A new adaptive sampling approach for Genetic Programming

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

A new adaptive sampling approach for Genetic Programming

Hmida Hmida
Université de Tunis El Manar, Faculté des Sciences de Tunis,
LR11ES14 LIPAH, 2092,
Tunis, Tunisie
hhmida@gmail.com

Sana Ben Hamida
Université Paris Dauphine, PSL Research University,
CNRS, UMR[7243], LAMSADE,
75016 Paris, France
sana.mrabet@dauphine.psl.eu

Amel Borgi
Université de Tunis El Manar, Faculté des Sciences de Tunis,
LR11ES14 LIPAH, 2092,
Tunis, Tunisie
amel.borgi@insat.rnu.tn

Marta Rukoz
Université Paris Dauphine, PSL Research University,
CNRS, UMR[7243], LAMSADE,
75016 Paris, France
marta.rukoz@lamsade.dauphine.fr

***Abstract*—Genetic Programming (GP) is afflicted by an excessive computation time that is more exacerbated with data intensive problems. This issue has been addressed with different approaches such as sampling techniques or distributed implementations. In this paper, we focus on dynamic sampling algorithms that mostly give to GP learner a new sample each generation. In so doing, individuals do not have enough time to extract the hidden knowledge. We propose adaptive sampling which is half-way between static and dynamic methods. It is a flexible approach applicable to any dynamic sampling. We implemented some variants based on controlling re-sampling frequency that we experimented to solve KDD intrusion detection problem with GP. The experimental study demonstrates how it preserves the power of dynamic sampling with possible improvements in learning time and quality for some sampling algorithms. This work opens many new relevant extension paths.**

***Index Terms*—Genetic Programming, Machine Learning, Training set Sampling, adaptive sampling, sampling frequency control, KDD intrusion detection.**

I. Introduction

Evolutionary Algorithms (EA) [1]–[3] are meta-heuristics that complies with a wide range of problems and especially those related to optimization and machine learning. Their flexibility and expressiveness comes with two major flaws: an excessive computational cost and a problematic parameters setting. Genetic Programming (GP) [4] is the most affected EA by these drawbacks.

In supervised learning field, the lack of data leads to unsatisfactory learners. This is no longer an issue with numerous data sources and high data volume that we witness in the era of Big Data. Nonetheless, this toughens up the computation problem of GP and precludes its application in data intensive problems.

Many techniques have been deployed to tackle the computation time issue that can be grouped in two main categories: hardware acceleration based techniques and software based techniques. The most affordable is software based solutions that do not require any specific hardware configuration. Sampling is the mainstream approach in this category. It relies on reducing processing time by reducing data while keeping relevant records.

In [5], we exposed a review of sampling methods used with GP. Mainly, sampling methods can be classified with regard to three properties: re-sampling frequency, sampling scheme or strategy and sampling quantity. According to re-sampling frequency, algorithms use a unique or a renewable sample. They are called respectively static or dynamic. On the one hand, in static sampling like the Historical Subset Selection [6] and bagging/boosting [7], [8], a selection of representative training set records needs to be performed. With large datasets, this poses a problem of combining downsizing and data coverage objectives. On the other hand, dynamic sampling creates samples per generation according to its selection strategy. Consequently, GP individuals do not have enough time to learn from sampled data. The population might waste some good resources for solving some difficult cases in the current training set. Otherwise, re-sampling at each GP iteration might be computationally expensive, especially when using some sophisticated sampling strategies.

We propose, in this paper, a new extension to sampling techniques in which sample renewal is controlled through a parameter that adapts the sampling to the learning process. This extension aims to preserve original sampling strategy while making an enhancement in learning robustness and/or learning time.

This paper is organized as follows: in the first section, we expose the background of this work in GP and sampling methods. Section III introduces the novel sampling approach and gives how it can extend dynamic sampling. Then, in section IV, an experimental study gives the proof of concept

978-1-7281-0003-6/19/$31.00 © 2019 IEEE

of adaptive sampling and traces its effect on learning process through the discussion of registered results in section V. Finally, we give some conclusions and propose further developments.

II. Background and related works

A. Genetic Programming

As any EA, GP evolves a population of individuals through iterations called generations. Each individual represents a complete mathematical expression or a small computer program. The standard GP uses a tree representation of individuals built from a function set for nodes and a terminal set for leaves. When GP is applied to a classification problem, each individual is a candidate classifier. The terminal set is composed of dataset features and some randomly generated constants and the function set contains mostly arithmetic and logic functions. As for the fitness function, it is often based on learning performance measures such as accuracy.

The main steps of GP with dynamic or active sampling are:

1) Randomly create a population of individuals where tree nodes are taken from a given function and terminal sets. Then evaluate their fitness value by executing each program tree against the initial training subset.
2) According to a fixed probability, individuals are crossed or mutated to create new offspring individuals.
3) Select a new training subset with a given sampling algorithm.
4) New solutions are evaluated against new sample and a new population is made up by selecting best individuals from parents and offspring according to their fitness values.
5) Loop step 2 and 4 until a stop criteria is met.

The evaluation is the prevailing step with regard to the overall computation cost and it depends simultaneously on the sample size, the population size and each individual complexity.

B. Dynamic Sampling

In this section we present some dynamic sampling algorithms with focus on those used in the conducted experiments in section IV.

1) Random Subset Selection (RSS): Gathercole et al. [6] proposes a simple algorithm that selects at every generation g a record i among T records in the initial dataset with a uniform probability $P_i(g)$:

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

where S is the target subset size. The sampled subset has a fluctuating size around S.

2) Dynamic Subset Selection (DSS): DSS algorithm [9]–[11] is inspired by boosting techniques and aims to bias selection to keep difficult cases (i.e. fitness cases frequently unsolved by the best solutions) and fitness cases that have not been selected for several generations.

DSS computes two measures: 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 for each classification error and reset to 0 if the fitness case is solved. 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.
Selection probability$P_i(g)$ in eq. (3) depends on each fitness weight $W_i(g)$ (eq. (2)).

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

where d and a are given parameters denoting respectively the difficulty exponent and the age exponent.

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

3) Balanced sampling: Balanced sampling [12] aims to improve classifier 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 reduces the number of instances like the methods studied in this paper. Several approaches are proposed, we summarize hereafter three sampling techniques used with GP. First, Static Balanced Sampling that selects cases with uniform probability from each class without replacement until obtaining a balanced subset of the desired size. Then, Basic Under-sampling (BUSS) (resp. Basic Over-sampling) selects all minority (resp. majority) class instances and then an equal number from the majority (resp. majority) class randomly.

4) RSS-DSS and DSS-DSS: Curry et al. conceived an extension to the DSS algorithm into a 3 levels hierarchy [13]. At level 0, the data set is first partitioned into blocks that are sufficiently small to reside within RAM alone. Then, at level 1, one block is chosen from this partition based on RSS or DSS. Finally, at level 2, the selected block is considered as the full dataset on which DSS was applied for several generations. Depending on the level 1 algorithm, two approaches are possible: *RSS-DSS* hierarchy or *DSS-DSS* hierarchy.

5) RSS-TBS (TBS: Topology Based Selection): Based on the same idea, Hmida et al. proposed two new variants of hierarchical sampling : the RSS-TBS and the BUSS-RSS-TBS [14]. The RSS-TBS uses the Topology Based Subset Selection at level 2 instead of RSS or DSS. The second variant BUSS-RSS-TBS extends the first variant with a Basic Under-sampling at the level 0 block creation. BUSS favors the minority class by calculating the block size according to their cardinalities. For majority class, an equal number of instances are selected randomly.

III. Controlling Sampling Frequency

A. Sampling approach

Sampling frequency (f) is a main parameter for any active sampling technique. It defines how often the training subset is changed across the learning process. When f=1, the training sample is extracted at each generation and the sampling approach is considered as a generation-wise sampling technique.

Most of the sampling techniques applied with GP belong to this category. This is the case of the techniques described in section II-B. When f is set to 1, individuals in the current population have only one generation to adapt their genetic materials to the current environment characterized by the training sample. It is very difficult even impossible for a population to solve all cases in a training set in one generation. A higher value of f corresponds to a lower number of samples to be generated. We think that the sampling frequency must be updated according to the evolution state and the difficulty of the current training set.

The fundamental idea behind adaptive sampling by controlling the sampling frequency is to add an extra parameter to sampling algorithms acting as a moderator or re-sampling regulator. While active or dynamic methods use a fixed renewal frequency equal to 1, adaptive sampling decides to generate a new sample for the subsequent generations according to a condition that must be true.

Fig. 1 depicts this approach. GP based learner interacts with sampling process by providing adequate data about the learning process in order to perform the underlying selection strategy. For example, DSS needs to know misclassified cases to update the difficulty value. Then, the sampling algorithm delivers a new sample generated according to the updated data. With adaptive sampling, a predicate controls the re-sampling decision. We assume that any data required to evaluate this predicate is available within the data dispensed by the GP engine.

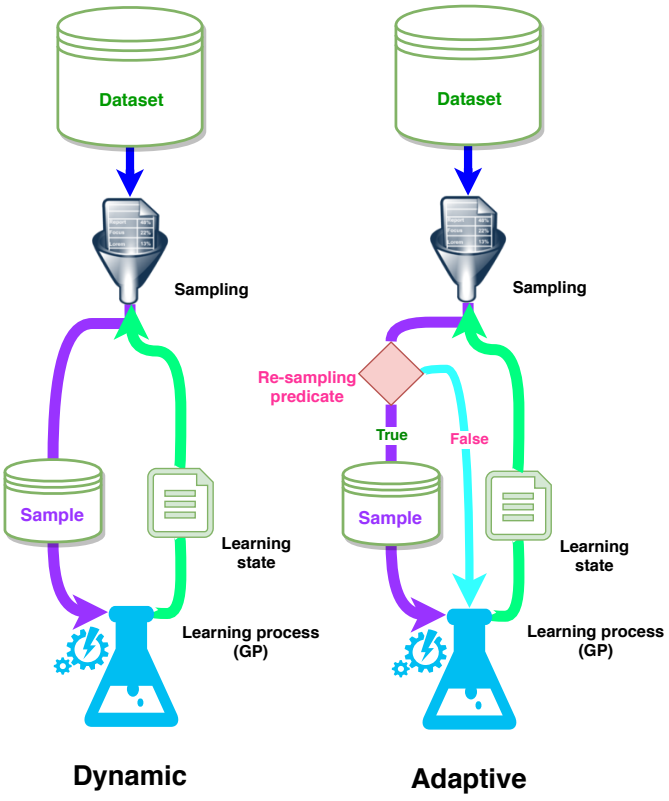


Fig. 1. Adaptive vs Dynamic sampling

Various predicates can be used for adaptive sampling that are based on other parameters such as:

- generations number,
- population mean fitness,
- mean fitness improvement rate,
- best fitness improvement rate,
- etc.

The straightforward approach is to define a threshold per measure and the predicate is then a comparison of the current value to the corresponding threshold.

For example, if we define a threshold of 0.002 to best fitness improvement rate then GP will continue to use the same sample if the best fitness of the current generation is better than the previous generation with 0.2%. Otherwise a new sample must be created. In a more complex approach, threshold can be auto-adapted to learning process.

We propose below some new techniques to adapt the sampling frequency. After a brief recall of the classic method considering the sampling frequency as an EA input parameter with a fixed value usually set to 1, we introduce the deterministic sampling frequency which the value evolves according to predefined schema. We then present the adaptive sampling frequency based on the current state of the population. It uses either the evolution of the mean fitness or the number of resolved cases to decide whether to create a new sample or to carry on learning with the previous one.

B. Fixed Sampling Frequency

The sampling frequency f is set before starting the GP run like any GP parameter. This value remains unchanged till the last generation. This can be represented by the following algorithm:

Input: $f\{\text{sampling frequency}\}$
1: **for all** generation $g < g_{max}$ **do**
2: **if** $g \mod f = 0$ **then**
3: re-sample
4: **end if**
5: **end for**

C. Deterministic sampling frequency

When a deterministic control of the sampling frequency is applied, the frequency f gets different values during the GP run. It is guided by a function that delivers the same values at each run. Thus, frequency can increase, decrease or have a complex curve.

An increasing frequency starts with samples having short lifetime. By the end of the run, samples are used for a larger number of generations. This can be achieved through the following steps:

1: **for all** generation $g < g_{max}$ **do**
2: $f = (C \times g)^{\alpha}$ $\{C, \alpha \in \mathbb{R}\}$
3: **if** $g \mod f = 0$ **then**
4: re-sample
5: **end if**
6: **end for**

The opposite process (i.e. decreasing frequency) uses the same steps but updates frequency with a decreasing function such as: $f = (C \times (g_{max} - g))^{\alpha}$.

D. Adaptive sampling frequency

In this technique, sampling frequency value is adjusted to be more suitable to the current learning state. Therefore, it can increase or decrease by a varying amount. We assume, that less performing learners need more time to improve their performance and symmetrically efficient learners on a particular sample need to see different data from a new sample.

Adaptive sampling can rely on various learning performance indicators. It retrieves these indicators from the GP engine. Hereafter, two examples of adapting techniques.

The first uses a threshold for the population mean fitness to detect if the population is making improvements or not. In the latter case, a new sample is generated since the old one is fully exploited.

Input: $r\{$mean fitness variation rate: (mean fitness(g)-mean fitness(g-1))/mean fitness(g-1)$\}$
Input: $t\{$threshold$\}$
1: **for all** generation $g < g_{max}$ **do**
2: **if** $r > t$ **then**
3: re-sample
4: **end if**
5: **end for**

The second example is based on measuring the mean number of individuals that have resolved each record in the sample. When this value reaches a designated value then new records are selected in a new sample.

In the following sections, we give details about the settings used for the conducted experiments and implementation of adaptive sampling over some dynamic sampling algorithms discussed in section II-B. Then we expose the experimental results and discuss them to analyze the effect of sampling frequency and adaptive sampling on GP performance in resolving KDD intrusion detection problem.

IV. Experimental settings

A. Cartesian Genetic Programming

Cartesian Genetic Programming [15] is a GP variant where individuals represent graph-like programs. It is called "Cartesian" because it uses a two-dimensional grid of computational nodes implementing directed acyclic graphs. Each graph node encodes a function from the function set.

CGP shows several advantages over other GP approaches. Unlike trees, there are more than one path between any pair of nodes. This enables the reuse of intermediate results. A genotype can also have multiple outputs which make CGP able to solve many types of problems and classification problems in particular [16]. Otherwise, CGP has the great advantage of counteracting the bloating effect (genotype growth), frequent phenomena with other GP representations. CGP is easy to implement, and it is highly competitive compared to other GP methods.

1) CGP settings: The design of CGP parameters used in this work is summarized in Table I. In this work, the parameter tuning is not fully explored.

TABLE I
CGP parameters.

Parameter	Value
Population size	256
Sub-populations number	1
Generations number	200
CGP nodes	300
Inputs	49
Outputs	1 (2 classes)
Tournament size	4
Crossover probability	0.9
Mutation probability	0.04
Fitness	Minimize classification error

2) Terminal and function sets: The terminal set includes 41 features of the benchmark KDD-99 dataset. The function set includes basic arithmetic, comparison and logical operators reaching 17 functions (table II).

TABLE II
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
Random Constants	8 in $[-2, 2[$

B. Dataset

The KDD-99 intrusion detection problem consists in classifying connections into normal or attack classes. It uses a large dataset called 10% KDD-99 dataset [17]. The dataset is already divided into training and test sets which are presented in Table III.

TABLE III
KDD-99 dataset.

Class	**Number of instances**	
	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 examples	494021	311029

Each fitness case is described by 41 features. The original data is preprocessed with the following steps:

- Transforming discrete nominal attributes to numeric values,

- Scaling data using MinMax scaler:

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}},$$

- Binarization of attack classes: the problem is converted to a binary classification problem with a 'Normal'class and 'Attack'class. The original four attack types (Dos, Probe, R2L and U2R) are fused in a single class.

C. Performance Metrics

We recorded, for each run, its accuracy (eq. (4)) and False Positive Rate (FPR) (eq. (3)) to measure the learning performance on both training and test sets. We also recorded the learning time measuring the computational cost.

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ patterns} \quad (4)$$

$$FPR = \frac{False\ Positives}{False\ Positives + True\ Negatives}. \quad (5)$$

D. Framework

Software framework: Among several evolutionary computation frameworks, Sean Luke's ECJ [18] was used in this work to implement and test the CGP. It's an open source framework written in Java and benefits of many contribution packages as the one used here for implementing Cartesian GP developed by David Oranchak [19]. This framework provides a very flexible API using parameter files well documented in the ECJ owner's manual.

Hardware framework: Experiments are performed on an Intel $i7 - 4810MQ$ $(2.8GHZ)$ workstation with $8GB$ RAM running under Windows $8.164 - bit$ Operating System.

E. Sampling settings

In the first set of experiments, we tested six values for the sampling frequency: 1, 10, 20, 30, 40 and 50 on four sampling methods BRSS, BUSS, DSS and RSS. BRSS is Balanced RSS. It is an RSS variant where the random sample is balanced according to a given ratio between problem classes.

In the second part, we implemented four different techniques to control sampling frequency. Two deterministic techniques and two adaptive ones as follows:

- *Deterministic+*: deterministic controlling with the increasing function $f = (2 \times g)^{0.5}$,
- *Deterministic-*: deterministic controlling with the decreasing function $f = (2 \times (g_{max} - g))^{0.5}$,
- *Average Fitness*: adaptive controlling based on population average fitness with a threshold of 0.001,
- *Min Resolved*: adaptive controlling based on the average population proportion of the population representing the individuals that resolved sample records. We use a minimum threshold of 0.5.

The underlying active sampling algorithms have their own parameters described in Table IV.

TABLE IV
COMMON SAMPLING PARAMETERS.

Method	Parameter	Value
All (except BUSS)	Target Size	5000
BRSS	Balancing method	Full dataset distribution
BUSS	Target size	416
DSS	Difficulty exponent	1
	Age exponent	3.5

V. Results and Discussion

This study is organized in 2 types of experiments. The first aims to study the impact of sampling frequency variation on learning time and learning performance indicators.

For each value of re-sampling frequency and controlling technique, 21 runs of each sampling algorithm are conducted. We report the mean learning time of each configuration and accuracy and FPR of the best individual.

A. Effect of sampling frequency

Fig. 2 illustrates learning time variation on re-sampling frequency for the studied algorithms. Fig. 3 and Fig.4 show the effect of sampling frequency on two learning quality measures accuracy and FPR.

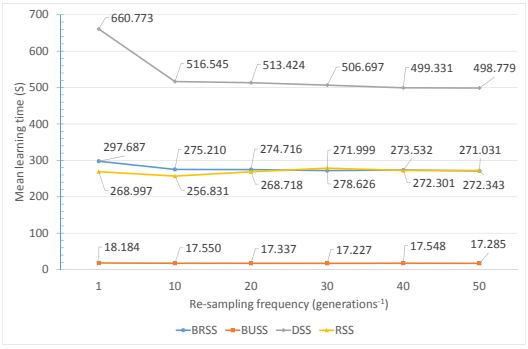
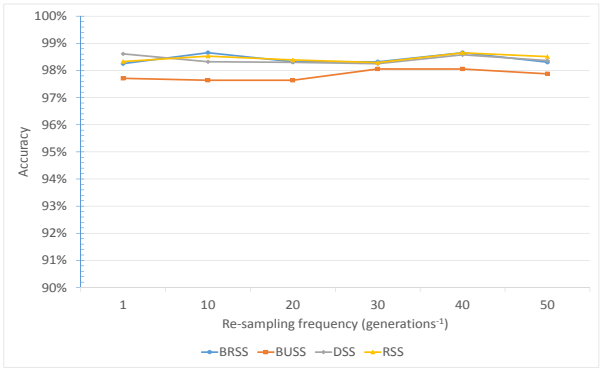


Fig. 2. Variation of the mean learning time with the re-sampling frequency.

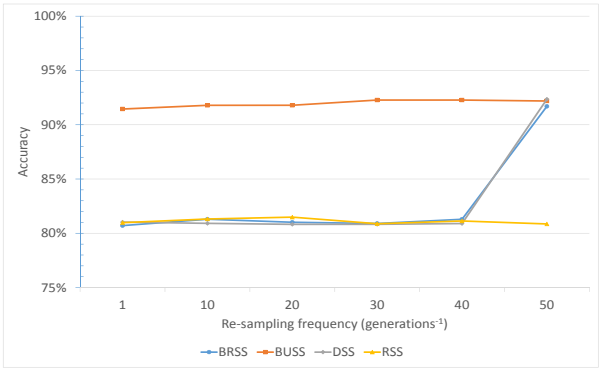
The shape of the curves in Fig. 2 reveals two distinct behaviors when the sampling frequency increases. The first concerns the BUSS, BRSS and RSS algorithms that have recorded an insignificant decreasing variation of the average learning time. The second behavior is that of DSS with a more remarkable decrease.

Indeed, for a frequency equal to 1, 200 sampling operations are made and for a frequency of 50 this number goes down to 4. Time saving depends on the time needed to perform sample creation with respect to the time spent for a whole generation. This is why the decrease in time is not very important since population evaluation is the predominant step in the learning time for GP.

DSS algorithm differs from other algorithms by updating certain parameters (age and difficulty). But this does not affect

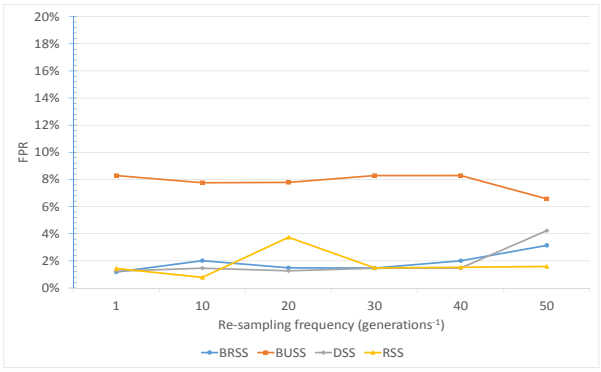


(a) Training set

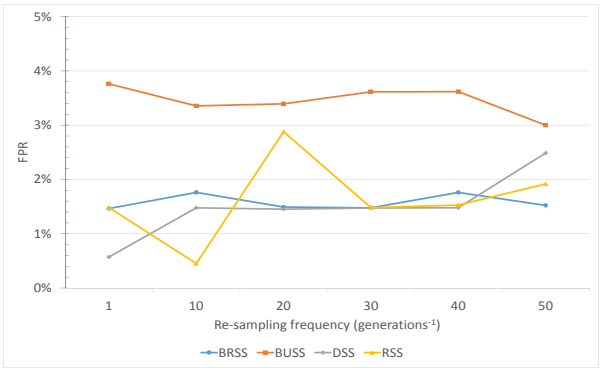


(b) Test set

Fig. 3. Variation of the best individual accuracy with the re-sampling frequency.



(a) Training set



(b) Test set

Fig. 4. Variation of the best individual FPR with the re-sampling frequency.

the learning time since it is performed at each generation regardless of the re-sampling frequency. In DSS, the selection of a learning case requires the calculation of a probability based on the age and difficulty values of the whole dataset and subsequently a greater time than the other techniques. This is at origin of the difference in learning time saving.

From Fig. 3, the variation of the rate of accuracy, contrary to expected results, did not make any improvement for the 2 sets. Although, there is an increase of Accuracy at frequency 50, for DSS and BRSS for the test set. However, this remains irregular and can not be generalized.

The most noticeable shift is that of the FPR (Fig. 4). However, no empirical correlation with the variation of the re-sampling frequency for BRSS, DSS and RSS can be made. Only BUSS realizes a decrease in FPR value decrease when re-sampling frequency increases for training and test sets.

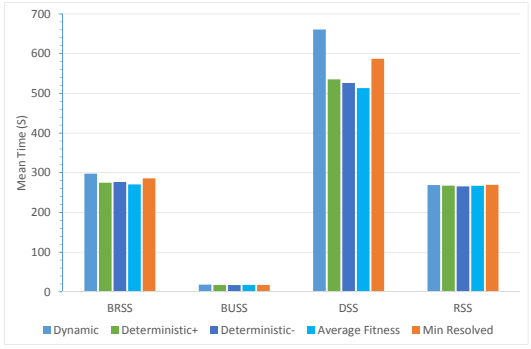
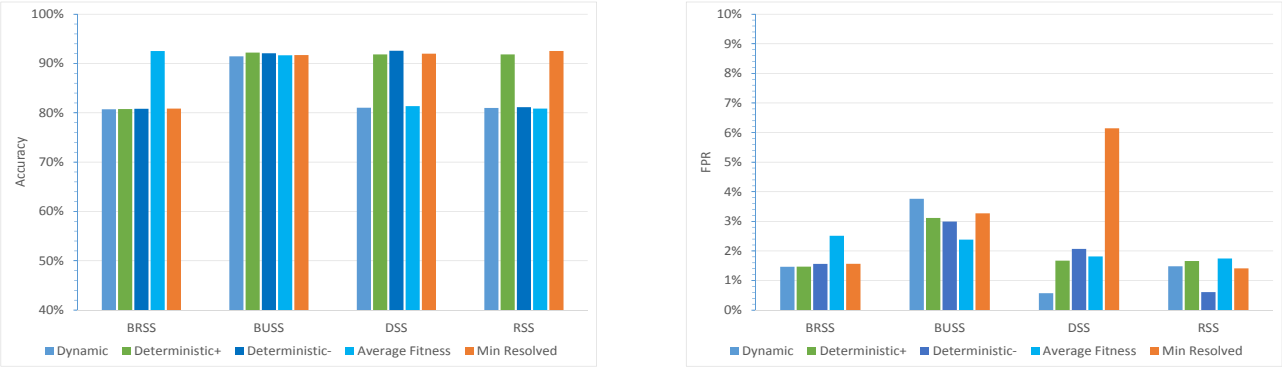


Fig. 5. Variation of the mean learning time according to the re-sampling frequency controlling strategy.

B. Adaptive sampling

Figures 5 and 6 report the experimental results of 4 sampling algorithms extended with 4 different sampling frequency techniques (section IV-E): deterministic based on an increasing or decreasing function (Deterministic+ and Deterministic-) and adaptive controlling based on the population average fitness value (Average Fitness) or the average number of individuals that resolved the samples cases (Min Resolved). The obtained results are compared to those from original



(a) Accuracy

(b) FPR

Fig. 6. Variation of the accuracy and FPR measures according to the re-sampling frequency controlling strategy.

dynamic algorithms (Dynamic).

For the mean learning time, results from Fig. 5 confirm the previous remarks about this behaviour in section V-A. It is only learning time of DSS which is clearly affected by introducing any controlling technique whether it is deterministic or adaptive. This effect is more or less strong according to reducing samples number used per GP run.

As regards learning performance on the test set, results do not rise a general response to controlling sampling frequency by means of the 4 studied techniques.

For BUSS algorithm, accuracy records an infinitesimal improvement with controlled frequency techniques. In addition to that, FPR made a similar change in the same direction (Fig. 6).

For each sampling method, there is at least one frequency control technique that allows it to improve, largely or slightly, its learning performance. For example, for the RSS, the 'Deterministic+' and 'Min Resolved' techniques have allowed the accuracy to move from values around 80% to values greater than 90%. Similarly, the two deterministic methods (deterministic + and deterministic-) and the Adaptive method 'Min Resolved' have improved significantly the accuracy values for DSS sampling, whose values increased by 10% to 12%. However, a deterioration of the FPR has been recorded.

For the BUSS, only the adaptive controlling technique 'Average fitness' improved the accuracy, but the other techniques do not impact its performance comparing to the generation-wise frequency. In fact, the control of the sampling frequency, if it does not allow an improvement of the results, does not generate a deterioration of the performance, except for some cases for the FPR measure.

VI. Conclusion

The idea of adaptive sampling has its origins in the assumption that a population needs enough time (generations) to adapt to learning data. We have formalized this principle in the form of a re-sampling predicate that is based on a condition to decide whether to modify the current sample or to keep it.

Experiments are limited to testing adaptive sampling by controlling sampling frequency with simple predicates. The results showed a slight effect on learning time. This effect is in the direction of a decrease but with different degrees depending on the sampling method.

With adaptive sampling, computational cost can be improved according to the underlying dynamic sampling algorithm only if the selection process is time consuming as it is for DSS.

The used predicates did not deteriorate the learning accuracy. However, it did not have a proven positive and generalizable effect on the quality of learning. Thus, they need to be refined.

Many new research paths emerges from this study that are worthy of further investigation. A first path is the exploration of other predicates that take into account the characteristics of the training dataset and the underlying problem to find more relevant predicates for GP classifier improvements. A second one is to extend the scope of adaptive sampling to other sample properties. For instance, an adaptive sampling can downsize or upsize the sample instead of generating a new one. We may also combine several sampling strategies and algorithms in a single method. Then, according to the learning evolution, a sample is generated using the suitable strategy in an interleaved way: we use a different algorithm at each time we need to create a new sample.

References

[1] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms*, ser. Decision Engineering. London: Springer London, 2010.

[2] D. Simon, *Evolutionary Optimization Algorithms*. USA: John Wiley & Sons, 2013.

[3] A. Ptrowski and S. Ben-Hamida, *Evolutionary Algorithms*. USA: John Wiley & Sons, 2017.

[4] J. R. Koza, *Genetic programming - on the programming of computers by means of natural selection*, ser. Complex adaptive systems. MIT Press, 1992.

[5] H. Hmida, S. B. Hamida, A. Borgi, and M. Rukoz, "Sampling methods in genetic programming learners from large datasets: A comparative study," in *Advances in Big Data - Proceedings of the 2nd INNS Conference on Big Data, October 23-25, 2016, Thessaloniki, Greece*, ser. Advances in Intelligent Systems and Computing, P. Angelov,

Y. Manolopoulos, L. S. Iliadis, A. Roy, and M. M. B. R. Vellasco, Eds., vol. 529, 2016, pp. 50–60. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-47898-2_6

[6] C. Gathercole and P. Ross, "Dynamic training subset selection for supervised learning in genetic programming," in *Parallel Problem Solving from Nature III*, ser. LNCS, Y. Davidor, H.-P. Schwefel, and R. Manner, Eds., vol. 866. Jerusalem: Springer-Verlag, 9-14 oct 1994, pp. 312–321. [Online]. Available: http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/94-006.ps.gz

[7] H. Iba, "Bagging, boosting, and bloating in genetic programming," in *The 1st Annual Conference on Genetic and Evolutionary Computation, Proc.*, ser. GECCO'99, vol. 2. San Francisco, CA, USA: Morgan Kaufmann, 1999, pp. 1053–1060.

[8] G. Paris, D. Robilliard, and C. Fonlupt, "Exploring overfitting in genetic programming," in *Artificial Evolution, 6th International Conference, Evolution Artificielle, EA 2003, Marseilles, France, October 27-30, 2003*, ser. Lecture Notes in Computer Science, P. Liardet, P. Collet, C. Fonlupt, E. Lutton, and M. Schoenauer, Eds., vol. 2936. Springer, 2003, pp. 267–277. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-24621-3_22

[9] C. Gathercole and P. Ross, "Dynamic training subset selection for supervised learning in genetic programming," in *Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation, Proc.*, ser. Lecture Notes in Computer Science, vol. 866. Springer, 1994, pp. 312–321.

[10] C. Ghatercole and P. Ross, "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.* San Francisco, CA: Morgan Kaufmann, 1997, pp. 111–118.

[11] C. Gathercole, "An investigation of supervised learning in genetic programming," Thesis, University of Edinburgh, 1998.

[12] R. Hunt, M. Johnston, W. N. Browne, and M. Zhang, "Sampling methods in genetic programming for classification with unbalanced data," in *AI 2010: Advances in Artificial Intelligence - 23rd Australasian Joint Conference, Proc.*, ser. Lecture Notes in Computer Science, vol. 6464. Springer, 2010, pp. 273–282.

[13] R. Curry and M. I. Heywood, "Towards efficient training on large datasets for genetic programming," in *Advances in Artificial Intelligence, 17th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI 2004, Proc.*, ser. Lecture Notes in Computer Science, vol. 3060. Springer, 2004, pp. 161–174.

[14] H. Hmida, S. B. Hamida, A. Borgi, and M. Rukoz, "Hierarchical data topology based selection for large scale learning," in *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France, July 18-21, 2016*. IEEE, 2016, pp. 1221–1226. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0186

[15] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming, European Conference, Proc.*, ser. Lecture Notes in Computer Science, vol. 1802. Springer, 2000, pp. 121–132.

[16] S. Harding and W. Banzhaf, "Implementing cartesian genetic programming classifiers on graphics processing units using gpu.net," in *GECCO 2011 Computational intelligence on consumer games and graphics hardware (CIGPU)*, S. Harding, W. B. Langdon, M. L. Wong, G. Wilson, and T. Lewis, Eds. Dublin, Ireland: ACM, 12-16 July 2011, pp. 463–470.

[17] UCI, "Kdd cup," 1999. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/KDD+Cup+1999+Data

[18] S. Luke, "Ecj homepage," 2017. [Online]. Available: http://cs.gmu.edu/~eclab/projects/ecj/

[19] CGP, "Cartesian gp website," 2009. [Online]. Available: http://www.cartesiangp.co.uk