operator, then an individual is selected (we use fitness proportion based selection) from the current population and it is copied into the new population. Reproduction replicates the principle of natural selection and survival of the fittest.
   (ii) If it is the crossover operator, then we apply the FEDS crossover.
   (iii) If the selected operator is mutation, we apply conventional mutation for 50% generations and then FEDS mutation for concluding 50% generations.
4) Continue step 3, until the new population gets solutions. This completes one generation.
5) Steps 2 to 4 are repeated till a desired solution is achieved. Otherwise, terminate the GP operation after a predefined number of generations.

4. EXPERIMENTAL WORK

We have designed a MultiClass Classifier as an application to demonstrate the results obtained by using our new mutation operator. We have used Java 6.0 as a front end tool and MySql as a back end tool to develop our system. We have used six real data sets for training and validating our methodology. These are IRIS, WBC, BUPA, Vehicle, WDBC and Wine dataset. These datasets contains small, medium and large dimensional data. Table.1 shows number of classes and number of features present in each dataset.

Table.1. Datasets

Name of the Dataset	Number of Classes	Number of Features	Size of the Dataset
IRIS	3	4	150
WBC	2	9	683
BUPA	2	6	345
VEHICLE	4	18	846
WDBC	2	30	569
WINE	3	13	178

4.1 DATASETS

Following is the brief description of the datasets used for testing the methodology:

1) IRIS: This is the well-known Anderson's Iris data set. It contains a set of 150 measurements in four dimensions taken on Iris flowers of three different species or classes. The four features are sepal length, sepal width, petal length, and petal width. The data set contains 50 instances of each of the three classes.
2) Wisconsin Breast Cancer (WBC): This data set consists of 699 samples in 9-dimension distributed in two classes (malignant and benign).
3) BUPA Liver Disorders (BUPA): It consists of 345 data points in six dimensions distributed into two classes on liver disorders.

4) Vehicle: This data was originally gathered at the TI in 1986-87 by JP Siebert. It was partially financed by Barr and Stroud Ltd. The purpose is to classify a given silhouette as one of four types of vehicle, using a set of features extracted from the silhouette. The vehicle may be viewed from one of many different angles. This data set has 846 data points distributed in four classes. The classes are OPEL, SAAB, BUS and VAN. Each data point is represented by 18 attributes.
5) WDBC: This dataset contains observations on 569 patients with either Malignant or Benign breast tumor. Each data point consists of 30 features. Out of 569 samples, 357 belong to malignant class and remaining 212 samples belong to benign class.
6) Wine: Wine data set consists of 178 points in 13-dimension distributed in three classes. These data are the results of chemical analysis of wines grown in a particular region of Italy but derived from three different cultivators. The analysis determined the quantities of 13 constituents found in each of the three types of wine.

4.2 GP PARAMETERS

The GP parameters which we have used and are common for all the data sets are given in Table.2. These parameters are required for any GP based classifier design. To control the size of the trees during evolution, we have taken 6 as the maximum height of the tree while initialization of population. We have considered larger populations for higher dimensional data since use of a large population helps GP to evolve to a good solution without using many generations. Hence we have kept population size varying from 100-600 and number of generations varying from 2-100.

4.3 RESULTS AND DISCUSSION

We have randomly divided the samples into training set and testing set and then run GP algorithm to perform our experiments. We have repeated the experiment by changing population size, training set, testing set and keeping all other parameters constant. We have then compared the training accuracy and generalization accuracy of the classifiers obtained by using conventional crossover and mutation and FEDS mutation and crossover. The performance evaluation (average classification accuracy in %) on the test data for six data sets is summarized in Table.3. The training and generalization results show that the classifier designed using FEDS mutation outperforms the performance of conventional mutation for the datasets.

Table.2. Common Parameters for all Datasets

Parameters	Values
Probability of crossover operation, pc	0.80
Probability of reproduction operation, pr	0.05
Probability of mutation operation, pm	0.15
Total number of generations the GP evolved, M	2-100
Maximum height of a tree	6
Minimum height of a tree	2
Population Size	100-600

Table.3. Performance Evaluation of Datasets

Dataset	Conventional Mutation		FEDS Mutation	
	Training Accuracy (%)	Generalization Accuracy (%)	Training Accuracy (%)	Generalization Accuracy (%)
IRIS	89.00	87.64	94.42	93.26
WBC	85.66	83.64	95.60	94.08
BUPA	68.76	64.93	68.21	66.13
VEHICLE	58.91	56.65	68.36	68.54
WDBC	90.16	89.23	95.18	94.45
WINE	85.04	83.67	87.89	86.34

5. CONCLUSION

In this paper, we have proposed a new modified mutation operator called FEDS mutation to control the problem of bloat and to enhance the performance of classifiers designed using GP. FEDS mutation combines the concept of fitness, elitism, depth limit and tree size for generating the next generation individuals through mutation operation. To impart good effect on performance of classifiers, we have also utilized FEDS crossover operator. To demonstrate and validate our approach we have designed a multiclass classifier and presented the results on six different real datasets. To describe the usefulness of our approach, we have compared our method with the conventional method which is not considering different parameters as presented and obtained satisfactory results in terms of accuracies.

REFERENCES

[1] J.R. Koza, "*Genetic Programming: On the Programming of computers by Means of Natural Selection*", A Bradford Book, MIT Press, 1992.
[2] W. Banzhaf, P. Nordin, R. E. Keller and F.D. Francone, "*Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Application*", Morgan Kaufmann Series in Artificial Intelligence Series, 1998.
[3] Anuradha Purohit, Arpit Bhardwaj, Aruna Tiwari and Narendra S. Chaudhari, "Handling the Problem of Code Bloating to Enhance the Performance of Classifier Designed Using Genetic Programming", *Proceedings of the 5th Indian International Conference on Artificial Intelligence*, pp. 333-342, 2011.
[4] Anuradha Purohit, Arpit Bhardwaj, Aruna Tiwari and Narendra S. Chaudhari, "Removing Code Bloating in Crossover Operation in Genetic Programming", *International Conference on Recent Trends in Information Technology*, pp. 1126-1130, 2011.
[5] W. Banzhaf and W. B. Langdon, "Some Considerations on the Reason for Bloat", *Genetic Programming and Evolvable Machines*, Vol. 3, No. 1, pp. 81–91, 2002.
[6] Oana Muntean, Laura Diosan and Mihai Oltean, "Best SubTree Genetic Programming", *Proceedings of the 9th*

Annual Conference on Genetic and Evolutionary Computation, pp. 1667-1673, 2007.

[7] Alan Piszcz and Terence Soule, "Genetic Programming: Parametric Analysis of Structure Altering Mutation Techniques", *Genetic and Evolutionary Computation Conference*, pp. 220-227, 2005.

[8] Khaled M S Badran and Peter I Rockett, "The Roles of Diversity Preservation and Mutation in Preventing Population Collapse in Multiobjective Genetic Programming", *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 1551-1557, 2007.

[9] Hammad Majeed and Conor Ryan, "Context-Aware Mutation: A Modular, Context Aware Mutation Operator for Genetic Programming", *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 1651-1658, 2007.

[10] Durga Prasad Muni, Nikhil R. Pal and Jyotirmoy Das, "A Novel Approach to Design Classifiers Using Genetic Programming", *IEEE Transactions on Evolutionary Computation*, Vol. 8, No. 2, pp. 183-196, 2004.

[11] N. S. Chaudhari, Anuradha Purohit and Aruna Tiwari, "A multiclass classifier using Genetic Programming", *10th International Conference on Control, Automation, Robotics and Vision*, pp. 1884-1887, 2008.

[12] R. Poli, "A simple but theoretically- motivated method to control bloat in genetic programming", *Proceedings of the 6th European Conference on Genetic Programming*, Vol. 2610, pp. 204– 217, 2003.

[13] M. Riekert, K.M. Malan and A.P. Engelbrect, "Adaptive Genetic Programming for Dynamic Classification Problems", *IEEE Congress on Evolutionary Computation* pp. 674-681, 2009.

[14] R. Poli and N.F. McPhee, "Exact GP Schema Theory for Headless Chicken Crossover and Subtree Mutation", *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 2, pp. 1062-1069, 2001.