Physica 22D (1986) 385–399
North-Holland, Amsterdam

APPROACHES TO COMPLEXITY ENGINEERING*

Stephen WOLFRAM**
The Institute for Advanced Study, Princeton, NJ 08540, USA

Principles for designing complex systems with specified forms of behaviour are discussed. Multiple scale cellular automata are suggested as dissipative dynamical systems suitable for tasks such as pattern recognition. Fundamental aspects of the engineering of such systems are characterized using computation theory, and some practical procedures are discussed.

The capabilities of the brain and many other natural systems go far beyond those of any artificial systems so far constructed by conventional engineering means. There is however extensive evidence that at a functional level, the basic components of such complex natural systems are quite simple, and could for example be emulated with a variety of technologies. But how large numbers of these components can act together to perform complex tasks is not yet known. There are probably some rather general principles which govern such overall behaviour, and allow it to be moulded to achieve particular goals. If these principles could be found and applied, they would make new forms of engineering possible. This paper discusses some approaches to such forms of engineering with complex systems. The emphasis is on general concepts and analogies. But some of the specific systems discussed should nevertheless be amenable to implementation and detailed analysis.

In conventional engineering or computer programming, systems are built to achieve their goals by following strict plans, which specify the detailed behaviour of each of their component parts. Their overall behaviour must always be simple enough that complete prediction and often also analysis is possible. Thus for example motion in conventional mechanical engineering devices is usually constrained to be simply periodic. And in conventional computer programming, each step consists of a single operation on a small number of data elements. In both of these cases, much more complex behaviour could be obtained from the basic components, whether mechanical or logical, but the principles necessary to make use of such behaviour are not yet known.

Nature provides many examples of systems whose basic components are simple, but whose overall behaviour is extremely complex. Mathematical models such as cellular automata (e.g. [1]) seem to capture many essential features of such systems, and provide some understanding of the basic mechanisms by which complexity is produced for example in turbulent fluid flow. But now one must use this understanding to design systems whose complex behaviour can be controlled and directed to particular tasks. From scientific descriptions of the behaviour observed in complex systems, one must learn how to engineer complex systems with specified behaviour.

Complexity in natural systems typically arises from the collective effect of a very large number of components. It is often essentially impossible to predict the detailed behaviour of any one particular component, or in fact the precise behaviour of the whole system. But the system as a whole may nevertheless show definite overall behaviour, and this behaviour usually has several important features.

Perhaps most important, it is robust, and is typically unaffected by perturbations or failures of

*Loosely based on an invited talk entitled "Cellular automaton engineering" given at the conference.

**Address from August 1986: Center for Complex Systems Research, University of Illinois, IL 61801, USA.

0167-2789/86/$03.50

individual components. Thus for example a change in the detailed initial conditions for the system usually has little or no effect on the overall outcome of its evolution (although it may have a large effect on the detailed behaviour of some individual elements). The visual system in the brain, for example, can recognize objects even though there are distortions or imperfections in the input image. Its operation is also presumably unaffected by the failure of a few neurons. In sharp contrast, however, typical computer programs require explicit account to be taken of each possible form of input. In addition, failure of any one element usually leads to catastrophic failure of the whole system.

Dissipation, in one of many forms, is a key principle which lies behind much of the robustness seen in natural systems. Through dissipation, only a few features in the behaviour of a system survive with time, and others are damped away. Dissipation is often used to obtain reliable behaviour in mechanical engineering systems. Many different initial motions can for example be dissipated away through viscous damping to bring particular components to rest. Such behaviour is typically represented by a differential equation whose solution tends to a particular fixed point at large times, independent of its initial conditions. Any information on the particular initial conditions is thus destroyed by the irreversible evolution of the system.

In more complicated systems, there may be several fixed points, reached from different sets of initial conditions. This is the case for an idealized ball rolling on a landscape, with dissipation in the form of friction. Starting at any initial point, the ball is "attracted" towards one of the local height minima in the landscape, and eventually comes to rest there. The set of initial positions from which the ball goes to a particular such fixed point can be considered as the "basin of attraction" for that fixed point. Each such basin of attraction is bounded by a "watershed" which typically lies along ridges in the landscape. Dissipation destroys detailed information on particular initial conditions, but preserves the knowledge of which basin of attraction they were in. The evolution of the system can be viewed as dividing its inputs into various "categories", corresponding to different basins of attraction. This operation is the essence of many forms of pattern recognition: despite small changes, one must recognize that a particular input is in a particular category, or matches a particular pattern. In the example of a ball rolling on a landscape, the categories correspond to different regions of initial positions. Small changes in input correspond to small changes in initial position.

The state of the system just discussed is given by the continuous variables representing the position of the ball. More familiar examples of pattern recognition arise in discrete or digital systems, such as those used for image processing. An image might be represented by a 256×256 array of cells, each say black or white. Then a simple image processing operation would be to replace any isolated black cell by a white cell. In this way certain single cell errors in the images can be removed (or "damped out"), and classes of images differing just by such errors can be recognized as equivalent (e.g. [3]). The process can be considered to have attractors corresponding to the possible images without such errors. Clearly there are many of these attractors, each with a particular basin of attraction. But in contrast to the example with continuous variables above, there is no obvious measure of "distance" on the space of images, which could immediately be used to determine which basin of attraction a particular image is in. Rather the category of an image is best determined by explicit application of the image processing operation.

Length n sequences of bits can be considered as corners of an n-dimensional hypercube. The Hamming distance between two sequences can then be defined as the number of edges of the hypercube that must be traversed to get from one to the other, or, equivalently, the total number of bits that differ between them. It is possible using algebraic methods to devise transformations with

basins of attraction corresponding to spheres which enclose all points at a Hamming distance of at most say two bits from a given point [4]. This allows error-correcting codes to be devised in which definite messages can be reconstructed even though they may contain say up to two erroneous bits.

The transformations used in error-correcting codes are specially constructed to have basins of attraction with very simple forms. Most dissipative transformations, however, yield much more complicated basins of attraction, which cannot for example be described by simple scalar quantities such as distances. The form of these basins of attraction determines what class of perturbations do not affect a system, and what classes of inputs can be recognized as equivalent.

As a first example, consider various idealizations of the system discussed above consisting of a ball rolling with friction on a landscape, now assumed one dimensional. In the approximation of a point ball, this is equivalent to a particle moving with damping in a one-dimensional potential. The attractors for the system are again fixed points corresponding to minima of the potential. But the basins of attraction depend substantially on the exact dynamics assumed. In the case of very large friction, the particle satisfies a differential equation in which velocity is proportional to force, and force is given by the gradient of the potential. With zero initial velocity, the basins of attraction in this case have a simple form, separated by boundaries at the position of maxima in the potential. In a more realistic model, with finite friction and the inertia of the ball included, the system becomes similar to a Roulette wheel. And in this case it is known that the outcome is a sensitive function of the precise initial conditions. As a consequence, the basins of attraction corresponding for example to different holes around the wheel must have a complicated, interdigitated, form.

Complicated basin boundaries can also be obtained with simplified equations of motion. As one example, one can take time to be discrete, and assume that the potential has the form of a polynomial, so that the differential equation is approximated by an iterated polynomial mapping. The sequence of positions found from this mapping may overshoot the minimum, and for some values of parameters may in fact never converge to it. The region of initial conditions which evolve to a particular attractor may therefore be complicated. In the case for example of the complex iterated mapping $z \to z^2 + c$, the boundary of the basin of attraction (say for the attractor $z = \infty$) is a Julia set, and has a very complicated fractal form (e.g. [5]).

The essentials of the problem of finding basins of attraction already arise in the problem of finding what set of inputs to a function of discrete variables yields a particular output. This problem is known in general to be computationally very difficult. In fact, the satisfiability problem of determining which if any assignments of truth values to n variables in a Boolean expression make the whole expression true is NP complete, and can presumably be solved in general essentially only by explicitly testing all 2^n possible assignments (e.g. [6]). For some functions with a simple, perhaps algebraic, structure, an efficient inversion procedure to find appropriate inputs may exist. But in general no simple mathematical formula can describe the pattern of inputs: they will simply seem random (cf. [7]).

Cellular automata provide many realistic examples of this problem. Cellular automata consist of a lattice of sites with discrete values updated in discrete steps according to a fixed rule which depends on their neighbours. The image processing operation mentioned above can be considered as a simple example of a single step in the evolution of a particular two-dimensional cellular automaton (cf. [8]). Other cellular automata show much more complicated behaviour, and it seems in fact that appropriate cellular automata capture the essential features of many complex systems in nature (e.g. [1]). The problems of complexity engineering addressed in nature are thus presumably already present in cellular automata.

Most cellular automata are dissipative, or irreversible, so that after many steps, they evolve to attractors which contain only a subset of their states. In some cellular automata (usually identified as classes 1 and 2), these attractors are fixed points (or limit cycles), and small changes in initial conditions are usually damped out [9]. Other cellular automata (classes 3 and 4), however, never settle down to a fixed state with time, but instead continue to show complicated, chaotic, behaviour. Such cellular automata are unstable, so that most initial perturbations grow with time to affect the detailed configuration of an ever-increasing number of sites. The statistical properties of the behaviour produced is nevertheless robust, and is unaffected by such perturbations.

It can be shown that the set of fixed points of a one-dimensional cellular automata consists simply of all those configurations in which particular blocks of site values do not appear [10]. This set forms a (finite complement) regular language, and can be represented by the set of possible paths through a certain labelled directed graph [10]. Even when they are not fixed points, the set of states that can occur after say t time steps in the evolution of a one-dimensional cellular automaton in fact also forms a regular language (though not necessarily a finite complement one). In addition, the basin of attraction, or in general the set of all states which evolve after t steps to a given one, can be represented as a regular language. For class 1 and 2 cellular automata, the size of the minimal graph for this language stays bounded, or at most increases like a polynomial with t. For class 3 and 4 cellular automata, however, the graph often increases apparently exponentially with t, so that it becomes increasingly difficult to describe the basin of attraction. In general, in fact, the problem of determining which states evolve to a particular one after t steps is a generalization of the satisfiability problem for logical function mentioned above, and is thus NP complete. The basin of attraction in this case can thus also presumably only be found by explicit testing of essentially all $\mathcal{O}(2^t)$ possible initial configurations (cf. [11]). Its form will again often be so complicated as to seem random. For two-dimensional cellular automata, it is already NP complete just to find fixed points (specifically, to find say $n \times n$ blocks of sites with specified boundaries that are invariant under the cellular automaton rule) [12].

It is typical of complex systems that to determine their behaviour requires extensive computation. This is a consequence of the fact that the evolution of the systems themselves typically corresponds to a sophisticated computation. In fact, the evolution of many complex systems is probably computationally irreducible: it can be found essentially only by direct simulation, and cannot be predicted by any short-cut procedure [11, 13]. Such computational irreducibility is a necessary consequence of the efficient use of computational resources in a system. Any computational reducibility is a sign of inefficiency, since it implies that some other system can determine the outcome more efficiently.

Many systems in nature may well be computationally irreducible, so that no general predictions can be made about their behaviour. But if a system is to be used for engineering, it must be possible to determine at least some aspects of its behaviour. Conventional engineering requires detailed specification of the precise behaviour of each component in a system. To make use of complex systems in engineering, one must relax this constraint, and instead require only some general or approximate specification of the overall behaviour of systems.

One goal is to design systems which have particular attractors. For the example of a ball rolling on a landscape, this is quite straightforward (cf. [14]). In one dimension, the landscape could be given by the polynomial $\prod_i (x - x_i)^2$, where the x_i are the desired minima, or attractors. This polynomial is explicitly constructed to yield certain attractors in the dynamics. However, it implies a particular structure for the basins of attraction. If the attractors are close to equally spaced, or are sufficiently far apart, then in fact the boundary of the basins of attraction for successive attractors

will be roughly half way between them. Notice, however, that as the parameters of the landscape polynomial are changed, the structure of the attractors and basins of attraction obtained can change discontinuously, as described by catastrophe theory.

For a more complex system, such as a cellular automaton, it is more difficult to obtain a particular set of attractors. One approach is to construct cellular automaton rules which leave particular sequences invariant [15]. If these sequences are say of length L, and are arbitrarily chosen, then it may be necessary to use a cellular automaton rule which involves a neighbourhood of up to $L-1$ sites. The necessary rule is straightforward to construct, but takes about 2^{L-1} bits to specify.

Many kinds of complex systems can be considered as bases for engineering. Conventional engineering suggests some principles to follow. The most important is the principle of modularity. The components of a system should be arranged in some form of hierarchy. Components higher on the hierarchy should provide overall control for sets of components lower on the hierarchy, which can be treated as single units or modules. This principle is crucial to software engineering, where the modules are typically subroutines. It is also manifest in biology in the existence of organs and definite body parts, apparently mirrored by subroutine-like constructs in the genetic code.

An important aspect of modularity is the abstraction it makes possible. Once the construction of a particular module has been determined, the module can be treated as a single object, and only its overall behaviour need be considered, wherever it appears. Modularity thus divides the problem of constructing or analysing a system into many levels, potentially making each level manageable.

Modularity is used in essentially all of the systems to be discussed below. In most cases, there are just two levels: controlling (master) and controlled (slave) components. The components on these two levels usually change on different time scales. The controlling components change at most slowly, and are often fixed once a system say with a particular set of attractors has been obtained. The controlled components change rapidly, processing input data according to dynamical rules determined by the controlling components. Such separation of time scales is common in many natural and artificial systems. In biology, for example, phenotypes of organisms grow by fast processes, but are determined by genotypes which seem to change only slowly with time. In software engineering, computer memory is divided into a part for "programs", which are supposed to remain fixed or change only slowly, and another part for intermediate data, which changes rapidly.

Even though it is not part of their construction, many systems seem dynamically to develop a modular structure, often with arbitrarily many hierarchical levels arranged say on a balanced tree [16]. But for engineering purposes, it seems best to start with systems that are modular from the outset.

Multiple scale cellular automata provide simple but quite general examples of such systems. An ordinary cellular automaton consists of a lattice of sites, with each site having say k possible values, updated according to the same definite rule. A two-scale cellular automaton can be considered to consist of two lattices, whose site values change on different characteristic time scales. The values of the sites on the "slow" lattice control the rules used at the corresponding sites on the "fast" lattice. With q possible values for the slow lattice sites, there is an array of q possible rules for each site on the fast lattice. Such a two-scale cellular automaton can always be emulated by specially chosen configurations in an ordinary cellular automaton with at most qk possible values at each site.

If the sites on the slow lattice are fixed, then a two-scale cellular automaton acts like a dynamic random field spin system (e.g. [17]), or a spin glass (e.g. [18]) (cf. [19]). Examples of patterns generated by cellular automata of this kind are shown in fig. 1. And if instead the "slow" lattice sites change rapidly, and take on essentially random

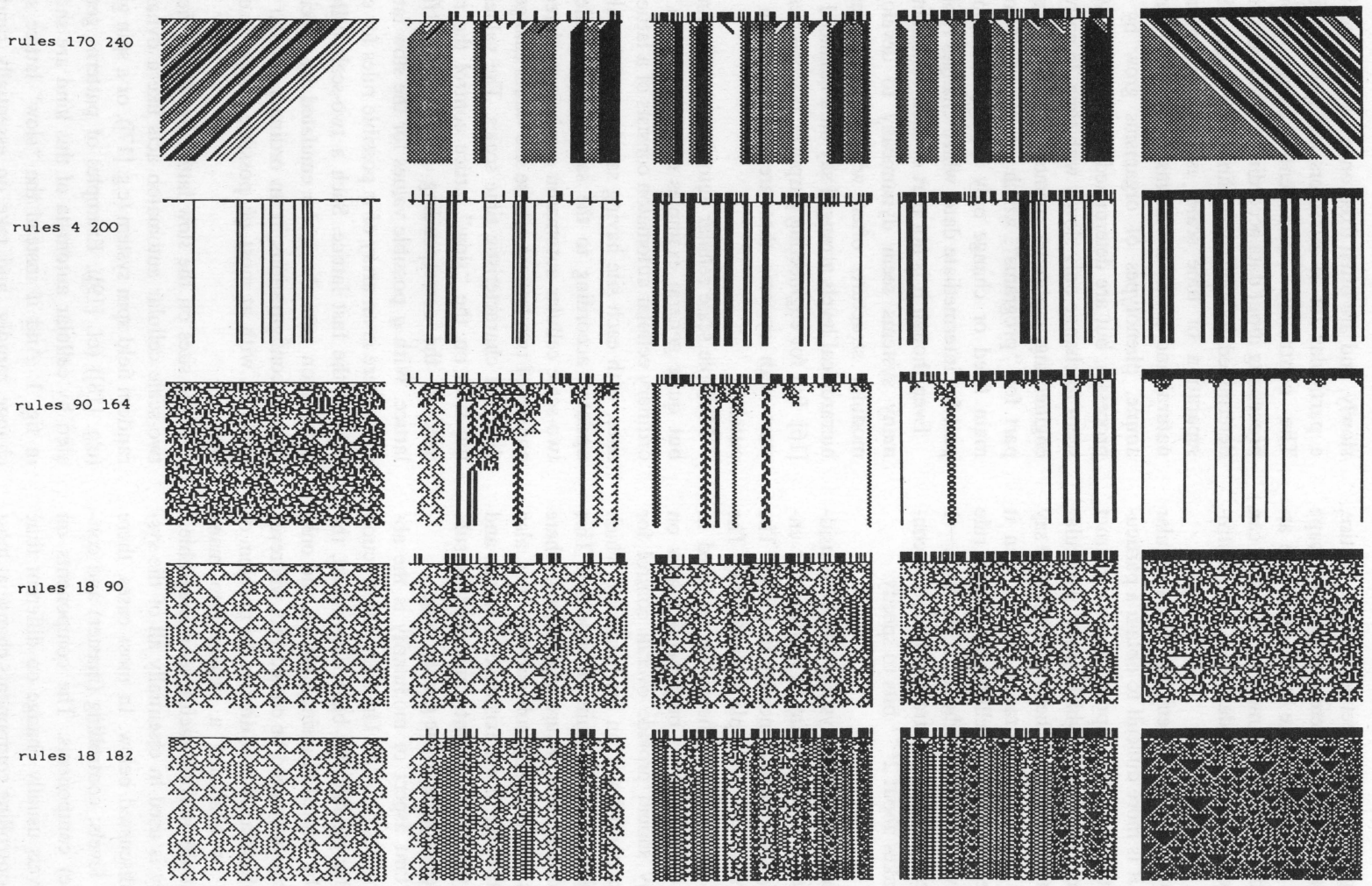

Fig. 1. Patterns generated by two-scale cellular automata with $k = 2$, $q = 2$ and $r = 1$. The configuration of the slow lattice is fixed in each case, and is shown at the top. The rule used at a particular site on the fast lattice is chosen from the two rules given according to the value of the corresponding site on the slow lattice. (The rule numbers are as defined in ref. [20].)

values, perhaps as a result of following a chaotic cellular automaton rule, then the evolution of the fast lattice is like that of a stochastic cellular automaton, or a directed percolation system (e.g. [21]).

With dissipative dynamics, the evolution of the fast lattice in a two-scale cellular automaton yields attractors. The form of these attractors is determined by the control configuration on the slow lattice. By choosing different slow lattice configurations, it is thus possible to engineer particular attractor structures.

In a typical case, a two-scale cellular automaton might be engineered to recognize inputs in different categories. Each category would be represented by a fixed point in the fast lattice dynamics. The system can then be arranged in several ways. Assume that the input is a one-dimensional symbol sequence (such as a text string). Then one possibility would be to consider a one-dimensional cellular automaton whose fixed points correspond to symbol sequences characteristic of each category. But if the required fixed points are arbitrarily chosen, few of them can be specified by a single slow configuration. If the cellular automaton has N sites, then each fixed point of the fast lattice is specified by $N \log_2 k$ bits. A configuration of the slow lattice involves only $N \log_2 q$ bits. As a consequence, the number of arbitrarily-chosen fixed points that can be specified is just $\log q / \log k$, a result independent of N.

Usually, however, it is not necessary to give all $N \log_2 k$ bits of a fixed point to specify the form of the attractor for a particular category. The number of bits actually needed presumably increases with the number of categories. It is common to find a small number of possible categories or responses to a wide variety of input data. The responses can then for example be represented by the values of a small number of sites on the fast lattice of a two-scale cellular automaton. The input data can be used to give initial values for a larger number of sites, possibly a different set. (In an analogy with the nervous system, some sites might receive input from afferent nerves while others, typically smaller in number, might generate output for efferent nerves.)

A second possibility is to consider a two-dimensional two-scale cellular automaton, in which the input is specified along a line, and the dynamics of the fast lattice transfers information only in a direction orthogonal to this line [22] (cf. [23]). This arrangement is functionally equivalent to a one-dimensional two-scale cellular automaton in which the slow lattice configuration changes at each time step. In its two-dimensional form, the arrangement is very similar to a systolic array [24], or in fact to a multilayer generalization of standard modular logic circuits, such a programmable logic arrays [24]. In an $N \times M$ system of this kind, a single slow lattice configuration can specify $M \log q / \log k$ length N fixed points in the fast configuration (cf. [2]).

In the approaches just discussed, input is given as an initial condition for the fast lattice. An alternative possibility is that the input could be given on the slow lattice, and could remain throughout the evolution of the fast lattice. The input might for example then specify boundary conditions for evolution on a two-dimensional fast lattice. Output could be obtained from the final configuration of the fast lattice. However, there will often be several different attractors for the fast lattice dynamics even given boundary conditions from a particular slow lattice configuration. Which attractor is reached will typically depend on the initial conditions for the fast lattice, which are not specified in this approach. With appropriate dynamics, however, it is nevertheless possible to obtain almost unique attractors: one approach is to add probabilistic elements or noise to the fast lattice dynamics so as to make it ergodic, with a unique invariant measure corresponding to a definite "phase" [25].

Cellular automata are arranged to be as simple as possible in their basic microscopic construction. They are discrete in space and time. Their sites are all identical, and are arranged on a regular lattice. The sites have a finite set of possible values, which

are updated synchronously according to identical deterministic rules that depend on a few local neighbours. But despite this microscopic simplicity, the overall macroscopic behaviour of cellular automata can be highly complex. On a large scale, cellular automata can for example show continuum features [12, 26], randomness [7], and effective long-range interactions [27]. Some cellular automata are even known to be universal computers [28], and so can presumably simulate any possible form of behaviour. Arbitrary complexity can thus arise in cellular automata. But for engineering purposes, it may be better to consider basic models that are more sophisticated than cellular automata, and in which additional complexity is included from the outset (cf. [2]). Multiple scale cellular automata incorporate modularity, and need not be homogeneous. Further generalizations can also considered, though one suspects that in the end none of them will turn out to be crucial.

First, cellular automaton dynamics is local: it involves no long-range connections which can transmit information over a large distance in one step. This allows (one or two-dimensional) cellular automata to be implemented directly in the simple planar geometries appropriate, for example, for very large-scale integrated circuits. Long range electronic signals are usually carried by wires which cross in the third dimension to form a complicated network. (Optical communications may also be possible.) Such an arrangement is difficult to implement technologically. When dynamically-changing connections are required, therefore, more homogeneous switching networks are used, as in computerized telephone exchanges, or the Connection Machine computer [29]. Such networks are typically connected like cellular automata, though often in three (and sometimes more) dimensions.

Some natural systems nevertheless seem to incorporate intrinsic long range connections. Chemical reaction networks are one example: reaction pathways can give almost arbitrary connectivity in the abstract space of possible chemical species [30, 31, 32]. Another example is the brain, where nerves can carry signals over long distances. In many cases, the pattern of connectivity chosen seems to involve many short-range connections, together with a few long-range ones, like motorways (freeways) or trunk lines [33]. It is always possible to simulate an arbitrary arrangement of long range connections through sequences of short range connections; but the existence of a few intrinsic long range connections may make large classes of such simulations much more efficient [33].

Many computational algorithms seem to involve arbitrary exchange of data. Thus for example, in the fast Fourier transform, elements are combined according to a shuffle-exchange graph (e.g. [29]). Such algorithms can always be implemented by a sequence of local operations. But they seem to be most easily conceived without reference to the dynamics of data transfer. Indeed, computers and programming languages have traditionally been constructed to enforce the idealization that any piece of data is available in a fixed time (notions such as registers and pipelining go slightly beyond this). Conventional computational complexity theory also follows this idealization (e.g. [34]). But in developing systems that come closer to actual physical constraints, one must go beyond this idealization. Several classes of algorithms are emerging that can be implemented efficiently and naturally with local communications (e.g. [35]). A one-dimensional cellular automaton ("iterative array") can be used for integer multiplication [36]. Two dimensional cellular automata ("systolic arrays") can perform a variety of matrix manipulation operations [24].

Although the basic rules for cellular automata are local, the rules are usually applied in synchrony, as if controlled by a global clock. A generalization would allow asynchrony, so that different sites could be updated at different times (e.g. [37]). Only a few sites might, for example, be updated at each time step. This typically yields more gradual transitions from one cellular automaton configuration to another, and can prevent certain instabilities. Asynchronous updating makes it more difficult for information to propagate

through the cellular automaton, and thus tends to prevent initial perturbations from spreading. As a result, the evolution is more irreversible and dissipative. Fixed point and limit cycle (class 1 and 2) behaviour therefore become more common.

For implementation and analysis, it is often convenient to maintain a regular updating schedule. One possibility is to alternate between updates of even and odd-numbered sites (e.g. [38]). "New" rather than "old" values for the nearest neighbours of a particular cell are then effectively used. This procedure is analogous to the implicit, rather than explicit, method for updating site values in finite difference approximations to partial differential equations (e.g. [39]). It is known often to lead to better convergence (e.g. [39]). The scheme also yields, for example, systematic relaxation to thermodynamic equilibrium in a cellular automaton version of the microcanonical Ising model [40]: simultaneous updating of all sites allows undamped oscillations in this case [38].

One can also consider systems in which sites are updated in a random order, perhaps one at a time. Such systems can often be analysed using "mean field theory", by assuming that the behaviour of each individual component is random, with a particular average (cf. [2]). Statistical predictions can then often be made from iterations of maps involving single real parameters. As a result, monotonic approach to fixed points is more easily established.

Random asynchronous updating nevertheless makes detailed analysis more difficult. Standard computational procedures usually require definite ordering of operations, which can be regained in this case only through mechanisms such as semaphores (e.g. [41]), typically at considerable cost.

Rather than introducing randomness into the updating scheme, one can instead include it directly in the basic cellular automaton rule. The evolution of such stochastic cellular automata can then be analogous to the steps in a Monte Carlo simulation of a spin system at nonzero temperature [42]. Randomness typically prevents the system from being trapped in metastable states, and can therefore accelerate the approach to equilibrium.

In most practical implementations, however, supposedly random sequences must be obtained from simple algorithms (e.g. [36]). Chaotic cellular automata can produce sequences with a higher degree of randomness [7], presumably making explicit insertion of external randomness unnecessary.

Another important simplifying feature of cellular automata is the assumption of discrete states. This feature is convenient for implementation by digital electronic circuits. But many natural systems seem to involve continuously-variable parameters. There are usually components, such as molecules or vesicles of neurotransmitter, that behave as discrete on certain levels. But very large numbers of these components can act in bulk, so that for example only their total concentration is significant, and this can be considered as an essentially continuous variable. In some systems, such bulk quantities have simple behaviour, described say by partial differential equations. But the overall behaviour of many cellular automata and other systems can be sufficiently complex that no such bulk or average description is adequate. Instead the evolution of each individual component must be followed explicitly (cf. [11]).

For engineering purposes, it may nevertheless sometimes be convenient to consider systems that involve essentially continuous parameters. Such systems can for example support cumulative small incremental changes. In a cellular automaton, the states of n elements are typically represented by $\mathcal{O}(n)$ bits of information. But bulk quantities can be more efficiently encoded as digital numbers, with only $\mathcal{O}(\log n)$ bits. There may be some situations in which data is best packaged in this way, and manipulated say with arithmetic operations.

A potential advantage of using continuous variables is the possibility of applying standard mathematical analysis techniques, as in traditional mathematical physics. But the advantage is probably largely illusory. Extensive results can typically be obtained only when the behaviour of the

system is too simple to show the complexity required (cf. [43]).

Having selected a basic system, the problem of engineering then consists in designing or programming it to perform particular tasks. The conventional approach is systematically to devise a detailed step-by-step plan. But such a direct constructive approach cannot make the most efficient use of a complex system.

Logic circuit design provides an example (e.g. [44]). The task to be performed is the computation of a Boolean function with n inputs specified by a truth table. In a typical case, the basic system is a programmable logic array (PLA): a two-level circuit which implements disjunctive normal form (DNF) Boolean expressions, consisting of disjunctions (ORs) of conjunctions (ANDs) of input variables (possibly negated) [24, 25]. The direct approach would be to construct a circuit which explicitly tests for each of the 2^n cases in the truth table. The resulting circuit would contain $\mathcal{O}(n2^n)$ gates. Thus for example, the circuit which yields 1 if two or more of its three inputs a_i are one would be represented by $a_1a_2a_3 + a_1a_2\bar{a}_3 + a_1\bar{a}_2a_3 + \bar{a}_1a_2a_3$, where multiplication represents AND and addition represents OR. (This function can be viewed as the $k = 2$, $r = 1$ cellular automaton rule number 232 [20]).

Much smaller circuits are however often sufficient. But direct constructive techniques are not usually appropriate for finding them. Instead one uses methods that manipulate the structure of circuits, without direct regard to the meaning of the Boolean functions they represent. Many methods start by extracting prime implicants [44, 45]. Logical functions of n variables can be considered as colourings of the Boolean n-cube. Prime implicants represent this colouring by decomposing it into pieces along hyperplanes of different dimensions. Each prime implicant then corresponds to a single conjunction of input variables: a circuit for the original Boolean function can be formed from a disjunction of these conjunctions. This circuit is typically much smaller than the one obtained by direct construction. (For the majority function mentioned above, it is $a_1a_2 + a_1a_3 + a_2a_3$.) And while it performs the same task, it is usually no longer possible to give an explicit step-by-step "explanation" of its operation.

A variety of algebraic and heuristic techniques are used for further simplification of DNF Boolean expressions [45]. But it is in general very difficult to find the absolutely minimal expression for any particular function. In principle, one could just enumerate all possible progressively more complicated expressions or circuits, and find the first one which reproduces the required function. But the number of possible circuits grows exponentially with the number of gates, so such an exhaustive search rapidly becomes entirely infeasible. It can be shown in fact that the problem of finding the absolute minimal expression is NP hard, suggesting that there can never be a general procedure for it that takes only polynomial time [6]. Exhaustive search is thus effectively the only possible exact method of solution.

In most cases, however, it is not necessary to find the absolutely minimal circuit: any sufficiently simple circuit will suffice. As a result, one can consider methods that find only approximately minimal circuits.

Most approximation techniques are basically iterative: they start from one circuit, then successively make changes which preserve the functionality of the circuit, but modify its structure. The purpose is to find minima in the circuit size or "cost" function over the space of possible circuits. The effectiveness of different techniques depends on the form of the circuit size "landscape".

If the landscape was like a smooth bowl, then the global minimum could be found by starting at any point, and systematically descending in the direction of the local gradient vector. But in most cases the landscape is presumably more complicated. It could for example be essentially flat, except for one narrow hole containing the minimum (like a golf course). In such a case, no simple iterative procedure could find the minimum.

Another possibility, probably common in practice, is that the landscape has a form reminiscent of real topographical landscapes, with a complicated pattern of peaks and valleys of many different sizes. Such a landscape might well have a self similar or fractal form: features seen at different magnifications could be related by simple scalings. Straightforward gradient descent would always get stuck in local minima on such a landscape, and cannot be used to find a global minimum (just as water forms localized lakes on a topographical landscape). Instead one should use a procedure which deals first with large-scale features, then progressively treats smaller and smaller scale details.

Simulated annealing is an example of such a technique [46]. It is based on the gradient descent method, but with stochastic noise added. The noise level is initially large, so that all but the largest scale features of the landscape are smeared out. A minimum is found at this level. Then the noise level ("temperature") is reduced, so that smaller scale features become relevant, and the minimum is progressively refined. The optimal temperature variation ("annealing schedule") is probably determined by the fractal dimension of the landscape.

In actual implementations of the simulated annealing technique, the noise will not be truly random, but will instead be generated by some definite, and typically quite simple, procedure. As a consequence, the whole simulated annealing computation can be considered entirely deterministic. And since the landscape is probably rather random, simple deterministic perturbations of paths will probably suffice (cf. [47]).

In the simulated annealing approach, each individual "move" might consist of a transformation involving say two logic gates. An alternative procedure is first to find the minimal circuit made from "modules" containing many gates, and then to consider rearranging progressively smaller submodules. The hierarchical nature of this deterministic procedure can again mirror the hierarchical form of the landscape.

The two approaches just discussed involve iterative improvement of a single solution. One can also consider approaches in which many candidate solutions are treated in parallel. Biological evolution apparently uses one such approach. It generates a tree of different genotypes, and tests the "fitness" of each branch in parallel. Unfit branches die off. But branches that fare well have many offspring, each with a genotype different by a small random perturbation ("genetic algorithm" [48]). These offspring are then in turn tested, and can themselves produce further offspring. As a result, a search is effectively conducted along many paths at once, with a higher density of paths in regions where the fitness is improving. (This is analogous to decision tree searching with, say, $\alpha\beta$-pruning [49].) Random perturbations in the paths at each generation may prevent getting stuck in local minima, but on a fractal landscape of the type discussed above, this procedure seems less efficient than one based on consideration of progressively finer details.

In the simplest iterative procedures, the possible changes made to candidate solutions are chosen from a fixed set. But one can also imagine modifying the set of possible changes dynamically [50] (cf. [51]). To do this, one must parametrize the possible changes, and in turn search the space of possibilities for optimal solutions.

The issues discussed for logic circuit design also arise in engineering complex systems such as two-scale cellular automata. A typical problem in this case is to find a configuration for the slow lattice that yields particular fixed points for evolution on the fast lattice. For simple linear rules, for example, a constructive algebraic solution to this problem can be given. But for arbitrary rules, the problem is in general NP hard. An exact solution can thus presumably be found only by exhaustive search. Approximation procedures must therefore again be used.

The general problem in complex systems engineering is to find designs or arrangements of complex systems that behave in specified ways. The behaviour sought usually corresponds to a

comparatively simple, usually polynomial time, computation. But to find exactly the necessary design may require a computation that effectively tests exponentially many possibilities. Since the correctness of each possibility can be tested in polynomial time, the problem of finding an appropriate design is in the computational complexity class NP (non-deterministic polynomial time). But in many cases, the problem is in fact NP complete (or at least NP hard). Special instances of the problem thus correspond to arbitrary problems in NP; any general solution could thus be applied to all problems in NP.

There are many NP complete problems, all equivalent in the computational difficulty of their exact solution [6]. Examples are satisfiability (finding an assignment of truth values to variables which makes a Boolean expression true), Hamilton circuits (finding a path through a graph that visits each arc exactly once), and spin glass energy minima (finding the minimum energy configuration in a spin glass model). In no case is an algorithm known which takes polynomial time, and systematically yields the exact solution.

Many approximate algorithms are nevertheless known. And while the difficulty of finding exact solutions to the different problems is equivalent, the ease of approximation differs considerably. (A separate consideration is what fraction of the instances of a problem are difficult to solve with a particular algorithm. Some number theoretical problems, for example, have the property that all their instances are of essentially equivalent difficulty [52].) Presumably the "landscapes" for different problems fall into several classes. There is already some evidence that the landscapes for spin glass energy and the "travelling salesman" problem have a hierarchical or ultrametric, and thus fractal, form [53]. This may explain why the simulated annealing method is comparatively effective in these cases.

Even though their explicit forms cannot be found, it could be that certain, say statistical, features of solutions to NP problems could easily be predicted. Certainly any solution must be distinguished by the P operation used to test its validity. But at least for some class of NP hard problems, one suspects that solutions will appear random according to all standard statistical procedures. Despite the "selection" process used to find them, this would imply that their statistical properties would be typical of the ensemble of all possibilities (cf. [54]).

There are many potential applications for complex systems engineering. The most immediate ones are in pattern recognition. The basic problem is to take a wide variety of inputs, say versions of spoken words, and to recognize to which category or written word they correspond (e.g. [55]).

In general, there could be arbitrary mapping from input to output, so that each particular case would have to be specified explicitly. But in practice the number of possible inputs is far too large for this to be feasible, and redundancy in the inputs must be used. One must effectively make some form of model for the inputs, which can be used, for example, to delineate categories which yield the same output.

One might for example allow particular kinds of distortion or error in the input. Thus in studies of DNA sequences, changes associated with substitution, deletion, insertion, or transposition of elements are usually considered [56]. A typical problem of pattern recognition is then to determine the category of a particular input, regardless of such changes.

Several approaches are conventionally used (e.g. [55]).

One approach is template matching. Each category is defined by a fixed "template". Then inputs are successively compared with each of these templates, and the quality of match is determined, typically by statistical means. The input is assigned to the category with the best match.

A second approach is feature extraction. A fixed set of "features" is defined. The presence or absence of each feature in a particular input is then determined, often by template-matching techniques. The category of the input is found

from the set of features it contains, typically according to a fixed table.

In both these approaches, the pattern recognition procedure must be specially designed to deal with each particular set of categories considered. Templates or features that are sufficiently orthogonal must be constructed, and small changes in the behaviour required may necessitate large changes in the arrangement used.

It would be more satisfactory to have a generic system which would take a simple specification of categories, and recognize inputs using "reasonable" boundaries between the categories. This can potentially be achieved with dissipative dynamical systems. Different categories are specified as fixed points (or other attractors) for the system. Then the dynamics of the system determines the form of the basins of attraction for each of these fixed points. Any input within a particular basin will be led to the appropriate fixed point. In general, however, different inputs will take varying numbers of steps to reach the fixed point. Conventional pattern recognition schemes typically take a fixed time, independent of input. But more flexible schemes presumably require variable times.

It should be realized, however, that such schemes implicitly make definite models for the input data. It is by no means clear that the dynamics of such systems yield basin structures appropriate for particular data. The basins are typically complicated and difficult to specify. There will usually be no simple metric, analogous to the quality of template matches, which determines the basis for a particular input from the fixed point to which it is "closest". Fig. 2 shows a representation of the basins of attraction in a two-scale cellular automaton. No simple metric is evident.

While the detailed behaviour of a system may be difficult to specify, it may be possible to find a high-level phenomenological description of some overall features, perhaps along the lines conventional in psychology, or in the symbolic approach to artificial intelligence (e.g. [58]). One can imagine, for example, proximity relations for attractors analogous to semantic networks (e.g [58]). This

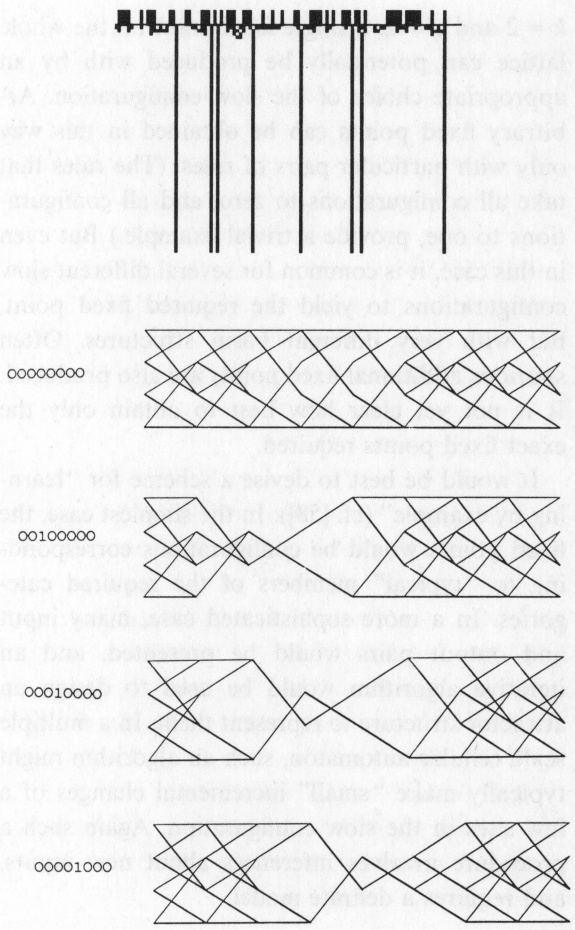

Fig. 2. Representation of the basins of attraction for fixed points in a length 8 two-scale cellular automaton with $q = 2$ and rules 36 and 72. The configurations in each basin correspond to possible paths traversing each graph from left to right. Descending segments represent value one, ascending segments value zero.

high level description might have the same kind of relation to the underlying dynamics as phenomenological descriptions such as vortex streets have to the basic equations of fluid flow.

To perform a particular pattern recognition task, one must design a system with the appropriate attractor structure. If, for example, categories are to be represented by certain specified fixed points, the system must be constructed to have these fixed points. In a two-scale cellular automaton with

$k = 2$ and $q = 2$, a single fixed point on the whole lattice can potentially be produced with by an appropriate choice of the slow configuration. Arbitrary fixed points can be obtained in this way only with particular pairs of rules. (The rules that take all configurations to zero, and all configurations to one, provide a trivial example.) But even in this case, it is common for several different slow configurations to yield the required fixed point, but with very different basin structures. Often spurious additional fixed points are also produced. It is not yet clear how best to obtain only the exact fixed points required.

It would be best to devise a scheme for "learning by example" (cf. [58]). In the simplest case, the fixed points would be configurations corresponding to "typical" members of the required categories. In a more sophisticated case, many input and output pairs would be presented, and an iterative algorithm would be used to design an attractor structure to represent them. In a multiple scale cellular automaton, such an algorithm might typically make "small" incremental changes of a few sites in the slow configuration. Again such a procedure involves inferences about new inputs, and requires a definite model.

Acknowledgements

I am grateful for discussions with many people, including Danny Hillis, David Johnson, Stuart Kauffman, Alan Lapedes, Marvin Minsky, Steve Omohundro, Norman Packard, Terry Sejnowski, Rob Shaw, and Gerry Tesauro.

References

[1] S. Wolfram, Cellular automata as models of complexity, Nature 311 (1984) 419; S. Wolfram, ed., Theory and Applications of Cellular Automata, (World Scientifc, Singapore, 1986).

[2] J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. 79 (1982) 2554.

[3] W. Green, Digital Image Processing (Van Nostrand, Princeton, 1983).

[4] R. Hamming, Coding and Information Theory (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[5] H.-O. Peitgen and P. Richter, The Beauty of Fractals: Images of Complex Dynamical Systems (Springer, Berlin, 1985).

[6] M. Garey and D. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness (Freeman, San Francisco, 1979).

[7] S. Wolfram, Random sequence generation by cellular automata, Adv. Applied Math. 7 (1986) 123.

[8] K. Preston and M. Duff, Modern Cellular Automata (Plenum, New York, 1984).

[9] S. Wolfram, Universality and complexity in cellular automata, Physica 10D (1984) 1.

[10] S. Wolfram, Computation theory of cellular automata, Commun. Math. Phys. 96 (1984) 15.

[11] S. Wolfram, Undecidability and intractability in theoretical physics, Phys. Rev. Lett. 54 (1985) 735.

[12] N. Packard and S. Wolfram, Two-dimensional cellular automata, J. Stat. Phys. 38 (1985) 901.

[13] S. Wolfram, Computer software in science and mathematics, Sci. Amer. (September 1984).

[14] R. Sverdlove, Inverse problems for dynamical systems in the plane, in: Dynamical Systems, A.R. Bednarek and L. Cesari, eds. (Academic Press, New York, 1977).

[15] E. Jen, Invariant strings and pattern-recognizing properties of one-dimensional cellular automata, J. Stat. Phys. 43 (1986) 243.

[16] M. Mezard, G. Parisi, N. Sourlas, G. Touluse and M. Virasoro, Nature of spin glass phase, Phys. Rev. Lett. 52 (1984) 1156.

[17] J. Villain, The random field Ising model, in: Scaling Phenomena and Disordered Systems, NATO ASI, Geilo, Norway (April 1985).

[18] Proc. Heidelberg Colloq. on Spin Glasses, Heidelberg (June 1983); K.H. Fischer, Phys. Status Solidi 116 (1983) 357.

[19] G. Vichniac, P. Tamayo and H. Hartman, Annealed and quenched inhomogeneous cellular automata, J. Stat. Phys., to be published.

[20] S. Wolfram, Statistical mechanics of cellular automata, Rev. Mod. Phys. 55 (1983) 601.

[21] W. Kinzel, Phase transitions of cellular automata, Z. Phys. B58 (1985) 229.

[22] T. Hogg and B. Huberman, Parallel computing structures capable of flexible associations and recognition of fuzzy inputs, J. Stat. Phys. 41 (1985) 115.

[23] T. Sejnowski, to be published.

[24] C. Mead and L. Conway, An Introduction to VLSI Systems (Addison–Wesley, New York, 1980).

[25] D. Ackley, G. Hinton and T. Sejnowski, A learning algorithm for Boltzmann machines, Cognitive Sci. 9 (1985) 147.

[26] U. Frisch, B. Hasslacher and Y. Pomeau, "A lattice gas automaton for the Navier–Stokes equation", Los Alamos preprint LA-UR-85-3503; J. Salem and S. Wolfram, "Thermodynamics and hydrodynamics with cellular automata", IAS preprint (November 1985).

[27] S. Wolfram, "Glider gun guidelines", report distributed through Computer Recreations section of Scientific American; J. Park, K. Steiglitz and W. Thurston, "Soliton-like behaviour in cellular automata", Physica 19D (1986) 423.
[28] A. Smith, Simple computation-universal cellular spaces, J. ACM 18 (1971) 331; E.R. Berlekamp, J.H. Conway and R.K. Guy, Winning Ways for Your Mathematical Plays (Academic Press, New York, 1982).
[29] D. Hillis, The Connection Machine (MIT press, Cambridge, MA, 1985).
[30] S. Kauffman, Metabolic stability and epigenesis in randomly constructed genetic nets, J. Theoret. Biol. 22 (1969) 437; Autocatalytic sets of proteins, J. Theor. Biol., in press.
[31] A. Gelfand and C. Walker, "Network modelling techniques: from small scale properties to large scale systems", University of Connecticut report (1982).
[32] E. Goles Chacc, "Comportement dynamique de reseaux d'automates", Grenoble University report (1985).
[33] C. Leiserson, Fat trees: universal networks for hardware efficient supercomputing, IEEE Trans. Comput. C-36 (1985) 892.
[34] J. Hopcroft and J. Ullman, Introduction to Automata Theory, Languages and Computation (Addison–Wesley, New York, 1979).
[35] S. Omohundro, "Connection Machine algorithms primer", Thinking Machines Corporation (Cambridge, Mass.) report in preparation.
[36] D. Knuth, Seminumerical Algorithms (Addison–Wesley, New York, 1981).
[37] T.E. Ingerson and R.L. Buvel, Structure in asynchronous cellular automata, Physica 10D (1984) 59.
[38] G. Vichniac, Simulating physics with cellular automata, Physica 10D (1984) 96.
[39] C. Gerald, Applied Numerical Analysis (Addison–Wesley, New York, 1978).
[40] M. Creutz, Deterministic Ising dynamics, Ann. Phys. 167 (1986) 62.
[41] C.A.R. Hoare, Communicating sequential processes, CACM 21 (1978) 666.
[42] E. Domany and W. Kinzel, Equivalence of cellular automata to Ising models and directed percolation, Phys. Rev. Lett. 53 (1984) 311.
[43] M. Minsky and S. Papert, Perceptrons (MIT press, Cambridge, MA, 1972).
[44] Z. Kohavi, Switching and Finite Automata Theory (McGraw-Hill, New York, 1970).
[45] R. Brayton, G. Hachtel, C. McMullen and A. Sangiovanni-Vincentelli, Logic Minimization Algorithms for VLSI Synthesis (Kluwer, Deventer, 1984).
[46] S. Kirkpatrick, C. Gelatt and M. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671.
[47] J. Hopfield and D. Tank, Neural computation of decisions in optimization problems, Biol. Cybern. 52 (1985) 141.
[48] J. Holland, "Genetic algorithms and adaptation", Tech. Rep. #34, Univ. Michigan (1981).
[49] A. Barr and E. Feigenbaum, The Handbook of Artificial Intelligence (Heuris Tech Press, 1983), vol. 1.
[50] J. Holland, "Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule-based systems", University of Michigan report.
[51] D. Lenat, Computer software for intelligent systems, Scientific American (September 1984).
[52] M. Blum and S. Micali, How to generate cryptographically strong sequences of pseudo-random bits, SIAM J. Comput. 13 (1984) 850.
[53] S. Kirkpatrick and G. Toulouse, Configuration space analysis of travelling salesman problems, J. Physique 46 (1985) 1277.
[54] S. Kauffman, Self-organization, selection, adaptation and its limits: a new pattern of inference in evolution and development, in: Evolution at a Crossroads, D.J. Depew and B.H. Weber, eds., (MIT press, Cambridge, MA, 1985).
[55] C.J.D.M. Verhagen et al., Progress report on pattern recognition, Rep. Prog. Phys. 43 (1980) 785.
[56] D. Sankoff and J. Kruskal, eds., Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison (Addison–Wesley, New York, 1983).
[57] M. Minksy, Society of Mind, in press.
[58] L. Valiant, A theory of the learnable, CACM 27 (1984) 1134.