# The Evolution of Technology within a Simple Computer Model

W. Brian Arthur[§†] and Wolfgang Polak[†]

[§] *Santa Fe Institute, Santa Fe, New Mexico 87501, USA*

[†] *FX Palo Alto Laboratory, Palo Alto, California 94304, USA*

December 17, 2004

New technologies are constructed 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. In this sense, technology (the collection of mechanical devices and methods available to a culture) builds itself out of itself. Thus in 1912 the amplifier circuit was constructed from the already existing triode vacuum tube in combination with other circuit components. The amplifier in turn made possible the oscillator, and this with other components made possible the heterodyne mixer; these components in conjunction with other standard ones went on to make possible continuous wave radio transmission and reception, which made possible radio broadcasting. In its collective sense, technology forms a set – a network – of elements from which novel elements can be constructed. And over time, this set builds out by bootstrapping itself from simple elements to more complicated ones, and from few building-block elements to many. In this paper we model this buildout of technology and examine its properties within a simple artificial system.

We define a *technology* (singular) as a means to fulfill a purpose. Such a means may be instantiated as a physical device, or a method, or an industrial process, or an algorithm. Each technology executes at least one function and in this sense we can call it an *executable*. All technologies are put together or constructed from components (at least one), or assemblies, or subroutines, or stages. But each component or subsystem also has a function or assignment to carry out; each component therefore is an object with a purpose – and is also an executable. A technology therefore is an executable that is a combination of executables. Thus a given technology – think of an aircraft gas-turbine powerplant, for example – can be broken out into its component building blocks and these remain technologies – or executables. In this sense a technology is recursive in structure.

The evolution of technology is driven by human needs and by needs created by other technologies. A technology is useful when it can satisfy a need better or more economically than previous technologies. Usefulness may change over time as new inventions lead to more efficient technologies that quickly replace obsolete ones; for example, the transistor has eliminated most uses of vacuum tubes. The collection of all methods and devices ever used we call the *standing reserve*. The technologies that occur in the economy, those that are commercially viable and have not yet been replaced, we call the *active repertoire*. Technologies that are successful and used repeatedly in different circumstances become encapsulated as modules or building blocks that in combination with other building blocks make possible the construction of further technologies.

We explore several questions within our artificial system: What are the properties of technology evolution? By what steps does the network of technology evolve? How do complicated technologies arise from simple ones? We choose logic circuits as the elements in our experimental system. The domain of logic circuits has the advantage that their function can be described in a formal manner and there are simple combination rules for forming them. We imagine a world that has certain logical needs (certain Boolean functionalities) to be potentially satisfied by logic circuits; and we start from a primitive technology and explore whether we can artificially create combinations of elements that might bootstrap their way to satisfying these needs. New circuits (technologies) are constructed by wiring together existing ones and testing the result against existing needs. If such a circuit proves

useful it adds to the active repertoire. 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 which has been solved.

## The Experimental System

The experiment starts with only primitive components (elementary logic gates), and the computer generates new circuits by randomly wiring together several components in a non-cyclic way. A component can be a primitive logic gate or another circuit that has been encapsulated (think of it as a chip) with designated input and output pins. We specify in Table 1 below a set of *needs* or *goals*, useful logical functions to be achieved possibly by the combinations. These are akin to the needs that drive technology evolution. Ideally we would like these needs to be generated by some 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 the circuits might achieve. Because each need is a specific logic function, each has its own truth table. Circuits that are useful toward satisfying one of the given needs are encapsulated into new components and become new technologies – new elements that serve as building blocks for possible further combination.

| 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 |

Table 1: Needs are common logic functions for $2 \leq n \leq 15$, $1 \leq k \leq 8$, and $2 \leq m \leq 7$.

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. The behavior of a component is described by a Boolean function which specifies its truth table, that is, the logical value of the output pins as a function of the logical values of the input pins. This function is the *phenotype* of a component; its *genotype* is the architecture or internal circuit that realizes this function. Many different genotypes can generate the same phenotype.

Each evolutionary step in our computer experiment randomly creates a new circuit to be tested. This is put together by randomly wiring together between 2 and 12 circuits selected from all previously existing technologies according to a *choice function* which 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 satisfy each of the goals (have least incorrect entries in their truth tables). Each candidate circuit is tested against these to see if it improves upon them. If it does, the novel circuit is encapsulated – it becomes a new technology – and replaces the previously best circuit for a particular need. A need is *satisfied* if a new technology with its exact truth-table has been found. A novel circuit may also have the same phenotype (truth table) as an existing circuit and "cost" less. (The cost of a circuit is determined by the number of its components and by *their* respective cost.) In this case, the novel circuit is encapsulated and replaces

the circuit it improves upon both directly and in all circuits where it is used as a component. A newly created circuit of course cannot replace one of its own components. Details of our implementation of these general algorithmic steps are listed in a section below.

In its essence the experiment is simple. Starting from a set of primitive technologies and given needs, in each evolutionary step novel circuits are randomly created from existing ones. If a novel circuit improves on a previously best one for a particular need it becomes a new technology and replaces that technology. And if its function is identical to that as an existing technology but costs less, it replaces that technology. In this way the set of encapsulated technologies builds out. In our particular experiment, we use only one primitive, a `nand` gate, with phenotype $\neg(x \wedge y)$. Useful components are named (e.g. `tech-256` or `full-adder`) 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`).

The correspondence to the real world requires some comment. New technologies in the real world are indeed combinations of existing ones, but these days they 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 which 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. [2] 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. [5]

## Experimental Results

The most complex circuits "invented" within 250,000 steps in our basic experimental design were `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. 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.

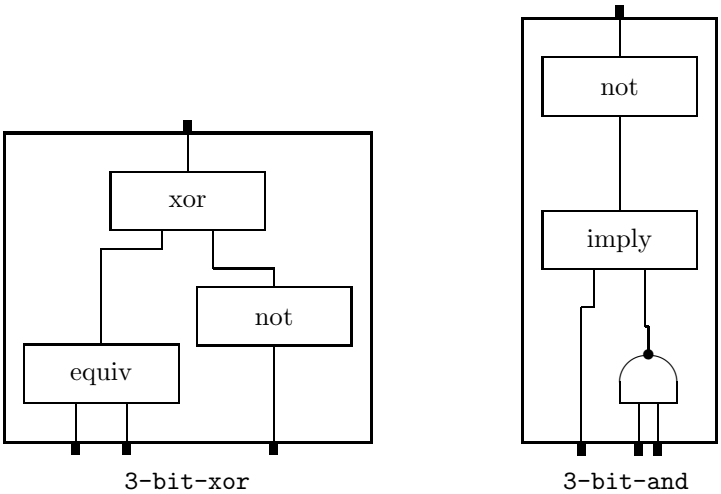


Figure 1: Two circuits "invented" for simple goals.

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.

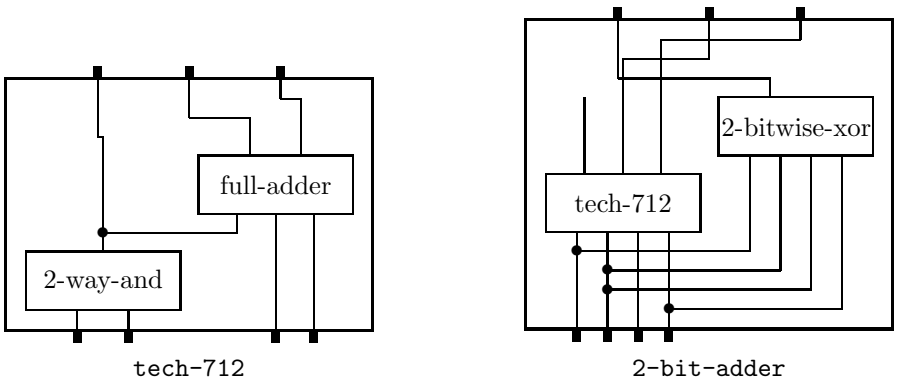


Figure 2: `tech-712` is useful towards satisfying the `2-bit-adder` goal since the two high-order bits are computed correctly. (The low-order bit is on the left. For multi-bit adders, input bits are interleaved.)

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 towards 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. 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.

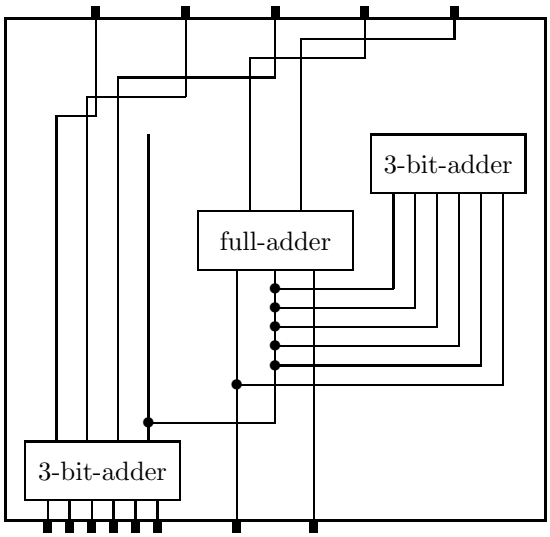


Figure 3: A 4-bit adder circuit with an unconnected module.

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 $2^{m2^n}$ 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.

## The Buildout of Technologies

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.

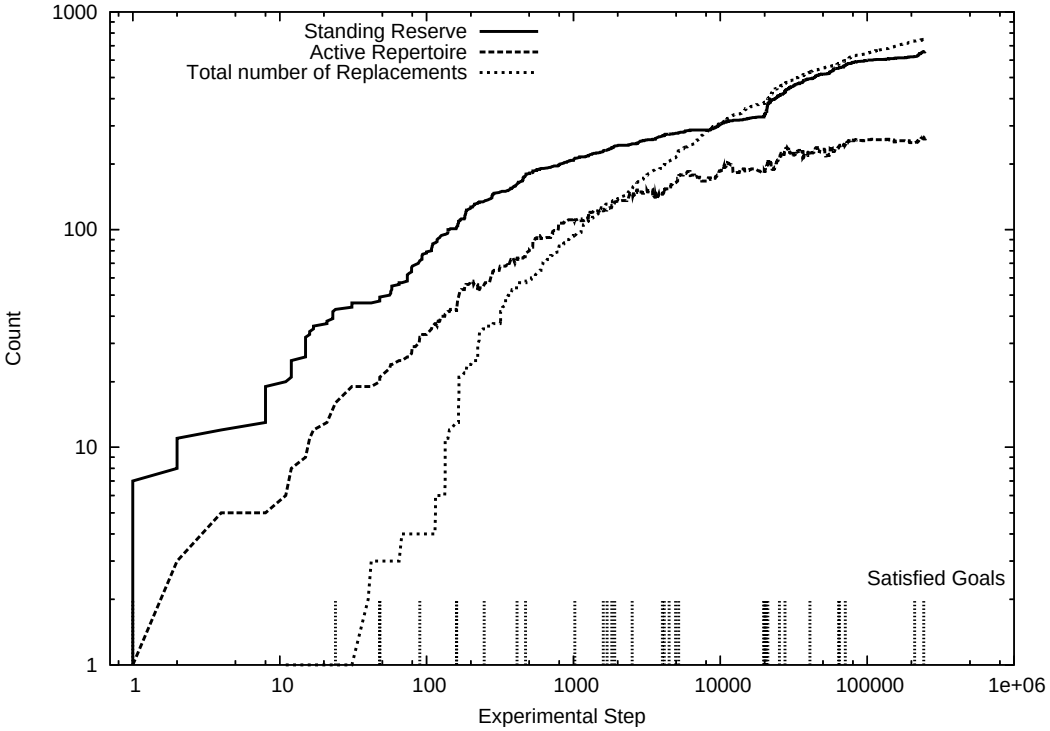


Figure 4: The standing reserve, by definition, grows monotonically. The same is not true for the active repertoire since new inventions may improve upon and replace several existing ones.

Tick marks along the time axis of Figure 4 indicate when one of the needs has been satisfied. The experiment runs for some time without meeting any goals exactly, 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, 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 buildout of species in biological evolution. [5]

The order of invention makes a difference. While negation is a simpler function than implication, it happens 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 a functionally equivalent but more efficient implication.

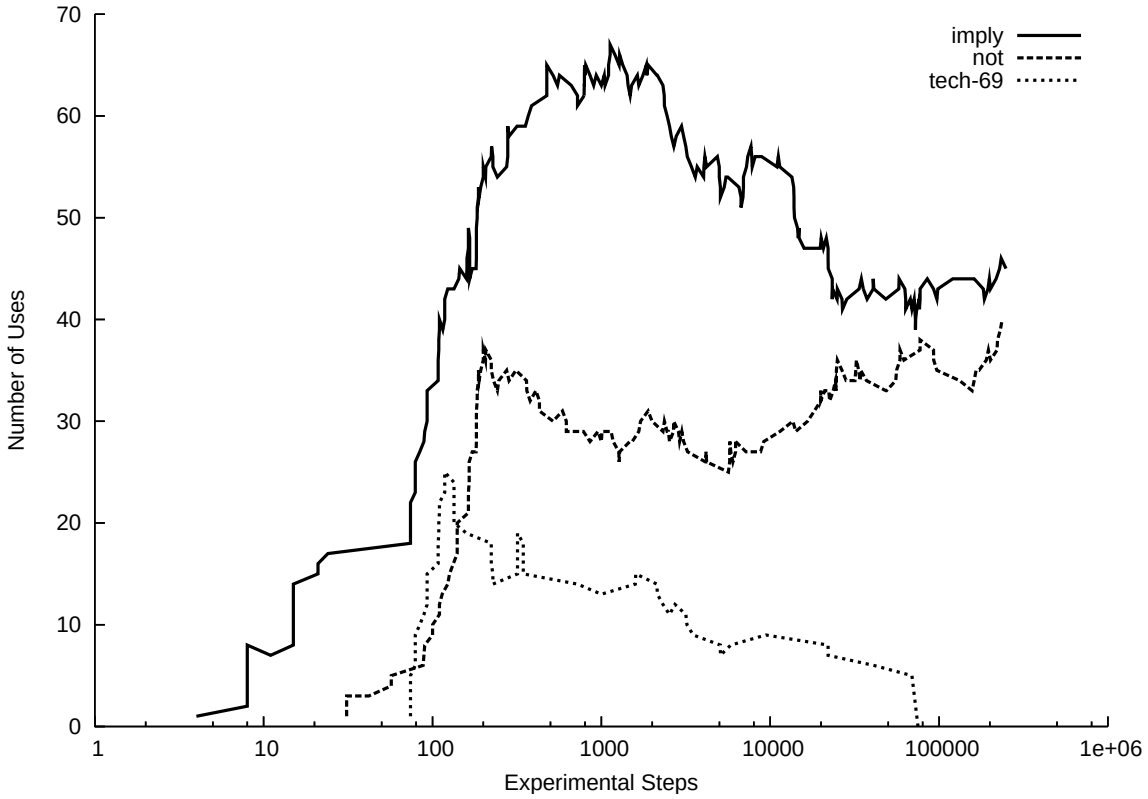


Figure 5: Implication, being invented before negation in this example, is used more heavily. Usage declines over time as better technologies are invented.

There is a tradeoff 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 all of the goals listed in Table 1, it 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 does not develop. 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 at each step only circuits containing fewer than 12 existing components are considered defines a set of experimental circuits – a large number – near to those already existing, one of which is likely to emerge next. We can think as the "adjacent probable" (cf. [4]). They form a probabilistic cloud that surrounds the existing technologies and that gradually leads 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 in combination. What if we choose complicated goals that are not easy to realize by relatively simple 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.

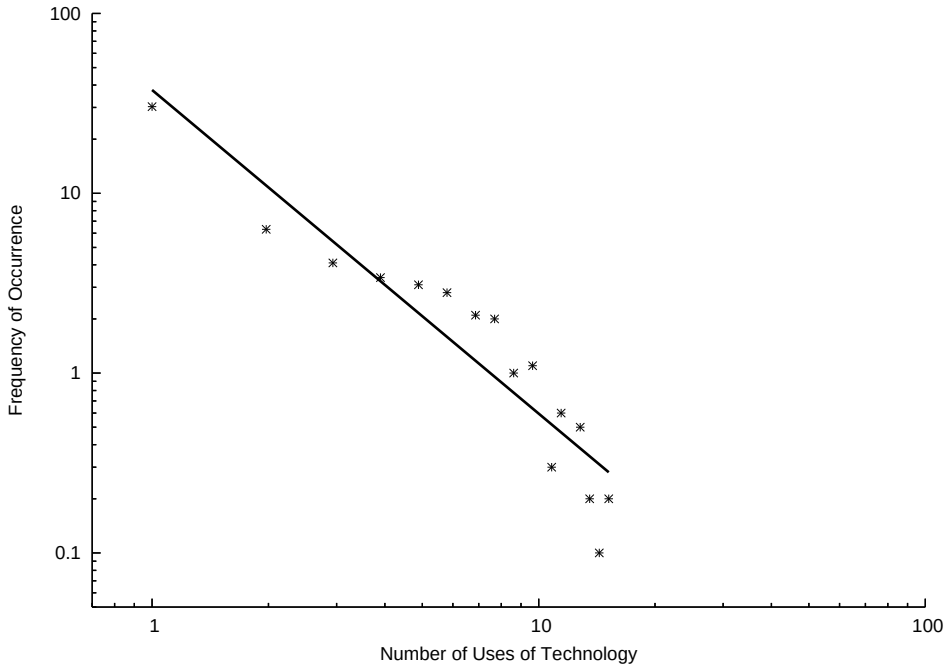


Figure 6: Very few key technologies are used heavily to directly construct new ones. The plot shows the average over 10 experiments at their termination of 250,000 steps each.

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 which use it directly or indirectly as a component). Replacement is immediate in our algorithm.

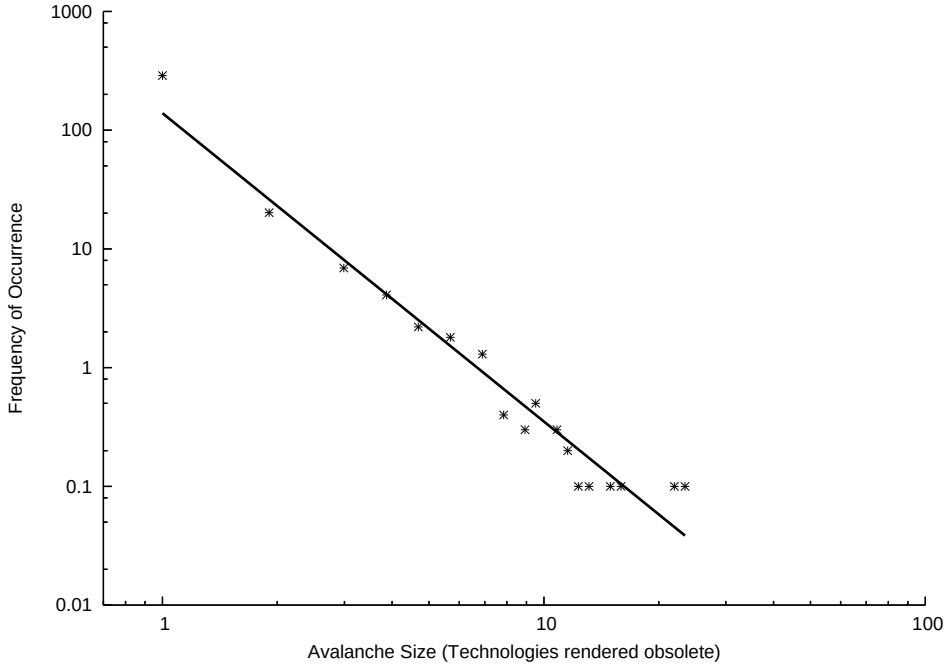


Figure 7: Gales of destruction (or avalanches of collapse in use).

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 `tech-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 called such avalanches "gales of destruction" [6]: when the automobile replaces the dray horse, horse husbandry collapses; when horse husbandry disappears, so does blacksmithing. 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. [1]

# Conclusion

Our experiment shows that technology can bootstrap itself up from extreme simplicity to a complicated level, within an artificial system. In the real world, of course, novel technologies are not usually constructed by random combination (although this is increasingly the case), 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 that in this aspect technology is subject, at least approximately, to similar statistics as earthquakes and sand-piles. Our model also shows that the buildout of technology bootstraps itself from the existence of earlier technologies constructed for intermediate or simpler needs. The same can be said for technology in real life. Without components needed for simpler electronic functions such as amplification and wave generation, radar could not have been invented.

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. Our algorithm, if we view it 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; instead it attempts to create a toolbox of useful functionalities that can be further used for further problem solving. And 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 setting 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 $C_2$ is either connected randomly with an input of $C_1$, or an output of $C_1$, or it becomes an input of the combination. The step terminates when a useful circuit has been found, or when a limit to the number of 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) [3]. 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 all outputs $g$ 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.

## Acknowledgements

This work was mainly carried out at and supported by FX 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. That workshop was supported by the Packard Foundation and the Thaw Charitable Trust. The notion that invention proceeds by combination goes back to Schumpeter [6]; Arthur thanks John Holland and Stuart Kauffman for many discussions on this idea over the years, dating back to August 1987. (Kauffman's subsequent development of these thoughts is summarized in [4].) Both authors thank Eleanor Rieffel for helpful discussions in the early phases of this paper. The idea to use logic circuits as our experimental medium was inspired by Lenski et.al. [5].

# References

[1] Per Bak and Kurt Wiesenfeld. Self-organized criticality: An explanation for $1/f$ noise. *Physical Review A*, 38:364, 1988.

[2] Annette Beck-Sickinger and Peter Weber. *Combinatorial Strategies in Biology and Chemistry.* Wiley, Chichester, England, 2001.

[3] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[4] Stuart Kauffman. *Investigations.* Oxford University Press, New York, 2002.

[5] Richard E. Lenski, Charles Ofria, Robert T. Pennock, and Christoph Adami. The evolutionary origin of complex features. *Nature*, 423:139–143, 1988.

[6] Joseph A. Schumpeter. *The Theory of Economic Development.* Transaction Publishers, New Brunswick, NJ, 1911/1934/1996.