Lamarckian Evolution, The Baldwin Effect and Function Optimization

Darrell Whitley, V. Scott Gordon and Keith Mathias

Computer Science Department, Colorado State University, Fort Collins, CO 80523
whitley@cs.colostate.edu

Abstract. We compare two forms of hybrid genetic search. The first uses Lamarckian evolution, while the second uses a related method where local search is employed to change the fitness of strings, but the acquired improvements do not change the genetic encoding of the individual. The latter search method exploits the Baldwin effect. By modeling a simple genetic algorithm we show that functions exist where simple genetic algorithms without learning as well as Lamarckian evolution converge to the same local optimum, while genetic search utilizing the Baldwin effect converges to the global optimum. We also show that a simple genetic algorithm exploiting the Baldwin effect can sometimes outperform forms of Lamarckian evolution that employ the same local search strategy.

1 Introduction

A "hybrid genetic algorithm" combines local search with a more traditional genetic algorithm. The most common form of hybrid genetic algorithm uses local search to improve the initial population as well as the strings produced by genetic recombination. The resulting improvements are then coded onto the strings processed by the genetic algorithm. This is equivalent to a form of Lamarckian evolution.

Local search in this context can be thought of as being analogous to a kind of learning that occurs during the lifetime of an individual string. But there is another way in which learning (i.e., local search) and evolution can interact. Instead of coding the improvements back onto the string, the fitness value of the improvement can be transferred to the individual. This has the effect of changing the fitness landscape, but the resulting form of evolution is still "Darwinian" in nature. Various phenomena associated with this form of combining learning and evolution are collectively known as the *Baldwin effect* [2] [7] [1].

In this paper we compare a form of Lamarckian evolution and a simple genetic algorithm that exploits the Baldwin effect. We also compare these approaches to a simple genetic algorithm without learning. Our empirical and analytical tests look specifically at function optimization problems. We use a binary encoding for these problems; thus, local search in this context involves changing each of the L bits in a string encoding. Local search is limited to one step in the search space. When transferring fitness from one string to another under the Baldwinian strategy, we would like each string to have a unique evaluation. We therefore use steepest ascent as our local search instead of next ascent. [1]

Our analytical and empirical results indicate that Lamarckian strategies are often an extremely fast form of search. However, functions exist where both the simple genetic algorithm without learning and the Lamarckian strategy used in this paper converge to local optima while the simple genetic algorithm exploiting the Baldwin effect converges to a global optimum. We study this phenomenon using exact models of a simple genetic algorithm developed by Whitley [10] which are a special case of the Vose and Liepins models [8]. We also show that equivalence classes exist for functions, such that the functions in an equivalence class are processed in an identical fashion under specific forms of Lamarckian and Baldwinian search strategies.

1.1 Lamarckian Evolution versus the Baldwin Effect

Many of the "hybrid genetic algorithms" that combine local search and genetic search use what is in effect a form of Lamarckian evolution. Another way that learning and evolution can interact is to allow learning (i.e., local optimization) to change the fitness of an individual without altering the individual's genetic code. Local search, or learning, will have the effect of changing the fitness landscape. This is part of what is known as the *Baldwin effect* [7] [1]. Gruau and Whitley [5] have found that when evolving Boolean neural networks using grammar trees, genetic search that used learning to change the fitness function was just as effective as Lamarckian strategies: both Lamarckian strategies and search exploiting the Baldwin effect were more efficient and effective than genetic search alone.

This paper explores the combination of local and genetic search in the domain of function optimization for problems that have been encoded as binary strings. Our local search algorithm uses one iteration of steepest ascent. Given a current solution, flip each of the L bits; if the current solution is not a local optimum, then the *improved solution* is the best of the L possible neighbor states. If the improved solution is coded back onto the chromosome, we will refer to this as a Lamarckian search strategy. In this case, the next state replaces the current state. If the improved solution is merely used to change the fitness of the current state, then the search strategy will be referred to as Baldwinian.

Figure 1, taken from Gruau and Whitley [5], illustrates how local search can alter the fitness landscape. Taking N steps deepens the basin of attraction, thus

[1] In next ascent, the first improvement found is taken; if the neighbors of a string are checked in random order, next ascent does not yield a unique solution. For steepest ascent, all L bit changes are tested and the best improvement is taken.

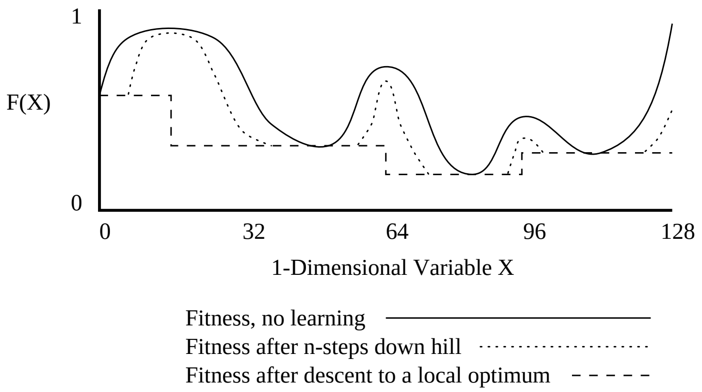

Figure 1: *The effect of local search on the fitness landscape of a one dimensional function. Improvements move downhill on the fitness landscape.*

making the function flatter around the local optimum. If local search is done to convergence, then the fitness function becomes flat in each basin of attraction. Note that each basin of attraction, however, has a potentially different evaluation corresponding to the evaluation of the local optimum. Changing the fitness landscape in this way creates the potential for impacting genetic search by increasing the likelihood of allocating more samples in certain basins of attraction.

Hinton and Nolan [7] were the first researchers to explore the *Baldwin effect* using genetic algorithms. They illustrate the Baldwin effect using a genetic algorithm and a simple random learning process that develops a simple neural network. The genetic encoding specifies a neural network topology by indicating which of 20 potential connections should be used to connect a given set of artificial neurons. The objective function (without learning) is such that the "correct" network results in increased fitness, and all other networks have the same, inferior fitness. This creates a spike at the "correct" solution in an otherwise flat fitness landscape. The genetic encoding uses a 3 character alphabet (0, 1, ?) which specifies the existence of a connection (i.e., 1), the absence of a connection (i.e., 0) or the connection can be unspecified (i.e., ?). If the connection is unspecified it can be set randomly or set via learning. Learning in this case is merely a random search of possible assignments to unspecified connections denoted by the ? symbol.

Hinton and Nolan found that the "?" symbol never fully disappeared from their genetic code when the search got close enough to the global optimum that it could be reached via learning. This appears to be a by-product of genetic drift coupled with premature convergence. Harvey [6] provides a good explanation for the anomalous results of Hinton and Nolan with respect to the "puzzle of the persistent questions marks." Belew [3] also offers a review and a critique of Hinton and Nolan.

2 Analytical Results

To better understand Lamarckian and Baldwinian search strategies an exact model of the simple genetic algorithm was used to study both approaches for solving small test problems. The model was used to track the expected representation of strings in an infinitely large population.

To model the effect of the Baldwinian strategy, only the fitness function needs to be changed. For each string i we will refer to evaluation-A as the value returned by the evaluation function used by the simple genetic algorithm without learning. When a Baldwinian search strategy is used, a second evaluation function is constructed. For each string i we define the evaluation-B for string i as the maximum evaluation-A value among the set of strings composed of string i and it's L neighbors in Hamming space. Running a simple genetic algorithm on function evaluation-B produces results identical to running a simple genetic algorithm on function evaluation-A and using one iteration of steepest ascent to change the evaluation of the current string.

An example function illustrates the effect of modeling the Baldwinian search strategy. The following 4 bit function was created by sorting the binary strings of length four according to their integer value. The string 0000 is given the value 28; as the integer value of the strings increases by 1 the value assigned to the string is decremented by 2. String 1110 has value 0. String 1111 is then assigned value 30. The following function will be denoted Function 1:

```
e(0000) = 28     e(0100) = 20     e(1000) = 12     e(1100) =  4
e(0001) = 26     e(0101) = 18     e(1001) = 10     e(1101) =  2
e(0010) = 24     e(0110) = 16     e(1010) =  8     e(1110) =  0
e(0011) = 22     e(0111) = 14     e(1011) =  6     e(1111) = 30
```

The function is posed as a maximization problem. This function is not fully deceptive [4] [9] since $\mu(***1) > \mu(***0)$ and $\mu(**11) > \mu(**00)$, where $\mu(\xi)$ is the average evaluation of all strings in hyperplane ξ. However, it does have a significant amount of deception. The following function $eb(i)$ represents evaluation-B constructed from Function 1.

```
eb(0000) = e(0000) = 28     eb(1000) = e(0000) = 28
eb(0001) = e(0000) = 28     eb(1001) = e(0001) = 26
eb(0010) = e(0000) = 28     eb(1010) = e(0010) = 24
eb(0011) = e(0001) = 26     eb(1011) = e(1111) = 30
eb(0100) = e(0000) = 28     eb(1100) = e(0100) = 20
eb(0101) = e(0001) = 26     eb(1101) = e(1111) = 30
eb(0110) = e(0010) = 24     eb(1110) = e(1111) = 30
eb(0111) = e(1111) = 30     eb(1111) = e(1111) = 30
```

Note that the basins of attraction are made deeper and flatter around each local minimum.

Under the Lamarckian strategy used here, the string with the best value in the local neighborhood of string i would always replace string i unless i has the best value, in which case string i is a local optimum. In this implementation of the Lamarckian search strategy the string distributions are altered at the beginning of each generation to model the effects of one iteration of steepest ascent. A string residing at a local optimum, for example, increases its own distribution by the sum of the distributions of all of its immediate L neighbors. However, these neighbor points do not necessarily end up with zero representation. Neighbors at a Hamming distance of two from the local optimum will also move downhill one step (assuming they are in the same basin of attraction), thus creating new representations for some strings that are one step closer to the local optimum.

Let $P(i,t)$ be the representation of string i in the population at time t. At the beginning of each generation the proportional representation of strings are changed by doing one iteration of steepest ascent. In the following equations, $P'(i,t)$ represents the distribution of strings after steepest ascent. Note that the redistribution of string representations will usually be different for functions that are different. For function 1, those changes are

```
P'(0000,t) = P(0000,t) + P(0001,t) + P(0010,t) + P(0100,t) + P(1000,t)
P'(0001,t) = P(0011,t) + P(0101,t) + P(1001,t)
P'(0010,t) = P(0110,t) + P(1010,t)
P'(0100,t) = P(1100,t)
P'(1111,t) = P(1111,t) + P(1110,t) + P(1101,t) + P(1011,t) + P(0111,t)
P'(0011,t) = 0
P'(0101,t) = 0
P'(0110,t) = 0
P'(0111,t) = 0
P'(1000,t) = 0
P'(1001,t) = 0
P'(1010,t) = 0
P'(1011,t) = 0
P'(1100,t) = 0
P'(1101,t) = 0
P'(1110,t) = 0.
```

Strings which lie on a saddle between basins of attraction have no representation after one iteration of steepest ascent. The number of points falling on the saddle between local optima is quite high in a small 4 bit function. What about larger functions? Obviously, a 100 bit function with two local optima will have a smaller percentage of saddle points as compared to the size of the search space; it follows that fewer strings will have zero representation in an infinite population genetic algorithm. However, two issues are worth considering.

1. As the number of local optima increase, generally the number of saddle points will also increase. When the maximal number of local optima exists in a binary coded space, then exactly half the points in the space are local optima and half are saddle points between the local optima. More

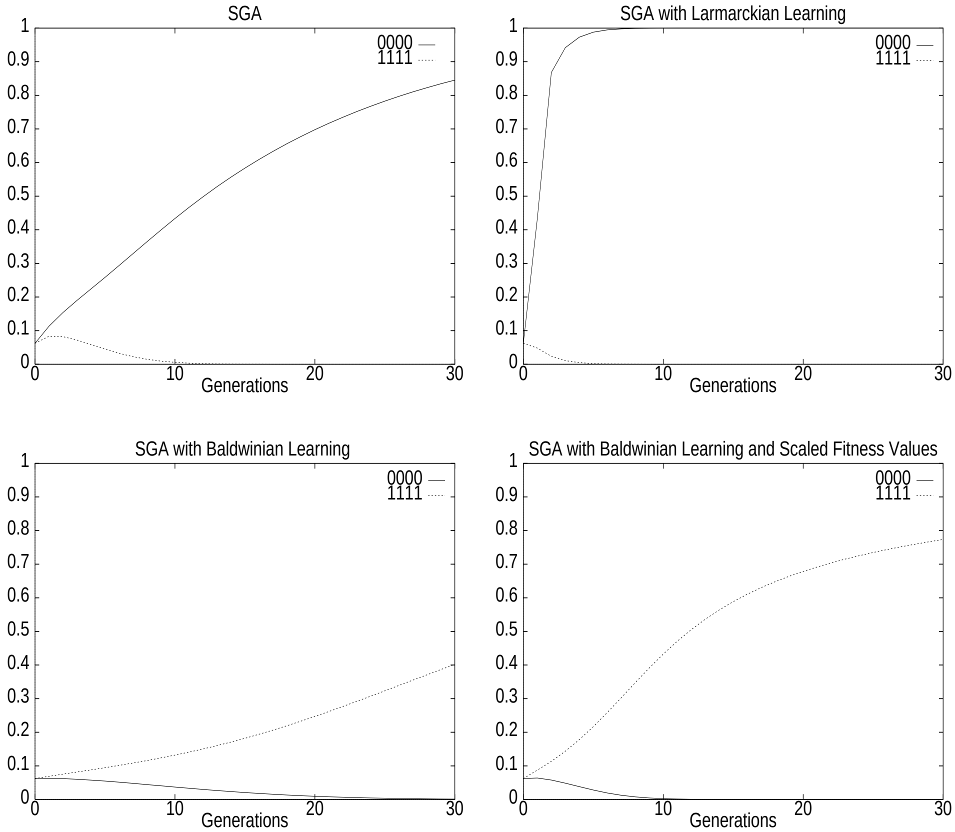

Figure 2: SGA behavior with and without Lamarckian and Baldwinian Learning. The string 0000 is a local optimum and 1111 the global optimum.

generally, when the basins of attraction have a radius greater than 1, the number of saddle points for each local optima will grow as a function of the Hamming distance from the local optima to the nearest saddle point.

2. In any finite population genetic algorithm, many points that have representation in the population before local search is applied may have no representation after local search is applied.

Figure 2 illustrates the results obtained using the exact model of a simple genetic algorithm to process Function 1. Figure 2 also includes the results of executing the evaluation-B version of Function 1 which models the behavior of the Baldwinian search strategy, as well as the results of the Lamarckian search strategy. The crossover rate was 0.6 and the mutation rate was 0 for all experiments. Both the simple genetic algorithm without learning and the Lamarckian search strategy converge to a local optimum. The Baldwinian search strategy converges to the global optimum, but does so slowly. One reason for the slow convergence of the Baldwinian search strategy is that there is less variation in the evaluation of strings in the space under the evaluation-B version of the function. All strings in the evaluation-B version of Function 1 have an evaluation between 20 and 30, and over half of the strings have an evaluation of either 28 or 30. Thus a one-time scaling of the function was performed by subtracting 20 from the value of all points in the space. This did not completely remove the scaling problem, but it did cause the genetic algorithm to converge faster.

2.1 Equivalent Classes Under Hybrid Genetic Search

Strings which lie on a saddle between basins of attraction have no representation after one iteration of steepest ascent; it follows that the actual evaluations of these strings are in some sense irrelevant. To be more specific, the evaluations of strings that are saddle points can change without effecting Lamarckian search as long as the gradient directions induced by local search do not change. Thus, Function 1 could be converted to the following function, which we refer to as Function 2:

```
e(0000) = 28      e(0100) = 20      e(1000) = 18      e(1100) = 18
e(0001) = 26      e(0101) = 24      e(1001) = 24      e(1101) = 18
e(0010) = 24      e(0110) = 22      e(1010) = 22      e(1110) = 18
e(0011) = 24      e(0111) = 22      e(1011) = 22      e(1111) = 30
```

About half of the hyperplanes now have fitness averages that favor the global optimum 1111 (e.g., $\mu(**1*) > \mu(**0*)$, $\mu(***1) > \mu(***0)$, $\mu(*1*1) > \mu(*0*0)$, $\mu(**11) > \mu(**00)$). While this function is in many ways quite different, it behaves identically to Function 1 under the Lamarckian search strategy defined here. This is because the set of equations describing the redistribution of strings under Lamarckian search remains exactly the same. Also note that any string in Function 1 which has representation after one iteration of steepest ascent does not change its evaluation in Function 2. Only saddle points change their evaluation. Thus, the same moves occur in both functions when one iteration of steepest ascent is applied. This implies that Function 1 and Function 2 are processed in an identical fashion under the Baldwinian search strategy as well, since all saddles points acquire their evaluation from other points in the space. Since the actual values of these saddle points are irrelevant, it is clear that there are sets of functions that are equivalent under the Lamarckian and Baldwinian search strategies; however, the different functions within these sets are still unique to the simple genetic algorithm without learning.

When a simple genetic algorithm without learning is executed on Function 2, the search again converges to the same local minimum. The results of using either the Lamarckian or Baldwinian strategy are identical to the results obtained for Function 1.

3 Empirical Function Optimization Tests

We tested the effects of adding Baldwinian and Lamarckian search strategies to a simple genetic algorithm on the following numerical optimization problems:

$$\textit{Rastrigin:}\ f(x_i|_{i=1,20}) = 200 + \sum_{i=1}^{20} x_i^2 - 10cos(2\pi x_i), \qquad x_i \in [-5.12, 5.11]$$

$$\textit{Schwefel:}\ f(x_i|_{i=1,20}) = V + \sum_{i=1}^{20} -x_i sin(\sqrt{|x_i|}), \qquad x_i \in [-512, 511]$$

$$\textit{Griewank:}\ f(x_i|_{i=1,20}) = \sum_{i=1}^{20} \frac{x_i^2}{4000} - \prod_{i=1}^{20} cos(\frac{x_i}{\sqrt{i}}) + 1, \qquad x_i \in [-512, 511]$$

For Schwefel's function V is the negative of the global minimum, which is added to the function so as to move the global minimum to zero. The exact value of V depends on system precision; for our experiments $V = 8379.65723$.

In these experiments, we used an elitist form of the standard simple genetic algorithm (*ESGA*). The probability of crossover was set to 0.7 and the mutation rate was 0.003. Tournament selection (with tournaments of size 2) and Gray coding were used in every case. The string length was 200 for all three problems. Local optimization is employed in the form of *steepest ascent*.

Table 1 shows performance data for each algorithm on each of the functions for a variety of population sizes. In particular, we consider population sizes of 50 and 100 for all three functions, as well as a population size of 400 for Griewank (since it is the hardest of the three functions for the genetic algorithm to solve). Table 1 shows how many times the genetic algorithm finds the global solution (out of 30 runs) within 1000 generations. (Note that these are *minimization* problems; the problems in section 2 were *maximization* problems.)

	Rastrigin		Schwefel		Griewank		
popsize =	50	100	50	100	50	100	400
ESGA	0	0	0	0	2	1	1
ESGA+Baldwin	30	30	19	30	6	8	15
ESGA+Lamarckian	30	30	9	25	14	16	30

Table 1: Performance of *ESGA* on three numeric functions. The number of times that the GA finds the optimal solution, out of 30 runs, is shown.

If one wishes to obtain results quickly, the Lamarckian strategy is consistently the best. However, if one is interested in the long term effects of these search strategies, then Lamarckian search appears to work better on the Rastrigin and Griewank functions, while Baldwinian learning works better on the Schwefel function. Figure 3 shows convergence graphs for the Rastrigin and Schwefel

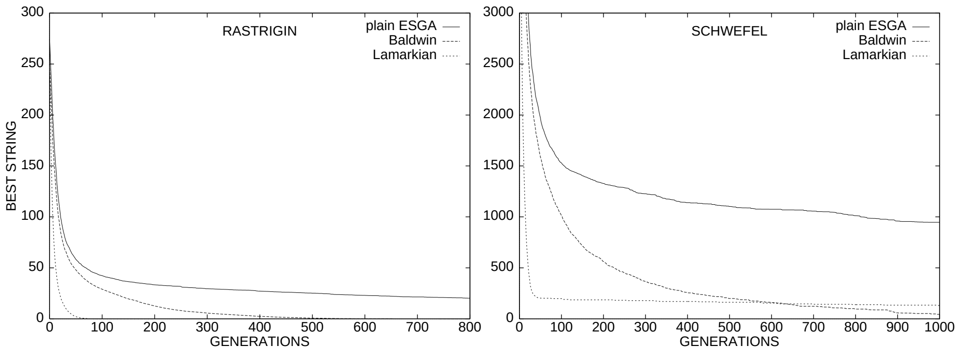

Figure 3: Baldwin vs. Lamarckian learning in two numeric optimization functions.

functions; the value of the best string in the population (averaged over 30 runs) is plotted for each genetic algorithm, using a population size of 50. Similar results are observed for larger population sizes. In all cases Lamarckian search results in faster improvement. However, on the Schwefel function the average best solution is poorer for Lamarckian search than Baldwinian learning after an extended period of time.

4 Conclusions

The results presented here are largely exploratory in nature. However, the results clearly indicate that a Baldwinian search strategy as defined in this paper can sometimes be more effective than a Lamarckian strategy using the same form of local search. It should be noted that by more effective, we mean that the Baldwinian search strategy will sometimes converge to a global optimum when the Lamarckian strategy converges to a local optimum. However, in all of the cases presented here, the Baldwinian search strategy is much slower than the Lamarckian search.

It is also clear that there exist equivalence classes of functions for Lamarckian and Baldwinian search strategies. An equivalence class in this case is defined such that all functions in the equivalence class are processed in an identical fashion under either a Lamarckian or a Baldwinian search strategy.

One notable difference between the results reported here and the results obtained by Gruau and Whitley [5] is that the Baldwinian search strategy did not prove to be as efficient as the Lamarckian search strategy on all of the function optimization problems studied in this paper. That may be due in part to the fact that these test problems are too easy. For the Rastrigin and Schwefel functions, the evaluation subfunction associated with each parameter is nonlinear, but the subfunctions applied to each parameter are simply summed together. Thus there are no non-linear interactions between the individual parameters. We have found that all three of these problems are relatively easy to solve using stochastic hill-climbing methods and thus it is not surprising that Lamarckian search is so effective on these problems. Ideally, the effectiveness of a Baldwinian search strategy should be evaluated in more complex domains. It should also be noted that Gruau and Whitley used a type of genetic algorithm based on the steady state genetic algorithm model rather than a simple genetic algorithm; this too could impact their results.

5 Acknowledgement

The original idea for using local search to change the fitness landscape was communicate to us by Rik Belew. This work was supported in part by NSF grant IRI-9312748.

References

[1] D.H. Ackley and M. Littman, (1991) Interactions between learning and evolution. In, *Proc. of the 2nd Conf. on Artificial Life*, C.G. Langton, ed., Addison-Wesley, 1991.

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

[3] R.K. Belew, (1989) When both individuals and populations search: Adding simple learning to the Genetic Algorithm. In *3th Intern. Conf. on Genetic Algorithms*, D. Schaffer, ed., Morgan Kaufmann.

[4] D. Goldberg, (1989) Genetic algorithms and walsh functions: Part II, deception and its analysis. *Complex Systems 3*:153-171.

[5] F. Gruau and D. Whitley, (1993) Adding learning to the cellular development of neural networks: evolution and the Baldwin effect. *Evolutionary Computation* 1(3):213-233.

[6] I. Harvey, (1993) The puzzle of the persistent question marks: a case study of genetic drift. In *5th Intern. Conf. on Genetic Algorithms*, S. Forrest, ed., Morgan Kaufmann.

[7] G.E. Hinton and S.J. Nolan, (1987) How learning can guide evolution. *Complex Systems*, 1:495–502.

[8] M. Vose and G. Liepins, (1991) Punctuated equilibria in genetic search. *Complex Systems* 5:31-44.

[9] D. Whitley, (1991) Fundamental principles of deception. *Foundations of Genetic Algorithms.* G. Rawlins, ed. Morgan Kaufmann.

[10] D. Whitley, R. Das, C. Crabb, (1992) Tracking primary hyperplane competitors during genetic search. *Anals of Mathematics and Artificial Intelligence,* 6:367-388.