EVOLVING NEURAL NETWORKS WITH COLLABORATIVE SPECIES*

Mitchell A. Potter and Kenneth A. De Jong
Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375
email: mpotter@aic.nrl.navy.mil

Keywords: Genetic algorithms, coevolution, neural networks, parallel processing

Abstract

We present a coevolutionary architecture for solving decomposable problems and apply it to the evolution of artificial neural networks. Although this work is preliminary in nature it has a number of advantages over non-coevolutionary approaches. The coevolutionary approach utilizes a divide-and-conquer technique in which species representing simpler subtasks are evolved in separate instances of a genetic algorithm executing in parallel. Collaborations among the species are formed representing complete solutions. Species are created dynamically as needed. Results are presented in which the coevolutionary architecture produces higher quality solutions in fewer evolutionary trials when compared with an alternative non-coevolutionary approach on the problem of evolving cascade networks for parity computation.

1 INTRODUCTION

A natural approach for solving many complex problems is to utilize a divide-and-conquer technique in which the problem is decomposed into simpler subproblems. Instances of these problems are numerous. For example, route planning tasks such as the traveling salesman problem, which can be decomposed into simpler subtours; behavior learning tasks such as those one would encounter in the domain of robotics, where complex behavior can be decomposed into simpler subbehaviors; and tasks involving complex structures such as artificial neural networks, which may be decomposed into simpler substructures.

If the task is decomposable into independent subtasks, a conventional evolutionary algorithm could be used to evolve solutions to the subtasks either sequentially or in parallel. However, this approach does not address the difficult issues of how the task is to be decomposed, evaluated, and later recombined into a complete solution. In addition, when interdependencies exist between subtasks it is desirable to extend the standard evolutionary algorithm to enable the performance-oriented evaluation of individual subsolutions to be performed within the context of a complete solution.

In this paper we present a cooperative coevolutionary approach for solving decomposable problems which, although preliminary in nature, is an initial step towards addressing these issues. In our approach multiple instances of an evolutionary algorithm (EA) are run in parallel. Each EA instance evolves a species of individuals representing competing solutions to a subtask. Rather than evaluating solutions to these subtasks independently, the EA instances communicate with each other for the purpose of forming collaborations. This is accomplished by selecting representatives from each of the EA populations (the species), and combining them into a single composite structure capable of being evaluating on the top level goal task. Credit from evaluating the composite structures flows back to the individual subcomponents reflecting how well they collaborate with the other subcomponents to achieve the top level goal. This credit is then used by the local EAs to evolve better subcomponents. We call such systems *Cooperative Coevolutionary Algorithms* (CCAs).

We initially tested this CCA architecture in two quite different domains: the well-studied and well-understood domain of function optimization and the domain of learning rule sets for controlling a simulated robot. In both cases we achieved considerable performance improvements over more traditional centralized architectures (Potter and De Jong 1994; Pot-

*To appear in Proceedings of the 1995 Summer Computer Simulation Conference, July 24-26, 1995, Ottawa, Ontario, Canada

ter, De Jong, and Grefenstette 1995). The particular CCA instantiation used in these earlier studies utilized a fixed number of species. In both domains, the required number of species could be determined *a priori*.

In this paper we extend the CCA architecture to support the dynamic creation of species and apply it to the problem of evolving *cascade networks*. Cascade networks are a type of multilayer feed-forward neural network in which hidden nodes are created incrementally and their output signals cascaded. These networks will be described in greater detail in section three. Results are presented showing that the coevolutionary architecture produces higher quality solutions in fewer evolutionary trials when compared with a traditional single-population EA on the problem of evolving cascade networks for parity computation.

In the next section we describe the CCA architecture in more detail. Section three describes specifically how the algorithm is applied to the problem of evolving cascade networks and presents an alternative non-coevolutionary approach. This is followed by a section presenting empirical results comparing the performance of the two approaches. The paper concludes with a brief discussion of the results and ideas for future studies.

2 BASIC FRAMEWORK FOR COEVOLUTION

The hypothesis underlying the ideas presented here is that, in order to evolve solutions to more and more complex problems, explicit notions of modularity need to be introduced in order to provide reasonable opportunities for complex solutions to evolve in the form of interacting co-adapted subcomponents. The difficulty comes in finding reasonable computational extensions to our current evolutionary paradigms in which such subcomponents "emerge" rather than being hand designed as in (de Garis 1990). At issue here is how to represent such subcomponents and how to apportion credit to them for their contributions to the problem solving activity such that the evolution of a solution to the top level goal proceeds without a human in the loop.

Classifier systems attempt to accomplish this via a single population of interacting rules whose individual fitnesses are determined by their interactions with other rules via a simulated micro-economy (Holland and Reitman 1978). Other extensions to current evolutionary paradigms have been proposed to encourage the emergence of niches and species in a single population (De Jong 1975; Deb and Goldberg 1989; Davidor 1991; Forrest, Javornik, Smith, and Perelson 1993; Giordana, Saitta, and Zini 1994; Moriarty and Miikkulainen 1996) in which individual niches compete for the allocation of trials.

The use of multiple interacting subpopulations has also been explored as an alternate mechanism for coevolving niches using the so-called *island model* (Grosso 1985; Cohoon, Hegde, Martin, and Richards 1987; Tanese 1989; Whitley and Starkweather 1990). In the island model a fixed number of subpopulations evolve competing rather than cooperating solutions. In addition, individuals occasionally migrate from one subpopulation (island) to another, so there is a mixing of genetic material. The previous work that has looked at cooperating rather than competing subpopulations has involved a user-specified decomposition of the problem into species (see, for example, (Husbands and Mill 1991) or (Hills 1990)).

Cooperative coevolutionary algorithms (CCAs) combine and extend ideas from these earlier systems in several ways. As shown in figure 1, a CCA consists of a collection of evolutionary algorithms, each attempting to evolve subcomponents (species) which are useful as modules for achieving higher level goals. For example, in the neural network domain each species may represent a subnet whose output is correlated with the presence or absence of a particular input feature. Unlike the island model, the individuals from the separate subpopulations do not migrate; therefore, no interbreeding occurs. The existence of distinct subpopulations eliminates haphazard and often destructive recombination between dominant species.

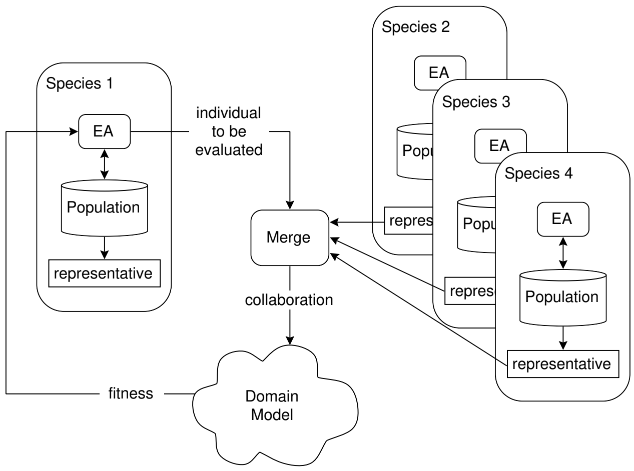

Figure 1: Cooperative coevolutionary architecture from the perspective of species number one

Complete solutions are obtained by assembling representative members of each of the species present. Credit assignment at the species level is defined in terms of the fitness of the complete solutions in which the species members participate. This provides evolutionary pressure for species to cooperate rather than compete. However, competition still exists among individuals within the same subpopulation.

In the system used in this paper, the evolution of each species was handled by a standard GA (although we could just as easily have chosen some other evolutionary algorithm). We emphasize this choice by adopting the more specific terminology CCGA.

3 EVOLVING CASCADE NETWORKS

In this section we describe the cascade network topology in greater detail and present both a traditional single-population GA approach and a CCGA approach to cascade network evolution. There have been many previous publications on the combination of genetic algorithms and neural networks. For a survey see (Schaffer, Whitley, and Eshelman 1992).

A cascade network is a form of multilayer feed-forward artificial neural network in which all input nodes have direct connections to all hidden nodes and to all output nodes. Furthermore, the hidden nodes are ordered and their outputs are cascaded; that is, each hidden node sends its output to all later hidden nodes and to all output nodes. A cascade network with two input nodes, a bias signal, two hidden nodes, and two output nodes is shown in figure 2. There is a connection weight corresponding to each box located on the line intersections in the figure. The significance of the three box colors will be discussed below.

Cascade networks were originally used in conjunction with the cascade correlation learning architecture (Fahlman and Lebiere 1990). Cascade correlation constructs and trains the network one hidden node at a time using a gradient descent technique. Similar architectures in which a GA is used as a replacement for gradient descent have also been explored (Potter 1992; Karunanithi, Das, and Whitley 1992).

Because cascade networks have a well defined topology only the connection weights and network size need to be evolved. A direct approach to the evolution of neural network connection weights is to create a population of individuals whose chromosomes represent connection weight assignments for the network. The connection weights may be encoded in the chromosomes simply as a string of floating point numbers or with a more traditional GA binary encoding in

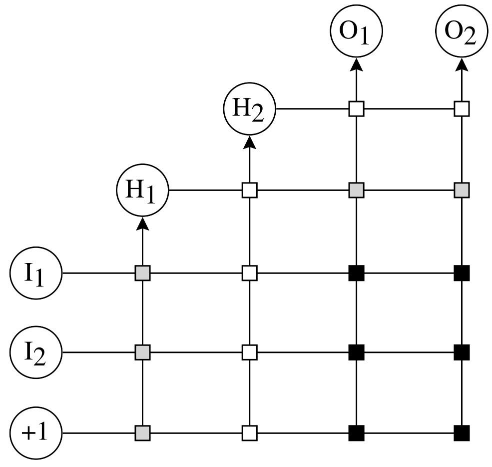

Figure 2: Example cascade network

which each real-valued connection weight is mapped to a segment of a string of ones and zeros. The population is initialized with individuals consisting of random connection weight values. The GA is then used to adjust the connection weights by evolving individuals in the population until a member emerges with an acceptable level of fitness. The fitness of an individual is determined by computing the sum-squared error as training patterns are fed forward through the network[1]. Individuals with lower sum-squared errors are considered more highly fit.

Cascade networks were designed to be constructed one hidden node at a time. A traditional single-population GA can incrementally construct these networks by initially evolving a population of networks without hidden nodes. If evolution stagnates before a network with sufficiently high fitness has been found, additional random values representing weighted connections of a new hidden node are added to the end of each chromosome. This cycle of evolving until stagnation and lengthening the chromosomes will continue until an acceptable network is found or until a predetermined number of network evaluations have been performed. This approach will be referred to as the "standard GA" throughout the rest of this paper.

For example, if a cascade network of three inputs[2] and two outputs as shown in figure 2 were being evolved the initial population would consist of chromosomes representing the six connection weights associated with the direct connections from the input nodes to the output nodes marked with black boxes.

[1] Supervised learning is assumed in which a set of input vectors and desired outputs exist for evaluating the network

[2] The bias signal is simply a fixed-value input

This population would be evolved until improvement stagnates, which could be determined by monitoring the increase in fitness from one generation to the next. Five random values would then be added to the end of each chromosome to represent the connection weights associated with the first hidden node. These weights are marked with gray boxes in the figure. Evolution of the population would proceed until stagnation is again detected. At this point, six additional random values would be added to the end of each chromosome representing the connection weights associated with the second hidden node marked with white boxes.

To evolve a cascade network with a CCGA we decompose the network into substructures and map the substructures into species. The decomposition we have chosen corresponds to the chromosome segments described in the proceeding paragraph. The first species consists of individuals representing the weights on the direct connections from the input nodes to the output nodes (black boxes), the second species consists of individuals representing the weights belonging to the first hidden node (gray boxes), and so on. Using this decomposition scheme, a cascade network with k hidden nodes would be decomposed into $k+1$ species.

As with the standard GA, the CCGA constructs the cascade network in stages. Initially only the first species representing the direct input to output node connection weights exists. This species is evolved until improvement stagnates. A new species will then be created to evolve weights belonging to the first hidden node. Now that two species exists, neither the first species nor the second species represent complete solutions. To evaluate one of these partial solutions it will first be combined with an individual from the other species to form a complete network. We use the *current best* individual from the other species to construct this collaboration. Evolution of these two species proceeds in parallel until stagnation is again detected. At this point a third species is created to evolve weights belonging to the second hidden node. This cycle continues until a termination condition is achieved as in the standard GA.

4 EMPIRICAL RESULTS

The cooperative coevolutionary approach was evaluated by comparing its performance with the performance of the standard GA described in the previous section on the problem of evolving cascade networks for computing N-bit parity. The primary metric for comparing the algorithms was the sum-squared error produced by the evolved networks. Additional metrics included the consistency of solutions over multiple runs and the size of the networks produced.

The N-bit parity problem has N inputs and a single output representing whether there is an even or odd number of one-bits in the input. The experiments described here were all performed on the 6-bit version of this problem. This was implemented with a cascade network consisting of six input nodes, a bias signal, a single output node, and an unspecified number of hidden nodes. The actual number of hidden nodes was evolved along with the connection weights as previously described. A network output of $+0.5$ represented odd parity and an output of -0.5 represented even parity. The training set for this problem consisted of all sixty-four (2^N) correctly classified 6-bit patterns.

The implementations of the two algorithms were as similar as possible to enable them to be compared solely on the use or nonuse of coevolution. Both the CCGA and the standard GA implementations encoded each real-valued connection weight as a six-bit segment of a binary string, e.g., a chromosome representing ten connection weights would be realized as a sixty-bit string. Other attributes in common included a population size of 100, a mutation rate of 0.001, a two-point crossover rate of 0.6, fitness scaling, and an elitist strategy in which a single copy of the best individual is always preserved from one generation to the next. The implementations also used identical procedures for determining when to add a new hidden node and had identical termination conditions.

The sum-squared error over all sixty-four training patterns was used for each evaluation (trial) of an individual. Table 1 summarizes the sum-squared error produced by the best network from the final generation of each of ten runs. Each run was terminated after 100,000 trials or when a network capable of correctly classifying all training patterns was found. As the table shows, the CCGA networks performed better than those evolved by the standard GA both in terms of the final sum-squared error produced and with respect to the consistency of the solution from one run to the next. Statistical significance of the sum-squared error data was verified using a two-sample t-test. This resulted in a p-value of 0.0053.

Table 1: Comparison of final sum-squared error produced in ten runs of 6-bit parity problem

Algorithm	*Mean*	*Min*	*Max*	*Stdev*
CCGA	0.80	0.48	1.21	0.24
Standard GA	4.79	0.78	9.54	3.44

The graph in figure 3 shows the sum-squared error performance curves for the “best so far” networks evolved by the two algorithms averaged over ten runs. This graph shows the superiority of the networks evolved by CCGA throughout the entire evolutionary period.

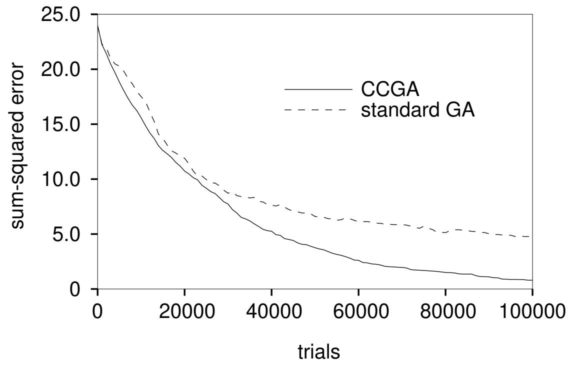

Figure 3: Average sum-squared error performance curve comparison in ten runs of 6-bit parity problem

The graph in figure 4 shows the hidden node creation of the two algorithms averaged over ten runs. Although the hidden node creation pattern of the two algorithms is similar, the graph shows that CCGA was able to achieve better results with fewer hidden nodes.

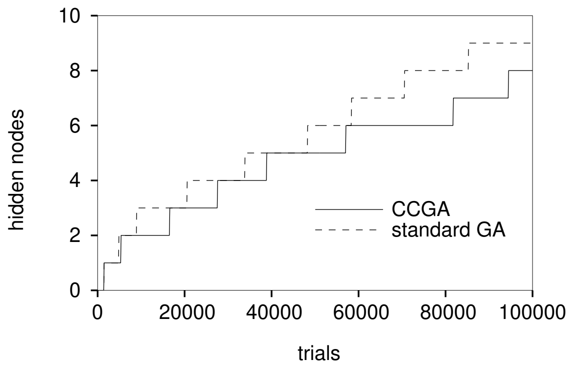

Figure 4: Average hidden node creation in ten runs of 6-bit parity problem

5 CONCLUSIONS AND FUTURE WORK

A general framework for cooperative coevolution has been presented in which a collection of evolutionary algorithms running in parallel evolve subcomponents which are combined into a single composite structure capable of being evaluated on a top level goal. Because credit assignment at the species level is defined in terms of the fitness of the complete solutions in which the species members participate, there is evolutionary pressure for individuals to collaborate rather than compete with other individuals in coevolving species.

We have extended the CCA architecture to support the dynamic creation of species and applied it to the problem of evolving cascade networks for parity computation. Results have been presented in which the coevolutionary architecture produces higher quality solutions in fewer evolutionary trials when compared with a traditional single-population GA approach. Although we have achieved considerable performance improvements, our primary motivation has been a better understanding of issues related to the evolution of interacting co-adapted subcomponents.

Future studies will explore less structured domains and focus on the area of emergent problem decomposition.

Acknowledgements

This work was supported in part by a student contract with the Navy Center for Applied Research in Artificial Intelligence.

References

Cohoon, J., S. Hegde, W. Martin, and D. Richards (1987). Punctuated equilibria: A parallel genetic algorithm. In J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 148–154. Lawrence Erlbaum Associates.

Davidor, Y. (1991). A naturally occuring niche & species phenomenon: The model and first results. In R. Belew and L. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 257–263. Morgan Kaufmann.

de Garis, H. (1990). Building artificial nervous systems using genetically programmed neural network modules. In B. Porter and R. Mooney (Eds.), *Proceedings of the Seventh International Conference on Machine Learning*, pp. 132–139.

De Jong, K. (1975). *Analysis of Behavior of a Class of Genetic Adaptive Systems*. Ph. D. thesis, University of Michigan, Ann Arbor, MI.

Deb, K. and D. Goldberg (1989). An investigation of niche and species formation in genetic function optimization. In J. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 42–50. Morgan Kaufmann.

Fahlman, S. and C. Lebiere (1990). The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, Carnegie Mellon University.

Forrest, S., B. Javornik, R. Smith, and A. Perelson (1993). Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation 1*(3), 191–211.

Giordana, A., L. Saitta, and F. Zini (1994). Learning disjunctive concepts by means of genetic algorithms. In W. Cohen and H. Hirsh (Eds.), *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 96–104. Morgan Kaufmann.

Grosso, P. (1985). *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model*. Ph. D. thesis, University of Michigan, Ann Arbor, MI.

Hills, D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen (Eds.), *Artificial Life II*, pp. 313–324. Addison-Wesley.

Holland, J. and J. Reitman (1978). Cognitive systems based on adaptive algorithms. In D. Waterman and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*. Academic Press.

Husbands, P. and F. Mill (1991). Simulated co-evolution as the mechanism for emergent planning and scheduling. In R. Belew and L. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 264–270. Morgan Kaufmann.

Karunanithi, N., R. Das, and D. Whitley (1992). Genetic cascade learning for neural networks. In *COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 134–145. IEEE Computer Society Press.

Moriarty, D. E. and R. Miikkulainen (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning 22*(1). In Press.

Potter, M. (1992). A genetic cascade-correlation learning algorithm. In *COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 123–133. IEEE Computer Society Press.

Potter, M. and K. De Jong (1994). A cooperative coevolutionary approach to function optimization. In Y. Davidor and S. H.-P. (Eds.), *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pp. 249–257. Springer-Verlag.

Potter, M., K. De Jong, and J. Grefenstette (1995). A coevolutionary approach to learning sequential decision rules. In L. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann.

Schaffer, J., D. Whitley, and L. Eshelman (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 1–37. IEEE Computer Society Press.

Tanese, R. (1989). Distributed genetic algorithms. In J. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 434–439. Morgan Kaufmann.

Whitley, D. and T. Starkweather (1990). Genitor II: a distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence 2*, 189–214.