massachusetts institute of technology — artificial intelligence laboratory

Implementing Universal Computation in an Evolutionary System

Justin Werfel

AI Memo 2002-010 July 2002

Abstract

Evolutionary algorithms are a common tool in engineering and in the study of natural evolution. Here we take their use in a new direction by showing how they can be made to implement a universal computer. We consider populations of individuals with genes whose values are the variables of interest. By allowing them to interact with one another in a specified environment with limited resources, we demonstrate the ability to construct any arbitrary logic circuit. We explore models based on the limits of small and large populations, and show examples of such a system in action, implementing a simple logic circuit.

This research was supported by a National Defense Science and Engineering Graduate Fellowship from the U.S. Department of Defense, with additional support from the David and Lucile Packard Foundation.

1 Introduction

While modern computers have typically been constructed from such elements as vacuum tubes and transistors, schemes have been proposed showing how universal computers could be constructed, in theory, using frameworks as disparate as billiard balls [1], Conway's Game of Life [2], sliding-block puzzles [3], DNA and associated chemicals [4], and genetically modified bacteria [5]. In what follows, we show that evolutionary processes can also be used as a framework to perform arbitrary logical computations.

Evolutionary algorithms represent an area of considerable research interest [6], both as an engineering tool to discover solutions in situations too complex to tackle by hand[7], and as a tractable framework with which to study characteristics of the process of evolution itself[8]. In a genetic algorithm, a population of creatures, each described by a set of genes $\mathbf{x}$ (its genotype), is subjected to a process of artificial evolution as follows. Some fitness function $f(\mathbf{x})$ gives the fitness of each individual based on its genotype; fitter creatures have a greater chance of surviving and/or reproducing, depending on the details of the framework. Creatures then reproduce, with mutation resulting in offspring not identical to their parents, introducing new variations into the system. Finally, some creatures die off. These steps constitute one generation, and take place repeatedly. As the generations pass, the population comes to be composed of creatures whose genotypes are concentrated around local maxima of the fitness function (the so-called fitness peaks). Selection draws the population to the peaks; mutation keeps them spread out in a cloud in this genotype space. Evolutionary processes are commonly simulated in this way on standard general-purpose digital computers; however, the reverse has not hitherto been true.

To achieve this converse goal, we encode the variables of interest in the values of the genes $\mathbf{x}$ of the creatures in our artificial populations. Then by allowing individuals and populations to interact with one another in specified ways in an environment with limited resources, as elaborated below, we demonstrate the ability to create and link together a functionally complete set of logic gates.

2 The model

To illustrate the idea here, consider first a population of creatures having a single real-valued gene $x \in \Re$. Let the fitness function be characterized by two fixed sources of fitness in this one-dimensional genotype space, located at $x = a$ and $x = b$ ($a < b$):

$$f(x_i) = S_1 g(x_i - a) + S_2 g(x_i - b) \tag{1}$$

where i indexes members of the population, S_1 and S_2 are binary variables (for now equal to 1), and $g(x)$ is any bounded function which monotonically decreases as $|x|$ increases, such that the fitness landscape $f(x)$ has maxima at a and b; here we arbitrarily use

$$g(x) = \frac{1}{(cx)^2 + 1} \tag{2}$$

with c a constant, the choice of which may depend on a and b. The distinct values a and b correspond to the binary values represented by the population. To make the correspondence to Boolean logic most explicit, we would choose $a = 0, b = 1$; in general, any values may be chosen.

For continued simplicity, let's start by considering the small-population limit, where a population consists of a single creature. In each generation, the creature reproduces, the fitnesses of parent and child are calculated, and the fitter of the two survives (in the case of the tie, the parent is the survivor). Thus if the size of mutations is small compared to the distance between peaks $b - a$, the population will be confined to a local fitness maximum, and will stably retain a locally optimal value of x even if a higher peak exists elsewhere in genotype space. Let mutations be discrete, so that the offspring of a creature with gene value x will have value $x + \epsilon$ or $x - \epsilon$ with probability $\mu/2$ of each case; with probability $1 - \mu$, the gene value of the offspring will be identical to that of the parent.

Consider now what happens to a population of one at $x = a$ if we eliminate the fitness peak there by setting $S_1 = 0$. An offspring with $x = a - \epsilon$ or $x = a$ will have fitness less than or equal to that of the parent, and the value the population represents will remain unchanged. An offspring with $x = a + \epsilon$ will have greater fitness and so will replace its parent. In this way, the population will move to occupy the peak at $x = b$ after $(b-a)/\epsilon$ favorable mutations. (We limit ourselves to values of ϵ which are $b - a$ divided by a positive integer,

so that a population initialized at either a or b will be able to evolve to reach the other fitness peak exactly.) Conversely, a population starting at $x = b$ will not be affected by the elimination of the peak at $x = a$. Figure 1 illustrates how setting S_1 or S_2 to 0 in this way allows the population to be assigned the value $x = a$ or $x = b$.

2.1 Operations

In order to implement the various logic gates and functions, we allow the following three operations on the environment. First, we can turn off a given resource for a period of time, setting S_1 or S_2 to 0, as we did above in the case of assignment.

Second, we can raise or lower physical barriers, allowing once-separate populations to mingle, or dividing a single population into isolated subpopulations which can thereafter no longer interact and may be treated separately. Note that this physical space is entirely distinct from the genotype space in which the populations evolve. The former determines which populations can interact and which are confined to be separate from one another; the latter specifies the values which the population represents.

Finally, we can poison a given population, to eliminate it from the system entirely. Because large populations, once mixed, are impossible to separate again, and in order to limit interactions to two populations at a time for tractability, it is frequently necessary to eliminate selected populations; and since one may be receiving a benefit from another which is enough to sustain it in the absence of other resources, setting S_1 and S_2 both to zero is insufficient to ensure an extinction. Thus, some explicit form of poison or plague is required.

Armed in this way with famine, tectonics, and pestilence, we can proceed to construct gates.

2.2 Logic gates

For binary logical operations, we put two populations into the same spatial region, and let them interact with each other, the members of one providing a benefit or penalty to those of the other:

$$f(x_i) = S_1 g(x_i - a) + S_2 g(x_i - b) + K \sum_j g(x_i - y_j) \qquad (3)$$

where K is a constant, and the summation is over all members of the other population (represented by y rather than x). In the population-

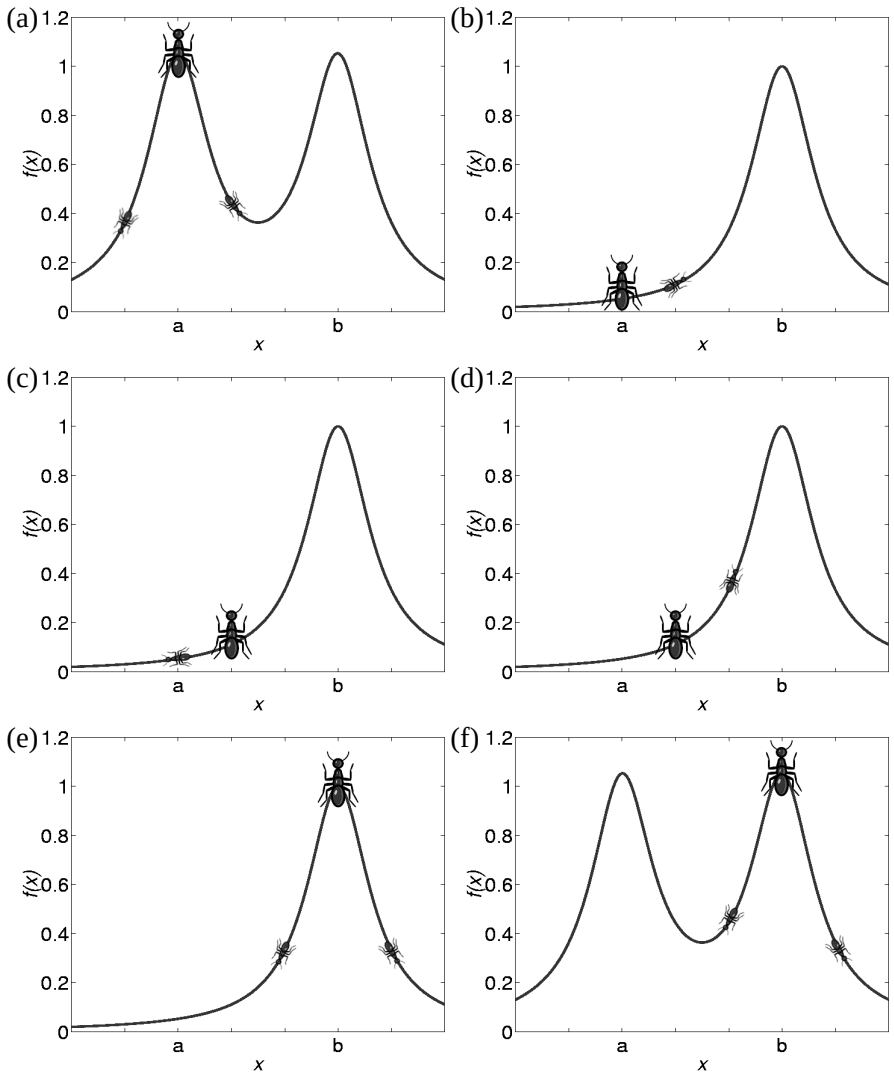

Figure 1: Setting a population of size 1 to the desired value $x = b$. (a) An individual at $x = a$ (shown here as an ant) is at a fitness peak, so that any mutant offspring (shown as smaller ants) will be of lower fitness and thus will not replace it. (b) Setting S_1 to 0 eliminates that peak, so that an offspring of value closer to $x = b$ will replace its parent. (c) If a mutant offspring happens to have value further from $x = b$, it will have lower fitness and so will not replace the parent. (d) Each successive mutation in the direction of $x = b$ moves the population toward this one remaining fitness peak. (e) After $(b - a)/\epsilon$ such favorable mutations, the population has attained the value $x = b$. Further mutant offspring, or offspring of an individual which had started at $x = b$, will not result in further changes to the value represented by the population. (f) When S_1 is reset to 1 and the lower peak is restored, the population remains stably at $x = b$.

one case, of course, the above expression reduces to

$$f(x) = S_1 g(x-a) + S_2 g(x-b) + K g(x-y) \tag{4}$$

One constraint on the constants a, b, c, K, ϵ is that they must be chosen so that the fitness landscape defined by $f(x)$ remains bimodal for $S_1 = S_2 = 1$ in the presence of a second population y, so that populations remain stable on either peak. For instance, $\{a = 0, b = 1, c = 3, K = 1, \epsilon = 1/3\}$ is one set of choices that satisfies this constraint.

Let two populations be interacting in the same region in this way, each with its gene value stably at a or b. Now let S_2 be set to 0, eliminating the upper fitness peak. If either population is at $x = a$, the remaining fixed resource there will keep it fixed at that value; if exactly one population is at $x = b$, its fitness landscape will be unimodal with peak at $x = a$, and so it will migrate to that lower peak; but if both populations are at $x = b$, each will provide a fitness resource for the other, thus maintaining a local maximum there so that both can remain there stably.

This, then, is an AND gate. Figure 2 illustrates the idea graphically. The inputs are encoded in the values x of the two one-member populations before they are first allowed to interact; at the end of the computation, both populations encode the output. To use the result in another operation, one population would need to be poisoned to remove it, as discussed above, leaving the remaining population isolated at that value and ready to serve again as an input.

An OR gate is implemented similarly, but setting S_1 rather than S_2 to 0; an analogous argument applies.

NOT is slightly more complicated. Let there be three classes of populations, ordered circularly: individuals of one class receive a benefit from those of the same class ($K = K_1 > 0$), are penalized by those of the previous class ($K = K_2 < 0$), and have no interaction with those of the next class ($K = 0$). If we have a population whose value we wish to invert, we first introduce into the same spatial region a new population belonging to the following class; there is no effect on the original population ($K = 0$), but the new one, penalized by the presence of the old, is driven to whichever peak was unoccupied. The original population is then poisoned to remove it from the system. A third population, of the class after the second, is introduced; it similarly evolves to the complementary peak, after which the second population is poisoned. Finally, a fourth and last population of the class following the third is introduced; it again moves to the op-

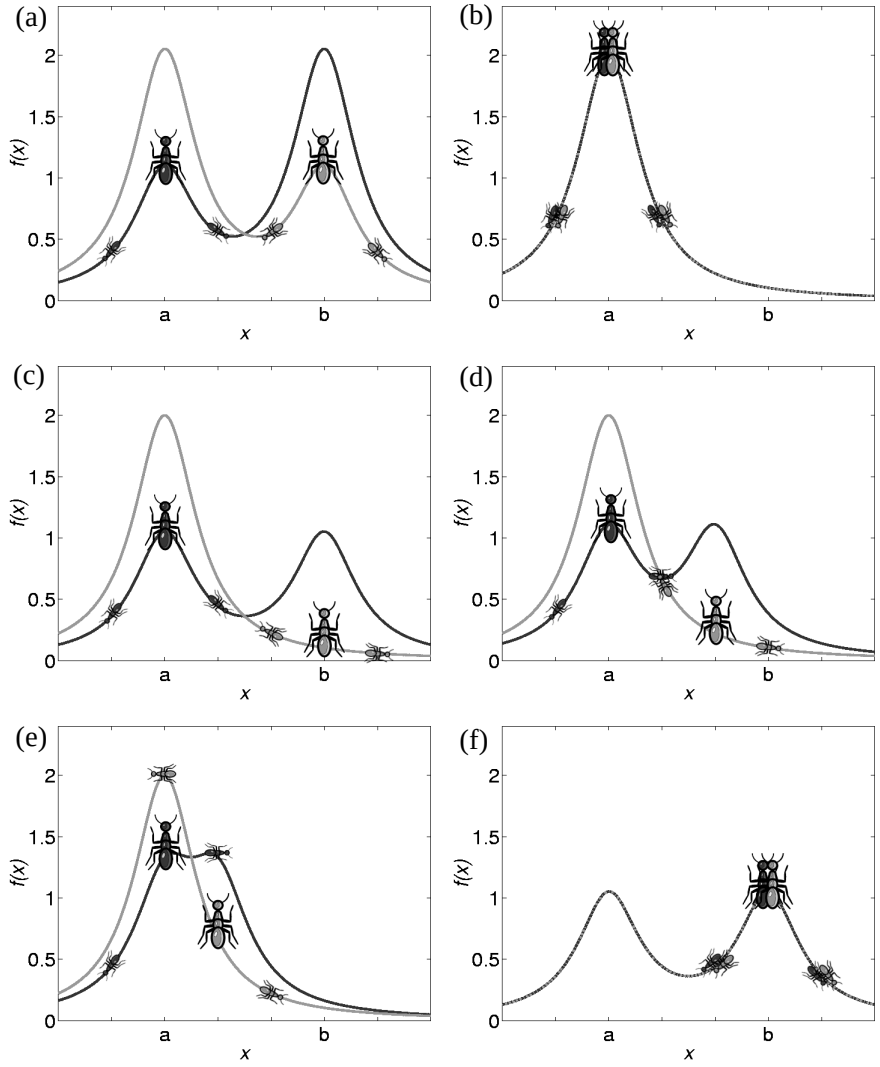

Figure 2: Implementing an AND gate with two populations each of size 1. The two populations are shown as light and dark ants, each evolving on a separate fitness landscape of its own color. Possible offspring are shown as small ants. (a) With one population on each peak and $S_1 = S_2 = 1$, the fitness landscapes for both populations are such as to confine them to their current values. (b) If both populations are at $x = a$ when S_2 is set to 0, the single remaining fitness peak at $x = a$ for both populations will confine both to that value. (c) If one population is at $x = a$ and the other at $x = b$, the respective fitness peaks at $x = a$ for both populations will confine the former to that value and provide selection pressure for the latter to evolve to that value (d,e), finally resulting again in the configuration shown in (b). (f) If both populations are at $x = b$ when S_2 is set to 0, the resource each supplies for the other will provide the local fitness peak necessary to keep both populations at that higher value.

posite peak, and the third population is poisoned. The net result is a population of the original class on the opposite fitness peak, three negations having taken place.

FAN-OUT can be implemented by adding a new population for which $S_1 = S_2 = 0$, while S_1 and S_2 remain 1 for the original population; the single resource represented by the existing population will lead the new one to join it, while the presence of the fixed resources for the original population will keep its value stable despite the influence of the new one (strictly, this requires $K_1 \leq 1$). Raising a spatial barrier such that one population falls on one side and one on the other then results in two copies of the original state.

With these gates, the system is functionally complete.

2.3 Dynamics

We next consider the dynamics of the evolution. If the environment of a creature with, for example, gene value $x = a$ changes to eliminate the local fitness peak there, then a child with x closer to b will replace it with certainty. The probability of such a child arising in any generation is $\mu/2$; with $\epsilon = (b-a)/n$ (n a positive integer), the probability of the one-member population reaching the other peak within N generations is

$$\sum_{i=n}^{N} \left(1 - \frac{\mu}{2}\right)^{i-n} \left(\frac{\mu}{2}\right)^{n} \binom{i-1}{n-1} \tag{5}$$

For $\mu = 2/3, n = 3$, for instance, this gives a 99.9% probability of the population migrating from a to b within 29 generations. An arbitrary degree of certainty of a logic operation successfully being performed can thus be attained at the cost of reducing the clock speed of the system, that is, increasing the number of generations allocated to the operation.

It may appear at first that we can maximally speed the search through genotype space by choosing $\mu = 1$, since the fitter of parent and child always survives, so with no risk of losing a more favorable variant and nothing to gain by duplicating the parent exactly, the child should always explore a new variant. However, when we extend the analysis to multiple genes per creature, it becomes crucial that a child be able to duplicate the value of each parental gene in order for genes to vary independently and an unconstrained search through genotype space to take place.

We can extend these logic operations to many-bit variables in exactly this way, by increasing the number of genes per individual (making x a vector), and using a fitness function which is simply the sum of the fitness functions for each individual gene as above. Evolution then proceeds to evaluate the desired logic function, as described in more detail below, on all genes in parallel. However, in the population-one limit, a favorable mutation in one gene may be accompanied by an unfavorable one in another, with the total fitness increasing, so that this child replaces its parent. In the worst case, while one gene value moves towards the fitness peak, another may repeatedly move away; that one will then require an additional $1/\epsilon$ steps to reach its optimal value, during all of which the first must maintain its own value; a third gene may have been moving in the wrong direction during this entire process, and so on. As a result, the number of generations required to ensure that all genes change their value, if necessary, with a given degree of certainty ends up scaling exponentially with the number of genes. For this reason, and because the population-one case can be made arbitrarily but not perfectly reliable, we now turn our attention to the opposite extreme.

2.4 Infinite-population limit

With an infinite number of creatures in the population, every possible value of x for which $f(x) > 0$ will eventually become populated. The most convenient representation of the population is as a density function $p(x)$ (or, in this case with discrete values of x, a vector) expressing for each x what fraction of the population takes that value. The update rule we use in this case is for the population at each x to produce a fraction of the next generation according to its fitness $f(x)$, with μ of those offspring mutating into neighboring values of x as before:

$$\begin{aligned} p(x,t+1) &= p(x,t)f(x,t)(1-\mu) \\ &\quad + p(x-\epsilon,t)f(x-\epsilon,t)\frac{\mu}{2} \\ &\quad + p(x+\epsilon,t)f(x+\epsilon,t)\frac{\mu}{2} \end{aligned} \tag{6}$$

after which the population is renormalized to ensure that $\int p(x)dx = 1$. The value represented by the population is that x with the greatest fraction of the population, $\operatorname{argmax}_x p(x)$.

In this case, we no longer have the property that a population will stay isolated on a local suboptimal fitness peak; given enough time, the population will always come to be distributed about the global maximum. This means that, while we can use essentially the same approach to constructing logic gates as above[1], we can only eliminate a fixed resource for a limited time before the population shifts, perhaps inappropriately, to center around whichever value still has its fitness resource turned on.

With that consideration, in order to make this implementation work, we make a few additional changes to the algorithm, sharpening the peaks associated with the sources of fitness and adding a self-excitation term for increased stability:

$$\begin{aligned} f(x) &= S_1(g(x-a))^2 + S_2(g(x-b))^2 \\ &\quad + K\int_{y=-\infty}^{y=+\infty}(p'(y)g(x-y))^2dy \\ &\quad + E\int_{y=-\infty}^{y=+\infty}(p(y)g(x-y))^2dy \end{aligned} \tag{7}$$

where $p(x)$ is the population density for the population of interest, $p'(x)$ that for the other population it interacts with, and E the strength of this self-excitation, reflecting how much support an individual receives from other, similar members of its own population.

Tables 1 and 2 demonstrate that this scheme is not sensitive to very careful choice of parameters. The first shows several disparate choices; for each, the second shows the range of intervals of turning off a fixed resource for which each gate performs correctly—fewer generations and a switch may fail to occur, more time and a population may switch when it shouldn't. Each of these operations should be followed by a buffer, a period of a few generations when both fixed resources are turned on, to stabilize the population(s) at the new value(s).

This infinite-population model can also be scaled up to many-bit variables, without the number of generations required for an operation increasing as in the population-one model. Let creatures possess traits $\{x_1, x_2, \ldots, x_n\}$, and let the total fitness function be the sum of fitness

[1]The easiest way to implement FAN-OUT in this case is simply to raise a physical barrier for the population that is to be duplicated, so that part falls on one side and part on the other; because we assume the population is infinite and spatially well-mixed, both new subpopulations reflect the composition of the original, and thereafter can be treated as separate copies of it.

Parameter set	a	b	c	μ	ϵ	K_1	K_2	E
I	0	1	3	2/3	1/7	1	-4	2.25
II	-1	1	1	1/2	1/4	.7	-5	1
III	5	10	1/5	1/4	1/10	1	-8	.7

Table 1: Three example sets of values for the eight parameters of the model.

Parameter set	Assignment	AND, OR	NOT (each of three steps)
I	9 or more	6–12	22 or more
II	5 or more	3–6	7 or more
III	40 or more	37–58	53 or more

Table 2: For each of the example parameter settings of Table 1, shows for each logical function the range of number of generations for which the gate will function properly. FAN-OUT takes effect within a generation.

functions as before for each separate trait:

$$
\begin{aligned}
f(x_1,\ldots,x_n) \;=\; & \sum_{i=1}^{n}\Big(S_1(g(x_i-a))^2+S_2(g(x_i-b))^2\Big) \qquad (8)\\
& +K\int_{y_1=-\infty}^{y_1=+\infty}\cdots\int_{y_n=-\infty}^{y_n=+\infty}(p'(y_1,\ldots,y_n)\\
& \cdot(g(x_1-y_1)+\cdots+g(x_n-y_n)))^2dy_1\cdots dy_n\\
& +E\int_{y_1=-\infty}^{y_1=+\infty}\cdots\int_{y_n=-\infty}^{y_n=+\infty}(p(y_1,\ldots,y_n)\\
& \cdot(g(x_1-y_1)+\cdots+g(x_n-y_n)))^2dy_1\cdots dy_n
\end{aligned}
$$

Populations then evolve to perform the designated logic operations on each of the traits in parallel. Parameter settings and other details of the model may need to be changed for higher dimensions to ensure that the gates work correctly, but the number of generations required for each operation stays low.

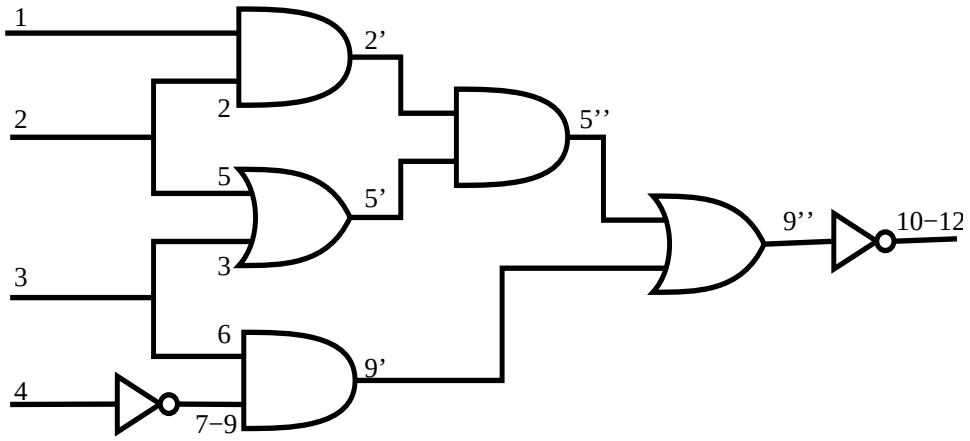

Figure 3: Logic circuit instantiated in the example. Wires are labeled with the numbers of the populations which represent them. Primed labels refer to the same population after an operation has been performed.

3 Demonstration

To illustrate the use of this model, we use it to implement the arbitrary logic circuit shown in Figure 3, the truth table for which is given in Table 3. For ease of presentation, we use the infinite-population model, with one trait per individual. Parameter values are those given as set I in Table 1.

Table 4 gives the sequence of basic operations that are performed in order to implement this circuit in a dozen populations. Figure 4 shows the result.

1	2	3	4	$2'$	$5'$	9	$9'$	$5''$	$9''$	12
0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	1
0	0	1	0	0	1	1	1	0	1	0
0	0	1	1	0	1	0	0	0	0	1
0	1	0	0	0	1	1	0	0	0	1
0	1	0	1	0	1	0	0	0	0	1
0	1	1	0	0	1	1	1	0	1	0
0	1	1	1	0	1	0	0	0	0	1
1	0	0	0	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1
1	0	1	0	0	1	1	1	0	1	0
1	0	1	1	0	1	0	0	0	0	1
1	1	0	0	1	1	1	0	1	1	0
1	1	0	1	1	1	0	0	1	1	0
1	1	1	0	1	1	1	1	1	1	0
1	1	1	1	1	1	0	0	1	1	0

Table 3: Truth table for the logic circuit of Figure 3. Numbers refer to the populations encoding each value, matching the labeled wires in the figure.

Generation	Operation
2	Split population 2 into 2 and 5 Split population 3 into 3 and 6 Introduce 7 into same spatial area as 4 (NOT, step 1)
27	Poison population 4 Introduce 8 into same area as 7 (NOT, step 2)
49	Poison 7 Introduce 9 into same area as 8 (NOT, final step)
71	Poison 8
76	Introduce 1 into same area as 2 Introduce 5 into same area as 3 Introduce 6 into same area as 9
77	Set $S_2 = 0$ in area of 1 and 2 (AND) Set $S_1 = 0$ in area of 3 and 5 (OR) Set $S_2 = 0$ in area of 6 and 9 (AND)
84	Set $S_1 = S_2 = 1$ everywhere
85	Poison 1, 3, and 6
86	Introduce 2 into same area as 5 Set $S_2 = 0$ in area of 2 and 5 (AND)
93	Set $S_2 = 1$ in area of 2 and 5 Poison 2
95	Introduce 5 into same area as 9
96	Set $S_1 = 0$ in area of 5 and 9
103	Set $S_1 = 1$ in area of 5 and 9 Poison 5
104	Introduce 10 into same area as 9 (NOT, step 1)
126	Poison 9 Introduce 11 into same area as 10 (NOT, step 2)
148	Poison 10 Introduce 12 into same area as 11 (NOT, final step)
170	Poison 11 Final answer is now in 12.

Table 4: Sequence of operations that implements the circuit of Figure 3. The system in action is shown in Figure 4. Populations 7 and 10 are of one class, 8 and 11 of the following class, and all others of the following class (which in turn precedes that of 7 and 10).

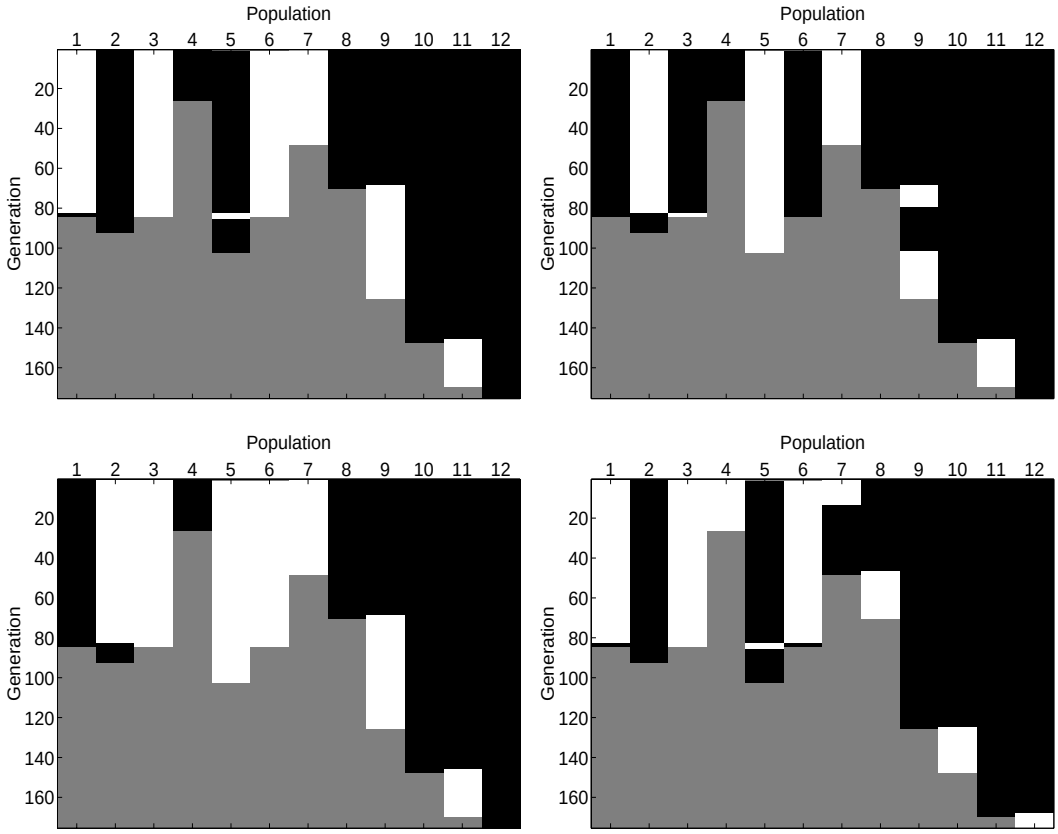

Figure 4: Illustration of the progression of evolution for four sets of initial conditions (populations 1 through 4): top row, {1010}, {0100}; bottom row, {0110}, {1011}. Populations are colored black (0) or white (1) according to the value they encode, as described in the text; extinct populations are shown in gray. The final output of the system is represented in population 12.

4 Conclusion

In the preceding, we have described the construction and use of an evolutionary system which implements universal computation, for a wide range of parameter values. It would be desirable to simplify the model still further—for instance, finding a way to eliminate the need for poison, or for three separate population classes—for increased elegance. However, the framework as described is a general one which could conceivably lend itself to instantiation in some physical system, outside pure simulation.

With this demonstration of how evolutionary systems can perform universal computation, some reader is bound to ask the question: could the evolutionary system of the Earth's biosphere itself be performing some unthinkable computation [9]? In some sense, yes; every natural or artificial process can be thought of as carrying out some computation [10], even though it may be one no deeper than "What state does this physical process bring this system to?" But is the world around us arranged in such a way as to implement logical operations on discrete values, or could the underlying cause for the phenomena we see be the deliberate evaluation of a logical expression? There is no evidence to suspect such a thing to be the case, but it is interesting that the idea is not a strict impossibility.

Acknowledgments

This work developed as an offshoot of ongoing research with Whitman Richards, who additionally provided encouragement and made very helpful comments on the manuscript. The work was supported by a National Defense Science and Engineering Graduate Fellowship from the U.S. Department of Defense, with additional support from the David and Lucile Packard Foundation.

References

[1] Fredkin, E., and T. Toffoli. Conservative logic. *Int. J. Theor. Phys.*, **21**(3/4):219-253, 1982. Available at `http://kh.bu.edu/qcl/`.

[2] Berlekamp, E., J. Conway, and R. Guy. Winning ways for your mathematical plays. London/New York: Academic Press, 1982.

[3] Hearn, R., and E. Demaine. The nondeterministic constraint logic model of computation: reductions and applications. In *Proc. Int. Colloq. Automata, Languages, and Programming*, 2002. To appear.

[4] Winfree, E. Algorithmic self-assembly of DNA. Ph.D. thesis from California Institute of Technology. Available at http://www.cs.caltech.edu/~winfree/old_html/Papers/thesis.ps.gz.

[5] Weiss, R., G. Homsy, and T. Knight. Toward *in vivo* digital circuits. In *Proceedings of the Dimacs Workshop on Evolution as Computation*, January 1999. Available at http://www.swiss.ai.mit.edu/~rweiss/bio-programming/dimacs99-evocomp.ps.

[6] Holland, J. Adaptation in natural and artificial systems, 2nd edition. Cambridge: MIT Press/Bradford Books, 1992.

[7] Koza, J. Genetic programming: on the programming of computers by means of natural selection. Cambridge: MIT Press, 1992.

[8] Mitchell, M., J. Crutchfield, and R. Das. Evolving cellular automata with genetic algorithms: a review of recent work. In *Proceedings of the First International Conference on Evolutionary Computation and its Applications*, Russian Academy of Sciences, 1996. Available at http://www.santafe.edu/~evca/Papers/evca-review.html.

[9] Adams, D. The hitchhiker's guide to the galaxy. New York: Crown Publishers, Inc., 1979.

[10] Fredkin, E. A new cosmogony. In *Proc. Physics and Computation Workshop*, 1992. Available at http://www.digitalphilosophy.org/new_cosmogony.htm.