Bayesian Genetic Programming

Byoung-Tak Zhang
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea
http://scai.snu.ac.kr/~ btzhang

1 Introduction

Genetic programming (GP) provides a powerful tool for learning models of some unknown process from specific fitness cases. Models are typically represented as tree-structured programs, and thus GP has a wide range of application domains. However, the general applicability of GP suffers from large amount of *space* and *time* required for generating intermediate solutions.

Space requirements of GP are proportional to the product of population size and the size of each program. Given a fixed population size, space requirements can be reduced by minimizing the program size at each generation [3, 4, 5, 11].

Time requirements of GP are proportional to the product of the population size, individual program size, data size, and the number of generations. Other parameters being equal, the evolution time can be minimized by reducing the effective data size for each generation [1, 2, 6].

In this article, we present a class of GP methods that attempt to evolve *compact* programs *fast* without loss of their *generalization* accuracy. All these methods are based on the Bayesian evolutionary framework (see [8] for more details) and uses *adaptive fitness functions* that take into account the dynamics of evolutionary processes.

By Bayesian evolution we mean the evolutionary process in which Bayes theorem is used to estimate the posterior fitness of individuals from their prior fitness values. Offspring individuals are then generated by sampling from the posterior distribution combined with the transition probabilities formed by genetic variation operators. The evolutionary inference steps from the prior via posterior distribution of parent fitness to the expected fitness distribution of offspring are essential elements in Bayesian evolution.

In the following, we briefly describe the Bayesian evolutionary algorithm for genetic programming (Section 2) and its applications to *program growth control* (Section 3) and acceleration of evolution speed by *data selection* (Section 4). Further applications are discussed in Section 5.

2 Bayesian Formulation of Genetic Programming

Genetic programming works by initializing a population of programs and iteratively producing the next generation $\mathcal{A}(g)$ of fitter programs:

$$\mathcal{A}(g) = \{A_i^g\}_{i=1}^M, \tag{1}$$

where A_i^g is the ith program at generation g and M is the population size. Genetic operators such as mutation and crossover are used to produce offspring programs from the parent programs. New generations are produced repeatedly until some termination condition is satisfied.

The goodness or fitness of a program is measured in terms of a set of fitness cases or a training data

$$D = \{(\mathbf{x}_c, y_c)\}_{c=1}^N, \tag{2}$$

and the programs can be considered as a model of the unknown process f generating the data with some noise $\epsilon(\mathbf{x}_c)$: $y_c = f(\mathbf{x}_c) + \epsilon(\mathbf{x}_c)$.

We define the *best* program as the *most probable* model, given the data D plus the prior knowledge on the problem domain. Bayes theorem provides a direct method for calculating such probabilities. It states that the posterior (i.e. after observing the data D) probability of a model A is

$$P(A|D) = \frac{P(D|A)P(A)}{P(D)} \tag{3}$$

$$= \frac{P(D|A)P(A)}{\int_{\mathcal{A}} P(D|A)P(A)dA}. \tag{4}$$

where $\mathcal{A}$ is the space of *all* possible models and the integral symbol is to be replaced by a summation in case of A taking discrete values. Here $P(A)$ is the prior (i.e. before observing the data) probability distribution for the models, and $P(D|A)$ is the likelihood of the model for the data.

Bayesian evolutionary algorithms (BEAs) [8], i.e. Baysian genetic programming in this case, approximate the posterior probability distribution (4) by a fixed-size population $\mathcal{A}(g)$ which is typically a small *subset* of the entire model space $\mathcal{A}$. That is, the fitness value of model A_i, i.e. its posterior probability $P_g(A_i|D)$, is computed with respect to the gth population $\mathcal{A}(g)$:

$$P_g(A_i|D) = \frac{P(D|A_i)P(A_i)}{\sum_{A_j \in \mathcal{A}(g)} P(D|A_j)P(A_j)}. \quad (5)$$

The posterior distribution $P_g(A_i|D)$ is then used to generate the offspring $\mathcal{A}(g+1)$.

In effect, the evolutionary inference step from generation $g-1$ to g is considered to induce a new fitness distribution $P_g(A_i|D)$ from $P(A_i)$ following Bayes rule.

Based on this theoretical foundation we present in the following sections two specific examples of Bayesian genetic programming that employ techniques for effective control of evolutionary dynamics.

3 GP with an Adaptive Occam's Razor

The first Bayesian GP focuses on the fact that genetic programming iteratively searches populations of more probable (more likely in terms of the data and the priors) programs and thus the information gained in the previous generation can be used in the next generation to revise the prior (before seeing the data) belief in true models. This leads to an adaptive fitness function

$$P_g(A_i|D) = \frac{P(D|A_i)P_{g-1}(A_i)}{\sum_{A_j \in \mathcal{A}(g)} P(D|A_j)P_{g-1}(A_j)}, \quad (6)$$

where the priors $P_{g-1}(A_j)$ are defined now as a function of generation instead of the fixed prior $P(A_i)$ as in equation 5. It is revised by a belief update function $h(\cdot)$:

$$P_g(A_i) = h(P_{g-1}(A_i), P_g(A_i|D)). \quad (7)$$

By appropriately choosing the initial priors $P_0(A)$ on the models and by making use of the relation between the probability $P(A)$ and its complexity (i.e., optimal code length $\log \frac{1}{P(A)}$ for $P(A)$) from coding theory, we can derive a Bayesian technique based on this adaptive fitness function for evolving parsimonious and accurate models. It can be shown that the adaptive Occam method described in [11] is derived from the Bayesian genetic programming framework.

4 GP with Incremental Data Inheritance

In the second example of the Bayesian approach to GP, we make use of the fact that the Bayes formula suggests an incremental, evolutionary learning rule: infer models of higher posterior probability from the existing models by observing new data. This leads to redefine the fitness as

$$P_g(A_i^g|D_i^g) = \frac{P(D_i^g|A_i^g)P(A_i^g)}{\sum_{A_j^g \in \mathcal{A}(g), D_j^g \in \mathcal{D}(g)} P(D_i^g|A_j^g)P(A_j^g)}, \quad (8)$$

where the data set D_i^g is now a variable of generation number g and specific to a single program A_i^g. The collection of D_i^g constitutes the data population $\mathcal{D}(g)$, where each D_i^g is associated with an individual A_i^g in the program population $\mathcal{A}(g)$.

The Bayesian evolutionary algorithm with the incremental data inheritance (IDI) method [9] is an example of this approach, where D_i^g specifically satisfies the following conditions:

$$D_i^g \subset D, \quad |D_i^{g-1}| < |D_i^g|, \quad (9)$$

where D is the given data set. In this method, the fitness of programs is estimated on incrementally chosen data subsets D_i^g, rather than on the whole data set D, and thus the convergence is accelerated by reducing the effective number of fitness evaluations [7]. In some situations, novel examples can be generated (rather than just selected from existing ones) by a genetic algorithm as suggested in [12].

5 Discussion

The Bayesian approach to genetic programming provides a number of important features. One is that by viewing the GP process as Bayesian inference, principled techniques for driving evolutionary dynamics of conventional GPs can be developed. Two examples of them are the Bayesian GP with the adaptive Occam method [11] and the Bayesian GP with incremental data inheritance.

Another interesting feature of Bayesian inference is that it allows background knowledge in the problem domain to be incorporated in a formal way. For example, if we can guess the size of good solutions, this knowledge can be reflected in the prior probability distribution $P_0(A_i)$ in equation (6). This property is important especially for solving real-life problems of practical interest using genetic programming [9].

Decision-making can be made more robust by combining multiple models instead of a single model. Genetic programming generate by definition a number of models. The Bayesian genetic programming framework provides a principled way to combine multiple programs to build a committee machine [10] for more robust decision-making.

Finally, the Bayesian framework for genetic programming appears to be general enough to incorporate existing genetic programming procedures as special (finite-sample) approximations to the full Bayesian approach. The generality of the framework is crucial for exploration and comparative analysis of various genetic programming.

Acknowledgments

This research was supported in part by the Korea Research Foundation (KRF) under Grant 1998-001-E01025.

References

[1] J. M. Daida, F. Bersano-Begey, S. J. Ross, and J. F. Vesecky (1996). Computer-assisted design of image classification algorithms: Dynamic and static fitness evaluations in a scaffolded genetic programming environment. In J.R. Koza (eds.), *The First Genetic Programming Conference* (GP-96), Morgan Kaufmann, pp. 279-284.

[2] C. Gathercole, and P. Ross (1997). Small populations over many generations can beat large populations over few generations in genetic programming. In J.R. Koza (eds.). *The Second Genetic Programming Conference* (GP-97), Morgan Kaufmann, pp. 111-118.

[3] W. B. Langdon and R. Poli (1998). Fitness causes bloat: Mutation, In W. Banzhaf, R. Poli, M. Schoenauer and T. Fogarty (Eds.), *The First European Workshop on Genetic Programming* (EuroGP'98), Paris, LNCS 1391, Springer-Verlag, Berlin, pp. 37-48.

[4] J. P. Rosca (1997). Analysis of complexity drift in genetic programming, *The Second Genetic Programming Conference* (GP-97), In J. R. Koza (eds.), Morgan Kaufmann, pp. 286-294.

[5] T. Soule and J. A. Foster (1998). Effects of code growth and parsimony pressure on populations in genetic programming, *Evolutionary Computation*, 6(4):293-309.

[6] A. Teller, and D. Andre (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. In J. R. Koza (eds.) *The Second Genetic Programming Conference* (GP-97), Morgan Kaufmann, pp. 321-328.

[7] B.-T. Zhang (1994). Accelerated learning by active example selection, *International Journal of Neural Systems*, 5(1):67-75.

[8] B.-T. Zhang (1999). A Bayesian framework for evolutionary computation, *The 1999 Congress on Evolutionary Computation* (CEC99), Special Session on Theory and Foundations of Evolutionary Computation, Washington, D.C., IEEE Press, (to appear).

[9] B.-T. Zhang and J.-G. Joung (1999). Genetic programming with incremental data inheritance, *The 1999 Genetic and Evolutionary Computation Conference* (GECCO-99), Orlando, Florida, Morgan Kaufmann, (to appear).

[10] B.-T.Zhang and J.-G. Joung (1997). Enhancing robustness of genetic programming at the species level, *The Second Genetic Programming Conference* (GP-97), Morgan Kaufmann, pp. 336-342.

[11] B.-T. Zhang and H. Mühlenbein (1995). Balancing accuracy and parsimony in genetic programming, *Evolutionary Computation*, 3(1):17-38.

[12] B.-T. Zhang and G. Veenker (1991). Neural networks that teach themselves through genetic discovery of novel examples, *Proc. Int. Joint Conf. on Neural Networks* (IJCNN-91), IEEE Press, pp. 690-695.