Genetic Programming

William H. Hsu, Kansas State University, USA

INTRODUCTION

Genetic programming (**GP**) is a subfield of evolutionary computation first explored in depth by John Koza in *Genetic Programming: On the Programming of Computers by Means of Natural Selection* and independently developed by Nichael Lynn Cramer. and It is a method used to allow computer programs to evolve according to some user-defined goal. It uses evolutionary patterns including crossover, selection, replication and mutations to evolve the programs, which are usually represented by Lisp expressions. To work effectively, it requires an appropriate selection of operators and variables.

Genetic programming uses methods which are similar to genetic algorithms (GA), but is based on programs which perform tasks whose results can then be evaluated to deliver a fitness function similar to GAs. Instead of using pools of parameter lists to be evaluated by some evaluation procedure, GP uses pools of programs which are to be run to perform the required task. A technical difference between GAs and GPs is that GAs use list structures, often of fixed size, to store their data, while GPs use tree structures which can vary in size and shape for each program used in the program pools.

BACKGROUND

The application of a tree representation (and required genetic operators) for using genetic algorithms to generate programs was first described in 1985 by Cramer. Koza, though he did not invent genetic programming, is indisputably the field's most prolific and persuasive author.

So far GPs have successfully solved some toy problems, such as the lawn mower problem, but the method is very computationally intensive, and may not compare favourably where simpler methods, such as genetic algorithms or random optimisation can be used instead. It is possible that some more complex problems may be more amenable to solution using GPs than other optimization methods.

Unfortunately, due to the lack of solid theory regarding the performance of genetic programming vs. traditional search methods (such as hill climbing), genetic programming remains a sort of pariah amongst the various techniques of search. While genetic programming has achieved results that are as good as and sometimes better than human-generated results, more work needs to be done on the theory in order to bring the technique into more widespread use.

MAIN THRUST OF THE CHAPTER

Evolutionary Computation in Java (ECJ) and Simulator

All GP experiments were all conducted using Luke's Evolutionary Computation in Java (ECJ) package [Lu02]. A set of operators was developed in Java for the 3-on-1 keep-away task and is described in the Experiment Design section.

Where specified, ECJ defaults [Lu02] were overridden. These overrides, in turn, follow Gustafson's original implementation using an earlier version of ECJ [Gu00]. All variations (simple GP, ADF-GP, GP-ISLES, and incremental ADF-GP) use ramped half-and-half initialization, tournament selection with tournament size 7. The genetic crossover operator generates 90 percent of the next generation; tournament selection generates the other 10 percent. [Ko92] The GP variations use no mutation, permutation, over-selection, or elitism.

Fitness evaluations are made using Gustafson's 20-by-20 grid-based abstract simulator for keep-away soccer [Gu00]. Previous work by Stone and Sutton [SS01] and by Hsu and Gustafson [HG02] on the 3-on-1 task defined minimization of turnovers (change in possession) as the objective function for the full keeper policy. Let us define this problem specification as 3-on-1-turnovers and easier subtasks, based upon the number of passes completed, as k-on-t-passing (for $k \leq 3$, $t \leq 1$).

For standardization, all runs use a generational GP with population size 4000 and 101 generations. Experiments with population size 1000, 2000, 4000, and 8000 and with fewer (51) and more (201) generations showed the above parameters to be effective for this test bed, as Gustafson also reports [Gu00].

Monolithic Simple GP and ADF-GP

As a baseline for comparison, we used SGP and ADF-GP with the single monolithic, or non-incremental, objective of minimizing the number of turnovers that occur in a simulation.

The ADF-GP is initialized with maximum size 6 for initial random programs. Hybrid variations also use this constraint, but have no restrictions on ADF seeding (as documented in the next sections). ADF-GP allows each tree for kicking and moving to have two additional trees that represent ADFs, where the first ADF can call the second, and both have access to the full function set available for SGP.

The next three sections describe incremental reuse: first using easy missions (GP-ISLES), then using single-mission incremental ADFs.

FUTURE TRENDS

Basic GP-ISLES and Related Work

A basic version of GP-ISLES is described by Gustafson and called layered learning GP (LLGP) [Gu00]. To modify standard GP for incrementally staged learning, we must develop a learning objective for each layer, i.e., the fitness at each layer that selects ideal individuals for the easier subtask. The GP-ISLES system focuses on automatically discovering how to compose passing agents into keep-away soccer agents. GP-ISLES has two layers; the fitness objective for the first layer is to maximize the number of accurate passes (a two-agent task evaluated over teams of three copies of the same individual, on the same size field as the keep-away soccer task), while fitness objective for the second layer is to minimize the number of turnovers.

In comparing this incremental approach to the monolithic systems (using a simple GP and a GP with ADFs), Gustafson found that it outperformed both GP and ADF-GP on average, achieving a best-of-run fitness of 5.8 turnovers in a 200-time step simulation [Gu00]

REFERENCES

[Gu00] S. M. Gustafson. *Layered Learning in Genetic Programming for A Cooperative Robot Soccer Problem.* M.S. thesis, Department of Computing and Information Sciences, Kansas State University, 2000.

[HG02] W. H. Hsu and S. M. Gustafson. Genetic Programming and Multi-Agent Layered Learning by Reinforcements. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, New York, NY, 2002.

[Lu00] S. Luke. *Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat*. Ph.D. Dissertation, Department of Computer Science, University of Maryland, College Park, MD, 2000.

[Lu04] S. Luke. *Evolutionary Computation in Java v*9. Available from URL: http://www.cs.umd.edu/projects/plus/ec/ecj/.

[SF98] T. Soule and J. A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC-1998)*, p. 781-786. IEEE Press, 1998.

[SV00b] P. Stone and M. Veloso. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots*, 8(3): 345-383. Kluwer, 2000.

Brameier, M. & Banzhaf, W. (2001). Evolving Teams of Predictors with Linear Genetic Programming. *Genetic Programming and Evolvable Machines* **2**(4), p. 381-407.

Burke, E. K., Gustafson, S. & Kendall, G. (2004). Diversity in genetic programming: an analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation* **8**(1), p. 47-62.

Cantu-Paz, E. (1999). *Designing Efficient and Accurate Parallel Genetic Algorithms*. Ph.D. thesis, University of Illinois at Urbana-Champaign. Technical report, Illinois Genetic Algorithms Laboratory (IlliGAL).

Cramer, Nichael Lynn (1985), "A representation for the Adaptive Generation of Simple Sequential Programs" in *Proceedings of an International Conference on Genetic Algorithms and the Applications*, Grefenstette, John J. (ed.), CMU

Duda, R. O., Hart, P. E., & Stork, D. (2000). *Pattern Classification, Second Edition*. New York, NY: John Wiley and Sons.

Hsu, W. H., Ray, S. R., & Wilkins, D. C. (2000). A Multistrategy Approach to Classifier Learning from Time Series. *Machine Learning*, *38*, 213-236.

Hsu, W. H., Welge, M., Redman, T., & Clutter, D. (2002). Constructive Induction Wrappers in High-Performance Commercial Data Mining and Decision Support Systems. *Data Mining and Knowledge Discovery*.

Keijzer, M. & Babovic, V. (2002). Declarative and Preferential Bias in GP-based Scientific Discovery. *Genetic Programming and Evolvable Machines* **3**(1), p. 41-79.

Kishore, J. K., Patnaik, L. M., Mani, V. & Agrawal, V.K. (2000). Application of genetic programming for multicategory pattern classification. *IEEE Transactions on Evolutionary Computation* **4**(3), p. 242-258.

Koza, J.R. (1992), *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press

Koza, J.R. (1994), *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press

Koza, J.R., (2003). *Genetic Programming IV*, Morgan Kaufmann.

Koza, J.R., Bennett, F.H., Andre, D., and Keane, M.A. (1999), *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann.

Krawiec, K. (2002). Genetic Programming-based Construction of Features for Machine Learning and Knowledge Discovery Tasks. *Genetic Programming and Evolvable Machines* **3**(4), p. 329-343.

Muni, D. P., Pal, N. R. & Das, J. (2004). A novel approach to design classifiers using genetic programming. *IEEE Transactions on Evolutionary Computation* **8**(2), p. 183-196.

Nikolaev, N. Y. & Iba, H. (2001). Regularization approach to inductive genetic programming. *IEEE Transactions on Evolutionary Computation* **5**(4), p. 359-375.

Nikolaev, N. Y. & Iba, H. (2001). Accelerated Genetic Programming of Polynomials. *Genetic Programming and Evolvable Machines* **2**(3), p. 231-257.

Principé, J. & Lefebvre, C. (2001). *NeuroSolutions v4.0,* Gainesville, FL: NeuroDimension. URL: http://www.nd.com.

Wong, M. L. & Leung, K. S. (2000). Data Mining Using Grammar Based Genetic Programming and Applications (Genetic Programming Series, Volume 3). Norwell, MA: Kluwer.

TERMS AND THEIR DEFINITION

Clickstream: The sequence of mouse clicks executed by an individual during an online Internet session.

Data Mining: The application of analytical methods and tools to data for the purpose of identifying patterns and relationships such as classification, prediction, estimation, or affinity grouping.

Discovery Informatics: The study and practice of employing the full spectrum of computing and analytical science and technology to the singular pursuit of discovering new information by identifying and validating patterns in data.

Evolutionary Computation: Solution approach guided by biological evolution, which begins with potential solution models, then iteratively applies algorithms to find the fittest models from the set to serve as inputs to the next iteration, ultimately leading to a model that best represents the data.

Knowledge Management: The practice of transforming the intellectual assets of an organization into business value.

Neural Networks: Learning systems, designed by analogy with a simplified model of the neural connections in the brain, which can be trained to find nonlinear relationships in data.

Rule Induction: Process of learning, from cases or instances, if-then rule relationships consisting of an antecedent (if-part, defining the preconditions or coverage of the rule) and a consequent (then-part, stating a classification, prediction, or other expression of a property that holds for cases defined in the antecedent).