Coevolution Learning: Synergistic Evolution of Learning Agents and Problem Representations

Lawrence Hunter

National Library of Medicine
8600 Rockville Pike
Bethesda, MD 20894
hunter@nlm.nih.gov

Abstract

This paper describes exploratory work inspired by a recent mathematical model of genetic and cultural coevolution. In this work, a simulator implements two independent evolutionary competitions which act simultaneously on a diverse population of learning agents: one competition searches the space of free parameters of the learning agents, and the other searches the space of input representations used to characterize the training data. The simulations interact with each other indirectly, both effecting the fitness (and hence reproductive success) of agents in the population. This framework simultaneously addresses several open problems in machine learning: selection of representation, integration of multiple heterogeneous learning methods into a single system, and the automated selection of learning bias appropriate for a particular problem.

Introduction

One clear lesson of machine learning research is that problem representation is crucial to the success of all inference methods (see, e.g. (Dietterich, 1989; Rendell & Cho, 1990; Rendell & Ragavan, 1993)). However, it is generally the case that the choice of problem representation is a task done by a human experimenter, rather than by an automated system . Also significant in the generalization performance of machine learning systems is the selection of the inference method's free parameter values (e.g. the number of hidden nodes in an artificial neural network, or the pruning severity in a decision tree induction system), which is also a task generally accomplished by human "learning engineers" rather than by the automated systems themselves.

The effectiveness of input representations and free parameter values are mutually dependent. For example, the appropriate number of hidden nodes for an artificial neural network depends crucially on the number and semantics of the input nodes. This paper describes exploratory work on a method for simultaneously searching the spaces of representations and parameter settings, using as inspiration a recent mathematical model of the coevolution of genetics and culture from anthropology (Durham, 1991). An important additional feature of this work is that is provides a simple, seamless and effective way of synergistically combining multiple inference methods in an integrated and extensible framework. When only a single inference method is used, this framework reduces to a variant on constructive induction. However, with multiple and diverse learning agents, the system is able to generate and exploit synergies between the methods and achieve results that can be superior to any of the individual methods acting alone.

The work presented here fits into a growing body of research on these issues. The importance of bringing learning engineering tasks within the purview of the theory and practice of automated systems themselves was described at length in (Schank, et al., 1986). Some effective computational methods for aspects of this task have been reported recently. (Kohavi & John, 1995,) describes an automated method for searching through the space of possible parameter values for C4.5. Genetic algorithms have also been used to search the space of parameter values for artificial neural networks (Yao, 1993). There has also been recent work on selecting appropriate ("relevant") subsets of features from a superset of possible features for input representation in machine learning ((Langley, 1994) is a review of 14 such approaches), as well as a long history of work on constructive induction ((Wnek & Michalski, 1994) includes a review, but see also (Wisniewski & Medin, 1994) for critical analysis of this work. This paper describes an exploratory approach that appears to have promise in addressing these issues in an integrated way.

Background

The idea of coevolution learning is to use a kind of genetic algorithm to search the space of values for the free parameters for a set of learning agents, and to use a system

metaphorically based on recent anthropological theories of cultural evolution to search through the space of possible input representations for the learning agents. This section provides some brief background on these ideas.

Evolutionary algorithms are a weak search method related to, although clearly distinguishable from, other weak methods (e.g. beam search). They are based on a metaphor with naturally occurring genetic evolution. In general, evolution has three components: inheritance, or the passage of traits from one individual to another: variation, of the generation of novel traits or combinations of traits; and selection, a competition between individuals based on their traits that effects the probability that an individual will have descendants that inherit from it. There are many ways of building computational models that evolve, including genetic algorithms, genetic programming and evolutionary strategies; see (Angeline, 1993) for an excellent survey and analysis of the approach.

Anthropological theories of culture have been converging for some time on what are now known as "ideational" theories. They hold that culture "consists of shared ideational phenomena (values, beliefs, ideas, and the like) in the minds of human beings. [They refer] to a body or 'pool' of information that is both public (socially shared) and prescriptive (in the sense of actually or potentially guiding behavior)." (Durham, 1991), p. 3. Note that this view is different from one that says culture *is* some particular set of concrete behavior patterns; it instead suggests that culture is just one of the factors that shapes behavior. An individual's phenotype (here, its behavior) is influenced by its genotype, its individual psychology and its culture. Durham summarizes "the new consensus in anthropology regards culture as systems of symbolically encoded conceptual phenomena that are socially and historically transmitted within and between populations" (p. 8-9). This historically rooted, social transmission of ideational elements can be analyzed as an evolutionary process. The characterization of that evolutionary process (and its relationship to genetic evolution) is the subject of Durham's book, and also the subject of a great deal other research dating back at least one hundred years (much of which is surveyed by Durham).

There are several significant differences between cultural and genetic evolution. Cultural traits are transmitted differently than genetic ones in various ways. It is possible to transfer cultural traits to other members of your current generation, or even to your cultural "parents." It is also possible for one individual to pass cultural traits to very many more others than he or she could genetically. The selection pressure in the competition among cultural entities is not based on their reproductive fitness as with genetic evolution, but on the decisions of individuals to adopt them, either through preference or imposition. And most importantly for the purposes of this work, cultural evolution involves different sources of variation than genetic evolution. Rather than relying on mutation or sexual recombination of genomes to provide novel genetic variants, culture relies on individual's own discovery and synthesis as the source of novelty. By providing a computational model in which individuals are able to learn from their experiences and share what they have learned, it becomes possible to simulate a cultural kind of evolution.

A key aspect of any model of cultural evolution is the specification of the smallest unit of information that is transmitted from one agent to another during cultural transmission, christened the "meme" by Richard Dawkins. Although there is a great deal of argument over what memes are (ideas, symbols, thoughts, rules patterns, values, principles, postulates, concepts, essences and premises have all been suggested), and a great deal of theoretical analysis describing how memes compete, are transformed, interact with genes, etc., I am aware of no attempts to operationalize the term so that it would be possible to build computational simulations of populations of memes.

A meme must play several roles in a simulation of cultural inheritance. First, it must be able to have an effect on the behavior of the individuals in simulation. Second, individuals must be able to create new memes as a result of their experiences (e.g. by innovation, discovery or synthesis). Third, it must be possible for other individuals in the simulation to evaluate memes to determine whether or not they will adopt a particular meme for its own use. In the sections below, I will show how the input representation used by a machine learning system can be used to meet these requirements.

A Formal Definition of Coevolution Learning

A coevolution learning system functions by evolving a population of learning agents. The population is defined by a classification task **T**, a set of classified examples of that task expressed in a primitive representation $\mathbf{E_p}$, a fitness function for agents f_A , and a set of learning agents **A**:

$$P_{coev} \equiv \{T, E_p, f_A, A\}$$

A fitness function for agents maps a member of the set **A** to a real number between 0 and 1. For convenience, it is useful to define $\mathbf{P_L}$ to be the subset of $\mathbf{P_{coev}}$ where all the agents use learning method **L** (see below).

Each learning agent **A** is defined by a learning method **L**, a vector of parameter values **v**, a fitness function for memes f_m, and an ordered set of problem representation transformations **R**:

$$A_i \equiv \{L, v, f_m, R\}$$

The vector of parameter values may have different length for different learning methods. Each member of the set of problem representation transformations $\mathbf{R_i}$ is a mapping from an example ($\mathbf{e_p} \in \mathbf{E_p}$) to a value which is a legal element of an input representation for **L** (e.g. a real number, nominal value, bit, horn clause, etc.). The individual mappings are called *memes*. The ordered set of memes ($\mathbf{R_i}$) is called the agent's *memome*, and the vector of parameter values is called its *genome*. The fitness function for memes $\mathbf{f_m}$ is a function that maps a member of the set $\mathbf{R_i}$ to a real number between 0 and 1.

A transformed example $\mathbf{e_i}$ is the result of sequentially applying each of the problem representation transformations $\mathbf{r_i} \in \mathbf{R_i}$ to an example $\mathbf{e_p} \in \mathbf{E_p}$. The application of the set of problem representation transformations to the set of examples in primitive representation results in a set of transformed examples, which is called $\mathbf{E_i}$. When $\mathbf{E_i}=\mathbf{E_p}$, the transformation is said to be the identity.

Given a population as defined above, the process of coevolution is defined in the pseudocode in figure 1. The creation of the next generation of agents is a minor variant on traditional genetic algorithms. Instead of having the genome be "bits" it is a vector of parameter values. The crossover, mutation and fitness proportional reproduction functions are all identical with the related operations on bit vectors in genetic algorithms. It is, of course, possible to transform parameter vectors into bitstring representations themselves, if it were desirable. In addition to using parameter vectors instead of bits, the other difference is that each subpopulation using a particular learning method (the P_L's) has its own type of parameter vector, since the free parameters and their legal values vary among learning methods. These parameter vectors may be of different sizes, so crossover can only be applied with members of the same subpopulation.

The main difference between coevolution learning and other evolutionary methods derives from the creation and exchange of memes. The meme creation process takes the output of learning agents that have been applied to a particular problem, and identifies combinations of the input features that the learner determined were relevant in making the desired distinction. The process of parsing output and creating new memes is specific to each learning method. For example, a program that learns rules from examples might create new memes from the left hand sides of each of the induced rules. Or, a program that learned weights in a neural network might create new memes' that were the weighted sum of the inputs to each of its hidden nodes (perhaps thresholded to remove marginal contributions). Specific meme generation methods are discussed in more detail in the implementation section, below.

Initialize the population with random legal parameter vectors and the identity representation transformation.

* **Determine the phenotype** of each agent *i* by applying learning algorithm L_i to transformed examples E_i using parameter values v_i using K-way cross-validation. The phenotype is an ordered set of: the output of the learner's cross-validation runs, the cross validation accuracies and the learning time.

Determine the fitness of each agent by applying f_A to the phenotype of each agent.

Create new memes by parsing the output in each learner's phenotype and extracting important feature combinations. The union of all memes generated by all members of the population is called the meme pool.

Exchange memes. For each agent i, apply its meme fitness function F_m to elements of the meme pool. Select the agent's target number of memes (a free parameter) from the meme pool with a probability proportional to the fitness of the memes.

Repeat from * meme-transfers-per-generation times

Determine phenotype of the agents with their new memes

Determine the fitness of the agents

Create the next generation of agents by mutation, crossover and fitness proportional reproduction. Agents whose genomes are mutated keep their memomes intact. For each pair of agents whose genomes are crossed over to create a new pair of agents, the memomes of the parents are arbitrarily assigned to the offspring.

Repeat from * until a member of the population can solve the problem

Figure 1: Pseudocode of coevolution algorithm

An Implementation of a Coevolution Learner

COEV (short for "the beginning of coevolution") is a simple implementation of the coevolution learning framework, written in Carnegie Mellon Common Lisp 17f and the PCL Common Lisp Object System, running on a Silicon Graphics Indigo^2 workstation. The current implementation includes the C4.5 decision tree induction system and C4.5rules rule extraction tool (Quinlan, 1991), the LFC++ constructive induction program (Rendell & Ragavan, 1993; Vilalta, 1993) and the conjugate gradient descent trained feedforward neural network (CG) from the UTS neural network simulation package (van Camp, 1994). Each learning method is associated with an object class which defines how to execute it, how to parse the results returned, and what the vector of free parameters is for that type of learner.

Most of the documented parameters for each of type of learning program is included in the free parameter vector for that system, along with a specification of either an upper and lower bound for the parameter's values or a list of possible parameter values. For example, the parameter vector for the UTS conjugate gradient descent learner includes the number of hidden nodes in the network, the maximum number of iterations before halting, the output tolerance (specifying how close to 1 or 0 an output has to be to count as true or false), a flag for whether or not to use competitive learning on the output, two nominal parameters specifying the kind of line search and direction finding method to use, and three parameters specifying the maximum number of function evaluations per iteration, the minimum function reduction necessary to continue the search and the maximum slope ratio to continue the search. Theses parameters are explained more fully in the documentation available with the source code.

Each learner must also have a method for extracting new features from its output. LFC++, like other constructive induction programs, specifically defines new features as part of its output. For C4.5, the output of the C4.5rules tool was parsed so that each left hand side of each rule was reified into a new feature definition. For the neural network learner, a new feature was created for each hidden node defined by the weighted sum of the inputs to that node, with any input whose contribution to that sum was less than the threshold of the node divided by the number of inputs removed from the sum. These extracted features are added to the meme pool.

Memes (feature combinations) must be assigned fitnesses by agents. It is possible for each agent to have its own method of determining meme fitness. However, a simplified method is used in COEV. Whenever a program generates a new meme, its fitness is defined to be the average "importance" of the feature combination, weighted by the accuracy of the learners that used the meme. In Durham's terms, the inventor of the new meme "imposes" its belief in its value on others. In addition, receivers of memes have a parameter which determines how much weight they put on memes they generate themselves versus memes generated by others. This mechanism could be straightforwardly generalized to weight the fitnesses of memes generated by different classes of agents differently. Importance is defined differently for different learning methods. For C4.5 and LFC++, the importance of a feature combination is the ratio of correctly classified examples that triggered the rule containing the feature to the total number of occurrences of the feature. So, for example, if "A and not B" is a feature extracted from a single C4.5 learner that had a 80% cross-validation accuracy, and 9 of the 10 examples the feature appeared in were classified correctly by the rule containing the feature, it's importance would be 9/10 and its fitness would be 0.9 * 0.8 = 0.72. For UTS, the importance of a feature is the absolute value of the ratio of the weight from the hidden node that defined the feature to the threshold of the output unit. If there is more than one output unit, it is the maximum ratio for any of the output units. Features that play a significant role in learners that are accurate have higher fitness than features that do not play as much of a role or appear in learners that are less accurate. It is possible for a feature to have relatively low prevalence in the population and yet still be important if it tends to generate correct answers when it is used.

In addition to defining the fitness of features, COEV must define the fitness of learners themselves for the evolution of parameter values. The fitness function for learners was selected to evolve learners that are accurate, robust and fast:

$$f(A_i) = C(A_i) - \sqrt{S(A_i)} - k\left(\frac{t_{A_i} - t_A}{S(t_A)}\right)$$

where $C(A_i)$ is the cross-validation accuracy of the agent A_i, $S(A_i)$ is the standard deviation of that accuracy, t_{A_i} is the time it took for that agent to learn, t_A is the mean execution time for that type of agent, $S(t_A)$ is the standard deviation of that mean, and k is a constant that trades off the value of accuracy versus that of learning time. Execution time is measured as the number of standard deviations from the mean learning time for that class of learners so that classes of learners with different training times can coexist in the same population. In the COEV system, k was selected to be 3 and accuracies and standard deviations are measured in percentage points. So, a learner that had a mean accuracy of 80%, a standard deviation of 9% and took 1 standard deviation less than the mean training time for that class of learner would get a fitness score of $80 - \sqrt{9} - (-1) = 78$.

Several other aspects of the framework must be specified in order to implement the system. Meme definitions must be stored in a canonical form so that it is possible to detect when two or more generated memes are in effect identical and should have their fitness scores combined. Memes in COEV are represented as boolean combinations of primitives or mathematical formula over primitives, which

can be straightforwardly compared, although the extension to first order predicate calculus would make this a more difficult problem. The number of memes that an agent uses (i.e. the dimensionality of its input representation) is treated as a free parameter of the agent, and allowed to vary from two to twice the number of primitives. In addition, agents are allowed to prefer memes of their own creation to memes created by other learners. A single parameter, ranging over [0,1] specifies the internal meme preference of each agent.

Simulation Results on a Simple Test Problem

The COEV system was applied to an artificial problem designed to be moderately difficult for the machine learning programs included in the system. The problem was to identify whether any three consecutive bits in a nine bit string were on, e.g. 011011011 is false, and 001110011 is true. This problem is analogous both to detecting a win in tic-tac-toe and to solving parity problems, which are known to be difficult for many types of learning systems. There are 512 possible examples, and the problem is simple enough so that the machine learning systems used (even the neural network) run reasonably quickly. This is important, since each program is executed a large number of times (see the computational complexity section, below).

This problem was run on a system that used only LFC++ and the CG learners. The population consisted of 10 LFC++ learners and 10 CG learners, and was run for 10 generations. Six fold cross-validation was used, and there was one meme exchange per generation. This rather modest problem therefore required 2400 learner executions, taking more than four days of R4400 CPU time. (The CPU time use was dominated by the 1200 neural network training runs.) Despite the large amount of computation required for this simple problem, it is reasonable to hope that the amount of computation required will grow slowly with more difficult problems (see the computational complexity section, below).

This simulation illustrated three significant points. First, the coevolution learning system was able to consistently improve as it evolved. The improvement was apparent in both the maximum and the average performance in the population, and in both the cross-validation accuracies of the learning agents and in their fitnesses (i.e. accuracy, speed and robustness). Figure 2 shows these results. In ten generations, the average fitness of the population climbed from 51%% to 75%, and the standard deviation fell from 21% to 12.5%. Looking at the best individual in the population shows that the maximum fitness climbed from 84% to 94%, and the maximum cross-validation accuracy climbed from 82.3% to 95%. Average execution time fell by more than 25% for both classes of learners (it fell somewhat more sharply, and in much greater absolute magnitude for the neural network learners.)

Figure 3 illustrates the separate contributions of the genetic evolution component and the memetic evolution component compared to coevolution in the performance of the best individual learning agent in the population. Consider first the purely genetic evolution of the free parameter vectors. When there is no memetic component, there is no interaction between the different classes of learners, since crossover and other sources of genetic variation apply only to a specific type of learner. Using genetic search to find the best free parameter values for a neural network is known to be slow (Yao, 1993). In this genetic-only neural network simulation, only a small improvement was found in the fifth generation. Similarly for the genetic evolution of LFC++'s free parameters, a modest difference was found in the sixth generation. Memetic evolution alone (without changing any of the free parameter values) showed a more significant effect. Because both the neural network and the constructive induction program make contributions to memetic evolution, any synergies between them will appear in the memetic-only curve (see discussion of figure 4, below). However, the steady rise of the coevolution curve, compared to the memetic-evolution only curve suggests that some adjustment of the free parameter values to match the representation changes induced by memetic transfer may have a positive effect. At the end of 10 generations, the maximum accuracy of the coevolved population is more than 5 percentage points higher than the memetic only population. However, it is worth noting that the memetic only population equaled that performance figure earlier in the simulation, and then drifted away.

Figure 4 illustrates the synergy that coevolution finds between the CG agent population and the LFC++ agent population. When run with only a single type of learner, the memetic evolution part of COEV becomes a kind of constructive induction program, where the output of the learner is feed back as an input feature in the next iteration. Since LFC++ is already a constructive induction program, it is somewhat surprising to note that LFC++ agents coevolving with each other still manage a small amount of improvement. Perhaps this is due to the fact that the appropriate features for this problem have three conjuncts, and LFC++ under most free parameter values tends to construct new features from pairs of input features. CG alone does not do very well on this problem. Even when coevolved with itself, its most accurate agent improves only from about 60% to a bit over 70% in ten generations. However, when these two learning methods are combined in a single coevolution learner, the results are clearly better than either of the methods used alone. The gap between the best performance of the pair of methods and the best single method tends to be about 5 percentage points in this simulation, and the difference tends to be larger as the population evolves.

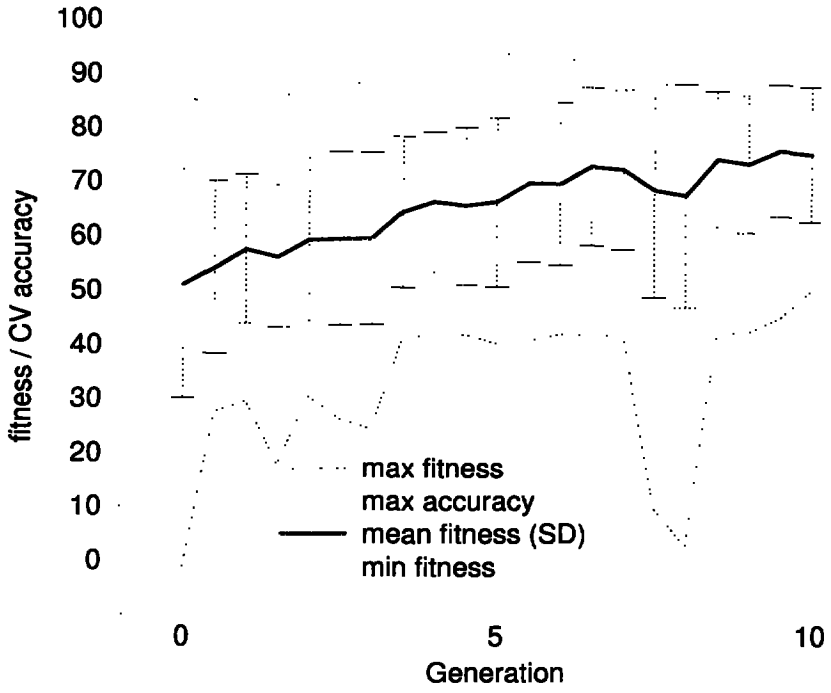

Figure 2: Fitness of best, worst and population average (with S.D.) fitness over 10 generations.

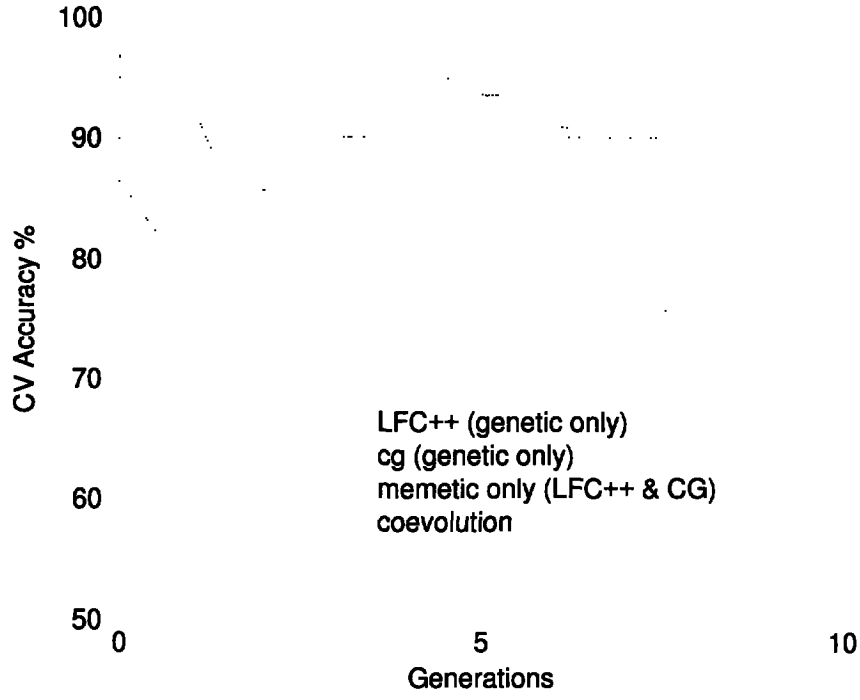

Figure 3: Genetic evolution only, memetic evolution only and coevolution

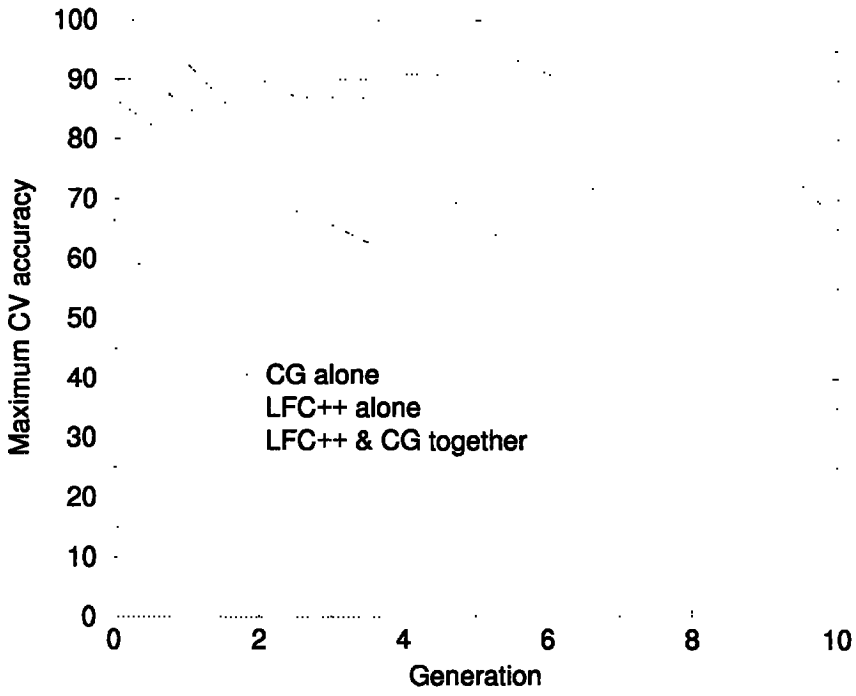

Figure 4: Synergistic interaction of the different types of learners

Why does Coevolution Appear to Work?

This is intended to be an exploratory paper, describing a novel approach to learning. It does not present any proofs nor any detailed empirical evaluations. However, it is possible to speculate about why the method appears to work, and this speculation may be useful in guiding future research in this area.

Constructive induction systems have not been able to solve problems that are a great deal more difficult than their root form of induction, even though they automatically generate relevant feature combinations to use in representation. One possible explanation is that there features are constructed with the same learning biases as the method that uses those features to build concepts, limiting the value of composing these actions. Since the learning agents in coevolution learning express a variety of biases, it may be that features discovered by one learner can more significantly facilitate learning by another learner which embodies a different bias.

Coevolution learning has a different advantage over most other forms of multistrategy learning. In these other forms, the relationships between different learning methods tend to be predefined by the learning researcher (see the introduction and many of the chapters in (Michalski & Tecuci, 1994)). For example, the KBANN system (Towell, et al., 1990) uses a domain theory to specify a neural network architecture, then trains the net and extracts a revised domain theory. Flexibility in the type of learning used and the order in which learning agents are executed may make possible important synergies that are difficult to capture with a predefined relationship.

My own earlier work was dedicated to building a planner which could select among and combine multiple learning methods based on reasoning about explicit goals for knowledge (Hunter, 1989; Hunter, 1990; Hunter, 1992) . However, first principles planning for achieving learning goals is even more difficult than planning for physical goals. Due to the large inherent uncertainty in predicting the outcome of the execution of any particular learning method, it is extremely difficult to reason about the likely outcome of any significant composition of learning actions. I also pursued the use of plan skeletons, templates and strategies, which are the usual response to the difficulties of first principles planning, but these run into the same problems in maintaining flexibility as other predefined approaches to multistrategy learning. Coevolution learning is able to flexibly combine learning methods (even in novel ways) without having to reason ahead of time about what the outcome of that combination will be.

The reason that coevolution appears to work better than just memetic evolution alone may be the two way linkage between parameter values and representations. Not only does it appear to be the case that which parameter values produce the best outcome depends on the input representation used, but it also appears that the selection of parameter values has a significant effect on what new

features are constructed. This seems especially relevant when considering that two of the parameters in COEV specify how many input features each learner will use, and what the propensity of each learner is for taking those features from external sources is.

Computational Complexity and Parallelization Issues

One of the drawbacks to coevolution learning is the large amount of computation required. Doing an evolutionary search is slow in general, and when the evaluation of a single individual can take minutes or hours (as training a neural network can) then the problem is exacerbated. There are three reasons to be optimistic that this approach can be used to solve large scale problems.

First, the fitness function for the population includes a time factor, which tends to increase the speed of learning as the evolution progresses. Average running time of both LFC++ and CG was substantially lower at the end of ten generations than it was at the beginning. The search through the space of parameter values not only finds values that provide good results, but also ones that provide them quickly.

Second, as better input representations are found, most learning methods increase their learning speed. Since the concepts learned from good representations tend to be simple (that's what makes a representation good), and most machine learning methods have a computational cost that is dependent on the complexity of the learned concept, as the system evolves better representations, the speed of learning should go up.

Finally, coevolution learning is well suited to parallelization. The vast majority of the computation is done by the learning agents, which are entirely independent of each other during learning. The only communication required is at the time of meme and gene exchange, where all agents must register their fitnesses and the fitnesses of their memes. It is also possible to relax even that requirement, so that agents use only a sample of the fitnesses from other agents to determine who to exchange memes and genes with. Since the selection of memes and genetic partners is probabilistic anyway, sampling will be indistinguishable from exhaustive search if the sample size is large enough. By using sampling, it is also possible to do asynchronous updating of the population.

Future directions

There are three current research goals for coevolution learning currently being pursued. The first is to empirically demonstrate its effectiveness on realistic problems. In particular, we are hoping to demonstrate that this approach solves problems that are very difficult for other machine learning or statistical approaches. Solving real world problems is likely to require additional learning methods, larger agent populations and longer evolution times.

The second goal is to add several new learning techniques to the system. In particular, we are interested in adding:

- a function finding system that can generate memes that identify mathematical relationships between features;
- a feature relevance metric (e.g. (Kira & Rendell, 1992)) that can be added to the meme fitness function;
- a relation learning system (e.g. (Quinlan, 1990) or one of the ILP learners) that can help the system transcend the limits of boolean feature combinations

Each of these desired additions presents challenges to the design of the current system that must be addressed.

Our third current goal is to add at least some coarse grained parallelism to the system so that it can be run on a network of workstations. Due to the nature of the computations, we expect nearly speedup nearly linear with the number of workstations used, up to the point where there is one workstation per agent.

Conclusion

Coevolution provides a powerful metaphor for the design of a machine learning system. Models of genetic evolution have already driven significant advances in machine learning. When nature added cultural evolution to genetic evolution, the result was spectacular. The computational model of culture proposed in this paper is tremendously empovershed in comparison to people. Nevertheless, the ability of even this drastically simplified model of coevolution to synergize the abilities of disparate learners appears promising.

References

Angeline, P. J. (1993) Evolutionary Algorithms and Emergent Intelligence. Ph.D. diss, Ohio State University.

Dietterich, T. (1989). Limitations on Inductive Learning. In Proceedings of Sixth International Workshop on Machine Learning, (pp. 125-128). Ithaca, NY: Morgan Kaufman.

Durham, W. H. (1991). *Coevolution: Genes, Culture and Human Diversity*. Stanford, CA: Stanford University Press.

Hunter, L. (1989) Knowledge Acquisition Planning: Gaining Expertise Through Experience. PhD diss. Yale University, Available as YALEU/DCS/TR-678.

Hunter, L. (1990). Planning to Learn. In Proceedings of The Twelveth Annual Conference of the Cogntive Science Society, (pp. 26-34). Boston, MA:

Hunter, L. (1992). Knowledge Acquisition Planning: Using Multiple Sources of Knowledge to Answer Questions in Biomedicine. *Mathematical and Computer Modelling*, 16(6/7), 79-91.

Kira, K., & Rendell, L. (1992). The Feature Selection Problem: Traditional Methods and a New Algorithm. In Proceedings of National Conference on AI (AAAI-92), (pp. 129-134). San Jose, CA: AAAI Press.

Kohavi, R., & John, G. (1995). Automatic Parameter Selection by Minimizing Estimated Error. In Proceedings

of ECML-95: The Eighth European Conference on Machine Learning.

Langley, P. (1994). Selection of Relevant Features in Machine Learning. In Proceedings of AAAI Fall Symposium on Relevance. New Orleans, LA: AAAI Press.

Michalski, R. S., & Tecuci, G. (Ed.). (1994). *Machine Learning IV: A Multistrategy Approach*. San Mateo, CA: Morgan Kaufman Publishers.

Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Machine Learning*, 5(3), 239-266.

Quinlan, J. R. (1991). C4.5. In Sydney, Australia: Available from the author: quinlan@cs.su.oz.au.

Rendell, L., & Cho, H. (1990). Empirical Learning as a Function of Concept Character. *Machine Learning*, 5(3), 267-298.

Rendell, L., & Ragavan, H. (1993). Improving the Design of Induction Methods by Analyzing Algorithm Functionality and Data-based Concept Complexity. In Proceedings of IJCAI, (pp. 952-958). Chambery, France:

Schank, R., Collins, G., & Hunter, L. (1986). Transcending Inductive Category Formation In Learning. *Behavioral and Brain Sciences*, 9(4), 639-687.

Towell, G. G., Shavlik, J. W., & Noordewier, M. O. (1990). Refinement of Approximate Domain Theories by Knowledge-Based Artificial Neural Networks. In Proceedings of Seventh International Conference on Machine Learning, .

van Camp, D. (1994). UTS/Xerion. In Toronto, Canada: Available by anonymous file transfer from ftp.cs.toronto.edu:/pub/xerion/uts-4.0.tar.Z.

Vilalta, R. (1993). LFC++. In Available from the author: vilalta@cs.uiuc.edu.

Wisniewski, E., & Medin, D. (1994). The Fiction and Nonfiction of Features. In R. Michalski & G. Tecuci (Eds.), *Machine Learning IV: A Multistrategy Approach* San Francisco, CA: Morgan Kaufmann.

Wnek, J., & Michalski, R. (1994). Hypothesis driven constructive induction in AQ17: A method and experiments. *Machine Learning* 14:139-168.

Yao, X. (1993). Evolutionary artificial neural networks. International Journal of Neural Systems, 4(3), 203-221.