Learning Computer Programs with the Bayesian Optimization Algorithm

Moshe Looks
Object Sciences Corporation
6359 Walker Lane Suite 100
Alexandria, VA 22310
1-703-253-1181
mlooks@objectsciences.com

Ben Goertzel
Novamente LLC
1405 Bernerd Place
Rockville, MD 20851
ben@novamente.net

Cassio Pennachin
Novamente LLC
1405 Bernerd Place
Rockville, MD 20851
cassio@novamente.net

ABSTRACT
We describe an extension of the Bayesian Optimization Algorithm (BOA), a probabilistic model building genetic algorithm, to the domain of program tree evolution. The new system, BOA programming (BOAP), improves significantly on previous probabilistic model building genetic programming (PMBGP) systems in terms of the articulacy and open-ended flexibility of the models learned, and hence control over the distribution of instances generated. Innovations include a novel tree representation and a generalized program evaluation scheme.

Categories and Subject Descriptors
I.2.2 [**AI**]: Automatic Programming – *program synthesis*

General Terms
Desgin, Algorithms

Keywords
Genetic Programming, Representations, Empirical Study

1. OVERVIEW AND MOTIVATION
Recently, attempts have been made to extend traditional genetic algorithms and genetic programming through the incorporation of explicit modeling of good solutions. This is the idea behind probabilistic model building genetic algorithms, such as the (hierarchical) Bayesian Optimization Algorithm (BOA) [6]. BOA is asymptotically more effective than the GA across a wide range of problems. We hope that the same scalability and performance improvements can be obtained in the domain of automated program tree evolution, or genetic programming [3]. We propose an extension of BOA for the evolution of program trees, called BOA programming (BOAP). BOAP is a probabilistic model building genetic programming system (PMBGP). We compare with previous PMBGPs and provide experimental results on time series prediction and bioinformatics applications.

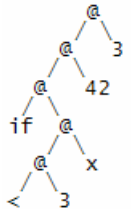
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright is held by the authors.
GECCO'05, June 25-29, 2005, Washington, DC, USA.
ACM 1-59593-010-8/05/0006.

2. REVIEW OF BOA
The Bayesian Optimization Algorithm (BOA) is a population-based optimization algorithm that works on bit strings. BOA significantly outperforms the genetic algorithm on a range of optimization problems by maintaining a centralized probabilistic model of the population it is evolving [6]. Different variations on BOA can be obtained by using different types of probabilistic models, most notably in hierarchical BOA [6], which is utilized in this work, but the basic algorithm remains the same:

(1) Generate a random initial population P(0)
(2) From the promising instances in P(t) learn a model M(t)
(3) Generate a new set of instances O(t) from M(t)
(4) Merge O(t) and P(t), creating P(t+1)
(5) Iterate steps (2) through (4) until termination criteria are met.

Following this algorithm, good collections of variable assignments will be preserved in instance generation (if the modeling is accurate). Thus, BOA can explore new areas of the search space in a more directed and focused way than GA/GP, while retaining the benefits of a population-based optimization algorithm (diversity of candidate solutions and non-local search).

3. BOA PROGRAMMING
Previous work by Ocenasek [5] has extended binary BOA to fixed-length strings with non-binary discrete and continuous variables. We extend his approach to variable-shaped trees. We represent program trees in curried form [1], where all function applications are single argument, and all internal nodes (denoted by '@') represent function applications. Programs expressed in this form are stored using a new representational scheme, zigzag trees, which are a special kind of binary tree. A zigzag tree of size N can be encoded as a list of N leaves, and a list of N-1 Boolean values (which indicate at each point in which direction to keep growing the tree). An important property of zigzag trees is that if a couple of simple rewrite rules are allowed to live in leaves, they become fully general, and can represent any possible binary tree. See a sample zigzag tree below.



This tree is encoded as:

```
Leaves:
   < 3 x if 42 3
Structure:
   Right Right Left Right Right
```

This representation can be modeled by BOA, but the models may produce syntactically incorrect structures, as the type of a parent doesn't necessarily constrain the types of its children to be compatible. We address this problem with the use of a more general evaluation scheme that can evaluate such incorrect expressions through local transformations.

While this BOA extension is effective at optimizing small subcomponents, and keeping components correctly intact, it does not, on its own, lead to the addition of new components, because it lacks a global genetic operator. To overcome this, we have added a low-frequency traditional GP crossover operator, which can cause trees to grow, and suffices to move subcomponents to new locations, where they can be optimized by BOA.

4. COMPARISON WITH PRIOR WORK

An early PMBGP was probabilistic incremental program evolution (PIPE) [7]. PIPE uses standard GP function trees as its representation. New trees are generated by drawing, in a depth-first fashion, terminals and leaves from the distribution at each (absolute) tree position, with an additional mutation operator. Some more recent work, such as program evolution with explicit learning (PEEL) [8], uses more complex models, in this case program generation grammars learned via ant colony optimization. The primary limitations of PIPE and hPIPE are the fixed complexity of their models (no dependencies learned between positions), and the complete reliance on absolute position. PEEL, on the other hand, is capably of learning variable-complexity models, but the dependencies they capture are based on location of elements in the tree, not on tree content.

5. EXPERIMENTS

In all of the following experiments, BOAP is configured to generate a new population as described in [6], and the crossover operator is then applied to a random selection of 20% of the instances, excluding the single best instance in the population.

5.1 Sunspot Time Series Prediction

A well-studied time series prediction problem is the yearly sunspot data for 1700-1979. Published benchmarks are available for PEEL [8] and GP [2]. In order to allow for meaningful comparison to these results, we used the error measure

$$\frac{1}{\sigma^2} \cdot \frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - \hat{x}_i)^2$$

where n is the number of data points, the x_i are the values, and σ^2=0.41056 is the variance of the entire dataset. 600,000 fitness evaluations are used per run, and 15 independent runs were carried out for BOAP and PEEL, and 10 for GP.

Table 1. Sunspot Time Series Prediction

	Training (1700-1920)	Generalization (1921-1955)	Generalization (1956-1979)
BOAP	0.124±0.007	0.169±0.018	0.289±0.06
PEEL	0.137±0.016	0.185±0.039	0.291±0.071
GP	0.125±0.006	0.182±0.037	0.37±0.06

5.2 Gene Function Inference

In this set of experiments, we turn to a new and challenging supervised categorization problem: the automated placement of human genes in functional categories based on gene expression data. In order to cope with the vast amounts of new bioinformatics data made available by new biological technologies, researchers have developed a variety of tools, including controlled vocabularies for discussing biological phenomena, and collaborative knowledge bases constructed using these vocabularies. One such effort is the multi-organism Gene Ontology (GO) [4]. Genes are annotated into the Gene Ontology by scientists throughout the world, but currently the GO is still very sparse. We utilized genomic expression data to create models for five categories of the GO. These models can then be used to assign function to previously unnanotated genes, offering biologists new and important insights. This is an exceedingly hard problem given the scarce and noisy nature of the data, and GP and SVMs fail to generate any useful models even in-sample. BOAP, on the other hand, learns good models in-sample for all categories, and generalizes very well in one of them, Ribosomes.

Table 2: Gene Function Categorization Precision

	Train	Test
Catalytic Activity	0.729±0.009	0.464±0.025
Intracellular	0.766±0.01	0.526±0.027
Ribosome	0.907±0.005	0.814±0.022
Cell Growth/Maintenance	0.751±0.016	0.5084±0.043
Metabolism	0.773±0.007	0.549±0.023

6. REFERENCES

[1] Field, A. J., and Harrison, P. G. 1988. *Functional Progamming*. Boston, MA: Addison-Wesley.

[2] Jäske, Harri. 1996. Prediction of Sunspots by GP. In *Proc. of the Second Nordic Workshop on Genetic Algorithms and their Applications (2NWGA)*, pages 79-88.

[3] Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press.

[4] Gene Ontology Consortium. 2000. Gene Ontology: Tool for Unification of Biology. *Nature Genet*, 25:25-29.

[5] Ocenasek, J. 2002. *Parallel Estimation of Distribution Algorithms*. PhD. thesis, Brno University of Technology.

[6] Pelikan, M. 2002. *Bayesian Optimization Algorithm: From Single Level to Hierarchy*. Ph.D. thesis, University of Illinois at Urbana-Champaign.

[7] Salustowicz, R. P., and Schmidhuber, J. 1997. Probabilistic Incremental Program Evolution. *Evolutionary Computation*, 5(2):123-141.

[8] Shan, Y., McKay, R. I., Abbass, H. A., and Essam, D. 2003. Program Evolution with Explicit Learning: a New Framework for Program Automatic Synthesis. Technical Report CS04/03. School of Computer Science, Univ. College, Univ. of New South Wales.