Coevolution for Problem Simplification

Gary L. Haith*
Recom Technologies
Computational Sciences Division
NASA Ames Research Center

Silvano P. Colombano
Computational Sciences Division
NASA Ames Research Center

Jason D. Lohn
Caelum Research Corporation
Computational Sciences Division
NASA Ames Research Center

Dimitris Stassinopoulos
National Research Council
Computational Sciences Division
NASA Ames Research Center

Abstract

This paper explores a coevolutionary approach applicable to difficult problems with limited failure/success performance feedback. Like familiar "predator-prey" frameworks this algorithm evolves two populations of individuals – the solutions (predators) and the problems (prey). The approach extends previous work by rewarding only the problems that match their difficulty to the level of solution competence. In complex problem domains with limited feedback, this "tractability constraint" helps provide an adaptive fitness gradient that effectively differentiates the candidate solutions. The algorithm generates selective pressure toward the evolution of increasingly competent solutions by rewarding solution generality and uniqueness and problem tractability and difficulty. Relative (inverse-fitness) and absolute (static objective function) approaches to evaluating problem difficulty are explored and discussed. On a simple control task, this coevolutionary algorithm was found to have significant advantages over a genetic algorithm with either a static fitness function or a fitness function that changes on a hand-tuned schedule.

*Mail Stop 269-3, Mountain View, CA 94035-1000
e-mail: haith@ptolemy.arc.nasa.gov

1 Theoretical Background

Traditional evolutionary algorithms evaluate the fitness of an individual by evaluating its ability to minimize an objective function which is typically static and independent of the evolutionary algorithm. For example, if the goal is to evolve a posture controller for a robot, the fitness of an individual controller could be its success in minimizing movement in the robot body under a gravity load. In coevolutionary algorithms, the fitness of an individual in the evolving population(s) depends on interactions with other individuals in the same generation. The problems (e.g. the forces on the robot) faced by individuals in a coevolutionary algorithm are dynamic and are shaped by the algorithm itself. Extending the robot example, the situations that a robot faces (e.g. forces on the robot like gravity) could be coevolving with the controllers such that the set of situations on which controllers are evaluated changes from generation to generation.

1.1 Coevolution: Competition and Cooperation

A growing body of research explores coevolutionary approaches that capitalize on this dynamic quality (for review, see Paredis, 1998) . This coevolutionary work has largely concentrated on competitive interactions. The interactions can be between individuals that compete in a symmetric game-like context (Pollack *et al.*, 1996; Sims, 1994; Rosin, 1997) or between populations of different types of individuals that compete in predator/prey type relationships (Hillis, 1991; Paredis, 1994b; Paredis, 1994a; Cliff & Miller, 1996; Juille &

Pollack, 1998; Rosin, 1997; Rosin & Belew, 1996). In these cases, individuals are rewarded if they defeat the individuals with which they compete. These interactions can support "arms-races" in which the individuals force each other to become increasingly competent.

A few studies have investigated the role of cooperation and how it can help solve some problems endemic to evolutionary methods, like the difficulty of choosing an appropriate encoding for the individuals (Paredis, 1995) and the difficulty of decomposing composite problems (Jong & Potter, 1995). Other studies have found that a balance of cooperation and competition is necessary to prevent evolutionary algorithms from getting trapped in local minima, or "Mediocre Stable States" (Ficici, 1995).

1.2 The Current Approach

The approach outlined in this paper has features of both competitive and cooperative coevolutionary approaches. The algorithm tries to ensure a tractable learning gradient for the solutions by rewarding only those problems on which at least one solution was successful. The fitness of these tractable problems is proportional to their absolute and/or relative difficulty — providing pressure for the solutions to become more generally competent. In practice, this requirement generates an initial simplification and gradual increase in problem difficulty over evolution. The aim is to select for problems that are on the edge of what is solvable by the current population of solutions, ensuring a useful fitness gradient throughout evolution.

This requirement that the problem must be tractable has been relatively unstressed in the literature, with a couple notable exceptions. Rosin (1997) suggests a mechanism (the "Phantom Parasite") that rewards problems that are solvable by at least one solution. This mechanism will tend to allow easy problems to survive in a population of very difficult problems. Juille and Pollack (1998) use a domain specific approach to selecting for problem tractability by rewarding problems that tend to be easier by an objective measure.

Dealing with problem tractability is not an issue in problem domains where the problems provide partial fitness measures (Hillis, 1991; Ficici, 1995) or have a baseline success rate that is fairly high, akin to a multiple choice problem (Juille & Pollack, 1998; Paredis, 1994b). In these cases, there is always a fitness gradient for the solutions to follow in the form of the number of problems solved. However, in many real problem domains the performance of a set of randomly chosen solutions on a randomly chosen problem would be so low that an observer or fitness function would be unable to differentiate between the performance of the candidate solutions.

1.3 Difficult Tasks

Many problems require a surprisingly high level of expertise to even be approached. Faced with such a problem a naive learner must be given some bias, or a structured learning environment (termed a "gradient engineered fitness landscape" by Ficci and Pollack, 1998) to have a hope of mastering the task. In developmental terms, the current task must be kept in the "Zone of Proximal Development" (Vygotsky, 1986), or ZPD, in order to be tractable and useful to the learner. If the problem is outside the ZPD, then the learner will be unable to gain competence through experience with the task. In evolutionary terms, a fitness function that is too far beyond the competency of the individuals will fail to usefully differentiate between the individuals, and evolution will be unable to select for competency.

The challenge of staying within the ZPD is especially relevant in difficult reinforcement learning problems. In these problems: there is an absolute measure of performance (as opposed to a game with relative performance), the measure of performance is mainly limited to success/failure, and the baseline probability of success for a solution given a typical problem is very low. For example, the control or design of a complex structure like an automobile engine depends on many pieces coming together in just the right way before any success at all is achieved. This seemingly impossible design task has only been tractable because the task itself has evolved over history. Originally the task was simply to translate heat into rotational energy. Details that are crucial to current engines like gearing, internal combustion, carburation, etc. were only added as each progressively more complex design was realized. In this paper we explore some mechanisms that could help make complex problems tractable to evolutionary algorithms by providing a gradient of problem difficulty/complexity over evolution.

2 Problem/Control Framework: 2D Free-Space Vehicle

This work uses a relatively simple simulation framework that allows for quick exploration of coevolutionary mechanisms. The problem is to control the thrusters on a craft floating in free space such that the craft goes to a given point (the "origin") and comes to rest within a given period of time (1 sec, 5 time-steps). The movement of the craft is limited to 2 dimensions, and is simulated approximately using discreet timesteps. At the end of the time period, a solution "succeeds" if the craft is resting (within some error) at the origin at the end of the time period — otherwise it "fails". This method of evaluation converts the available continuous error signals to a reinforcement learning signal.

Problem difficulty can be easily parameterized in this framework. An optimal solution in this framework would be able to steer the craft toward the origin from any position and initial velocity and would learn to stop at the origin within the time period. Because solution performance is evaluated over a limited period of time, a large initial distance and velocity require the solution to generate strong and accurate thruster firing. In contrast, small initial distances and velocities can be successfully navigated with weak and relatively inaccurate thruster firing (see Sec. 5.1 for limitations of this interpretation). In general, problem difficulty is proportional to the craft's initial distance (D_p) from the origin and initial velocity (D_v).

Candidate solutions in this simulation are simple linear networks where the change in the XY thrust at each the next time step is a weighted sum of the current XY thrust, velocity, and position.[1] A candidate solution is a set of weights for this network.

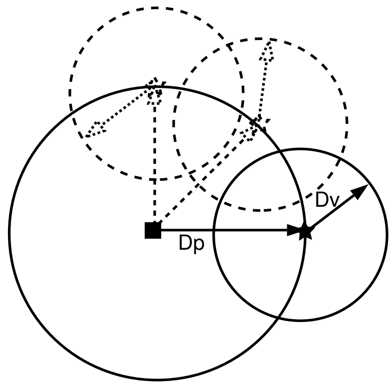

Figure 1: The figure represents the possible initialization conditions represented by a given problem (D_p, D_v). The square at the center of the large circle is the origin. D_p is the difficulty/distance of the initial position, D_v is the difficulty/magnitude of the initial velocity. The actual initial configuration, a starting position in the case of D_p (the star) and a velocity vector in the case of D_v, is chosen randomly from the set of points on the circles. The dotted circles represent other possible starting positions (dotted stars) with their associated sets of possible velocity vectors.

3 Evolutionary Framework: Coevolutionary GA

Each problem is described by 2 scalars (see Fig. 1): initial distance from the origin (D_p), and initial velocity (D_v). The actual position and velocity of each problem in each generation is chosen randomly from the points on the circles described by the two problem scalars, thus at each generation a problem describes an initial XY position and velocity. In this way the difficulty of the problems (the magnitude of the problem scalars) can be preserved or changed from generation to generation, while the specific problems are randomly sampled each generation.

Each generation every solution is evaluated on every problem. The weights of the solutions/networks are evolved using a genetic algorithm.[2] In simulations, an initial population (N = 50) of solutions is chosen at random with relatively small weights ([-.05, .05]). These solutions are then evaluated on the set of problems (N = 50) present that generation. The genomes of the solutions are lists of the 12 floating point weights in the network. Each generation new candidate solutions are generated by probabilistically choosing parents (based on their sigma-scaled fitness, Mitchell, 1996), re-combining them in pairs via 2 point cross-over, and with some probability (%10 mutation rate) mutating each weight of the new solutions by adding a random number (selected from [-1,1]). The best 5% of the solutions at each generation are replicated exactly in the following generation.

The focus of this paper is on different methods of choosing the evaluation problems (initial conditions). This work compares three methods of generating problem difficulties (D_p's and D_v's) for the sample problems at each generation: the standard evolutionary approach, the gradient/developmental approach, and the coevolutionary approach (with 2 particular instantiations). Note that in all methods the specific problems (starting position and velocity) were randomly generated by selecting the starting point and velocity vector from the circles described by D_p and D_v. Even if the problem difficulties were identical across generations, the specific problems would be different.

[1] Some simulations were run using a feed-forward neural network architecture (2 layer, 2 hidden units with a hyperbolic transfer function). These simulations yielded qualitatively similar results.

[2] Although in this case the control networks could be trained using backpropogation or a similar neural network training algorithm, a genetic algorithm was used to find effective weights so as to explore coevolutionary mechanisms.

3.1 Standard Evolutionary Approach

The first method is to randomly select each (D_p, D_v) from a uniform distribution across [0,D_m], where D_m is the maximum problem difficulty (typically 50). In this method the average problem difficulty is constant at $\frac{D_m}{2}$ (see Fig. 2, heavy line). This first method is meant to reflect the most common/standard practice where throughout evolution the solutions are evaluated on the full set of possible problems, or a fully complex target problem.

3.2 Gradient/Developmental Approach

The second method is inspired by the developmental considerations discussed above. This method presents an increasingly difficult set of problems to the population of solutions. The difficulties (D_p, D_v) at each generation are chosen from a uniform distribution across [0,$D_m \times G(t)$], where $G(t)$ is a monotonically increasing function of generation number (t). Typically $G(t)$ is a simple linear increase from 0 at generation 1 to D_m at the last generation (see Fig. 2, medium lines). In this method the average problem difficulty increases monotonically over training. This second method reflects the developmental theory (Elman, 1991; Newport, 1988) and intuitive heuristic, that hard problems are easier to learn if problem complexity starts off low and increases gradually over training as the competency of the solution improves.

3.3 Coevolutionary Approach

The third method coevolves the difficulties of the evaluation problems and the the weights of the candidate solutions. Like the solutions, problem difficulties are evolved with selection, cross-over and mutation. The average problem difficulty is under the control of the evolutionary algorithm in this method. A central focus of this paper is to determine if this coevolutionary algorithm can discover and optimize the hand-coded gradient/developmental method described in the previous section (for actual behavior see Fig. 2, fine lines).

We explored two methods of evaluating the raw fitnesses of the solutions and problems: absolute and relative. In the absolute method, the raw fitness of a solution (Fp) is the sum of the difficulties of the problems that it completed successfully, where N is the number of problems, and S^i_j is 1 if problem i is successfully solved by solution j and 0 otherwise (see Eq. 1).

$$Fp_j = \frac{1}{2}\sum_{i=1}^{N}(D^i_v + D^i_p) \times S^i_j \tag{1}$$

The absolute raw fitness of a problem is its difficulty ($\frac{1}{2}(D_v + D_p)$) if it was completed successfully by at least one solution and 0 otherwise, satisfying the tractability constraint.

The relative method is similar to the "inverse-fitness", or "competitive fitness sharing" method used by previous researchers (Paredis, 1998; Juille & Pollack, 1998; Rosin, 1997; Rosin & Belew, 1996). Fitness of the solutions is proportional to the number of problems that they successfully solve, with the reward for each problem being inversely proportional to the number of solutions that solved it (see Eq. 2) — a rough measure of how "easy" it is.

$$Fp_j = \sum_{i=1}^{N} \frac{S^i_j}{\sum_{j=1}^{N} S^i_j} \tag{2}$$

The fitness of each problem is inversely proportional to the number of solutions that complete it successfully, with a tractability constraint. A problem not successfully completed by any solution gets zero fitness (instead of the maximum fitness in the traditional "inverse-fitness" approach).

$$Fs_i = \frac{T^i}{\sum_{j=1}^{N} S^i_j} \tag{3}$$

Here T^i (the tractability of problem i) is 1 if any of the solutions successfully completed problem i and 0 otherwise.

4 Results

Two measures are displayed for each of the 3 evolutionary methods. Displayed results are the average of 10 runs in each method, with the same parameters in all runs.[3] The first reports the mean difficulty of the problems ($\frac{1}{2N}\sum_{i=0}^{N}(D^i_v + D^i_p)$). The second is a measure of the performance of the most fit solution. In order to get a standardized measure of the solution performance, the solution with the highest fitness in each generation was evaluated on a standard set of

[3]Simulation Parameters: 50 problems, 50 solutions, 2 seconds of controller time, time step of .2 sec, 250 generations total, linear solution networks, .05 elitism, .1 mutation rate, and mutation step size is randomly drawn from $[-1, 1]$.

625 initial conditions selected so as to sample a regular grid of initial positions and velocities.

$$P_j = 100 \times (1 - \frac{\sum_{i=1}^{T} (D_{pj}^i + D_{vj}^i)}{\sum_{i=1}^{T} (D_{p0}^i + D_{v0}^i)}) \qquad (4)$$

The performance of the highest fitness network (P_j) was evaluated by summing the errors in the final position (D_{pj}) and velocity (D_{vj}) reached from the test set of initial conditions (indexed by i, T total) with thrusters controlled by the highest fitness network (network j). This sum was then compared to the final errors in position and velocity reached with no thrusters firing D_{p0} and D_{v0}, and the proportion was normalized such that perfect performance would correspond to a performance score of 100 (see Eq. 4). It should be noted that the performance score is negative if the given solution is worse (i.e. results in larger D_{pj}'s and D_{vj}'s) than the 0 thrust case. In fact, a negative performance score is overwhelmingly likely given a randomly generated solution (only 11/1000 randomly generated solutions had a positive performance score, and the average performance was -5000).

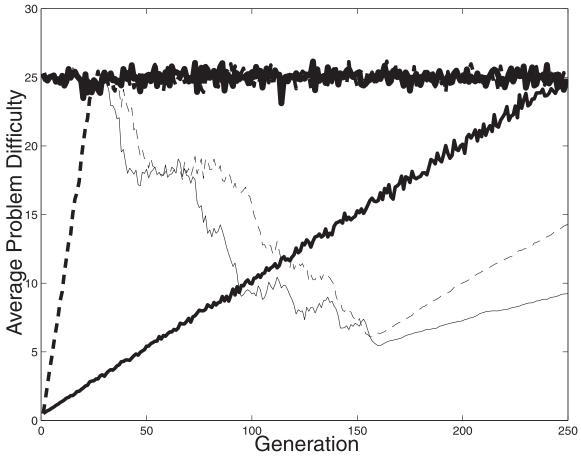

Figure 2: Each line shows the problem difficulty at each generation for a given approach. Each line is averaged over all the problems in that generation and over 10 runs. 50 is the maximum initial difficulty in the coevolutionary runs, final difficulty in the gradient runs, and maximum difficulty throughout the standard run. The heavy line is the standard approach. The fine lines are coevolutionary approaches with problem difficulty evaluated absolutely (dotted) and relatively (solid). The medium lines are hand-tuned gradient approaches with a fast rise in task difficulty (dotted) and a slow rise in task difficulty (solid). See text for details.

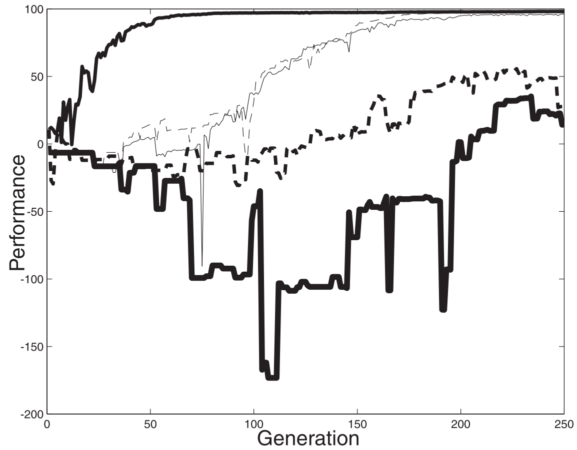

Figure 3: Each line shows the performance of the best solution at each generation for a given approach. Each line is the average of 10 runs, and 100 is the maximum performance. The solid heavy line is the standard approach. The fine lines are coevolutionary approaches with problem difficulty evaluated absolutely (dotted) and relatively (solid). The medium lines are hand-tuned gradient approaches with a fast rise in task difficulty (dotted) and a slow rise in task difficulty (solid) — see Fig. 2. The main parameters were held constant in all runs. See text for details.

4.1 The Standard Approach

In some parameter regimes, the standard case (selecting D_p and D_v from a uniform distribution across [0,D_m] throughout evolution) generally failed to find a generally successful solution (See Fig. 3, heavy line) over the course of evolution. The negative performance of the solutions is probably due to "fortuitous" initialization/solution matches, where the solution is unable to generalize its successful performance to the test set of initializations/problems. For example, a solution that continually fires the left thruster might be successful in a generation where one of the initial positions is just off to the left, but it (and its offspring) will be unable to generalize that success to another random sample of problems. In these 10 runs the standard algorithm came up with a relatively poor solution with an average performance of 15.

4.2 The Gradient/Developmental Approach

The evolution of competent solutions is made much more robust by gradually increasing the average problem difficulty over evolution (see Fig. 3, solid medium line). The general success of this approach is due to the fact that it can ensure that the problems are always simple enough for some of the solutions to solve, enabling evolution to get a foothold in differentiating solution fitness based on performance. Only solutions that have been selected for many generations face difficult problems late in a given evolutionary run.

This approach has the shortcoming that the rate of problem difficulty increase must be tuned to the given problem and rate of competency growth in the solutions. If the difficulty of the problems is increased too quickly, then the success of some solutions is not overwhelmingly likely and, as in the standard case, the run may fail to find a generally successful solution (see Fig. 3, dotted medium line). In the case of a too-steep gradient, the gradient approach yielded a final controller with a fitness of only 25. Generally speaking, if the difficulty of the problem is increased too slowly, then little evolutionary pressure is put on the solutions to have general competency and suboptimal solutions will result (but see Sec. 5.1 for discussion of this problem as an exception).

4.3 The Coevolutionary Approach

The coevolutionary method retains advantages of the gradient/developmental approach, but avoids the necessity of selecting the schedule of increasing problem difficulty at an arbitrary rate (See Fig. 3, fine lines). The coevolutionary approach has the advantage of automatically adjusting problem difficulty to match solution competence (see Fig. 4). Even though the average problem difficulty starts off large, easy problems have much higher fitness early in evolution because they are the only problems that can be successfully completed by relatively incompetent solutions. Easy problems tend to take over the population of problems just after a tractable problem is found, (see Fig. 4) while the solutions are still relatively incompetent. Problems tend to get harder over evolution because they are rewarded for being solvable only by a few solutions (relative) or for being more difficult by some absolute measure (absolute). Any problem that increases in difficulty too quickly will be penalized because it will not be successfully completed by any of the solutions.

5 Discussion

This paper presents an approach to using coevolution to simplify complex problems. By rewarding a coevolved population of problems for being at the edge of what is currently solvable by the population of solutions, the method generates a usable fitness gradient for the solutions while encouraging general solution competency at difficult problems. This approach takes some small steps toward making coevolutionary algorithms more applicable to a difficult and important class of problems. The results that were presented demonstrate that, in some domains, the approach can be more effective than a traditional evolutionary approach and more flexible than a hand-coded approach.

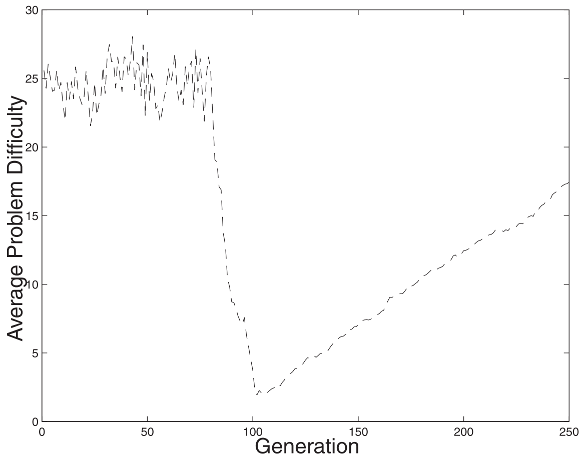

Figure 4: The average difficulty of the population of problems during a representative run (taken from the 10 averaged runs) using coevolution with selection for absolute problem difficulty. Note the random search, followed by problem simplification and a gradual increase in problem difficulty. See text for details.

5.1 Limitations

This work has some limitations that are important for proper interpretation.

First, initial success in the coevolutionary and standard approaches is simply probabilistic. Even with the tuned parameters in the simulations reported above, several generations often pass without any successful solution/problem pairings. Indeed this difficulty was explicitly chosen, because if the problem is made too simple (e.g. by increasing the error threshold for successful performance) than there is a sufficient fitness gradient for the standard approach to perform as well as the coevolutionary approach. During these unsuccessful generations, the algorithm does a random search for solvable problems, and all but the elite solutions undergo random evolution or genetic drift. In a very difficult problem domain, randomly generated solutions will almost never successfully solve randomly generated tasks. This issue could be addressed by seeding the initial population with simple problems that are thought to be applicable to the fully complex problem, thus ensuring some success in even a random population of solutions.

Second, the algorithm here involves problem simplification instead of problem decomposition. In the case of simplification it is straightforward to generate estimates of problem difficulty or problem match to a target objective function, therefore is it easy to evaluate problems on their absolute difficulty. In problems that are compositional, hierarchical, or otherwise complex this assignment of absolute difficulty is not as straightforward. Unfortunately, it is also hard to get a useful measure of the intrinsic difficulty in complex problems. The issue is that an evolving problem must be difficult in the same way as the ultimate target problem, and usually there are many other ways to be difficult. The challenge is to find a problem representation that allows simple evaluation of the similarity or applicability of candidate problems to the target problem. Such a representation allows an absolute difficulty measure to help guide the explorations generated by the intrinsic difficulty measure.

Third, the fact that this domain provides only for problem simplification ensures that solutions that succeed at simple problems will tend to succeed at hard problems as well. The most vivid illustration of this fact is that runs with the gradient/developmental approach and with a very low maximum problem difficulty (D_m) evolve a solution with competence nearly matching a gradient/developmental approach with a relatively high D_m. The result is that there is relatively little intrinsic pressure for the problems to become more difficult. This fact limits the usefulness of this problem domain for study of these coevolutionary mechanisms.

5.2 Future work

We plan to test this coevolutionary approach in problem domains that avoid the limitations mentioned above. One candidate domain is coevolving analog filters and their target frequency response. Previous work on the evolution of simple analog filters has found the efficiency of evolutionary search to be highly dependent on a proper choice of fitness functions (Lohn & Colombano, 1998). In addition, somewhat complex filters, like passive cross-over filters are relatively difficult to design and optimize by hand. This well-explored domain should allow us to test the ability of coevolution to provide a usable gradient through simplification and decomposition.

A second candidate domain coevolving a gait controller for a walking robot. Previous work has found that decomposing a locomotion problem into behaviors provides a many fold speed-up in controller evolution (Gruau, 1996). The chore of deciding how to usefully decompose a robotic control task is generally not straightforward and thus far has depended on the insights and patience of a human programmer. We plan to use a coevolutionary approach to evolve a controller for a semi-rigid walking robot under current development at NASA Ames Research Center.

Acknowledgments

References

Cliff, D., & Miller, G. F. 1996. Co-evolution of pursuit and evasion ii: Simulation methods and results. *Pages 506–515 of:* Maes, P., Mataric, M., Meyer, J. A., Pollack, J., & Wilson, S. (eds), *From animals to animats 4: Proceedings of the fourth international conference on simulation of adaptive behavior (sab96).* MIT Press Bradford Books.

Elman, Jeffrey L. 1991. *Incremental learning, or the importance of starting small.* Tech. rept. 9101. Center for Research in Language, University of California, San Diego, CA.

Ficici, Sevan G. 1995. Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states.

Gruau, F. 1996. *Cellular encoding for interactive robotics.* Tech. rept. Sussex University.

Hillis, Daniel W. 1991. Co-evolving parasites improve simulated evolution as an optimization procedure. *Pages 313–324 of:* Langton, C., Taylor, C., Farmer, J. D., & Rasmussen, S. (eds), *Artificial life 2*, vol. X. Redwood City, CA: Addison-Wesley.

Jong, Kenneth A. De, & Potter, Mitchell A. 1995. Evolving complex structures via cooperative coevolution. *Pages 307–317 of: Proceedings of the fourth annual conference on evolutionary programming.* MIT Press.

Juille, Hugues, & Pollack, Jordan B. 1998 (July 22-25). Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules. *In: Proceedings of the third annual genetic programming conference (gp-98).*

Lohn, Jason D., & Colombano, Silvano P. 1998. Automated analog circuit synthesis using a linear representation. *Pages 125–133 of: Proceedings of the second international conference on evolvable systems: From biology to hardware.* Berlin: Springer-Verlag.

Mitchell, Melanie. 1996. *An introduction to genetic algorithms.* Cambridge, MA: MIT Press.

Newport, Elissa L. 1988. Constraints on learning and their role in language acquisition: Studies of the acquisition of american sign language. *Language sciences*, **10, Number 1**, 147–172.

Paredis, Jan. 1994a. Coevolutionary constraint satisfaction. *Pages 46–55 of: Proceedings of the third international conference on parallel problem solving from nature*, vol. 866. Springer-Verlag.

Paredis, Jan. 1994b. Steps towards co-evolutionary classification neural networks. *Pages 102–108 of:* Brooks, R., & Maes, P. (eds), *Artificial lie iv.* Cambridge, MA: MIT Press.

Paredis, Jan. 1995. The symbiotic evolution of solutions and their representations. *Pages 359–365 of:* Eshelman, L. (ed), *Proceedings of the sixth international conference on genetic algorithms.* San Mateo, CA: Morgan Kaufmann.

Paredis, Jan. 1998. *The handbook of evolutionary computation.* Oxford University Press. Chap. Coevolutionary Algorithms.

Pollack, J., Blair, A., & Land, M. 1996. Coevolution of a backgammon player. *In:* Langton, C. (ed), *Proceedings artificial life 5.* MIT Press.

Rosin, Christopher D. 1997. *Coevolutionary search among adversaries.* Ph.D. thesis, University of California, San Diego.

Rosin, Christopher D., & Belew, Richard K. 1996. *New methods for competitive coevolution.* Tech. rept. CS96-491. Department of Computer Science and Engineering, University of California, San Diego.

Sims, Karl. 1994. Evolving 3d morphology and behavior by competition. *Pages 28–39 of:* Brooks, R., & P.Maes (eds), *Artificial life 4 proceedings.* MIT Press.

Vygotsky, Lev Semonovich. 1986. *Thought and language.* Cambridge, Mass.: MIT Press.