Sampling Methods in Genetic Programming Learners from Large Datasets: A Comparative Study

Hmida Hmida, Sana Ben Hamida, Amel Borgi, Marta Rukoz

Sampling Methods in Genetic Programming Learners from Large Datasets: A Comparative Study

Hmida Hmida[1,2], Sana Ben Hamida[2], Amel Borgi[1], and Marta Rukoz[2]

[1] Faculté des sciences de Tunis, Université Tunis El Manar, LIPAH, Tunis, Tunisia
[2] Université Paris Dauphine, PSL Research University, CNRS, UMR[7243], LAMSADE, 75016 Paris, France

Abstract. The amount of available data for data mining and knowledge discovery continue to grow very fast with the era of Big Data. Genetic Programming algorithms, that are efficient machine learning techniques, are face up to a new challenge that is to deal with the mass of the provided data. Active Sampling, already used for Active Learning, might be a good solution to improve the EA training from very big data sets. This paper present a review of sampling techniques already used with active GP learner and discuss their ability to improve the GP training from very big data sets. A method in each sampling strategy is implemented and applied on the KDD problem using very close parameters. Experimental results show that sampling methods outperforms results obtained with full dataset but some of them cannot be scaled to large datasets.

1 Introduction

Genetic Programming GP[10] is an Evolutionary algorithm considered as a universal solving problem method. This paradigm have shown its potential by reinventing previously invented patents and by creating new patentable inventions.

Applied to supervised learning and classification problems, GP generates a population of classifiers (genetic programs) by means of genetic operators. The performance of each classifier is measured by a fitness function that needs the evaluation (execution) of every program against the complete training dataset. This leads to a computational overhead increased specially with large datasets and proportional to the product of the number of instances in the dataset, the population size and the number of generations that will be carried out during the evolution process. Otherwise, using the full learning data might be impossible when the input data doesn't fit within the main memory which is often the case with Big Datasets.

The problem of reducing this calculation cost can be addressed mainly with two approaches:

- Speeding evaluation by means of hardware acceleration and parallelization.
- Reducing the number of evaluations by selecting a training subset much smaller than the entire dataset: training set sampling.

A large variety of sampling methods was investigated in many classification problems using GP and demonstrated a success comparing to the use of entire training set[3, 5, 9, 21, 12]. However, the results obtained from the earlier experiments cannot be cross-compared since they are based on different GP variants (Linear GP, Standard GP, Cartesian GP), different datasets and different GP parameters.

Large datasets are nowadays abundant, they are needed to reach an accurate classifier. Nevertheless, the increasing size heightens the calculation time problem of GP algorithms.

In this paper, we will try to put a representative set of sampling methods in very close experimental conditions, in order to show their impact on learning speed and accuracy.

2 Sampling Methods: a Review

The main objective of the studied sampling methods, in this paper, is to reduce the original training dataset size by substituting it with a representative subset much smaller and hence the evaluation cost of GP algorithm.
Two major classes of sampling techniques can be layed out : static sampling and dynamic sampling.

2.1 Static Sampling Methods

With static sampling, the learner obtains all the input training set at once and stills unmodified across the learning process. This type of sampling creates a subset of patterns with a given size from the training dataset. These patterns are selected independently from the training process in which they will be used. Some methods use a unique subset during all GP runs. Other ones can assign different subsets for each run. Hereafter, some techniques used to build this subset.

Fixed Sampling The *Historical Subset Selection* **HSS** (Gathercole and Ross 1994)[6] gathers misclassified instances from previous GP runs (seven for example) at each generation using the best individual of population. The subset contains not only difficult cases but also easy ones that are collected at earlier generations. These preliminary runs are conducted using the full dataset.

Boosting This machine learning technique was applied to GP in [9, 16] but not as a speeding method but rather to improve GP quality by evolving several subpopulations with different training sets generated by selecting the base dataset with replacement. Each training pattern has a weight reflecting his difficulty and most difficult cases may appear more often since the target size is equal to the original dataset size.

2.2 Dynamic Sampling Methods

Dynamic sampling methods are also referred to as active data selection methods as derived from *Active Learning*[2]. The underlying sampling techniques are tightly related to the learning process evolution. Generally, it depends on a particular feature like unresolved or difficult cases, the number of generations carried out, the fitness...

Random Sampling The selection of fitness cases is based on a uniform probability among the training subset. This stochastic selection helps to reduce any bias within the full dataset on evolution. In *Random Subset Selection* **RSS**[6], at each generation g, the probability of selecting any case i is equal to $P_i(g)$ such that :

$$\forall i : 1 \leq i \leq T, \quad P_i(g) = \frac{S}{T}. \tag{1}$$

where T is the size of the full dataset and S is the target subset size. The sampled subset have a fluctuating size around S. Fixed Random Selection (**FRS**) [21] is a very similar method with a fixed number of cases selected at every generation. *Stochastic Sampling* **SS** (Nordin and Banzhaf 1997)[15] is another method using the same probabilistic selection to construct subsets for each individual per generation.

The single parameter of this category of sampling is the subset size, whether a crisp or flexible target, is set like other GP parameters (population size, crossover probability, ...) and most likely by several GP runs.

Weighted Sampling. The selection of fitness cases that will be added to the subset is based on a calculated weight which is a non uniform probability that measures how much a pattern is worthy and can help to sharpen the population quality. It is highly inspired by boosting technique in machine learning field, originally used to improve accuracy of weak learners.
The very first algorithm in this category is *Dynamic Subset Selection* **DSS** [6, 7, 5]. This algorithm is intended to preserve training set consistency while alleviating its size by keeping only the *d*ifficult cases with ones not selected for several generations. Each dataset record is assigned a difficulty degree $D_i(g)$ and an age $A_i(g)$ starting with 0 at first generation and updated at every generation. The difficulty is incremented per one individual misclassification and reset to 0 if the fitness case is selected. The age is equal to the number of generations since last selection, so it is incremented when the fitness case has not been selected and reset to 0 otherwise.
The resulting weight W of the i^{th} case is calculated with this sum:

$$\forall i : 1 \leq i \leq T, \quad W_i(g) = D_i(g)^d + A_i(g)^a. \tag{2}$$

where d is the difficulty exponent and a is the age exponent. The selection probability is biased by fitness case difficulty and age:

$$\forall i : 1 \leq i \leq T, \quad P_i(g) = \frac{P_i(g) * S}{\sum_{j=1}^{T} W_j(g)}. \tag{3}$$

DSS needs three parameters to be tuned: difficulty exponent, age exponent and target size. Gathercole [6] tried different subset sizes, and different population sizes with The thyroid classification problem [3] using a full training set size of 3772 . With 10000 individuals DSS realizes better results than neural networks. DSS reduced evaluations by 20% whilst obtained similar classifiers.

Hierarchical Sampling. This kind of sampling is based on multiple levels of sampling methods inspired by the concept of a memory hierarchy. It combines several sampling algorithms that are applied in different levels.

RSS-DSS and DSS-DSS. Robert Curry and Malcom Heywood conceived an extension to DSS algorithm into a 3 levels hierarchy[4]. At level 0, the dataset was first partitioned into blocks that were sufficiently small to reside within RAM alone. Then, at level 1, blocks were then chosen from this partition based on RSS or DSS. Finally, at level 2, the selected block is considered as the full datset on which DSS was applied for several rounds. Depending on the level 1 algorithm, we have **RSS-DSS** hierarchy or **DSS-DSS** hierarchy. Besides DSS parameters(see Section 2.2), new parameters are added: level 0 block size, level 1 number of iterations, level 2 iterations, maximum level 2 iterations and tournament iterations. Only level 2 iterations is calculated by:

$$I_b(i) = I_{max} * E_b(i-1). \tag{4}$$

$I_b(i)$: number of level 2 iterations on block b in i_{th} level 1 iteration.
I_{max}: maximum level 2 iterations.
$E_b(i-1)$: error rate over block b performed by the best case at previous iteration.
A modified DSS in level 2 is used where two roulette wheels exist per block, one is used to control the selection of patterns with respect to age and the other difficulty, the roulette wheels being selected in proportion to the corresponding probability for age and difficulty (2 additional parameters).
For the DSS-DSS hierarchy, block weight and block selection probability are defined by Equation 5 to be used in level 1.

$$[!ht]Block(i)_{weight} = \frac{\%diff * Block_{diff}(i)}{\sum_j Block_{diff}(j)} + \frac{\%age * Block_{age}(i)}{\sum_j Block_{age}(j)}. \tag{5}$$

$$P(Block(i)) = \frac{Block(i)_{weight}}{\sum_j Block()_{weight}}.$$

Where %diff and %age are difficulty and age weighting used by the two roulette wheels; $Block_{diff}(i)$ and $Block_{age}(i)$ are the respective block difficulty and age for block i[4].
Tested against KDD'99[18] and Adult dataset[4], both algorithms realize competitive results in very shorter time. DSS-DSS outperforms RSS-DSS in these experiments.

[3] See UCI Machine Learning Repository at http://archive.ics.uci.edu/ml/

BB-DSS. An extension to this work is made in [3] with the *Balanced Block DSS*by altering mainly the level 0 block selection with the goal of obtaining a balanced block in level 1. A balanced block which does not reflect the original datset classes' distribution but a fixed ratio for each class regardless of their original ratios. At level 0, the full dataset is sorted and divided into separate class partitions. To create a level 1 block, one partition per class will be used. The size of class partition depends on the level 1 block size and the class ratio. For a block size equal to 1000 with 2 classes having respectively 25% and 75% ratios, class partition sizes will be respectively 250 and 750. To apply DSS on partitions, partition difficulty and age are defined to calculate partition weight and selection probability[3]. BB-DSS was compared to RSS-DSS and CasGP[4] against different datasets : Adult, Census, Shuttle and KDD'99[3]. Experimental results showed that Balanced Block algorithm is able to approach the classification performance of the CasGP algorithm, whilst retaining the computational speedup of the original RSS-DSS algorithm.

Incremental Sampling. The training subset starts with a few cases and acquires more cases at every generation to reach the full subset size.
Zhang [21, 20] proposes to perform a uniform data crossover simultaneously with genetic programs evolution. Data crossover means that when crossover operation is applied to genetic programs to produce new programs, their subsets are crossed to obtain new offspring subsets inducing data inheritance through generations. This helps to preserve the knowledge acquired by the parents. This method called *Incremental Data Inheritance* **IDI** starts with n_0 sized subsets, and increases by λ at every generation with respect to an import rate depending on a third parameter ρ. In addition to that Zhang uses an *Adaptive Fitness Function* varying from an individual to another[21]. This method is applied to the evolution of collective behaviors for multiple robotic agents[20] and was compared to the standard GP and GP with Incremental Random Selection (**IRS**) [20] with 100 training cases. In a second study [21], IDI was faced to full dataset and FRS in a context of time series prediction problem.

Experimental evidence supports that evolving programs on an incrementally selected subset of fitness cases can significantly reduce the fitness evaluation time without sacrificing generalization accuracy of the evolved programs.

Topology Based Sampling. Inspired by the idea that structure influences the efficiency of heuristics working on it [17, 19], this method consists at building a topology of the problem from the knowledge acquired by individuals about fitness cases. Lazarczyk [12] suggests a *Fitness case Topology Based Sampling* **TBS** in which relationship between fitness cases in the training set is represented by an undirected weighted graph. Vertices are fitness cases and edges have a weight measuring a similarity or a distance induced from individuals performance. Then,

[4] Cascaded GP: uses the RSS-DSS algorithm to provide the basis for building cascades of GP classifiers, up to a predefined number of layers

cases having a tight relationship with respect to a threshold cannot be selected together in the same subset assuming that they have an equivalent difficulty for the population. Edge values start at 0 and are updated after evaluation phase by increasing the weight of all edges between solved cases by each individual and decreasing the weight of all edges by a loss rate λ.
This algorithm uses a unique subset for all the individuals renewed each generation after updating the topology graph. The TBS has the following steps:

1. Empty the set of selected fitness cases and initialize the candidate set with all fitness cases.
2. while there are more candidate fitness cases and the target subset size not reached do:
   (a) Select randomly a fitness case into the subset from the set of candidate fitness cases
   (b) remove the selected case and all fitness cases connected to it with an edge weight exceeding the threshold.

The threshold is calculated during evolution process to have a value adapted to the current topology. The threshold must not be too high or too low. In the first case, there is no exclusion and TBS is equivalent to Stochastic Sampling. In the second case, very few cases are selected and the target size could not be reached. Threshold is calcultaed using a binary-search type algorithm.

Experimental results on classification problems (intertwined spirals [11] and the Thyroid problems) demonstrated improvements in mean fitness value through generations compared to DSS and SSS. However, the additional computational cost of TBS caused by the threshold calculation algorithm and the topology update has not been measured. Moreover, this method has not been tested on a large dataset.

Other Sampling Techniques. *Rational Allocation of Trials* **RAT** is an algorithm that allocates fitness cases only to those individuals for which the cost of evaluating another fitness case is out weighed by the expected utility that the new information will provide. Although, it is a proven speed up mechanism and leads to a reduced and varying number of fitness cases per individual, it's not a training subset selection method -in my opinion- since it does not select cases according to a certain probability but use a minimal set of fitness cases for evaluation and then decides for each individual whether to carry on evaluation with the next fitness case according to the probability that they might win some tournament that they are losing or lose some tournament they are winning.

Balanced sampling[8] is a method aiming to improve classifiers accuracy by correcting the original dataset imbalance within majority and minority class instances. It has some methods based on the minority class size and thus reduce the number of instances like the methods studied in this paper. The following sampling methods are used with GP:

– **Static Balanced Sampling:** selects cases with uniform probability from each class without replacement until obtaining a balanced subset with equal

number of majority and minority class instances of the desired size at every generation.
- **Basic Under-sampling:** select all minority class instances and then an equal number from majority class randomly.
- **Basic Over-sampling:** select all majority class instances and then an equal number from minority class randomly with replacement.
- **Under-sampling A/B:** Multiple balanced samples are created at each generation with Basic under-sampling method. At every generation, all individuals are evaluated against all of the sample sets and are attributed the average fitness across all the sample sets (and the minimum fitness for version B).
- **Over-sampling A/B:** Creates several balanced samples using Basic Over-sampling. The final fitness is the average on all subsets (and the minimum fitness for version B).

3 Design Issues for the Comparative Study

3.1 Cartesian GP.

CGP is a form of GP, proposed by Julian Miller[14], that represent genetic programs as directed acyclic graphs and mostly inspired by digital circuits called FPGA. In this graph, nodes representing functions many inputs and one output, and are layed into two dimensional matrix very similar to neural networks. This form of GP has the following advantages:

- Unlike trees, it allows more than one path between any pair of nodes permitting the reuse of previous results.
- Highly competitive with other GP methods.
- Does not bloat.
- Is easy to implement.
- Can have multiple outputs and then solve many types of problems (eg. classification problems).

CGP parameters. The implementation of CGP involves different parameters that determine its efficiency. A common approach in tuning GP is to undertake a series of trials to make parameter choices for the final GP runs. Since the main objective is to compare sampling methods in closely context, this tuning procedure is not fully investigated.s The final design of CGP parameters used in this work is summarized in Table 1.

3.2 Implemented Sampling Methods

RSS and DSS were implemented identically to their author's description. The remaining methods have been altered as described hereafter.

BRSS. This method is Balanced RSS in which the subset generated by RSS reflects the original dataset classes' frequencies. The number of each class patterns is calculated with respect to the target size.

Table 1. CGP parameters.

Parameter	Value
Population size	512 for RSS, DSS, BUSS BRSS and RSS-DSS 128 for IDS, TBS and Full Subset
Subpopuations number	1
Number of generations	500 for RSS, DSS, BUSS and BRSS 100 for IDS, TBS depending on the RSS iteration for RSS-DSS
CGP nodes	300
Inputs/Outputs	49/1
Tournament size	4
Crossover/Mutation probability	0.9/0.04
Fitness	Minimize classification errors

RSS-DSS. The implemented RSS-DSS here does not use the same fitness as proposed by authors. It uses the same fitness function as the other implemented methods. The effect of this transformation will discussed in Section 5. Two configurations was tested as shown on the Table 4 changing target size, RSS and DSS iterations simultaneously. The second configuration is referred to with RSS-DSS2.

IDI. Since we use a fixed subset grow increment, the full dataset size is not reached at the last generation.

TBS. We calculated the threshold applying a more simpler steps:

1. Use the lowest edge weight as the initial threshold to have the smallest possible subset.
2. Repeat until target size reached.
   (a) Apply TBS selection using current threshold on the remaining patterns.
   (b) remove the selected patterns.
   (c) If the subset size is too small, the new threshold value is the next edge weight and the previsouly excluded patterns are added to the candidate ones.

BUSS. KDD-99 have a normal class and 4 attack classes. The subset selection uses the U2R attack (52 patterns) class as minority class and then randomly selects an equal number of patterns from the remaining 3 attacks to obtain 208 attack patterns. In a first experiment, 52 normal were added obtaining a subset of 260 patterns. In the second experiment (referred to as BUSS2), 208 normal patterns were chosen to get a final subset of 416 patterns. Sampled subsets are renewed each generation.

3.3 Performance Metrics

By the end of each run, the best individual based on the fitness function is tested on the whole dataset and then the test dataset. Results are recorded in a confusion matrix from which accuracy, True Positive Rate (TPR) and False Positive Rate (FPR) are calculated.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ patterns}. \tag{6}$$

$$TPR = \frac{True\ Positives}{True\ Positives + False\ Negatives}. \tag{7}$$

$$FPR = \frac{False\ Positives}{False\ Positives + True\ Negatives}. \tag{8}$$

The training time is the amount of elapsed seconds between the first and the last generation (see Table 1 for the number of generations).

4 Experimental Design

4.1 Evolutionary Framework

ECJ. Among several evolutionary computation frameworks, Sean Luke's ECJ [13] was used in this work to implement and test the sampling algorithms. ECJ implements evolutionary algorithms like GP, Evolution Strategies, Genetic Algorithms... It's an open source framework written in Java, offering the most complete range of GP representations (Koza, linear, grammatical, ...). This framework provides a very flexible API using parameter files well documented in the ECJ owner's manual.

ECJ is an active project at George Mason University's Evolutionary Computation Laboratory having frequent releases[5] and benefit of many contribution packages as the one used here for implementing Cartesian GP developed by David Oranchak[1].

4.2 Learning Data

We used here a large dataset, which was firstly created for the competition held at the Fifth International Conference on Knowledge Discovery and Data Mining KDD-99 and is about intrusion detection problem. The original training dataset contains 5 million lines conveying data about connections from the simulated traffic and their labels. Another set was called corrected set, constructed with the same way is used as test dataset. The training process is run on the derived 10% KDD-99 dataset to learn how to classify normal connections and attacks. The best individual of each run is then tested on the test set. Both datasets are presented in Table 2.

4.3 GP Design

The EA used for the experimental study is the Cartesian GP (CGP). With CGP, programs are coded with integer linear chromosome divided into groups. Each group corresponds to a position in a 2-D array.

[5] The last release is ECJ23 on June 15^{th} 2015.

Table 2. KDD-99 datasets composition.

Class	**Number of Patterns**	
	10% Training Set	*Test Set*
Normal	97278	60593
Dos	391458	229853
probe	4107	4166
R2L	1126	16347
U2R	52	70
Total Attacks	396743	250436
Total Patterns	494021	311029

Terminal and function sets. The terminal set includes input variables which are the 41 KDD-99 features set with 8 randomly generated constants.
The function set includes basic arithmetic, comparison and logical operators reaching 17 functions.

Table 3. Terminal and function sets for GP.

Function (node) set	
Arithmetic operators:	$+, -, *, \%$
Comparison operators:	$<, >, <=, >=, =$
Logic operators:	AND, OR, NOT, NOR, NAND
Other :	NEGATE, IF (IF THEN ELSE),
	IFLEZE(IF <=0 THEN ELSE)
Terminal set	
KDD-99 Features	41
Ephemeral Random Constants	8 in $[-2, 2[$

4.4 Specific Parameters for Sampling Methods

The values of additional parameters brought by each sampling method are assigned with respect to original method paper. Some of theses methods have been modified in a order to fit in a comparable context. The Table 4 below shows each method configuration.

5 Experimental Results

Experiments are performed on an Intel i7-4810MQ (2.8GHZ) workstation with 8 GB RAM running under Windows 8.1 64-bit Operating System. GP programs are trained to distinguish between normal and attack patterns. To compare the performance of the implemented sampling methods, six measures are recorded cross the 21 runs performed for each technique: the best and the mean values for the accuracy, TPR and FPR metrics. The best measures are illustrated by the Figure 3 whereas the mean values are presented in the figures 1 and 2. Otherwise, to compare the methods according to the computational cost, the spent time to complete each run and to find the best of run individual for all the trials are

Table 4. Specific methods' parameters.

Method	Parameter	Value
Standard CGP	-	-
RSS	Target Size	5000
BRSS	Target Size	5000
	Balancing method	Full dataset distibution
DSS	Target Size	5000
	Difficulty/Age exponent	1/3.5
RSS-DSS	Target Size(level 2 block size)	2500 then 100
	Level 0 block size	5000
	RSS iterations	200 then 500
	Max DSS iterations	20 then 50
	Individuals evaluated per DSS iteration	100 (20% of population size)
	Difficulty/Age exponent	1/3.5
	Difficulty/Age Roulette	70%/30%
TBS	Target Size	1000
	Loss Rate	0.7
	Iterations	see paragraph below
IDI	Initial Subset Size	1000
	Increment	10
	Diversity Factor	0.3
Basic Under (BUSS)	-	-

recorded. Figure 4 illustrates the corresponding mean values for all implemented techniques. Experimental results show that introducing a sampling method to

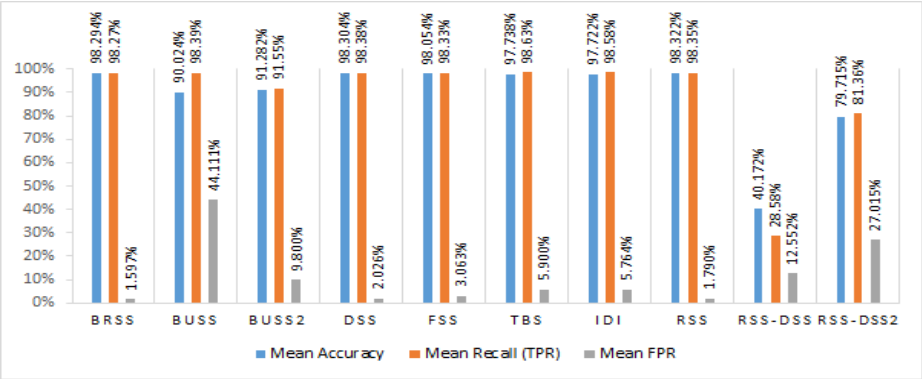

Fig. 1. Mean Data Set Performance Metrics

the GP system improves its performance when training from large data sets. All the implemented sampling methods have better results than the conventional

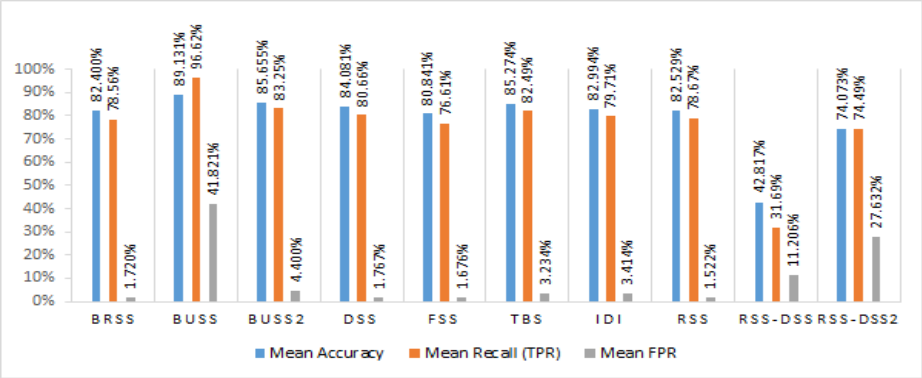

Fig. 2. Mean Test Set Performance Metrics

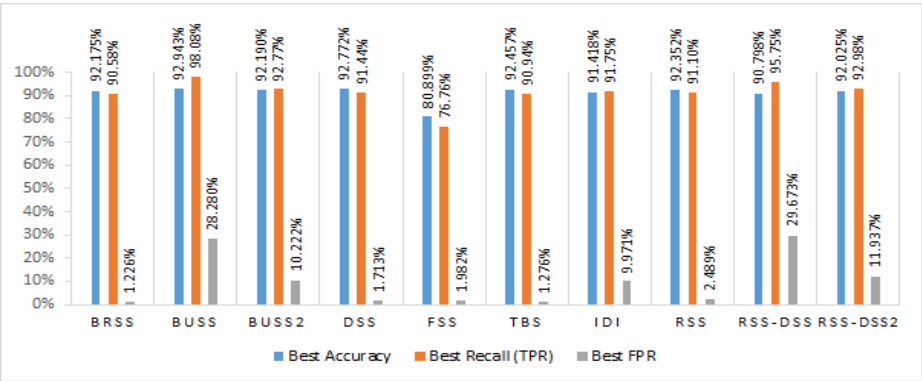

Fig. 3. Best Individual against Test dataset

GP trained on the whole data set (FSS), and this is according to the three performance metrics (Figure 2.

From Figures 1 and 2, we can make the following observations. Except for the basic version of RSS-DSS technique, all the implemented sampling techniques have given satisfactory results. As shown in the Figure 3, the accuracy and the True Positive Rate of the best solutions given by all the methods are quite similar. However, this finding is not available for the best FPR values. Indeed, according to the Accuracy and TPR metrics of the best solutions, BUSS method performed better than all the other sampling approach, but it has the 2^{nd} worst TPR value (after the RSS-DSS method).

For the two methods BUSS and RSS-DSS, we tried a second configuration mentioned as BUSS2 and RSS-DSS2 (see Section 3.2). BUSS2, while using a larger subset (more normal cases than BUSS), has lost accuracy and TPR but realizes a much better FPR. With RSS-DSS2 settings, this method enhances its performance but FPR remains very high.

TBS, IDI and FSS have been tested with a reduced number of generations, population size, and target subset size due to the huge amount of time needed for each generation. Nevertheless, TBS and IDI reached a comparable performance level. All 3 methods have a very increased run time. Moreover, both IDI and TBS spent more time than using the full dataset assuming that the computing time needed for training set sampling is very high compared with the evaluation time.

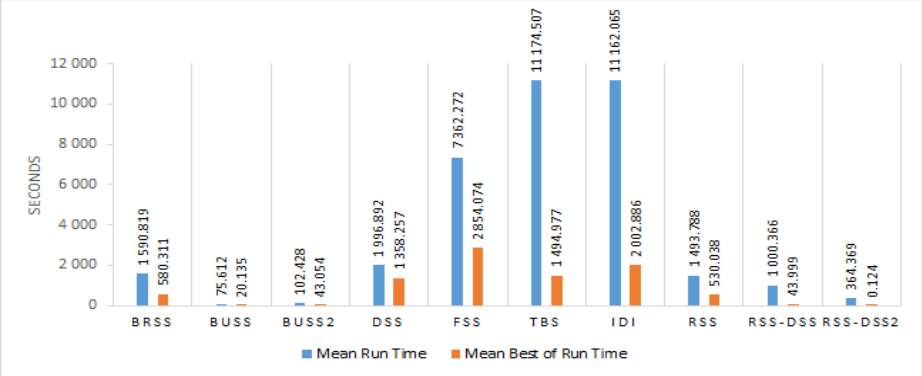

Fig. 4. Time Measures

From Figure 4, we can see that RSS-DSS in both settings finds the best individual of run in the very first generations. BUSS makes the fastest runs because it uses the smallest subset size but this result depends on the classification problem itself and classes of the learning data.

6 Conclusion

The aim of this study was to compare dynamic sampling methods and their behavior when faced with large datasets under fixed parameters. Those parameters (CGP parameters) were not tuned for each sampling method.

IDI and TBS, even though they had shown good results with smaller training sets, they suffer from a very high computing time and cannot be applied to large datasets.

RSS-DSS, in this experimental context has lost its performance. The main change with earlier experiments is the use of a different fitness function.

To deal with run time problem of IDI and TBS, mixing them with other sampling methods in a hierarchical sampling is a promising solution. Fitness function affects the quality of GP results and this needs to be experimented on the sampling methods tested here.

References

1. CGP: Cartesian gp website, `http://www.cartesiangp.co.uk`

2. Cohn, D., Atlas, L.E., Ladner, R., Waibel, A.: Improving generalization with active learning. In: Machine Learning. pp. 201–221 (1994)
3. Curry, R., Lichodzijewski, P., Heywood, M.I.: Scaling genetic programming to large datasets using hierarchical dynamic subset selection. IEEE Transactions on Systems, Man, and Cybernetics: Part B - Cybernetics 37(4), 1065–1073 (2007)
4. Curry, R.C., Heywood, M.: Towards efficient training on large datasets for genetic programming. Lecture Notes in Computer Science 866(Advances in Artificial Intelligence), 161–174 (2004)
5. Gathercole, C.: An Investigation of Supervised Learning in Genetic Programming. Thesis, University of Edinburgh (1998)
6. Gathercole, C., Ross, P.: Dynamic training subset selection for supervised learning in genetic programming. In: Parallel Problem Solving from Nature - PPSN III. Lecture Notes in Computer Science, vol. 866, pp. 312–321. Springer (1994)
7. Gathercole, C., Ross, P.: Small populations over many generations can beat large populations over few generations in genetic programming. In: Genetic Programming 1997: Proc. of the Second Annual Conf. pp. 111–118. Morgan Kaufmann, San Francisco, CA (1997)
8. Hunt, R., Johnston, M., Browne, W.N., Zhang, M.: Sampling methods in genetic programming for classification with unbalanced data. In: Australasian Conference on Artificial Intelligence. Lecture Notes in Computer Science, vol. 6464, pp. 273–282. Springer (2010)
9. Iba, H.: Bagging, boosting, and bloating in genetic programming. In: Proc. of the Genetic and Evolutionary Computation Conf. GECCO-99. pp. 1053–1060. Morgan Kaufmann, San Francisco, CA (1999)
10. Koza, J.R.: Genetic programming: On the programming of computers by means of natural selection. Statistics and Computing 4(2), 87–112 (1994)
11. Lang, K.J., Witbrock, M.J.: Learning to tell two spirals apart. In: Proceedings of the 1988 Connectionist Models Summer School, pp. 52–59. Morgan Kaufmann (1988)
12. Lasarczyk, C.W.G., Dittrich, P., Banzhaf, W.: Dynamic subset selection based on a fitness case topology. Evolutionary Computation 12(2), 223–242 (2004)
13. Luke, S.: Ecj, a java-based evolutionary computation research system, `http://cs.gmu.edu/~eclab/projects/ecj/`
14. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Proceedings of the Third European Conference on Genetic Programming (EuroGP-2000). LNCS, vol. 1802, pp. 121–132. Springer Verlag, Edinburgh, Scotland (2000)
15. Nordin, P., Banzhaf, W.: An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. Adaptive Behaviour 5(2), 107–140 (1997)
16. Paris, G., Robilliard, D., Fonlupt, C.: Artificial Evolution: 5th International Conference, Evolution Artificielle, EA 2001 Le Creusot, France, October 29–31, 2001 Selected Papers, chap. Applying Boosting Techniques to Genetic Programming, pp. 267–278. Springer, Berlin, Heidelberg (2002)
17. Tad Hogg, T.: Refining the phase transition in combinatorial search. Artif. Intell. 81(1-2), 127–154 (1996)
18. UCI: Kdd cup (1999), `http://archive.ics.uci.edu/ml/machine-learning-databases/kddcup99-mld/kddcup99.html`
19. Walsh, T.: Search in a small world. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages. pp. 1172–1177 (1999)

20. Zhang, B.T., Cho, D.Y.: Genetic Programming with Active Data Selection, vol. 1585, chap. Simulated Evolution and Learning, pp. 146–153. Springer, Berlin, Heidelberg (1999)
21. Zhang, B.T., Joung, J.G.: Genetic programming with incremental data inheritance. In: Proceedings of the Genetic and Evolutionary Computation Conference. vol. 2, pp. 1217–1224. Morgan Kaufmann, Orlando, Florida, USA (13-17 July 1999)