Improving the Performance of Evolutionary Optimization by Dynamically Scaling the Evaluation Function*

Alex S. Fukunaga and Andrew B. Kahng
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095-1596, USA
{fukunaga,abk}@cs.ucla.edu

ABSTRACT

Traditional evolutionary optimization algorithms assume a static evaluation function, according to which solutions are evolved. *Incremental evolution* is an approach through which a dynamic evaluation function is scaled over time in order to improve the performance of evolutionary optimization. In this paper, we present empirical results that demonstrate the effectiveness of this approach for genetic programming. Using two domains, a two-agent pursuit-evasion game and the Tracker [6] trail-following task, we demonstrate that incremental evolution is most successful when applied near the beginning of an evolutionary run. We also show that incremental evolution can be successful when the intermediate evaluation functions are more difficult than the target evaluation function, as well as when they are easier than the target function.

1. Introduction

Genetic programming (GP) [8] is an automatic programming method, inspired by biological evolution, which has been successfully applied to a wide variety of program induction tasks. While genetic algorithms [5] typically apply biologically-inspired evolutionary operators to fixed-length representations of task solutions, GP applies analogous operators (selection, crossover, mutation) to tree-structured programs (such as LISP S-expressions).

Like other approaches to evolutionary optimization, GP is computationally intensive: methods for accelerating the learning process are necessary. A number of techniques for improving the efficiency of GP have been proposed. These include extensions to the basic genetic programming model (e.g., mechanisms such as automatically defined functions [9]) and variations on basic genetic operators (e.g., brood selection [17]). These previous approaches concentrate on the search *algorithm*, i.e., the mechanism by which the space of genetic programs are explored. A complementary approach is to focus on the searchability genetic programs that is explored (that is, by the fitness function over the space of possible genetic programs. In this paper, we propose *incremental evolution*, a method for decreasing the computational effort of evolving the solution to a difficult problem by first evolving solutions to "easier" problems.

*Partial support for this work was provided by NSF Young Investigator Award MIP-9257982. The UCLA Commotion Laboratory is supported by NSF CDA-9303148.

The intuition behind this approach is attractive:

- It is often easier to learn difficult tasks after simpler tasks have been learned; and
- It may therefore be advantageous to use an "easier" fitness function when evolving solutions to complex problems.

This intuition is consistent with the phenomenon of scaffolding, which has been studied in psychology [15].

This paper presents an empirical study of incremental evolution applied to GP. In Section 2, we define the technique more precisely. Section 3 then presents experimental results demonstrating that incremental evolution can yield performance improvements in GP. We show that the successful use of this technique requires more than simply choosing an easier evaluation function on which to first evolve the population, and we also present extensive empirical results that demonstrate the complexity of obtaining performance improvements using this technique. Related work is discussed in Section 4. Finally, Section 5 gives some conclusions and directions for future work.

2. Incremental Evolution

The essential idea of incremental evolution is to *scale* the evaluation function (i.e., the "fitness function" against which, say, a robot controller is evolved) over time, with the aim of minimizing the overall time spent evolving a controller that achieves the prescribed task. Suppose that our goal is to generate, within a prescribed time limit T, a program to optimize some evaluation function G. The problem of incremental

evolution is to derive a set of intermediate evaluation functions $\mathcal{G} = (G_0, G_1, \ldots, G_{k-1} = G)$ and a schedule $\mathcal{S} = (t_0, t_1, \ldots, t_{k-1})$, such that $t_0 + t_1 + \ldots + t_{k-1} = T$. The population of controllers is sequentially evolved using evaluation function G_k for time t_k, beginning with G_0 for time t_0.

Let $\tau(\mathcal{G}, \mathcal{S}, Q)$ be the total processing effort (e.g., CPU time) required to evolve a solution of quality Q for the task G, given the sequence of tasks $\mathcal{G}$ and the schedule $\mathcal{S}$. Given any final evaluation function G and a desired solution quality Q, we wish to be able to choose $(\mathcal{G}, \mathcal{S})$ so that $\tau(\mathcal{G}, \mathcal{S}, Q)$ is minimized. This is a non-trivial, meta-level optimization, and a methodology for computing optimal $(\mathcal{G}, \mathcal{S})$ sequences for arbitrary G is unlikely. Indeed, certain choices of $(\mathcal{G}, \mathcal{S})$ may result in a performance degradation when compared with the trivial schedule that uses $\mathcal{G}' = (G)$ and $\mathcal{S}' = (t_0 = T)$, that is to say, $\tau(\mathcal{G}, \mathcal{S}, Q) > \tau(\mathcal{G}', \mathcal{S}', Q)$. In this paper, we seek effective heuristics for choosing $(\mathcal{G}, \mathcal{S})$. We believe that many research issues must be addressed in order to be able to make principled, effective use of incremental evolution; the experiments described below are a first step in addressing these issues.

3. Empirical Studies

To gain initial understanding of the mechanisms and the utility of incremental evolution, we performed an empirical analysis of the case where there is only one intermediate task (i.e., evaluation function) and only one transition between evaluation functions. In other words, we use $k = 1$, $\mathcal{G} = (G_0, G_1)$, and $\mathcal{S} = (t_0, t_1 = (T - t_0))$; it is understood that G_1 is the final, or "target", evaluation function G.

3.1. Task Domains

The principal task domain we study is the two-agent differential game of planar pursuit-evasion, involving a faster *pursuer* agent chasing a slower *evader* agent.[1] In our experiments, the task is to evolve a controller for the evader. The world is continuous (although the simulation occurs in discrete steps), two-dimensional, and is populated by only the pursuer and evader (i.e., no obstacles). The evader's evaluation function is the number of time steps that it eludes the pursuer, plus its final distance from the pursuer.[2] In order to apply incremental evolution, we generated pairs (G_0, G_1) by choosing different relative speeds of the evader with respect to the pursuer. Clearly, all else being equal, it is easier for a faster evader to succeed (achieve a higher fitness score) than a slower evader.[3]

Our secondary task domain is the Tracker problem [6], a complex task inspired by the trail-following behavior of ants.[4] A hungry, artificial ant is placed in a two-dimensional, toroidal grid world populated by food arranged in an irregular trail, and the task is to generate a controller that maximizes the amount of food picked up by the ant (the ant is given a limited amount of time during which to pick up the food). The ant has an orientation of up, down, left or right; it is able to sense whether there is food in the cell ahead of it and move horizontally or vertically on the grid. When an ant moves onto a cell containing food, the cell is cleared (i.e., it is assumed that the ant picks up the food).

The difficulty of the Tracker problem stems from the irregularity and the "gaps" in the trail; see [6] for a thorough analysis. We used the *Santa Fe* trail [8] (see Figure 1, reproduced from [8]) as the target evaluation function (G_1) for optimization. To apply incremental evolution, we first removed all gaps at corners, to obtain the *Intermediate* trail. The trail was further simplified to the *Easy* trail by replacing double gaps in the trail with single gaps. Note that the intermediate and easy trails were shortened at the end to maintain the total amount of food at 89 units; thus, the maximum fitnesses achievable on all three trails are the same.

We used steady-state GP [14] with tournament selection (Figure 2). Incremental evolution was implemented by changing the fitness function at generation t_0.[5] No mutation was used. The population size was 500, and there were a total of 50 generations. The maximum depth of the initial S-expressions was 6, and the depth of S-expressions created by crossover was limited to 17.

3.2. Evidence of Priming

We say that (G_0, t_0) *primes* for G_1 if $\tau((G_0, G_1), (t_0, t_1), Q) < \tau((G_1), (t_0 + t_1), Q)$, i.e., the incremental evolution reduces the time required to reach the prescribed solution quality Q.

The performance of the GP algorithm (best fitness achieved after 50 generations, taking the mean over

[1] Koza [8] evolved both pursuers and evaders using genetic programming. Recently, Reynolds [13] has used coevolution to evolve pursuers and evaders, and the merits of this task as a testbed for the evolution of adaptive behavior have been discussed in [11].

[2] To be specific: the pursuer moves a distance of 1.0 in every time step, and there are a total of 50 time steps. The initial vector from pursuer to evader is a random lattice point in $[-5, 5] \times [-5, 5]$. The final distance is taken to be zero if the evader is captured. Note that other relative weightings of time steps and final distance in the evaluation function are possible.

[3] We tested this intuition experimentally – see Section 3.2.

[4] Jefferson et al. [6] addressed this problem by evolving finite-state-automata and neural network controllers. The same task was subsequently addressed by Koza [7] using genetic programming.

[5] Because it is a steady-state GP, by "generation" we mean a "generational equivalent", or 500 individual fitness evaluations. To decrease the noise in the experimental values and isolate the effects of the parameters we controlled, our code allows re-use of the random seeds used to generate initial populations. In other words, run $\#k$ of one experimental group (corresponding to a single entry in each of the tables below) used the same random seed as run $\#k$ of another experimental group. Thus, for example, we could track the difference between $(\mathcal{G}, \mathcal{S}) = ((G_0, G_1), (t_0, t_1 = (T - t_0)))$ and $(\mathcal{G}', \mathcal{S}') = ((G_1), (T))$ for each initial population.

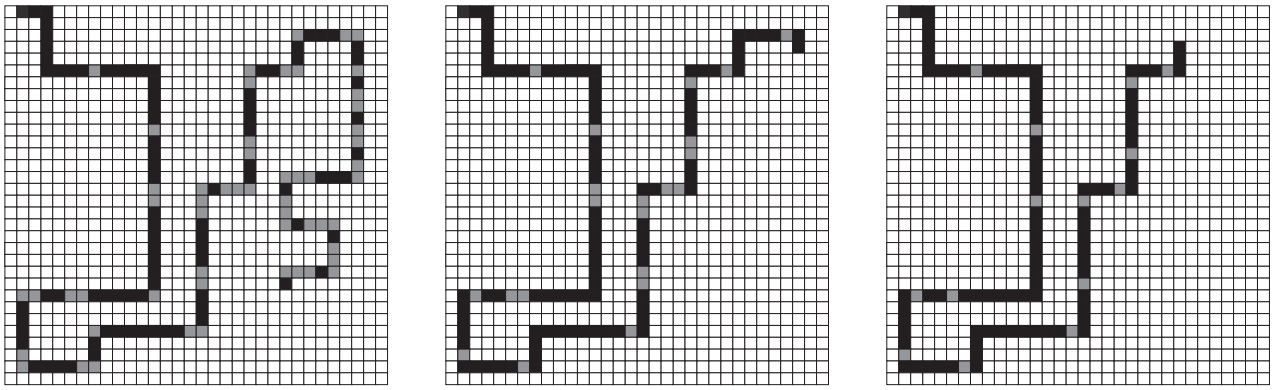

Fig. 1: The Tracker Problem. From left to right, these are the $SantaFe$, $Intermediate$, and $Easy$ trails. These are 32×32 toroidal worlds with 89 units of food arranged in an irregular trail. The dark regions indicate regions occupied by food. The light regions indicate gaps on the trail. The ant is initially placed at the top left corner of this world.

```
Initialize:
    Initialize population P
    Initialize F = G_0
    Evaluate each genome in P using fitness function F
for generation = 1 to NumberOfGenerations
  If generation = t_0 + 1
    F = G_1 .
  for i = 1 to PopulationSize
    Select two parents by tournament selection.
    Create a child by crossing the parents.
    Select m ∈ P by tournament selection.
        (weighted to select lower fitnesses).
    Overwrite m with the newly created child.
    Evaluate child using the current fitness function F.
  end for
end for
```

Fig. 2: Steady-State Genetic Programming with Incremental Evolution.

30 separate runs) was measured on the pursuit-evasion problem for each of a number of evader speeds ranging from 5% to 100% of the pursuer speed. These served as control data against which incremental evolution was tested. As expected, the GP performs better when evaders are faster, verifying the intuition that these are indeed "easier" evaluation functions (problems).

We assessed the effectiveness of incremental evolution under various conditions. For pursuit-evasion, G_1 was set to the evaluation function in which the evader's speed was 70% of the pursuer's speed. The GP algorithm was run for 50 generations total (i.e., $t_0 + t_1 = 50$). Two experimental parameters were varied: t_0 was varied at 10-generation intervals (i.e., $t_0 = (10, 20, 30, 40)$), and the speed of the evader in G_0 was set to 5, 10, 30, 50, 60, 70 (control), 80, 90, and 100 (percent of the pursuer's speed). The data (Table 1) show the fitnesses of the best member of the population after 50 generations. The values shown are the mean of 30 trials, along with standard error.[6]

We first studied the effect on performance of the time t_0 at which the fitness function is changed. Consider the mean ($N = 30$) of the best fitness achieved after 50 generations total (i.e. $t_0 + t_1 = 50$) as t_0 is varied at regular intervals from 10 to 40 generations. We observed strong evidence of priming: as t_0 is increased, the performance curves have a unimodal peak when t_0 is close to the beginning of the run (≤ 20 generations); as t_0 was increased, we observed a degradation of performance to levels significantly worse than the control values corresponding to evaders being evolved for all 50 generations without incremental evolution (i.e., $t_0 = 50$). It is interesting to note that when t_0 is less than its best value, performance is still consistently higher than that of the control group. In other words, it seems that even if incremental evolution fails to yield improvement for a particular case, its performance is no worse than nonincremental evolution if t_0 is relatively small with respect to the total amount of time ($t_0 + t_1$).

We next sought to obtain a correlation between the difficulty of G_0 and the performance of incremental evolution.[7] Table 1 shows the comparative performances of the GP algorithm as G_0 was varied between evader speeds of 5% to 100% of the pursuer's speed.

[6] For both pursuit-evasion and Tracker, each 50-generation run took about 20 minutes of time on a Sun Sparc-5 workstation. Thus, each entry in the tables represents approximately 10 hours of CPU time.

[7] The precise definition of *difficulty* is not yet clear, and robust parameterizations of difficulty remain an open research issue. Therefore, for now we informally say that G_0 is *more difficult* than G_1 with respect to a given algorithm (e.g., a GP optimization with fixed parameterization) if the performance of the algorithm on G_0 after a given time is better than the performance of the algorithm on G_1 (vis-a-vis the maximum attainable values for each evaluation function). For example, in our experiments, the pursuit-evasion problem is easier when the evader's speed is higher, because if all other things are equal, it is easier for the evader to elude the pursuer for a longer time, and thus obtain a higher fitness value.

	Relative Speed of Evader in G_0 (% of Pursuer Speed)								
t_0 (generations)	5	10	30	50	60	70	80	90	100
10	376.50 ± 1.98	380.33 ± 2.06	386.67 ± 2.38	387.63 ± 2.18	390.43 ± 1.61	381.73 ± 1.90	388.23 ± 1.92	386.83 ± 2.27	375.87 ± 2.37
20	377.50 ± 2.27	378.97 ± 2.26	385.60 ± 2.85	388.2 ± 2.42	387.47 ± 2.05	381.73 ± 1.90	388.77 ± 2.77	384.23 ± 2.12	372.8 ± 2.83
30	368.57 ± 2.85	372.13 ± 1.66	376.3 ± 1.66	383.87 ± 2.05	382.87 ± 1.77	381.73 ± 1.90	382.23 ± 2.51	380.63 ± 2.18	372.33 ± 2.58
40	358.27 ±2.29	361.83 ± 2.18	377.77 ± 2.98	378.73 ± 2.33	379.07 ± 1.86	381.73 ± 1.90	376.17 ± 2.35	379.47 ± 2.06	357.73 ± 2.12

Table: 1: Pursuit-Evasion: Performance of incremental evolution vs. t_0 = (10,20,30,40) generations, and evader speeds of 5-100% of pursuer's speed. t_1 = 50 - t_0. Fitnesses of best member of population after 50 generations (mean of 30 runs ± standard error) are shown. The control is the case where the $G_0 = G_1$ = (evader's speed = 70% of pursuer's speed), for which fitness is 381.73±1.90 (note that this is the same as t_0=0).

Surprisingly, the performance seems to have greatest dependence on the degree of similarity between G_0 (in this domain, similarity means that the speeds of the evader in G_0 and G_1 are similar), and the performance is bimodal around G_1.[8] There are *two* maxima on either side of $G_0 = G_1$ (the control), and performance drops as G_0 becomes more dissimilar to G_1. *In other words, more difficult tasks can prime easier tasks.* We then studied evader speeds between 65% and 75% in more detail to obtain a finer-grained view of the region where the maxima lie (Table 2). Statistically significant improvements in performance were found, especially for $t_0 = 10, 20$.[9]

	Relative Spd of Evader in G_0 (% Pursuer Spd)				
t_0	65	67	70	73	75
10	385.43 ± 1.92	387.77 ± 1.89	381.73 ± 1.90	394.23 ± 1.99	389.47 ± 2.73
20	385.50 ± 1.92	383.80 ± 1.78	381.73 ± 1.90	389.43 ± 2.15	387.00 ± 2.23
30	382.57 ± 1.81	381.47 ± 1.94	381.73 ± 1.90	388.90 ± 2.33	383.97 ± 2.16
40	382.10 ± 1.93	378.43 ± 1.82	381.73 ± 1.90	380.73 ± 2.19	374.90 ± 2.11

Table: 2: Pursuit-Evasion: Performance of incremental evolution vs. t_0 = (10,20,30,40) generations and evader speeds of 65-75% of pursuer's speed. The control is the case where the $G_0 = G_1$ = (evader's speed = 70% of pursuer's speed). t_1 = 50 - t_0. Fitnesses of best member of population after 50 generations (mean of 30 runs ± standard error) are shown.

To ascertain that priming could be observed for other values of G_1, and to observe the performance as t_0 was varied, we next let G_0 = (evader speed = 70% of pursuer speed), and varied G_1 between 10% to 100% of the pursuer's speed. Once again, t_0 was varied between 10 and 40.

Table 3 shows the results of this experiment, which indicate that priming occurs for various values of G_1, and that the relationship of performance to t_0 is similar for other values of G_1 (i.e., our previous results seem to be general for this domain). Again, statistically significant results for a 95% confidence interval can be seen in the table.

	Relative Spd. of Evader in G_1 (% Pursuer Spd)					
t_0	10	50	60	70	80	100
0	6.90 ± 0.06	159.03 ± 0.80	242.67 ± 1.03	381.73 ± 1.90	526.60 ± 0.76	668.07 ± 0.98
10	6.97 ± 0.06	160.33 ± 0.82	246.23 ± 1.40	381.73 ± 1.90	530.47 ± 0.88	670.00 ± 0.98
20	7.00 ± 0.00	161.93 ± 0.69	246.37 ± 1.55	381.73 ± 1.90	529.53 ± 0.81	669.23 ± 1.15
30	7.00 ± 0.00	160.33 ± 0.86	247.33 ± 1.31	381.73 ± 1.90	528.83 ± 1.03	668.33 ± 1.03
40	6.67 ± 0.09	157.80 ± 0.82	238.87 ± 1.09	381.73 ± 1.90	528.37 ± 1.05	666.83 ± 0.99

Table: 3: Pursuit-Evasion: Performance of incremental evolution vs. t_0 = (0,10,20,30,40) generations and evader speeds of G_0 = 70%, G_1 = 10-100% of pursuer's speed, and t_0 = 0. The control is the case where the $G_0 = G_1$. t_1 = 50 - t_0. Fitnesses of best member of population after 50 generations (mean of 30 runs ± standard error) are shown.

Finally, we studied the performance of the incremental evolution method in the Tracker domain, for all pairs of (G_0, G_1), where $G_0, G_1 \in (Easy, Intermediate, SantaFe)$.

Table 4 shows fitnesses of the best member of the population after 50 generations (mean of 30 runs ± standard error). The results are similar to those for pursuit-evasion (statistically significant performance improvements were found).[10]

An additional interesting observation is that there seems to be no "ordering" relationship between pairs of tasks (G_0, G_1) with respect to priming. That is, if (G_0, t_0) primes for G_1 for some t_0, then it is possible that (G_1, t_0') primes for G_0.

[8] We initially hypothesized that it would be better to use G_0 which is either easier or harder than G_1, and that the performance curve would be unimodal to one side of G_0.

[9] For the given sample size (N = 30), the differences in performance of the GP are statistically significant at a 95% confidence level when there is no overlap of the intervals bounded by the best fitness ± standard error.

[10] As with the pursuit-evasion domain, we verified that the *Easy* trail was easier than the *Intermediate* trail, which in turn was easier than the *Santa Fe* trail (see the "Control" row in Table 4).

	Trails used in G_0 and G_1					
t_0 (Generations)	G_0=I, G_1=E	G_0=SF, G_1=E	G_0=E, G_1=I	G_0=SF, G_1=I	G_0=E, G_1=SF	G_0=I, G_1=SF
0 (Control)	75.53 ± 2.09	75.53 ± 2.09	68.10 ± 1.77	68.10 ± 1.77	62.17 ± 1.53	62.17 ± 1.53
10	73.97 ± 2.04	70.80 ± 2.19	70.87 ± 1.96	70.37 ± 2.05	67.63 ± 1.92	63.33 ± 1.85
20	72.23 ± 1.92	72.10 ± 1.66	74.03 ± 1.94	66.20 ± 1.79	64.07 ± 1.71	61.17 ± 1.70
30	71.10 ± 1.90	69.53 ± 1.87	66.60 ± 1.93	68.03 ± 2.09	61.70 ± 2.02	61.23 ± 1.56
40	67.37 ± 1.89	67.27 ± 1.99	63.93 ± 2.15	66.17 ± 2.27	60.97 ± 2.22	60.13 ± 1.64
50	62.80 ± 2.02	58.33 ± 2.53	51.07 ± 2.94	56.53 ± 2.24	48.27 ± 2.95	59.47 ± 2.00

Table: 4: Tracker: Performance of incremental evolution vs. t_0=(10,20,30,40,50) and $(G_0, G_1) \in (Easy, Intermediate, SantaFe)$. E = $Easy$, I = $Intermediate$, SF = $SantaFe$. Fitnesses of best member of population after 50 generations (mean of 30 runs ± standard error) are shown.

4. Related Work

Previous work has addressed the problem of optimization in a dynamic environment ([12, 2, 10]. These researchers have considered the problem of adapting to a given dynamic environment. Our work differs fundamentally in that we consider the problem of *making* the environment dynamic in order to improve performance.

Recently, Harvey et al. [3, 4] have proposed this strategy of *incremental evolution*. They reported that evolving a neural network controller to visually guide a robot toward a small target in the environment took less total computational effort if the controllers were first evolved using a larger target. Our work differs from that of Harvey et al. [3, 4] in at least two major respects: (i) their representation scheme is different (a dynamical neural network), and (ii) they have only considered incremental evolution from an easier task to a harder task. As we have discovered, it is possible for incremental evolution to be successful when the intermediate task is more difficult than the final task.

Another somewhat related technique is the use of a *multi-phasic fitness environment* [1]. In [1], the task (collection of objects in a grid world) was separated into two *phases*: In the first phase (mapping), the agent looks for valuable objects and "remembers" their location, and in the second phase (planning), the agent plans a sequence of actions (consisting of movements and digging actions). The genetic program is separated into two branches[11] – the first branch is active during the mapping phase, and the second branch is active during the planning phase. This approach differs from incremental evolution in that the user intervenes to identify the subproblems that must be solved and manually impose a structure on the solution.

Incremental evolution also differs from previously suggested heuristics regarding test case selection. For example, Koza [8] suggested that test cases should be chosen to be a representative sample of the possible inputs to the genetic program.[12] In related work in evolutionary computation, Shultz [16] has studied biasing of test cases in order to improve performance in a genetic algorithm.

Last, we note that our approach to optimizing performance using incremental evolution also differs from co-evolution. Co-evolution maintains a high selective pressure by evolving test cases that become more difficult over time by adapting to the population being evolved. This is particularly useful after the population has been evolving for a while, as it avoids the low selective pressure that can result when a static test case is used (i.e., all members of the population have adapted to the test case). However, the initial population of test cases may already be relatively difficult, and it may be possible to improve performance by starting with an easier population of test cases. Recall that our results indicate that incremental evolution is most effective near the beginning of an evolutionary run.

5. Discussion and Future Work

Our experiments with incremental evolution have shown that incremental evolution can be used as a technique for improving the performance of GP. We have observed that for two-step incremental evolution, statistically significant performance improvements can be gained by choosing G_0 which is relatively similar to G_1, and transitioning between the two at an early stage of the optimization.

Our experiments have yielded some very interesting results, e.g., that performance may be dependent on the similarity, rather than relative ease or difficulty, of intermediate evaluation functions to the final evaluation function. It is clear that successful use of incremental evolution requires more than the simple intuition that it is easier to learn difficult tasks after

[11] Recall that a genetic program is an s-expression, which is naturally represented as a tree.

[12] In many cases, the space of inputs that can be given to a genetic program can be much larger than the feasible size of a set of test cases.

learning easier tasks. More work is necessary in order to determine more precisely the relationship between the intermediate evaluation function and the performance of the technique, and to fully understand the mechanisms that lead to performance gains. The understanding of these mechanisms will enable us to derive more useful heuristics for applying the technique.

We have attempted to obtain a stronger correlation between the relationship between G_0 and G_1 and performance. This has included studying the variance in the fitnesses of the members of the population, as well as observing the rate of convergence of the GP with respect to G_1 when a population was evolved for G_0.[13] Unfortunately, we have not yet been able to obtain a significant correlation. In future work, we plan to to track the genetic diversity (we have only considered phenotypic variance so far) of populations in order to shed some light on the underlying mechanism for priming. One factor that has made this analysis difficult so far is our use of genetic programming, for which the space of genotypes is very large, (i.e., there are many redundant solutions), and for which the neighborhood structure is less easily intuited than that of a standard genetic algorithm. Since there is every reason to believe that the underlying mechanism of incremental evolution is largely independent of the peculiarities of genetic programming, we are currently investigating the incremental evolution mechanism using genetic algorithms with fixed-length genotypes. This should enable a better understanding of the mechanism. Ultimately, we will scale up this research effort to analyze incremental evolution with more than one transition between test cases. This will involve many open issues regarding the optimization of the transition schedule between test cases.

Finally, the utility of incremental evolution must be assessed in additional domains. The technique seems to be naturally applicable in task domains (such as pursuit-evasion and Tracker) where controllers are generated for agents that perform tasks in domains whose "difficulty" is easily parameterizable (using domain-specific knowledge). More work is necessary in order to better formalize and exploit this notion of domain difficulty.

[13] We performed the following experiment: Let $Fit(I, G)$ be the fitness value of a genetic program I according to the evaluation function G, and $Best\_Of(Pop, t, G)$ be the member I^* of population Pop at time t with highest fitness according to G — in other words, $I^* = Best\_Of(Pop, t, G)$ maximizes $Fit(I, G)$ over all $I \in Pop$. A population Pop_0 was evolved in the usual manner using evaluation function G_0 for $t = 25$ generations. However, at each generation $1 \leq i \leq 25$ we also evaluated the current population using evaluation function G_1, and recorded the value of $Fit(Best\_Of(Pop, i, G_1), G_1)$. In other words, we evolved the population using G_0 as the evaluation function, but at every generation we also computed the fitness of the best individual in the population according to G_1 and saved this value. Using the same random seed and control parameters, we then evolved a population Pop_1 for $t = 30$ generations using G_1 as the evaluation function (note that at generation 0, Pop_1 is identical to Pop_0). For all values of t, we compared $Fit(Best\_Of(Pop_0, t, G_1), G_1)$ with $Fit(Best\_Of(Pop_1, t, G_1), G_1)$.

References

[1] D. Andre. Evolution of mapmaking: Learning, planning and memory using genetic programming. In *Proc. IEEE Int. Conf. Evolutionary Computation*, pages 250–255, 1994.

[2] H. Cobb and J. Grefenstette. Genetic algorithms for tracking changing environments. In *Proc. Fifth International Conference on Genetic Algorithms*, pages 523–530, 1993.

[3] I. Harvey, P. Husbands, and D. Cliff. Issues in evolutionary robotics. In *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pages 364–374, 1992.

[4] I. Harvey, P. Husbands, and D. Cliff. Seeing the light: Artificial evolution, real vision. In *From Animals to Animats 3: Proceedings of the Third International Conference on Adaptive Behavior*, pages 392–401, 1994.

[5] J. Holland. *Adaptation in natural and Artificial Systems.* University of Michigan Press, 1975.

[6] D. Jefferson, R. Collins, C. Cooper, M. Dyer, M. Flowers, R. Korf, C. Taylor, and A. Wang. Evolution as a theme in artificial life: The genesys/tracker system. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 549–577. Addison-Wesley, 1992.

[7] J. Koza. Genetic evolution and co-evolution of computer programs. In C. Langton, C. Taylor, J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 603–629. Addison-Wesley, 1992.

[8] J. Koza. *Genetic Programming: On the Programming of Computers By the Means of Natural Selection.* MIT Press, 1992.

[9] J. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

[10] M. Littman and D. Ackley. Adaptation in constant utility non-stationary environments. In *Proc. Fourth International Conference on Genetic Algorithms*, pages 136–142, 1991.

[11] G. F. Miller and D. Cliff. Protean behavior in dynamic games: Arguments for the co-evolution of pursuit-evasion tactics. In D. Cliff, P. Husbands, J.-A. Meyer, and S. W. Wilson, editors, *From Animals to Animats 3: Proceedings of the Third International Conference on Adaptive Behavior*, 1994.

[12] C. Ramsey and J. Grefenstette. Case-based initialization of genetic algorithms. In *Proc. Fifth International Conference on Genetic Algorithms*, pages 84–91, 1993.

[13] C. Reynolds. Competition, coevolution and the game of tag. In *Artificial Life IV*, 1994.

[14] C. Reynolds. An evolved, vision-based model of obstacle avoidance behavior. In C. Langton, editor, *Artificial Life III*, pages 327–346. Addison-Wesley, 1994.

[15] J. Rutkowska. Emergent functionality in human infants. In *From Animals to Animats3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, pages 179–188. MIT Press, 1994.

[16] A. C. Schultz. Adapting the evaluation space to improve global learning. In *Proc. Fourth International Conference on Genetic Algorithms*, pages 158–164, 1991.

[17] W. Tackett and A. Carmi. The unique implications of brood selection for genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence.* IEEE Press, 1994.