Coevolution in the Computer: The Necessity and Use of Distributed Code Systems

by

George Kampis

Cognitive Science Laboratory, Dept. of Ethology
ELTE University Budapest, Hungary

and

Dept. of Theoretical Chemistry
University of Tübingen, Germany

Abstract

If evolution is the genetic change of a population due to a given selection force, then coevolution should be defined as the production and application of recursively defined selection forces in a semi-closed network. Whereas it is not difficult to model the first (in terms of optimization and adaptation), the second poses a paradox. This paradox has to do with a "complexity bottleneck" experienced when formulating the problem: in a system of limited complexity (such as in the starting phase of the evolution), a system with higher complexity should be anticipated, in order to allow for a simulation of the new selection forces and the new adaptations.

We suggest a solution to this problem, based on the new idea of a "distributed code system". Distributed code systems can be conceived as populations of computers where the coding scheme of a computation (that is, the "reading frame" of a given program) is computed by other machines, on a mutual basis. This makes it possible for the system to introduce new interactions automatically, without having to pre-specify them in an encompassing initial definition. Distributed code systems offer, therefore, a mechanism for the dynamic production of information. Realizations in terms of computer models are discussed.

Life's perhaps most striking capability is its *evolvability*. It is no wonder the working definitions of life often proceed by defining evolution first and deriving the concept of life from that (Maynard Smith 1975, Dawkins 1986, Ray 1992). Consequently, "Artificial Evolution" should be one of the most important topics in AL. In this paper we deal with a fundamental question that emerges next: By what methods can we implement evolutionary processes in a computer?

I. Evolutionary Programming

Evolutionary programming is the idea of mimicking evolutionary development in order to obtain programs in the cheap way. The most recent major work being that of Koza (1992), this problem has been of concern for a long time. Not counting early philosophical discussions, the true story begins, as does so much else, with John von Neumann (1966). He was the first to ask in the language of mathematics, whether computers can evolve. He pointed out that the question cannot be solved

at once in this broad generality. He suggested to consider better questions like: Can a series of machines be produced automatically, such that all elements in the series differ from each other, from a behavioral point of view? Or, alternatively, can the successive machines produced in such a series show better and better performance in a task environment, according to some suitable criterion? Von Neumann was able to answer these questions to the affirmative, by constructing and analysing what is today known as the "von Neumann self-reproducing cellular automaton" (Burks 1971). Of course, much depends here on the chosen criterion for performance. For instance, J. Myhill (1971) has been able to show only much after Neumann's death that program evolution is possible under such mathematically plausible performance measures, as the one expressing the number of theorems a given program as a formal system can prove. From the algorithmic point of view, these works generated significant interest. A distinct approach, adopted in the recent GA industry and originated in works like (Holland 1975, Eigen 1971, Edelman 1987) is based on the population genetic paradigm of evolution conceived as optimization (an idea that goes back to Wright's concept of *adaptive landscape*). This approach too defines an entire field of study of its own. And above all, there is Tierra (Ray 1992), an important new version of Vyssotsky's *"Darwin"*, a Bell Lab fun product in the early sixties (AlephNull 1972), or of Core War games, especially of the variants capable of mutation (Daiber 1988), a principle increasingly utilized in the development of real-life computer viruses (McAfee and Haynes 1992).

These origins define the current enterprise of evolutionary modeling as one clearly related to three categories: direct computational (i.e. *instructional* or *ortho-evolutionary* (like Myhill's systems), direct selectional (like GAs), or mixed selectional-computational (such as Tierra, where programs both perform actions and are subjects of selection). With these developments at hand, we have examples for different types of systems that show characteristics of evolution.

At the same time, there exist a few warnings. Despite all this success, the operating principles of evolution may considerably differ from those of, say, "traditonally conceived computing". Evolvability/programmability tradeoffs identified by Conrad (1985), information-theoretic considerations of Cariani (1989) or problems of definability theory (Kampis 1988, 1991) indicate that some of the most basic issues will require further study.

II. Evolution and Coevolution

We shall examine the selectional or selectional-computational approaches more closely, in order to point out a difficulty, and to offer a solution to it. Let us first specify a little more concretely, what should be meant under evolutionary modeling or evolutionary programming.

Darwinian evolution has two main components: a *selective force* and a *variable population*. Evolution occurs as an interaction of the two. It is known since long ago that this interaction can be bidirectional: that is, typically, not only populations, but also selection constraints are affected by evolutionary change. Hence evolution is, to a great extent, its own product. This is true in a double sense. Organisms play a role in

- the production of the environment (Van Valen 1973, Stenseth and Maynard Smith 1984): for a given organism, the relevant environment mostly consists of other organisms (or rather, of other species), which in turn are products of earlier evolution in their earlier environment.
- the selection of the environment (Lewontin 1983): of a given encompassing en-

vironment, the genetically determined life strategies of organisms select the relevant part; for instance, genes can determine whether their own relevant environment is in in the air, on the ground, or in the soil.

Any evolutionary event potentially involves the change of these factors. Consequently, evolution consists of two threads that go hand in hand: we have *change of populational composition* (or adaptation), and *change of selective force* (or self-organization)† For simplicity, it is convenient now (but also raises well-known unsolved problems) to think of the first as *microevolution* and of the second as *macroevolution.*

The issue on which we would like to focus arises when we understand that *a proper modelling of evolution is not only a modelling of the origin of specific adaptations as consequences of selection forces, but also of the modelling of the origin and production of selectional forces as consequences of adaptations (or whatever).* Let us hasten to note that this complicated aspect of evolution is vastly neglected in the computational literature, and is sporadic in the biological one. It is hard to find any work at all with a direct reference to this (among the few exceptions are Rosen 1991, Kampis 1991, Goertzel 1992, or — in population ecology — Roughgarden 1979). And, even worse than that, there is a serious new methodological difficulty involved here, waiting to be discovered.

III. The Coevolutionary Trouble

What we said so far depicts coevolution as a *bootstrapping process* at the ecological level. To analyze the situation, it is instructive to think of the adaptive part of evolution as some kind of *problem solving* in a given space. Obviously, the first step should be the setting up of a problem. But, by the hypothesis of coevolution, this problem must be a product of the previous solution which then corresponds to a former problem, and so on; ultimately, it has to be all generated by the first problem and the first solution, which are, by the second hypothesis that this is an evolution process, extremely simple compared to the end result. How is this possible‡?

Anybody with even a minimal skill in mathematical modelling will no doubt share the intuition that a solution can never be more complex than the problem it solves. In fact if the solution "fits" to the problem, they behave like a key and a lock, so their relationship is closest to that of an *equivalence*, if anything. This intuition can be made quite precise mathematically, by using algorithmic complexity theory (Li and Vitanyi 1990, Löfgren 1987, Kampis 1991). In these terms, in order for the solution complexity to change, the task complexity must also change; in particular, for the former to become more complex, the latter has to become more complex. No short-cuts are permitted, no royal road. The perhaps most elegant way of putting these sorts of questions is by the formulation suggested by Chaitin (1975): Can you prove a 100-pound theorem in a 10-pound axiom system? (Housewife version: Can you buy 100 dollars for 10 dollars?). Put this way, we all kmow the answer to be negative.

---

† Closely related are the concepts of "recursive evolution" and "deuteroadaptation" by Rössler (1984) and Conrad (1983).

‡ To make our motivation quite clear: in the context of coevolution we speak of using nothing but a seed to let a whole evolutionary process construct itself. A natural but somewhat distant possibility for experiencing such a process would be colonizing a virgin planet. Or, one may think, by analogy, of the suggestions of A.C. Clarke's *Rendezvous with Rama* (1973), where (in my reading) the point was that an initial triggering event, in this case the entering of the ship, could launch an autonomous development that starts from virtually nothing. Of course, that is precisely what life did on Earth.

Besides such mathematical obstacles, but perhaps not quite independently from them, hidden in our problem there is a philosophical one: can an effect be "bigger" than the cause†?

We conclude that there is a well-identifyable *complexity bottleneck* in the modelling of a genuine coevolutionary process. If a highly evolved system contains more information than the original "primordial soup", the question is, where did this information come from, in the first place? In the realm of traditional algorithms no mechanism for "bootstrapping" is permitted; we have to pay the full prize (in cash advance payment) for everything we need. This just won't produce money. Formulated as a "coevolution paradox": evolution in a closed system is not possible, for such systems cannot generate new and more complex tasks, only new consequences of the old ones. (It's no accident that this paradox is not unlike some problems familiar from AI research and cognitive science.)

IV. From Selective to Distributed Code Systems

Whereas the above reasoning seems to be fundamentally correct, and it poses a serious challenge to all evolutionary computer models, nothing forces us to stop here, and to accept it as a merely negative result. Indeed we propose here an approach which circumvents this problem in a sense, if not by means of some brute force attempt such as the suggestion of writing "cleverer" programs would be, because that is precisely what does not seem to be possible, but by *recursively introducing new functional information* in the evolving system. Thereby we can get new evolution potential. And the trick is that this can be done automatically, though not quite *algorithmically* — not in the usual restricted sense at least.

Let us give out the idea first. Algorithms are conceived, ultimately, as recursive functions. These are, in turn, made equivalent to programs completely definable as syntactic strings. So after all, classical programming is nothing but typography. A precondition to this simplification procedure is the criterion that programs of the same universe of discourse must all have the same semantics. What we suggest is simply to manipulate exactly this aspect of algorithms, in order to achieve higher flexibility. But let us proceed sequentially. We shall first spend some time with semi-formal questions that make the road for the result we seek to formulate.

IV.1. "Central" Encodings

Of importance is the fact that a *selective system* such as those based on $P(t) = \{x_1(t), x_2(t), \ldots, x_n(t)\}$ where P is a population of structures x_i that are candidate solutions to an objective function f always operate by means of a fixed *"central encoding"*.

What does that mean? By a central encoding we mean that the complete code is available, in the same unchanged form, at every one point of the system. More precisely, consider now "artificial organisms" x_i as instances of program strings $p_1, p_2, \ldots p_i, \ldots, p_n$ (essentially in the same sense as suggested by Stahl and Goheen (1963), or Laing (1989), or as found in the Tierra-like systems). Consider furthermore a selection situation where these programs are executed in order to achieve a goal. We choose a graphical representation following Smith (1969) and Kampis and Csányi (1987):

† Many people will recognize this wording to be related to the infamous question: "Can God create a rock so heavy that even he himself cannot lift it?" Unlike this one, where paradox arises as a consequence of verbal ill-statedness, the original question seems to permit a solution.

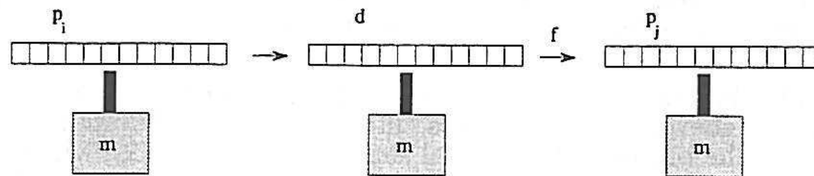

Figure 1.

The illustration shows a system where program execution produces data, which will be externally evaluated. This leads to the formation of new programs (by means of mutations of otherwise. Such details don't make a difference; nor does it matter whether we allow the computation of $f(p_i)$ to be part of a further embeddig program not depicted here.). Here the program and the data can be thought of as strings on a tape, and the machine as a processor unit that inspects and processes exactly one square of this tape in every time step.

The graphical representation makes it explicit that the process depends on two distinct parts (just as in our discussion of Darwinian evolution!): on the program p_i and on the interpreting machine m. It is natural to require m to be an instance of a Universal Computer (i.e. any machine for which Turing's simulation result holds), but that still leaves plenty of freedom for this machine's actual choice. In particular, we have yet to fix a *translation table* between words of the program symbols and those of the semantic primitives of m. That is where the encoding enters. For a concrete example this translation may look somehow like this:

p syntax	m syntax	m semantics
0101	*abab*	`write 1 to the tape and stay where you are`
1100	*gfca*	`write 0 to the tape and move right`

etc., in which case the code would be $C = \{01 \Rightarrow ab, 11 \Rightarrow gf, 00 \Rightarrow ca, \ldots\}$, and so on, and the machine would always have to read four blocks before turning to execution.

Fixation of such a translation rule (which in the simplest case can be an identity mapping and may thus mislead people into thinking it's not there) is necessary in order to have a definite meaning for the p_i-s. So, the operation of defining an encoding C must be a prerequisite for obtaining a well-defined expression $f(p_i)$ in the end. This amounts to expressing the "phenotype" (that is, the performance) as a function of the "genotype" (the program)†: that is, to have $f(p_i) = h(C(p_i))$ for some fixed function h. (In the limiting case when C is an identity, $h = f$.)

IV.2. Encodings and Coevolution

Having spent some time with the development of the above concepts, it can be readily seen that the idea of central encoding is more general than just a way of looking at Turing Machines. Such encodings have unexpected side effects. Consider, for instance, the quite natural idea to extend Figure 1. towards a system capable of coevolution by allowing the p_i-s to produce new programs p_j with new environmental functions f_j directly coupled to them. In other words, we can allow the environmental selection function to become simply part of a program:

† An interesting biological question concerning the nature of ontogenetic transformation is implicit at this point.

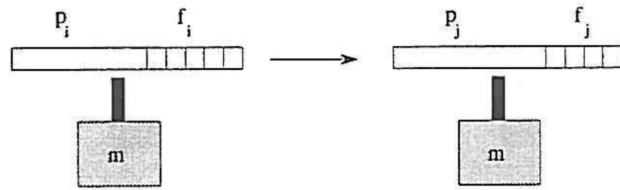

Figure 2.

Is this now a good model for a coevolution process? Not quite. What this amounts to is to try to change the evolutionary tasks upon execution, while keeping the encodins (i.e. using the same machine as before). Now here instead of f we get a set f_i, and that may allow for a more complicated behaviour indeed, but at a closer look, this solves nothing. By the assumption of invariant encoding, this amounts to having a set of *preassigned* selection problems of exactly the same type as earlier. This is unavoidable as the encoding must relate the f-s to certain *a priori* physical actions. We could have as well started with this $\{f_i\}$ instead of f in Figure 1.

In short, in such a system no self-selecting property can be expected to emerge. The lesson is that is a system with a centralized code one can at most transform a simple GA-style function optimizer into a *path-dependent multiple criteria optimizer* (for it can depend on the given realization which tasks are on and which are off), but that is not coevolution yet. In other words, *as long as a machine keeps working in one and the same way, it can't help but work on pre-defined and pre- encoded (and anticipated) problems*†. If we want to solve the coevolution paradox, we have to solve the problem of the brittleness of computers. The same old problem in new clothes.

IV.3. Distributed Code Systems

And finally, we come to the idea for which we have been preparing all along. It depends on a method by which we can exploit about as much of the possibility of "tinkering" with the definitions of computations as possible.

We suggest to use programs for computing new encodings and machines for interpreting programs. When interpreted differently, one and the same "program" (more precisely, the same program string) can behave like several different programs. The idea is to let programs interact so that the one defines the machine (or rather the encoding) on which the other runs, and so on, on a mutualistic basis: all information comes from within the system process.

As a result, the code will be distributed over the evolving programs, so that at no point will we have exactly the same information, and at no point is this information complete. Only the population of programs as a whole determines what encodings are valid for the system, and even that can change as it is relative to the current state.

How this is achieved is very simple (Figure 3.).

† This same conclusion can be reached by various trains of thought.

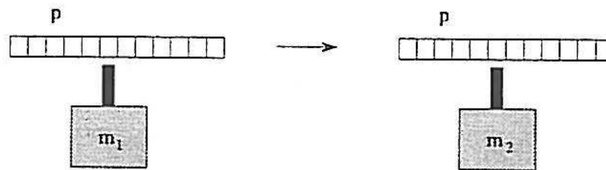

Figure 3.

Let $p_i \in S^*; S = \{0,1\}; i = 1,\ldots,n$. The Figure shows an example where $n = 2$ and $p_1 = p_2$. That is, here we have one program with two machines m_1 and m_2. Obviously, then, $m_1 \circ p = \varphi_i(.), m_2 \circ p = \varphi_j(.), i \neq j$, where φ_i is the i-th partial recursive function in an arbitrary Gdel-numbering and $\circ$ is the sign for composition. What can now turn this system into a Distributed Code System is a relationship

$$m_1 = H(m_2 \circ p)$$
$$m_2 = H(m_1 \circ p).$$

The definition can be easily extended to arbitrarily many m-s and p- s, the only criterion being that all indices should occur exactly once at the left hand side and at the right hand side of the equality sign, respectively. (More general definitions that use H-s with several arguments, or ones that allow function H to be modulated by the computations could also be entertained. Such constructs are necessary in order to be able to extend the pool of programs, i.e. to let n increase without having to redesign the system completely. However, we do not discuss these technicalities here.)

The suggested scheme realizes an unusual coupling between *hardware* and *software*, two worlds so neatly separated in systems with a central code. (We can, of course, revert the argument: *because* the hardware and the software is so well separated in the usual machines, one can introduce a central code that covers all pieces of program in them.) In our scheme, new progams help build new machines†. In this way, they can in principle offer the kind of flexibility that is needed in order to produce new tasks. In fact, when using a new method of reading, every task will appear as genuinely new.

V. Distributed Code Systems versus Computations and Molecular Systems

By its definition the notion of a distributed code system goes beyond that of computation understood in the usual sense. As emphasized repeatedly, every definition of computation must start with laying down a basic syntax and semantics, none of which can be changed afterwards. In this sense, our result means that evolution is very different from computing.

On the other hand, a distributed code system can be easily simulated on an ordinary computer. All we need to that is an embeddig program that lets us simulate machines together with their programs, and then, of course, to simulate new machines with new programs, and so on.

This can be imagined somewhat in the style of von Neumann's famous self- reproducing cellular automaton. There the point was to achieve self- reproduction so

† *Emulation* can be a key word here. In a sense, it is possible to build new computer hardware entirely from within software, that is, without having to take physical components apart.

that *not only the target configuration but also the machine which performed the reproduction was reproduced.* Here the situation is quite analogous, with the exception that not always the same machine is produced over and over again. Otherwise, the parallel is complete: *in the coevolutionary context, not only new programs should be produced but also new machines that interpret old programs.*

A natural counterpart of an abstract distributed code system is a macromolecular system. Macromolecular reactions produce, in general, new macromolecules. These in turn can often make use of hidden or yet undefined properties of their fellow molecules. The information content of these molecules is actively manipulated by the system of which they are part. There is a correspondance between this mechanism and the formal structure we have been discussing above. Examples that utilize related mechanisms range from *shifting reading frames* (in the sense used in molecular evolution) to *function change* and to Jacob's *"evolutionary tinkering"*. Discussion of these applications is beyond the scope of this paper, however.

VI. Analysis of Tierra-like Systems and the Use of Distributed Code Systems

In this final section we use Tierra as a paradigmatic example to illustrate some points of this paper. The amazing variety of complex forms that can evolve in Tierra-like systems is bound to important limitations. It is now easy to formulate what they are.

In the current versions of the model, the digital organisms all compete for the same room, therefore, ecologically speaking, they belong to the same species (whereas real elephants and mice are not in competition as they have different needs). The various behavioural roles (parasitism, coalitions, etc.) developed by members of the evolving abstract population resemble more to genetic polimorphisms in an ESS-like intraspecies situation or to adaptations of social structure than to evolution on the large scale.

More importantly, that there is just one room means that *there is just one task*, which is essentially invariant and is externally given. This task cannot be modulated by the evolution process. That nevertheless various strategies exist in the system is a consequence of the emergence of subgoals rather than of new domains for development. This can be best seen on the fact that in these systems *there are no producers*, only consumers†. By producers we do not necessarily mean producers-in-the-ecological-sense (which are, ultimately, the plants). Plants produce energy-rich substances other organisms can consume. This is an important phenomenon. However, we can also speak of "producers" with respect to the role species play in niche-theory. In this sense, a "producer" is one that makes the opening or the occupation of a new niche possible; in our present terminology, it is one that defines a new task field.

Without this kind of productivity, one of the most important biological mechanisms that makes the emergence of multi-species structure possible cannot be incorporated. This mechanism can be called *"competition avoidance"*. That not everybody must compete with everybody else helps generate evolutionary complexity by letting goups of variants to differ from each other, without being punished or rewarded. That is why the generation of new tasks is important. Incorporation of a mechanism like that of distributed code systems would therefore seem to be an essential requirement for the takeoff of even an abstract evolution process.

† The idea in this form was first formulated in a discussion of the author with E. Minch, Stanford.

Acknowledgments

This work was supported by research grant OTKA 2314 of the Hungarian National Foundation for Scientific Research. The paper was written during the author's stay at the University of Tübingen, Germany. The author wishes to thank the hospitality and help of Professor O.E. Rssler. The manuscript was produced using Eberhard Mattes' EmTeX package and Thomas Rokicki's dvips driver.

References

- Aleph Null 1972: "Darwin", *Software-Practice and Experience* 2, 93- 96. (A.N. was a columnist for SPE's "Computer Recreations").
- Burks, A.W. (ed.) 1971: Essays on Cellular Automata, University of Illinois Press, Urbana.
- Cariani, P. 1989: On the Design of Devices with Emergent Semantic Functions, Ph.D. dissertation, Dept. of Systems Sci., SUNY at Binghamton. Available through University Microfilms, Ann Arbor, MI.
- Chaitin, G.J. 1975: Randomness and Mathematical Proof, *Scientific American* **232** 47-52. Also: Information, Randomness and Incompleteness, World Scientific, Singapore, 1987.
- Clarke, A.C. 1973: Rendezvous with Rama, Harcourt Brace and Jovanovich, New York.
- Conrad, M. 1985: On Design Principles for a Molecular Computer, *Comm. ACM* **28**, 464-480.
- Conrad, M. 1983: Adaptability. The Significance of Variability from Molecules to Ecosystem, Plenum, New York.
- Daiber, A. 1988: Core War Genes: A New Standard, *The Core War Newsletter*, Spring 1988, 4-10.
- Dawkins, R. 1986: The Blind Watchmaker, W.W. Norton, New York.
- Edelman, G.M. 1987: Neural Darwinism: The Theory of Neuronal Group Selection, Basic Books, New York.
- Eigen, M. 1971: Selforganization of Matter and the Evolution of Biological Macromolecules, *Naturwissenschaften* **58**, 465-505.
- Goertzel, B. 1992: The Evolving Mind, Plenum, to be published.
- Holland, J. 1975: Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor.
- Kampis, G. 1988: Kampis, G. 1988c: On the Modelling Relation, *Systems Research* **5**, 131-144.
- Kampis, G. 1991: Self-Modifying Systems in Biology and Cognitive Science: A New Framework for Dynamics, Information and Complexity, Pergamon, Oxford.
- Kampis, G. and Csnyi, V. 1987a: Replication in Abstract and Natural Systems, *BioSystems* **20**, 143-152.
- Laing, R. 1989: Artificial Organisms: History, Problems, and Directions, in: Artificial Life (ed. Langton, Ch.), Addison-Wesley, Reading, Mass., pp. 49-62.
- Lewontin, R. 1983: The Organism an the Subject and Object of Evolution, *Scientia* **118**, 65-82.
- Li, M. and Vitanyi, P.M.B. 1990: Kolmogorov complexity and Its Applications, in: Algorithms and Complexity. Handbook of Theoretical Computer Science. Vol A. (ed.: Van Leeuwen, J.), MIT Press, Cambridge.
- Löfgren, L. 1987: Complexity of Systems, in: Systems and Control Encyclopedia (ed.: Singh, M.), Pergamon, Oxford, pp. 704-709.
- Maynard Smith, J. 1975: The Theory of Evolution, Penguin, London.
- McAfee, J. and Haynes, C. 1992: Computer Viruses, Worms, Data Diddlers, Killer Programs, and Other Threats to Your System, St. Martin's Press, New York.

- Myhill, J. 1971: in Burks 1971 (this is a reprint of Myhills 1963 original paper)
- Neumann, J. von 1966: Theory of Self-Reproducing Automata (completed and edited by A.W. Burks), University of Illinois Press, Urbana.
- Ray, T. 1992: Tierra & Tierra documentation. (Although there have been numerous writings on Tierra in the press, it is hard to find a quality publication. Hence, currently the texts that accompany the PD versions of the software are of primary importance. The whole package can be pulled down from the Net through: life.slhs.udel.edu)
- Rosen, R. 1991: Life Itself, Columbia University Press, New York.
- Roughgarden, J. 1979: Theory of Population Genetics and Evolutionary Ecology: An Introduction, Macmillan, New York.
- Rössler, O.E. 1984: Deductive Prebiology, in: Molecular Evolution and Protobiology (ed. Matsuno, K., Dose, K., Harada, K, and Rohlfing, D.L.), Plenum, New York, pp. 375-385.
- Smith, A.R. III 1969: Cellular Automata Theory, Technical Report No. 2, Digital Systems Lab., Stanford University, California.
- Stahl, W.R. and Goheen, H.E. 1963: Molecular Algorithms, *J.Theor.Biol.* **5**, 266-287.
- Stenseth, N.C. and Maynard Smith, J. 1984: Coevolution in Ecosystems: Red Queen Evolution or Stasis?, *Evolution* **38**, 870-880.
- Van Valen 1973: A New Evolutionary Law, *Evolutionary Theory* **1**, 1-30.