A Study on Genetic Programming with Layered Learning and Incremental Sampling

Nguyen Thi Hien
Le Quy Don University
100 Hoang Quoc Viet St
Hanoi, Vietnam
Email: nguyenthihienqn@gmail.com

Nguyen Xuan Hoai
IT R&D Center,
Hanoi University
9th Km, Nguyen Trai St
Hanoi, Vietnam
Email: nxhoai@hanu.edu.vn

Bob McKay
Gwanak-gu, Gwanangno 599
School of Computer Science and Engineering
Seoul National University 302-427
rimsnucse@gmail.com

***Abstract*—In this paper, we investigate the impact of a layered learning approach with incremental sampling on Genetic Programming (GP). The new system, called GPLL, is tested and compared with standard GP on twelve symbolic regression problems. While GPLL does not differ from standard GP on univariate target functions, it has better training efficiency on problems with bivariate targets. This indicates the potential usefulness of layered learning with incremental sampling in improving the efficiency of GP evolutionary learning.**

I. Introduction

Genetic Programming (GP), since its introduction and development by Koza, has been seen as a potential machine learning method [12]. Its main objective is to discover, by evolutionary means, relations between input and output data in the form of a function or program. GP has been applied successfully to numerous real world problems, of which many are learning tasks [20]. Despite the initial successes, GP researchers and practitioners, in the early days, seldom paid attention to the generalization capability of GP. They focused more on how to use GP to fit the given data set by trying to learn the exact solution/relation (which is often impossible in real world situations).

In the field of machine learning (ML) [15], however, generalization has been seen as one of the most desirable properties of any learning method. To generalize effectively, any learning machine should avoid overfitting the training data. Recently, GP generalization has attracted more attention from GP researchers, with the number of related publications increasing [19], [5], [9], [14], [2], [3], [19], [7], [11]. In addition, there have been a number of publications on applying traditional ML techniques and practices to the learning processes of GP, in order to improve its generalization capability. Successful examples of this work have been presented in [25], [5], [9].

Layered learning (LL) is a machine learning technique that tries to decompose the learning task into subtasks, and then learn each subtask in layered (staged) fashion [21], [22]. Although LL has been applied to GP [8], [10], it has generally been used with a different purpose. Most of the related works so far have tried to combine LL with GP in learning by overfitting the training set part by part (for each subtask of the learning problem) in order to find exact solutions to problems in multi-robotic agent controls [8] and learning Boolean functions [10]. Zhang and Joung's [25] in particular studied combining LL with incremental sampling to improve the learning efficiency of GP systems. Despite the strong theoretical foundation for this combination [16] (at least for PAC learners), there does not appear to have been any previous follow-up. In particular, to the best of our knowledge, the work reported here is the first investigation of the impact of layered learning on GP's generalization capacity or learning/traing efficiency.

In this paper, we present our first investigation on the combination of layered learning and incremental sampling for training GP. The rest of the paper is organized as follows. In the next section, related work on GP generalization issues, layered learning, and the theoretical motivation for coupling incremental sampling with layered learning is presented. Section III outlines our new method for training GP. The experimental settings are given in sections IV. Section V presents the experimental results with discussion. The paper concludes with section VI, where some future work is highlighted.

II. Related works

This section gives a brief overview of the literature on the generalization capability of GP, and on the use of incremental training-set size for GP, providing the motivation for the work in this paper.

A. Genetic Programming Generalization

Although achieving high generalization capability is the main objective any learning machine [15], it was not seriously considered in the field of GP for a long time. Before Kushchu published his seminal paper on the generalization ability of GP [13], there was limited work in the literature dealing with GP generalization. In [4], Francone et al. proposed a new GP system called Compiling GP (CGP), the authors comparing its generalization ability with that of other machine learning techniques. They showed that the generalization ability of CGP is comparable with a number of more traditional machine learning approaches. The impact of the extensive use of

978-1-4244-7835-4/11/$26.00 ©2011 IEEE

the mutation operator on CGP's ability to generalize was investigated; the results showed positive effects.

Zhang and Mühlenbein [27] proposed a method to avoid overfitting in GP based on the Minimum Description Length principle, with an adaptive mechanism for balancing between accuracy and complexity according to individual preference. The method was shown to be robust for a wide class of tasks with noisy or incomplete data. In [9], Iba incorporated Bagging and Boosting into GP (BagGP and BoostGP). The results showed that these techniques could help to improve the robustness (generalization ability) of GP on the problems of discovering trigonometric identities, chaotic time series prediction, and 6 bit multiplexer [9]. These early works were good examples of the application of more traditional machine learning techniques to the learning process of GP.

Recently, the generalization aspect of GP has deservedly gained more attention from researchers and practitioners in the field. In [18], Panait and Luke investigated the impact of six common sampling methods on the robustness of GP solutions. None of the methods dominated the others on all problems, suggesting that the impact of sampling method is dependent on problem domain features.

Paris et al. [19] used GP as the core learning algorithm in a boosting framework to trigger over-fitting; GP with boosting turned out to be substantially better than standard GP on both the problems studied. Mahler et al. [14] tried Tarpeian control on symbolic regression problems and tested the side effects of this method on the generalization accuracy of GP. The results were mixed: Tarpeian control can either increase or reduce the generalization power of GP solutions depending on the problem.

In [5], Gagné et al. investigated two methods to improve generalization in GP-based learning: selection of the best-of-run individuals through three separate data sets (training, validation, and test); and the application of parsimony pressures to reduce the complexity of learned solutions. The results indicated the value of a validation set, showing increased stability of the best-of-run solutions on the test sets.

Costa and Landry [2] proposed a new GP system called relaxed Genetic Programming (RGP) with generalization ability better than traditional GP.

More recently, Costelloe and Ryan [3] also investigated the role of generalization in GP learning. They showed that popular GP techniques such as Linear Scaling [11] may only improve the fit on training data, not on testing/unseen data. They proposed a method to improve GP generalization by combining Linear Scaling with the No Same Mate strategy [7].

Vanneschi and Gustafson [24] improved GP generalization through a crossover-based similarity measure. They keep a list of over-fitted individuals, and try to prevent similar individuals entering the next generation (based on structural distance or a similarity measure based oh subtree crossover). The method was tested on a real-life drug discovery regression problem and showed improved generalization ability. In [23], Vanneschi et al. proposed a method to quantify/detect over-fitting during the GP learning process.

Nguyen Q U et al. [17] showed that semantic information could be used to guide the crossover operator of GP in reducing the code bloat, improving its generalization capability on real-valued symbolic regression problems.

B. Layered learning

The layered learning paradigm was first formally introduced by Stone and Veloso [22] as an extension and formalization of earlier work by Asada et al. [1] and de Garis [6]. The main idea of LL is to solve the learning problem in a hierarchical and bottom-up fashion. The problem is decomposed into subtasks, often as a lower order form of the original learning problem. The learning process is then conducted in stages (layers). At each stage, the learning machine learns to solve a subtask, once the solution for the subtask has been obtained the learning machine starts learning in the next stage (layer) to solve the task in the next level, and has access to the solutions learnt in the previous stages (layers). The principles of LL can be summarized as follows [22], [21]:

1) Direct learning of a task might be intractable.
2) Bottom-up and hierarchical task decomposition could be possible.
3) A learning machine could exploit data to train and/or adapt its learning process separately at each level.
4) The output of learning process in one layer feeds into the next layer.

Early work on LL mainly focused on the learning tasks in multi agent systems. In [21], Stone applied LL to the problem of learning skills for soccer agents in a multi-agent environment. He tried three layers of learning for each agent, with different learning skills (from individual skills such as intercepting the ball to team skills as ball passing) to be acquired. Different learning machines (methods) were used to accomplish the learning task at each layer [21].

Gustafson and Hsu [8] proposed an LL approach to learn strategies for the keep-away soccer game, in which a team of four players must prevent a single opposition player coming into contact with the ball. They designed two layers for the learning problem. The learning objective in the first layer is to maximize the number of accurate passes, while in the second layer it is to minimize the number of ball turnovers to the opposition player. Both layers used GP, and each layer lasted a fixed number of generations. The population at the final generation of the first layer was the initial population for the next layer, and the fitness function switched to the next task at the same time.

In [10], Jackson and Gibbons applied two approaches of LL to GP learning on more GP traditional problem domains, learning Boolean functions. In the first, they used two layers of learning, with the first layer learning a subset of fitness cases of the 4-parity problem, and the second learning the whole set of fitness cases. The experiments with this approach gave disappointing results, in over-fitting to the exact solution for the problem. They then modified the approach so that the first layer was used for learning a simplified version of the original problem (with fewer input variables) while the

second exploited what had been learnt in the first to discover exact solutions for the original problem. The results showed the positive effects of using LL compared to standard GP and GP with ADF.

C. Incremental Sampling

In [16], Muggleton theoretically analyzed the combination of layered learning with incremental sampling, deriving general lower-bound results for Probably Approximately Correct (PAC) learning. The learning process begins by taking a small sample from a stream of available training data, and uses it to construct an approximately correct theory (using any PAC learning machine). A second approximately correct theory is then constructed based on the error of the first theory, using a new sample set which is a superset of the first. Further layers of correcting theories are then added using successively larger samples, until a predefined level of accuracy of the overall theory is achieved. Muggleton showed that if the sample size increases linearly (with respect to the VC dimension of the hypothesis space), the lower bound for generalization error will exponentially decrease over the learning layers. Muggleton suggested that the use of LL and incremental sampling might not be subject to results on hardness of learning that apply to PAC learning by machines with a single layer of learning.

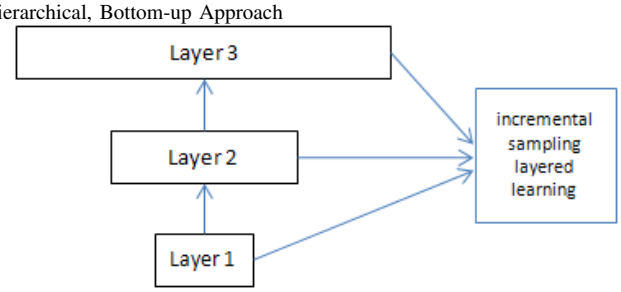
In [26], Zhang proposed a Bayesian framework for GP learning. From Bayesian theory, he suggested that the training set for GP learning should increase during the learning/training phase. In [25], [26], GP with incremental data inheritance (IDI) was proposed, and applied to the task of evolving cooperation strategies for robotic agents. IDI distinguishes two populations: program population and data population. Both the program and its data concurrently evolve toward an optimal combination. The size of the training set used in the experiments was increased by a seemingly arbitrary increment of 6 at each generation. In the results, IDI's generalization error was non-significantly improved over standard GP, but the training time was significantly shorter.

III. Proposed Method

Our proposed training method for GP is motivated by the theoretical study of layered learning in [16]. It resembles Jackson and Gibbons' first approach [10] and aspects of Zhang and Joung [25], [26] but differs from them in a number of ways. While [10] focuses on training set accuracy, we emphasize GP generalization. The learning process in [25], [26] is a special case of LL, in which the length of each layer is one generation, and the increase in sample size is somewhat arbitrary (6). Our approach is more faithful to LL, with the increase in training sample size motivated by theory from [16]. The problems tested in [10] and [25], [26] are frp, Boolean and Multi-agent domains, while we concentrate on more traditional real-valued symbolic regression from the GP literature. Finally, while the previous approaches to GP with LL use a fixed number of generations as the length of each layer, we employ a different stopping criterion, which will be described next.

The learning/evolutionary process of our Layered learning GP (GPLL) system is divided into m layers. It starts as in standard GP, except that only a subset of the training examples are presented to the system. When the stopping criterion is satisfied in each layer, the next layer commences. The population in the last generation of the previous layer becomes the initial population of the next layer, and the training sample set is incremented with newly added samples (drawn under the same distribution from the problem training samples). At the ith layer, the size of the training set (fitness cases) is $|D_i| = k|D_{i-1}|$, where k a predefined constant (that is, the training cases increase exponentially). The process is repeated until the required number of layers (3 in this paper) have been completed or some stopping criteria are met. Figure 1 summarizes the GPLL learning process.

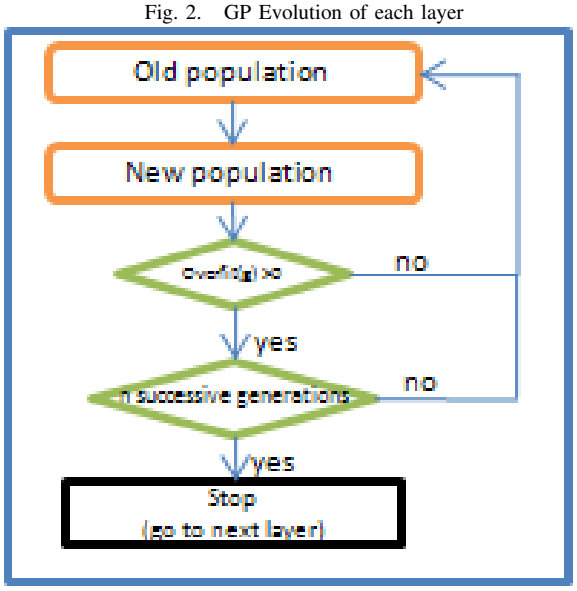
Fig. 1. Incremental Sampling Layered Learning Framework based on a Hierarchical, Bottom-up Approach



At each layer, when the new population is created from the old population by the applications of selection, crossover, and mutation operators (generational transition), a stopping criterion is tested. It checks whether there is overfitting in the new population. If overfitting has been detected for n successive generations (i.e. the learning is no longer productive), the learning layer is ended. Figure 2 depicts this process.

Overfitting is estimated as proposed in Vanneschi et al.[23]. The detail of the method is presented in algorithm 1. In the algorithm, bvp is "best valid point" and denotes the best validation fitness (error on the validation set) found up to the current generation, excluding those generations (usually at the beginning of a run) where the best individual on the training set has higher accuracy on the validation set than the training set. $tbtp$ stands for "training at best valid point" – i.e. the training fitness of the individual that has validation fitness equal to btp. Training_Fit(g) is a function that returns the best training fitness in the population at generation g. Val_Fit(g) returns the validation fitness of the best individual on the training set at generation g. In this paper, we have made a further simplification, in that we ignore the absolute value of over_fit(g), just noting whether it is positiv. The stopping criterion for an layer in GPLL is satisfied if it has been detected as overfitting in n successive generations, where n is a tunable parameter.

Fig. 2. GP Evolution of each layer



Algorithm 1: Calculation of the Degree of Over-fitting at each Generation.

over_fit(0)=0;
bvp = Val_Fit(0);
$tbtp$ = Training_Fit(0);
foreach *generation* $g > 0$ **do**
  if *Training_Fit(g)* $>$ *Val_Fit(g)* **then**
    over_fit(g) = 0;
  else if *Val_Fit(g)*$<$ bvp **then**
    over_fit(g) = 0;
    bvp = Val_Fit(g);
    $tbtp$ = Training_Fit(g);
  else
    over_fit(g)
    $=|Training\_Fit(g) - Val\_Fit(g)| - |tbtp - bvp|$;

IV. Experimental Settings

To test the impact of layered learning and incremental sampling on the learning efficiency of GP, we tested GPLL and a standard GP system (GPM) on twelve symbolic regression problems. These problems have been widely used as benchmarks for testing the generalization performance of GP systems. Among these twelve, the target learning functions in six problems are univariate functions ($f : R \rightarrow R$), with the remaining six being bivariate ($f : R^2 \rightarrow R$). Their mathematical formulae are given in equations 1 to 12. The parameter settings are given in Table I. We emphasize that GPLL uses the same algorithm and settings (even, the same initial random seed) as GPM, except that in the former, the training set (fitness cases) increases at each of the three learning layers. Table II shows how the training, validation, and test data sets were formed in our experiments.

All runs were conducted on a Compaq Presario CQ3414L computer with Intel Core i3-550 Processor (4M Cache, 3.20GHz) running on Ubuntu Linux operating system.

$$F_1(x) = x^4 + x^3 + x^2 + x \quad (1)$$
$$F_2(x) = x^3 - x^2 - x - 1 \quad (2)$$
$$F_3(x) = \arcsin x \quad (3)$$
$$F_4(x) = \sqrt{x} \quad (4)$$
$$F_5(x) = \sin(2\pi x) \quad (5)$$
$$F_6(x) = \cos(3x) \quad (6)$$
$$F_7(x, y) = x^y \quad (7)$$
$$F_8(x, y) = xy + \sin((x - 1)(y - 1)) \quad (8)$$
$$F_9(x, y) = x^4 - x^3 + \frac{y^2}{2} - y \quad (9)$$
$$F_{10}(x, y) = 6\sin(x)\cos(y) \quad (10)$$
$$F_{11}(x, y) = \frac{8}{(2 + x^2 + y^2)} \quad (11)$$
$$F_{12}(x, y) = \frac{x^3}{5} + \frac{y^3}{2} - y - x \quad (12)$$

TABLE I
Parameter Settings for the Genetic Programming System

Population Size	500
Maximum generation	150
Tournament size	3
Crossover probability	0.9
Mutation probability	0.05
Initial Max depth	6
Max depth	15
Non-terminals	+, -, *, / (protected version) , sin, cos, exp, log (protected version)
Number of runs	100
Standardized fitness	mean absolute error
Elitism	

V. Results and Discussions

For each run of GPLL and GPM, we recorded the generalization (test set) error (GE) of the best individual of the run, the size of that best individual, the total run time, and the first generation where the best individual was discovered. We conducted three sets of GPLL experiments, with n (the number of generations required before detection of over-fitting) set to 3, 6 and 9. Tables 3 and 4 present these results for $n \in 3, 9$, averaged over 100 runs, and with standard deviations). We omitted $n = 6$ results as being intermediate in nature between the other two sets.

To test the significance of the difference in generalization error between GPLL and GPM, we used a two-tailed pairwise t-test with confidence level of 0.95 ($\alpha = 0.05$); the p-values are shown.[1] Our null and alternative hypotheses are as follows:

- H_0 = "the average test-set error of GPLL and GPM are the same".
- H_1 = "GPLL and GPM have different test-set errors".

[1]p-values shown as 0.0000 are actually numbers less than 0.00005.

TABLE II
DATA SETS FOR THE TEST FUNCTIONS.
RANGES ARE DENOTED USING [START:STEP:STOP] NOTATION WHEN THE SET IS CREATED USING REGULAR INTERVALS.
THE NOTATION [MIN, MAX] DEFINES RANDOM (UNIFORM) SAMPLING IN THE RANGE.
THE MESH([] × []) DEFINES REGULAR SAMPLING IN TWO DIMENSIONS.

Num	Function	Training set (3 layers)	Validation set	Test set
1	F_1	[40,80,160] points of [-1,1]	80 points of [-1,1]	200 points of [-1:0.01:1]
2	F_2	[40,80,160] points of [-1,1]	80 points of [-1,1]	200 points of [-1:0.01:1]
1	F_3	[40,80,160] points of [-1,1]	80 points of [-1,1]	200 points of [-1:0.01:1]
4	F_4	[40,80,160] points of [0,4]	80 points of $[0, 4]$	200 points of [0:0.02:4]
5	F_5	[40,80,160] points of [-1,1]	80 points of [-1,1]	200 points of [-0.5:0.01:1.5]
6	F_6	[40,80,160] points of [-1,1]	80 points of [-1,1]	200 points of [0:0.01:2]
7	F_7	[100,200,400] points of [0,1]x[0,1]	150 points of $[0, 1] \times [0, 1]$	10000 points of $[0 : 0.01 : 1] \times [0 : 0.01 : 1]$
8	F_8	[100,200,400] points of [-3,3]x[-3,3]	150 points of $[-3, 3] \times [-3, 3]$	3600 points of $[-3 : 0.1 : 3] \times [-3 : 0.1 : 3]$
9	F_9	[100,200,400] points of [-3,3]x[-3,3]	150 points of $[-3, 3] \times [-3, 3]$	3600 points of $[-3 : 0.1 : 3] \times [-3 : 0.1 : 3]$
10	F_{10}	[100,200,400] points of [-3,3]x[-3,3]	150 points of $[-3, 3] \times [-3, 3]$	3600 points of $[-3 : 0.1 : 3] \times [-3 : 0.1 : 3]$
11	F_{11}	[100,200,400] points of [-3,3]x[-3,3]	150 points of $[-3, 3] \times [-3, 3]$	3600 points of $[-3 : 0.1 : 3] \times [-3 : 0.1 : 3]$
12	F_{12}	[100,200,400] points of [-3,3]x[-3,3]	150 points of $[-3, 3] \times [-3, 3]$	3600 points of $[-3 : 0.1 : 3] \times [-3 : 0.1 : 3]$

Fig. 3. Results for n = 3

	p-value of t-test	testing error		size of best		running time		first generation.	
		GPLL	GPM	GPLL	GPM	GPLL	GPM	GPLL	GPM
F_1	**0.0001**	0.0193 ±0.0157	0.0105 ±0.0148	83.5300 ±45.7352	128.8900 ±59.6300	58.6800 ±66.6306	73.7200 ±26.4915	68.3200 ±44.9280	132.4300 ±31.1452
F_2	**0.0000**	0.0544 ±0.0364	0.0152 ±0.0124	82.0400 ±43.1572	149.3000 ±54.9895	44.3000 ±55.0319	88.8700 ±22.1691	43.7700 ±27.6094	141.0600 ±12.1421
F_3	0.7516	0.0163 ±0.0155	0.0066 ±0.0110	73.3800 ±38.2054	110.5300 ±64.0733	34.8100 ±49.1289	49.4400 ±23.3841	47.9400 ±31.8535	132.8300 ±30.2141
F_4	**0.0000**	0.0163 ±0.0155	0.0073 ±0.0054	73.3800 ±38.2054	118.8700 ±51.2070	34.8100 ±49.1289	74.9100 ±58.6421	47.9400 ±31.8535	131.3600 ±28.6302
F_5	**0.0047**	0.2062 ±0.2141	0.1309 ±0.1527	90.4600 ±56.2888	138.8200 ±53.6998	57.2200 ±68.2256	84.6300 ±26.2373	63.2600 ±47.2638	137.6400 ±18.2942
F_6	**0.0092**	0.2055 ±0.1897	0.1390 ±0.1655	101.1100 ±51.2398	128.5700 ±59.0400	71.8700 ±55.4077	77.7500 ±20.8076	86.2700 ±45.3576	127.4100 ±37.8978
F_7	**0.0000**	0.0295 ±0.0143	0.0196 ±0.0105	79.7900 ±44.3274	117.1300 ±51.7673	35.2300 ±38.3566	159.9300 ±57.3712	67.1100 ±41.0476	134.0900 ±30.8296
F_8	**0.0065**	0.5680 ±0.1350	0.5182 ±0.1203	24.5300 ±27.4449	83.4300 ±72.1029	6.6600 ±11.1021	90.1000 ±83.1028	37.0600 ±37.3198	110.8600 ±59.4218
F_9	**0.0000**	2.1922 ±0.8661	1.1590 ±0.5723	85.7200 ±41.4407	152.2300 ±53.5076	29.4000 ±23.4676	207.9200 ±62.2555	51.0600 ±20.6172	144.8300 ±7.6410
F_{10}	**0.0059**	0.6395 ±0.6080	0.4083 ±0.5632	90.6600 ±48.0581	127.2100 ±54.5544	42.3500 ±36.4177	170.9100 ±52.5563	79.8200 ±39.5258	133.0900 ±26.8374
F_{11}	**0.0000**	0.2865 ±0.1598	0.1536 ±0.1207	53.2200 ±31.1833	128.7900 ±43.5557	17.2200 ±16.4897	174.2000 ±50.2991	33.2500 ±22.0538	143.6400 ±7.7973
F_{12}	**0.0000**	0.7524 ±0.1928	0.5975 **±0.1400**	74.8900 ±40.7126	126.8800 ±37.6476	28.4300 ±35.3074	148.9200 ±41.6099	62.3900 ±39.8406	145.7900 ±5.3073

Fig. 4. Results for n = 9

	p-value of t-test	testing error		size of best		running time		first generation	
		GPLL	GPM	GPLL	GPM	GPLL	GPM	GPLL	GPM
F_1	0.2526	0.0135 ±0.0222	0.0105 ±0.0148	108.8300 ±52.9978	128.8900 ±59.6300	92.2300 ±64.7408	73.7200 ±26.4915	97.9600 ±41.3778	132.4300 ±31.1452
F_2	**0.0000**	0.0264 ±0.0204	0.0152 ±0.0124	130.1700 ±39.9538	149.3000 ±54.9895	106.6200 ±57.3514	88.8700 ±22.1691	93.8400 ±33.0628	141.0600 ±12.1421
F_3	*0.0105*	0.0035 ±0.0042	0.0066 ±0.0110	100.4600 ±51.0513	110.5300 ±64.0733	59.2800 ±40.7818	49.4400 ±23.3841	119.1500 ±27.5939	132.8300 ±30.2141
F_4	**0.0035**	0.0120 ±0.0150	0.0073 ±0.0054	101.5500 ±48.1686	118.8700 ±51.2070	74.0100 ±22.7119	74.9100 ±58.6421	80.6300 ±35.1547	131.3600 ±28.6302
F_5	0.0812	0.1735 ±0.1891	0.1309 ±0.1527	128.8100 ±55.7530	138.8200 ±53.6998	95.9600 ±56.4484	84.6300 ±26.2373	102.0300 ±37.1789	137.6400 ±18.2942
F_6	0.3000	0.1657 ±0.1948	0.1390 ±0.1655	108.6800 ±51.6047	128.5700 ±59.0400	92.1700 ±51.2228	77.7500 ±20.8076	110.2700 ±38.6794	127.4100 ±37.8978
F_7	0.2928	0.0211 ±0.0091	0.0196 **±0.0105**	115.3900 ±44.8105	117.1300 ±51.7673	65.0600 ±29.0895	159.9300 ±57.3712	120.9600 ±31.5027	134.0900 ±30.8296
F_8	0.5091	0.5301 ±0.1345	0.5182 ±0.1203	55.2400 ±58.3468	83.4300 ±72.1029	34.3000 ±44.8728	90.1000 ±83.1028	69.2300 ±48.3073	110.8600 ±59.4218
F_9	**0.0124**	1.4077 ±0.5621	1.1590 ±0.5723	142.7900 ±55.3865	152.2300 ±53.5076	92.7400 ±40.3319	207.9200 ±62.2555	121.4900 ±27.5199	144.8300 ±7.6410
F_{10}	0.6389	0.4454 ±0.5431	0.4083 ±0.5632	121.8600 ±58.1744	127.2100 ±54.5544	68.8500 ±35.8919	170.9100 ±52.5563	127.2500 ±28.6220	133.0900 ±26.8374
F_{11}	0.0724	0.1887 ±0.1480	0.1536 ±0.1207	104.2100 ±44.0409	128.7900 ±43.5557	75.2300 ±30.7406	174.2000 ±50.2991	99.5800 ±37.6915	143.6400 ±7.7973
F_{12}	0.0907	0.6356 **±0.1725**	0.5975 ±0.1400	109.5600 ±38.9556	126.8800 ±37.6476	59.2200 ±29.4030	148.9200 ±41.6099	113.1200 ±33.8882	145.7900 ±5.3073

In tables 3 and 4, if H_0 is accepted the p-value is printed in normal face, otherwise it is printed in bold face if GPM performs better than GPLL, or in italic face if the reverse is true.

It can be seen from the tables that stopping too eagerly in each layer severely degrades the generalization capacity of GP. When $n = 3$, on almost all functions, GPLL generalises worse than GPM. The results in the last two columns of Table 3 (showing the first generations, averaged over 100 runs, where the best solutions were found) explain this degradation. Setting $n = 3$ causes GPLL to stop prematurely, while GPM continues on and generally finds the best solutions close to the end of each run. Consequently, the average training time and solution complexity (measured in number of nodes) obtained by GPLL were significantly smaller than those of GPM.

When n was increased to 9 (table 4, the generalization performance of GPLL improved, achieving statistically indistinguishable generalization performance from GPM on 8 out of 12 functions.

For the univariate functions, the results are mixed, with GPLL giving better generalization on one function, GPM on two, and the other three being statistically indistinguishable (though GPLL generalizes slightly worse). GPLL finds rather less complex solutions, but takes longer to do so.

On the bivariate functions the results are more consistent. On almost all problems (except F9), the test set errors of GPLL and GPM are similar (H_0 is accepted). However, the training times for GPLL were substantially smaller than for GPM. The solutions found by GPLL were also somewhat smaller.

VI. Conclusion

In this paper, we have investigated the impact of layered learning with incremental sampling on GP learning efficiency. The experimental results on twelve benchmark symbolic regression problems indicate that while the impact is small for univariate functions it is more useful in the case of bivariate functions: layered learning with incremental sampling improves the training efficiency of GP by reducing the training (learning) time, while maintaing the quality of the solutions and reducing their complexity.

The stopping criterion for learning in each layer used in this paper is a rather simplified version of that presented in [23]. In future, we plan to test more sophisticated stopping criteria based on the full version of the algorithm. Testing GPLL on more complicated real-world problems is also an objective of our near-future work.

VII. Acknowledgment

This work was funded by The Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01.14.09. The first author would like to thank the school of Information Technology, Le Quy Don University for providing financial support for her travel to CEC'2011 to present this paper.

References

[1] Asada, M., Noda, S., Tawaratsumida, S., Hosoda, K.: Purposive behaviour acquisition for a real robot by vision-based reinforcement learning. Machine Learning - Special issue on robot learning archive **23**, 279 – 303 (1996)

[2] Costa, L.E.D., Landry, J.: Relaxed genetic programming. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, vol. 1, pp. 937–938. ACM Press, Seattle, Washington, USA (2006)

[3] Costelloe, D., Ryan, C.: On improving generalisation in genetic programming. In: Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009, *LNCS*, vol. 5481, pp. 61–72. Springer, Tuebingen (2009)

[4] Francone, F., Nordin, P., Banzhaf, W.: Benchmarking the generalization capabilities of a compiling genetic programming system using sparse data sets. In: Genetic Programming 1996: Proceedings of the First Annual Conference, pp. 72–80. MIT Press, Stanford University, CA, USA (1996)

[5] Gagné, C., Schoenauer, M., Parizeau, M., Tomassini, M.: Genetic programming, validation sets, and parsimony pressure. In: Proceedings of the 9th European Conference on Genetic Programming, *Lecture Notes in Computer Science*, vol. 3905, pp. 109–120. Springer, Budapest, Hungary (2006)

[6] Garis, H.: Genetic programming artificial nervous systems artificial embryos and embryological electronics. In: Parallel Problem Solving from Nature, *Lecture Notes in Computer Science*, vol. 496, pp. 117–123. Springer Berlin / Heidelberg (1991)

[7] Gustafson, S., Burke, E.K., Krasnogor, N.: On improving genetic programming for symbolic regression. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 1, pp. 912–919. IEEE Press, Edinburgh, UK (2005)

[8] Gustafson, S.M.: Layered learning in genetic programming for a cooperative robot soccer problem. Master's thesis, Kansas State University, Manhattan, KS, USA (2000)

[9] Iba, H.: Bagging, boosting, and bloating in genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1053–1060. Morgan Kaufmann, Orlando, Florida, USA (1999)

[10] Jackson, D., Gibbons, A.: Layered learning in boolean GP problems. In: Proceedings of the 10th European Conference on Genetic Programming, *Lecture Notes in Computer Science*, vol. 4445, pp. 148–159. Springer, Valencia, Spain (2007)

[11] Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: Genetic Programming, Proceedings of EuroGP'2003, *LNCS*, vol. 2610, pp. 70–82. Springer-Verlag, Essex (2003)

[12] Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992)

[13] Kushchu, I.: Genetic programming and evolutionary generalization. IEEE Transactions on Evolutionary Computation **6**(5), 431–442 (2002)

[14] Mahler, S., Robilliard, D., Fonlupt, C.: Tarpeian bloat control and generalization accuracy. In: Proceedings of the 8th European Conference on Genetic Programming, *Lecture Notes in Computer Science*, vol. 3447. Springer, Lausanne, Switzerland (2005)

[15] Mitchell, T.M.: Machine Learning. McGraw-Hill (1997)

[16] Muggleton, S.: Optimal layered learning: A pac approach to incremental sampling. In: Proceedings of the Fourth International Workshop on Algorithmic Learning Theory, pp. 37–44. Springer (1993)

[17] Nguyen, Q.U., Nguyen, T.H., Nguyen, X.H., O'Neill, M.: Improving the generalisation ability of genetic programming with semantic similarity based crossover. In: Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010, *LNCS*, vol. 6021, pp. 184–195. Springer, Istanbul (2010)

[18] Panait, L., Luke, S.: Methods for evolving robust programs. In: Genetic and Evolutionary Computation – GECCO-2003, *LNCS*, vol. 2724, pp. 1740–1751. Springer-Verlag, Chicago (2003)

[19] Paris, G., Robilliard, D., Fonlupt, C.: Exploring overfitting in genetic programming. In: Evolution Artificielle, 6th International Conference, *Lecture Notes in Computer Science*, vol. 2936, pp. 267–277. Springer, Marseilles, France (2003)

[20] Poli, R., Langdon, W., McPhee, N.: A field guide to genetic programing (2008)

[21] Stone, P.: Layered learning in multi-agent systems. Ph.D. thesis, Carnegie Mellon University (1998)

[22] Stone, P., Veloso, M.: Layered learning. In: Proceedings 17th International Conference on Machine Learning, pp. 369–381. Springer-Verlag, Budapest, Hungary (2006)
[23] Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation, pp. 877–884. ACM, Portland, Oregon, USA (2010)
[24] Vanneschi, L., Gustafson, S.: Using crossover based similarity measure to improve genetic programming generalization ability. In: GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 1139–1146. ACM, Montreal (2009)
[25] Zhang, B., Joung, J.: Genetic programming with incremental data inheritance. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2, pp. 1217–1224. Orlando, Florida, USA (1999)
[26] Zhang, B.T.: Bayesian methods for efficient genetic programming. Genetic Programming and Evolvable Machines **1**(3), 217–242 (2000)
[27] Zhang, B.T., Mühlenbein, H.: Balancing accuracy and parsimony in genetic programming. Evolutionary Computation **3**(1), 17–38 (1995)