Analysis of Genetic Programming Techniques for Unbalanced Data

Jessica Pauli de C. Bonson, Dalhousie University

***Abstract*—In this work Genetic Programming techniques are explored to improve the performance on unbalanced datasets. The initial system uses linear GP, modularity, sampling heuristics and dynamic structures. The methods analyzed are introns removal, complex instruction set, and diversity maintenance mechanisms. The results were compared using statistical methods, and were proved to improve the system performance, complexity, and runtime.**

***Keywords*—machine learning, linear genetic programming, diversity maintenance**

I. Introduction

THIS project expands the previous work on the class sandboxes to test and understand additional features for Genetic Programming (GP). The goal is to find which new features are capable of improving the performance of the system on unbalanced datasets. The modified system will be benchmarked against the results of the current system for the datasets Thyroid [1] and Shuttle [2], focusing on the later. This works assumes that all classes are equally important, so the benchmark is based on the class-wise detection rate (DR) metric. The methods tested in this work are introns removal, non-linear instructions, and diversity maintenance approaches. The next paper sections are organized as follows. Section 2 explains the current system implementation and present the benchmark. Section 3 analyzes an initial tuning on the parameters and restrictions on the runs. In Section 4 results regarding introns removal and complex instruction set are presented. Section 5 explore three diversity maintenance approaches. The final results for the datasets are shown on Section 6. Finally, Section 7 presents the conclusion and future work.

II. System Implementation and Benchmark

The system currently implemented is a GP algorithm for multi-class classification using linear representation. The system is able to evolve modular behaviors using teams of programs. Each program has an action, where an action means the class that the program will output a membership value. In a team, the program with the highest output value defines what class the team will classify the input. At each generation, a percentage of worst teams are replaced. If a program no longer participates in any team because of them were removed, then it is also removed. The system replaces the removed programs by cloning the same number of programs from the resulting population using uniform probability. There are three types of mutations for programs. The first mutation randomly selects an instruction and make a small change, e.g. using another registers as the target or using another arithmetic operation. The second and third mutations randomly add and remove an instruction. After creating the new programs, new teams are produced by cloning the remaining teams using proportional selection based on their fitness. Two mutations may affect a clone, adding or removing a program. If a program is added, it must be one of the newly created programs.

Other relevant features of the current system are: both the team size and the instruction size of the programs are variable, so the solutions structure are dynamic; a balanced sampling heuristic with oversampling is used to obtain the training data each generation, in order to ensure the solutions learn the patterns for all classes; the stop criterion is simply the end of the last generation; and the fitness function is the accuracy metric. The parameters used to run the system for the Thyroid and Shuttle datasets are shown in Table I. The parameters were chosen so that they would give a good performance in a suitable time. Due to different amount of classes to be learned, each dataset uses a different maximum of programs per team. For this configuration, the best solution across 30 runs is 0.94 for the Thyroid dataset and 0.425 for the Shuttle dataset.

Programs Population	400
Teams Population	200
Total Generations	20
Total Runs	30
Total Registers	2 (1 output and 1 extra)
Team Replacement Rate	0.2
Point Replacement Rate	1.0
Sampling Size	120
Change Instruction Mutation	0.9
Add Instruction Mutation	0.9
Remove Instruction Mutation	0.8
Add Program Mutation	0.8
Remove Program Mutation	0.7
Min. Program Size	1
Max. Program Size	30
Initial Program Size	10
Min. Team Size	2
Max. Team Size	9 (Thyroid), 14 (Shuttle)
Initial Team Size	3

TABLE I: Initial system configuration.

III. Parameters Tuning

Initially this work attempted to use the parameters from the previous section as the baseline to apply the new GP techniques. However, the system performed very poorly, and mainly due to the small number of generations the diversity maintenance methods ended up producing worse results than

the benchmark. So various modifications were made from the previous runs to attempt to improve the results. The initial implementation replaced all data points at each generation, this was modified to a replacement rate of 0.2, in order to modify the environment smoothly so the solutions would be able to continuously adapt to it. The team replacement rate was modified to 0.6, so it would evolve faster than the environment. The populations were modified from 400 programs and 200 teams to 160 programs and 80 teams, and the generations from 20 to 500. This modification intend to give the solutions more time to evolve and benefit from diversity maintenance techniques. Restrictions were also added, they ensure that all teams always have at least one action per class. Since the final objective is to obtain a detection rate of 100

The most relevant change was the modification of the generations and populations. The new best solutions are 0.97 and 0.74, respectively. The results improved with significance at p less than or equal to 0.01, using the Mann-Whitney U-Test. The modification of the point replacement rate, the balance restrictions, and the action mutation didn't significantly improved the results, but were maintained in order to ensure the best functionality of the algorithm. The final choice of parameters is shown on Table II.

Programs Population	160
Teams Population	80
Total Generations	500
Total Runs	30
Total Registers	2 (1 output and 1 extra)
Team Replacement Rate	0.6
Point Replacement Rate	0.2
Sampling Size	120
Change Instruction Mutation	0.9
Change Action Mutation	0.1
Add Instruction Mutation	0.9
Remove Instruction Mutation	0.8
Add Program Mutation	0.8
Remove Program Mutation	0.7
Min. Program Size	1
Max. Program Size	30
Initial Program Size	10
Min. Team Size	total classes
Max. Team Size	9 (Thyroid), 14 (Shuttle)
Initial Team Size	total classes

TABLE II: Improved system configuration.

IV. Non-Linear Instructions and Introns Removal

The first feature added to the system is non-linear instructions: ln, exp, cos, if ¡, and if ¿=. To work around the problem of overflow, if an overflow occurs the returned result is the variable value before the execution of the operation. The hypothesis is that non-linear instructions will result in a higher accuracy, that leads to a higher DR. A better accuracy is expected since now problems will be able to increase their complexity to match complex data patterns. The second experiment is to add introns removal to the GP algorithm, so instructions with no effect on the final output are ignored while processing the instruction set. It is expected that the runtime performance will improve, without affecting the DR performance.

The results for the combination of runs using the simple and complex set instructions, with and without introns removal for 10 runs is shown on Table III. For both cases the introns removal successfully reduced the runtime for both sets of instructions. The main difference occurs in the complex set, where the introns removal was able to reduce the runtime by almost half of the total time. Since the complex set includes the instruction 'if', that when it is detected to be an intron it is fully removed along with its instruction and the following 'if' instructions, introns removal affect the complex instructions set to a greater extent.

The analysis of the DR performance showed that for both datasets there are statistically no difference between using the simple or the complex set of instructions. This may have occurred because these datasets don't get a great benefit out of complexity. However, the interesting result is that using the complex instruction set and introns removal the system was able to maintain the same performance running on nearly half the time.

Thyroid Dataset	Avg. Runtime	Std. Dev. Runtime
Linear Instructions	985	88
Linear Instructions + Introns Removal	827	89
Complex Instructions	1016	271
Complex Instructions + Introns Removal	596	92
Shuttle Dataset	**Avg. Runtime**	**Std. Dev. Runtime**
Linear Instructions	3703	396
Linear Instructions + Introns Removal	3270	495
Complex Instructions	3741	122
Complex Instructions + Introns Removal	2083	303

TABLE III: Runtimes in seconds for intron removal and instructions sets.

V. Diversity Maintenance Methods

In this section analyzes how genotypic and behavioural diversity maintenance affects the system performance. The genotypic distance will be calculated using the method described by [3], where the distance between teams is the the ratio of active programs common to both teams. An active program is one that was selected at least once by the team to produce the output of the team. The diversity is calculated as the average of the pairwise distance between the K-Nearest Neighbors (KNN) across the genotype distance search space, with k = 8. The objective of using KNN in this context is to give a higher fitness to individuals that occupy a sparse local region in genotypic space. The fitness is calculated by the formula (1-p)*(raw fitness) + p*diversity, where p can be tuned to modify the weight of the parameters. The p value used is 0.1, selected empirically.

The behavioral diversity will be calculated using fitness sharing as defined in [4], where individuals share their fitness on the correct classified a samples. Therefore individuals are rewarded by performing well against specific samples that

other individuals are not able to classify. A second behavioral diversity maintenance method is a modified version of fitness sharing, named class-wise fitness sharing. The formula is modified so the fitness sharing is calculated separately for each class, and then averaged by the total number of classes. The goal of this modified version is to ensure that fitness sharing it prioritizing classes with more samples.

All the comparisons were performed using the Mann-Whitney U-Test for 30 runs and for p less than or equal to 0.05. For the Thyroid dataset, there was no statistically significant improvement across the diversity maintenance methods. I attempted to obtain better results by doubling the total of generations and increasing the number of runs, but the results still had no significant differences. For the Shuttle dataset both fitness sharing methods were outperformed by the default configuration, with no diversity maintenance. Also, there was no significant difference between the class-wise and traditional fitness sharing. For both the training fitness of the solutions varied around 0.01, the best one was only 0.024. It seems that the fitness sharing enforced a policy where it was nearly impossible to get a good fitness and it improved at very small steps, so the search in the landscape wasn't able to evolve. Although the bad results for fitness sharing, for the Shuttle dataset the genotype diversity maintenance method was able to perform similarly to the default configuration and produced the final best solution. It is worth noting that the genotype diversity method with complex instructions was able to outperform the one with simple instructions. So besides the previous results of statistically irrelevant differences between the simple and complex instructions set, and between the no diversity maintenance and genotype diversity maintenance, the combination of the complex set with genotype diversity actually produced an improvement in the class-wise detection rate.

Figure 1 compares de class-wise detection rate over the generations for each class for the four methods. It is clear that the fitness sharing method was not being able to stabilize over generations, probably due to problems pointed in the behavior of the fitness values. The class-wise fitness sharing plot is now being shown since it also had this problem. Regarding the default method and the genotype diversity maintenance method, the charts show that the diversity maintenance method was able to improve the population evolution so that the classes are learned more smoothly and continuosly over time.

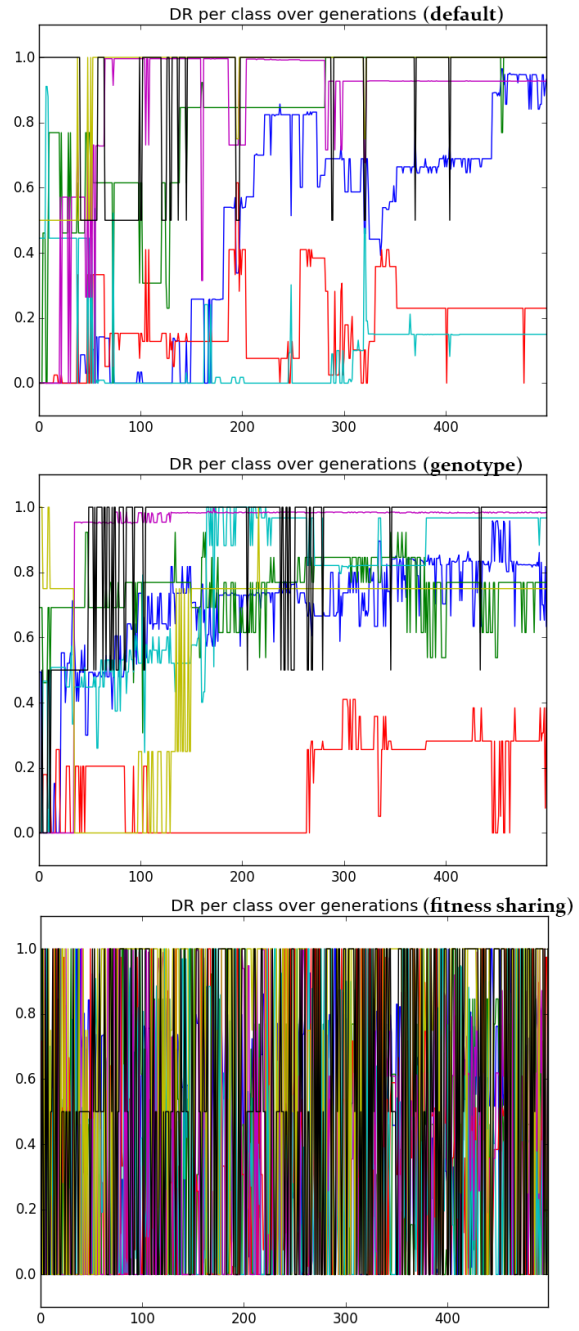

Fig. 1: Comparison of DR per class over generations.

VI. Final Result

The previous sections focused on improving the distribution of the results, this section will analyze how these improvements affected the final solution. Since the most significant differences were shown on the Shuttle dataset, that is also the more complex dataset, this will be the focus of this section. The initial best result for the Shuttle dataset was a DR of 0.42563, with a recall of 0.71, 0.54, 0.0, 0.0, 0.73, 0.0, 1.0, respectively for the classes 0 to 6. The final solution had four programs, with actions for the classes 0, 1, 4, and 6, the only ones were it was able to get a good recall. The average runtime was 82 seconds, and each program had around 10.75 instructions. After the modifications on Section 3, the system was improved and the best solution got a DR of 0.74, with recalls 0.83, 0.77, 0.33, 0.33, 0.92, 1.0, 1.0. The best team had 14 programs, with at least one for each class, four programs for class 0, two for classes 1, 2, 5 and 6. Each program had an average of 11.64 overall instruction, with around 10.21 non-intron instructions. The mean runtime was 3300 seconds. Adding the complex instructions set and intron removal on

Section 4 gave an equivalent final solution performance, but composed of only 11 programs, with a decreased mean runtime (2129 seconds), 12.36 overall instructions, and 6.54 effective instructions per program. So adding the complex instruction set reduced the overall complexity of the final team without reducing its performance. Finally, by adding the genotype diversity maintenance on Section 5, the final solution was able to obtain a DR equal to 0.784 with the cost of the runtime increasing just by 0.03. The recall per class was 0.63, 0.77, 0.38, 0.97, 0.98, 0.75, 1.0. The total programs in the final team was also 11, but now no class had more than 2 programs dedicated to it, so the recalls per class were more balanced. There was no significative difference regarding the total of instructions.

VII. Conclusion

This paper presented an investigatory analysis of GP techniques on the unbalanced datasets Thyroid and Shuttle. The methods studied were intron removal, complex instructions sets and the following diversity maintenance approaches: fitness sharing, classwise fitness sharing, and genotype diversity based on active programs in the teams. As expected, introns removal decreased the runtime of the algorithm, mainly when coupled with the complex instruction set. The fitness sharing methods didn't provide significant improvements, and it would be necessary to further study them in this domain to understand why they are not improving the algorithm performance. The genotype diversity maintenance along with the complex instructions set and introns removal sucessfully improved the solutions while decreasing their complexity and runtime. An initial work on the balance between the genotype diversity and the fitness showed to significant differences across balance values between 0.1 and 0.3, but it would be interesting to try to find the better balance between them in order to obtain the best performance improvement. Other future works would be investigate how pareto and multi-objetive methods would interact with GP for these unbalanced datasets.

References

[1] UCI, "Thyroid Disease Data Set ," https://archive.ics.uci.edu/, 1987, [Online; accessed 12-April-2015].

[2] ——, "Statlog (Shuttle) Data Set," https://archive.ics.uci.edu/, [Online; accessed 12-April-2015].

[3] S. Kelly and M. I. Heywood, "Genotypic versus behavioural diversity for teams of programs under the 4-v-3 keepaway soccer task," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[4] R. I. McKay, "Fitness sharing in genetic programming." in *GECCO*, 2000, pp. 435–442.