Adaptive Genetic Programming applied to Classification in Data Mining

Nailah Al-Madi and Simone A. Ludwig
Department of Computer Science
North Dakota State University
Fargo, ND, USA
nailah.almadi@my.ndsu.edu, simone.ludwig@ndsu.edu

***Abstract*—Classification is a data mining method that assigns items in a collection to target classes with the goal to accurately predict the target class for each item in the data. Genetic programming (GP) is one of the effective evolutionary computation techniques to solve classification problems, however, it suffers from a long run time. In addition, there are many parameters that need to be set before the GP is run. In this paper, we propose an adaptive GP that automatically determines the best parameters of a run, and executes the classification faster than standard GP. This adaptive GP has three variations. The first variant consists of an adaptive selection process ensuring that the produced solutions in the next generation are better than the solutions in the previous generation. The second variant adapts the crossover and mutation rates by modifying the probabilities ensuring that a solution with a high fitness is protected. And the third variant is an adaptive function list that automatically changes the functions used by deleting the functions that do not favorably contribute to the classification. These proposed variations were implemented and compared to the standard GP. The results show that a significant speedup can be achieved by obtaining similar classification accuracies.**

Keywords-Evolutionary Computation, Classification, Adaptive Genetic Programming.

I. Introduction

The term optimization describes a process whereby the search for the optimum solution from a set of candidate solutions is sought. Based on specific performance criteria, the optimum solution is searched for. The field of optimization is broad and has found applications in manufacturing, finance, engineering, and logistics to name a few. Before the optimization process can be started, all problems to be optimized should be formulated as a system with its status controlled by a few input variables and its performance specified by a well-defined objective function. The goal of optimization is to find the best value for each variable in order to achieve satisfactory performance. In practical terms, this means to accomplish a task in the most efficient way or the highest quality or to produce maximum yields given limited resources.

There are many different optimization techniques that exist. Evolutionary algorithms are one category of optimization techniques that are inspired by processes of biological evolution. The common idea underlying evolutionary algorithms is that given a population of individuals, natural selection (biologically referred to as survival of the fittest) is used to improve the fitness of the overall population. Given a function to be maximized, a set of candidate solutions is randomly created and the fitness function, used as a fitness measure (the higher the better), is applied. Based on this fitness measure, some of the better candidates are chosen to undergo recombination and mutation. Recombination is applied to two candidates and results in one or more new candidates; whereas mutation is only applied to one candidate and results in one new candidate. After recombination and mutation is applied a set of new candidates replace the old ones and the next generation begins. This process is iterated until a candidate with sufficient quality is found or a predefined number of iterations is reached.

There are two fundamental components in this evolutionary process that form the basis of evolutionary algorithms [1]:

- Variation by combination and mutation create the necessary diversity and;
- Selection provides the quality.

The combined effect of variation and selection leads to improving fitness values in consecutive populations.

There are various different flavors of evolutionary algorithms that follow the above procedure, and only differ to some extend. For instance, the representation of a candidate solution is often used to differentiate the different flavors. In genetic algorithms for example, strings represent the candidates over a finite alphabet; in evolutionary strategies real-valued vectors are used; for evolutionary programming finite state machines are used for the representation; and trees are used in genetic programming. A given representation might be preferred over others if it matches the problem better, i.e., the encoding of candidate solutions is easier or more natural. For instance, for evolving a computer program that can play a board game, trees are well-suited given that the parsing of the trees is necessary, thus a genetic programming approach is the best option [1].

As mentioned above, genetic programming is distinguished from other evolutionary algorithms by the use of a tree representation of variable size rather than of a linear string of fixed length representation as in genetic algorithms. The flexible representation scheme is important for the underlying structure of the data to be discovered automatically. One primary difficulty however, is that the number of solutions may become excessive without any improvement of their generalizability. However, genetic programming has proven to be a very powerful optimization technique.

Evolutionary computation is also applied to the domain of data mining. Data mining is a relatively broad field that deals with the automatic knowledge discovery from databases and it is one of the most developed fields in the area of artificial intelligence. Given the rapid growth of data collected in various realms of human activity and their potential usefulness requires efficient tools to extract and make use of the gathered knowledge. Evolutionary algorithms are very powerful tools that can be utilized to make use of knowledge hidden in the data collected [2].

This paper applies genetic programming (GP) to classification in data mining. Classification is the task of automatically categorizing data into different classes. It basically derives a classifier that distinguishes two (binary classification) or more classes (multiclass classification) in order to apply the classifier to new data that then determines the class the data instance belongs to. In particular, an adaptive GP method is introduced in this paper that allows certain parameters of the GP to be adapted according to the iterative process of the GP algorithm.

The GP process has four main steps. It starts with initializing a random population of programs, then a fitness ranking is calculated for each program using the fitness function (for the classification task, the fitness function indicates the classification accuracy). After that, a selection process chooses two programs (parents) in order to generate new programs (children). The new children are generated in the fourth step using the natural operations of crossover and mutation. All these steps are repeated until a new population with the same size than the old population is produced, as shown in Figure 1. This process is repeatedly applied until a stopping criterion, such as reaching a predefined maximum number of generations is reached, or finding a solution with the best fitness.

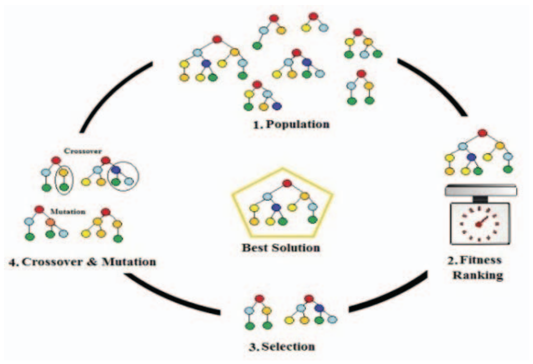


Fig. 1. GP Process

This paper is structured as follows. Section II describes related work in the area of GP including different GP variants. In Section III, our adaptive GP is described. Section IV presents the experiments as well as the results obtained, and Section V concludes this paper.

II. Related Work

Related work in the area of GP is manifold. In particular, several adaptive techniques have been applied to GP in the past. For example, in [3] the authors proposed a self-adaptive selection mechanism for genetic algorithms, which was called offspring selection. Offspring selection is based on the idea whereby the fitness value of the evenly produced offspring is compared to the fitness values of its own parents in order to decide whether or not to accept the offspring as a member for the next generation. The offspring is accepted as a candidate for the further evolutionary process if and only if the reproduction operator was able to produce an offspring that could outperform the fitness of its own parents; where the fitness of the offspring is compared with the worst fitness of the parents, and only is accepted if it is better. This mechanism was tested on some real valued test functions with a scalable degree of difficulty and found to produce better results.

However in [4], the authors used this type of offspring selection in combination with hybrid formula structures combining logical expressions and classical mathematical functions in GP, and it was applied to three medical benchmark classification problems (Wisconsin, Thyroid, and Melanoma) and compared to other machine learning methods (Linear modeling, k-nearest neighbour, artificial neural network, and standard GP). The results showed that this approach outperforms classical machine learning algorithms frequently used for solving classification problems, namely linear regression, neural networks and neighborhood-based classification.

In [5], a swarm-based improvement of the crossover operator was proposed and tested on symbolic regression. The proposed approach consists of two main parts, the first is function couplings, which constructs a matrix that represents a coupling from every function to every other function or terminal. Once this matrix is defined, crossover can be applied to preserve important couplings, where the modified crossover is applied and the probability of choosing a node as root of the sub-tree for crossover is proportional to the amount of pheromone for the coupling with its parent node. Their approach was tested on three test functions by varying the population size. Constructing a matrix and using the swarm intelligence technique is a very time consuming process, which may improve the quality of the GP process but makes the time for a run even worse, especially given that GP suffers from long execution times.

A GP method called GPTM (GP with Tree Mining) was proposed in [6]. The method first identifies the sub-trees repeatedly appearing in the chromosomes of superior individuals (ones with very high fitness), and then protects them from undesirable crossover operations. To find such sub-trees, GPTM uses a FREQT-like efficient tree-mining algorithm. GPTM was evaluated by three benchmark problems, and the results indicate that GPTM is comparable and finds the optimal individual earlier. The tree mining method is a good idea since all solutions in GP are represented as trees; however, mining every tree is computationally very expensive especially for large population sizes.

In [7], an integrated adaptive genetic algorithm was proposed, containing two adaptive techniques. The first technique dynamically adapts the rates of the crossover and mutation operators, and the second adapts the behavior of these operators whether unary (mutation), binary (crossover), or multiary. The main idea is to record the behavior of different types of these operators by using a "mini" genetic algorithm to modify the rates and the used operators. This algorithm was tested using the royal road function, and the results were compared with the standard genetic algorithm. The results show that this algorithm is more robust and less sensitive to errors with the initial parameter setting.

A tree-based crossover operator that probabilistically crosses branches based on the behavioral similarity between the branches was introduced in [8]. Node behavior is defined as the range of values that it propagates upward while the tree is evaluated; where each node records the minimum and maximum values. A distance between the two parents is calculated using these recorded values, then this distance is normalized and used as a probability of the crossover operation. This proposed technique was compared with the GP method without crossover, random crossover, and a deterministic form of the crossover operator in the symbolic regression domain, and achieved better results. In GP, usually the population size is large, and each chromosome contains nodes that represent the function list and the terminals and constants, therefore, recording the minimum and maximum values for each node adds an overhead to the whole process.

In [9], an adaptive technique for crossover and mutation probabilities in genetic algorithms was proposed. The probabilities are varied depending on the fitness of the solution, whereby higher fitness is protected, while sub-average fitness is totally disrupted. Therefore, the problem of defining the optimal values of these probabilities is solved. After selecting the parents the crossover probability is calculated depending on their fitness and the average probability of the population fitness. If the fitness is high then the resulted probability is low, and if the fitness is low then the resulted probability is high.

This paper, proposes an adaptive GP with three components. The first adaptive component concerns the selection process which is similar to the one proposed in [3, 4], however, the selection is applied not only on the children, but also on the parents. The second component implements the adaptive probabilities of the crossover and mutation processes that were suggested in [9] for genetic algorithms, and test them in GP. And the last adaptive component addresses the function list to be used in GP since it is hard to decide which functions are better to represent the solution. The adaptive function list starts with a complete list of functions, then after some generations the GP updates this list based on the functions that produce the best fitness values. Further details on the proposed adaptive GP approach are discussed in the next section

III. PROPOSED APPROACH

We propose an adaptive GP approach that consists of three adaptive variations. The first variation can be termed as the adaptive selection process (referred to as SGP from now on) and is based on ensuring that the fitness of the next generation is better than the previous one. This is done by selecting the parents based on their fitness compared to the average of the population's fitness; thereby, only parents whose fitness values are larger than the average of the population's fitness are accepted. Furthermore, the fitness of the children that are produced by the crossover of these parents is calculated for each child and is compared to the average of the old population's fitness. If larger, then this child is selected to be in the next generation. This way we ensure that the next generation's fitness is better than the previous one.

The second adaptive variation involves adaptive probabilities of the crossover and mutation operators. Since these probabilities differ based on the fitness of the solution, higher fitness has a low probability of crossover and mutation to be applied, whereas low fitness has a high probability of these operators to be applied. Keeping in mind that the mutation operator is designed to discover better variants of a single chromosome, whereas, the crossover operator combines useful genetic substructures of multiple chromosomes. These probabilities are calculated, as adopted from [9], as follows:

$$P_{\text{Crossover}} = \begin{cases} C1 * \frac{(F_{\text{Max}} - F_{\text{PMax}})}{(F_{\text{Max}} - F_{\text{Avg}})} & , F_{\text{PMax}} \geq F_{\text{Avg}} \\ C1 & , F_{\text{PMax}} < F_{\text{Avg}} \end{cases} \quad (1)$$

$$P_{\text{Mutation}} = \begin{cases} C2 * \frac{(F_{\text{Max}} - F_{\text{Ind}})}{(F_{\text{Max}} - F_{\text{Avg}})} & , F_{\text{Ind}} \geq F_{\text{Avg}} \\ C2 & , F_{\text{Ind}} < F_{\text{Avg}} \end{cases} \quad (2)$$

where C_1 and C_2 are constants and are equal to 1 and 0.5 respectively. F_{max} is the maximum fitness of the population, F_{Avg} is the average of the population's fitness, F_{PMax} is the maximum fitness of the parents, and F_{Ind} is the fitness of the individual chromosome that the mutation operation is applied to. This way, the individuals with high fitness values are protected and copied into the next generation. Using a simple equation to adapt the probabilities that do not need to record data or affect the execution time of the process is an effective way. This variation of GP is called CGP from now on.

The third adaptive variation of the proposed adaptive GP is the idea of an adaptive function list (referred to as FGP from now on). The list of functions that are used in the GP, adapts to the problem and search space. The main goal is to ease the process of selecting the functions that best represent the problem. The idea is based on modifying the function list that is used within the evolutionary process to construct programs, and an update of the function list is performed every 30 generations and after the first 100 generations, in order for the GP to be able to learn the functions that are more suitable for the problem. The number of 30 generations was determined by preliminary experiments since it achieved the best results by evaluating function lists of different generations. The modification process proceeds as follows: first, 10 best fitness programs (Best) and 10 worst fitness programs (Worst) are chosen from the population. Then, the functions used in both lists (Best and Worst) are determined, and the frequency of usage of each function is counted. As an example, the output of this step is shown in Table I.

TABLE I. Example of Adaptive Function List

Function	Worst	Best	Function	Worst	Best
Add	**2**	0	**Greater Than**	1	1
Subtract	**1**	0	**Exponential**	0	0
Multiply	0	2	**Log**	**3**	0
Divide	1	2	**Sine**	**1**	0
Power	0	0	**If Else**	0	3

Then the frequency of each function is compared; if the frequency of Worst > 0 and frequency of best = 0, then the function is deleted from the function list, and thus is not be used in the next population. Based on the example above, after this step the functions Add, Subtract, Log and Sine are deleted. At last, an update of the set of functions used in the next generations is performed.

IV. Experiments and Results

To evaluate the proposed variations and see how effective the different adaptations are to find optimized solutions and/or shorten the run time of GP, experiments were performed using the Java Genetic Algorithms Package (JGAP) [10] on two types of datasets; one with binary classes (true and false), and the other with multiple classes.

A. Datasets

The experiments are applied on the two types of datasets [12]. The binary datasets are the Wisconsin Diagnostic Breast Cancer (WDBC) dataset that predicts the two breast cancer diagnoses of benign and malignant. The Hepatitis dataset predicts whether a person lives or dies, and the Heart datasets that refers to the presence of heart disease in the patient. The Diabetes dataset is the Pima Indians Diabetes Database and the diagnostic investigated is whether the patient shows signs of diabetes or not. The details for these datasets including the number of features, and the number of records are shown in Table II.

The second type of datasets that were used are multiclass datasets that include the Dermatology dataset, which determines the type of Eryhemato-Squamous disease, the Lymph dataset, which is about the Lymphography disease, the Splice dataset, which is about the Primate splice-junction gene sequences (DNA) with associated imperfect domain theory, and the CTG dataset, which is about fetal Cardiotocograms (CTGs). This dataset is split into two outcome categories, the first one classifies a morphologic pattern (CTG-Class), and the second is related to a fetal state (CTG-NSP). All details including the number of features and records of these datasets are shown in Table III.

B. Experiments

The experiments were conducted as follows. First of all, feature selection was performed on the datasets using WEKA [11], which reduced the number of features as shown in the brackets in Table II for the binary datasets, and Table III for the multiclass datasets, respectively. Also, the GP variants were applied on these datasets using 66% for training, and 34% for testing.

After that, our GP algorithm was run using the following settings: population size = 500, number of generations = 1000, crossover probability = 0.5, mutation probability = 0.1, maximum initial depth = 5, maximum crossover depth = 8, function probability = 8, dynamize arity probability = 0.05, new chromosome percent = 0.2.

The experiments applied the standard GP (GP) and the 3 proposed adaptive GP adaptations variations; SGP, CGP, and FGP. Combinations of these adaptive ideas were implemented, such as adaptive selection and adaptive crossover were combined (referred to as SC-GP), and the combination of adaptive selection and adaptive function list (SF-GP), and also the adaptive crossover and adaptive function list (CF-GP). In addition, the combination of all three implementation ideas was done and is referred to as AGP.

C. Results

To evaluate all adaptive GPs, they are compared with standard GP measuring the accuracy and execution time. The accuracy results for applying the GP and the adaptive GPs on the binary and multiclass datasets are shown in Figure 2, where the solid point represents the average of 30 runs, the solid bar depicts the mean, the lower end of the dashed line represents the minimum accuracy observed by the corresponding GP variant, and the upper end of the dashed line depicts the maximum accuracy value.

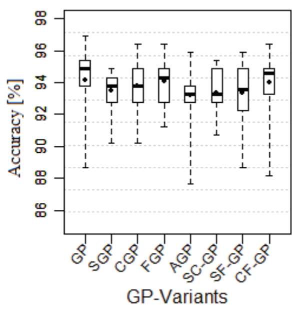
Figure 2 (a) shows the results for dataset D1, where GP, FGP and CF-GP (94.14%, 94.11% and 94.02 respectively) all have higher average accuracies compared to other GP variants. Standard GP has the highest maximum accuracy. Regarding the mean value, CF-GP has the closest value compared to the standard GP mean value.

TABLE II. Binary Datasets

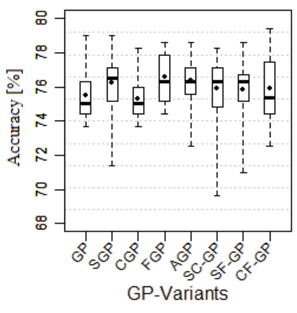
	Dataset	Classes	Features (Filtered)	Records
D1	**Breast cancer**	2	31 (11)	568
D2	**Diabetes**	2	8 (4)	768
D3	**Heart**	2	13 (9)	269
D4	**Hepatitis**	2	19 (10)	155

TABLE III. Multiclass Datasets

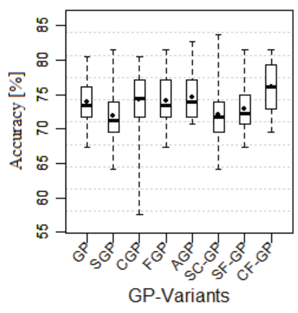
	Dataset	Classes	Features (Filtered)	Records
D5	**Lymph**	4	19 (10)	148
D6	**Splice**	3	61 (22)	3190
D7	**Dermatology**	6	33 (19)	366
D8	**CTG_NSP**	3	22 (7)	2126



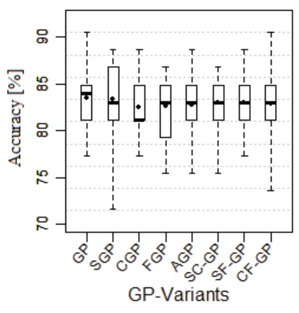
(a) D1 accuracy results



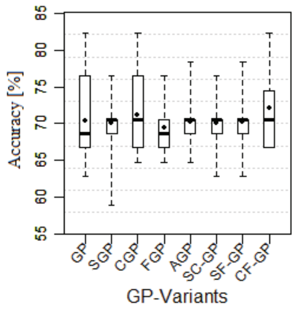
(b) D2 accuracy results



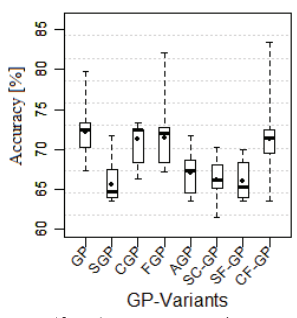
(c) D3 accuracy results



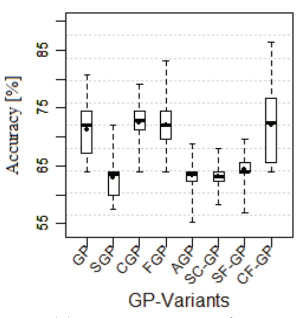
(d) D4 accuracy results



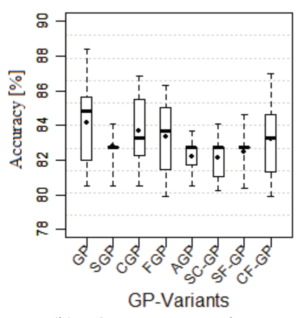
(e) D5 accuracy results



(f) D6 accuracy results



(g) D7 accuracy results



(h) D8 accuracy results

Fig. 2. Accuracy results for GP variants applied on binary and multiclass datasets

The accuracy results for D2 are shown in Figure 2 (b), where FGP has the highest average value of 76.6%, whereas standard GP scores 75.5%, however, SGP, AGP, SF-GP and CF-GP all have average accuracies higher than GP with 76.22%, 76.38%, 75.83%, and 75.89%, respectively. For the maximum accuracy values, CF-GP has the highest value. Looking at the mean, GP and CGP both have their means at the same level, but lower than all other GP variants.

D3's results shown in Figure 2 (c) reveal that the GP's average result is 73.94%, whereas CF-GP has the highest average result of 76.05%, and CGP, FGP, and AGP all have accuracies higher than GP (74.27%, 74.02% and 74.60%, respectively). Looking at the maximum values, it is clear that nearly all GP variants have higher values than GP. In addition, the mean values for GP is in the middle compared to the other GP variants, whereby FGP, AGP and CFGP all have higher mean values.

For the last binary dataset, D4, the results are shown in Figure 2(d), whereby GP's average result is 83.52%, while SGP has an accuracy of 83.33%, and SC-GP's accuracy is 83.02%. The highest maximum values are achieved by GP and CF-GP. The mean value of GP is noted to be the highest compared to other GP variants.

In summary, it was found from the binary dataset results that for the average accuracy values, FGP and CF-GP both have two results (for D2 and D3) higher than GP with one being very close (D1). SGP, CGP and SF-GP all have one higher value compared to GP (for D2, D3, and D2, respectively). CF-GP scores two highest maximum values for D2 and D4.

For the multiclass datasets, the accuracy results of D5 are shown in Figure 2 (e), where it is observed that GP's average result is 70.32%, where CGP and CF-GP both have higher accuracies with 71.17% and 72.02%, respectively. Whereas AGP, SC-GP and SF-GP all have close accuracy values (70.13%, 70.06%, and 70.13%, respectively). The highest maximum value is achieved by GP, CGP and CF-GP. The mean value for all GP variants, except FGP, is higher than GP, whereas FGP's mean is at the same level as GP's.

D6's accuracy results are shown in Figure 2 (f). GP achieves the highest accuracy with 72.13%, however, CGP, FGP and CF-GP all have close values with 71.27%, 71.46%, and 71.34%, respectively. The highest maximum values are found by FGP and CF-GP.

The accuracy results for D7 are shown in Figure 2 (g), where GP achieved an average accuracy of 71.33%, while CGP, FGP and CF-GP all have higher accuracy values with 72.37%, 71.97% and 72.05%, respectively. The highest maximum values are found by FGP and CF-GP.

The mean value of CGP and CF-GP are higher than GP. For D8, GP has the highest average accuracy of 84.14%, while CGP, FGP, and CF-GP all have close values (83.68%, 83.32% and 83.18%). In addition, the highest maximum value is also found by GP.

To summarize the results of the multiclass datasets, CGP and CF-GP both have two highest average accuracies (for D5 and D7) compared to GP. While FGP has the highest accuracy once (D7) and two close accuracy values (D6 and D8). The highest maximum accuracy is found by CF-GP for three datasets out of the four.

The average execution times for the 30 runs applied to the GP variants on the binary datasets are shown in Figure 3. For D1, SGP and CF-GP both have shorter execution times with 67.86 and 69.62 seconds, respectively, compared to GP (75.86 seconds), whereas FGP and SF-GP have close values (79.78 and 76.10 seconds). For D2, GP took on average

94.40 seconds, whereas SGP, AGP, SC-GP and SF-GP all have shorter execution times (71.68, 77.05, 75.13, 76.59 seconds, respectively). The execution time for GP applied on D3, was 40.75 seconds, whereas SGP and SC-GP both have shorter execution times with 36.82 and 40.23 seconds respectively, and AGP and SF-GP both have similar execution times (41.58 and 42.15 seconds). For the last binary dataset D4, GP had the shortest time with 22.84 seconds, followed by FGP with 25.88 seconds and CGP with 29.37 seconds.

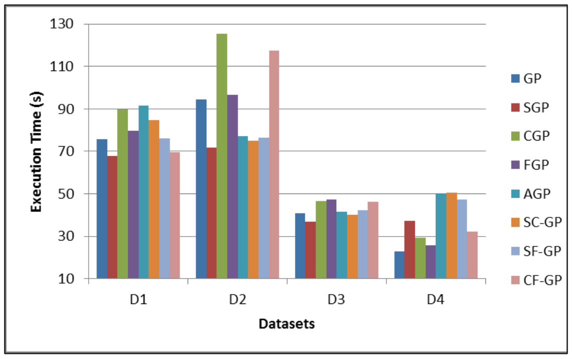


Fig. 3. Execution times for all GP variants (Binary datasets)

For the multiclass datasets the results of the execution time are shown in Figure 4 (please note that the y-axis is in logarithmic scale), whereas for D5, GP took on average 19.51 seconds to run, while FGP took 18.36 seconds, and CF-GP took 19.25 seconds. D6's execution time for GP is 555.78 seconds on average, whereas SGP, AGP, SC-GP and SF-GP all have shorter execution times than GP (392.23, 498.73, 513.01 and 446.26 seconds, respectively). For D7, GP has the shortest execution time with 47.14 seconds, closely followed by SGP and CGP with 48.24 and 49.45 seconds, respectively. GP's execution time for D8 is 213.91 seconds, whereby SGP and CGP took 203.75 and 205.14 seconds, respectively, while F-PG, SF-GP and CF-GP all have shorter execution times with 183.06, 179.43 and 178.39 seconds, respectively.

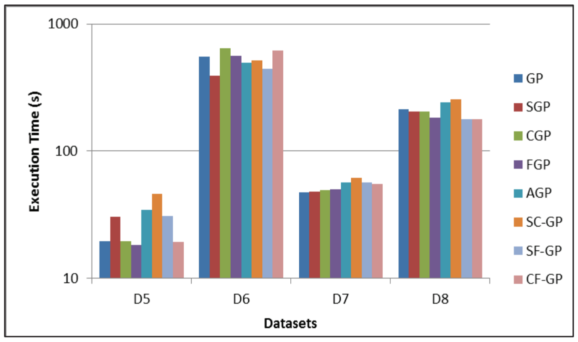


Fig. 4. Execution times for all GP variants (Multiclass datasets)

In summary, the accuracy results show that CF-GP can achieve a higher accuracy compared to GP for both binary and multiclass datasets (4 out of 8 datasets), and can obtain a close accuracy values compared to GP for the other 3 datasets, while FGP achieves a higher accuracy in three datasets, and results close to GP for other 3 datasets. Moreover, CGP can achieve a higher accuracy than GP for 3 datasets, and two datasets with a close accuracy compared to GP. While at the same time when focusing on the execution time, CF-GP has a shorter run time compared to GP for four datasets out of eight, and the same goes for FGP, whereby it has two shorter execution times compared to GP. Likewise, SGP has shorter times for 5 datasets; SC-GP and SF-GP have shorter times for 3 datasets. A possible reason for the higher accuracy of CF-GP and FGP is that the function list is being modified in order to only have the functions that suit the problem best. This focuses the process on solving the problem rather than trying many different ways (functions). In addition, for CGP the programs with high fitness are being protected from the crossover and mutation operations, while CF-GP combines these two reasons.

We note that the execution time for the FGP and all combinations have the shortest time due to the smaller function list, which affect the tree size, as mentioned in [13]: "A small tree size is a desirable outcome, particularly when an uncomplicated function set is used, as the evolved results may be easier to interpret", therefore, the tree size affects the execution time - the smaller the tree size the shorter the execution time. However, SGP ensures that the accuracy of the parents and their children are higher than the population's average fitness. High fitness values mean that the program best fits to the target problem, thus all programs in the population will fit well to the target problem and the subsequent generations programs will have higher accuracies. Furthermore, most of the generated programs with high accuracy values take less time compared to low accuracy programs, which contain a lot of functions and variables that construct long equations needing longer time to be executed on all records of the dataset.

V. CONCLUSIONS

This paper proposed a modified GP that adapts the parameters of the GP runs automatically and performs the classification task faster than standard GP.

This adaptive GP has three variations, the first considers the selection process (SGP), which is based on ensuring that the fitness of the next generation is better than the previous one; this is done by selecting the parents based on its fitness and selecting the produced children also based on its fitness compared to the average of the population's fitness.

The second variant concerns the crossover and mutation probabilities (CGP) that are adaptively changed based on the fitness of the chromosome by protecting the fittest chromosomes from these operations. The third variant is an adaptive function list (FGP), whereby the function list is adapted based on functions that performed well during the earlier classification task.

All these adaptive variations were tested individually, but also combinations were evaluated and compared with standard GP. The measures used were accuracy and execution time. Both, binary datasets and multiclass classification datasets were used. The results revealed that in particular FGP and CF-GP achieve better or similar accuracy values, by having shorter execution times. Also CGP and

FGP both help in determining the best crossover and mutation probabilities as well as the function list that best suits the problem.

Future work involves testing the proposed approach on more datasets, and implementing other adaptation techniques such as the selection based on age. To further speed up the execution time of GP, the code will be parallelized.

REFERENCES

[1] A.E. Eiben, and J.E. Smith, “Introduction to Evolutionary Computing”, 1st edition, *Natural Computing Series*, Springer, Berlin, Heidelberg, New York, 2003.

[2] Lakhmi C. Jain and Ashish Ghosh, “Evolutionary Computation in Data Mining (Studies in Fuzziness and Soft Computing)”, Springer-Verlag New York, Inc., Secaucus, NJ, 2005.

[3] M. Affenzeller and S. Wagner. “Offspring selection: A new self-adaptive selection scheme for genetic algorithms”. *Adaptive and Natural Computing Algorithms*, pp. 218-221, 2005.

[4] S. Winkler, M. Affenzeller and S. Wagner. “Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis - An empirical study”, *Proc. 8th annual Genetic and Evolutionary Computation Conf. (GECCO '06)*, Seattle, Washington, USA, 2006.

[5] T. White and A. Salehi-Abari. “A sawrm-based corssover operator for genetic programming”. *Proc. 10th annual Genetic and Evolutionary Computation Conf. (GECCO '08)*, Atlanta, Georgia, USA, 2008.

[6] Y. Kameya, J. Kumagai and Y. Kurata. “Accelerating Genetic Programming by frequent Subtree Mining”. *Proc. 10th annual Genetic and Evolutionary Computation Conf. (GECCO '08)*, Atlanta, Georgia, USA, 2008.

[7] H. Luchian and O. Gheorghies. “Integrated-Adaptive Genetic Algorithms”. *Proc. 7th European Artificial Life Conf. (ECAL)*, Germany, pp. 635–642, 2003.

[8] J. Bongard. “A Probabilistic Functional Crossover Operator for Genetic Programming”. *Proc. 12th annual Genetic and Evolutionary Computation Conf. (GECCO '10)*, Portland, Oregon, USA, 2010.

[9] M. Srinivas abd L. Patnaik. “Adaptive probabilities of corssover and mutation in genetic algorithms”. *IEEE Transactions on systems, man and cybernetics*, vol 24, no 4, April 1994.

[10] Java Genetic Algorithms Package (JGAP), source code, http://jgap.sourceforge.net, last retrieved January 2012.

[11] Ian H. Witten; Eibe Frank, Mark A. Hall. “Data Mining: Practical machine learning tools and techniques”, 3rd Edition. Morgan Kaufmann, San Francisco, USA, 2011.

[12] A. Asuncion and D. Newman, UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences, 2007.

[13] J. Fitzgerald and C. Ryan, “Drawing boundaries: using individual evolved class boundaries for binary classification problems”, Proc. 13th annual Genetic and evolutionary computation Conf. (GECCO '11), Dublin, Ireland, July 12-16, 2011.