Co-Evolving an Effective Fitness Sample: Experiments in Symbolic Regression and Distributed Robot Control

Brad Dolin
FX Palo Alto Laboratory, GeNeura Team, and Stanford University
PO 13525
Stanford, CA 94309
dolin@stanford.edu

Forrest H Bennett III
Pharmix Corporation
200 Twin Dolphin Drive, Suite F
Redwood Shores, CA 94065
1-650-637-1200 x16
forrest@pharmix.com

Eleanor G. Rieffel
FX Palo Alto Laboratory
3400 Hillview Avenue, Bldg 4
Palo Alto, CA 94304
1-650-813-7077
rieffel@pal.xerox.com

ABSTRACT
We investigate two techniques for co-evolving and sampling from a population of fitness cases, and compare these with a random sampling technique. We design three symbolic regression problems on which to test these techniques, and also measure their relative performance on a modular robot control problem. The methods have varying relative performance, but in all of our experiments, at least one of the co-evolutionary methods outperforms the random sampling method by guiding evolution, with substantially fewer fitness evaluations, toward solutions that generalize best on an out-of-sample test set.

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

General Terms
Algorithms, Performance, Reliability, Experimentation.

Keywords
Genetic programming, genetic algorithms, co-evolution, fitness cases, symbolic regression, robot control, distributed control.

1. INTRODUCTION
When using evolutionary methods to generate robust solutions, determining an effective set of fitness cases can be a show-stopping difficulty. When the problem space is large, fitness can be computed on only a small fraction of possible test cases. For simple problems, randomly chosen test cases may be effective, though nothing prevents such sets from being too easy, too hard, or unrepresentative of the entire space of problems. Choosing a sample by hand can work only when the experimenter has sufficient *a priori* knowledge. We investigate techniques for automatically generating effective sets of fitness cases.

In co-evolution, the fitness of an individual in one population is determined with respect to fitness cases sampled from the other population. For our purposes, the populations are asymmetric in that we have a population of learners – represented as S-expressions and evolved using genetic programming – and a population of problems – represented as linear strings and evolved using genetic algorithms.

2. RELATED WORK
Various researchers have explored ways to evolve an effective population of fitness cases [10], [11], [12], [17], [18], [19], [21]. We modify shared fitness and shared sampling [17] to allow for attributions of partial credit. Uniform sampling and distinction sampling, developed in this paper, are novel, as well as the mutation scheme for the population of test cases.

"Teams" of functions that cooperate to solve a symbolic regression problem have been co-evolved [20]. We are unaware of any previous work on using co-evolution to evolve test points for symbolic regression.

Much work has been done on co-evolving robot control software [7], [8], [14] [15], [16]. A limited amount of work has been done on developing decentralized control software for modular robots [3], [4], [22], including the use of single-population genetic programming to generate such control software [1], [2]. A scheme for co-evolving fitness cases for a modular robot task is described in [5], but with inconclusive results.

3. CO-EVOLVING THE FITNESS SAMPLE
We give an outline of the co-evolutionary algorithm implemented here:

1. Construct initial random populations
   a. Learners
   b. Problems
2. Choose a random sample of problems
3. Compete each learner against sample of problems
4. Compute fitness of each learner using shared fitness
5. Choose a sample of learners using uniform sampling
6. Evolve the next generation of learners using GP

7. Compete each problem against sample of learners
8. Compute the fitness of each problem using tractable shared fitness or distinction fitness
9. Choose a sample of problems using tractable shared sampling or distinction sampling
10. Evolve the next generation of problems using GA
11. Loop to 3

Each loop through the algorithm creates a new generation of individuals in each population.

4. FITNESS MEASUREMENT TECHNIQUES

Whereas it is relatively obvious what behavior to encourage in the learners – perform well against as many of the test cases as possible – it is less obvious what behavior to encourage in the problems. Therefore, while we use shared fitness for the population of learners, we experiment with two techniques – tractable shared fitness and distinction fitness – for the population of problems. Throughout this paper, higher fitness is better.

4.1 Shared Fitness

In this learner-specific technique, each problem is seen as a resource to be shared among the individuals that defeat it. We modify the standard algorithm [17] to allow partial credit in measuring the results against each fitness case.

Let N_j be the sum total of partial credit awarded to all learners pitted against a given test case j from the sample. Assume that an individual i received partial credit pc_{ij} when pitted against test case j. Then individual i will get pc_{ij} / N_j added to its fitness score. Individual i's total fitness will then be $\Sigma\, pc_{ij} / N_j$, over all test cases j. We use the negation of this sum when partial credit is negative. Fitness sharing maintains diversity by rewarding individuals not only for defeating opponents in the sample, but for defeating opponents that few other individuals defeat.

4.2 Tractable Shared Fitness

This problem-specific technique modifies the shared fitness algorithm by assigning the worst fitness value to "intractable" individuals. Here, intractable means that no learner scored a "hit" (defined later for each experiment) on this problem. Problems are rewarded for defeating many learners, and for defeating different learners, but not for being impossibly difficult (and, as such, uninformative).

4.3 Distinction Fitness

We modify a technique given in [6]. For an individual i, we determine all ordered pairs of learners that this problem is able to distinguish, i.e., score significantly better against the first than against the second. For each such pair p, we add h_p / N_p to i's distinction fitness, where h_p is the number of "hits" earned by the weaker individual of the pair during the most recent fitness evaluation, and N_p is a discount factor giving the total number of problems in the population which distinguish the pair p. Individual i's total fitness will then be $\Sigma\, h_p / N_p$, summed over all pairs p. The numerator rewards problems which are able to demonstrate weaknesses in adept individuals (i.e., individuals with many hits). The denominator punishes problems for failing to demonstrate novel distinctions.

4.4 Sampling Techniques

Sampling reduces the number of costly fitness competitions by selecting some subset of the current population for competition against each individual in the opponent population. The most recent competition results are used to select a sample that is likely to be informative.

4.4.1 Uniform Sampling

We use uniform sampling for the population of learners. First rank the entire population of learners by fitness. For a sample of size $|S|$, partition the population into $|S|$ equally sized bins and randomly choose one member from each bin for the sample.

4.4.2 Tractable Shared Sampling

We use tractable shared sampling in conjunction with tractable shared fitness for the population of problems. The goal is to choose a sample that rewards a diverse set of learners.

1) Calculate the "sample fitness" of each individual not in the current sample, SC. The sample fitness of an individual i is simply the fitness it would receive under tractable shared fitness computed over the population $SC \cup \{i\}$.
2) Add the individual with the highest sample fitness to the sample SC.
3) Loop to (1) until reaching the desired sample size, $|S|$.

4.4.3 Distinction Sampling

We use distinction sampling in conjunction with distinction fitness for the population of problems. The algorithm is the same as that for tractable shared sampling except that distinction rather than tractable shared fitness is used in step (1).

5. APPLICATION TO SYMBOLIC REGRESSION

We designed three symbolic regression problems to test the co-evolutionary techniques. The goal of symbolic regression is to discover a function that approximates a given set of points. The candidate solutions (learners) are S-expressions built from a set of mathematical functions and a terminal that stands for the independent variable.

Here each fitness case represents a value for the independent variable, at which the error of each learner (absolute distance from the value given by the target function) is evaluated. Best fitness is achieved by minimizing the total error at each fitness case. We terminate the run when any individual in the population matches, within some tolerance, each member from a set of out-of-sample points.

The first two problems are approximated well by simple functions except in small fractions of the domain. We investigate whether the co-evolutionary techniques we describe are able to evolve samples concentrated in these "interesting" regions. For the third problem, multiple interesting regions are present.

5.1 Learner GP Setup

5.1.1 Function Set

All three experiments use at least the following set of functions: `Add`, `Sub`, `Mul`, `ProtectedDiv`, `Sin`, `Cos`, `Exp`, `Rlog`.

`ProtectedDiv` returns 1 if the denominator is 0, and `(Rlog v)` returns 0 if `v == 0`, otherwise `ln(|v|)`.

5.1.2 *Terminal Set*

All four problems use at least the terminal X, which returns the value of the independent variable (as given by the current fitness case).

5.1.3 *Competition*

A competition occurs whenever a learner (GP tree) is evaluated on a particular fitness case (single value of the independent variable). The outcome, which depends on the error at the fitness case, is given by:

$$-\min\left(\left|\frac{targetValue - learnerValue}{100}\right|, 1\right).$$

A hit occurs whenever the error, $|targetValue - learnerValue|$, is less than or equal to 0.01.

5.1.4 *Evolution*

Parameter choices are based on those given in [1]. The internal node crossover rate is 79%; the leaf node crossover rate is 1%; the internal node mutation rate is 9%; the leaf node mutation rate is 10%; and the reproduction (cloning) rate is 1%. We use the generational breeding model with tournament selection, and a tournament size of 4. Elitism is used. The method for creating program trees in generation 0 and in mutation operations is the "ramped half and half" method [13]. The maximum depth for program trees in generation 0 is 6. The maximum depth for program trees created by crossover and mutation is 10. The maximum depth of subtrees created by mutation is 4. We use strongly typed genetic programming [9]. The population size is 5000, and we sample 50 individuals for problem testing.

5.2 Problem Random Sampling Setup (Control Condition)

Ten fitness cases are chosen each generation from a random uniform distribution over [-100,100].

5.3 Problem GA Setup (Experimental Conditions)

5.3.1 *Allele*

Each problem consists of a single allele, a floating point number which gives the value of the independent variable. The allele is initialized to a random value uniformly distributed over [-100,100].

5.3.2 *Competition*

For a problem, the result of a competition is simply the negation of the outcome of the competition for the learner.

5.3.3 *Fitness*

We experiment with both tractable shared fitness and distinction fitness. With distinction fitness, a problem is said to distinguish between a pair of learners if the competition results differ by at least 0.01.

5.3.4 *Evolution*

The population of problems is sorted by fitness. Then, each problem (a single floating point number) undergoes Gaussian mutation, in which the value is replaced with a new floating point value normally distributed about the original value with standard deviation equal to

$$\left[\frac{rank\ in\ population}{population\ size}\right]^3 (domain\ length).$$

The technique conducts a more aggressive search around less fit individuals.

The population size is 200, from which we sample 10 individuals for learner testing. Note that the number of fitness competitions per generation for the co-evolutionary methods (60,000) is larger than the number of fitness competitions per generation for the random method (50,000). To control for this difference when reporting results, we use fitness competitions instead of generations as our principal measure of required computer time.

5.4 Experiments

A run is terminated and deemed a success if any individual from the current population matches the target function within 0.01 tolerance over a set of 50 out-of-sample test cases in the domain [-100,100]. Note that this domain is substantially wider than the "interesting" region of the curve. Results are averaged over 120 independent runs for each experimental condition. The locations of the out-of-sample points vary for each experiment. Note that this out-of-sample measure is used purely for analysis and does not affect evolutionary selection at all. If no solution is found after 100 generations, the run is terminated.

The performance of each method is measured by calculating "computational effort": we give the least number of fitness computations one would need to perform in order to have 99 percent probability of a successful run, given population size [13]. For some conditions we also give a histogram of the "sample density," the fraction of fitness cases that fall within each of 200 evenly spaced bins, from –100 to 100. The counts are taken over all generations of all 120 runs for the given condition.

Table 1. Computational effort for the co-evolutionary methods and the random method. Results are in total number of fitness competitions, also expressed (in parentheses) as a fraction of the number of fitness competitions required by the random technique.

Computational Effort (As Fraction of Random) for All Experiments			
	Tractable Shared	**Distinction**	**Random**
1. Gaussian	33,840,000 (0.06)	50,700,000 (0.08)	600,600,000 (1.00)
2. Piecewise-defined Linear and Cubic	334,800,000 (sol.)	100,440,000 (sol.)	no solution
3. Piecewise-defined Quadratics	21,120,000 (0.37)	47,040,000 (0.83)	56,550,000 (1.00)

5.4.1 *Gaussian*

Target Function:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

Additional Learner Primitives: We add terminals for *pi* and *e*.

Out-of-Sample Cases: Twenty-five out-of-sample test cases are spaced evenly over the domain [-100,100]. The other 25 cases are spaced evenly over the bell-shaped region of the curve [-3,3]. Note that f(x) = 0 fits the curve within 0.01 tolerance over [-100,-3.5)∪(3.5, 100], or about 97 percent of uniformly sampled values. Therefore we over-represent the set of out-of-sample cases in the interval [-3,3].

Results: The computational effort (CE) for the co-evolutionary methods is less than one-tenth the CE required by the random method; results for all experiments are summarized in Table 1. Whereas the random technique draws test cases uniformly from [-100,100], the co-evolutionary methods select points almost entirely in the "interesting" region of the curve, near the origin. Since, as noted above, 97 percent of uniformly sampled values are approximated closely by f(x) = 0, a uniform random sample presents the learners with an uninformative set of fitness cases.

5.4.2 *Piecewise-Defined Linear And Cubic*

Target Function:

$$f(x) = \begin{cases} x^3 - x^2 + 1 & \textit{on } [-1,1] \\ x & \textit{elsewhere} \end{cases}$$

Additional Learner Primitives: We add the following functions to facilitate generation of piecewise functions: `And`, `If`, `Or`, `GreaterThanOrEqual`, `LessThan`.

Out-of-Sample Cases: Twenty-five out-of-sample test cases are spaced evenly over the entire domain [-100,100]; the other 25 cases are spaced evenly over the cubic region [-1,1].

Results: The random method failed to find a solution during any of the 120 independent runs. The co-evolutionary methods found solutions, with the distinction method taking considerably less computational effort (Table 1). Both tractable shared and distinction methods devote a large portion of the sample to points in [-1,1]: 71 percent and 50 percent, respectively (not shown). The random method does poorly because it rarely samples this interesting region, which occupies only 0.01 of the curve and the same fraction of test cases.

5.4.3 *Piecewise-Defined Quadratics*

Target Function:

$$f(x) = \begin{cases} (x+50)^2 & \textit{on } (-\infty,0] \\ (x-50)^2 & \textit{elsewhere} \end{cases}$$

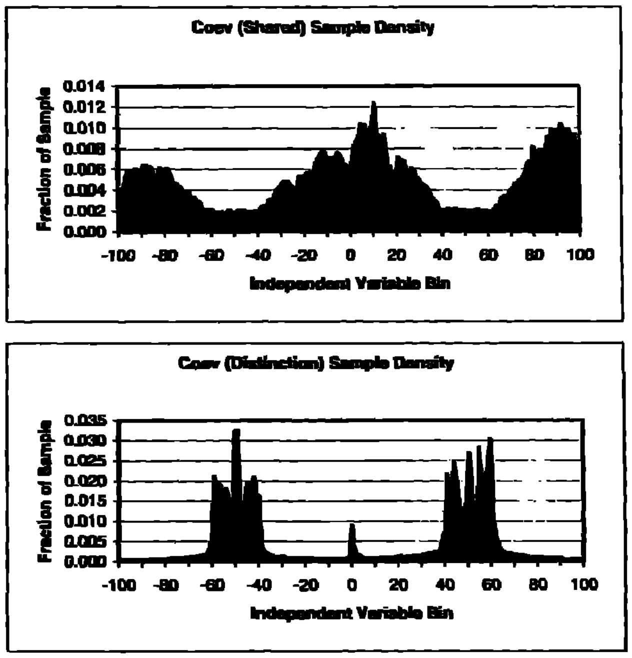

Figure 1: Sample density using the co-evolutionary tractable shared method (top) and distinction method (bottom) on the piecewise-defined quadratics problem.

Additional learner Primitives: We use all of the functions used in Experiment 2, and add the constant `50`.

Out-of-sample cases: The 50 out-of-sample test cases are spaced evenly over the domain [-100,100].

Results: The co-evolutionary tractable shared method outperforms the other methods. The distinction method has a slight edge over the random method, while the tractable shared method uses less than half the computational time required by the random method (Table 1). Both the tractable shared method and the distinction method devote an increased fraction of the sample to points near the origin (Figure 1). Whereas the tractable shared method emphasizes extreme values (top), the distinction method de-emphasizes these points in favor of points near the local minima (bottom).

6. APPLICATION TO SELF-RECONFIGURABLE MODULAR ROBOTICS

A self-reconfigurable modular robot is composed of many simple, identical modules, each of which has its own power, CPU, memory, motors, and sensors. A module on its own cannot move, but a group of modules, or robot, can reconfigure and locomote. All modules run the same control program, but take different actions based on internal state. See [5] for a detailed description of the modular robot we simulate here.

Our goal is to evolve distributed control software that enables a modular robot to move through a variety of worlds with passages. Each learner represents a program that runs on each module in the robot. Each problem encodes a simulated world with a passage. We evaluate the learner with the most hits on an out-of-sample set of 500 randomly generated problems.

6.1 Learners

The GP primitives used for this experiment are exactly as given in [5]. These include primitives for movement, communication, sensing, simple arithmetic, and flow control.

6.1.1 Competition

A competition occurs whenever a program is simulated in a test world. The outcome is given by

$$1-\frac{\textit{average distance of modules from topmost row}}{\textit{height of the world}}$$

at the end of the simulation.

6.1.2 Simulation

A simulation of the robot's actions in the world is run for a number of turns equal to twice the height of the world, or until the entire robot reaches the top of the world. In each turn, the evolved program tree is executed once for each module. The order of module execution remains fixed throughout the run.

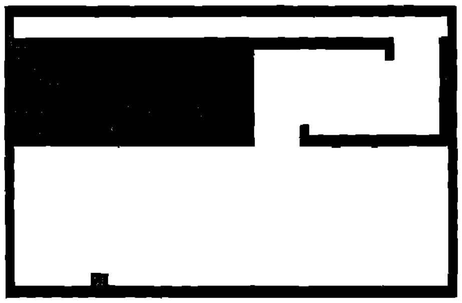

Figure 2. A typical world. The "crook" is the wall that extends horizontally into the passage, and the "kink" extends vertically into the passage. "Handedness" is 0 if the top crook is on the left, and 1 if it is on the right. The robot is in the bottom row of the world, to the left of center.

6.1.3 Evolution

The GP parameters are the same as those given for the symbolic regression problem, except: the population size is 1000, and we sample 10 individuals for problem testing.

6.2 Problems

The simulation worlds for the modular robot are rectangular, and bordered on all sides by impassable walls. The modular robot is composed of two modules, and its initial configuration is at a randomly chosen position on the bottom of the world. The initial facing direction of each module is east.

Each world optionally contains a horizontal wall, with a passage that may be straight, crooked, or crooked with a kink (Figure 2). Every such world can be completely specified by the ten parameters in Table 2.

6.2.1 Chromosome

Each world is specified by a linear chromosome of length 10; each locus specifies a raw value between 0 and 500 that is then scaled to become a valid value, given in Table 2, for the relevant world parameter. When the crooks extend across half the passage, the kink height is reduced to one less than half the height of the passage, in order to maintain an open path.

Table 2. Valid values for world parameters. There is an additional restriction on the height of the kink. The meaning of the variables in the right column are given in the leftmost column.

World Parameter	Min	Max
a := World width	*5*	*60*
b := World height	*10*	*30*
c := Passage left X	*1*	*a - 3*
d := Passage bottom Y	*5*	*b - 3*
e := Passage width	2	*a - 1 - c*
f := Passage height	*0*	*b - 1 - d*
g := Width of crook	*0*	*e - 1*
h := Height of kink	*0*	*f - 2*
i := Handedness	*0*	*1*
j := Initial robot X	*1*	*a - 4*

6.2.2 Problem Random Sampling Setup (Control Condition)

Five fitness cases are created at random, each generation, by choosing uniformly distributed random values for each allele.

6.2.3 Problem GA Setup (Experimental Conditions)

6.2.3.1 Competition

The result of a problem competition is given by

$$\frac{\textit{average distance of modules from topmost row}}{\textit{height of the world}}$$

after performing a simulation. The problem gets a hit if the competition result is less than one.

6.2.3.2 Fitness

We experiment with both tractable shared fitness and distinction fitness. With distinction fitness, a problem is said to distinguish between a pair of learners if one gets a hit and the other does not.

6.2.3.3 Evolution

Both Gaussian mutation and sparse crossover are used. The population of problems is sorted according to fitness. Then, with equal random chance, each problem either undergoes Gaussian mutation or is replaced with the result of a sparse crossover operation. Gaussian mutation is as described above, except operating on integer values. Sparse crossover first selects two parents (from the original population) using tournament selection with a tournament size of two. Then, with probability 0.10, each allele is swapped with the allele in the same position in the other parent. One individual is randomly designated the result and copied over the current individual in the population.

The population size is 100, and we sample 4 individuals for learner testing. Note that the number of fitness competitions per generation for the co-evolutionary methods (1000*4 + 100*10 = 5000) is equal to that of the random sampling method.

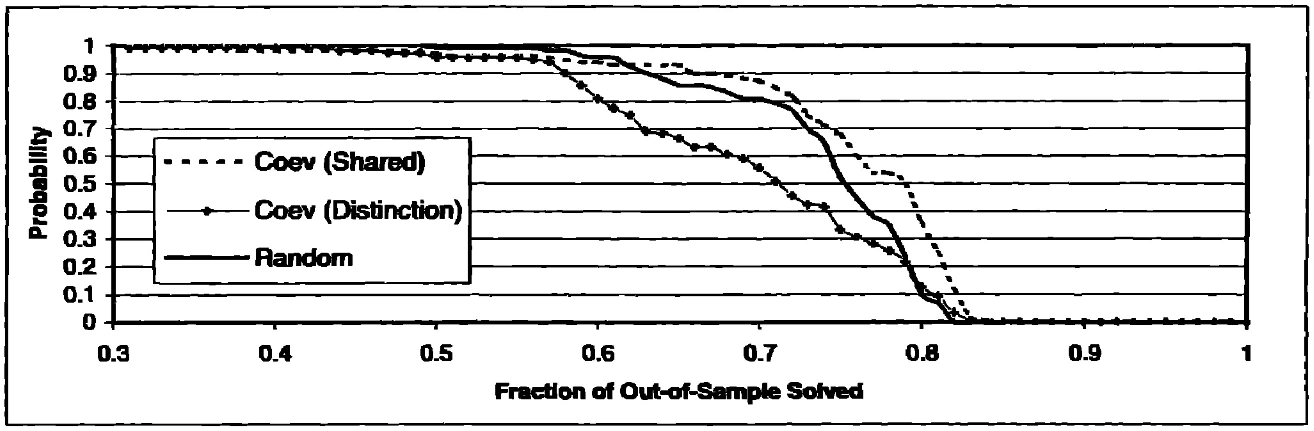

Figure 3. Fraction of out-of-sample test cases solved vs. probability of reaching that fraction in any of 100 generations, as estimated by 120 independent runs.

Table 3. CE required to evolve a program that solves the given fraction of the out-of-sample test set with 0.99 probability. The number in parenthesis expresses the CE as a fraction of the CE required by the random technique.

	Computational Effort (As Fraction of Random)		
Fraction of Out-of-Sample to Solve	**Tractable Shared**	**Distinction**	**Random**
0.60	850,000 (1.21)	1,470,000 (2.10)	700,000 (1.00)
0.70	1,275,000 (0.90)	2,970,000 (2.11)	1,410,000 (1.00)
0.80	5,390,000 (0.25)	17,325,000 (0.80)	21,780,000 (1.00)
0.82	18,810,000 (solved)	60,520,000 (solved)	no solution

6.3 EXPERIMENTAL RESULTS

Results are averaged over 120 independent runs of 100 generations. We record the out-of-sample score for the individual with the most hits at each generation; when there are several, we randomly choose one. Some fraction of the worlds (unknown, but less than 15 percent) have long narrow passages that are impossible for the small robot used here to traverse. As such, we are unsure whether any method produced a fully general solution.

The relative performance of the methods varies with the desired amount of generalization. For each condition, we report the estimated probability that the best-of-generation individual is able to solve at least some fraction *fs* of the sample, where *fs* is varied from 0 to 1. We plot this probability of generalization in Figure 3. The random method most reliably gives up to 0.62 generalization, though only by a narrow margin over the other methods.

For all higher levels of generality, the shared method is substantially more reliable than the random method. Up to the highest levels of generalization, however, the random method outperforms the distinction method; then, both co-evolutionary methods are most reliable. For example, the random method has only a 6.7 percent chance of evolving an individual capable of solving 81 percent of the out-of-sample set over 100 generations, while the tractable shared method has a 25 percent chance, and the distinction method has a 9.2 percent chance. Moreover, the random method never solved 82 percent of the out-of-sample test set, whereas the tractable shared method has an 11.7 percent chance, and the distinction method has a 3.3 percent chance.

In terms of CE, as Table 3 indicates, the random method has the edge when we are lax about the out-of-sample generalization we require. But when we are seeking a maximally robust solution, the co-evolutionary methods pay off. With the shared method, we can evolve a control program which solves 80 percent of the out-of-sample set using roughly a quarter of the computer time required by the random method.

7. CONCLUSIONS

The co-evolutionary techniques are better able to evolve robust solutions than the random technique. In fact the random method failed to find any solutions to Experiment 2, and failed to find modular robot software that generalized to 82 percent of the test cases. The co-evolutionary methods beat the random method even when, as in Experiment 3 and the robot problem, the random fitness cases were drawn from the same distribution as the out-of-sample test set.

The two co-evolutionary methods have widely differing levels of success depending on the problem, and, for the robot problem, depending on the desired level of generality. The two methods performed comparably in Experiment 1 with the shared method having a slight edge. In Experiment 2, the distinction method substantially outperformed the shared method, needing only a third of the computational effort. But the shared method needed only half of the computational effort in Experiment 3, and substantially outperformed the distinction method until the very highest levels of generality on the robot problem. More puzzling is that the distinction method performed markedly worse than the random method on the robot problem until the very highest levels of generality were reached.

In future work we hope to develop a better understanding of when to use each co-evolutionary method. We are also interested in developing a co-evolutionary method that combines the strengths of both methods presented here.

8. ACKNOWLEDGMENTS

The authors gratefully acknowledge Nils J. Nilsson for a continued dialogue about higher-level issues, and John R. Koza for his suggestions about obtaining empirical results. Many thanks to Adil Qureshi for his Gpsys 1.1 Java GP source code on which FXPAL's system is rooted.

9. REFERENCES

[1] Bennett, Forrest H, III, Rieffel, Eleanor G. "Design of Decentralized Controllers for Self-Reconfigurable Modular Robots Using Genetic Programming." Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware, 2000.

[2] Bennett, Forrest H, III, Dolin, Brad, and Rieffel, Eleanor G. "Programmable Smart Membranes: Using Genetic Programming to Evolve Scalable Distributed Controllers for a Novel Self-Reconfigurable Modular Robotic Application." Proceedings of the 4th European Conference (EuroGP), 2001.

[3] Bojinov, Hristo, Casal, Arancha, and Hogg, Tad. "Emergent Structures in Modular Self-reconfigurable Robots." IEEE Intl. Conf. on Robotics and Automation (ICRA), 2000.

[4] Bonabeau, Eric, Dorigo, Marco, and Theraulaz, Guy. Swarm Intelligence: from Natural to Artificial Systems. Oxford Univ. Press, 1999.

[5] Dolin, Brad, Bennett, Forrest H, III, Rieffel, Eleanor G. "Methods for Evolving Robust Distributed Robot Control Software: Coevolutionary and Single Population Techniques." Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware, 2001.

[6] Ficici, Sevan G., Pollack, Jordan B. "Pareto Optimality in Coevolutionary Learning." Computer Science Technical Report CS-01-216, 2001.

[7] Floreano, D., Mondada, F., and Nolfi, S. "Co-evolution and Ontogenetic Change in Competing Robots." Robotics and Autonomous Systems, 1999.

[8] Haith, Gary L., Colombano, Silvano P., Lohn, Jason D., and Stassinopoulos, Dimitris. "Coevolution for Problem Simplification." Proc. 1999 Genetic and Evolutionary Computation Conference, (GECCO-99), 1999.

[9] Haynes, Thomas, Wainwright, Roger, Sen, Sandip, and Schoenfeld, Dale. "Strongly Typed genetic Programming in Evolving Cooperation Strategies." Proceedings of the Sixth International Conference on Genetic Algorithms, 1995.

[10] Hillis, Daniel W. "Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure." Artificial Life II, Addison-Wesley, 1991.

[11] Juillé, Hugue. "Incremental Co-evolution of Organisms: A New Approach for Optimization and Discovery of Strategies." Proceedings of the Third European Conference on Artificial Life, 1995.

[12] Juillé, Hugue, Pollack, Jordan B. "Coevolving the 'Ideal' Trainer: Application to the Discovery of Cellular Automata Rules." Proceedings of the Third Annual Genetic Programming Conference, 1998.

[13] Koza, John R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, 1992.

[14] Lee, W., Hallam, J., and Lund, H. "A Hybrid GP/GA Approach for Coevolving Controllers and Robot Bodies to Achieve Fitness-Specified Tasks." Proceedings of IEEE 3rd International Conference on Evolutionary Computation, 1996.

[15] Luke, S., Hohn, C., Farris, J., Jackson, G., and Hendler, J. "Co-evolving Soccer Softbot Team Coordination with Genetic Programming." RoboCup-97: Robot Soccer World Cup I, 1998.

[16] McNutt, Greg. "Using Co-evolution to Produce Robust Robot Control." Genetic Algorithms and Genetic Programming at Stanford, (Stanford University Bookstore) 1997.

[17] Rosin, Christopher D., Belew, Richard K. "Methods for Competitive Co-evolution: Finding Opponents Worth Beating." Proceedings of the Sixth International Conference on Genetic Algorithms, 1995.

[18] Rosin, Christopher D. "Coevolutionary Search Among Adversaries." Ph.D. Thesis, University of California San Diego, 1997.

[19] Sipper, Moshe. "Co-evolving Non-Uniform Cellular Automata to Perform Computations." Physica D, 1996.

[20] Soule, T. "Using Genetic Algorithms to Evolve Cooperative Teams." Proceedings of the Genetic and Evolutionary Computation Conference, 2001

[21] Werfel, Justin, Mitchell, Melanie, and Crutchfield, James P. "Resource Sharing and Coevolution in Evolving Cellular Automata." IEEE Trans. Evol. Comp., 1999.

[22] Yim, Mark, Lamping, John, Mao, Eric, Chase, J. Geoffrey. "Rhombic Dodecahedron Shape for Self-Assembling Robots." Xerox PARC SPL TechReport P9710777, 1997.