Adaptation in constant utility non-stationary environments

Michael L. Littman & David H. Ackley
Cognitive Science Research Group
Bell Communications Research
Morristown, NJ 07960

December 20, 1995

Abstract

Environments that vary over time present a fundamental problem to adaptive systems. Although in the worst case there is no hope of effective adaptation, some forms environmental variability do provide adaptive opportunities. We consider a broad class of non-stationary environments, those which combine a variable *result function* with an invariant *utility function*, and demonstrate via simulation that an adaptive strategy employing both evolution and learning can tolerate a much higher rate of environmental variation than an evolution-only strategy. We suggest that in many cases where stability has previously been assumed, the constant utility non-stationary environment may in fact be a more powerful viewpoint.

1 Non-stationary environments

An adaptive system within an environment performs two basic tasks. First, there is the search for, and the representation of, regularities in the history of interactions with the environment. Second, there is the attempt to gain some advantage from the constructed representation, by basing future actions on the assumption that those regularities will persist. If that fundamental adaptive assumption fails utterly, and the environment totally lacks such regularities, adaptation can provide no benefit.

Of course, not all non-stationary environments are so pathological. Even if the environment, when viewed as a monolithic function, may be observed to change over time in unpredictable ways, it may be possible to view that environment as being formed from interacting components some of which *do* possess persistent regularities. If that is possible, an adaptive system possessing such a view can gain advantage — even though the environment will always have some surprises in store — by finding, representing, and exploiting those component regularities.

1.1 Constant utility

In this paper we demonstrate one way this can occur within the context of evolutionary adaptation via the genetic algorithm [11, 8]. Consider this scenario in which an organism is faced with a series of interactions with an environment: In each interaction, the environment presents the organism with a *situation*, and then the organism responds with an *action*, and then the environment responds with a *result*. A *utility* function, hidden from the organism, reduces the result to a scalar value. The average utility of the results created by the organism over some given lifespan determines the organism's *fitness*. The nasty trick is this: At unpredictable times and in unpredictable ways, the environmental mapping from situation and action to result changes. As a consequence, if averaged over many environmental changes, the fitness produced by any single mapping will be at the chance level. The adaptive challenge is to produce organisms that yield high *online fitness* (i.e., fitness averaged over successive generations of organisms, see [6]), in the face of the changing environment.

Although this scenario is familiar enough to genetic algorithm researchers in some ways, it also has certain unusual aspects that are central to the present endeavor. To begin with, the environment is non-stationary: The fitness of a given situation-action map changes over time. Also, the fitness function can readily be decomposed into two sub-functions: One function mapping from situation and action to result, and another function mapping from result to utility. And finally, although the result function varies over time, the utility function does *not* change.

We call environments that possess these properties *constant utility non-stationary environments*. Although adaptation may not be beneficial in worst-case non-stationary environments, we can exploit the persistent regularity of the utility function to build an effective adaptation algorithm for this special class.

1.2 Evolutionary reinforcement learning

If an oracle were available to supply the organism with the utility of each result as it occurred, all that would be necessary is a learning algorithm to optimize the utility function during the lifetime of the organism. It could experiment with various action functions and see which lead to increased utility. Changes in the environmental result function would just mean some more learning to do. If such an oracle is unavailable, learning runs into trouble. In our scenario, the only information available is the lifetime average value of the utility function (i.e., the fitness), and that information is not available until end of the organism's lifespan, rendering moot the possibility of using it directly for learning.

The essential idea of *evolutionary reinforcement learning* (ERL) [3, 1] is that an evolutionary algorithm can supply learning with such an oracle. We can include in the genetic material a representation of a *goodness function*, and let learning proceed during an organism's lifetime under the *assumption* that the goodness function is an accurate representation of the utility function. When that goodness function is a poor predictor of utility, well then, learning won't help and the organism possessing it will have low fitness, but that's no worse than possessing a poor action function in an evolution-only algorithm. On the other hand, if the goodness function is a reasonable predictor of utility, then learning can be helpful and higher fitness can be the result, even when the environmental result function is changing.

By separating the stationary and varying portions of the environment, and genetically representing a predictor of the stationary part, and employing learning to compensate for the variable part, we can, at least in principle, and within limits imposed by the response time of the learning system, *stabilize* a non-stationary environment.

2 A simulation study

To demonstrate this effect, we require an abstract computational model of an organism facing a constant utility non-stationary environment, and an adaptive system employing both evolution and learning. Here, we employ very simple models in both of these roles. This choice helps isolate the phenomena of interest, and also reduces simulation costs, but it also rules out myriad effects — many probably harmful, some possibly useful — that could occur in more complex systems.

2.1 Notation

Let $T = 1 \ldots N$ index generations, and let $t = 1 \ldots n$ index interactions with the environment during one lifespan. Let $E = (r_t^T, U)$ denote an environment, consisting of a time-varying result function r and a constant utility function U. Let $S = (\Omega_T, G, L)$ denote an adaptive system, consisting of a sequence of populations $\Omega_T = \{x_p : 1 \leq p \leq P\}$ of binary strings $x = (0,1)^l$, a genetic algorithm G, and a learning algorithm L.

A string x yields an organism according to a constant genetic expression function $e(x) = o$. Each organism $o = (a_0, \{a_t\}, g)$ possesses an initial action function a_0, a goodness function g, and a set of action functions $a_1 \ldots a_n$ specified by the learning algorithm. During a run of the model, a set of PN organisms $\{o_p^T\}$ will be produced, performing a total of $I = PNn$ interactions with the environment.

The interaction of organism o_p^T with the environment at time t takes the following form: A uniform random binary situation vector v_s of length w — the "width" of the environment — is presented to the organism, which produces an action vector $v_a = a_{t-1}(v_s)$. The environment then produces a result vector $v_r = r_t^T(v_s, v_a)$. The organism then performs learning $L(a_{t-1}, g, v_s, v_a, v_r) = a_t$. The scalar-valued utility of the result is then determined $u_{pt}^T = U(v_r)$.

At each moment (T, t) in time, p interactions occur, one for each member of the population at time T. With some probability α — the environmental variation rate parameter — an alteration is made to r_t^T to produce r_{t+1}^T, otherwise $r_{t+1}^T = r_t^T$. To begin a new generation, $r_1^{T+1} = r_{n+1}^T$.

At the end of the lifespan of o_p^T, the fitness of the corresponding genetic code is evaluated

$$F(x_p^T) = \frac{1}{n} \sum_{t=1}^{n} u_{pt}^T.$$

After P lifespans, the genetic algorithm updates the population $\Omega_{T+1} = G(\Omega_T, \{F(x_p^T)\})$. The online system fitness

$$F_S = \frac{1}{NP} \sum_{T=1}^{N} \sum_{p=1}^{P} F_p^T.$$

is the quantity to be maximized by adaptation.

At birth: Extract a_0 from x, giving weights w_{sa} and biases w_{0a}. Extract v_g from x. Let $u_* = 0$.

At time $1 \le t \le n$:

1. *(Situation)* Pick random v_s; compute output probabilities $v_o(i) = 1/\left(1 + e^{-(v_s(i)w_{ii} - w_{0i})}\right)$.
2. *(Action)* Given a uniform random variable $\xi \in [0,1]$, $v_a(i) = 1$ if $v_o(i) \ge \xi$ and 0 otherwise.
3. *(Result)* Compute result vector $v_r(i) = r_i^{(R(2i-1),R(2i))}(v_s(i), v_a(i))$. If $v_g = v_r$ go to 4, else go to 5.
4. *(Reward)* Update weights and biases: $\Delta w_{ji} = 10(v_o(i) - v_a(j))\max(0.1, v_s(j)(1 - v_s(j)))$. Go to 6.
5. *(Punish)* Update weights and biases: $\Delta w_{ji} = 0.5(1 - v_o(i) - v_a(j))\max(0.1, v_s(j)(1 - v_s(j)))$.
6. *(Utility)* Increment u_* by $U(v_r, v_d)$.

At death: Exit returning fitness score u_*/n.

Figure 1: The lifetime of an organism.

2.2 A bit-wise independent environment

Our test environment presents situations involving $w = 8$ bits, requiring 8 bit actions, and producing 8 bit results. There is a one-to-one correspondence between situation, action, and result bits, so that each bit of the result vector depends only on the corresponding situation and action bits. The adaptation problem is drastically simplified by building this "spatial" independence into the genetic representation of the action function, which specifies eight parallel functions each mapping one situation bit to the corresponding action bit.

The result function r_t^T consists of eight independent boolean functions each of which maps one bit pair $(v_s(i), v_a(i))$ to one result bit $v_r(i)$. The function is represented by a 16 bit vector R, interpreted as eight two bit values. Each value selects one of four boolean functions:

$$r_i^{(0,0)} = v_a(i), \quad r_i^{(0,1)} = \neg v_a(i),$$

$$r_i^{(1,0)} = v_s(i) \,\mathrm{eqv}\, v_a(i), \quad r_i^{(1,1)} = v_s(i) \,\mathrm{xor}\, v_a(i).$$

R_{t+1}^T is formed by first copying R_t^T, and then, with probability α, inverting a randomly chosen bit of R_{t+1}^T.

The utility function U is controlled by an eight bit desired result vector v_d:

$$U(v_r, v_d) = \frac{1}{8}\sum_{i=1}^{8} v_r(i) \,\mathrm{eqv}\, v_d(i).$$

v_d is unknown to the adaptive system at the outset but constant over all generations; it represents the stable component of the environment that is to be discovered by evolution and exploited by learning.

2.3 An ERL algorithm

In this study, we use a rather simple and generic form of evolutionary reinforcement learning. Here, the GENESIS genetic algorithm system [9] provides the evolutionary component, and a simplified form of the CRBP (complementary reinforcement back-propagation) algorithm [2] provides the learning component.

We adopted the default values for all the parameters in the GENESIS system (population size = 50, crossover rate = 0.6, mutation rate = 0.001, generation gap = 1.0, scaling window = 5, elitist strategy; see [9]), except for the "total trials" parameter (raised to 2,500), the "structure length" parameter (raised to 72 bits), and the use of the "a" option (since fitness is stochastic).

CRBP performs reinforcement learning using a back-propagation-style neural network [13], and a reinforcement function that returns a success or failure signal for each generated output. In this simulation, we used the simplest network — containing 8 input-output weights and 8 output biases — sufficient to represent all possibly optimal action functions.

The genetic representation of the initial action function a_0 occupies 64 bits, representing the 16 weights and biases in a range of ± 4, scaled into four bits each and gray-coded (see [9]). The reinforcement function compares each result vector to a genetically specified 8 bit goodness vector v_g, returns

Figure 2: Population average fitness versus time of evolution-only and evolution-learning.

success if and only if they match exactly.[1] The overall length of the genetic material is $64 + 8 = 72$ bits. Figure 1 details the algorithm.

2.4 Comparative data

We compared the behavior of evolution-only and evolution-learning strategies in these environments. The evolution-only strategy is identical to the evolution-learning strategy except steps 4 and 5 in Figure 1 are skipped. Fitness is scored as the fraction of maximum utility achieved, averaged over 1000 situations per organism, 50 organisms per generation, 50 generations per run, and 10 runs. A fitness score near 1.0 indicates that almost every situation is handled correctly by almost every individual in almost every generation in all the runs.[2] A fitness of 0.5 indicates chance performance.

We employ the online fitness measure for F_S because other common performance measures — such as best structure found, offline fitness, and final

[1]Thus, in this case, the reinforcement function, playing the role of the goodness function, attempts only to signal optimal values of the utility function, rather than to duplicate it.

[2]Since GENESIS is configured to minimize rather than maximize, we provided function values based upon averaging $100(1 - U(v_r, v_d))$.

Figure 3: Online fitness scores of evolution-only and evolution-learning algorithms, as a function of the environmental variation rate α. Except when $\alpha = 0$, all differences between the two algorithms are significant ($p < .001$).

population average — are either very noisy or dramatically misleading due to the varying environment. However, since the global averaging involved in online fitness can mask transient phenomena that occur during a run, in Figure 2 we display typical data produced by the evolution-only and evolution-learning strategies, as measured by average population fitness sampled at five generation intervals. Compared to typical optimizations in stationary environments, the population fitnesses over time are highly variable, but the overall effect is obvious: The evolution-learning strategy quickly attains and consistently maintains higher fitness compared to evolution alone.

The independent variable in our study is α, which determines the rate at which the result function r_t^T changes. When $\alpha = 0$, the environment is stationary. The data in Figure 2 were gathered at $\alpha = 0.005$. Figure 3 reports the system fitness F_S for the two algorithms, averaged over ten runs apiece for nine different values of α. Figure 4 displays the data from the lower α values in the form of fitness ratios of the two strategies, illustrating the relative advantage provided by learning.

As in other studies of evolution and learning (e.g., [12]), we observe an "inverted-U"-shape relating the independent parameter and the advantage

Figure 4: Ratios of evolution-learning to evolution-only system fitness at the lowest six tested values of α.

of learning. In a stationary environment, at the left of the graph, both algorithms perform almost equally well. This indicates that the populations using the evolution-only strategy were able to evolve an optimal action function reasonably quickly and therefore were able to produce the correct action in almost all of the cases. Similarly, the populations using the evolution-learning strategy evolved the correct goodness function (and perhaps the action function as well) and therefore were able to learn an effective action function and could spend most of their lives producing the correct action.

In a highly variable environment, at the right of Figure 3, neither strategy is able to find a useful action function. Performance for both strategies is at or very close to the chance level in these environments. As Figure 4 suggests, however, in between these extremes learning provides a substantial benefit. The evolutionary algorithm by itself tolerates only slow variation in the environment before its performance drops to chance. It seems that the genetic algorithm converges on an action function, and when environmental change necessitates an alternate action function, the population has difficulty responding quickly.

The evolution-learning algorithm, on the other hand, performs significantly better than chance even when the environment variation rate is so large that on average several changes occur in each lifespan. By "rolling with the punches," the evolution-learning strategy is able to stave off the decay towards chance even in quite variable environments.

Although not quite significant ($p < 0.015$ by the sign test), the evolution-learning strategy achieved slightly better mean performance even in a stationary environment. One might have expected that learning would be a drawback in that case, since each generation would waste time relearning the same action function. Actually, learning can serve as a "scout" to help evolution find good directions in which to evolve [10, 15, 5, 7]. Such "Baldwin effects" may account for evolutionary reinforcement learning's edge in the stationary environment.

3 Analysis

At least in this abstract and simplified case, we have seen that evolution combined with learning can outperform evolution alone across a wide range of environmental variation rates. But how good is good? How well are the algorithms really doing?

To investigate, we wrote down a differential equation modelling instantaneous fitness growth per situation under the assumption that in a stationary environment the growth rate would be proportional to distance from maximum fitness: $\frac{df}{dt} \propto (1 - f)$. Although we lack a rigorous justification for this choice, it can be motivated at least in part by noting that under certain conditions a hillclimbing algorithm will display such a growth rate. Far from the maximum uphill directions are easy to find, but as the maximum is approached, more and more time is spent looking at directions that lead downhill, and progress slows.

In our specific scenario, the constant of proportionality involves at least two factors: An inverse factor of $2w$ representing the size of the search space and an inverse factor of n representing the number of situations per lifespan. In addition, the non-stationary case requires an additional term to reflect fitness changes due to the environment's random walk in the $2w$ dimensional result function space, leading to

$$\frac{df}{dt} = \frac{\rho}{2wn}(1 - f) - \alpha\left(\frac{2f - 1}{2w}\right),$$

with a scale factor ρ that depends on the effectiveness of the algorithm. We solved this differential equation and integrated to obtain a theoretical online fitness measure, and then fitted the experimental data to the measure to estimate ρ's for the two algorithms. The evolution-only data yielded a $\rho_e = 0.56$ with an excellent fit — a test to distinguish the empirical data from the theoretical curve via z-scores fails ($p > 0.5$). This result suggests that although the evolution-only algorithm is successfully hillclimbing over the course of generations, and in spite of non-zero α, its hillclimbing rate is too slow to keep up except when α is small.

Although fitting the evolution-learning data to this model yields an impressive $\rho_{el} = 11.52$, the model in fact misses the empirical data, on average, by almost four standard deviations (z-score = 3.95). It is clear that this model — which does not recognize the value of constant utility — is inappropriate to the combined algorithm. To account for the higher performance of the evolution-learning algorithm properly, more sophisticated models, incorporating both the timescale of learning as well as the timescale of evolution, are needed.

4 Discussion

There is a growing selection of model computational systems displaying interaction effects between learning and evolution, with "learning" taken to refer to just about any sort of intra-individual-lifetime adaptive process. In these models, according to various criteria, learning provides an improvement compared to evolution alone. In Hinton & Nowlan's model [10], learning guides an evolutionary search toward a well-hidden maximum. Smith [15] points out that learning can also help compensate for the disruptive effects of sexual recombination. Fontanari & Meir [7] provide simulations and analytic results of a model that shows how learning allows an evolutionary search to function successfully under much higher mutation rates. In Miller & Todd's model [12], learning speeds evolutionary search given noisy environmental cues. In the present model, the addition of learning allows evolution to track a basically deterministic but rapidly changing environment.

In one sense, all that these models do is show that learning definitely is helpful under conditions expressly chosen to highlight learning's ability to help. On that view, the successes of these evolutionary learning algorithms are all very obvious. . . *after* the natures of the chosen conditions are understood. The search for computational strategies is as much a search for useful problem settings as it is for effective problem solvers, and it is in that sense that these models are particularly relevant. It is all too easy to buy into some simple conceptualization and then overlook alternate or more complex approaches in the attempt to fit as much of reality as possible into the adopted view.

Viewing a model solely in terms of an adaptive system and an environment, we feel, can often be such an overly-simple perspective. In the present model we have, in effect, a three-way distinction between adaptive system, changeable environment, and constant (or very slowly changing) utility. If one forces this scenario into the simpler two-way view, the constancy of the utility function is obscured by the changeable environment, increasing the plausibility of the mistaken notion that non-stationary environments are fundamentally incompatible with adaptive systems simply by virtue of being non-stationary.

Although the "adaptive system versus environment" viewpoint is undeniably parsimonious, the proposed three-cornered approach may in reality be a much more useful perspective. After all, the nominal "environment" of many natural and artificial adaptive systems includes both an "external" environment and a privileged subsystem — a "body" — that controls and is controlled by the adaptive system. In such cases, although the utility function — some measure of reproductive success, perhaps — requires much of a lifetime to evaluate, the proximal needs of the body — hunger, thirst, and so forth — provide a goodness function that is a very useful first approximation to utility. With some ability to learn from the goodness function, new effective actions can be discovered fairly quickly in the face of all sorts of changes in the external environment, everything from the onset of an ice age or a river changing course down to the rise of home pizza delivery or the microwave oven.

A related circumstance occurs when a short-term-stable environment is significantly non-stationary over a longer term due to coevolution of multiple adaptive systems. The prey become harder to catch and the predators must change in response, or the predators become better at hunting and the prey must change in response. Of course, all such changes can simply be referred back for evolution to handle over the course of generations, and evidently that is what happens in many biological adaptive systems; what we have seen in this paper is that adding learning can provide a faster response than evolution can alone, allowing a greater rate of environmental change to be tolerated.

We believe that the perspective of the constant utility non-stationary environment perspective captures a wide range of natural and artificial adaptive systems. In such cases, evolutionary reinforcement learning is one obvious place to start. There is much to learn.

References

[1] Ackley, D.H. & Littman, M.L. (1990). Learning from natural selection in an artificial environment. In M. Caudhill, (ed.) *Proceedings of the International Joint Conference on Neural Networks, IJCNN-90-WASH-DC*, Volume I, 189–193, Lawrence Erlbaum Associates: Hillsdale, NJ.

[2] Ackley, D.H. & Littman, M.L. (1990). Generalization and scaling in reinforcement learning. In D. Touretzky, (ed.) *Advances in Neural Information Processing Systems – 2*, 550–557, Morgan Kaufmann: San Mateo, CA.

[3] Ackley, D.H. & Littman, M.L. (1991, in press). Interactions between evolution and learning. In C.G. Langton, (ed.) *Artificial Life II*, Morgan Kaufmann: San Mateo, CA.

[4] Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist*, 30: 441-451, 536-553.

[5] Belew, R. (1989). Evolution, learning and culture: Computational metaphors for adaptive algorithms. University of California at San Diego, CSE Technical report CS89–156, La Jolla, CA.

[6] DeJong, K.A. (1980). Adaptive system design: a genetic approach. *IEEE Trans. Syst., Man, and Cyber., SMC-10*(9), 566–574.

[7] Fontanari, J.F., & Meir, R. (1990). The effect of learning on the evolution of asexual populations. *Complex Systems* **4**, 401–414.

[8] Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, MA.

[9] Grefenstette, J.J. (1987). *A user's guide to GENESIS.* (Available with the GENESIS system via electronic mail to gref@NRL-AIC.ARPA).

[10] Hinton, G.E. & Nowlan, S.J. (1987). How learning can guide evolution. *Complex Systems*, **1**, 495–502.

[11] Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

[12] Miller, G.F. & Todd, P.M. (1990). Exploring adaptive agency I: Theory and methods for simulating the evolution of learning. In D.S. Touretzky, J.L. Elman, T.J. Sejnowski, & G.E. Hinton (Eds.), *Proceedings of the 1990 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann, 65–80.

[13] Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. Chapter 8 of D.E. Rumelhart & J.L. McClelland (Eds.) *Parallel Distributed Processing: Explorations in the microstructures of cognition. Volume 1: Foundations.*

[14] Schull, J. (1990). Are species intelligent? *Behavioral and Brain Sciences*, 13:1, 61–73.

[15] Smith, J.M. (1987). When learning guides evolution. *Nature*, **32**, 761–762.