

Revised 5/11
1991

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--91-896

DE91 009954

TITLE: INFORMATION DYNAMICS OF SELF-PROGRAMMABLE
MATTER

AUTHOR(S): CARSTEN KNUDSEN, RASMUS FELDBERG and
STEEN RASMUSSEN

SUBMITTED TO: Proceedings of "Complex Dynamics and Biological
Evolution" held August 6-10, 1990 in Denmark

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution or to allow others to do so, for U.S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

MASTER



INFORMATION DYNAMICS OF SELF-PROGRAMMABLE MATTER

Carsten Knudsen,[†] Rasmus Feldberg,[†] and Steen Rasmussen[‡]

[†]Physics Laboratory III and
Center for Modelling,
Nonlinear Dynamics and
Irreversible Thermodynamics
Technical University of Denmark
DK-2800 Lyngby
Denmark

[‡]Center for Nonlinear Studies and
Complex Systems Group,
Theoretical Division
MS B258
Los Alamos National Laboratory
Los Alamos, New Mexico 87545
USA

ABSTRACT

Using the simple observation that programs are identical to data, programs alter data, and thus programs alter programs, we have constructed a self-programming system based on a parallel von Neumann architecture. This system has the same fundamental property as living systems have: the ability to evolve new properties. We demonstrate how this constructive dynamical system is able to develop complex cooperative structures with adaptive responses to external perturbations. The experiments with this system are discussed with special emphasis on the relation between information theoretical measures (entropy and mutual information functions) and on the emergence of complex functional properties. Decay and scaling of long-range correlations are studied by calculation of mutual information functions.

INTRODUCTION

A fundamental feature of living organisms is the ability to compute, or process, information. Information processing takes place over a wide scale of complexity, ranging from the simple processes by which an enzyme recognizes a particular substrate molecule, to complicated feedback regulations containing many different levels of information processing, to the extremely complex processes of the human brain.

An example of biological information processing in a feedback loop is provided by one of the negative feedback loops described by Sturis et al. (1991) in their model of oscillatory insulin release. The feedback control can here be divided into at least four different components: (i) an increased amount of glucose in the plasma stimulates

insulin production in the pancreas and secretion of insulin into the plasma; (ii) from the plasma, insulin diffuses into the interstitial fluid; (iii) here insulin molecules attach to receptors on the surface of the cell; and (iv) insulin activated receptors enhance the uptake of glucose by the cells, which, of course, implies a decrease in glucose outside the cells.

This loop involves simple biochemical information processing such as the recognition of receptors by insulin molecules and the subsequent attachment of the molecules. In addition, there is a more complex information process involving active transport of glucose over the cell membrane facilitated by a cascade of conformational changes in the cell membranes protein molecules.

The most complex kind of biological information processing is probably the abstract and creative symbolic information processing in the human brain. Simple aspects of these processes are subjects of numerous investigations. In particular a variety of models of artificial neural networks have recently been proposed (see for example Palmer 1988, and Touretzky 1990).

The theoretical foundation of information processing in man-made machines can be described in terms of computation theory (Hopcroft and Ullman 1979). In computation theory, a number of different formalisms exist, of which the Turing machine (TM) for historical reasons is the most well-known. The Turing machine has been examined thoroughly by mathematicians and computer scientists because it is believed to be able to perform the most general type of computation, universal computation. This conjecture is known as the Church-Turing thesis (Hopcroft and Ullman 1979).

Besides Turing machines, several other systems have been shown to support universal computation, including cellular automata, the λ -calculus, Post systems, the hard billiard ball computer, general recursive functions, classifier systems, partial differential equations, von Neumann machines, and even C^∞ maps of the unit square onto itself. It has been shown that each of these formalisms is equivalent, since any one of them can simulate any other.

The information processing found in biological systems seems to be different in nature from that of a Turing machine. In fact, none of the above mentioned computational paradigms capture the full spectrum of biomolecular information processing. A fundamental property of computation in biomolecular systems arises from their ability to alter or program themselves. Self-programming occurs at all times and length scales in biomolecular systems. Although the above mentioned computational systems in principle can program themselves, this capacity has never been studied or used. It is known that any of the universal systems have the following properties: (I) the ability to store information, (II) the ability to communicate information, and (III) the ability to perform non-trivial information processing.

These abilities are also found in the living cell, although they are more difficult to classify. However, using the same scheme to discuss elements of biomolecular computation, we obtain:

(I) Storage and memory abilities: (1) single molecules, e.g. DNA and proteins, and (2) assembly/disassembly of supramolecular structures, e.g. the cytoskeleton and the cell membrane.

(II) Signal abilities: (1) diffusion (passive transport of materials, energy, and information) occurs everywhere in the cell, (2) active transport (non-specific (convection) and specific transport of materials, energy, and information) convection occurs in the cytoplasm and specific transport occurs for example over the cell membrane and along the microtubules, (3) conformational changes (transfer of energy and information), e.g. of dynein and kinesin in relation to cilia mobility, and (4) electromagnetic irradiation (transfer of energy and information), e.g. photochemical processes in chlorophyll.

(III) Transformation abilities: (1) chemical reactions - often using signal molecules as reactants to produce new signal molecules as products or using signals which act as catalysts or triggers, and (2) transcription of DNA to RNA and translation of RNA into protein molecules which fold up and act as constructive and regulatory units in the cell.

From this scheme it should be obvious that most fundamental biomolecular processes can be interpreted in terms of computation. These biomolecular processes are all coupled through a very complex network of functional interactions about which we only know certain details and in which the overall bauplan is still a mystery. The cell continuously programs and re-programs itself, and in multicellular organisms this self-programming also occurs at the organism level (recall the discussion of the feedback loop controlling insulin release).

Living systems can through a re-programming of some of their parts alter functional properties which are of vital importance for survival. Viewed over longer time scales this self-programming ability is also used to create new properties which are incorporated through the selection process of evolution. Since any computational universal system, in principle, is able to program itself, we shall modify one of them so that we can study self-programming as a phenomenon in a much simpler and more tractable system. We have chosen to modify the parallel von Neumann architecture. The modified von Neumann machine (MVNM) is easy to program since most modern digital computers are based on the von Neumann principle, and since the autonomous dynamics of such a system even at its lowest level (one single instruction) has a clear computational interpretation. We shall in the following focus on the emergence of new functional properties in MVNM's which most clearly reflect the evolutionary aspect of blocomputing.

SELF-PROGRAMMABLE MATTER

We can in general terms define self-programmable matter as a dynamical system of functional interacting elements, or compositions of elements, which through

autonomous dynamics can develop new compositions of functionally active elements. Such systems are characterized by an ability to construct novel elements within themselves. Thereby chemistry by definition becomes a particular kind of self-programmable matter. The physical properties (e.g. shape and charge) of the chemical species define the possible interactions with other molecules and thereby their functional properties. Chemical systems create new properties through recombination of molecules via chemical bonds. New combinations between existing molecules and combinations of new molecules with other molecules then define new functional properties. This defines a constructive or self-programming loop given by:

molecules → physical properties → functional properties → interactions → new molecules.

Description of Venus

As an example of a self-programming system, we have defined a modified von Neumann machine, called Venus. It consists of a one-dimensional memory array, called the core. This corresponds to the RAM (random access memory) in a modern digital computer. Each element of the core, a word, contains a machine code instruction. There are 10 different instructions, which are listed in Table 1. An instruction has three elements, an opcode and two fields, A and B. Each field consists of an addressing mode and a numeric integer. There are four different addressing modes, as shown in Table 2.

Many programs simultaneously execute instructions in the core. A monitor-like function always discovers whenever two or more instructions simultaneously try to obtain write access to the same core addresses. Thereby, write conflicts are resolved.

The model has several features which are not found in ordinary multi-tasking VNM's. One of the major differences is the presence of noise in our system. The task of an ordinary VNM is to perform very specific calculations, detailed via programs written by human subjects either in machine code or in a high-level language such as FORTRAN or C. In the presence of noise, most programs, e.g., ordinary differential equation solvers or bookkeeping programs, would crash or give more or less meaningless outputs. This is contrary to the computations in biological systems in which

OPCODE	FUNCTION
DAT B	Non-executable statement. Terminate the process currently executing. Can be used for storing data.
MOV A,B	Copy the contents of A to B.
ADD A,B	Add the contents of A to the contents of B and save the result in B.
SUB A,B	Subtract the contents of A from B and save the result in B.
JMP A	Move the pointer to A.
JMZ A,B	If B equals zero, move the pointer to A.
JMN A,B	If B differs from zero, move the pointer to A.
DJN A,B	Decrement B, and if B differs from zero, move the pointer to A.
CMP A,B	If A differs from B, skip the next instruction, e.g. move the pointer two steps ahead instead of one.
SPL B	Create a new process at the address pointed to by B.

Table 1.

noise usually has a very limited effect. One notion of noise in Venus is built into the execution of the MOV instruction. When the MOV instruction (see Table 1) copies a word, something might go wrong, and the word written to the memory can be altered. This is the reason why ordinary programs have a hard time executing properly. Such routines rely on perfect copying of data. There is an additional source of noise that drives our system. Once in awhile a random pointer is appended to the execution queue. Since processes can terminate by executing the DAT instruction, we make sure the system is supplied with pointers via this stochastic mechanism. The mutation frequency is one per 10^3 copyings, and a pointer is appended to the execution queue approximately every twentieth generation. A mutation of a machine code program always yields a new legal program, as opposed to a change in a high-level language, which almost certainly will result in a syntax error.

The Venus system also incorporates a notion of computational resources. This prevents the simultaneous execution of too many processes both in total and within a limited spatial addressing area. The first limitation is expressed in terms of an execution queue of fixed length, which in all the simulations to be discussed were of size 220. The execution queue contains the addresses of the instructions to be executed. The second limitation is due to address-localized computational resources, which are measured in fractions of one execution. Each address in the core y is at any time t associated with a certain fraction $r(t,y)$ of one execution. This value is incremented by a fixed amount Δr at each generation. However, the value can never exceed some constant predefined fraction r_{max} of one execution. When the system executes a pointer from the execution queue, it looks to the addresses in the immediate neighborhood of the pointer and finds the sum of computational resources. If

this sum exceeds one, then the instruction will be executed. If not, the pointer will disappear. The resource radius R_{res} defining the immediate neighborhood is three in all simulations.

Instructions are only allowed to communicate, e.g. to read and write data, locally. The allowed distance for read/write access is 800 in all simulations. However, in contrast with normal multi-tasking VNM's, all processes can overwrite anything in their neighborhood, as long as it does not occur simultaneously with other processes. This

ADDRESSING MODE	EFFECTIVE OPERAND
# (immediate)	The effective operand is the value in the data field. Example: MOV #3,.. , has the effective operand 3.
\$ (direct)	The effective operand is the word pointed to by the value in the data field. Example: MOV \$2,.. , has the effective operand located two words towards increasing addresses.
@ (indirect)	The effective operand is found by looking at the data field pointed to by the actual data field, and then using the direct mode.
< (autodecrement indirect)	As indirect, only the value pointed to by the actual data field is decremented before being used.

Table 2.

means that there is no notion of individual work space or, in biological terms, there are no predefined "proto-organisms" (cellularity).

Initialization of Venus

In all simulations, the system has been initialized randomly by a uniform distribution of opcodes, addressing modes, and data fields. Previous studies showed that the system can only evolve simple structures from a uniform distribution (Rasmussen et al. 1990). One can increase the complexity of the dynamics of the system greatly by supplying some kind of biasing of the initial core. In other words, we need to supply the system with a reactive potential, in the sense that the different machine code instructions need to be inhomogeneously distributed in the core to enhance many spontaneous computations. This potential is conveniently introduced by placing a self-replicating program in the randomly initialized core. This program has a replication cycle of 18 generations and will very quickly produce a considerable number of offspring, all of which will attempt to replicate unless they have been modified by noise or have been overwritten. As mentioned earlier, most programs designed to work in a noise-free environment very quickly crash by making erroneous copies. Another way in which these programs begin to malfunction is by copying on top of other offspring, which eventually happens since the core has a limited size, 3,584 addresses in the simulations to be discussed. By using our interactive graphics simulator, we have determined the lifetime of a well-functioning population of self-replicating programs to be around 200 generations. After this, no copies of the original programs are left. Only mutated versions with different functional properties are found. It is important to notice that the effect of the self-replicating program is a good mixing of instructions, and not a probing of the system with self-replicating properties. The last bit of this sophisticated self-replication is gone after 200 generations.

An alternative and conceptually more satisfying method for supplying the system with a reactive potential is to generate a random core by using a set of coupled Markov matrices. This approach is currently being investigated (Rasmussen et al. 1991).

Experimental Results

In the simulations with Venus, many different evolutionary paths have been observed. Typically, after extermination of well-functioning self-replicating programs, the system enters a phase in which massive copying of one or more instructions takes place. In the beginning, this is mainly caused by the copying loops of the former self-replicating programs. Typically such a partially malfunctioning loop will move copies of a single word out into the vicinity of the loop. As a result, large areas of the core will be filled with a single word, with unaltered opcode, amode, afield, and bmode, but possibly different bfields. This runaway process introduces a kind of sensitive dependence on initial conditions, as known from chaotic dynamical systems. However,

after some generations the copying loops are destroyed, either by noise or by MOV instructions overwriting them. This signals the beginning of a new epoch for the system. From this point on, the dynamics of the core is governed by large coherent groups of single instructions of sizes ranging from one word to several hundred words. This is in strong contrast to the dynamics in the first phase of the evolution, in which relatively few instructions placed in the self-replicating programs were responsible for the dynamics.

Naturally, the further development of the core depends heavily on the distribution of opcodes at this point. The distribution of opcodes is determined by the instruction copying, and therefore from this point on we see different evolutionary paths. For instance, a core consisting mainly of SPL instructions will lead to an evolution involving large areas crowded with pointers, while a core containing mainly MOV instructions will lead to a very dynamical behavior concerning the contents of the core, but also to a core with a small concentration of pointers. In the following, we shall take a look at three different evolutionary paths that are often observed in simulations with Venus.

If the core contains relatively many jump instructions (e.g. JMP, JMZ, JMN, and DJN) with immediate amode, the pointers will get caught at these, because an afield with immediate addressing mode points to its own location in the core. In other words, a pointer meeting one of these instructions will keep jumping to the same instruction over and over again, until it is either killed by lack of computational resources, or the jump instruction is overwritten by noise or a MOV instruction. Of course, such a core will be quite static with respect to the distribution of different instructions, because most pointers will be trapped at the jump instructions. Therefore, possible MOV instructions will only rarely be executed. This kind of core will be referred to as a fixed point core. This state constitutes a quite trivial form of cooperation. The self-reinforcing mechanism is characterized by the special mixture of instructions. With the presence of noise in this core, a new pointer will occasionally activate a successive number of the MOV instructions, which, with a probability close to one, will repeatedly copy other MOV instructions or one of the jump instructions until the new pointer gets trapped at one of the jump instructions. Here, it may or may not survive, depending on how many pointers each jump instruction in the neighborhood can carry in terms of computational resources. Such a fixed point structure is very stable towards perturbations.

As indicated above, another common evolutionary path evolves from a core with a considerable number of SPL instructions. In such a core, a large pointer concentration in an area will imply the execution of many SPL instructions, leading to even more pointers, constituting a positive feedback loop. The pointer concentration in such areas will increase until lack of resources or the limited execution queue length puts an end to the growth. These SPL colonies have a more interesting cooperative dynamics than the fixed point cores. The SPL instruction colonies cooperate in the sense described above: Locally, they distribute pointers to their neighbors, which do the same, and globally, the colony occupies all available computational resources in terms of pointers. Sometimes we observe several addressing areas with this kind of behavior.

Above a certain size, the different colonies compete for pointers, and an intermittent behavior between the areas can be observed. However, because of the lack of MOV instructions, the contents of the core will be conserved.

A third and even more interesting evolutionary path is that of the MOV-SPL structure. If a core has a large number of MOV and SPL instructions, a very complex cooperative structure can emerge. The cooperation works in the following way: SPL instructions supply the structure with pointers, both to SPL and MOV instructions. This is somewhat similar to their function in SPL colonies, where they distribute pointers to themselves. We can interpret this as a supply of free energy in thermodynamic terms. The MOV part of the structure makes sure that both MOV and SPL instructions are copied, giving the structure the ability to move around in its environment and to locally explore the spatial resource in terms of addresses in memory. The copying is thereby also responsible for the growth of the structure. This of course means that the structure does not have a well-defined genotype in a contemporary biological sense.

It generally takes several thousands of generations from the extermination of copying loops before the cooperative MOV-SPL structures appear. Typically the system goes through several phases with relatively large concentrations of instructions other than SPL and MOV before reaching the MOV-SPL state. Usually a core with a MOV-SPL structure consists of up to 80 percent SPL instructions, with most other instructions being MOV instructions; however, the ratio of SPL to MOV instructions may change during the epoch.

				
DAT	MOV	ADD	SUB	JMZ
				
JMZ	JMN	DJN	CMP	SPL

Figure 1. The shading used in core and cellular views.

The MOV-SPL structure is very stable, even though it is constantly subjected to perturbations, because the MOV instructions copy words continuously. However, the structure apparently does not change its basic functional properties. If we were to change the opcode of a word, the functional properties of this word would clearly be altered drastically. Since this happens all the time in the MOV-SPL structure, the system has found an area in rule space where it is stable towards such perturbations. Stability in Venus is an emergent property rather than an intrinsic element of the chemistry, because the elements of which the structures are composed are themselves very fragile.

In order to illustrate the micro- and macroscopic dynamics of the MOV-SPL structure, we have made two different kinds of projections of the high-dimensional

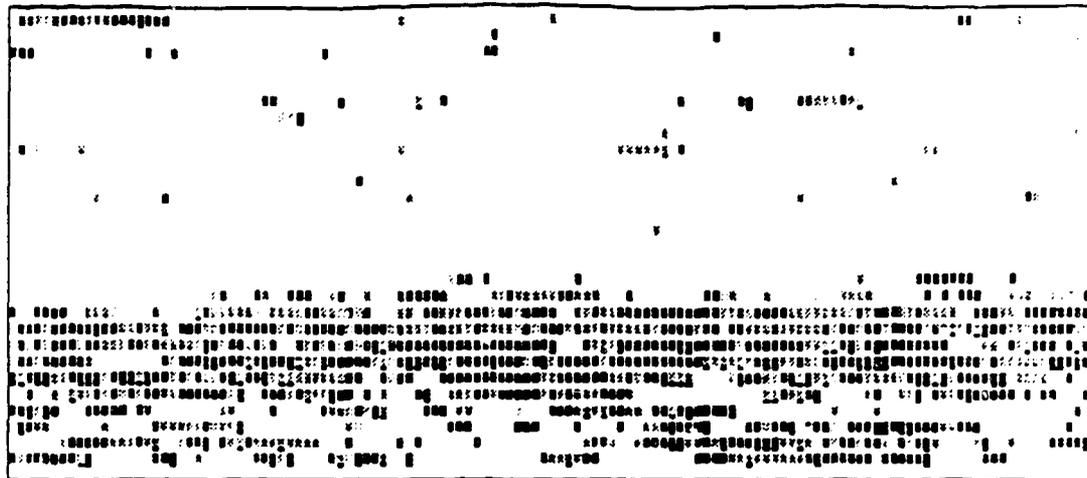


Figure 2. Core portrait of a MOV-SPL structure. The upper row shows the opcodes at addresses 0 through 127, the second row the addresses from 128 through 255, etc. Black underscores represent pointers. White squares indicate no recent references.

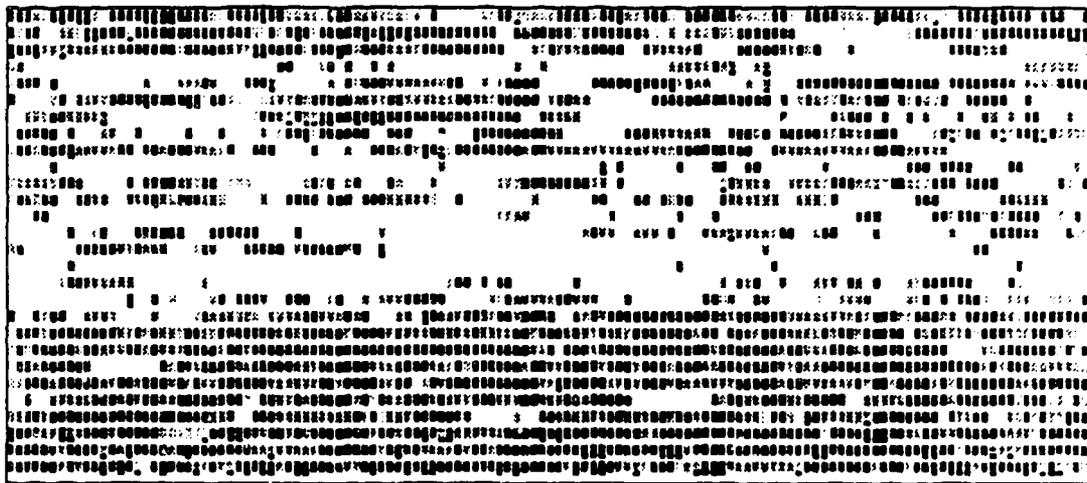


Figure 3. Core portrait of the structure shown in Figure 2, 100 generations later. Note how the activity has changed.

discrete phase space. The macroscopic dynamics is illustrated in Figures 2 and 3, in which each opcode in the core is represented by a small bar. The shade of each bar corresponds to the instruction type as explained in Figure 1. The figures show the opcodes in the core at two different times in the evolution. Note that in both of the core views, large parts are white. This merely means that these addresses have not been referenced for a while; there are no empty words in the core. In Figure 2, we see that the activity is restricted to higher addresses, whereas in Figure 3, approximately 100 generations later in the evolution, the structure has increased its domain. Note also that most words are occupied by either SPL- or MOV-instructions. The black underscores represent pointers.

Figure 4 shows a cellular automata-like view of part of the core. We have chosen 128 consecutive addresses within the MOV-SPL structure. The opcodes of the words in this addressing region are shown as horizontal lines at consecutive time points. In this figure, time increases downwards. The black underscores here also represent pointers. It is obvious that the microscopic dynamics is very irregular, although the macroscopic dynamics is preserved. We see how single words or sometimes consecutive words are overwritten. Also note that once in awhile an opcode different from MOV or SPL appears, caused by mutations. Groups of pointers suddenly appear or disappear. In biochemical terms, this structure has an irregular metabolism.

The evolutionary stories described here, and additional ones, were discussed in greater detail by Rasmussen et al. (1990 and 1991).

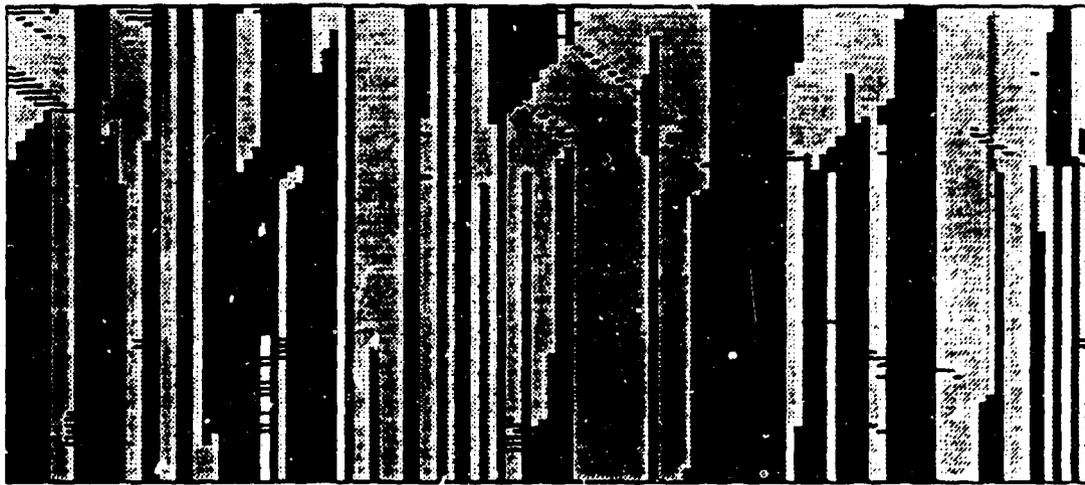


Figure 4. A cellular automata-like view of a part of the core.

INFORMATION DYNAMICS OF VENUS

The dynamics of self-programmable matter is generally quite complex, as described in the previous section, and at the same time simulations are computationally very time consuming. It is therefore preferable to have some quantitative measures that are easily computed and that signal changes in local or global dynamics. The calculation of such measures would enable us to characterize the system's behavior in terms of well-established quantities.

In order to make sampling frequent and simple, we have computed some simple information theoretic measures. The chosen measures contain both spatial and temporal ingredients. Such a combination of spatial and temporal measures is not necessarily ideal for all applications. It can be required, however, if the ingredients of the computa-

tional chemistries in question are such that there exists a preferred direction in space. To simplify things further, we shall here only consider the opcode, since the functional properties of a word are mainly determined by this element.

The simplest of the measures is the spatial entropy for templates of size one S_1 . The spatial entropy is defined as the usual Shannon entropy, calculated from the probability distribution p_k defining the probability of finding an instruction with the opcode k at an arbitrarily chosen site in the core

$$S_1(t) = - \sum_{k=0}^9 p_k \log_2(p_k).$$

Another quantity of interest is the mutual information M_1 . This is defined in terms of the spatial entropies S_1 and S_2

$$M_1(t) = 2S_1(t) - S_2(t)$$

where

$$S_2(t) = - \sum_{k=0}^9 \sum_{l=0}^9 p_{kl} \log_2(p_{kl}).$$

The interdependence between two events A and B can be measured by the mutual information. The mutual information can therefore reveal the emergence of correlations between neighboring instructions and thereby the occurrence of new properties of interactions. Of course, too large correlations, such as for the pattern "101010101010..." are not desirable, since they simply indicate that everything is overly dependent. The appearance of mutual information of intermediate values is of more interest, since this could indicate that the system is able to perform non-trivial information processing (Langton 1990).

The spatial entropy for templates of size two S_2 is calculated from a probability distribution p_{kl} which, as we shall discuss shortly, also captures some very important temporal correlations that determine the system's potential functional properties. The probability p_{kl} is defined as the probability of finding an instruction with opcode k at an arbitrary address n , and an instruction with opcode l at address $n+1$. Note that this probability is not the same as the probability of finding opcodes k and l as neighbors. The reasons for the chosen definition are that pointers are incremented one word after each execution, and that the pointers therefore in general travel towards increasing addresses, with the obvious exception of the jump instructions. This means that changes in S_2 can signal a change in the typical sequential order of execution of two neighboring instructions. A change in the typical sequential execution order of instructions is of considerable interest, since a change in the potential functional properties can indicate that the system is in the process of changing its global dynamics

through some self-organizing process. If the spatial entropy for templates of size one S_1 is approximately constant, the effect of a change in S_2 will immediately show up in the mutual information M_1 .

Experimental Results

All experiments are performed with the parameters and initial conditions discussed in the previous section. During simulations, the above measures were calculated frequently. The sampling rate is the same in all computer experiments, namely one sampling every 10 generations. As can be seen from Figures 5 and 6, the monitored measures change reasonably with this frequency, i.e. there are apparently no sudden jumps in the measured quantities between samplings.

Figure 5 shows the entropy and the mutual information vs. time for a particular simulation. In Figure 6a we see the mutual information vs. entropy for the same simulation, and Figure 6b shows the mutual information vs. entropy for another simulation. In the following we shall describe these simulations in terms of their information dynamics.

It appears that the process starts with a drastic increase in the mutual information, while the entropy is almost unchanged. This is observed in both Figures 6a and 6b (the simulation starts in the lower right-hand corner of the plots) and is caused by offspring of the program initially placed in the core distributing their code. This process does not influence the opcode distribution very much, since the program has a distribution of opcodes fairly close to that of a randomized core. However, the spreading of this code influences the spatial correlations in the core and thereby the mutual information. After about 200 generations the self-replication of programs stops because of malfunction introduced by noise and programs writing on top of each other. Now the entropy starts falling while the mutual information remains in the vicinity of 0.6 bits. This is especially clear in Figure 6b but can also be observed in Figure 6a as a noisy plateau at the right-most corner of the figure. The decreasing entropy in this phase is caused by partially malfunctioning copying loops spreading out a large number of a particular instruction.

For the simulations shown in Figures 5 and 6a, we see that after about 4,000 generations, the mutual information drops to a level of about 0.2 bits. At the same time the entropy drops to about 1.5 bits, and some oscillations in both the entropy and the mutual information are observed, giving rise to a "cloud-like" picture of the information dynamics in Figure 6a. In this part of the simulation, the core is mainly populated with MOV and SPL instructions and the pointer density is very high.

The epoch of the MOV-SPL structure continues until $t \approx 20,000$, where a large peak in the mutual information reflects a change in the functional properties of the system (recall the discussion on mutual information in the previous section). From this point on, the changes in the entropy as well as in the mutual information become less rapid and exhibit a stepwise character signalling yet another epoch of the system. In this phase the core mainly consists of MOV-, SPL-, and jump-instructions where most

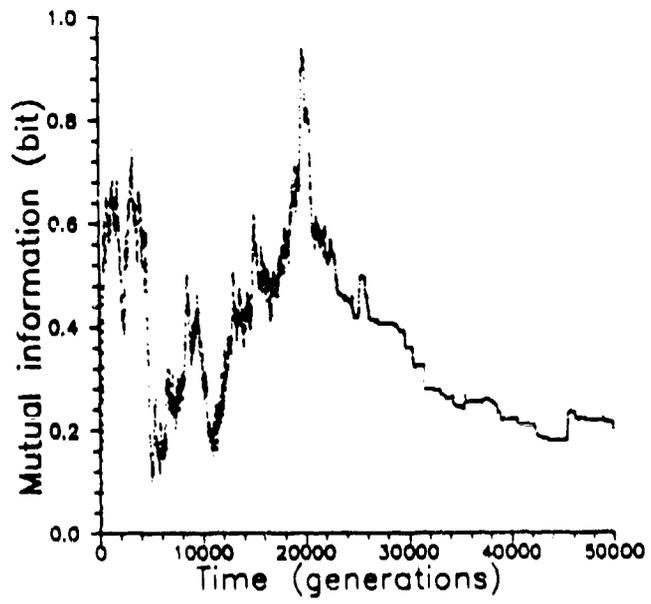
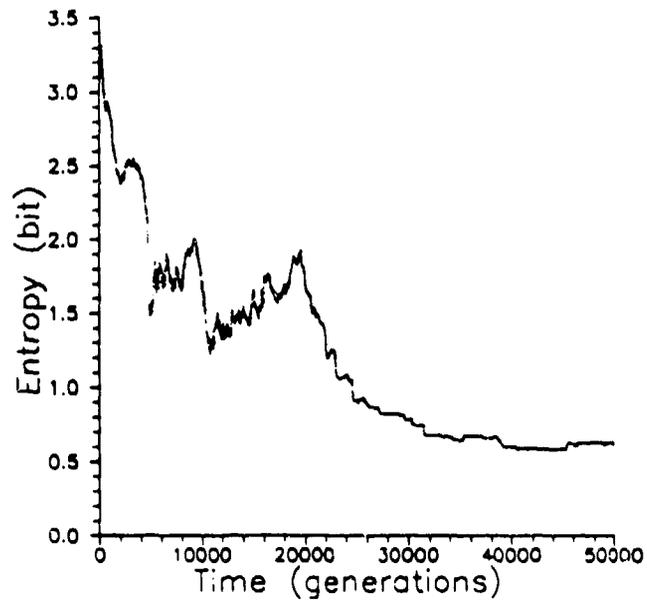


Figure 5. (a) The usual Shannon entropy of the opcode distribution versus time. (b) The mutual information versus time.

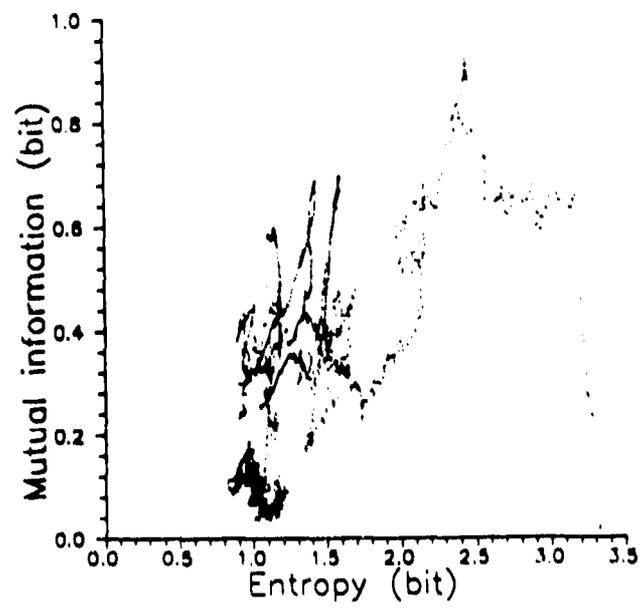
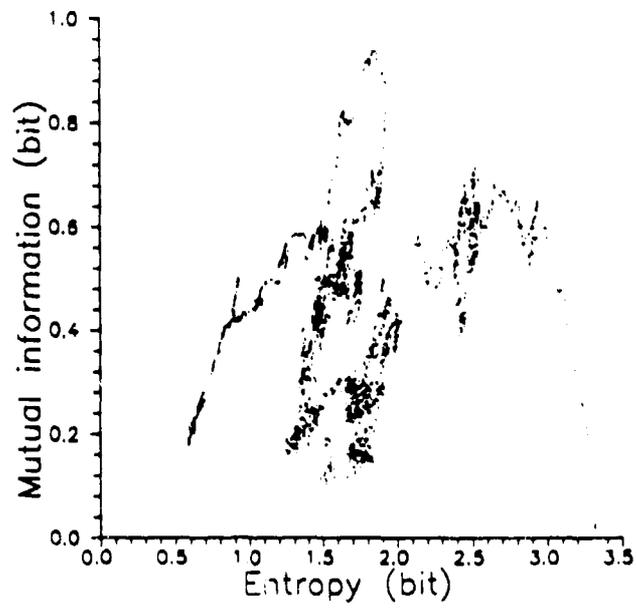


Figure 6. (a) shows the mutual information versus the usual Shannon entropy for the simulation shown in Figure 5. (b) shows mutual information versus entropy for another simulation.

of the pointers are trapped at jump instructions.

In the simulation shown in Figure 6b, the system leaves the plateau of partially malfunctioning copying loops at $t = 1,100$. At this point there is a rise in the mutual information to a little above 0.9 bits followed by a drastic drop to a value of about 0.2 bits. After this we see the same cloud-like distribution of points in the figure as observed in Figure 6a. Note that during most of this phase we see rapid changes of the mutual information, while the changes in the entropy are more moderate. These changes indicate that some dramatic changes occur, while the opcode distribution is only affected slightly. This kind of behavior is likely to be caused by MOV instructions shifting the contents of the core as opposed to the partially malfunctioning copying loops multiplying single instructions into large areas of the core. These somewhat organized changes signal that the MOV instructions are activated many at a time, reflecting some kind of structure in the part of the core responsible for the dynamics. Finally, at $t = 32,000$ we observe a drop in the mutual information to about 0.1 bits and the disappearance of the large oscillations in the mutual information. This phase of the simulation is observed in Figure 6b as a small point dense area just below the cloud-like point distribution. Compared to Figure 6a this final phase is somewhat more active, with small oscillations in the entropy and the mutual information. These oscillations are caused by MOV-instructions continuously re-organizing the contents of the core, however, in a less organized and dramatic way than in the previous phase of the simulation.

A rather coarse-grained resumé of the simple information dynamics would be that the system starts with a high degree of disorder, low complexity, and a high reactive potential in terms of the initial distribution of opcodes. The system then evolves, lowering the entropy towards intermediate values, while the complexity increases. Then the system wanders around in information space in a very complicated manner according to the information theoretic measures characterized by sudden changes in both order and complexity, where all major changes always are associated with changes in the functional properties. With a time horizon of 50,000 generations, most processes end up in one of two different dynamical states. One is best characterized as a frozen state, and the other may be characterized as recurrent or chaotic. The frozen state is a perturbed fixed point consisting of a variety of jump-instructions which have trapped the pointers. The fixed point dynamics is in this situation often perturbed by two factors. One factor is that some of the conditional jump-instructions periodically have their bfields counted down, allowing trapped pointers to escape. The other factor is the introduction of random pointers. The overall effect of both of these perturbations is small changes in the core reflected by small steps both in entropy and mutual information. This type of dynamics is seen in the simulation shown in Figure 6a. Another example of this type of dynamics is the collapse, where all pointers disappear. This also occurs quite often. The terminal state with recurrent dynamics is typically found in situations where a major part of the core is occupied by either MOV- or SPL-instructions. The MOV-dominated cores have many MOV-instructions executed at each

generation changing the content of the core all of the time. This type of dynamics is seen in the simulation shown in Figure 6b. In SPL-dominated cores the composition of instructions remains virtually constant, whereas the pointer dynamics exhibits pronounced intermittent behavior. The SPL-dominated cores of course have maximal pointer density. The MOV-SPL structure often emerges as a transient structure active from generation 4,000 until generation 20,000. In rare situations the MOV-SPL structure can survive even after 100,000 generations. Since Venus is a universal system it of course supports the three main ingredients of computation (recall the discussion in the previous section): (i) the capacity to store information, (ii) the capacity to exchange or communicate information, and (iii) the ability to process information in a non-trivial way. Dynamical systems with these computational capabilities can apparently, in thermodynamical terms, be characterized as operating in the immediate vicinity of a phase transition (Langton 1990).

Since the dynamics of Venus changes the instruction contents of the core and thereby the rules governing the dynamics, the system often changes its computational capabilities during a simulation. An example is one of the terminal states, the frozen state (fixed point cores with pointers trapped at jump instructions), very efficient in storing information. However, the system has in this state, which in thermodynamical terms is equivalent to a solid state, lost its ability to communicate and process information. The chaotic or recurrent state (for example cores mainly populated with MOV instructions) exhibits a pronounced ability to exchange or communicate information but only a limited ability to perform non-trivial information processing. This state can be characterized as a fluid state in thermodynamical terms. The MOV-SPL structure is from a computational point of view more interesting. It is clear that the MOV-SPL structure does not have a well-defined genotype, and consequently it does not store information in the usual sense of data storage (recall (i)). However, it is also clear from the previous analysis that the macroscopic dynamics is preserved. The system therefore, through some complicated coding, stores its phenotype rather than its genotype. With respect to (ii) and (iii), it is fairly obvious that non-trivial computations are performed, and that information is being communicated by the MOV instructions, within the structure itself.

Scaling of Correlations

One can generalize the mutual information between neighboring words to investigate long-range correlations. When looking at long-range correlations, almost all temporal effects are removed. Changes in the functional properties/potential are certain to have taken place if long-range correlations should appear or disappear.

In the following, we shall discuss the simulation shown in Figure 5 from this point of view.

At time $t=5,000$, an interesting phenomenon is observed in the simulation concerning the spatial correlations. Here, the mutual information of opcode, amode,

and bmode versus distance all decay smoothly. If we try to fit a power law to the decay, we can describe the decay by a scaling exponent α

$$M(n) = M(1) n^{-\alpha}.$$

What is particularly interesting here is that all three decays can be characterized by the same scaling exponent, $\alpha=0.4$. One might suspect that this is always the case since the three elements of a word cannot be altered, but for example at $t=1,000$, we find three different scaling exponents, ranging from 0.64 for the opcode to 1.1 for the amode. At later stages in the evolution, e.g. at time $t=20,000$ and $t=21,000$, the exponents differ by more than 30% from each other. In Figure 7, the decay of correlations for the opcodes is shown.

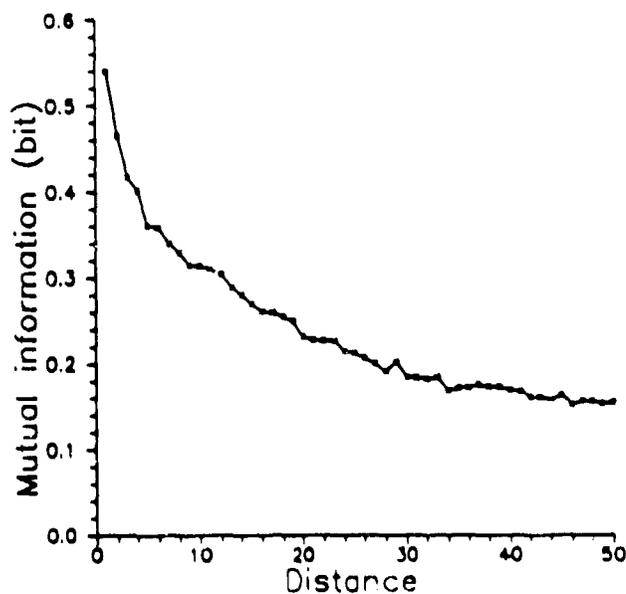


Figure 7. The figure shows the decay of correlations between opcodes as computed by the mutual information.

After approximately 20,000 generations, both the spatial entropy S_1 and the mutual information M_1 have maxima. Since both S_1 and M_1 change rapidly around this time, as can be seen in Figure 5, we expect to be able to find some indication in the long-range correlation. At time $t=19,000$, we see from Figure 8 that there is a smooth decay of M_1 to values below 0.01 bit. The same is observed at time $t=21,000$ (see Figure 8c), except that the value of the mutual information here slightly exceeds 0.01 bit. If we look at the correlations at time $t=20,000$ (Figure 8b), several peaks are observed. At

distances near 55 and 100, significantly greater values of the mutual information than in the surroundings can be seen, indicating long-range correlations. Since long-range correlations indicate some relationship between sites in the core separated with some distance, the emergence of such correlations signals that one or more MOV-instructions move the contents of the core that specific distance. Thus, the long-range correlations can be used to detect activity of one or more such MOV-instructions, enabling us to detect changes in the core that have only insignificant influence on the total distribution of opcodes, i.e. the entropy.

Yet another way of describing coherence in the core is by means of correlations defined by Markov matrices. In the case of opcode we can define a Markov matrix determining the probability p_{ij} of opcode i at address n is followed by opcode j at address $n+1$. Actually, such a matrix is intimately related to the spatial entropy $S_{\underline{z}}$. Furthermore, as we mentioned earlier, such Markov matrices can be used for supplying the system with a reactive potential, instead of using a small self-replicating program, simply by generating the initial conditions on the basis of a set of low order Markov chains. We have calculated how the Markov chains determining the correlations between an opcode at address n and an opcode at address $n+1$ actually look at the sampling times corresponding to Figure 8. Figure 9 shows 3-D pictures of these Markov matrices. Note that the figure shows $\log(N_{ij})$, where N_{ij} is the number of occurrences of opcode j following opcode i . Note that p_{ij} can readily be calculated from N_{ij} .

In Figure 9a the strong correlations between subsequent MOV- and SPL-instructions indicate that the core is mainly populated with these instructions at $t=19,000$. At $t=20,000$ several smaller peaks appear indicating fluctuations introducing JMZ and DJN instructions. This signals that an instability is underway leading to an increased concentration of instructions other than the ones ruling the core earlier. Finally, at $t=21,000$ a new set of high peaks appears revealing a strong presence of JMZ. The instability has thereby changed the macroscopic composition of the instructions. Note also that the small peaks indicating the emergence of DJN instructions have vanished. The DJN fluctuation has died out. The transition occurring in the system around $t=20,000$ indicated by the information dynamics is thereby also reflected in the opcode correlation matrices. In addition these matrices indicate which instructions are involved in the transition and which new instructions emerge in the new epoch.

Obviously, the chosen measures are not sufficient to characterize the dynamics in all details. However, the proposed measures have the advantage of being very accessible from a computational point of view. Also, some of the measures are closely related to the functional properties of the system. Several other more complicated measures have been proposed, but they are either very complex to compute or cannot be computed at all. Some of these measures are discussed in Bennett (1988 and 1989).

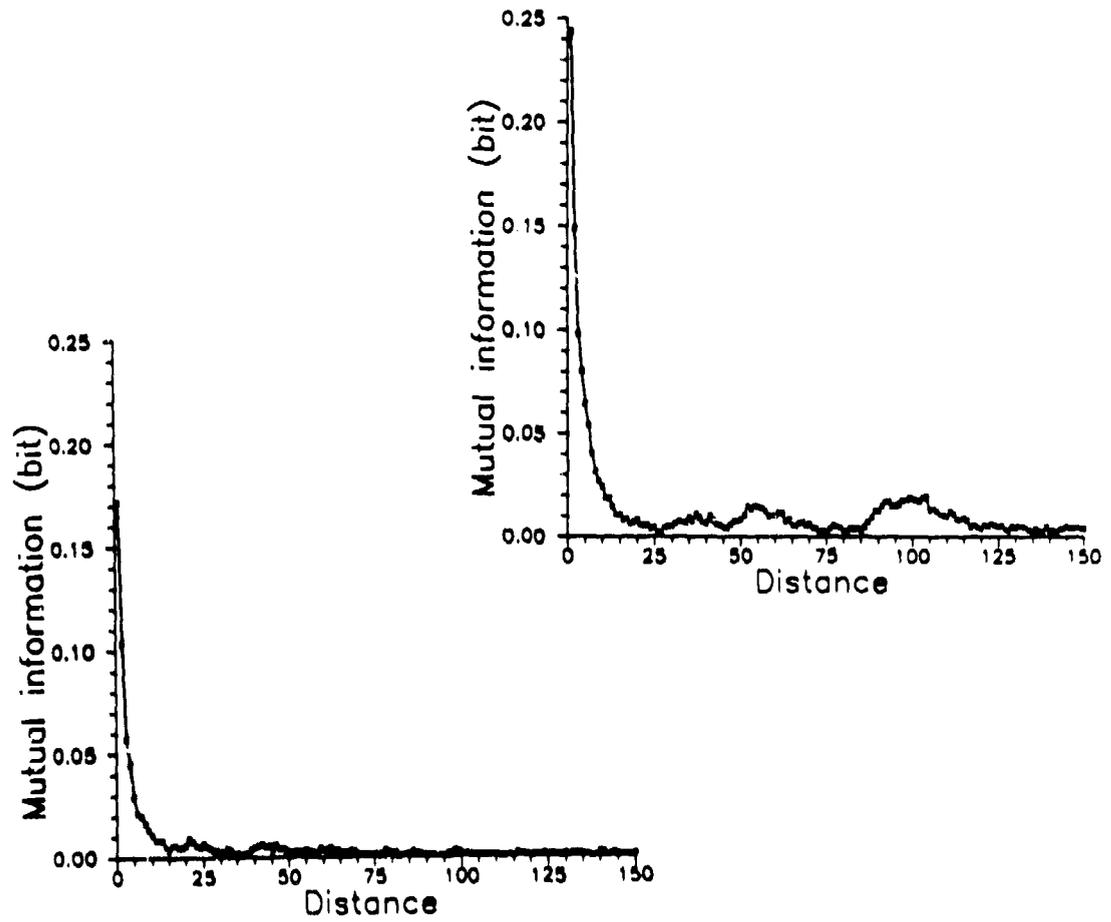
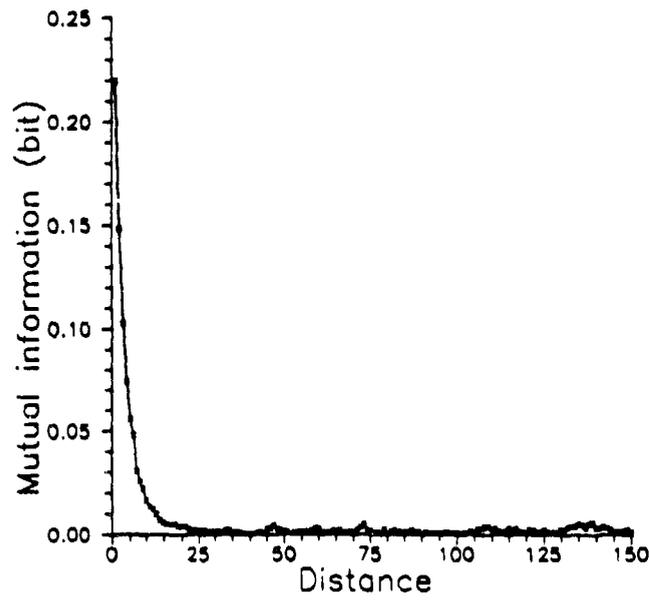


Figure 8. (a) The spatial mutual information vs. distance at $t=19,000$. (b-c) as (a) except at $t=20,000$ and $t=21,000$. Note the peaks in (b) around distances of 55 and 100.

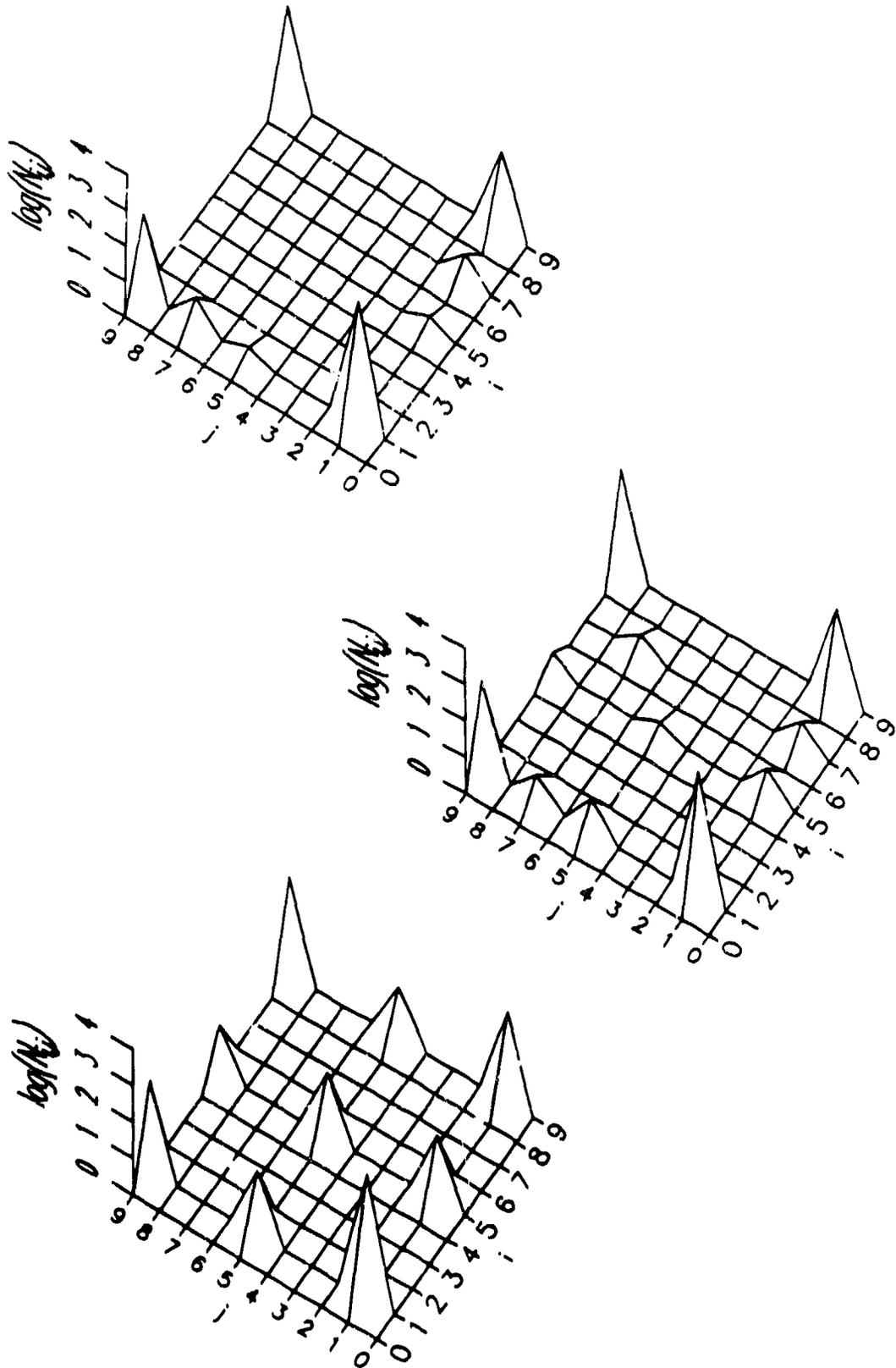


Figure 9. The occurrence of correlated opcodes at $t = 19,000$ (a), $t = 20,000$ (b), and $t = 21,000$ (c), respectively.

DISCUSSION

Of the information processing features associated with living systems, the ability of systems to alter themselves through an introduction of novel properties is what has made biological evolution possible. Novel properties are introduced through changes in the biomolecules that constitute the organisms. We refer to systems with these properties as being self-programmable. To investigate the self-programming property of living systems, we have designed a system far simpler and more tractable than contemporary biomolecular systems. Despite its simplicity, our system has the same fundamental constructive properties as contemporary biomolecular systems. We have seen how this parallel von Neumann based self-programmable system is able to successively develop novel functional properties and how complex cooperative structures spontaneously emerge in the system. Frozen accidents determine the different evolutionary paths in the system and thereby the particular details of the emerging structures. We have discussed cooperative properties of the MOV-SPL structure, of the SPL colonies, and of the jump cores.

These macroscopic cooperative structures are spontaneously generated in the computational system. They emerge in a similar way as macroscopic dissipative structures do, in, for instance, physico-chemical systems. A thermodynamical interpretation of the computational system yields an equivalence between the flux of free energy and the flux of computational resources (executions per iteration), and an equivalence between the microscopic degrees of freedom in the physico-chemical system and all the possible functional interactions in the computational system. A notable difference is, however, that our macroscopic computational structures change even with a constant pumping (executions per iteration). Due to our system's self-programming properties it does not stay in any fixed macroscopic pattern, as for instance a Raleigh-Benard convection or the chemical reaction waves in a Belousov-Zhabotinski reaction do. Our system has in addition the property biological systems also have: it can change itself and thereby undergo development.

How close the details of the processes and the details of the emerging cooperative structures are to evolutionary processes in biological systems and to the structures underlying contemporary living systems we cannot say. The detailed properties of biological evolution as well as the fundamental dynamics underlying life itself are yet unknown. However, our definition of constructive, or self-programmable, dynamical systems has allowed us to start a systematic investigation of truly evolutionary processes. We have freed our formal tools from any predefined evolutionary possibilities. Our system picks its own evolutionary route and constructs its own functional properties. This is in contrast to most formal approaches discussing evolutionary processes. A formal discussion of self-programmable systems is found in Rasmussen et al. (1991). Another constructive system based on the λ -calculus is discussed in Fontana (1991).

One of the major problems associated with self-programmable systems is their complexity. Since the functional properties of such systems depend on their dynamics,

any characterization of dynamics as well as functional properties is difficult. We would of course like to be able to detect when novel functional properties are introduced and how such new properties are characterized. It turns out that the Shannon entropy, the mutual information, and Markov chains constructed from correlations, in particular when combined, can be used for that purpose.

The simultaneous calculation of S_1 and S_2 (or M_1) is an efficient way to determine when a complex system is in a quasi-steady state and when the system is in a transition. An alternative is to compare the states of the system at subsequent generations, which perhaps is a little faster. For large systems such as Venus, this requires a considerable amount of storage and retrieval of data, which can be rather time consuming. The calculation of the Shannon entropy S_1 alone is, of course, fast, but if the instruction set includes an instruction to exchange the contents of two words (which is an instruction actually found in most modern microprocessors), then the execution of a large number of exchange instructions would in general alter the functional properties drastically, while S_1 would remain constant. In this case calculation of the mutual information captures that something happens. Another example of the mutual information changing while the entropy is virtually constant occurs when the small self-replicating programs distribute their code into the core.

By also considering long-range correlations and the Markov matrices we obtain more detailed information about the dynamics of the system. The emergence of long-range correlations indicates changes in the local interdependence of the core. Similarities and differences in the scaling exponents describing the decays of the correlations of the different fields as a function of distance can be used to uncover details about the dynamics of the individual fields. Finally, the Markov matrices reflect instabilities and fluctuations in the opcode distribution, and they in particular signal which opcodes are involved in the transitions of the system.

Obviously, these measures are not sufficient to characterize the dynamics in all details, and they do not uncover all the interesting details of the emerging cooperative structures. However, the proposed measures have the advantage of being very accessible from a computational point of view. Several other measures have been proposed, but they are either very complex to compute or cannot be computed at all. Some of these measures are discussed in Bennett (1988 and 1989).

ACKNOWLEDGMENTS

We would like to thank Jeppe Sturis, Walter Fontana, Doyne Farmer, Chris Langton, and Erik Mosekilde for valuable discussions. Erik Mosekilde and Ellen Buchhave are acknowledged for arranging such an enjoyable workshop.

REFERENCES

- Bennett, C. H., 1989, Dissipation, Information, Computational Complexity and the Definition of Organization, in: "Emerging Syntheses in Science," Pines, D., ed., Addison-Wesley, Reading.
- Bennett, C. H., 1988, Computational Measures of Physical Complexity, in: "Lectures in the Sciences of Complexity I," Stein, D. L., ed., Addison-Wesley, Reading.
- Fontana, W., 1991, Algorithmic Chemistry, in the proceedings of the Second Artificial Life Workshop, SFI Studies in the Sciences of Complexity, Farmer, J. D. et al., eds., Addison-Wesley (in press).
- Hopcroft, J. E., and Ullman, J. D., 1979, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading.
- Langton, C., 1990, Computation at the Edge of Chaos: Phase Transitions and Emergent Computation, *Physica D* 42, 12-34.
- Palmer, R., 1988, Neural Nets, in: "Lectures in the Sciences of Complexity I," Stein, D. L., ed., Addison-Wesley, Reading.
- Rasmussen, S., Knudsen, C., Feldberg, R., and Hindsholm, M., 1990, The Coreworld: Emergence and Evolution of Cooperative Structures in a Computational Chemistry, *Physica D* 42, 111-134.
- Rasmussen, S., Knudsen, C., and Feldberg, R., 1991, Dynamics of Programmable Matter, in the proceedings of the Second Artificial Life Workshop, SFI Studies in the Sciences of Complexity, Farmer, J. D. et al., eds. Addison-Wesley (in press).
- Sturis, J., Polonsky, K. S., Blackman, J. D., Knudsen, C., Mosekilde, E., and Van Cauter, E., 1991, Aspects of Oscillatory Insulin Secretion, these proceedings.
- Touretzky, D., ed., 1990, Proceedings of the Neural Information Processing Conference, NIPS 2, Morgan Kaufman Publishers.