Development and the Baldwin Effect

Keith L. Downing
Department of Computer Science
The Norwegian University of Science and Technology
Sem Selandsvei 7-9
7491 Trondheim, Norway
keithd@idi.ntnu.no

Abstract Baldwin's classic hypothesis states that behavioral plasticity can speed evolution by (a) smoothing the fitness landscape and (b) indirect genetic assimilation of acquired characteristics. This latter phase demands a strong correlation between genotype and phenotype space. But the natural world shows signs of this correlation at only a very coarse level, since the intervening developmental process greatly complicates the mapping from genetics to physiology and ethology. Hence, development appears to preclude a strong Baldwin effect. However, by adding a simple developmental mechanism to Hinton and Nowlan's classic model of the Baldwin effect, and by allowing evolution to determine the proper balance between direct and indirect mapping of genome to phenotype, this research reveals several different effects of development on the Baldwin effect, some promoting and others inhibiting. Perhaps the most interesting result is an evolved cooperation between direct *blueprints* and indirect developmental *recipes* in searching for unstructured and partially structured target patterns in large, needle-in-the-haystack fitness landscapes.

Keywords
Baldwin effect, development, genetic algorithms, trilateral adaptation

1 Introduction

To fully understand the origins of any trait, particularly a cognitive one, requires an investigation into the complex interplay between evolution and behavioral plasticity. Such an analysis spans both the wide range of geological time and space; as well as the gap between the microscales of genetics and the macroscales of ethology. While research on the Baldwin effect [3, 9, 17, 29] has yielded useful insights into the potential accelerating role of learning in evolution, both preconditions and caveats are plentiful, making the biological case plausible at best.

In a nutshell, the Baldwin effect consists of two phases. In phase I, learning gives a selective advantage to individuals with innate mental or physical abilities that place them relatively close to optimal behavior. Essentially, learning makes up the difference, allowing these individuals to achieve optimal behavior in the course of their lifetimes. Other individuals may also learn, but if their innate abilities are too distant from the optima, then a lifetime of plasticity is still not enough. So learning smooths the fitness landscape by giving some suboptimal individuals a fighting chance.

In phase II, learned optimal behaviors become innate by chance genetic events (mutation, inversion, and crossover). This only happens to genomes that survive the wait through many generations: those with the selective advantage incurred by learning plus fortuitous innate proximity to the optima. The natural-born talents then have a selective advantage over their predecessors, since they do not have to spend time and resources learning. The hypothesis of eventual dominance of the natural-born talents assumes that (a) the environment is nearly static, and (b) learning has a cost, which it normally does [17].

Another critical precondition for phase II appears to be a strong correlation between genotype and phenotype space [17]. Otherwise, the changes accomplished by phenotypic plasticity have little chance of eventually becoming encoded in the genome via chance genetic operations, or, if they do, they may happen to random individuals as opposed to those poised and waiting nearby in genotype space. Unfortunately, nature promises no such correlation, particularly when the phenotype space encompasses mental activities. The transition from genome to fully functioning brain is extremely intricate, even in very simple animals.

In short, the developmental process that governs the transition from genome to embryo to juvenile poses serious, possibly insurmountable, problems for the Baldwin effect. Development converts a linear string of DNA into a three-dimensional organism by a fascinating process, but one that severely confounds the genotype-phenotype mapping.

However, correlations do exist between the spatial locations of (a) homeobox genes on the chromosomes of all animals from *Drosophila* to humans, and (b) body parts (such as an insect's thoracic and abdominal segments) [30]. In fact, some homeobox gene locations even correlate with regions of the hindbrain, which is the evolutionarily most primitive brain region. However, each neural module consists of thousands or millions of neurons whose heterogeneous activation patterns determine the animal's overt behavior. No single pattern or group of similar patterns, hence no single behavior, maps directly to a gene. Hence, learned behaviors have no obvious possibilities for assimilation into the genome. To illustrate the absurdity of any strong cognitive Baldwin effect, consider the human brain, which consists of roughly 10^{11} neurons. Using the gross oversimplification that these neurons have but two states (on and off) gives a neural state space of size $2^{100000000000}$. This maps very poorly to the human genome, which has a mere 30–40 thousand genes.

Despite this major impediment, mental Baldwinism could be rescued via an acceptable refinement: although learning makes very specific behavioral changes, there could be corresponding genetic changes for that same *general type* of behavior. Thus, while a chess player learns specific opening moves, general board formations, and effective move sequences, these are manifest in synaptic changes to localized parts of brain regions such as the basal ganglia. Then, the coarse genetic correlate might be the modification of a few genes whose phenotypic consequences are slight changes in postnatal concentrations of particular neurotransmitters and neuroreceptors across the *entire* basal ganglia. Thus, the child may have innate talent for activities requiring good sequence memory, but no immediate prowess at chess. Similarly, the words of human language have no direct genetic correlates, but the ability to acquire language may have genetic components [23]. In fact, Pinker and Bloom claim that the Baldwin effect is instrumental in the emergence of both language and Chomsky's proposed language acquisition device [5]. Interestingly enough, simple ALife experiments reveal an evolving predisposition toward language [21, 4].

Unlike Lamarckianism [16], which implies the biologically improbable encoding of acquired traits directly and immediately (i.e., within the learning organism's lifetime) into the genome, Baldwinism remains a viable biological possibility. However, regardless of biological blessing or condemnation, both Lamarckianism and Baldwinism are useful tools for evolutionary computation [11, 2, 6], since they supplement evolution with learning to improve overall search. Furthermore, evolutionary algorithms have increasingly exploited *indirect* genomic representations, which are converted to phenotypes by an (often complex) developmental process. These are particularly useful for difficult search problems such as circuit design [15]. So, even if biological research eventually shows that development so dramatically corrupts the genotype-phenotype mapping as to soundly dismiss the Baldwin effect, there will remain a need to understand the interactions between development and the Baldwin effect in *artificial* adaptive systems.

This article begins that investigation back at the roots of the artificial Baldwin effect: the classic experiments of Hinton and Nowlan [9]. In their work, the genotype-phenotype mapping was 1-to-1, with no developmental process. Our work enhances their genome with a simple developmental mechanism, in the form of a restricted Turing machine. Evolution is then free to exploit development completely, partially, or not at all in searching for a target phenotype. The resulting system thereby combines evolution, development, and learning in a very abstract framework, suitable for exploring general principles of trilaterally adaptive systems.

2 Related Work

The evolutionary computation and ALife literature is replete with systems that combine evolutionary algorithms (EAs) with learning. Classifier systems [10, 24] introduced genetic algorithms (GAs) and bucket-brigade learning as early as the 1970s, but more than a decade passed before other EA-learning hybrids appeared. Montana and Davis [19] were the first to use GAs to evolve weights for artificial neural networks (ANNs). This technique has become very popular because it offers a major speedup over standard backpropagation on many types of problems. It is also widespread in fields such as evolutionary robotics [22], where (a) ANNs are the dominant controller, and (b) supervised learning (via backpropagation) is inappropriate. Ackley and Littman [1] evolved ANN weights for simulated organisms and became the first to observe the Baldwin effect in an ALife context. For an extensive review of evolving ANNs, see [31].

In another direction, genetic programming (GP) [14] has been combined with reinforcement learning (RL) [28] in attempts to improve RL [12] and/or GP [6]. This area has yet to be fully explored.

Also, the use of indirect-coded genomes that require developmental translation into phenotypes is widespread in evolutionary computation. Kitano [13] evolved graph-generating grammars for ANN topologies. Since the ANNs could then learn by backpropagation, Kitano was the first to combine evolution, development, and learning. Another classic is Gruau's cellular encoding [7], which evolves GP genomes for growing ANN topologies and determining connection weights. In truly extraordinary fashion, Karl Sims [25] pioneered the use of ALife in computer graphics by evolving simple graphs that governed the development of three-dimensional animat morphologies and their ANN controllers. With the advent of off-the-shelf physical simulators, many others have begun to follow in Sims' footsteps.

Clearly, combinations of evolution, development, and learning have a long history in the EA and ALife communities. But few systems combine all three adaptive mechanisms, and those that do generally use ANNs, since (a) their graphlike structure is relatively easy to grow from an embryonic genome, and (b) many well-established learning algorithms can be used to update weights during training. However, on performance tests, the most promising developmental approaches to ANN evolution lag well behind distinctly nonbiological systems such as SANE [20] and NEAT [26].

Basically, few relevant problem domains have been best served by trilateral adaptation. The more common discussions are over the relative merits of direct versus indirect coding, with no subsequent learning, or whether to include local search as a primitive learning method for directly coded phenotypes. Still, to fully understand intelligence and its emergence, a trilaterally adaptive system using neuronlike structures may be one of the few realistic alternatives.

Several researchers [17, 8, 18, 21] have expressed deep concern over the corrupting influence of development on the Baldwin effect. Although few have done trilaterally adaptive simulations to test their hypotheses, Gruau and Whitley [8] did find a negative effect of cellular-encoding-based development on Baldwinian evolution, as expected. To date, no positive influences have been documented.

Finally, Stanley and Miikkulainen [27] give a comprehensive overview of artificial embryogeny, covering a plethora of embryogenic systems and composing a useful new biologically based classification of them. However, by virtue of its simplicity and nonbiological nature, the developmental mechanism considered in this article has no useful place in their classification scheme. Their discussions of the evolution and embryogenesis of complex forms provide a nice parallel to the introduction of this article, although they do not touch on the Baldwin effect. Still, their five fundamental dimensions of development (cell fate, targeting, heterochrony, canalization, and complexification) merit further consideration when looking deeper into the genotype-phenotype mapping problem and its consequences for the Baldwin effect.

3 TRIDAP

Summarized in Figure 1, the trilaterally adaptive system (TRIDAP) uses a standard genetic algorithm [10] with a linear bit-vector genome. Its goal is to find a particular target string of 0's and 1's. Hinton and Nowlan [9] use the rough analogy between these strings and connection patterns of nodes in a neural network. The genome encodes a Turing-based developmental process, which runs on an initial tape, also encoded in the genome. The final, developed tape can further improve itself through learning, with the actual amount determined by evolution and development.

3.1 The Chromosome

Each chromosome contains four key regions: (1) ratio, (2) TM (Turing machine), (3) intron, and (4) tape. Only the ratio region has a fixed size: $3k$, which encode three integers: r_1, r_2, and r_3. If a chromosome has a length of N bits, and each ratio is encoded with k bits, then $N - 3k$ bits remain to encode the TM, intron, and tape regions. The ratio integers determine the fraction of those $N - 3k$ bits that are allotted to these three remaining regions. As a simple example, if the ratio integers are (8,5,7), then the TM gets the first 8/(8+5+7) = 40% of the $N - 3k$ bits, the introns get the middle 25%, and the tape gets the final 35%. For the examples in this article, $N = 300$ or 500, and $k = 5$. Also, all genes use standard binary encoding.

The variable-length tape region is interpreted as a list of m-bit segments, each of which encodes an integer between 0 and $2^m - 1$. These integers are partitioned into three nearly equal groups, such that each m-bit segment encodes the integers 0, 1, and 2, with nearly equal probability (given suitably large m). On the translated tape, 0 and 1 denote simple binary values, while 2 denotes a wildcard, which is used to support the local search or bit guessing that corresponds to learning in the Hinton-Nowlan model.

The intron region is completely ignored by the developmental and learning processes. Its sole purpose is to separate the TM and tape regions so that they can grow or contract relatively independently of one another. For example, the TM region can expand without taking bits away from the tape region, or both tape and TM can contract when the intron region expands, as determined by the ratio region.

The TM region encodes transition rules as 5-tuples of the form (s, x, s^*, x^*, a). Here, s is the current TM state and x is the tape symbol being read, while s^* is the next state of the TM, and x^* is the next symbol. The fifth element, a, is the action to perform: either overwrite x with x^*, or insert x^* to the immediate right of x on the tape. Each transition rule requires $2p + 2m + 1$ bits, where m and p are the numbers of bits used

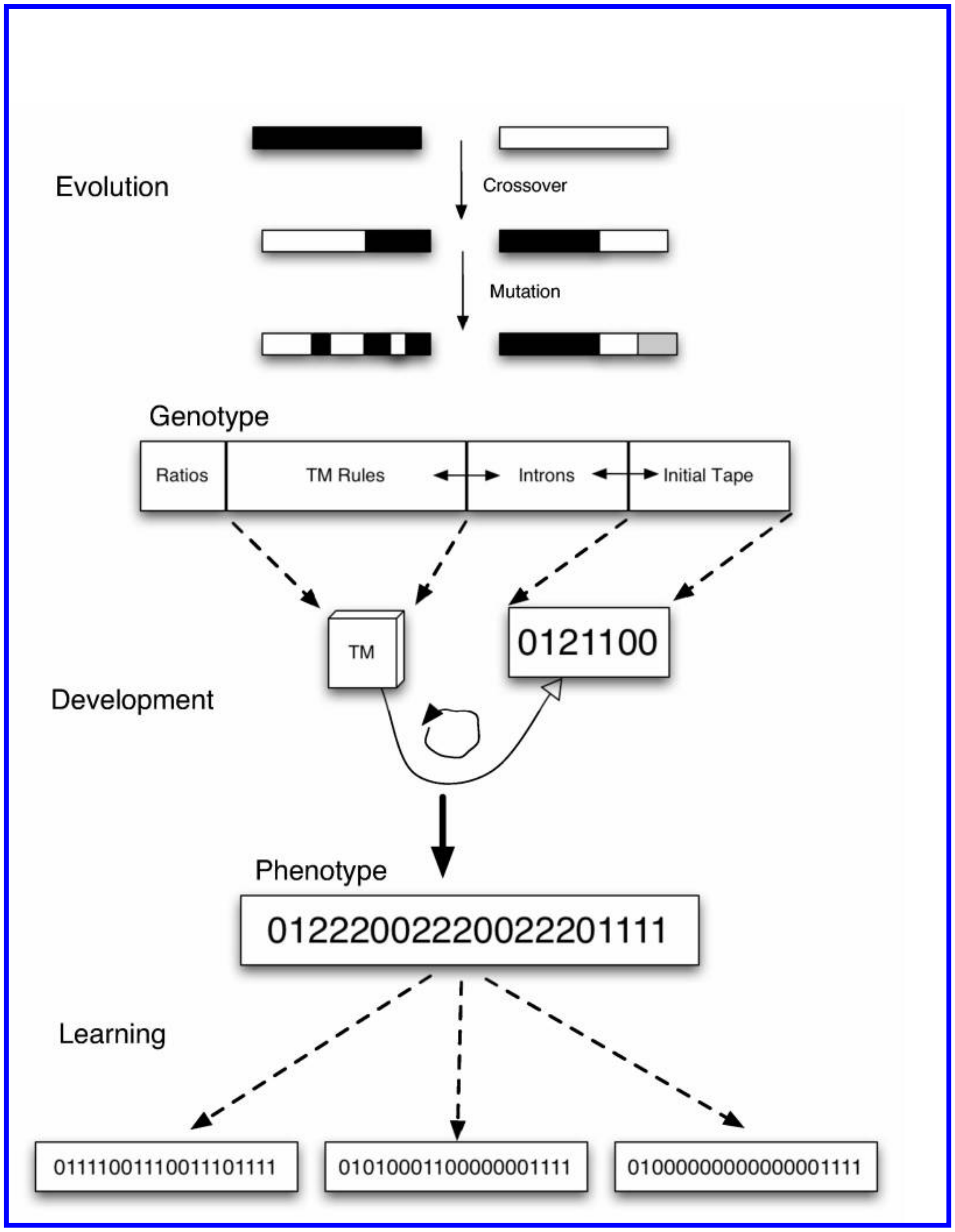

Figure 1. The three levels of adaptation in TRIDAP.

to encode a tape symbol and a TM state, respectively; the final bit encodes the action (overwrite or insert). For the examples in this article, $m = 5$ and $p = 3$. Hence, a TM has 8 (2^p) states, and each 5-tuple requires 17 bits.

3.2 The Developmental Process

The TRIDAP phenotype is a ternary string of length L or less. For this article, L is either 20, as in Hinton and Nowlan's experiments, or 40.

To grow the phenotype, TRIDAP decodes the genome into a TM and an initial tape. The TM 5-tuples are loaded into a hash table, with (s, x) as the key. If the initial tape is empty, a 0 is appended to it. The TM always begins the developmental process in state 0.

The TM then repeatedly runs from left to right across the tape in wraparound fashion. At each cell, the TM uses its current state and the tape symbol, (s, x), as an index into the hash table, which returns (s^*, x^*, a). In the absence of an entry for (s, x), the TM simply ignores the current symbol, remains in the same state, and moves one cell to the right. These movements, insertions, and overwrites continue until either the tape size reaches L or *max-devp-steps* Turing operations are performed. Since the TM always moves right one cell after each turn, the only variable action is whether to overwrite the current tape symbol or insert a new symbol to its immediate right. A newly inserted symbol is always the next one to be read.

If the initial tape already contains L cells, then, by convention, no development is performed, even though a series of overwriting actions could conceivably change the tape.

3.3 The Learning Process

The result of development is a string of 0's, 1's, and 2's, where the 2's denote wildcards. The values in the positions with wildcards can change via learning; all others are fixed.

TRIDAP has a target binary string that each phenotype tape tries to learn by randomly filling in its wildcards with 0's and 1's. If a phenotype has W wildcards, then one *guessing round* consists of randomly filling the W wildcards and then comparing the filled phenotype with the target.

If the filled phenotype perfectly matches the target, then learning halts and the number of necessary guessing rounds, *num-guesses*, is recorded. In this case, the number of matches, *num-hits*, is L. TRIDAP performs up to *max-guesses* rounds of guessing before giving up. It then returns *num-guesses* = *max-guesses* and *num-hits* as the number of matches in the best guessing round; *num-hits* $< L$.

Alternatively, TRIDAP can skip the guessing procedure and merely compute the probability, p_g, that the target string will be guessed within *max-guesses.* The calculation is straightforward: if corresponding locations on the target and tape are mismatching 0's and 1's, then the probability is 0. However, if every target bit either is matched directly or corresponds with a wildcard, then the probability is based on W, the total number of phenotypic wildcards:

$$p_g = 1 - (1 - 0.5^W)^{max\text{-}guesses} \qquad (1)$$

Here, $1 - 0.5^W$ is the probability of failing to match on a particular guessing round. Raising that to the *max-guesses* power yields the probability of failing to match on all guessing rounds. So the complement of that is the probability of matching on at least one of the *max-guesses* rounds.

3.4 The Fitness Functions

Two main fitness functions are used in the experiments reported below. The original Hinton-Nowlan function, f_{HN}, works as follows:

$$f_{\mathrm{HN}} = \begin{cases} 0 & \text{if } \mathit{num\text{-}bits} < L \\ L - 19^{\frac{\mathit{num\text{-}guesses}}{\mathit{max\text{-}guesses}}} & \text{otherwise} \end{cases} \tag{2}$$

This defines a very rugged fitness landscape in which only those phenotypes that can learn the target receive a nonzero evaluation. This score is inversely proportional to the duration of the guessing period, that is, the learning effort.

When the guessing probability p_g replaces the full guessing procedure, a probabilistic Hinton-Nowlan fitness $f_{p\mathrm{HN}}$ is more appropriate:

$$f_{p\mathrm{HN}} = Lp_g \tag{3}$$

The key difference between these two fitness functions involves chance. Consider a phenotype that is *correctly committed*: all of its non-wildcard symbols match the corresponding symbols of the target pattern. If its wildcard count, W, is large, then $f_{p\mathrm{HN}}$ will give it a low, but nonzero, fitness value. However, using f_{HN} and the full guessing procedure, the phenotype may *get lucky* by guessing the target early in the learning process, thereby garnering a high fitness value. At the other extreme, it may never guess the correct target.

Basically, $(1-p_g)\,\mathit{max\text{-}guesses}$ is the expected number of guesses required, so $f_{p\mathrm{HN}}$ is based on the average learning result for a phenotype, while f_{HN} leaves more to chance. Thus, $f_{p\mathrm{HN}}$ achieves a computational speedup by sacrificing the stochastic vagaries of the learning process. Essentially, then, f_{HN} treats each genome as an individual in a stochastic learning situation, while $f_{p\mathrm{HN}}$ treats each as a subpopulation or species by averaging over the individual uncertainty. On the larger (40-cell) tape problems and on batch runs, $f_{p\mathrm{HN}}$ was essential for producing results in reasonable time, given our computational constraints.

3.5 The Fitness Cases

Three general classes of target patterns are used: repetitive, semi-repetitive, and random. These were chosen to cover two extrema and a midpoint on the scale of difficulty for TRIDAP's developmental algorithm: repetitive patterns are usually easily generated by small sets of development rules, while random patterns are often their own shortest descriptions. On the other hand, for a fixed problem length, all three pattern types are equally difficult for non-developmental genetic algorithms.

The repetitive patterns involve adjacent copies of 1–4-bit patterns, such as 1111111... or 01010101... or 011001100110.... Similarly, the semi-repetitive patterns involve adjacent chunks of b bits, where the first c bits are the same across all chunks, while the remaining $b - c$ bits are randomly generated in each chunk. For example, the string 1010101110111010 is semi-repetitive with $b = 4$, $c = 3$, and the repetitive segment 101. In fully random patterns, in contrast, all bits are randomly generated.

3.6 Additional Genetic-Algorithm Details

The examples below use a population of 1,000 genomes, a mutation rate of 0.01 per bit, and a single-point crossover rate of 0.7, with no restrictions on the locations of the crossover points. Although standard fitness-proportionate selection was tried, it gave less interesting results due to early convergence. Hence, a tournament selec-

Table 1. Key parameters for the 20- and 40-bit test cases.

Target size	Population	Chromosome bits	Max. devp. steps	Max. guesses	Fitness function
20	1,000	300	100	1,000	f_{HN}
40	1,000	500	100	1,000,000	$f_{p\mathrm{HN}}$

tion mechanism with a tournament size of 4 is employed, along with single-individual elitism.

4 Blueprints versus Recipes

As discussed earlier, development confounds the Baldwin effect by greatly complicating the genotype-phenotype mapping, so that a learned change to the phenotype becomes nearly impossible to reverse-engineer into the genotype. So phase II of the Baldwin effect, genetic assimilation, can suffer under developmentally dominated mappings. Oddly enough, however, this does not preclude development from assisting a learning-driven evolutionary process. In fact, in some of the scenarios below, development is a necessary condition for both evolutionary progress and the Baldwin effect.

In the examples that follow, there is a clear interaction between two strategies: *blueprint* and *recipe*. The former involves a long initial tape and very little TM development, so the genome directly encodes the phenotype. Conversely, the recipe strategy employs development to grow the phenotype from a very short initial tape. Both strategies can exploit learning equally, since both can produce phenotypes with many wildcards.

To rephrase the problem of developmental Baldwinism in these terms: a recipe strategy has difficulty encoding learned patterns back into the recipe itself. A blueprint strategy has no such difficulty, since the phenotype and the initial tape of the genotype correlate so well. For blueprints, phase II of the Baldwin effect involves simply replacing some wildcards with the appropriate 0's and 1's. This evolutionary instantiation process can continue until the complete target pattern is encoded in the initial tape. Learning simply buys evolutionary time for the genome, while it gradually instantiates the wildcards. But whereas wildcards are relatively safe, wrong instantiations are fatal, given the fitness functions above. So instantiation proceeds very slowly, with many failed attempts going extinct.

Blueprints and recipes compete for dominance of the genome, with the test conditions often brokering the tradeoffs between winner-take-all and cooperative results. A brief tour through a series of repetitive, semi-repetitive, and random test cases, using two different target-pattern lengths (20 and 40), reveals the spectrum of these blueprint-recipe interactions and their influence upon the Baldwin effect. Table 1 summarizes the key differences between the 20- and 40-bit scenarios.

5 Results

As in Hinton and Nowlan's classic article [9], the results below are all typical runs of their respective scenarios, unless otherwise stated. This more clearly illustrates the Baldwin effect, particularly the often abrupt rises and falls in learning effort, than averages over multiple runs.

In all cases, the fitness landscape is so extensive and uninformative (one sharp peak rising from a flat plane) that evolution without learning reduces to a nearly fruitless random search that only rarely (3 of 50 attempts, as shown in case 20-fr-e of Table 2)

Table 2. Batch TRIDAP runs for 20-cell tapes. Each case was performed 50 times, with each run halting when either an individual with maximum fitness (of 20) arose or 50 generations had passed. A *recipe* solution is defined as one involving five or more developmental steps.

		Recipes			Blueprints	
Case	Target	Count	Generation	Steps	Count	Generation
20-2f	0101 . . .	42	15.0	26.8	8	21.1
20-fr	$0 \star 0 \star \ldots$	3	28.3	57.3	27	27.9
20-4f	0011 . . .	1	17.0	15.0	26	29.8
20-r	$\star\star\star\ldots$	1	25.0	8.0	30	29.0
20-fr-e	$0 \star 0 \star \ldots$	0	—	—	3	18.3

finds the target pattern in the 20-cell scenarios. The 40-cell scenarios are nearly impossible without learning, and no solutions (or even precursors to solutions) were found in the runs below. Conversely, with learning, the 20-cell scenarios often yield to either blueprints, recipes, or a cooperative combination within a few dozen generations. Hence, the general claim of the Baldwin effect—learning speeds evolution—is clearly evident in the examples below. The strength of presence of phases I and II varies across the scenarios, but the general trend is quite convincingly Baldwinian.

5.1 20-Cell Target Patterns

5.1.1 Repetitive Targets

First, consider a simple repetitive target pattern with chunk size 1—that is, a string of all 0's or all 1's. This is so easily generated by a TM running on a blank initial tape that neither blueprinting nor learning enters the picture. However, it often takes a 1000-individual TRIDAP population a few generations to find the combination of a blank (or nearly blank) initial tape and a TM with rules such as

$s_0, 0 \Rightarrow s_1, 1, \mathit{Overwrite}$

$s_1, 1 \Rightarrow s_1, 1, \mathit{Insert}$

These convert the default seed tape {0} into a string of 1's by first replacing the 0 with a 1 and moving into state 1. Moving right then moves the TM read head back to the start of the tape, which now consists of {1}. The second rule then tells the TM to insert a new 1 to the right of the current one and to remain in state 1. The read head then moves to the newly inserted 1 and performs rule 2 again, adding a third 1, moving to it, and so on. The recipe is even simpler if the initial tape is already {1}.

Generating 2-bit repetitive patterns, such as 10101010 . . ., is equally easy, with a typical developmental strategy containing the following core rules:

$s_0, 0 \Rightarrow s_1, 1, \mathit{Insert}$

$s_1, 1 \Rightarrow s_0, 0, \mathit{Insert}$

Adding more complexity, a 4-bit pattern such as 00110011 . . . requires a TM that can count to 2 in two ways (once for 0's and once for 1's). Finding such a recipe is nontrivial for TRIDAP, so a blueprint strategy normally wins the race (see case 20-4f in Table 2). These blueprints follow the typical evolutionary sequences described by Hinton and Nowlan: the best individuals of early generations contain many wildcards, and as the generations pass, these get filled by the correct binary values. Figure 2 shows the gradually improving phenotypes from a typical run.

⋆0 ⋆ ⋆00 ⋆ ⋆001 ⋆ ⋆011 ⋆ ⋆ ⋆ 1
00 ⋆ 100 ⋆ ⋆001 ⋆ ⋆011 ⋆ ⋆ ⋆ 1
00 ⋆ 1001 ⋆ 001 ⋆ ⋆011 ⋆ ⋆ ⋆ 1
00 ⋆ 1001 ⋆ ⋆011 ⋆ 011 ⋆ 011
00 ⋆ 100110011 ⋆ 011 ⋆ 011
0011001 ⋆ 0011 ⋆ 0110011
0011001 ⋆ 001100110011
00110011001100110011

Figure 2. A sequence of best-of-generation phenotypes from ascending, but non-contiguous generations for the 4-bit repetitive target 00110011 . . . . Wildcards are denoted by ⋆.

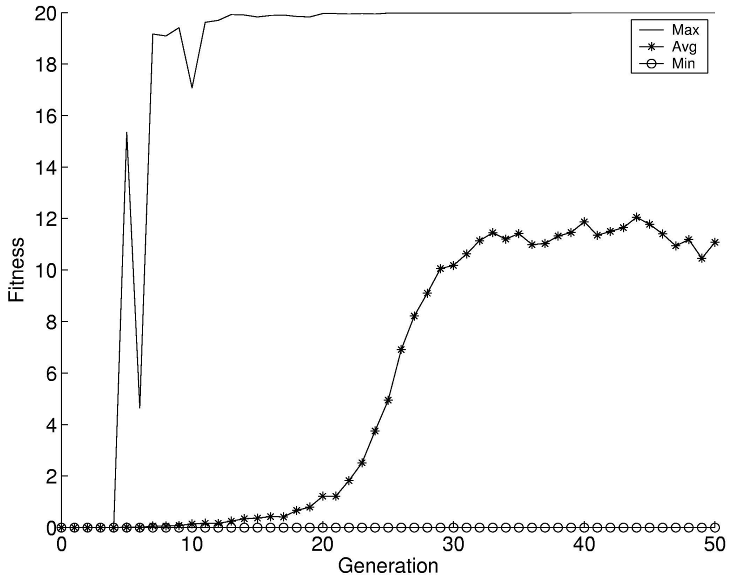

Figure 3. Fitness progression for the 20-bit repetitive target pattern 00110011 . . . .

This same run reveals a classic Baldwin effect, as shown in Figures 3 and 4. Fitness gradually increases, but with sporadic declines, since the early individuals employ excessive learning with its inherent stochasticity. For example, a phenotype with eight wildcards may correctly guess the target pattern in one generation, but an identical individual may fail to guess it in the next (or same) generation. As the blueprints evolve fewer wildcards and more correct bits, the good solutions are guaranteed to find the target, thus achieving a fitness value near the maximum of 20.

Figure 4 shows that learning, measured in terms of the population average of phenotypic wildcards, gradually increases, signifying phase I of the Baldwin effect. Phase II is then evident in the decline of learning as correct solutions become hardwired into the initial tapes of the genomes. Notice that development, quantified by the population average of TM actions, vanishes after 25 generations. Finally, the amount of genome used to encode the TM and initial tape rises slightly as development disappears.

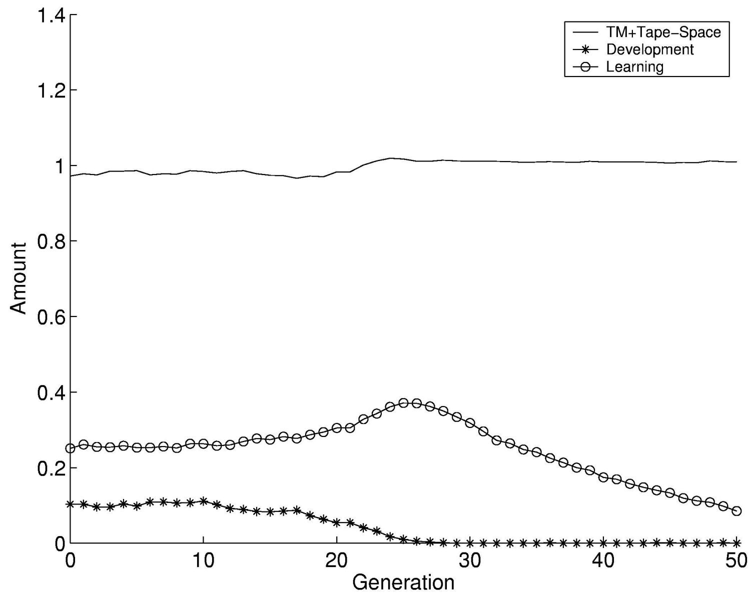

Figure 4. Evolving adaptive effort for the 20-bit repetitive target pattern 00110011 . . . .

In adaptive-effort graphs such as Figure 4, the fractions are:

1. *TM+tape space*: The sum of the number of TM rules and the number of symbols on the initial tape divided by the length of the target pattern.
2. *Development*: The number of developmental steps (TM actions) divided by the maximum allowable number of steps, 100.
3. *Learning*: The number of wildcards divided by the length of the target pattern.

5.1.2 Semi-repetitive Targets

In general, as repetitive patterns involve longer subpatterns, recipes get harder to evolve and blueprinting becomes the dominant strategy. Semi-repetitive patterns with short subpatterns offer a middle ground, where blueprints and recipes square off on approximately equal footing. Some runs are fully dominated by one or the other strategy, while others alternate highest-fitness strategy types from generation to generation. However, as shown in the 20-fr cases of Table 2, blueprints eventually win out in the vast majority of runs.

It is important to note that a solution in Table 2 is classified as a recipe if it involves five or more developmental steps, whereas a blueprint solution has four or less such steps. In most cases, blueprints have no developmental steps. Hence, many developmental solutions involve sizable initial genomes that hardwire many of the random bits; development then adds in semi-repetitive 0's, for example, to achieve the target match.

The sequence in Figure 5 of best-of-generation individuals illustrates a tight competition over the target pattern 01000101010001010101, which is semi-repetitive with a subpattern of $0X$, where X is a random number. Here, a simple recipe generates a 10-wildcard sequence that occasionally wins in a generation due to lucky guessing, that is, the target is found after only a few guessing rounds.

$$\begin{array}{lcl}
 & \star 100 \star 1 \star \star \star \star \star 0010 \star 0 \star 0 \star & \\
\emptyset & \Longrightarrow & 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star \\
 & 0 \star 00 \star 1 \star 10 \star 00010 \star \star \star 01 & \\
 & 0 \star 00 \star 1 \star \star 0 \star 000101 \star \star \star \star & \\
\emptyset & \Longrightarrow & 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star 0 \star \\
 & 0 \star 0001 \star 101000101 \star \star 0 \star & \\
 & 01000 \star \star \star 01000101 \star \star \star \star & \\
 & 01000101010001010101 &
\end{array}$$

Figure 5. A sequence of best-of-generation phenotypes from ascending, but non-contiguous generations for the two-bit semi-repetitive target 01000101010001010101. Wildcards are represented by $\star$. Arrows denote a developmental process from the initial tape (left) to phenotypic tape (right); phenotypes without arrows are pure blueprints.

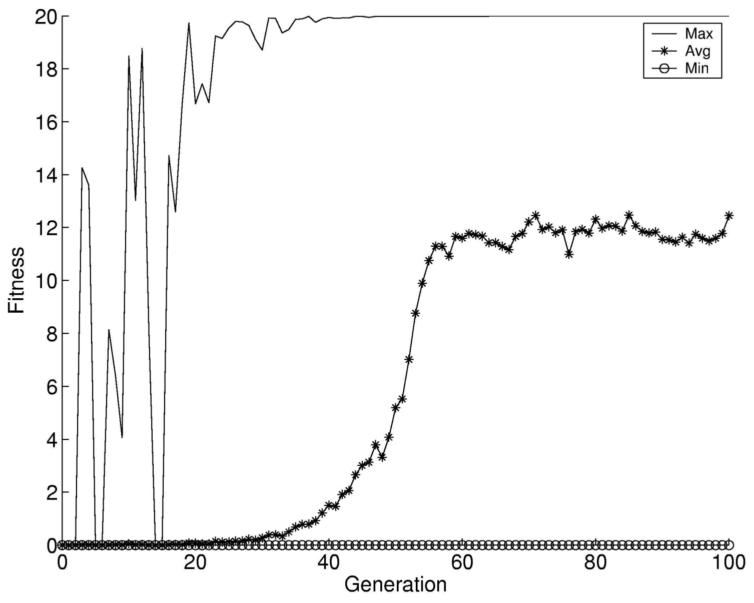

Figure 6. Fitness progression for a 20-bit semi-repetitive target pattern.

Figures 6 and 7 again reveal a classic Baldwin effect, but in contrast to the previous example, both blueprints and recipes remain viable until around generation 50, when development briefly rises but then suffers a sharp decline. This is also evident in the size of the active genotype, which dips down during development's short success but then returns to a high value once blueprints finally gain control. In the final 50 generations, learning decreases as the dominating blueprints assimilate more correct bits.

5.1.3 Random Targets

For the sake of brevity, no random 20-bit scenarios are shown, but, as illustrated by case 20-r in Table 2, these are almost always dominated by blueprints, as expected, since (a) compact rule sets are normally powerless to generate particular unstructured patterns on their own, and (b) the search space is small enough that unaided blueprinting can find solutions. With development largely out of the picture, the random-pattern cases show a classic Baldwin effect just as in Hinton and Nowlan's experiments.

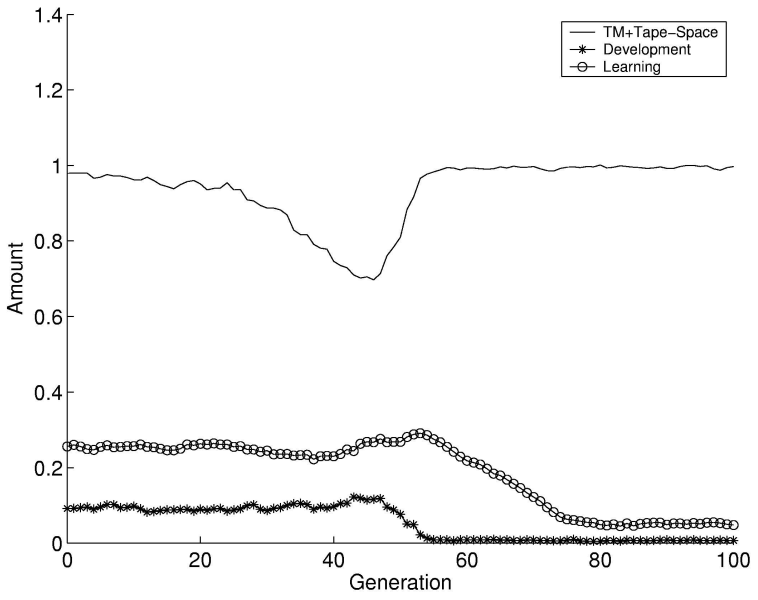

Figure 7. Evolving adaptive effort for a 20-bit semi-repetitive target pattern.

5.2 20-Cell Batch Runs

To add quantitative backing to the more qualitative descriptions above, a series of batch runs were performed to illustrate the interactions between recipes and blueprints. Table 2 summarizes the results for the batch tests on 20-cell patterns. The case names indicate the target length, the pattern, and the adaptive methods used. For example, 20-2f is a case involving a 20-cell target with a repeating pattern of 2 fixed (f) bits, while 20-fr involves one fixed and one random (r) bit in each 2-bit sequence. The first four cases involve the whole of TRIDAPs adaptive repertoire—evolution, development, and learning—while the fifth case, 20-fr-e, involves only evolution (e); it is therefore just a standard genetic algorithm evolving strings of 20 binary alleles.

Each case involves 50 randomly seeded runs, where each run halts after 50 generations, or earlier if an individual achieves the maximum fitness of 20. To make the batch runs tractable, the probabilistic fitness function, $f_{p\mathrm{HN}}$, was used for all cases. The 50 runs are sorted into three groups: (a) those in which no optimal individuals were found after 50 generations, (b) those in which an optimal individual was found and its genome included a significant amount of development, defined here as five or more developmental steps (counting the number of developmental rules is misleading, since many, sometimes all, are never used), and (c) runs in which the first optimal individual is primarily a blueprint (less than five developmental steps). Only the latter two categories are included in Table 2, where the number of runs in that category is listed, along with the average generation at which the optimal individual arose. For the recipe-dominated runs, the average number of developmental steps of the first optimal individual is also listed.

The results clearly indicate the dominance of recipes on the simple 2-bit repetitive target, while blueprints take over as the patterns get more complex. However, these results fail to capture the competition between recipes and blueprints. In a separate set of 50-run batch tests of the first four cases, the runs were halted as soon as an individual reached a fitness value of 10 (i.e., half the maximum). In those cases (not all results are shown), recipes were the halfway winners in 13 of 50 20-fr runs. This

$$0 \star 1 \star 011 \longrightarrow 0 \star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star 1 \star 011$$
$$0 \star 1 \star 011 \longrightarrow 0 \star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star\star 1 \star 011$$
$$\emptyset \longrightarrow 00\star\star 00\star\star 00\star\star 00\star\star 00\star\star 00\star\star 00\star\star 00\star\star 00\star\star 00\star\star$$
$$\emptyset \longrightarrow 00110011001100110011001100110011001100\star\star$$

Figure 8. A sequence of best-of-generation phenotypes for the 40-cell repetitive target 00110011 . . . .

matches the expectation that blueprints and recipes compete on relatively even footing on 2-bit semi-repetitive patterns, at least initially. Recipes also were halfway winners in four of the 20-4f cases and five of the 20-r cases. So, clearly, the two strategies do interact before blueprints take over. In the next section, we find that this dominance does not scale up.

Finally, the standard genetic algorithm, case 20-fr-e, finds the needle in the haystack three times out of 50 attempts. This is somewhat expected given the evolutionary parameters: 50 generations and 1,000 individuals gives a (very optimistic) maximum of 50,000 individuals tested per run, in a search space of 1 million points. The poor performance of pure evolution, compared with the successes of the other four cases, is the standard illustration that the Baldwin effect can indeed speed the course of evolution, particularly in rugged search spaces.

5.3 40-Cell Target Patterns

For these experiments, the chromosome length is increased from 300 bits (for the 20-cell targets) to 500 bits, giving ample space for a 40-cell initial tape on the genome and/or more TM rules. Also, the maximum number of learning trials (i.e., guesses) increases from 1,000 to 1,000,000. However, the population size remains the same, 1,000, as does the maximum number of developmental steps. The latter was not scaled up, because (a) 100 is sufficient for both 20- and 40-cell cases, and (b) the larger structured targets provide a major advantage to recipes over blueprints, and we did not want to give development any additional help. Due to the larger search space and the 1 million learning trials, the probabilistic fitness function, $f_{p\mathrm{HN}}$, was employed to make these tests computationally feasible.

In general, the search space has increased one-million-fold from the 20-cell case. Under these conditions, both blueprinting and development clearly have an uphill battle, even with learning and its landscape-smoothing effects. Note that to scale properly from the 20-cell to the 40-cell case, one would need a population of one billion, along with one billion learning trials. Failing to scale properly clearly diminishes the odds of success for semi-repetitive and random scenarios.

5.3.1 Repetitive Targets

Of course, for 1-bit and 2-bit repetitive sequences, development quickly finds perfect solutions. However, a 4-bit repeater such as 00110011 . . . poses a tougher challenge. The typical run needs a few dozen generations to converge to a solution such as

$$\emptyset \longrightarrow 001\star 001\star 001\star 001\star 001\star 001\star 001\star 001\star 001\star 001\star \tag{4}$$

Despite the 10 wildcards, this is essentially perfect, since the probability of guessing the correct 10 bits in the one million learning trials approaches 1.0.

Occasionally, the genetic assimilation is nearly complete, as in the run of Figure 8. This also gives a dramatic Baldwin effect, as depicted in Figures 9 and 10.

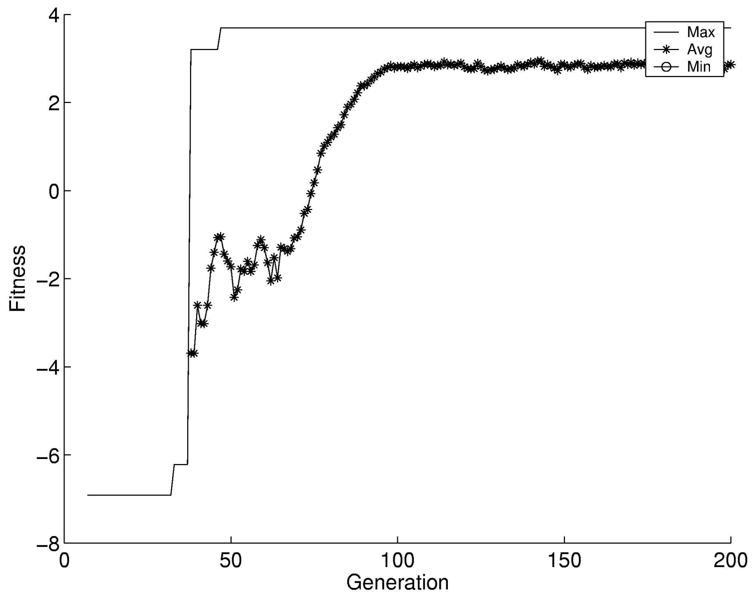

Figure 9. Fitness progression for the 40-cell repetitive target 00110011 . . . . The natural logarithm of fitness is plotted to clearly show the full range of evolutionary progress. The minimum-fitness line vanishes, since it is all zeros.

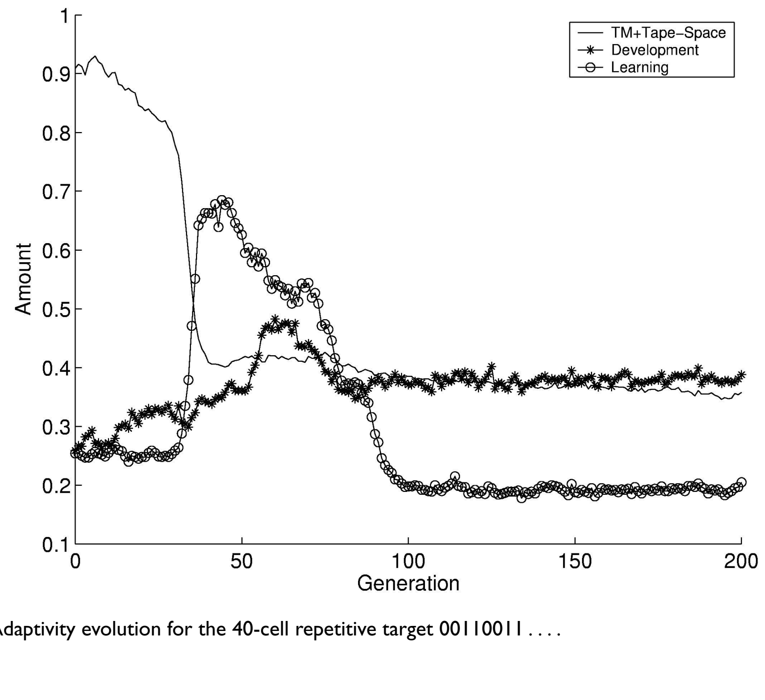

Figure 10. Adaptivity evolution for the 40-cell repetitive target 00110011 . . . .

$$\begin{aligned}
0\star &\longrightarrow 0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star \\
0010101 &\longrightarrow 0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star010101 \\
0010101 &\longrightarrow 010\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star0\star010101 \\
\emptyset &\longrightarrow 01010\star01010\star0\star01010\star0\star01010\star01010\star0\star0\star0\star
\end{aligned}$$

Figure 11. A sequence of best-of-generation phenotypes for the 40-cell semi-repetitive target 0101010101000001010001010100010101010101.

The final, nearly perfect solution in Figure 8 is generated by the following six developmental rules:

$s_0, 0 \Rightarrow s_2, 2,$ *Insert*
$s_2, 2 \Rightarrow s_3, 2,$ *Insert*
$s_3, 0 \Rightarrow s_1, 0,$ *Insert*
$s_1, 0 \Rightarrow s_2, 1,$ *Insert*
$s_2, 1 \Rightarrow s_1, 1,$ *Insert*
$s_1, 1 \Rightarrow s_3, 0,$ *Insert*

5.3.2 Semi-repetitive Targets

Moving to a semi-repetitive pattern $0X0X\ldots$, where X is a random bit, development also manages to find rather complex solutions by first generating highly plastic phenotypes (i.e., many wildcards) and gradually hardwiring more bits. For the scenario summarized in Figure 11, development is the dominant strategy. Blueprinting plays an intermediate role but eventually gives way to an interesting recipe with only four effective rules:

$s_0, 0 \Rightarrow s_1, 1,$ *Insert*
$s_1, 1 \Rightarrow s_1, 0,$ *Insert*
$s_1, 0 \Rightarrow s_2, 2,$ *Insert*
$s_2, 1 \Rightarrow s_0, 0,$ *Insert*

In analyzing these, note that when no TM rule exists for a state-symbol pair, such as $(s_2, 2)$ in the above situation, the read head simply moves right while maintaining the same state. Also, to generate the final phenotype of Figure 11, the read head wraps around the growing tape several times.

Figures 12 and 13 again show the Baldwin effect. In Figure 12, the improvements in maximum fitness after generation 50 are such small fractions that the increases are not detectable on the fitness graph. The final phenotype of Figure 11 does not arise until generation 319. In Figure 13, note the dramatic early spike in learning near generation 30 and then the gradual decline to under 20%. Other runs on these 40-cell semi-repetitive patterns show a mixed strategy, but only very rarely does a pure blueprint emerge. In all cases, the Baldwin effect is clearly evident, with learning playing a significant role early in evolution, where 50–70% of the bits are wildcards, and then gradually receding to 20–30%. With one million learning trials, percentages below 25 (i.e., 10 bits) give no significant fitness advantage under the probabilistic Hinton-Nowlan fitness function $f_{p\text{HN}}$.

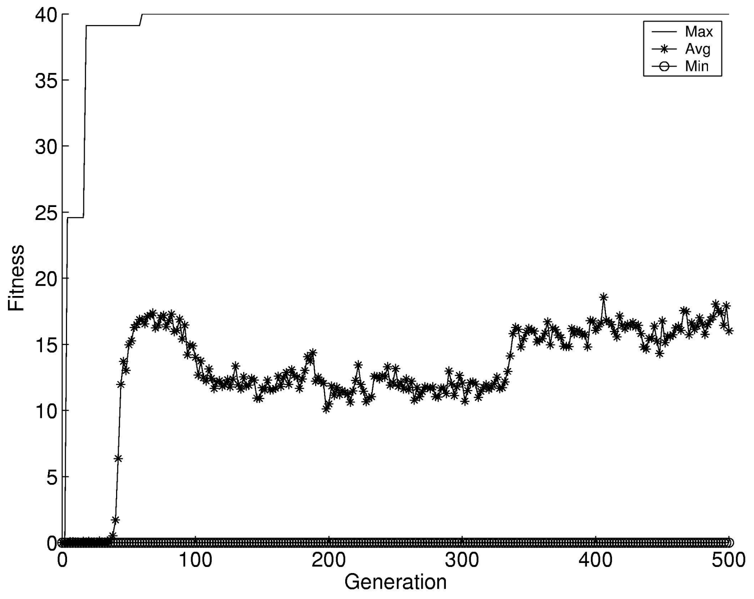

Figure 12. Evolutionary progression for the 40-cell semi-repetitive target 0101010101000001010001010100010101010101.

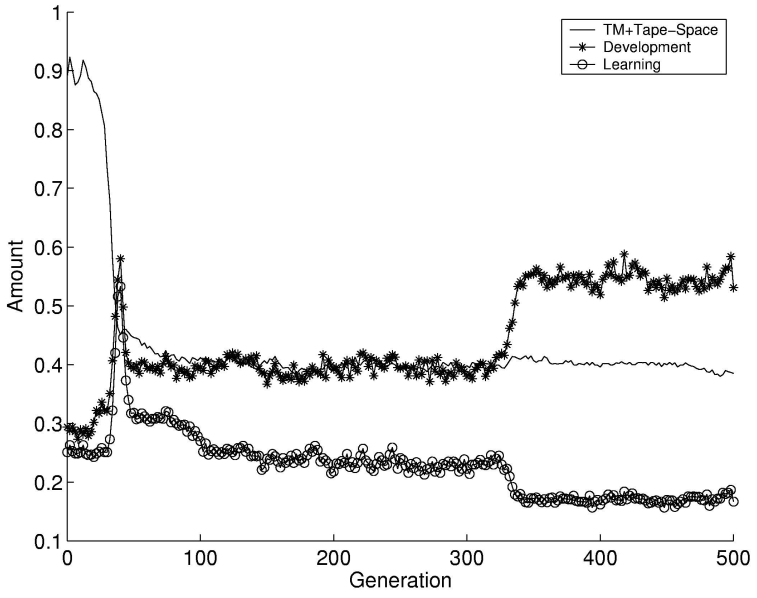

Figure 13. Adaptivity evolution for the 40-cell semi-repetitive target 0101010101000001010001010100010101010101.

⋆1⋆⋆⋆0 ⟶ 11⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆0

⋆⋆001⋆ ⟶ 1⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆001⋆

⋆0⋆010 ⟶ ⋆10⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆010

⋆100⋆0 ⟶ 1100⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆10

00⋆11⋆000⋆0 ⟶ 11⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆11⋆000⋆0

1⋆⋆⋆0⋆0⋆1⋆00010 ⟶ 1⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆0⋆0⋆1⋆00010

⋆00001⋆⋆00010 ⟶ ⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆00001⋆⋆00010

11⋆0011⋆00010 ⟶ 11⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆0011⋆00010

⋆000011000010 ⟶ ⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆000011000010

⋆000011000010 ⟶ ⋆1⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆0000011000010

⋆000011000010 ⟶ ⋆10010⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆0011000010

⋆000011000010 ⟶ ⋆10010⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆00011000010

11001001101⋆000011000010 ⟶ 11001001101⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆000011000010

11001001101⋆000011000010 ⟶ 11001001101⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆⋆0000011000010

Figure 14. A sequence of best-of-generation phenotypes from ascending, but non-contiguous, generations for the 40-cell random target 1100100110100010111011010100000011000010. Wildcards are represented by ⋆, and arrows denote a developmental process from the initial TM tape on the left to the phenotypic tape on the right.

5.3.3 Random Targets

Finally, consider a random 40-cell pattern:

$$1100100110100010111011010100000011000010 \qquad (5)$$

Intuitively, neither blueprinting nor development should have a chance of finding this pattern. However, the combination can often get close. As shown in Figure 14, the early solutions convert a simple initial tape into a large wildcard string. Although far from the target, these have a nonzero probability of learning the target, so f_{pHN} gives them a small positive fitness value—just enough to bias evolution in the proper direction. Then, over the course of a few hundred generations, the initial tape fills up with non-wildcard cells while maintaining at least one wildcard, which serves as the seed for the TM to grow the central wildcard region. Figures 15 and 16 show another classic Baldwinian progression as learning initially aids search but then gradually abates as more bits become assimilated. The probabilistic fitness function f_{pHN} gives nearly perfect fitness to a correctly committed individual with 16 wildcards or less (since there are one million learning trials), so progress to perfect genetic assimilation is impeded by the mastery effect.

This same general evolutionary sequence occurs frequently in other 40-cell random tests, although the blueprint often fills from just one side or the other, and not both simultaneously. Still, the number of wildcards rarely goes below 18. Using alternative fitness functions that penalize for the number of phenotypic wildcards provides little help and often hinders early evolution.

5.4 40-Cell Batch Runs

The batch cases of Table 2 were repeated with 40-cell tapes; the results appear in Table 3. Again, 50 runs of each case were performed, but each run lasted a maximum of 100 generations (instead of 50). Also, runs were halted whenever an individual with fitness 39 arose, since perfect individuals (fitness 40) were nearly impossible to find for all cases other than 40-2f and 40-4f. A fitness of 39.0 or higher implies a phenotype

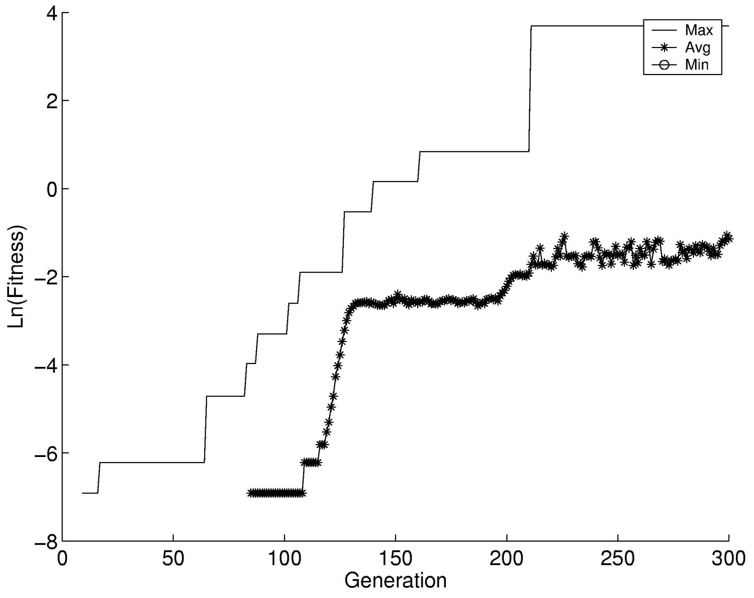

Figure 15. Fitness progression for the 40-cell random target pattern. The natural logarithm of fitness is plotted to clearly show the full range of evolutionary progress. The minimum-fitness line has disappeared, since it is all zeros.

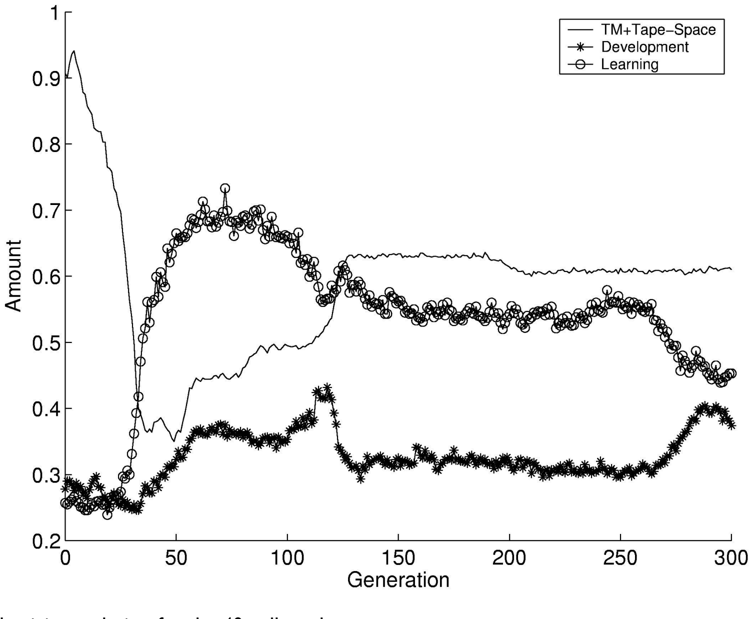

Figure 16. Adaptivity evolution for the 40-cell random target pattern.

Table 3. Batch TRIDAP runs for 40-cell tapes. Each case was performed 50 times, with each run halting when either an individual with nearly maximum fitness (of 39) arose or 100 generations had passed. A *recipe* solution is defined as one involving 10 or more developmental steps.

		Recipes			Blueprints	
Case	Target	Count	Generation	Steps	Count	Generation
40-2f	0101 . . .	50	13.0	43.3	0	—
40-fr	$0 \star 0 \star \ldots$	46	38.6	41.8	0	—
40-4f	0011 . . .	41	49.8	41.5	0	—
40-r	$\star\star\star \ldots$	0	—	—	0	—
40-fr-e	$0 \star 0 \star \ldots$	0	—	—	0	—
40-fr-el	$0 \star 0 \star \ldots$	0	—	—	0	—

having at least 22 correct bits, with the remaining 18 or less being wildcards. Given the condition of one million learning guesses and $f_{p\mathrm{HN}}$, such a phenotype is nearly optimal.

Impressively, developmental strategies find nearly optimal strategies within 100 generations in 46 of the 50 40-fr runs, and 41 of the 40-4f runs. However, note that no near-optimal recipes are found for the 40-r case. To further investigate this failure, an additional set of 25 40-r runs were performed, but this time over 300 generations (instead of 100). Although only one of these runs achieves a fitness of 39, 22 of the 25 runs show the distinct scaffolding effect of development in combination with blueprinting. That is, the genomes involved initial tapes of length 5–30 bits, which then were filled out by simple patterns, such as $\star\star\star \ldots$, during development. In the remaining three runs, the developmental filling was more distributed, and included expanding certain wildcards (dependent upon their context) into wildcard pairs or triples.

It is not that surprising that no pure blueprint winners were found in any of the runs. Even with developmental assistance during the early generations, it is extremely difficult to hardwire all 40 bits to the proper values. On the other hand, it is important to remember that all semi-repetitive and random solutions involve a cooperative strategy of developmental growth and (often considerable) hardwiring via the initial tape.

In comparing Tables 2 and 3, the 4-bit repetitive and 2-bit semi-repetitive cases (20/40-4f and 20/40-fr) are particularly interesting. In the 40-bit cases, note that good recipes are found, on average, in 38.6 and 49.8 generations. Since the 20-bit patterns were run for only 50 generations, it seems reasonable that development probably found many good solutions in the 20-4f and 20-fr runs as well. However, recipes simply lost in competition with blueprints. On the larger problem, the competition is greatly reduced, if not absent entirely.

To further illustrate TRIDAP's capabilities on the 40-cell tape, a series of 50 tests of 100 generations each, with population size of 1000, was run using the 8-bit repetitive pattern 10100011 . . . . In all, five runs produced an individual with fitness of 39 or better. Figure 17 shows four of the winning strategies; the final strategy was found in two different runs.

The final two cases in Table 3 involve a standard genetic algorithm. In case 40-fr-e, the GA attempts to directly evolve a 40-bit binary pattern. Complete failure is expected, given the immense size of the search space. To illustrate the nearly insurmountable nature of the task, even with learning but without development, a sixth case, 40-fp-el, was run. In this, TRIDAP evolves three-symbol tapes, as usual, but no development is permitted. Also, a run halts whenever an individual achieves a *nonzero* fitness value. So TRIDAP searches for an individual in which all bits are either correct matches to the target or wildcards, but without the help of development. As shown, 50 runs of 1,000

$$\begin{aligned}
111 &\longrightarrow 101\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star011 \\
\star1101\star0\star\star1101000111010001 1 &\longrightarrow \star\star\star\star\star\star\star\star\star\star\star\star\star\star\star1101\star0\star\star11010001110100011 \\
1010\star &\longrightarrow 1010\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star \\
1011 &\longrightarrow 101\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star01\star\star011
\end{aligned}$$

Figure 17. Four different strategies found by TRIDAP that earned a fitness of 39 or better on the 40-cell 8-bit repetitive target 10100011 . . . .

individuals over 100 generations produce nothing. Development is obviously crucial for the 40-bit scenarios.

6 Discussion

The previous examples illustrate both the competitive and cooperative interactions between blueprints and recipes during the evolutionary search process. While simple repetitive patterns are more easily found by developmental schemes, a small increase in subpattern complexity opens the door for blueprinting. Note that within a particular size category (e.g., 20 or 40 cells) all patterns are equally easy or difficult to generate via blueprinting. So as subpattern complexity increases in repetitive targets, blueprinting takes over only because development has greater difficulty designing string generators. However, as the comparison between cases 20-4f and 40-4f indicate, development is quite capable of generating the more complex patterns; it is merely beaten by pure blueprinting on the smaller 20-bit problems.

Perhaps the most interesting results are those of the random 40-cell patterns, where neither strategy has any clear competitive advantages. However, development gives evolution a small, but very significant, start, and cooperation with blueprinting then finds good results, with respect to the fitness function and the large number of learning trials.

Basically, development provides scaffolding for the progressive expansion of a blueprint. Without these simple repetitive recipes, evolution simply cannot guess enough correct bits. In this needle-in-the-haystack landscape, nothing encourages a blueprint strategy to add more wildcards, since, with extremely high probability, all pure blueprints will receive zero fitness. Blueprinting has the ominous task of designing a 40-cell pattern with either correct bits or wildcards in each cell, but it gets no partial information that wildcards are useful until one correctly committed individual arises. The probability of producing such an individual by randomly choosing each bit is

$$\left(\frac{2}{3}\right)^{40} \approx 0.00000006 \tag{6}$$

Thus, on average, a genetic algorithm with 1,000 individuals would need tens of thousands of generations to find a pure blueprint with nonzero fitness. This is the same small, but vital, hint that development can provide almost immediately.

It is critical to note that Hinton and Nowlan used a biased genotype generator that chose wildcards with 50% probability. This skews the above odds to a much more blueprint-friendly value:

$$\left(\frac{3}{4}\right)^{40} \approx 0.00001 \tag{7}$$

However, it is difficult to justify any general hardwired bias toward learning, especially given the (often extreme) costs of plasticity. Artificial evolution should be unfettered in tuning the balance between innate and acquired skills.

As for the computational issues involved in the decision to use development and/or learning with an evolutionary algorithm (EA), this work provides no conclusive answers, since (a) the problem domain is so abstract, and (b) the learning and development algorithms are so simple. Although there were no significant run-time differences between the TRIDAP and pure-GA runs described above, speed comparisons have little general utility in this context, since TRIDAP's learning is essentially free in comparison with sophisticated learning algorithms or even general local-search methods: it is only done implicitly by the probabilistic fitness function, $f_{p\mathrm{HN}}$. Similarly, the Turing-machine developmental mechanism is trivial compared to the more common developmental algorithms used with EAs [27]. Still, it is worth noting that the pure GA, and even the GA with learning but not development, was hopelessly lost in these difficult search spaces. There is little doubt that the combination of learning and development can benefit EAs in certain domains.

But what of our original skepticism toward development and the Baldwin effect? Have the simple abstract models above overturned our deepest suspicions? Unfortunately, no. Essentially, our results do nothing to dispute the claim that learned traits are extremely difficult to reverse-engineer into a recipe-type genome. But then, it would require quite a leap of faith to believe that any fixed, largely deterministic generative process could map all m-bit genomes into all n-bit phenotypes (where $n \gg m$).

However, this implies neither that (a) development prevents the Baldwin effect, nor that (b) recipes cannot compete with blueprints in difficult fitness landscapes where the Baldwin effect is often critical to search success, nor that (c) recipes cannot cooperate with blueprints to enhance Baldwinian search. Also, note that the Baldwin effect does not require complete genetic assimilation in phase II. Ideally, only the traits that correspond to static elements of the environment should become fixed, while those that are involved with dynamic aspects will remain plastic. Thus, many acquired traits will not and should not map to unique genes, but only to genes that facilitate one or another form of plasticity.

Although based on a simple model, the example scenarios above do show various relationships between development and Baldwinism in tough, needle-in-a-haystack search spaces. First, when the target phenotypes are highly structured, development can short-circuit the Baldwin effect by simply generating the proper phenotypes without involving learning at all. This effect is largely independent of the target length. Conversely, blueprints will, on average, show a similar Baldwin effect on all patterns, structured or random, but scaling to larger targets may block the effect completely, since the advantages of learning are never discovered in phase I.

Second, when the target phenotypes are partially structured or fully structured but with complex subpatterns, development often competes with blueprinting. It is conceivable, although not proven in our studies, that this competition puts added selection pressure on blueprints to hardwire their bits as quickly as possible, thus accelerating phase II. In other cases, development creates a structured plastic genome that gets close enough to fend off blueprint challengers. Here, phase I of the Baldwin effect begins abruptly but never gives way to phase II, since (a) evolution cannot invent complex enough generators, and (b) blueprinting capabilities largely disappear from the population.

Finally, when development and blueprinting cooperate, recipes often dominate in the earlier generations by laying out patterns of plasticity that gradually assimilate more bits as blueprinting (and thus longer initial tapes) enter the picture. This facilitates a complete Baldwin effect in domains that are either too large for blueprinting alone or too intricate for development alone.

Since living organisms are complex in terms of both component cardinality and intricacy, it seems fair to generalize from this third situation and speculate that development in the biological world could indeed enhance the Baldwin effect via a cooperative arrangement between genes that control general wide-scale properties of embryogenesis and those that have a more direct link to spatially localized phenotypic traits. In evolutionary computation, this cooperation might also be exploitable in problem domains where (a) solutions are complex but house intermittent structure, and (b) a hybrid recipe-blueprint genome is feasible, with evolution governing the relative contributions.

7 Conclusion

As seminal works often do, Hinton and Nowlan's classic article [9] creates plenty of heat to complement its light. One poignant source of controversy involves development and its potential incorporation within their framework:

> We have focused on the interaction between evolution and learning, but the same combinatorial argument can be applied to the interaction between evolution and development.... There is a selective pressure for genes which facilitate the development of certain useful characteristics in response to the environment. In the limit, the developmental process becomes *canalized*: The same characteristic will tend to develop regardless of the environmental factors that originally controlled. Environmental control of the process is supplanted by internal genetic control. [9, p. 500]

Granted, learning and development both involve adaptations to the environment (as development is also a flexible process governed by both genes and exogeneous physical and chemical factors). Hence, both map nicely to the abstract framework of Hinton and Nowlan's model. However, the interchangeable or parallel role of these processes must not overshadow their strongly serial nature: the brunt of development (i.e., everything except the final growth-rejuvenation stage) occurs prior to learning. For the most part, the brain needs to be assembled before it can begin learning. This serialism more clearly illustrates the gauntlet of transformations from gene to trait and the havoc this plays with the genotype-phenotype mapping, and hence upon the potential for canalization (i.e., phase II of the Baldwin effect).

Clearly, developmental Baldwinism is not simply an alternative instantiation of learning Baldwinism. Development exacerbates the whole problem, as also expressed by Mitchell and Belew [18]:

> In Hinton and Nowlan's model, the specification of a genetic trait and the guessing of it by a cognitive phenotype are directly interchangeable. This obviates the need for, indeed the possibility of, the complex biological processes of *development* by which genotype is transformed into phenotype. The entire model may well, therefore, turn out to be somewhat irrelevant to the real relation between learning at the individual level and evolution at the species level. [18, p. 445]

Although *irrelevant* seems somewhat harsh, the original model clearly abstracts away an essential natural process. However, it does appear to capture the basic essence of the Baldwin effect and is worthy of further expansion and investigation. Our extension allows very abstract notions of development and learning to serially interact, with the strength of each contribution controlled by evolution. The results do not solve any mysteries as to the indirect transfer of phenotypic to genotypic change, backward across a complex developmental process. But they do illustrate several roles that development can play in Baldwinian evolution, both inhibiting and enhancing.

The natural world is so much more complex than our models that one can often doubt the utility of this whole endeavor. However, as every student of physics 101 knows, we would flounder aimlessly without science's friendly abstractions. Nowhere is this more true than in studies of the Baldwin effect. Consider the trail from DNA to blastula to gastrula to neural tube to cerebrum to behavior to synaptic modification and learning...and then back again! Detailed coherent models of this entire sequence are far from realizable, although the pieces are falling together at an ever-increasing rate. Furthermore, lab or field studies of evolving and learning organisms are extremely demanding, since organisms with significant learning capacities tend to have longer life spans and older maturation ages. So although the results of ALife simulations cannot undeniably confirm or refute Baldwin's hypothesis, they offer the only realistic hope, at least in the short term, of making serious empirical progress on an issue that has not advanced beyond the stage of *plausible phenomena* for over a century. Like developmental recipes on long target strings, ALife could provide the essential scaffolding for this search.

References

1. Ackley, D. H., & Littman, M. L. (1992). Interactions between learning and evolution. in C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen (Eds.), *Artificial Life II* (pp. 487–509). Reading, MA: Addison-Wesley.
2. Bala, J., DeJong, K., Huang, J., Vafaie, H., & Wechsler, H. (1996) Using learning to facilitate the evolution of features for recognizing visual concepts. *Evolutionary Computation, 4*, 297–311.
3. Baldwin, J. M. (1896). A new factor in evolution. *The American Naturalist, 30*, 441–451.
4. Cangelosi, A. (1999). Modeling the evolution of communication: From stimulus associations to grounded symbolic associations. In J. N. D. Floreano and F. Mondada (Eds.), *ECAL99* (pp. 654–663). Berlin: Springer-Verlag.
5. Chomsky, N. (1998). *On language: Chomsky's classic works 'Language and responsibility' and 'Reflections on language' in one volume.* New York: New Press.
6. Downing, K. L. (2001). Reinforced genetic programming. *Genetic Programming and Evolvable Machines, 2*, 259–288.
7. Gruau, F. (1994). Genetic micro programming of neural networks. In P. J. Angeline and K. E. Kinnear, Jr. (Eds.) *Advances in Genetic Programming* (chapter 24, pp. 495–518). Cambridge, MA: MIT Press.
8. Gruau, F., & Whitley, D. (1993). Adding learning to the cellular development of neural networks. *Evolutionary Computation, 1*, 213–233.
9. Hinton, G. E., & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems, 1*, 495–502.
10. Holland, J. H. (1992). *Adaptation in natural and artificial systems* (2nd ed.). Cambridge, MA: MIT Press.
11. Houck, C. R., Joines, J. A., Kay, M. G., & Wilson, J. R. (1997). Empirical investigation of the benefits of partial Lamarckianism. *Evolutionary Computation, 5*, 31–60.
12. Iba, H. (1998). Multi-agent reinforcement learning with genetic programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg,

H. Iba, & R. Riolo (Eds.), *Genetic Programming 1998: Proceedings of the Third Annual Conference* (pp. 167–172). San Francisco: Morgan Kaufmann.

13. Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system, *Complex Systems, 4*, 461–476.
14. Koza, J. R. (1992). *Genetic programming: On the programming of computers by natural selection*. Cambridge, MA: MIT Press.
15. Koza, J. R., Andre, D., Bennett, F. H., III, & Keane, M. (1999). *Genetic programming 3: Darwinian invention and problem solving*. San Francisco: Morgan Kaufman.
16. Lamarck, J. B. (1914). Of the influence of the environment on the activities and habits of animals, and the influence of the activities and habits of these living bodies in modifying their organization and structure. In *Zoological Philosophy* (pp. 106–127). Translated by M. Elliott. London: Macmillan.
17. Mayley, G. (1996). Landscapes, learning costs and genetic assimilation: Modeling the evolution of motivation. *Evolutionary Computation, 4*(3), 213–234.
18. Mitchell, M., & Belew, R. (1996). Preface to Chapter 25. In R. Belew & M. Mitchell (Eds.), *Adaptive individuals in evolving populations: Models and algorithms* (pp. 443–445). Reading, MA: Addison-Wesley.
19. Montana, D. J., & Davis, L. D. (1989). Training feedforward networks using genetic algorithms. In *Proceedings the Eleventh International Joint Conference on Artificial Intelligence* (pp. 762–767).
20. Moriarty, D. E., & Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation, 5*, 373–399.
21. Munroe, S., & Cangelosi, A. (2002). Learning and the evolution of language: The role of cultural variation and learning costs in the Baldwin effect. *Artificial Life, 8*, 311–339.
22. Nolfi, S., & Floreano, D. (2000). *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. Cambridge, MA: MIT Press.
23. Pinker, S., & Bloom, P. (1990). Natural language and natural selection. *Behavioral and Brain Sciences, 13*, 707–784.
24. Riolo, R. L. (1991). Lookahead planning and latent learning in a classifier system. In J.-A. Meyer & S. W. Wilson (Eds.), *Proceedings of the First International Conference on Simulation of Adaptive Behavior: From animals to animats* (pp. 316–326). Cambridge, MA: MIT Press.
25. Sims, K. (1994). Evolving 3D morphology and behavior by competition. In R. Brooks & P. Maes (Eds.), *Artificial Life IV* (pp. 28–39). Cambridge, MA: MIT Press.
26. Stanley, K., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*, 99–127.
27. Stanley, K., & Miikkulainen, R. (2003). A taxonomy for artificial embryogeny. *Artificial Life, 9*, 93–130.
28. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.
29. Turney, P., Whitley, L. D., & Anderson, R. W. (1997). Introduction to the special issue: Evolution, learning, and instinct: 100 years of the Baldwin effect. *Evolutionary Computation, 4*, iv–viii.
30. Wolpert, L. (2002). *Principles of development*. New York: Oxford University Press.
31. Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE, 87*, 1423–1447.