The Evolution of Technology within a Simple Computer Model

W. BRIAN ARTHUR[1,2] AND WOLFGANG POLAK[2]
[1]Santa Fe Institute, Santa Fe, New Mexico 87501; and [2]Fuji Xerox Palo Alto Laboratory, Palo Alto, California 94306

Received April 12, 2006; accepted April 12, 2006

Technology—the collection of devices and methods available to human society—evolves by constructing new devices and methods from ones that previously exist, and in turn offering these as possible components—building blocks—for the construction of further new devices and elements. The collective of technology in this way forms a network of elements where novel elements are created from existing ones and where more complicated elements evolve from simpler ones. We model this evolution within a simple artificial system on the computer. The elements in our system are logic circuits. New elements are formed by combination from simpler existing elements (circuits), and if a novel combination satisfies one of a set of needs, it is retained as a building block for further combination. We study the properties of the resulting build out. We find that our artificial system can create complicated technologies (circuits), but only by first creating simpler ones as building blocks. Our results mirror Lenski et al.'s: that complex features can be created in biological evolution only if simpler functions are first favored and act as stepping stones. We also find evidence that the resulting collection of technologies exists at self-organized criticality. © 2006 Wiley Periodicals, Inc. Complexity 11: 23–31, 2006

Key Words: biological evolution; technology evolution; self-organized criticality

New technologies are never created from nothing. They are constructed—put together—from components that previously exist; and in turn these new technologies offer themselves as possible components—building blocks—for the construction of further new technologies.[1] In this sense, technology (the collection of mechanical devices and methods available to a culture) builds itself out of itself.[2] Thus in 1912 the amplifier circuit was constructed

Correspondence to: Wolfgang Polak, E-mail: polak@fxpal.com

[1]The idea that novel technologies are constructed from components—technologies—that already exist was observed by Ogburn in 1922 [8]. And Kaempffert in 1930 noted that novel technologies are "composites of mechanical elements that accumulated as part of the social heritage" [9]. See Arthur [10] for a fuller and more rigorous treatment of this idea.

[2]We can therefore say that in its collective sense technology is self-producing, or autopoietic. (The term "autopoietic" *was*

from the already existing triode vacuum tube in combination with other existing circuit components. The amplifier in turn made possible the oscillator (which could generate pure sine waves), and these with other components made possible the heterodyne mixer (which could shift signals' frequencies). These two components in combination with other standard ones went on to make possible continuous-wave radio transmitters and receivers. And these in conjunction with still other elements made possible radio broadcasting.

In its collective sense, technology forms a network of elements in which novel elements are continually constructed from existing ones.[3] Over time, this set bootstraps itself by combining simple elements to construct more complicated ones and by using few building-block elements to create many. This evolution is driven not just by the availability of previous technologies. It is driven by the large collection of human needs and also by needs brought into being by technologies themselves. Particular needs (in actual human history for food, transportation, cures for diseases, communication, and the drainage of fields and mines) are satisfied by simple technologies at first and then by more sophisticated ones that replace these simpler ones. Technologies that are replaced (think of horse transportation) become obsolete and in so doing may render other technologies that depend on them (carriage making and blacksmithing) obsolete, so that new elements not only add to the network but engender what Schumpeter called "gales of destruction" [1]. All this happens of course through the agency of the economy (which we can think of in shorthand as an organizational structure for arranging how technologies meet needs) and through the human agency of engineers, scientists, and developers.

It would be possible to explore this evolution of technology by historically examining its build out piece by piece over the course of human history. In this article we take a different course. We model the build out of technology by constructing a simple artificial world within the computer. In this world the technologies—the elements that build out—are logic circuits. (Logic circuits have the advantage that their function can be described exactly, and there are simple rules for forming them by combination.) We imagine that our artificial world has certain logical needs (for the ability to perform the exclusive-or function, say, or to be able to add 3-bit numbers), and these can be potentially satisfied by suitable logic circuits, providing they can be created. Starting from a primitive technology (in most of our experiments a simple NAND circuit), new circuits—new technologies—are constructed by randomly wiring together existing ones and testing the result to see whether they satisfy any existing needs. If a circuit proves useful—satisfies some need better than its competitors—it replaces the one that previously satisfied that need. It then adds to the active collection of technologies and becomes available as an element for the construction of still further circuits. In this way elements constantly add to the set of active technologies as they find uses and leave again if rendered obsolete by others. In this way the collection of technologies bootstraps upward by first creating simple technologies that satisfy simple needs, then from these more complex technologies that satisfy more sophisticated needs.

We ask several questions. What are the properties of technology evolution in our artificial system? By what steps does the network of technology evolve? Do some technologies emerge as enabling ones (like ore smelting or the transistor) that have many uses in further combination, so that usefulness in generating further technologies is highly skewed? Do we see Schumpeterian gales of destruction? And if we start from a primitive technology, can our system artificially create combinations of elements that satisfy complex needs: that is, could our system evolve from one primitive circuit to satisfying a need say for 4- or 8-bit addition? (Note that our interest is in studying the evolution of complex artifacts and not in the engineering problem of generating efficient logic circuits for Boolean functions that has been solved.)

We pay some attention to this last question. In real life, complex technologies are created both from the existence of simpler ones and from the particular needs that brought these simpler building blocks into being. Radar could not have been invented without the building blocks of amplification and wave generation—and the needs that brought these simpler functions into existence. We should therefore not expect complicated circuits to appear without intermediate elements and without the simpler intermediate needs that generate these. There is a parallel observation in biology. Complex organismal features such as the human eye cannot appear without intermediate structures and "needs" or uses for these intermediate structures [2,13].

We find that the collective of technology in our system can indeed bootstrap itself from extreme simplicity to surprisingly complicated circuits. We find, as we would expect, that most technologies created are not particularly useful as building blocks, but some turn out to be key in creating

coined by Maturana and Varela [11].) This assertion that technology creates itself from itself requires a qualification. At bottom all technologies are created from harnessed phenomena [10,12]. But phenomena are harnessed into use via existing physical devices and methods—by existing technologies. Thus, providing we think of phenomena as being harnessed by existing technologies and that we bracket the human activities that create new technologies, we can say that technology creates itself.

[3]*This network is more properly defined by what brings what into existence—what makes what possible—and not just by what components are contained in each new technology.*

© 2006 Wiley Periodicals, Inc.
DOI 10.1002/cplx

TABLE 1

Needs Are Common Logic Functions for $2 \leq n \leq 15$, $1 \leq k \leq 8$, and $2 \leq m \leq 7$

Name	Inputs	Outputs	Description
not	1	1	Negation
imply	2	1	Implication
n-way-xor	n	1	Exclusive or, addition mod 2
n-way-or	n	1	Disjunction n inputs
n-way-and	n	1	Conjunction n inputs
m-bitwise-xor	$2m$	m	Exclusive or on m input pairs
m-bitwise-or	$2m$	m	Disjunction on m input pairs
m-bitwise-and	$2m$	m	Conjunction on m input pairs
full-adder	3	2	add 2 bits and carry
k-bit-adder	$2k$	$k + 1$	Addition
k-bit-equal	$2k$	1	Equality
k-bit-less	$2k$	1	Comparison

descendant technologies. We find avalanches of replacement—Schumpeter's "gales of destruction." These follow a power law, so that the collective of technology shows evidence that it exists at self-organized criticality. And we find that the system arrives at complicated circuits only by first satisfying simpler needs and using the results as building blocks to bootstrap its way to satisfying more complex ones.

THE EXPERIMENTAL SYSTEM

We view each run of our artificial system as an experiment. Each experiment starts with only primitive components (usually one, an elementary logic gate), and the computer generates new circuits by randomly wiring together several components in a noncyclic way. A component can be a primitive logic gate or another circuit that has been created from this and has been encapsulated (think of it as a chip with designated input and output pins). We specify in Table 1 a set of *needs* or *goals*, useful logical functions to be achieved possible by the combinations. These are akin to the needs that drive technology evolution. Ideally we would like these needs to be generated by agents who occupy an artificial world in which logical functions such as adders or comparators have proved useful. But we avoid this complication and simply list a set of useful logical functionalities that suitable circuits, if they appear, might achieve.

Using an artificial system that asks for logical functionalities and provides ways for them to be realized has the advantage that needs and technologies can be easily compared. Each need for a particular logical functionality can be represented by a specific truth table: a set of desired output values for every possible set of input values presented. And each circuit created—each technology—provides a function that can also be represented as a truth table: for every set of binary values provided to its input pins it produces particular binary values on its output pins. Thus we can easily match experimental technologies with our list of needs. We can also think of a technology's behavior, its truth table, as the *phenotype* of this technology. Its *genotype* is the architecture or internal circuitry that realizes this function. Many different genotypes can generate the same phenotype.

Our computer model, then, consists of a set of primitives, a set of technologies or components constructed from primitives and from other components, and a set of needs to be fulfilled. [We normally use only one primitive, a NAND gate, with phenotype $\neg(x \wedge y)$.] The essence of the experiment is simple. In each evolutionary step novel circuits are created from existing ones by randomly wiring together between 2 and12 circuits selected from all previously existing technologies according to a *choice function* that specifies probabilities of selection. Different phenotypic versions of the new circuit are created by selecting different internal wires in different orders as output pins. At each time there is a set of existing technologies that best match each of the needs or goals (have least incorrect entries in their truth tables). Each candidate circuit is tested against these to see if it improves upon them. It may do so by better matching a need's truth table, or if it has a function identical to that of an existing circuit, by costing less. (The cost of a circuit is determined by the number of its components and by *their* respective cost.) In either case it replaces the circuit it has improved upon both directly and in all circuits where that circuit is used as a component. It is also encapsulated: it becomes a new component that can serve as a building block for possible further combination. In this way the set of encapsulated technologies builds out. A need is *satisfied* if a new technology with its exact truth-table has been found. And a newly created circuit of course cannot replace one of its own components. Useful components are named (e.g., tech-256 or full-adder-121) and can be used in higher level technologies. Components that exactly implement a need are given mnemonic names describing that need (e.g., 3-bit-adder). Details of our implementation of these general algorithmic steps are listed in a section below.

The correspondence to the real world requires some comment. New technologies in the real world are indeed combinations of existing ones, but nowadays are rarely invented by randomly throwing together existing components. Loosely however we can think of each step in our process as a set of laboratory tests that investigates a novel idea. Or more exactly we can think of our process as corresponding to that used in modern combinatorial chemistry or synthetic biology, where new functionalities are created from random combinations and tested for their usefulness [3]. This process builds up a growing library of useful elements that can be exploited for further combination.

We can also think of this process more generally as an algorithm, not for solving a particular problem but for building up a library or repertoire of useful functionalities that can be combined to solve problems. The algorithm mimics the actual evolution of technology by first constructing objects that satisfy simple needs and using these as building blocks to construct objects of progressively higher complication.

EXPERIMENTAL RESULTS

The most complex circuits invented within 250,000 steps in our basic experimental design were as follows:

8-way-xor, 8-way-and, 8-way-or, 3-bitwise-xor, 4-bit-equal, 3-bit-less, and 4-bit-adder.

A more streamlined design, discussed below, created an 8-bit adder (which adds 8 bits to 8 bits correctly, a not uncomplicated achievement). Within the basic design different runs of the experiment invented circuits in different order and not all of these circuits evolved in the same experiment run.

Early in the experiment simple goals are fulfilled. We see from Figure 1, that even for simple circuits non-obvious implementations are invented. These circuits then become encapsulated for further use.

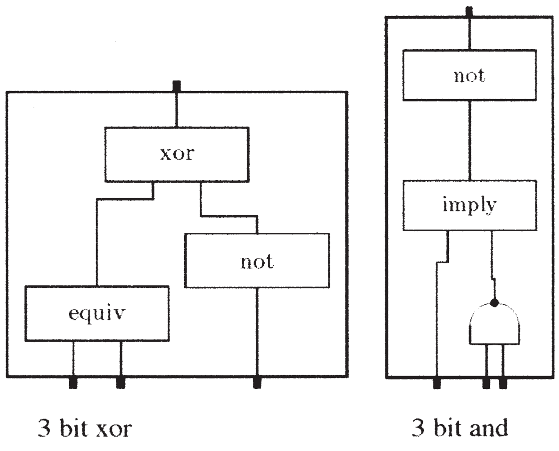
As the evolution proceeds more complicated circuits begin to construct themselves from simpler ones. The 2-bit-adder circuit shown in Figure 2 uses the supporting technology TECH-712. The latter circuit is an example of a technology that is useful toward satisfying a goal but that does not itself satisfy the goal 2-bit-adder (because the low-order (left) output bit is computed incorrectly). The circuit for 2-bit-adder is constructed from TECH-712 by adding circuitry to correct this error.

Some of our evolved circuits contain unused parts—they carry "junk DNA." The use of the 3-bit-adder on the right of Figure 3 is an example. In the course of the experiment such redundancies usually disappear because less "costly" circuits replace ones with needless complication.

Our experiment starts from the NAND primitive. In other versions of the experiment we used "implication" as the primitive. Similarly complicated circuits evolved. The process simply constructs the more elementary needs such as "not," and, and "xor" from the new "implication" primitive and proceeds as before.

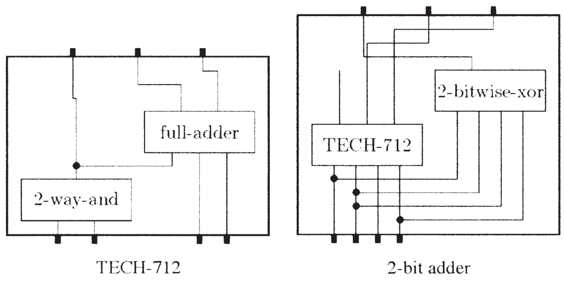
The emergence of circuits such as 8-bit adders seems not difficult. But consider the combinatorics. If a component has n inputs and m outputs there are $(2^m)^{(2^n)}$ possible phenotypes, each of which could be realized in a practical way by a large number of different circuits. For example, an 8-bit adder is one of over $10^{177,554}$ phenotypes with 16 inputs and 9 outputs. The likelihood of such a circuit being discovered by random combinations in 250,000 steps is negligible. Our experiment—or algorithm—arrives at complicated circuits by first satisfying simpler needs and using the results as building blocks to bootstrap its way to satisfy more complex ones.

FIGURE 1



Two circuits "invented" for simple goals.

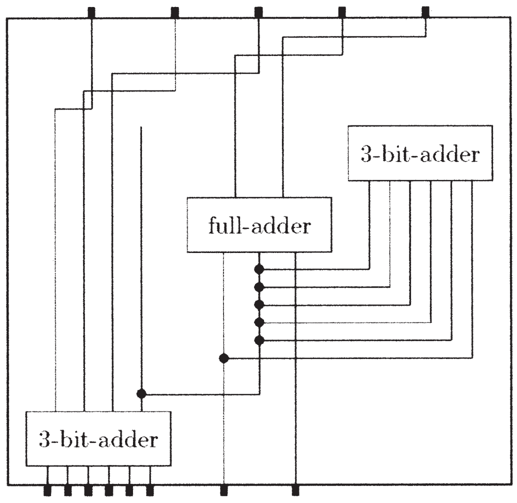
FIGURE 2



TECH-712 is useful towards satisfying the "2-bit-adder" goal because the two high-order bits are computed correctly. (The low-order bit is on the left. For multi-bit adders, input bits are interleaved.)

DOI 10.1002/cplx

FIGURE 3



A "4-bit-adder" circuit with an unconnected module.

THE BUILD-OUT OF TECHNOLOGIES

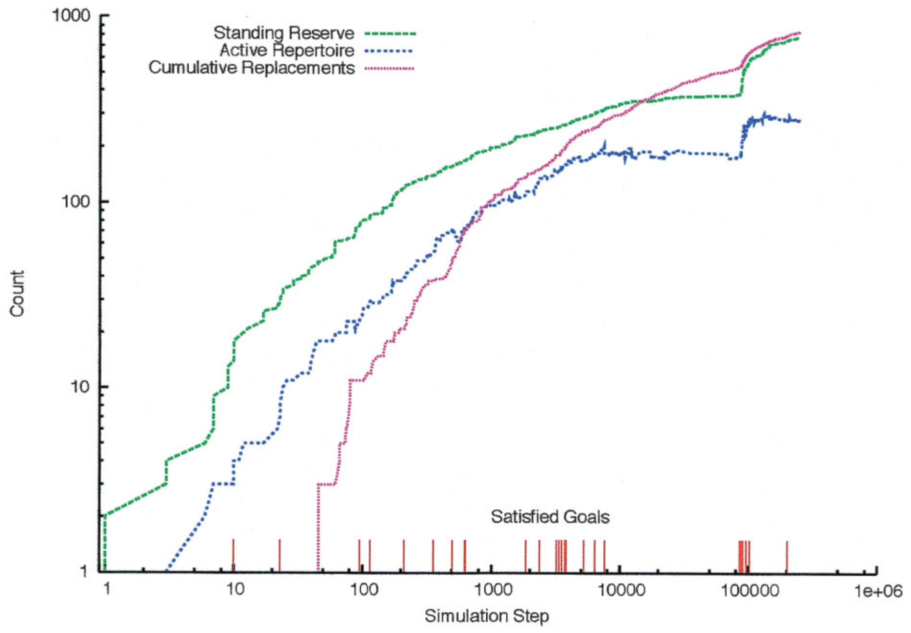
To talk about the build out of technologies we need two definitions. The collection of all methods and devices (all circuits) ever used we call the *standing reserve*. The technologies that are currently viable—in current use—and have not yet been replaced, we call the *active repertoire*.

Figure 4 shows the growth over time of the standing reserve, the technologies ever invented. In contrast the growth of the active repertoire, the number of technologies actually in use, is not monotonic. This indicates that important inventions render older technologies obsolete. Figure 4 also shows that there is continual improvement in accomplishing truth function "needs" as indicated by growing number of replacements.

Tick marks along the time axis of Figure 4 indicate when one of the needs has been satisfied. Progress is slow at first: the experiment runs for some time without meeting any goals exactly, and then functional species begin to appear leading to further species. The evolution is not smooth. It is punctuated by the clustering of arrivals because from time to time key technologies—key building block components—are "discovered" that quickly enable other technologies. For example, after a circuit for OR is invented, circuits for 3, 4, and 5-bit OR and bitwise-OR operations follow in short order. This appearance of key building blocks that quickly make possible further technologies has analogies in the real world (think of the steam engine, the transistor, the laser) and with the build out of species in biological evolution [2].

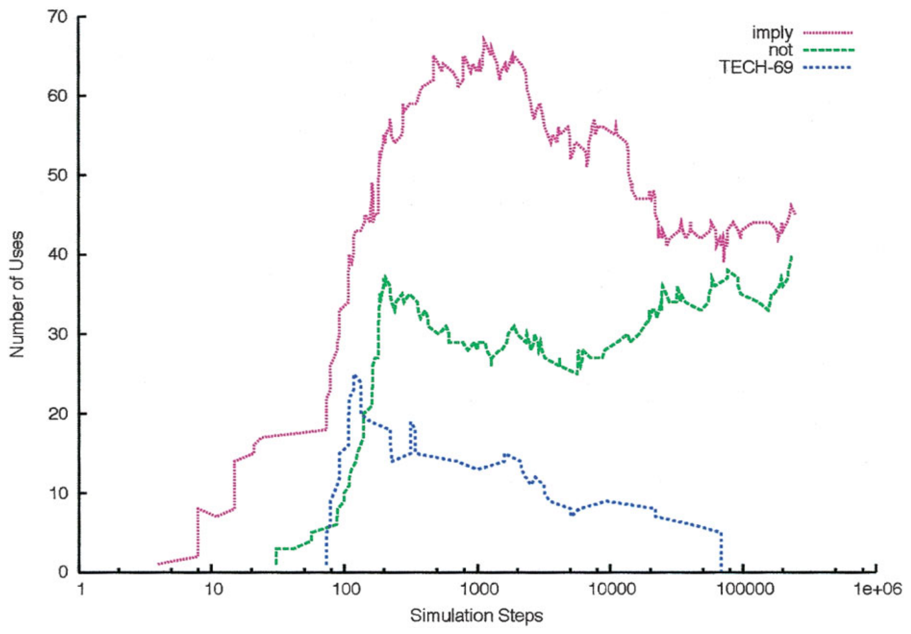
The order of invention makes a difference. Although "negation" is a simpler function than "implication", it hap-

FIGURE 4



The standing reserve, by definition, grows monotonically. The same is not true for the active repertoire because new inventions may improve upon and replace several existing ones.

FIGURE 5



"Implication," being invented before "negation" in this example, is used more heavily. Usage declines over time as better technologies are invented.

pens that in some runs of the experiment that the latter is invented first and is then used as a key building block. Figure 5 shows the result of one such experiment. "Implication" was used in a large number of other technologies and became much more prevalent than "negation." But eventually, its usage as a component declined as negation and other, less costly components offered themselves for combination. For comparison, the figure shows a third technology, TECH-69, which also performs implication but has 3 additional redundant inputs and contains unneeded components. Eventually, all uses of TECH-69 are replaced with the functionally equivalent but more efficient implication.

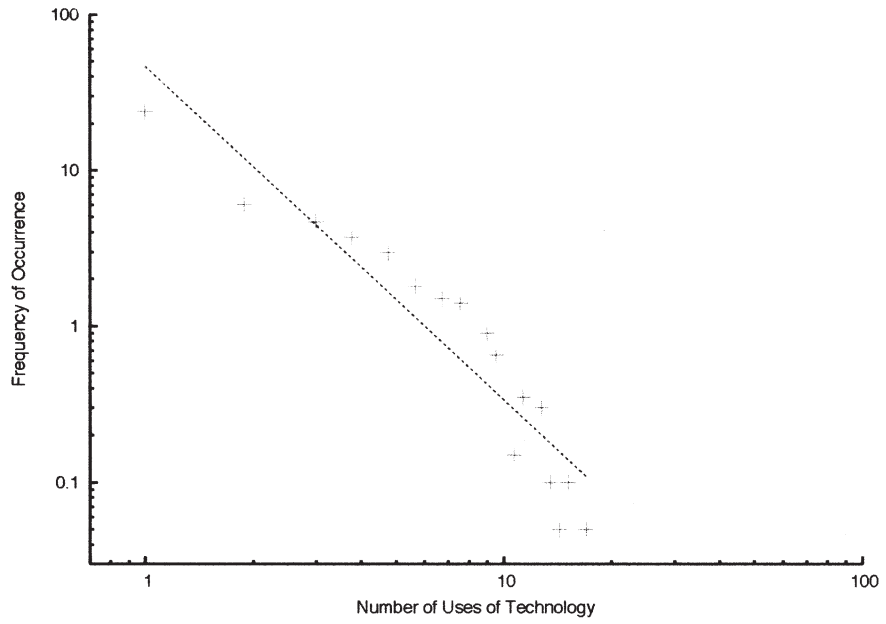
There is a trade-off between the number of needs or goals posted and the creation of new technologies. To illustrate this, we performed an experiment masking some of the needs and retaining a subset that we considered useful for the construction of adders: ("not, imply, 2-way-or, 2-way-xor, full-adder," and "k-bit-adder" for $1 \leq k \leq 8$). (We can also streamline the process by adding more difficult needs, as measured by the number of inputs and outputs, to the experiment only after simpler ones have been satisfied.) An 8-bit adder evolved very quickly within 64,000 simulation steps. In contrast, using more general goals, some simulation runs took over 675,000 steps before even a 4-bit adder evolved. A large disparate set of needs leads to broad generation of functionalities within the circuit design space, but is slow in arriving at particular complex needs. Narrowly focused goals lead to a deep search that reaches particular complex needs quickly, but produces a narrow library of functionalities.

The algorithm does not produce complex circuits without intermediate needs present. If we start without these, the repertoire of necessary building blocks is missing. For instance, if the "full-adder" goal is omitted from the goals for adders listed above, not even a 2-bit adder was found in one million steps. When the "full-adder" goal is present, it occasionally happens that the 2-bit adder is found before the "full-adder" is invented. This is because the invention of technologies that build toward the "full-adder" goal are also useful for the 2-bit adder.

The fact that as each step only circuits combining fewer than 12 existing components are considered defines a set of possible experimental circuits at any time—a large number—which we can think as the *adjacent probable* [4]. We can think of this as a probabilistic cloud that surrounds the existing technologies and that gradually lead to new ones by being realized by points near intermediate goals. Thus if a goal is too complicated it cannot be reached—realized—with reasonable probability, and so if stepping stone goals are not present the algorithm does not work.

The technologies we have listed as needs or goals are well-ordered in the sense that the more complicated ones can be constructed from the more elementary ones by repeating these in simple patterns. For example, a complicated circuit such as a 4-bit adder can be constructed from simpler elements such as adders and half-adders that repeat

FIGURE 6



Very few key technologies are used heavily to directly construct new ones. The plot shows the average over 20 experiments at their termination of 250,000 steps each.

in combination. What if we choose complicated goals that are not easy to realize by repetitive patterns? We can do this by selecting random truth tables with *n* inputs and *m* outputs as needs. Not surprisingly we find that often these can not be reached from our standard intermediate steps. By the same token, what if we replace our intermediate stepping-stone goals by random truth tables of the same intermediate size? Again, these also do not perform as well. The algorithm works best in spaces where needs are ordered (achievable by repetitive pattern), so that complexity can bootstrap itself by exploiting regularities in constructing complicated objects from simpler ones.

PROPERTIES OF THE NETWORK

Each technology (or encapsulated circuit) that is currently used to construct a technology is a node in the network of active technologies, and if two or more technologies are directly used to create a novel technology they show a (directed) link to this technology. A given technology *A* therefore links to its *user technologies*—the technologies it directly makes possible. As illustrated in Figure 6, some technologies have many links—are used very heavily to construct new ones—others have few. Usage approximates a power law (yielding a scale-free network) but by no means perfectly.

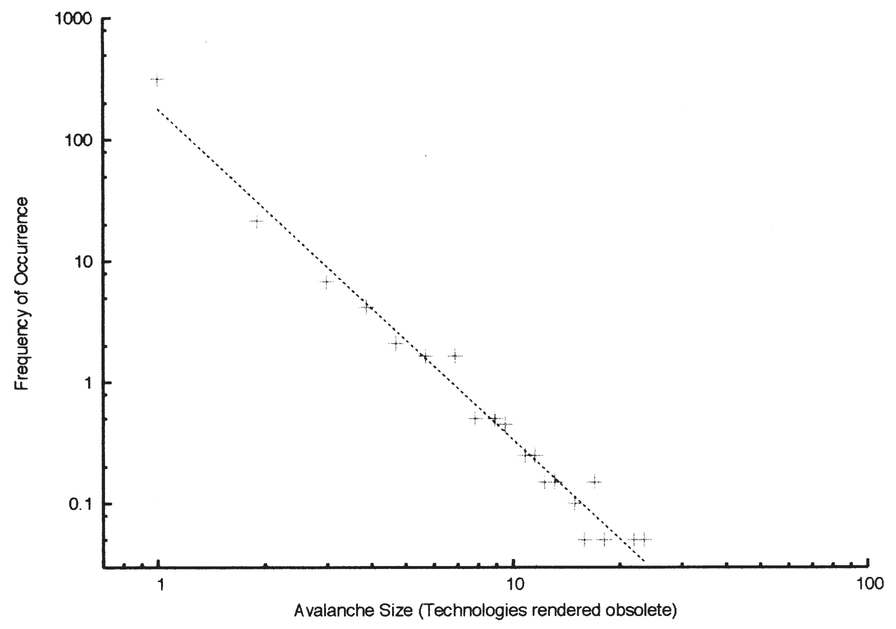
From time to time a new superior way of executing a certain functionality (or truth table function) is discovered. The new circuit may have fewer components or perform that function better. In this case the new circuit replaces the old one in all its *descendant* technologies (all the technologies below it in the network that use it directly or indirectly as a component). Replacement is immediate in our algorithm.

Replacement can also cause the collapse of technologies backward in the network. Suppose Tech 124 is used to construct Tech 136. Then, when a superior way to achieve Tech 136's functionality is found, Tech 124 may be left with no further use (it may neither satisfy any goal, nor be used in any active technology). In this case technology 124 disappears from the active set of technologies. In its disappearing, some of *its* component technologies may in turn be left with no further use. These also disappear from the active set. In this way technologies may be swept from active use in large or small avalanches of collapse—Schumpeter's "gales of destruction" [1]. Figure 7 shows that such sand-pile avalanches of collapse follow a power law. The scale on the size axis does not extend far however, because the number of technologies in the network is never large. We can take Figure 7 as suggestive that our system of technologies exists at self-organized criticality [5].

CONCLUSION

Using an artificial system, we have demonstrated how technology can bootstrap itself from extreme simplicity to a

FIGURE 7



Gales of destruction (or avalanches of collapse in use), average over 20 experiments.

complicated level, both in terms of the numbers of objects created and their intricacy. In the real world, of course, novel technologies are not usually constructed by random combination, nor are the needs for which technologies are created specified in a posted list. Nevertheless, all novel technologies are constructed by combining assemblies and components that already exist; the needs they satisfy are usually clearly signaled economically and technically; and existing technologies form a substrate or library of building blocks for future ones. The model captures certain phenomena we see in real life. Most technologies are not particularly useful as building blocks, but some (enabling technologies such as the laser or the transistor) are key in creating descendant technologies. Within our model, we find a strong indication that our collection of active technologies is subject to similar statistics as earthquakes and sand-piles: it exists at self-organized criticality. Our model also shows that the build out of technology depends crucially on the existence of earlier technologies constructed for intermediate or simpler needs. This mirrors the finding of Lenski et al. in biological systems that complex features can be created, but only if simpler functions are first favored and act as stepping stones [2].

A COMMENT ON THE ALGORITHM

Just as biological evolution has been the model for genetic algorithms and genetic programming, technology-based evolution may inspire a new form of automatic programming and problem solving. The algorithm we develop here, viewed abstractly, operates by discovering objects of intermediate complexity and bootstraps complication by using these as building blocks in further combinations. It bears some semblance to other evolutionary algorithms such as genetic programming. But unlike these it does not attempt to solve a given single problem. Rather, it attempts to create a toolbox of useful functionalities that can be further used for further problem solving. In doing so it sets up no parallel population of trial solutions for a problem. Instead it creates a growing collection of objects that might be useful in solving problems of increasing complexity within the same class. In this sense it does not resemble standard programming methods inspired by genetic mechanisms; rather, it is an abstraction of methods already used as combinatorial chemistry or synthetic biology or software construction that build libraries of objects for the creation of further objects.

EXPERIMENTAL CONDITIONS

Our experimental system is implemented in Common Lisp. Different sets of goals can be added to the system manually. The detailed behavior of the system is controlled by a number of configuration parameters. (The values we give below are default ones.) Extensive experiments with different settings have shown that our results are not particularly sensitive to the choice of these parameters.

To construct new circuits, at each step a small number of components (up to a maximum of 12) is selected and combined. The selection is made each time by randomly drawing a component either from the set of primitives, or the constants 0 and 1, or the set of circuits encapsulated as technologies, with probabilities 0.5, 0.015, and 0.485, respectively (and then choosing with equal probability within these sets). (For the purpose of selection, components that satisfy a goal exactly are added to the primitives' set.) Selected components may then be combined randomly to each other two circuits at a time, or to combinations of each other, to form new circuits for testing. To combine two circuits C_1 and C_2, each input of C_1 becomes an input of the combination; each input of C_2 becomes an input of the combination with probability 0.2; otherwise it is connected to a random output of C_1. All outputs of C_1 and C_2 become outputs of the combination. The step stops when a useful circuit has been found, or when a limit to combinations tested has been reached.

The cost of a circuit is the sum of the costs of its components. The cost of a primitive component is 1. The cost of a circuit encapsulated as a new technology/component is the number of its components. Thus, the cost of a technology is less than the cost of the circuit it encapsulates, reflecting the idea that it becomes a commodity. We use the cost function to decide when to replace an existing technology by a cheaper one.

Logic functions are represented by binary decision diagrams (BDDs) [6,7]. The phenotypes of goals and circuits are described by vectors of BDDs, one for each output wire. BDDs make it efficient to determine the equality of two logic functions. The representation also makes possible an efficient distance measure on logic functions, namely the number of input values for which two functions differ. We use this distance measure to define when one circuit C_1 better approximates a goal G than another circuit C_2. This is the case if for each output g of G circuit C_1 computes a function f that is closer to g than any of the functions computed by C_2. Note that this relation is a partial order, i.e., two circuits need not be comparable. A circuit C is encapsulated as a new technology if there is a goal G and no other circuit better approximates G than C. Only outputs of C appropriate for G become outputs of the new component, possibly making some parts of C redundant. In general, several circuits may approximate the same goal G at the same time, as when each circuit best satisfies some aspect (subset of the outputs) of the goal, but neither strictly dominates the other.

ACKNOWLEDGMENTS

This work was mainly carried out at and supported by FujiXerox Palo Alto Laboratory. We also thank the members of the Santa Fe Institute Workshop on Innovation in Natural, Experimental, and Applied Evolution (February 2004), for comments on an early version of these ideas. This workshop was supported by the Packard Foundation and the Thaw Charitable Trust. W.B.A. thanks Stuart Kauffman and John Holland for many discussions dating back to August 1987 on the notion of technologies creating technologies by combination. Both authors thank Eleanor Rieffel for helpful discussions in the early phases of this article, and Rob Carlson, Rich Lenski, and Peter Schuster for detailed comments.

REFERENCES

1. Schumpeter, J.A. The Theory of Economic Development; Transaction Publishers: New Brunswick, NJ, 1911/1934/1996.
2. Lenski, R.E.; Ofria, C.; Pennock, R.T.; Adami C. The evolutionary origin of complex features. Nature 2003, 423, 139–143.
3. Beck-Sickinger, A.; Weber, P. Combinatorial Strategies in Biology and Chemistry; Wiley: Chichester, England, 2001.
4. Kauffman, S.A. Investigations; Oxford University Press: New York, 2002.
5. Bak, P.; Wiesenfeld, K. Self-Organized Criticality: An Explanation for 1/*f* Noise. Phys Rev A 1988, 38, 364.
6. Bryant, R.E. Graph-based algorithms for Boolean function manipulation. IEEE Trans Comput 1988, 35(8), 677–691.
7. Bryant, R.E. Symbolic Boolean manipulation with ordered binary-decision diagrams. ACM Comput Surv 1992, 24(3), 293–318.
8. Ogburn, W.F. Social Change; 1922 (Dell: New York, 1966).
9. Kaempffert, W. Invention and Society. Reading with a Purpose Series, 56, American Library Association: Chicago, 1930.
10. Arthur, W.B. The Structure of Technology; 2007, in preparation.
11. Maturana H.; Varela F. Autopoiesis and cognition: the realization of the living. Boston Studies in the Philosophy of Science 42; Cohen R.S; Wartofsky, M.W., Eds.; D. Reidel Publishing: Dordrecht, 1973.
12. Arthur, W.B. The Logic of Invention. Santa Fe Institute Working Paper 2005-12-045; Santa Fe Institute: Santa Fe, NM, 2005.
13. Gehring, W.J. The genetic control of eye development and its implications for the evolution of the various eye-types. Int J Dev Biol 2002, 46, 65–73.