Coevolutionary Dynamics of a Multi–Population Genetic Programming System

Josh C. Bongard

School of Cognitive and Computing Sciences
University of Sussex, Brighton UK
joshuab@cogs.susx.ac.uk

Abstract. This report presents an asynchronous, distributed genetic programming (GP) system using a master/slave architecture. Using symbolic regression for fourier functions as the problem domain, the system was found to demonstrate cooperative coevolutionary dynamics when multiple client populations evolve solutions to similar, but different problems: specifically, closely coupled populations were found to promote continuous search, which in some cases leads to the discovery of better solutions.

1 Introduction

The abilities of various evolutionary algorithms to traverse large search spaces have been well documented[5][7], as well as their amenability to parallel implementation [3]. Much work has investigated various parallel architectures that partition, or otherwise reorganize a single problem[4][8][11]: focus is placed on the dispersion and synthesis of partial solutions, often referred to as building blocks or schemas.

However, there is little investigation of parallel algorithms that attempt to distill useful information from a set of similar, but different problems. It seems plausible that populations evolving solutions to similar problems may benefit from sharing their results.

It has been found that sharing is indeed possible, and is correlated to the degree of similarity between fitness landscapes. Most importantly, however, we have found that two (or more) populations that share solutions can help each other escape local optima and explore increasingly better optima. As opposed to Hillis'[6] appropriation of parasitic coevolutionary dynamics to guard against premature convergence, the formulation presented here uses cooperative coevolutionary dynamics—in the form of solution transmission between populations—to reach continually better solutions.

2 The Model

A distributed GP model using a master/slave architecture similar to that described in [9] is used to gather results in this report: clients send their current,

most fit solution to a central server, and download solutions generated by other clients from that server. Unlike the architecture described in [9], however, all solutions received by the server are broadcast to a requesting client, with the exception of solutions sent by the same client.[1]

We have chosen the problem domain of symbolic regression with which to gather quantitative measurements of the coevolutionary dynamics of the described model. Problem sets are defined as parametric fourier functions of the form

$$y_i = \sum_{j=1}^{5} a_j sin(jx_i) + b_j cos(ix_i)$$

, where each fourier function is characterized by selecting values for each a_j and b_j. Each problem set is composed of 40 values for the independent variables $\mathbf{x}$ uniformly distributed between $[-5, 5]$. The raw fitness of an s-expression s in the GP population is set to $f_s = \sum_{i=1}^{40} \|y_i - o_{i,p}\|$ where $o_{i,p}$ is the output of s when presented with the independent variables $\mathbf{x}$.

It is then possible to quantitatively measure the distance between two training sets t_1 and t_2 using

$$d_{t_1,t_2} = \sum_{i=1}^{40} \|y_{i,t_1} - y_{i,t_2}\| \tag{1}$$

if we assume that the set of dependent variables $\mathbf{x}$ is invariant across all possible training data.

All of the results below were generated using a GP instantiation with a function set covering arithmetic and trigonometric functions; a terminal set of the dependent variables and two floating point constants; one mutation for every 100 nodes involved in subtree crossover; a population size of 1000; and tournament selection with a tournament size of 10.

Each client population is assigned one fourier function as described above, and is run for 500 generations, which was chosen to ensure convergence in most populations. The benefits of convergence prior to migration are reported in [1] and [10]. At the end of the run the most fit solution from each population is sent to the central server, and the populations are then reinitialized with random solutions, as well as downloading solutions from the server. Transferred solutions then participate in selection and crossover depending on their fitness in the new population. Each population operates asynchronously; populations terminate and begin new runs independent of the other populations.

3 Results

For the case of a set of client populations assigned the same problem set, the model presented in this report operates as a distributed panmictic population:

[1] This minimizes the possibility of a client population reconverging to a solution it had previously uploaded to the server.

the client populations are equivalent to demes, in which solutions migrate at fixed intervals, and the demes are fully connected.[2]

Consider now a set of client populations evolving solutions to similar, but different problems. This situation was modelled by generating 40 fourier functions using random values of a_j and b_j as discussed in section 2. The distances between each pair of the 40 problem sets was determined using equation 1, and the two problem sets with the minimum distance were assigned to two client populations.

The two client populations were run for 500 generations each, sending their most fit solution at the end of each run to the server. The populations were then re-initialized with random solutions, as well as retrieving solutions from the server. This process was repeated over 20 runs. It was found that the two populations experienced a decrease in the normalized fitness of the most fit solution in each of the two populations at the end of each run, as the number of runs increases. The results obtained are shown in Figure 1. The results are contrasted against a control case: two populations were evolved for 500 generations over 20 runs with no sharing.

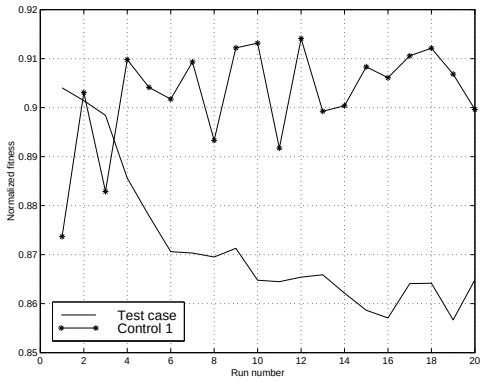

Fig. 1. A plot of the performance increase exhibited by client populations evolving solutions to similar training data. The test case involved two populations evolving solutions over 20 runs of 500 generations each, and trading their most fit solutions after each run via a central server. The control case involved two populations evolving over 20 runs of 500 generations each, but did not involve trading of solutions. Results from the test case and control case were averaged over 10 iterations.

4 Discussion

In modelling two client populations evolving solutions to similar, but different problem sets, it was found that populations do not converge on a single optimum, but rather continue to explore regions of the search space near some optimum

common to both populations. This opposes the findings for traditional, multi-deme GA populations reported in [3], in which the parameter dictating the migration rate of solutions between demes acts as a threshold, below which demes converge on different, but inferior optima, and above which demes (sometimes prematurely) converge on the best solution discovered across all demes.

The continuous search property of the model presented in this report is demonstrated in Figure 2, in which two client populations are contrasted. In the control population, 20 runs of 500 generations are performed, and averaged over 20 iterations. At the end of each run, the most fit solution in the population is preserved, but undergoes a large mutation, in which a sub-tree of depth two is replaced with a random sub-tree of depth two. The remaining solutions in the population are replaced with random solutions. No solutions from this population are sent to or received from the central server. The other population trades solutions with other evolving populations as described in section 2. Both populations evolve solutions to the same fourier function.

The large mutation experienced in the first population is meant to simulate the movement of a population a small distance away from the fitness optimum it had recently achieved. During subsequent evolution, the population may return to its previous optimum, or discover another, either inferior or superior optimum.

Over the 20 runs it was found that both populations continue to find better solutions, but that the population that trades solutions with other populations reaches better optima than those discovered by the control population. This indicates that the population that shares solutions discovers new, better optima more often than the population that undergoes repeated large mutations.

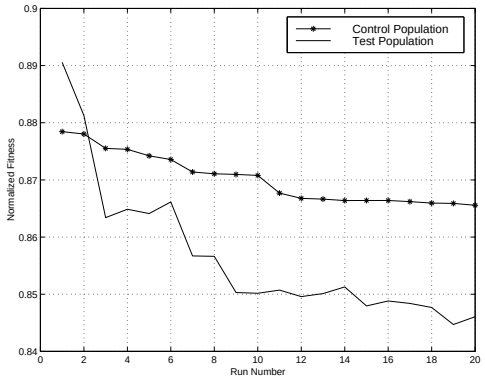

Fig. 2. Two populations evolving solutions to the same problem set for 20 runs of 500 generations each. In the control population, at the end of each run, the best solution undergoes a large mutation, while the remaining solutions are replaced with random solutions. In the test population, at the end of each run, the best solution is sent to the server, all solutions are replaced with random solutions, and solutions are then downloaded from the central server. The results from both populations were averaged over 20 iterations.

5 Conclusions

In this paper we have explored the cooperative, coevolutionary dynamics that appear in an asynchronous, distributed genetic programming architecture. As opposed to the usual formulation of a multideme, distributed GA with a specified transmission rate of solutions between populations, our architecture services client populations with different problem sets drawn from a common problem domain. Transmission of solutions is arbitrated by a central server.

It was found that mutual exchange of solutions between client populations with similar, but different problem sets exhibits a coevolutionary dynamic: solutions from one population, when inoculated in another population, can pull the second population away from its current local optimum, and often lead to the discovery of better optima.

References

1. Braun, H. C.: On Solving Travelling Salesman Problems by Genetic Algorithms. In: Schwefel H.-P. & R. Männer (eds.): Parallel Problem Solving From Nature, pp. 129–133. Springer-Verlag, Berlin (1990).
2. Cantú-Paz, E.: Topologies, Migration Rates, and Multi-Population Parallel Genetic Algorithms. To Appear in: GECCO-99, Genetic and Evolutionary Computation Conference, July 13–17, Orlando FL (1999).
3. Cantú-Paz, E.: A Survey of Parallel Genetic Algorithms. In: Calculateurs Paralleles. **10**:2 (1998).
4. Cantú-Paz, E. & D. Goldberg: Modelling Idealized Bounding Cases of Parallel Genetic Algorithms. In: Koza, J., K. Deb, M. Dorigo, D. Fogel, M. Garzon, H. Iba & R. Riolo (eds.): Genetic Programming 1997: Proceedings of the Second Annual Conference, pp. 353–361. Morgan Kauffman, San Francisco CA (1997).
5. Goldberg, D. E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Redwood City CA (1989).
6. Hillis, D. W.: Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure. In: Langton, C. G., Taylor, C. (eds.): Artificial Life II, pp. 313–322. Addison-Wesley, Redwood City CA (1992).
7. Koza, J. R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge MA (1992).
8. Mahfoud, S. W.: A Comparison of Parallel and Sequential Niching Methods. In: ICGA **6**, pp. 136–143. (1995).
9. Marin, F. J., O. Trelles-Salazar & F. Sandoval: Genetic Algorithms on LAN-message passing architectures using PVM: Application to the Routing Problem. In: Davidor, Y., H.-P. Schwefel & R. Männer (eds.): Parallel Problem Solving from Nature III, pp. 534–543. Springer-Verlag, Berlin (1994).
10. Munetomo, M., Y. Takai & Y. Sato: An Efficient Migration Scheme for Subpopulation-Based Asynchronously parallel genetic algorithms. In: Forrest, S. (ed.): Proceedings of the Fifth International Conference on Genetic Algorithms, p. 649. Morgan Kauffman, San Mateo CA (1993).
11. Miller, B. L. & M. J. Shaw: Genetic Algorithms with Dynamic Niche Sharing for Multimodal Function Optimization. In: IEEE International Conference on Evolutionary Computation, pp. 786–791. IEEE Press, Piscataway NJ (1996).