Bagging, Boosting, and Bloating in Genetic Programming

Hitoshi Iba
Dept. of Information and Communication Engineering,
School of Engineering,
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, 113-8656, Japan
Tel: +81 3 3812-2111 ext.7424, Fax: +81 3 5689 7231,
email: iba@miv.t.u-tokyo.ac.jp

Abstract

We present an extension of GP (Genetic Programming) by means of resampling techniques, i.e., Bagging and Boosting. These methods both manipulate the training data in order to improve the learning algorithm. In theory they can significantly reduce the error of any weak learning algorithm by repeatedly running it. This paper extends GP by dividing a whole population into a set of subpopulations, each of which is evolvable by using the Bagging and Boosting methods. The effectiveness of our approach is shown by experiments. The performance is discussed by the comparison with the traditional GP in view of the bloating effect.

1 Introduction

This paper presents an extension of GP (Genetic Programming) by means of resampling techniques. Our motivation is to enhance the robustness of GP. The robustness is an important feature of an evolved program [Ito *et al.*96]. It is defined as the ability to cope with noisy or unknown situations. For example, if we make an attempt to evolve a robot problem, the robustness could be examined by testing it for a variety of robot tasks. In pursuit of the robustness, we verify the validness of an evolved program for testing data, which are different from the training data.

Boosting and bagging are general resampling methods for improving the performance of any learning algorithm [Freund and Schapire96]. Both methods work by repeatedly running a given weak learning algorithm on various distributions over the training data. They rely on a resampling technique to obtain a different training set for each classifier.[1] Boosting can theoretically be used to significantly reduce the error of any weak learning algorithm which need only be a little bit better than random guessing. The resulting classifier has been shown to be more accurate than any of the individual classifiers making up the ensemble [Maclin and Opitz97].

While previous work has demonstrated that the Bagging and Boosting methods are very effective for decision trees [Quinlan96] or neural networks [Maclin and Opitz97], there has been little empirical testing with other adaptive methods. In this paper we extend GP by means of the Bagging and Boosting methods and establish systems called BagGP and BoostGP. The salient features of BagGP and BoostGP are as follows:

1. The whole GP population is divided into a set of subpopulations.
2. Each subpopulation is independently evolvable by using the fitness values derived from the training data, which are given by the resampling techniques.
3. The best individuals from each of the subpopulations vote in order to form a composite tree output for the test data.

We provide comparative experiments to show how successfully Bagging and Boosting techniques are applied to improve the robustness of GP. [Bauer and Kohavi98] measured the sizes of trees when Bagging was used for decision trees, and showed an interesting positive correlation between the average tree size and its success in reducing the error. We

[1] In this paper, we mean the pattern recognition by the classifier. For instance, it includes not only a classifier used in the classifier system, but an acquired tree by GP, a decision tree, and a neural network structure as well.

believe that controlling the bloating effect is closely related to the performance improvement in the case of GP. Thus, we compare the tree growth rates for the traditional GP, BagGP, and BoostGP, which shows that the resampling techniques are successful in reducing the tree size.

The rest of this paper is structured as follows. Section 2 describes the principle of the Bagging and Boosting techniques. Section 3 introduces a basic idea of extending GP by means of these methods in order to construct BagGP and BoostGP. Sections 4 and 5 show the experimental results with Boolean function generation, chaotic time series prediction, symbolic regression, and financial data prediction. The performance is compared with the traditional GP. Section 6 discusses our approach, followed by some conclusions in Section 7.

2 Bagging and Boosting

Following [Quinlan96] and [Drucker97], this section explains the basic ideas of Bagging and Boosting. The data for learning systems are supposed to consist of attribute-value vectors or instances. Both Boosting and Bagging manipulate the training data in order to generate different classifiers.

2.1 Bagging

Bagging produces replicate training sets by sampling with replacement from the training instances. For each trial $t = 1, 2, \cdots, T$, a training set of size N is sampled with replacement from the original instances. This training set is the same size as the original data, but some instances may not appear in it while others appear more than once. The learning system generates a classifier C^t from the sample and the final classifier C^* is formed by aggregating the T classifiers from these trials. To classify an instance x, a vote for class k is recorded by every classifier for which $C^t(x) = k$ and C^* is then the class with the most votes (ties being resolved arbitrarily).

2.2 Boosting

Boosting uses all instances at each repetition, but maintains a weight for each instance in the training set that reflects its importance Adjusting the weights causes the learner to focus on different instances and leads to different classifiers. When voting to form a composite classifier, each component classifier has the same vote in Bagging, whereas Boosting assigns different voting strengths to component classifiers on the basis of their accuracy.

Let w_x^t denote the weight of instance x at trial t. Initially, $w_x^1 = 1/N$ for every x. At each trial $t = 1, 2, \cdots, T$, a classifier C^t is constructed from the given instances under the distribution w^t, as if the weight w_x^t of instance x reflects its probability of occurrence. The error ϵ^t of this classifier is also measured with respect to the weights and consists of the sum of the weights of the instances that it misclassified. If ϵ^t is greater than 0.5, the trials are terminated, and T is altered to $t-1$. Conversely, if C^t correctly classifies all instances so that ϵ^t is zero, the trials terminate and T becomes t. Otherwise, the weight vector w^{t+1} for the next trial is generated by (1) multiplying the weights of instances that C^t classifies correctly by the factor $\beta^t = \epsilon^t/(1-\epsilon^t)$ and (2) renormalizing so that $\sum_x w_x^{t+1}$ is 1. The boosted classifier C^* is obtained by summing the votes of the classifiers $C^1, C^2, \cdots, C^T$, where the vote for classifier C^t is worth $\log(1/\beta^t)$ units.

[Freund and Schapire96] introduced a new boosting algorithm, called AdaBoost, which theoretically can significantly reduce the error of any learning algorithm if its performance is a little better than random guessing. We use a version of AdaBoost, i.e., AdaBoost.R [Drucker97], so as to extend GP in next section.

3 BagGP and BoostGP

We divide the whole GP population into a set of subpopulations $\{P_1, P_2, \cdots, P_T\}$, in order to use the Boosting and Bagging techniques. When GP is applied to evolving individuals in each subpopulation P_t, the training set TR_t is sampled by means of the sampling method.

BagGP uses the Bagging method, in which a training set of size N is sampled with replacement from the original instances for each subpopulation. Best individuals from each of the subpopulations vote in order to form a composite output for the testing data. The output value associated with most votes is considered as the final output. More precisely, we use the weighted median described below (**Step** 10) for this derivation.

BoostGP uses the following sampling method based on AdaBoost.R [Drucker97]. The following procedure (**Step** 2∼**Step** 8) is repeated from the first subpopulation P_1 to the last subpopulation P_T.

Step 1 Let t be set to 1. Each training pattern is assigned an equal weight, i.e., $w_i = 1$ for $i = 1, \cdots, N$, where N is the total number of the training data.

Step 2 The probability that a training sample i is included in the training set TR_t is $p_i = w_i / \sum w_i$, where the summation is over all members of the training set. Pick N samples with replacement to form the training set.

Step 3 Apply GP to the individuals in the subpopulation P_t with the above training set TR_t. The best evolved tree makes a hypothesis, i.e, $h_t : x \rightarrow y$.

Step 4 Pass every member of the training set TR_t through this tree h_t to obtain a prediction $y_i^{(p)}(x_i)$ for $i = 1, \cdots, N$.

Step 5 Calculate a loss $L_i = L(\mid y_i^{(p)}(x_i) - y_i \mid)$ for each training sample. The loss function L is defined later.

Step 6 Calculate the averaged loss, i.e., $\overline{L} = \sum_{i=1}^{N} L_i p_i$.

Step 7 Calculate the measure of confidence in the predictor, i.e., $\beta = \frac{\overline{L}}{1-\overline{L}}$.

Step 8 Update the weights by using $w_i := w_i^{1-\overline{L_i}}$.

Step 9 $t := t + 1$. If $t \leq T$, then go to **Step** 2.

Step 10 For a particular input x_i in the test data, each of the T acquired trees, i.e., the best evolved individuals from the subpopulations, makes a prediction h_t for $t = 1, \cdots, T$. Obtain the cumulative prediction h_f using the T predictors:

$$h_f = \min\{y \in Y : \sum_{t:h_t \leq y} \log(1/\beta_t) \geq \sum_{t} \log(1/\beta_t)\}. \tag{1}$$

In **Step** 5, the loss function L may be of any form as long as $L \in [0, 1]$. We use the following loss function,

$$L_i = \frac{\mid y_i^{(p)}(x_i) - y_i \mid}{\max_{i=1,\cdots,N} \mid y_i^{(p)}(x_i) - y_i \mid}. \tag{2}$$

β is a measure of confidence in the predictor, i.e., the lower the β value is, the higher the confidence. In **Step** 9, the smaller the loss is, the more the weight is reduced, which makes the probability smaller that this pattern will be chosen as a member of the training set for the next subpopulation. **Step** 10 represents the weighted median. Suppose that each best tree h_t has a prediction $y_i^{(t)}$ on the i-th pattern and an associated β_t value, and that the predictions are relabeled for pattern i as follows:

$$y_i^{(1)} < y_i^{(2)} < \cdots < y_i^{(T)}. \tag{3}$$

Then sum the $\log(1/\beta^t)$ until the smallest t is reached so that the inequality is satisfied. The prediction from that best tree t is taken to be the final prediction. If all β_t's are equal, it is the same as the median calculation.

4 Experimental Results

This section studies the performance of BagGP and BoostGP empirically. Suppose that the total population size is set to be N_{psize}. This is divided into 10 subpopulations of size $N_{psize}/10$ for running BagGP and BoostGP. For the sake of comparison, we also experiment in running the canonical GP, i.e., a standard GP without any sampling method for the single population of size N_{psize}. We have chosen sgpc1.1, a simple GP in the C language, as the canonical GP. The parameters used for the experiments are shown in Table 1.

GP is applied to the following problems:

Experiment 1 Discovery of trigonometric identities.
Koza used GP to find a new mathematical expression, in symbolic form, that equals a given mathematical expression, for all values of its independent variables [Koza92, ch.10.1]. In the following experiment, our goal is to discover trigonometric identities, such as

$$\cos 2x = 1 - \sin^2 x. \tag{4}$$

The training data consist of 100 pairs, i.e., $\{(x, \cos 2x + \Re)\}$, in which a random noise $\Re$ between –0.01 and 0.01 is added to the precise value. The testing data consist of 100 input-output pairs, i.e., $\{(x, \cos 2x)\}$, where no random noise is added to the output value.

Experiment 2 Predicting a chaotic time series.
The Mackey-Glass time series is generated by integrating the following delay differential equation and is used as a standard benchmark for prediction algorithms:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1 + x^{10}(t-\tau)} - bx(t), \tag{5}$$

with $a = 0.2$, $b = 0.1$ and $\tau = 17$. The trajectory is chaotic and lies on an approximately 2.1-dimensional strange attractor.

For the sake of comparison, all the parameters chosen were the same as those used in the previous study

Parameters of sgpc1.1	Problem Name		
	$\cos(2x)$ (Exp.1)	Chaos (Exp.2)	6-multiplexer (Exp.3)
Terminal Symbols	{X,1.0}	$\{X(t-1), X(t-2), \cdots, X(t-10), \Re\}$	$\{a_0, a_1, d_0, d_1, d_2, d_3\}$
Nonterminal Symbols	$\{+, -, *, \%, \sin\}$	$\{+, -, \times, \%, \sin, \cos, \exp_{10}\}$	{AND, OR, NAND, NOR}
no. of subpopulations	10	10	10
subpopulation_size	200	1024	1024
max_depth_for_new_trees	6	6	6
max_depth_after_crossover	17	17	17
max_mutant_depth	4	4	4
grow_method	GROW	GROW	GROW
selection_method	TOURNAMENT	TOURNAMENT	TOURNAMENT
tournament_K	6	6	6
crossover_func_pt_fraction	0.1	0.1	0.1
crossover_any_pt_fraction	0.7	0.7	0.7
fitness_prop_repro_fraction	0.1	0.1	0.1

Table 1: GP Parameters

[Oakley94, p.380, Table17.3], except that the terminal set consisted of ten past data for the short-term prediction (see Table 1). We use the first 100 time sequences for the training data, and the next 400 time sequences for the testing data.

Experiment 3 Boolean concept formation (6-multiplexer).
To show the effectiveness as a Boolean concept learner, we conducted a simple experiment (6-multiplexer), in which the goal function is the multiplexer function of 6 variables:

$$f(a_0, a_1, d_0, d_1, d_2, d_3) = \overline{a}_0\overline{a}_1 d_0 \vee a_0\overline{a}_1 d_1 \vee \overline{a}_0 a_1 d_2 \vee a_0 a_1 d_3 \quad (6)$$

The inputs to the Boolean 6-multiplexer function consists of 2 address bits a_i and 2^2 data bits d_i. The output is the Boolean value of the particular data bit that is singled out by the address bits of the multiplexer. For example, if $a_0 = 1$ and $a_1 = 0$, then the multiplexer singles out the data bit d_2 because $10_2 = 2$.

For both the training and testing data sets, we use the whole set of 64 pairs of 6 binary input variables, i.e., $\{(a_0, a_1, d_0, d_1, d_2, d_3) \in \{0,1\}^6\}$ and their output values derived by eq.(6).

Fig.1 shows the experimental results, which plots the mean square errors (MSE) for the testing data with generations, averaged over 20 runs. Fig.2 plots the standard fitness values of BoostGP with generations. As shown in the figure, the BoostGP was able to track the oscillation and adapt to a new training data quickly. Table 2 summarizes the experimental results. The table compares the MSE values for the test data at

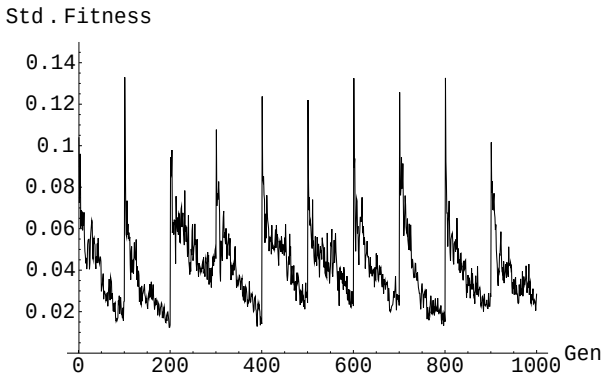
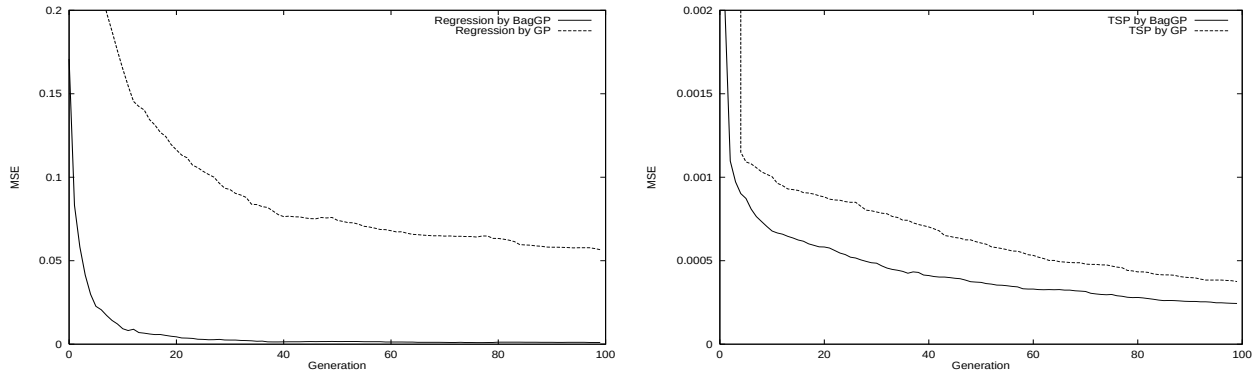


Figure 2: Regression by BoostGP.

the final generation for Experiments 1 and 2. For Experiment 3, the success generations to obtain the solution tree are listed. All these values are averaged over 20 runs. This comparison clearly shows the superiority of BagGP and BoostGP over the standard GP. The robustness of evolved trees by BagGP and BoostGP was enhanced in the sense that the validate fitness values, i.e., the mean square errors for the testing data, were reduced. The performance difference between BagGP and BoostGP was not statistically significant.

5 Application to a Financial Task

This section explains how BagGP and BoostGP are applied to a harder real-world problem, i.e., the price data prediction in Japanese stock market. The detailed task description is given in [Iba and Sasaki99]. The target financial data is the stock price average of Tokyo Stock Exchange, which is called Nikkei225. The Nikkei225 average is reported every minute by the Ni-



(a) Regression. (b) Time Series Prediction.

Figure 1: Experimental Results (Generation vs. MSE)

Problem Name	cos(2x) (Exp.1) MSE	Chaos (Exp.2) MSE	6-multiplexer (Exp.3) Success Gen.
GP	0.056606	0.000375	5.15
BagGP	0.001051	0.000244	4.80
BoostGP	0.019296	0.000237	4.88

Table 2: Summary of Experimental Results

hon Keizai Shimbun-Sha, a well-known financial newspaper publishing firm. The derivation is based upon the Dow formula. As of Feb.,10th,1999, the Nikkei average stood at 13960 yen. The data we use in the following experiments span over a period from April 1st 1993 to September 30th 1993. Fig.3 shows the example tendency of the Nikkei225 average during the above period. All data are normalized between 0.0 and 1.0 as the input value. The total number of data is 33,177. We use the first 3,000 time steps for the training data and the rest for the testing data.

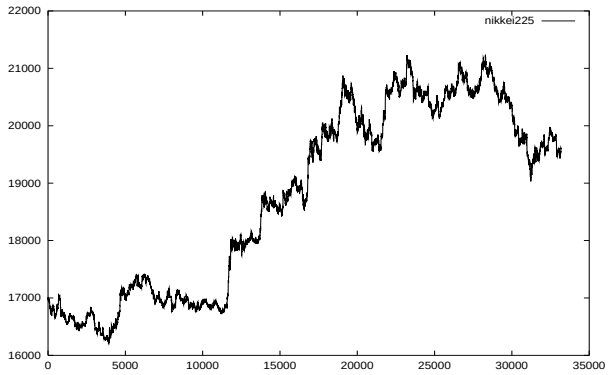


Figure 3: Nikkei225 Data.

We chose the same GP parameters as those for Exp.2 (see Table.1). The terminal set was $\{\mathrm{v1}, \cdots, \mathrm{v10}, \mathrm{r1}, \cdots, \mathrm{r10}, \Re\}$, where the terminals v$i$ and ri were defined as follows:

$$\mathrm{v}i = |x(t-i) - x(t-i-1)|$$

$$\mathrm{r}i = \frac{x(t-i) - x(t-i-1)}{x(t-i-1)}.$$

This terminal set is among the best ones given from the previous comparative study (see [Iba and Sasaki99] for details). The predicted value, i.e., the target output of a GP tree, is the difference between the current Nikkei225 price average and the price observed one minute before. Thus, the fitness value is defined to be the mean square error of the predicted value and the target data. The smaller fitness value, the better.

In order to confirm the validness of the predictor acquired by GP, we examined the best evolved tree with the stock market simulation during the testing period. We use the following rule to choose the dealing, i.e., to decide whether to buy or sell a stock based upon the prediction. Let the predicted output and the observed price data at time t be $\widetilde{Pr}(t)$ and $Pr(t)$, respectively. Remember that the output prediction of a GP tree is the difference between the current Nikkei225 price average and the price observed one minute before.

Step1 Initially, the total budget BG is set to be 1,000,000 yen. Let the time step t be 3000, i.e., the beginning of the testing period. The stock flag ST is set to be 0.

Step2 Derive the output, i.e., the predicted Nikkei225 average, of the GP tree.

Step3 If $0 < \widetilde{Pr}(t)$ and $ST = 0$, then buy the stock. That is, set ST to be 1.

Step4 Else, if $0 > \widetilde{Pr}(t)$ and $ST = 1$, then sell the stock. That is, set ST to be 0.

Step5 If $ST = 1$, let $BG := BG + Pr(t) - Pr(t-1)$.

Step6 If $BG < 0$, then return 0 and stop.

Step7 If $t < 33,177$, i.e., the end of the testing period, then $t := t + 1$ and go to **Step**2. Else return the total profit, i.e., $BG - 1,000,000$ yen.

The stock flag ST indicates the state of holding stock, i.e., if $ST = 0$, then no stock is shared at present, whereas if $ST = 1$, then a stock is shared. In **Step**5, the total property is derived according to the newly observed stock price. The satisfaction of the **Step**6 condition means that the system has gone into bankruptcy.

The above dealing rules were used for the validation of the acquired GP tree. For the sake of simplicity, we put the following assumptions on the market simulation:

1. At most one stock is shared at any time.
2. The dealing stock is imaginary, in the sense that its price behaves exactly the same way as the Nikkei225 average price.

The optimal profit according to the above dealing rule is 80,106.63 yen. This profit is ideally gained when the prediction is perfectly accurate during the testing period.

GP run was repeated under each condition, i.e., BagGP, BoostGP, and standard GP, over 10 times. The validation performance is shown in Table 3. The hit percentage means how accurately the GP tree made an estimate of the qualitative behavior of the price. That is, the hit percentage is calculated as follows:

$$\begin{aligned} \text{hit} &= \frac{N_{\uparrow_{-}\uparrow} + N_{\downarrow_{-}\downarrow}}{N_{\uparrow_{-}\uparrow} + N_{\uparrow_{-}\downarrow} + N_{\downarrow_{-}\uparrow} + N_{\downarrow_{-}\downarrow}} \\ &= \frac{N_{\uparrow_{-}\uparrow} + N_{\downarrow_{-}\downarrow}}{30,177}, \end{aligned}$$

where $N_{\uparrow_{-}\uparrow}$ means the number of times when the tree makes an upward tendency while the observed price rises, and $N_{\downarrow_{-}\uparrow}$ means the number of times when the tree makes a downward tendency while the observed price falls, and so on. The total number of the predictions is 30,177, which equals the number of testing data.

Fig.4 shows a typical prediction result of the normalized Nikkei225 price with the best evolved tree. The predicted difference between the current Nikkei225 price and the price one minute before is plotted. The predicted difference corresponds to the observed qualitative behavior, i.e., the upward or downward tendency, of the Nikkei225 price. This causes the high profit gain shown in Table.3. As can be seen in the table, the best performance by BagGP and BoostGP seems to be slightly better or almost the same as the standard GP. However, both BoostGP and BagGP are clearly superior to the standard GP in terms of the averaged performance. We believe that this is due to the robustness of the proposed approach. Although the performance difference between BoostGP and BagGP is not necessarily clear, yet the BoostGP result is a little disappointing to us. We have found that the simple loss function such as eq.(2) could be improved to pick up the training data more effectively. For instance, this function can be adaptively changed as generations proceed. We are working upon this extension as a future research topic.

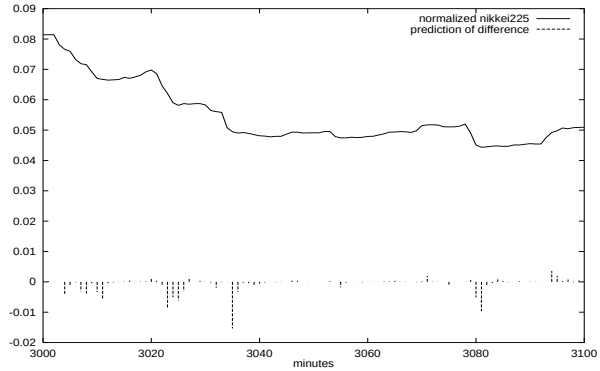


Figure 4: Prediction Result

6 Discussion

6.1 Related Works

[Gathercole and Ross97] investigated the use of small populations in GP. They proposed a method called DSS (Dynamic Subset Selection) to pick up a subset of training cases from the full training set for each generation. Each GP generation is evaluated using only

Table 3: Experimental Results (Financial Data Prediction)

Condition	Hit(%)		Profit gain(yen)	
	Average	Best	Average	Best
GP	0.619	0.625	29489.1	30765.6
BagGP	0.624	0.625	30749.5	30950.5
BoostGP	0.622	0.624	30410.5	30827.6

this subset. The cases are selected according to how difficult they are, i.e., how often they were misclassified by the population the last time they were selected and how old they were, i.e., how many generations since the last time they were selected. Older and more difficult cases have a greater likelihood of being selected to be in the next subset. They showed that GP+DSS with the small population size consistently produced a better answer using fewer tree evaluations than other runs using much large populations for two classification problems. DSS is closely related to the training set selection of Boosting method. However, the aging factor is not included in Boosting. Its effectiveness remains to be seen as a future research topic.

[Teller and Veloso95] proposed a new technique for an object recognition task. They established a system called PADO (Parallel Algorithm Discovery and Orchestration), which used an evolutionary strategy to solve the signal-to-symbol problem, i.e., the task of converting raw sensor data into a set of symbols that the data can be seen as representing. The object recognition for C classes is accomplished in PADO by the orchestration of C different systems. Each of these systems is composed of the S fittest programs from the corresponding group, i.e., subpopulation, of the current generation. Orchestration played a crucial role in the PADO architecture. It is very similar to a voting method used in BagGP and BoostGP. PADO does object recognition by orchestrating the responses of some systems. The confidence response of each system is initially weighted equally and the maximum confidence wins. During the first few test images, the weights are adjusted by telling PADO after its guess whether it was right or wrong. Although the orchestration method is not clearly defined as a weight vector tuning, their experimental results are very impressive considering that PADO does not require any help from users or domain specific information of any kind.

6.2 Bloating Effect

Many researchers have suggested that GP runs stagnate as a result of the bloating, i.e., the exponential growth of the introns or non-functional codes. This means that limiting the code sizes is essential because (1) smaller programs tend to show better generalization performance, and (2) shorter programs require less time and less space to run. Modifying the fitness function to include a penalty term that incorporates the tree size is one of the promising ways [Iba *et al.*94],[Zhang and Mühlenbein95]. We believe that BoostGP and BagGP provide another way to control the bloating effect by means of the resampling techniques. To confirm this, we plotted the number of nodes of a best tree with generations for a typical run of Experiment 1 (see Fig.5). This figure clearly shows that BoostGP and BagGP succeeded in limiting the code size at later generations, which leads to the improvement of GP search efficiency.

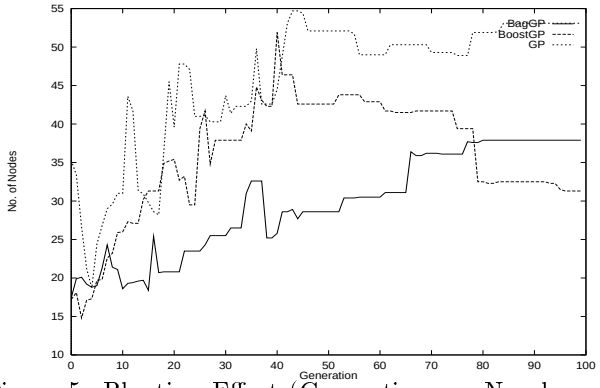


Figure 5: Bloating Effect (Generation vs. Number of Nodes)

6.3 Boosting vs. Co-evolutionary Method

Boosting is related to the co-evolving technique in evolutionary computation. For instance, [Hillis91] showed the addition of co-evolving parasites could improve the optimizing procedure by preventing the system from sticking at local maxima. His target task was to evolve a minimum sorting network, which is a parallel device for sorting lists with a fixed number of elements. In his experiment, the sorting networks were viewed as hosts, and the test cases (i.e., lists of numbers) as parasites. The fitness of a network was the percentage of test cases in the parasite that it sorted correctly. The fitness of a parasite was the percentage of its test cases that stumped the network. In his system, both pop-

ulations of networks and parasites were evolved, i.e., co-evolved, to beat each other, which resulted in the improvement of the search efficiency.

One key difference from co-evolution is that Boosting is only a resampling technique and has no mechanism to produce novel training data. The weight derivation in Boosting is correspondent to the fitness calculation in co-evolution method. However, there is no ensemble process, i.e., voting, in the co-evolution. The effectiveness of integrating these two methods remains to be seen, which is our future research concern.

7 Conclusion

This paper presented an extension of GP by means of resampling techniques, i.e., Bagging and Boosting, and established the systems called BagGP and BoostGP. In both systems, the whole population was divided into a set of subpopulations, each of which was evolvable by using Bagging and Boosting. The effectiveness of our approach was shown by experiments.

One of the target tasks for GP is the classification, such as Boolean function generation, system identification, and pattern recognition, which has been intensively studied in machine learning literatures as well. Although many evolutionary or adaptive techniques were proposed to solve this class of problems, there have been very few studies to improve the search efficiency in view of the machine learning technique. Boosting and Bagging are a general method to improve and analyze the learning performance. We believe this paper is a step toward the integration of GP and the theoretical field, i.e. machine learning.

Acknowledgments.
We are grateful to Institute of Investment Technology, Nikko Securities Co., Ltd., for providing the Nikkei225 stock price data. We also thank anonymous reviewers for helpful comments.

References

[Bauer and Kohavi98] Bauer,E., and Kohavi,R., An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants, in *Machine Learning*, vol.20, pp.1-33, 1998

[Drucker97] Drucker,H., Improving Regression using Boosting Techniques, *Proc. of International Conf. on Machine Learning (ICML97)*, 1997

[Freund and Schapire96] Freund,Y., and Schapire,R.E., Experiments with a New Boosting Algorithm, *Proc. of International Conf. on Machine Learning (ICML96)*, 1996

[Gathercole and Ross97] Gathercole,C., and Ross,P., Small Populations over Many Generations Can Beat Large Populations over Few Generations in Genetic Programming, *Proc. of Genetic Programming 1997 (GP97)*, 1997

[Hillis91] Hillis,W.D., Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure, in em Artificial Life ll, SFI Studies in the Sciences of Complexity,vol.X, Langton,C.G., Taylor,C., Farmer,J.D., and Rasmussen,S., (eds.), Addison-Wesley,1991

[Iba *et al.*94] Iba, H., deGaris,H., and Sato,T., Genetic Programming using a Minimum Description Length Principle, in *Advances in Genetic Programming*, (ed. Kenneth E. Kinnear, Jr.), pp.265–284, MIT Press, 1994

[Iba and Sasaki99] Iba, H., Sasaki,T., Using Genetic Programming to Predict Financial Data, in *Proc. 1999 Congress on Evolutionary Computation (CEC99)*, 1999

[Ito *et al.*96] Ito,T., Iba,H. and Kimura,M., Robustness of Robot Programs Generated by Genetic Programming, in *Genetic Programming 96*, MIT Press, 1996

[Koza92] Koza,J.R., Genetic Programming, On the Programming of Computers by means of Natural Selection, MIT Press, 1992

[Maclin and Opitz97] Maclin,R., and Opitz,D., An Empirical Evaluation of Bagging and Boosting, *Proc. of National Conf. on Artificial Intelligence (AAAI97)*, 1997

[Oakley94] Oakley, H., Two Scientific Applications of Genetic Programming: Stack Filters and Non-Linear Equation Fitting to Chaotic Data, in *Advances in Genetic Programming*, (ed. Kenneth E. Kinnear, Jr.), MIT Press, 1994

[Quinlan96] Quinlan,J.R., Bagging, Boosting, and C4.5, *Proc. of National Conf. on Artificial Intelligence (AAAI96)*, 1996

[Teller and Veloso95] Teller,A., and Veloso,M., PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System, CMU-CS-95-101, School of Computer Science, Carnegie Mellon University, 1995

[Zhang and Mühlenbein95] Zhang,B. and Mühlenbein,H., Balancing Accuracy and Parsimony in Genetic Programming, *Evolutionary Computation*, vol.3, no.1, 1995