Artificial Life

Prof. Dr. Rolf Pfeifer
Hanspeter Kunz
Marion M. Weber
Dale Thomas

Institut für Informatik
der Universität Zürich

26. Juni 2001

Contents

Chapter 1: Introduction

The stuff of life is not stuff.

Christopher G. Langton

In this first chapter we give a brief overview over the historical origin of the relatively young field in science called Artificial Life. Besides which, we try to give the reader an idea of the controversial understandings of Natural Life and the comparatively straight-forward definition of Artificial Life. In the third part of this first chapter we introduce the main methodology used in Artificial Life "the synthetic approach" which can briefly be explained by the phrase "understanding by building".

1.1 Historical origins

The branch of science named "Artificial Life" (AL) came into being at a workshop in September 1987 at the Los Alamos National Laboratory. Named the first workshop on Artificial Life, organized by Christopher G. Langton from the Center of the Santa Fe Institute (SFI). The SFI is a private, independent organization dedicated to multidisciplinary scientific research in the natural, computational and social sciences. The driving force behind its creation in 1984 was the need to understand those complex systems that shape human life and much of our immediate world - evolution, the learning process, the immune system and the world economy. The intent is to make new tools now being developed at the frontiers of the computational sciences and in the mathematics of nonlinear dynamics more readily available for research in the applied physical, biological and social sciences. The purpose of this workshop was to bring together the scientists working in a new and unknown niche. Langton writes:

> "The workshop itself grew out of my frustration with the fragmented nature of the literature on biological modeling and simulation. For years I had prowled around libraries, shifted through computer-search results, and haunted bookstores, trying to get an overview of a field, which I sensed, existed but which did not seem to have any coherence or unity. Instead, I literally kept stumbling over interesting work almost by accident, often published in obscure journals if published at all." (Langton, 1989, p. xv)

At this workshop 160 computer scientists, biologists, physicists, anthropologists, and other ``-ists" presented mathematical models for the origin of life, self-reproducing automata, computer programs using the mechanisms of Darwinian evolution, simulations of flocking birds and schooling fish, models for the growth and development of artificial plants and much more. During these five days it became apparent that all the participants with their previously isolated research efforts shared a remarkably similar set of problems and visions.

It became increasingly clear, that linear models simply could not describe many natural phenomena. In a linear model, the whole is the sum of its parts, and small changes in model parameters have little effect on the behavior of the model. However, many phenomena such as weather, growth of plants, traffic jams, flocking of birds, stock market crashes, development of multi-cellular organisms, pattern formation in nature (for example on sea shells and butterflies), evolution, intelligence, and so forth resisted any

linearization; that is, no satisfying linear model was ever found.

One vision that emerged at the workshop was to look at these problems from a different angle, trying to model them as *nonlinear* phenomena. Nonlinear models can exhibit a number of features not known from linear ones: for example chaos (small changes in parameters or initial conditions can lead to qualitatively different outcomes) and the occurrence of higher level features (emergent phenomena, attractors). `Higher level' means, that these features were not explicitly modeled. However, nonlinear models have the disadvantage that they typically cannot be solved analytically, in contrast to linear models. They are investigated using computer simulations and that is the reason why nonlinear modeling is a relatively new approach. Nonlinear modeling became manageable only when fast computers were available. The fact that those nonlinear models, and in AL nonlinear models are almost always used, cannot be treated analytically has one rather surprising positive side effect: One does not have to be a mathematician to work with AL models. Langton concludes:

> "I think that many of us went away from that tumultuous interchange of ideas with a very similar vision, strongly based on themes such as *bottom-up* rather than *top-down* modeling, *local* rather than *global* control, *simple* rather than *complex* specifications, *emergent* rather than *pre-specified* behavior, *population* rather than *individual* simulation, and so forth.
> Perhaps, however, the most fundamental idea to emerge at the workshop was the following: Artificial systems which exhibit lifelike behaviors are worthy of investigation on their own rights, whether or not we think that the processes that they mimic have played a role in the development or mechanics of life as *we* know it to be. Such systems can help us expand our understanding of life as it *could* be. By allowing us to view the life that has evolved here on Earth in the larger context of *possible* life, we may begin to derive a truly general theoretical biology capable of making universal statements about life wherever it may be found and whatever it may be made of". (Langton, 1989, p. xvi)

1.2 Natural and artificial life

Natural life

Preliminary remark: This topic is highly controversial and there is a lot of literature on it. Thus, the discussion in this section is very limited and only intended to provide an idea of some of the issues involved. Since the topic of the class is artificial life, we should have some idea of what natural life is. We will see that there are no firm conclusions.

There is no generally accepted definition of life, although everyone has a concept of whether he or she would call a particular thing living or not. Stevan Harnad, a well-known psychologist and philosopher is reluctant to give an answer:

> "What is it to be 'really alive'? I'm certainly not going to be able to answer this question here, but I can suggest one thing that's *not*: It's not a matter of satisfying a definition, at least not at this time, for such a definition would have to be preceded by a true theory of life, which we do not yet have." (Harnad, 1995, p. 293)

Aristotle first made the observation that a living thing can nourish itself and almost everybody would agree that the ability to reproduce is a necessary condition for life. However, there is a problem with this last issue in that it is certainly true for species but perhaps not so true for individual organisms. Some animals are incapable of reproducing, e.g. mules, soldier ants/bees or simply infertile organisms. Does this somehow make their whole life void? Packard and Bedau believe that life is a property that an organism has if it is a member of a system of interacting organisms (Bedau and Packard, 1991).

In *Random House Webster's Dictionary* the following definitions for life are found. Life is

— the general condition that distinguishes organism from inorganic objects and dead organisms, being manifested by growth through metabolism, a means of reproduction, and internal regulation in response to the environment.

— the animate existence or period of animate existence of an individual.

— a corresponding state, existence, or principle of existence conceived of as belonging to the soul.

— the general or universal condition of human existence.

— any specified period of animate existence.

— the period of existence, activity, or effectiveness of something inanimate, as a machine, lease, or play.

— animation; liveliness; spirit: (example: The party was full of life).

— the force that makes or keeps something alive; the vivifying or quickening principle.

For the most part of human history, the question “What is life?” was never an issue. Before the science of physics became important, everything was alive: the stars, the rivers, the mountains, the stones, etc. So the question was of no importance. Only when the deterministic mechanics of moving bodies became dominant the question was raised: If all matter follows simple physical laws, and we need no vitalistic explanation of the world's behavior, of movement in the world, then what is the difference between living and non-living things? That there is a difference is obvious, but to pin down what this difference exactly is, seems less obvious. According to Erwin Schrödinger, a famous physicist and one of the key figures in the development of quantum mechanics, it is something that cannot be explained based on the laws of physics alone. Something “extra” is required (Schrödinger, 1944). Again, what this “extra” is remains a conundrum. Still, according to Schrödinger, it can be related to the arrangements of the atoms and the interplay of these arrangements that differ in a fundamental way from those arrangements of atoms studied by physicists and chemists. Thus, it seems that Schrödinger sees the main differences in the organization of the particles rather than their intrinsic properties. This position is also endorsed by the better part of the researchers in artificial life.

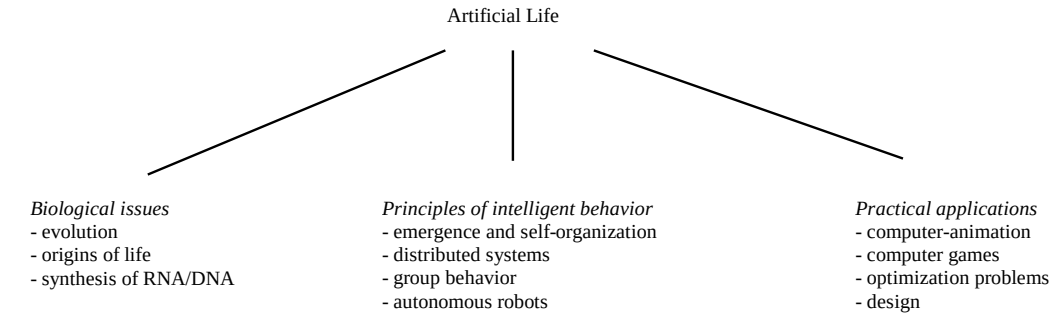
Artificial Life

While natural life is very hard to precisely define, Artificial Life (AL) can be characterized in better ways. Here is the definition by Christopher Langton, the founder of the research discipline of Artificial Life:

> “Artificial Life is the study of man-made systems that exhibit behaviors characteristic of natural living systems. It complements the traditional biological sciences concerned with the analysis of living organisms by attempting to *synthesize* life-like behaviors within computers and other artificial media. By extending the empirical foundation upon which biology is based *beyond* the carbon-chain life that has evolved on Earth, Artificial Life can contribute to theoretical biology by locating *life-as-we-know-it* within the larger picture of *life-as-it-could-be*. (Langton, 1989, p. 1)

In other words, the goal of AL is not only to provide biological models but also to investigate general principles of life. These principles can be investigated in their own right, without necessarily having to have

a direct natural equivalent. This is analogous to the field of artificial intelligence where in addition to building models of naturally intelligent systems, general principles of intelligence are explored. Figure 1 shows the three essential goals of the field of AL. In addition to studying biological issues and abstracting



principles of intelligent behavior, based on these principles, practical applications are to be developed.

Figure 1: The goals of Artificial Life.

1.3 Methodological issues and basic definitions

The Synthetic Approach

The field of AL is by definition synthetic. It works on the basis of "understanding by building": In order to understand a phenomenon, say the food distribution in an ant society, we build aspects of the ant society's behavior. Typically, computer simulations are employed, but sometimes researchers use robots.

Biology is the scientific study of life based on carbon-chain chemistry. AL tries to transcend this limitation to Earth bound life based on the assumption, that life is a property of the *organization* of matter, rather than a property of the matter itself. Furthermore, biology traditionally starts at the top, for example at the organism level, seeking explanations in terms of lower level entities in an analytic way, whereas AL starts at the bottom, for example at the molecular level, working its way up the hierarchy by synthesizing complex systems from many simple interacting entities. Biology works in an analytic way: Scientists are aiming to understand living beings by teasing them apart, looking for constituents, the constituents of the constituents, and so on down to cells, molecules, atoms, and elementary particles. Only recently scientists started to put these parts together again, to look how simple components can be combined to build larger systems.

Imagine, for example, that we wanted to build a model an ant colony. We would start specifying simple behavioral repertoires for the ants, and then, typically in a computer simulation, put many of these simple ants or "vants" (virtual ants) in a simulated environment. Then the vants would behave according to their (simple) rules and according to their environment. If we captured the essential spirit of ant behavior in the rules for our vants, the vants in the simulation in the simulated ant colony should behave as real ants in a real ant colony.

The analytic approach to science has been extremely successful in many disciplines like physics or chemistry. Most scientists believe that the universe is governed by laws of nature that apply for stars and galaxies as well as for elementary particles and atoms and living organisms. The question is whether — once we know the fundamental laws — we can explain everything in these terms: Can everything, including biological systems, be reduced to these principles? There is general agreement that this is not the case and that additional — organizational — principles are required (see also Schrödinger's comments above). The synthetic methodology is particularly suited to investigate such principles.

Levels of Organization

Life, as we know it on Earth, is organized into at least four levels of structure: the *molecular* level, the *cellular* level, the *organism* level, and the *population-ecosystem* level. Of course, and fortunately, AL studies do not have to start at the lowest level. At each level behavior of the entities and their interaction can be specified and the behavior of interest then is allowed to emerge.

AL researchers have developed a variety of models at each of these levels of organization, from the molecular to the population level, sometimes even covering two or three levels in a single model. The interesting point is that at each level, entirely new properties appear. Also, at each stage new laws, concepts and generalizations are necessary, requiring inspiration and creativity to just as great a degree as in the previous one. Psychology is not applied biology and biology is not applied chemistry (Anderson, cited in Waldrop, 1992).

Time perspectives on explanation

Explanations of behavior can be given at different temporal perspectives, (1) short-term, (2) ontogenetic and learning, and (3) phylogenetic. The short-term perspective explains why a particular behavior is displayed by an agent based on its current internal and sensory-motor state (in this context the term agent, which can be understood as human, animal or artificial creature, means robot). It is concerned with the immediate cause of behavior. The second perspective, ontogenetic and learning, not only resorts to current internal state, but to some events in the more distant past in order to explain current behavior. The third, the phylogenetic one, asks how particular behaviors evolved during the history of the species. Often, an additional, non-temporal, perspective is added. One can ask what a particular behavior is for, i.e. how it contributes to the agent's overall fitness. These perspectives are closely related to what is called "the four whys" in biology (Huxley, 1942; Tinbergen, 1963). For a full explanation of a particular behavior all of these levels have to be considered.

The Frame-of-Reference Problem

Whenever we want to explain behavior we have to be aware of the frame-of-reference problem (Clancey, 1991). The frame-of-reference problem conceptualizes the relationship designer, observer, agent to be modeled, or to-be-built artifact, and environment. There are three distinct issues (Pfeifer and Scheier, 1999), perspective, behavior vs. mechanism and complexity.

Perspective issue

We have to distinguish between the perspective of an observer looking at an agent and the perspective of the agent itself. In particular, descriptions of behavior from an observer's perspective must not be taken as the internal mechanisms underlying the described behavior of the agent.

Behavior-versus-mechanism issue

The observed behavior of an agent is always the result of a system-environment interaction. It cannot be explained on the basis of internal mechanisms only. Doing so would constitute a category error.

Complexity issue

Seemingly complex behavior does not necessarily require complex internal mechanisms. Seemingly simple behavior is not necessarily the results of simple internal mechanisms.

It is an open debate on where the description of behavior ends and where the description of the mechanism begins. Using an analytic approach we always end up with a description. If we employ the synthetic approach we not only have a description but a mechanism that actually underlies the observed behavior.

Synthetic tools

The tools of the synthetic methodology are computer simulations and robots. The field of behavior-based artificial intelligence or embodied cognitive science uses robots as modeling tools. However, in the field of artificial life, simulation is the tool of choice. Thus, for the present class we investigate mostly computer simulation.

Self-Organization

In AL the process of self-organization means the spontaneous formation of complex patterns or complex behavior emerging from the interaction of simple lower-level elements/organisms. It is an important concept and needs to be observed closely. The process of self-organization can either lead to the formation of reversible patterns (self-organization without structural changes) or to structural and therefore irreversible changes in the self-organizing system.

Emergence

The term emergence as used in AL means a property of a system as a whole not contained in any of its parts, i.e. the whole of a system being greater than the sum of its parts. Such emergent behavior results from the interaction of the elements of such system, which act following local, low-level rules. The emergent behavior of the system is often unexpected and cannot be deduced directly from the behavior of the lower-level elements.

Artificial Life and Artificial Intelligence

AL is concerned with the generation of *lifelike* behavior. The related field of Artificial Intelligence (AI) is concerned with generating *intelligent* behavior. In fact, AL and AI, at least new approaches in Artificial Intelligence have many topics in common. Mainly because AL and the new approaches in AI both work

bottom-up, combining many simple elements into more complicated ones, looking for emergence and principles of self-organization, using the synthetic methodology.

In summary, AL is based on the ideas of emergence and self-organization in distributed systems with many elements that interact with each other by means of local rules.

Bibliography

Bedau, M. A. and Packard, N. H. (1991). *Measurement of Evolutionary Activity, Teleology, and Life.* In C. G. Langton, C. Taylor, J. D. (eds.) Artificial Life II, Addison-Wesley.

Clancey, W. J. (1991). *The frame of reference problem in the design of intelligent machines.* In K. van Lehn (ed.). Architectures for intelligence. Hillsdale, N.J.: Erlbaum.

Harnad, S. (1995). *Levels of Functional Equivalence in Reverse Bioengineering.* In C. G. Langton (ed.): Artificial Life, An Overview, 293-301. MIT Press.

Huxley, J. S. (1942). *Evolution the modern synthesis.* Allen and Unwin, London.

Langton, C. G. (1989). *Artificial Life.* The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems. Addison-Wesley.

Pfeifer, R. and Scheier, C. (1999) *Understanding Intelligence*. The MIT Press, Cambridge, Massachusetts, London, England.

Schrödinger, E. (1944). *What is Life?* Cambridge University Press.

Tinbergen, N. (1963). *On aims and methods of ethology.* Z. Tierpsychologie, 20, 410-433.

Waldrop, M. M. (1992). *Complexity, The Emerging Science at the Edge of Order and Chaos.* Simon & Schuster.

Chapter 2: Pattern formation

God used beautiful mathematics in creating the world

Paul Dirac

In chapter two we will look at some examples illustrating basic principles of pattern formation in natural and artificial systems such as cellular automata, Lindenmayer systems (L-systems), and fractals. We will see that complex patterns can emerge from simple rules applicable to individual cells and local interactions of these cells. We will also see that the availability of many cells is a prerequisite as well as that all the rules valid for these cells are processed in parallel. The consequence of which will be that there is no need for central control.

2.1 Cellular automata

Cellular automata are examples of the large class of so-called complex systems. Complex Systems are dynamical systems that exhibit overall behavior that cannot directly be traced back to the underlying rules, that is, emergent or self-organized behavior. Complex systems typically consist of many similar, interacting, simple parts. 'Simple' means that the behavior of parts is easily understood, while the overall behavior of the system as a whole has no simple explanation. But often this emergent behavior has much simpler features than the detailed behavior of individual parts.

Introduction to Cellular Automata

Cellular automata (CA) are mathematical models in which space and time are discrete. Time proceeds in steps and space is represented as a lattice or array of cells (see figures 2.1 and 2.2). The size of this lattice is referred as the dimension of the CA. The cells have a set of properties (variables) that may change over time. The values of the variables of a specific cell at a given time are called the state of the cell and the state of all cells together form (as a vector or matrix for example) the global state or global configuration of the CA.

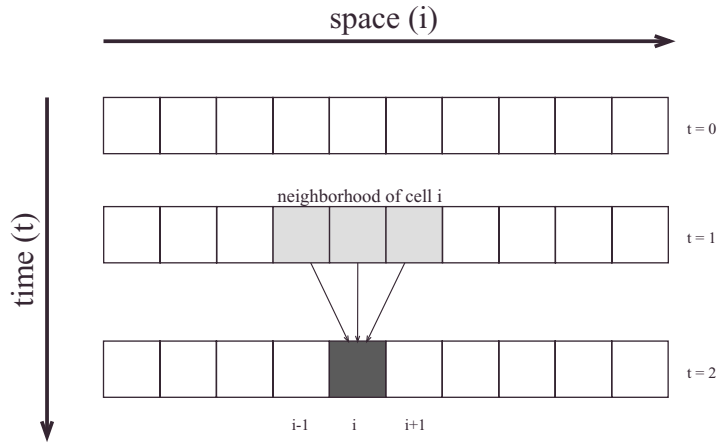


Figure 2.1: Space and time in a 1-dimensional CA.

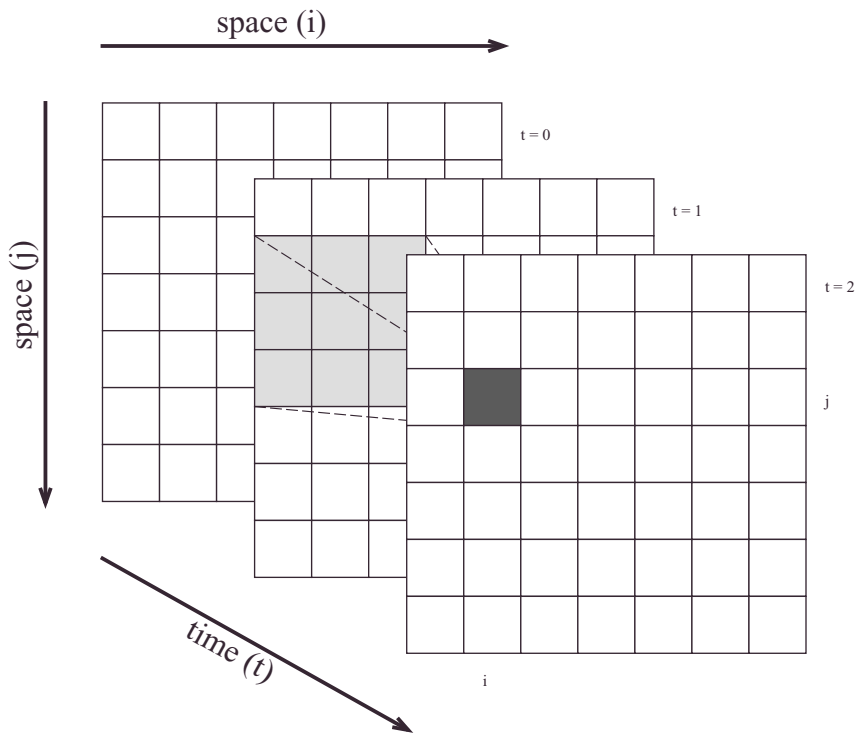


Figure 2.1: Space and time in a 2-dimensional CA.

We will consider only 1 and 2-dimensional CA. But the concept can be extended easily to any higher dimensional spaces. In table 2.1 the mathematical notation for 1 and 2-dimensional CA is summarized. Typically the state variables have discrete values. Also, a CA is discrete in time, discrete in space and therefore perfectly suited for simulation on a computer.

Table 2.1: Mathematical notation for 1- and 2-dimensional CA.

symbol	*Meaning*
t	Time
Δt	time step, typically 1
$a_i(t)$	state of cell at position i at time t (1 dim.)
$a_{ij}(t)$	state of cell at position (i,j) at time t (2 dim.)
$A(t)$	global state of the CA at time t

Local Rules

Each cell has a set of local rules. Given the state of the cell and the states of the cells in its neighborhood these rules determine the state of the cell in the next time step. These rules are local in two senses: First each cell has its own set of local rules and second the future state of the cell only depends on the neighbors of this cell. It is important to note that the states of all cells are updated simultaneously (synchronously) based on the (momentary) values of the variables in their neighborhood according to the local rules. If all cells have the same set of rules the CA is called homogeneous. We will consider only homogeneous CA.

The lattice of a CA can either be finite or infinite. Typically (especially if the CA is simulated on a computer) it is finite. Infinite lattices are mainly of mathematical interest. Infinite lattices have no borders, whereas on finite lattices one has to define what happens at the borders of the lattice, that is one has to define boundary conditions. The problem is that cells at the borders have only incomplete neighborhoods. There are three straightforward possibilities to solve this problem, either to assume that there are "invisible" cells next to the border-cells, which are in a given predefined state, (FIXED boundary), that the cells on the edge do not diffuse out of the system and only diffuse inwards (REFLECTIVE boundary), or to assume that the cells on the edge are neighbors of the cells on the opposite edge, (PERIODIC boundary), as depicted in figure 2.3.

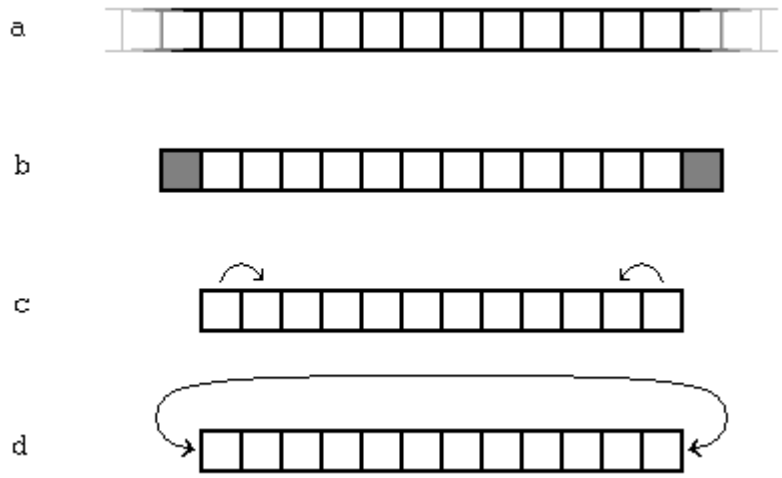


Figure 2.3: Four possibilities for boundary conditions in a 1-dimensional CA. a): infinite (unbounded) array of cells. b): finite array of cells with fixed boundaries. The end points have cells in their neighborhood with a fixed value. c): finite array of cells with reflective boundary. The leftmost cell can only diffuse to the right. d): finite array of cells closed to a circle, periodic boundary. The leftmost cell becomes a neighbor of the rightmost cell.

The initial values of all the state variables are referred to as the initial conditions. Starting from these initial conditions the CA evolves in time, changing the states of the cells according to the local rules. The evolution of the CA from its initial conditions is uniquely defined by the local rules, as long as they are deterministic (we will only consider deterministic rules). Thus, CAs are deterministic systems whose behavior results from local rules. Cells that are not neighbors do not directly affect each other. CAs have no memory in the sense that the actual state alone (and no other previous state) determines the next state. Because the rules and the states of the cells are local, any global pattern that might evolve is thus emergent.

Applications

CAs have been used for a wide variety of purposes. For example: for modeling nonlinear chemical systems (Greenberg et al., 1978) and the evolution of spiral galaxies (Gerola and Seiden, 1978; Schewe, 1981). In these two cases the lattice of cells in the CA corresponds directly to the physical space of the modeled system.

Any physical system satisfying (partial) differential equations may be approximated by a CA by discretisation of space, time, and state variables. Physical systems consisting of many discrete elements with local interaction are especially well suited to being modeled as CA. Also biological and social systems can often conveniently be modeled as CA.

CA and AL

CA are good examples of the paradigms of AL: complex systems made of similar (or identical) entities and local rules, parallel computation and thus local determination of behavior.

In the next few sections we will encounter some simple examples of 1-dimensional CA and explore the terms and concepts introduced. In section 2.2 we will see an example of a 2-dimensional CA.

1-dimensional Cellular Automata

Let's start with some very simple CA: a 1-dimensional CA with one variable at each cell taking only k possible values, say 0, 1, …, k-1. The value of the cell at position i at time t is denoted as $a_i(t+1)$. We will assume that the neighborhood consists always of the r nearest neighbors on each side[1] and the cell itself, thus the neighborhood consists of $2r+1$ cells. Each cell updates its state at every time step according to some set of rules Φ, and thus

$$a_i(t+1) = \Phi[a_{i-r}(t), a_{i-r+1}(t), \mathrm{K}\ , a_{i+r-1}(t), a_{i+r}(t)]$$

Let's look at a simple example. Assume that we have 256 cells in a row, and that each cell can take the values 0 or 1. Each cell updates its state depending on its own state and the state of its two immediate neighbors according to the following rule table:

Table 2.2: Example of a simple local rule (rule table).

$a_{i-1}(t)$	$a_i(t)$	$a_{i+1}(t)$	$a_i(t+1)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

This rule can be re written in a much more compact form using predicate calculus

$$a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t)$$

where $\oplus$ denotes addition modulo 2 (XOR). The graphical representation shown in figure 2.4 is much more intuitive. We will assume that the row of cells of the CA is closed to a circle (see figure 2.3), thus the leftmost cell is the neighbor of the rightmost cell.

[1] r: radius of neighbors

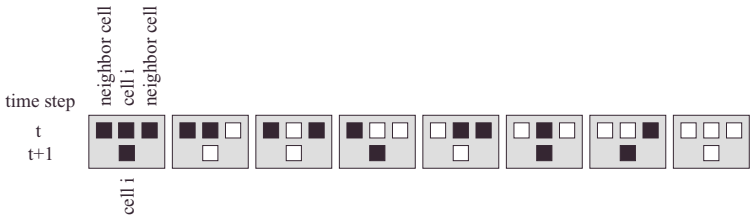


Figure 2.4: Graphical representation of a CA rule. The top row (in the gray boxes) corresponds to the configuration of a cell and its immediate neighbors. In our example there are eight possible configurations of cells (3 cells, 2 states (on, off) each). For each of these eight configurations the bottom row specifies the state of the cell in the next time step.

Figure 2.5 shows the pattern that is generated by the rule above when we start with one black cell in the middle of the CA array. (Remember that the rows correspond to the CA cells at subsequent time steps). Note the self-similarity of the patterns[2]. Although this figure is not a fractal[3] in the strict sense (because it has no infinitely fine structures) it is indeed very fractal-like. You can imagine that in an infinite CA array this pattern would grow forever, thereby generating bigger and bigger triangles, and repeat the patterns it has generated before. A very different picture is observed when we start the same CA (with the same rules) from a random initial configuration (figure 2.6). Note that the regular pattern observed before is gone. Still, the pattern is not a random one. Triangles and other structures appear over and over again, although at irregular times and at unforeseen places.

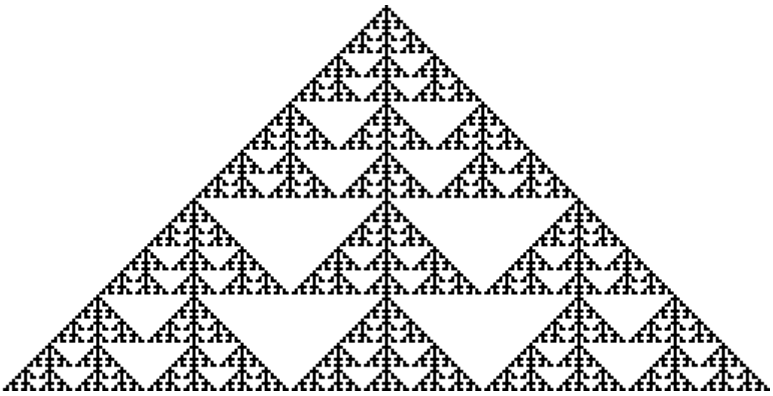


Figure 2.5: Pattern generated by 1-dimensional CA. The pattern is generated by the 1-dimensional CA rule introduced in the text. The top row corresponds to the initial configuration (one black cell in the middle) and the bottom one to the state of the CA after 128 time steps. Note the self-similarity of the patterns.

[2] Self-similarity at multiple levels is a key feature of fractality (see section 2.4).
[3] see section 2.4

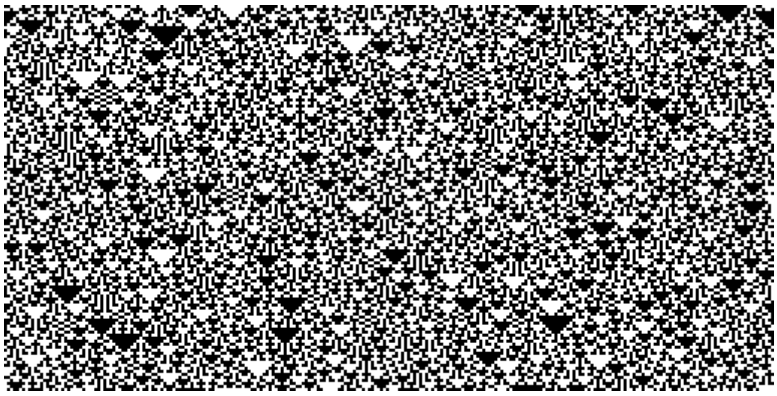


Figure 2.6: Pattern generated by a 1-dimensional CA. The same CA as in figure 2.5 is used, but in contrast to figure 2.5, the initial configuration (top row) is a random one. At first sight the patterns seems random too but at closer inspection many small structures that appear over and over again can be discerned.

We conclude from these two examples that even such simple deterministic systems as 1-dimensional CA can produce astonishingly complex patterns. These patterns are very regular if the initial configuration is regular too (figure 2.5). If the initial configuration is random the generated pattern is much less regular (figure 2.6). In both cases the patterns are complex but they reveal simple higher-level structures, the triangles. Note that by simply looking at the rules (figure 2.4) it is not at all obvious that such triangles will emerge. It is very common to find emergent structures in CA.

Number of Possible CA Rules

There are many different possibilities for CA rules. In the previous we had two states per cell, and three neighbors. Therefore there are $2^3 = 8$ entries in the rule table of a CA (the top row of figure 2.4) and thus $2^8 = 256$ possible rule tables. So there are 256 different possible cellular automata of this type. This number grows exponentially if we increase the number of states k per cell and the range r of the neighborhood (or the number of neighbors $2r + 1$). For k states per cell and $2r + 1$ neighbors we have

$$k^{2r+1}$$

entries in the rule table and

$$k^{k^{2r+1}}$$

possibilities for rule tables or CA. For $k = 10$ and $r = 5$ we have 10^{11} entries in the rule table and $10^{100'000'000'000}$ different possible CA. To put this number into perspective, there are only about 10^{80} molecules in the universe. Thus we will never be able to examine all or even a significant fraction of all possible CA.

The Four Classes of Cellular Automata*

In this section we will consider again 1-dimensional CA with 256 cells. Each cell can take the values 0 or 1 ($k = 2$). But this time we will use neighborhoods of a varying number of cells ($r = 1$; $r = 2$, that is the cell itself and the two nearest neighbors on each side; $r = 3$). Although these types of CA are, again, very simple they exhibit a wide variety of qualitatively different phenomena. In figure 2.7 typical examples of the

evolution of such cellular automata from random initial conditions are depicted. Structures of different quality and complexity are formed. Wolfram (1984a) divided the CA rules into four classes; according to what quality of structures they give rise. These are:

Class I: Tends to spatially homogeneous state (all cells are in the same state). Patterns disappear with time.

Class II: Yields a sequence of simple stable or periodic structures (endless cycle of same states). Pattern evolves to a fixed finite size.

Class III: Exhibits chaotic aperiodic behavior. Pattern grows indefinitely at a fixed rate.

Class IV: Yields complicated localized structures, some propagating. Pattern grows and contracts with time.

The classes II and III correspond to the different types of attractors (see appendix A):

Class II: Point attractor or periodic attractor.

Class III: Strange or chaotic attractor.

The four classes can also be distinguished by the effects of small changes in the initial conditions (Wolfram, 1984a):

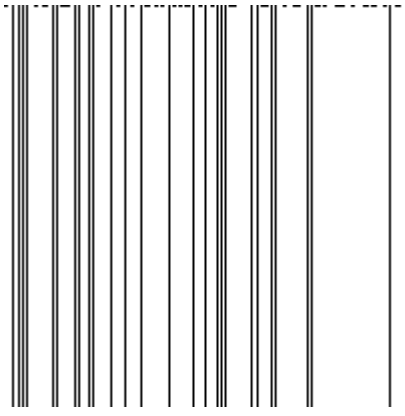
Class I: No change in final state.

Class II: Changes only in a region of finite size.

Class III: Changes over a region of ever-increasing size.

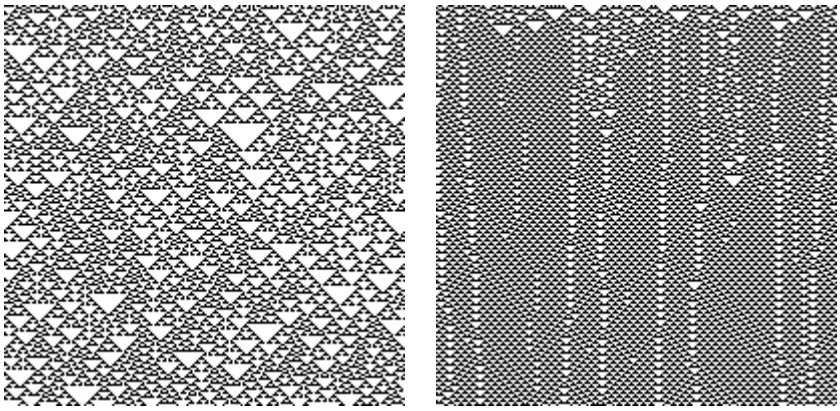
Class IV: Irregular changes.

Class I: empty (rule 128[4])



Class II: stable or periodic (rule 4[5])

[4] Rule 128: ‘111’ goes to ‘1’, else ‘0’.
[5] Rule 4: ‘010’ goes to ‘1’, else ‘0’.



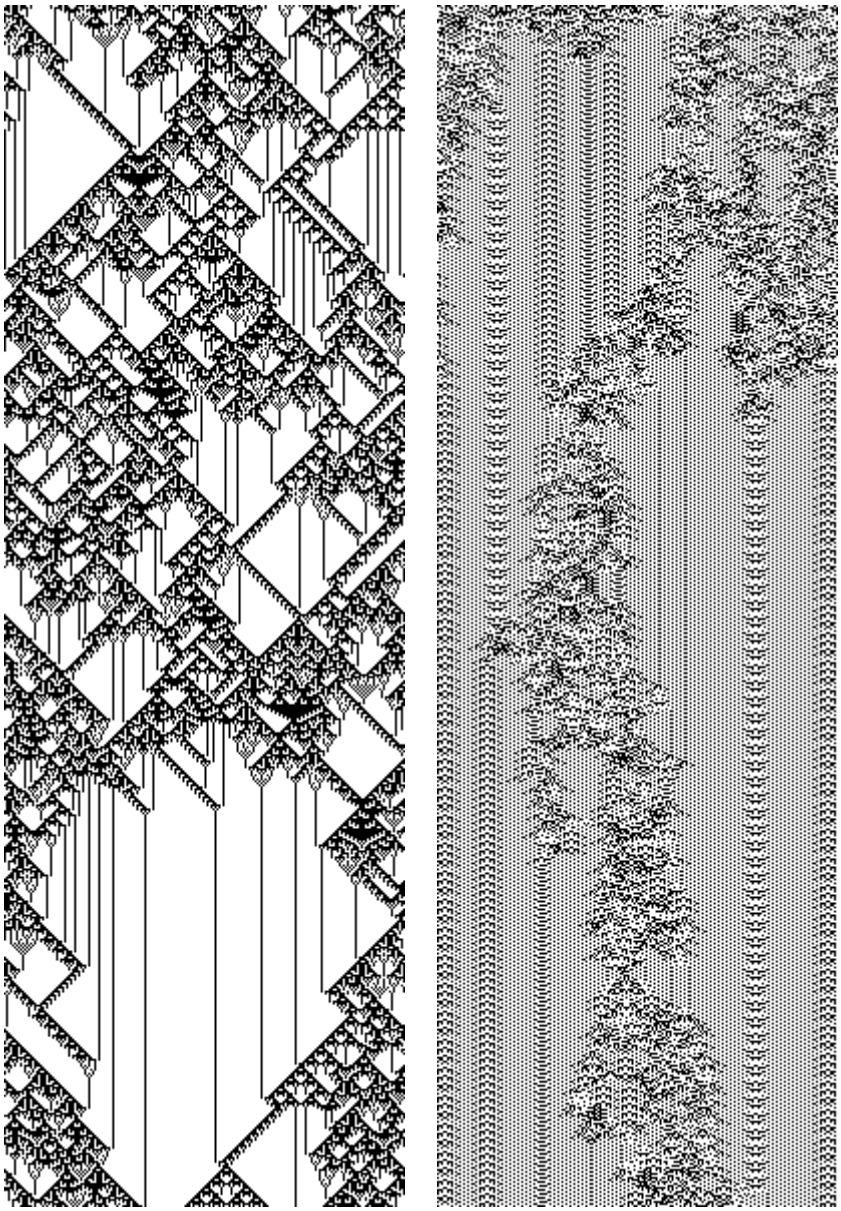
Class III: chaotic (rule 22[6]) Class IV: complex (rule 54[7])

Figure 2.7: Typical examples of CA (k = 2, r = 1) starting from a random initial configuration. Depicted here are examples of the four classes of CA as introduced by Wolfram (1984a).

Figures 2.8 and 2.9 show the behavior of class IV CA. Their behavior is difficult to describe. It is not regular, not periodic, but also not random. It contains a bit of each. Class IV CAs remain at the boundary between periodicity and chaos. Moreover, the behavior is not predictable without explicit calculation. That is very little information on the behavior of a class IV CA can be deduced directly from properties of its rules.

> It seems likely, in fact, that the consequences of infinite evolution in many dynamical systems may not be described in finite mathematical terms, so that many questions concerning their limiting behavior cannot be formally decided. Many features of the behavior of such systems may be determined effectively only by explicit simulation: no general predictions are possible. (Wolfram, 1984a, p. 23)

[6] Rule 22: ‘001’, ‘100’, and ‘010’ go to ‘1’, else ‘0’.
[7] Rule 54: ‘001’, ‘100’, ‘010’, and ‘101’ go to ‘1’, else ‘0’.



$k = 2$, $r = 2$ $k = 2$, $r = 3$

Figure 2.8: Two examples of class IV CA.

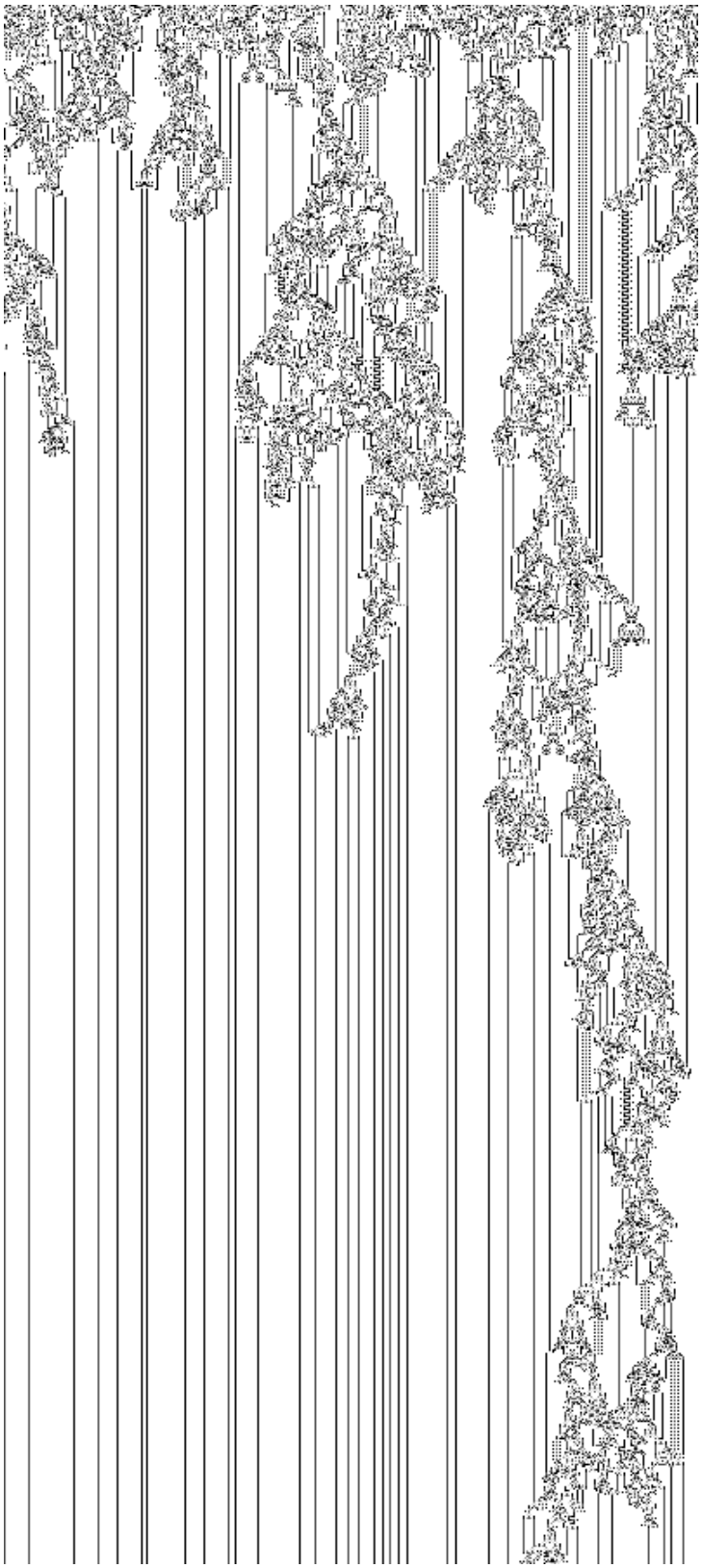


Figure 2.9: Another example of a class IV CA (k=5, r=2).

2.2 Game of Life

In the late 1960s John Conway, motivated by the work of von Neumann, used simple 2-dimensional CAs which he called the “game of life”. Each cell has two possible states 0 or 1 (or “dead” and “alive”, thus the name of the CA), and a very simple set of rules (Flake, 1998):

Loneliness: If a live cell has less than two neighbors, then it dies.

Overcrowding: If a live cell has more that three neighbors, then it dies.

Reproduction: If a dead cell has three live neighbors, then it comes to life.

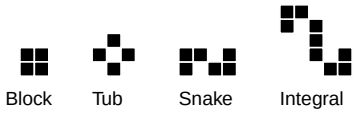
Stasis: Otherwise, a cell stays as it is.

In 1970 Martin Gardner described the Game of Life and Conway’s work in Scientific American (Gardner, 1970). This article inspired many people around the world to experiment with Conway's CA. Many interesting configurations were found. We will encounter some of them in the following discussion.

Patterns in the Game of Life are usually characterized by their behavior. There are several categories (of increasing complexity)[8]:

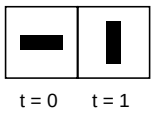
Type I (still-lives): Patterns that do not change; that are static.

Examples:



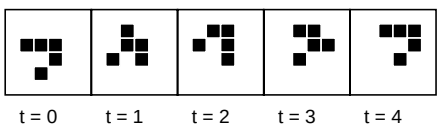
Type II (oscillators): Patterns that repeat themselves after a fixed sequence of states and return to their original state; periodic patterns.

The ‘blinker’ is an example of a 2-periodic oscillator:



Type III (spaceships): Patterns that repeat themselves after a fixed sequence of states and return to their original state, but translated in space, patterns that move at a constant velocity.

The ‘glider’ is one simple example:

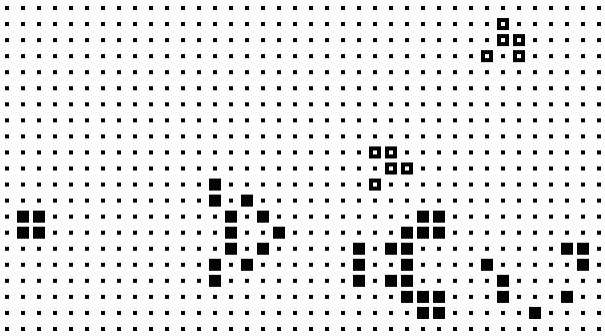


Type IV: Patterns that constantly increase in population size (living cells).

[8] For exhaustive collection of life patterns and animations see for example home.interserv.com/~mniemiec/lifeterm.htm.

Type IVa (guns): Oscillators that emit spaceships in each cycle.

Example: A glider gun (black squares) that emits gliders (empty squares):



Type IVb (puffers): Spaceships that leave behind still-life, oscillators, and/or spaceships.

Type IVc (breeders): Patterns that increase their population size quadratically (or even faster). For example, a breeder may be a spaceship that emits glider guns.

Type V (unstable): Patterns that evolve through a sequence of states, which never return to the original state. Small patterns that last a long time before stabilizing are called “Methuselahs”.

Again, the message is that despite the simplicity of the rules, amazingly complex and sophisticated structures can emerge in the Game of Life.

Universal Computation

Universal Computation means that there is the capability of computing anything that can be computed. A universal computer, i.e. a computer capable of universal computation, can do so. The best-known example of such a universal computer is a Turing Machine, an imaginary machine proposed in 1936 by Alan Turing, an English mathematician. A Turing Machine has a read/write head mounted to a tape of infinite length, i.e. consisting of an infinite number of cells. The action performed by the head (read, write, move forward, move backward or no action/movement) depends on the current state of the head and of the cell underneath. Due to the infinite length of the tape and the lack of any limitations regarding the number of possible states, the Turing machine can solve every computable problem and it is able of universal computation.

Looking again at the Game of Life from a computational point of view we can say that type I objects, i.e. static objects, can be seen as a kind of memory needed in every computer; type II objects, i.e. periodic patterns, can fulfill the task of counting or synchronizing parallel processes and type III objects which repeat themselves regularly but move in space are required for information flow in a computer, thus the Game of Life includes the basic elements necessary for a computer. Through repeated collisions of moving objects with static objects the latter get altered and increase in size, i.e. new objects are created. Such process of recursively assembling pieces to make larger and more complicated objects can be carried to the extreme of building a self-reproducing machine (Flake, 1999). Based on the above and since operations in computers are usually implemented in terms of logical primitives (AND, OR, NOT) we can say that it is possible to build a general-purpose computer in the Game of Life and that it can emulate any Turing

machine. As a consequence the Game of Life is unpredictable (in the same sense as the Class IV CA of Wolfram, see above)

> There are important limitations on predictions, which may be made for the behavior of systems capable of universal computation. The behavior of such systems may in general be determined in detail essentially only by explicit simulation. […] No finite algorithm or procedure may be devised capable of predicting detailed behavior in a computationally universal system. Hence, for example, no general finite algorithm can predict whether a particular initial configuration in a computationally universal cellular automaton will evolve to the null configuration after a finite time, or will generate persistent structures, so that sites with nonzero values will exist at arbitrarily large times. (This is analogous to the insolubility of the halting problem for universal Turing machines [see for example Beckmann, 1980].) (Wolfram, 1984b, p. 31)

Another way in which sophisticated structures can emerge from sets of simple rules are the so-called Lindenmeyer Systems.

2.3 Lindenmeyer systems

In 1968 the biologist Aristid Lindenmeyer invented a mathematical formalism for modeling the growth of plants. This formalism, known as Lindenmeyer system or L-system, is essentially a traditional production system. Productions, or rewriting rules, are rules, which state how new symbols or cells grow from old symbols, or cells. A production system as a whole states how at each time step its production rules are applied to symbols in such a way that as many old symbols as possible are simultaneously substituted by new symbols.

Consider the following L-system as a simple example:

Axiom: *B* *(starting cell or starting seed of the L-system)*

Rule 1: $B \rightarrow F[-B] + B$

Rule 2: $F \rightarrow FF$

If we like, we can interpret the effect of the individual rules in this rule system as follows.

Axiom: Initially, we start with a lone *B*-cell (see figure 2.10).

Rule 1: Each *B* cell divides, producing an *F* cell and two *B* cells arranged as depicted in figure 2.10. The brackets and the "+" and "-" signs indicate the arrangement of the cells ("+" rotate right, "-" rotate left).

Rule 2: Each *F* cell divides, producing two *F* cells arranged as shown in figure 2.10.

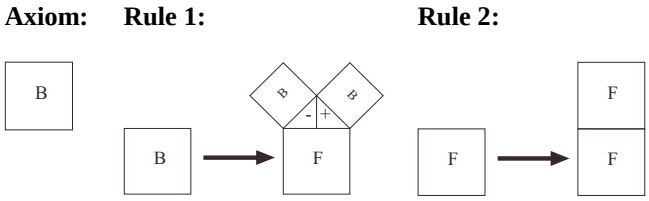


Figure 2.10: Effect of individual rules.

Note that the rules are applied in parallel, that is, all possible deductions (i.e. all possible applications of the rules) are performed simultaneously. Now let's look at the strings produced by the production system described above. The axiom tells us to start with a single *B* cell. Therefore the initial string is simply

B

Now we apply the rules to this string and obtain (only Rule 1 matches)

F[-B] + B

Note that the rules of the L-system are used as substitution rules. In the next step both rules match and are applied resulting in the following string:

FF[-F[-B] + B] + F[-B] + B

As can be seen, the length of the string grows dramatically and gets increasingly confusing:

FFFF[-FF[-F[-B] + B] + F[-B] + B] + FF[-F[-B] + B] + F[-B] + B

Let us perform one more step:

FFFFFFFF[-FFFF[-FF[-F[-B] + B] + F[-B] + B] + FF[-F[-B] + B] + F[-B] + B] + FFFF[-FF[-F[-B] + B] + F[-B] + B] + FF[-F[-B] + B] + F[-B] + B

Much more intuitive is the graphical representation of the same process as depicted in figure 2.11.

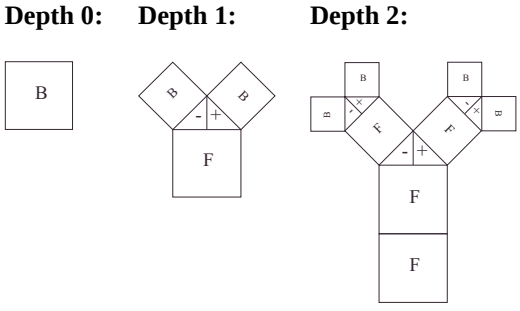


Figure 2.11: Effect of joint action of rule system. What emerges is a kind of tree structure.

Turtle Graphics

L-systems by themselves do not generate any images; they merely produce large sequences of symbols. In order to obtain a picture, these strings have to be interpreted in some way. In figures 2.10 and 2.11 we have already seen a possibility. More generally, these L-systems can be interpreted by turtle graphics.

The concept of *turtle graphics* originated with Seymour Papert. Intended originally as a simple computer language for children to draw pictures (LOGO), a modified turtle graphics language is well suited for drawing L-systems. Plotting is performed by a (virtual) turtle. The turtle sits at some position looking in some direction of the computer screen and can move forward, either with or without drawing a line, and can turn left or right. A brief summary of commands used for drawing L-systems is given in table 2.3. Note that without the brackets the drawing of branching structures is impossible.

Table 2.3: Turtle graphics commands

command	turtle action
F	draw forward (for a fixed length)
|	draw forward (for a length computed from the execution depth)
G	go forward (for a fixed length)
+	turn right (by a fixed angle)
-	turn left (by a fixed angle)
[	save the turtle's current position and angle for later use
]	restore the turtle's position and orientation saved by the most recently applied *[* command

Figure 2.12 shows the first five stages of the drawing process of the following L-system:

Axiom: *F*

Rule: *F=|[-F][+F]*

Angle: *20*.

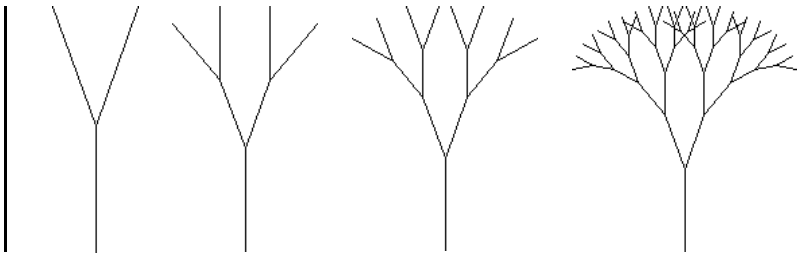


Figure 2.12: The first five L-system stages

The first drawing has an execution depth 0 and the drawing corresponds to the string

F

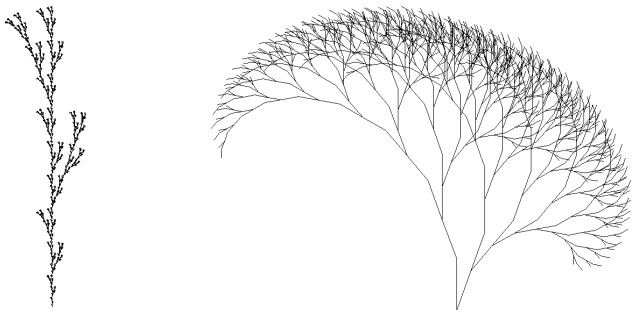
The execution depth denotes the number of times the rule is applied. The above string corresponding to execution depth 0 is therefore the axiom. In the second drawing, the rule is applied once, thus leads to the string

| [-F] [+F]

and the third drawing has execution depth 2 resulting in the string

| [-| [-F] [+F]] [+| [-F] [+F]]

In figure 2.13 two examples of L-systems are given. A sample program to generate these images can be downloaded from mitpress.mit.edu/books/FLAOH/cbnhtml/index.html.



Axiom: *F*
Rule: *F = F [-F] F [+F] F*
Angle: 20
Depth: 7

Axiom: *F*
Rule: *F = | [+F] | [-F] +F*
Angle: 20
Depth: 9

Figure 2.13: A few examples of L-systems (from Flake, 1998).

Development Models

The interpretation of an L-system can be extended to three dimensions by adding a third dimension to the orientation of the turtle. In order to simulate the development of plants additional information has to be included into the production rules. Also, an additional assumption is made that plants control the important aspects of their own growth. Such information can include a delay mechanism or the influence of environmental factors but also a stochastic element, so that not all the plants look the same. Some examples of such more complex models of development are depicted in figures 2.14 and 2.15 (from Prusinkiewicz, 1990). Additional information on these can be found on the really beautiful web site www.cpsc.ucalgary.ca/projects/bmv/vmm-deluxe/TitlePage.html. We will discuss additional models of how shapes can grow when we discuss artificial evolution in chapter 6.

Figure 2.14: A sophisticated plan (a mint) grown with L-systems

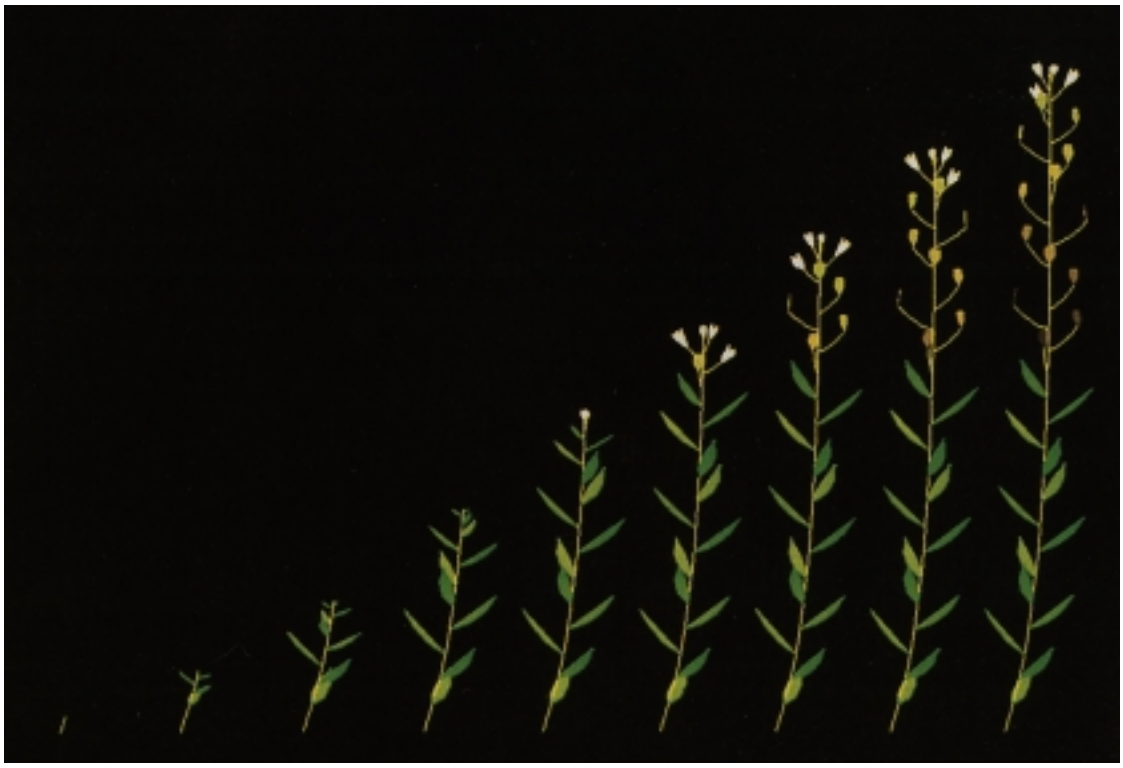


Figure 2.15: Simulated development of Capsella bursapastoris.

2.4 Fractals

The simple L-systems we met in the previous section are instances of the more general structures known as *fractals*. The term "fractal"[9] was first used by Benoit Mandelbrot (e.g., Mandelbrot, 1983). Fractals are geometric figures that have one striking quality: They are self-similar on multiple scales, which means that a small portion of a fractal often looks similar to the whole object (in theory a fractal is perfectly regular and has infinitely fine structures). A description of a fractal-like object could be something like this: "It has a miniature version of itself embedded inside it, but the smaller version is slightly rotated." For example, one branch of a particular L-system plant looks exactly like the whole plant, only smaller (e.g. figure 2.18). To be precise, this perfect self-similarity of L-systems holds only if the L-system is calculated to infinite depth, or to infinitely fine details. In this case the somewhat paradoxical statement holds that an arbitrary branch of the L-system plant is exactly the same as the whole plant, only rotated and scaled. In other words, a fractal contains itself. Not only that, a fractal consists of infinitely many copies of itself.

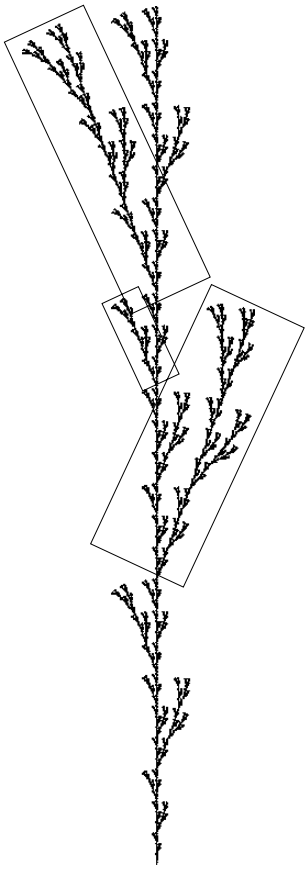


Figure 2.18: The fractal structure of L-system turtle graphics. Each branch in the boxes contains a rotated and re-scaled copy of the whole figure.

[9] The name fractal has been given based on the fractal dimension of these structures. A fractal can have a non-integer dimension meaning for example that it is "more than a line but less than a plane".

Fractals in Nature

Fractals often appear in nature. Not only plants like trees or ferns (see figure 2.19) have a fractal structure, but also snowflakes, and blood vessels (see figure 2.20).

Figure 2.19: A fractal fern (from www.mhri.edu.au/~pdb/fractals/fern/)

Fractals are nature's answer to hard "optimization" problems, i.e. how to find the optimal solution if there are conflicting goals. In case of the blood vessel system the hard task is to supply every part of the body with blood using as few resources as possible and in the same time minimizing the amount of time used for a single round trip. (Without this last condition one thin long blood vessel visiting every part of the body would do the job.) Because blood vessel systems are the result of millions of years of evolution one may think that they are not just any solution to the problem but a good one, one that is close to an optimal one.

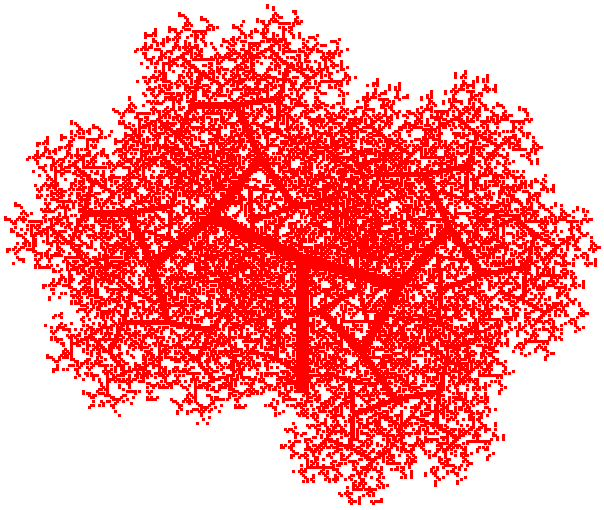


Figure 2.20: A fractal model of the blood vessel system (from www.cs.ioc.ee/ioc/res98/fractal.html).

Of course, fractals in nature are not perfect mathematical fractals; they have no infinitely fine structures and are not perfectly regular. Blood vessels, for example, do not become indefinitely small; there is some minimal diameter. Interestingly, the smallest blood vessels, the capillaries, are always of about the same

size. For example the capillaries of an elephant have the same diameter as those of a mouse. The difference is that the elephant's blood vessel system has a few more branching levels than that of the mouse.

Generating fractals

Above we introduced the concept of self-similarity, i.e. the fact that fractals contain miniature versions of themselves. The trick in generating fractals is to come up with a way to describe where and how the miniature version of the whole should be placed. In the previous section we have used L-systems and turtle graphics. In general, there are four types of transformations that one could imagine as being useful: translation (move to different place), scaling (alter size), reflection, and rotation. Algorithms for generating fractals are always recursive and based on self-similarity, using combinations of these basic four transformations. An example is the Multiple Reduction Copy Machine Algorithm (MRCM). Figure 2.21 shows a schematic representation of the MRCM algorithm[10].

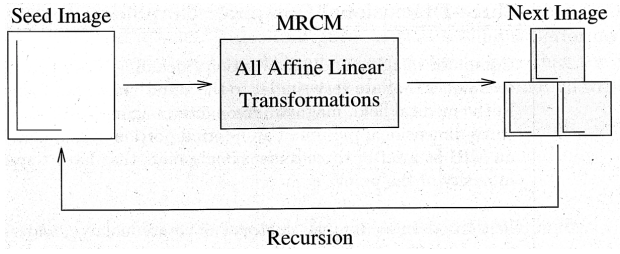


Figure 2.21: A schematic of the MRCM algorithm. The input image is simultaneously transformed by translation and scaling.

There is a vast literature on fractals. It would be beyond the scope of this class to provide extensive coverage. The interested reader is referred to Flake (1998), Chapter 7 (Affine Transformation Fractals), Barnsley (1988), Mandelbrot (1983), and Peitgen et al. (1992).

2.5 Sea shells

Another fascinating case of pattern formation that can be conveniently described by cellular automata is the evolution of the colorful patterns of seashells. We all know the pigment patterns of tropical seashells and are impressed by their beauty and diversity. The fascination comes from their mixture of regularity and irregularity (see figure 2.23). No two shells are identical but we can immediately recognize similarities. The patterns on the shell resemble the patterns we met in the sections on 1-dimensional CA. And this coincidence has a deeper reason.

The patterns on seashells consist of calcified material. A mollusk can enlarge its shell only at the shell margin. Therefore, in most cases the calcified material, and thus the pigmentation patterns, is added at this margin. In this way the shell preserves a time record of the pigmentation process that took place at its margin. This process is much like the 2-dimensional pictures of 1-dimensional CA that are a time record of

[10] The MRCM algorithm's name is based on the fact that it could at least partly be simulated with a real copy machine (make simultaneously several copies of an object and alter place and size, such process to be repeated several times).

the CA dynamics. In this sense, it is straightforward to simulate pattern growth on a seashell with a 1-dimensional CA. Let us look at this idea in some detail.

As Meinhardt argues in his book "The Algorithmic Beauty of Sea Shells" (Meinhardt, 1995) the process of pattern formation in seashells can be conceived in terms of an activator-inhibitor dynamics (figure 2.22) whereby the activator causes and the inhibitor suppresses pigmentation. These dynamics are often called reaction-diffusion dynamics. Pattern formation is the result of local self-enhancement (also called autocatalysis) and long-range inhibition.

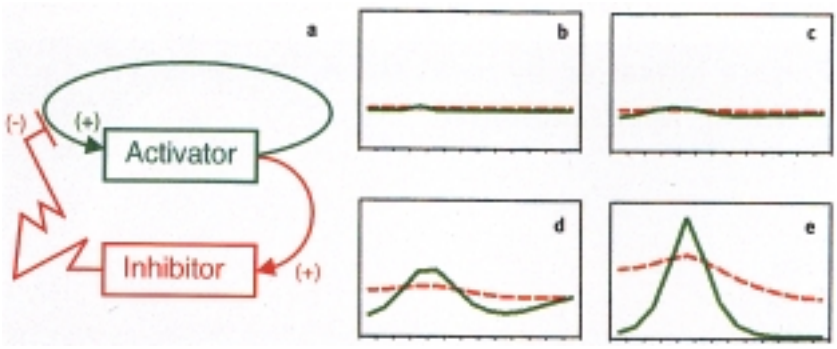


Figure 2.22: Reaction scheme for pattern formation by autocatalysis and long-range inhibition. An activator catalyzes its own production and that of its antagonist (the inhibitor). The diffusion constant of the inhibitor must be much higher than that of the activator. A homogenous distribution of both substances is unstable (b) (the x-axis represents position and the y-axis the concentration). A minute local increase of the activator (——) grows further (c, d) until a steady state is reached in which self-activation and inhibition (- - - -) are balanced (from Meinhardt, 1995).

Activator-inhibitor dynamics can be described either by a set of partial differential equations, or by cellular automata.

Meinhardt (1995) introduces the following differential equations to describe the dynamics of the activator-inhibitor system that relate the concentration change per time unit of both substances *a* and *b* as a function of the present concentration.

$$\frac{\partial a}{\partial t} = s\left(\frac{a^2}{b} + b_a\right) - r_a a + D_a \frac{\partial^2 a}{\partial x^2}$$

$$\frac{\partial b}{\partial t} = sa^2 - r_b + D_b \frac{\partial^2 b}{\partial x^2} + b_b$$

where a(x,t) is the concentration of an auto-catalytic activator at position *x* at time *t*, *b(x,t)* is the concentration of its antagonist, D_a and D_b are the diffusion coefficients and r_a and r_b are the decay rates of *a* and *b*.

Let us briefly outline the main intuitions why the interaction as stated in the above equations can lead to stable patterns. Let's assume all constants, and even the inhibitor concentration, are equal to 1, and disregard diffusion. This leads to the following simplified equations:

$$\frac{\partial a}{\partial t} = a^2 - a$$

Here the activator has a steady state but only at $a = 1$ otherwise the state will be unstable. Simplifying the equation for the inhibitor b leads to

$$\frac{\partial b}{\partial t} = a^2 - b$$

Now the steady state is at $b = a^2$.

Now let us include the action of the inhibitor in the equation for the activator. Under the assumption that the inhibitor reaches the equilibrium rapidly after a change in activator concentration, this can be expressed as function of the activator concentration alone

$$\frac{\partial a}{\partial t} = \frac{a^2}{b} - a \approx \frac{a^2}{a^2} - a = 1 - a$$

The inclusion of the inhibitor leads to a steady state at $a = 1$ which remains stable since for $a > 1$, $(1-a)$ is negative and the concentration returns to $a = 1$ (Meinhardt, 1999).

As seen above the action of the inhibitor leads to stabilization of the autocatalysis and to stable patterns. On shells, stable patterns lead to permanent pigment production in some positions caused by a higher concentration of activator a and its suppression in between (higher concentration of inhibitor b). This leads, for example, to an elementary pattern of stripes parallel to the direction of growth.

The above partial differential equations, which represent the continuous change over time, can be approximated by a system of difference equations representing change in discrete time steps. Accordingly the discrete i will take the role of x (the position). The differentials $\frac{\partial a}{\partial t}, \frac{\partial a}{\partial x}, \frac{\partial^2 a}{\partial x^2}$ can be approximated by differences:

$$\frac{\partial a}{\partial t}(x,t) \approx a_i(t+1) - a_i(t),$$

$$\frac{\partial a}{\partial x}(x,t) \approx a_{i+1}(t) - a_i(t), \text{ and}$$

$$\frac{\partial^2 a}{\partial x^2}(x,t) \approx \left[a_{i+1}(t) - a_i(t)\right] - \left[a_i(t) - a_{i-1}(t)\right].$$

Inserted into the system of differential equations as set out above the concentration of the activator would be

$$a_i(t+1) = a_i(t) + s\left(\frac{a_i^2(t)}{b_i(t)} + b_a\right) - r_a a_i(t) + D_a\left(a_{i-1}(t) + a_{i+1}(t) - 2a_i(t)\right)$$

Time is now discrete with a time step of Δt=1. If we interpret *i* as the number of a cell (in a row) the above equation is in fact a local rule for a 1-dimensional CA. Analogously the equation for the inhibitor *b(x,t)* can be reformulated and we obtain a local rule for $b_i(t)$.

$$b_i(t+1) = b_i(t) + sa_i^2(t) - r_b b_i(t) + D_b(b_{i-1}(t) + b_{i+1}(t) - 2a_i(t)) + b_b$$

Therefore our resulting CA has *two* variables $a_i(t)$ and $b_i(t)$ for each cell *i*. The difference to the CA discussed earlier is that the state variables here can take arbitrary values and not just discrete ones.

In figure 2.23 two examples of seashells and their simulated counterparts are shown (from Meinhardt, 1995). The patterns were calculated as discussed above and the mapped onto a 3-dimensional model of a seashell. The results are striking.

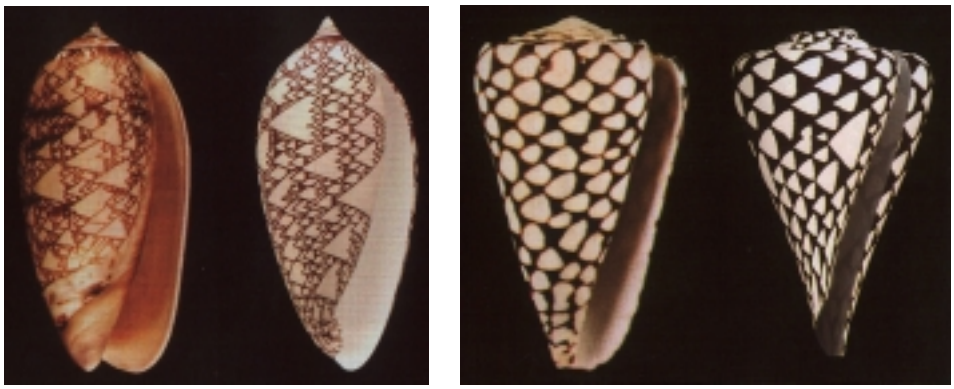


Figure 2.23: Two examples of seashells and the simulated patterns using — in essence — the dynamics described in this section (taken from Meinhardt, 1995, p. 179, 180).

2.6 Sand piles

While studying the fundamental question why nature is so complex and not as simple as the laws of physics would imply, the concept of self-organized criticality (SOC), a mathematical theory describing how systems reach dynamical behavior, has been discovered (Bak, 1997). SOC explains some complex patterns that we find everywhere in nature. SOC states that nature is perpetually out of balance, but organized in a poised state – the critical state[11] – where anything can happen within well-defined statistical laws.

A good and easily understandable example of SOC is the sandpile model. One can imagine a flat table onto which grains of sand are added randomly one at a time. In the beginning the grains will mostly stay where they land. With more sand added grains start to pile up and sand slides and avalanches occur. First such avalanches only have a local effect in one particular region of the table but with more sand added the piles cannot get any higher since the slope is too steep for additional grains of sand. Consequently the avalanches become stronger and also affect the piles in the other regions of the table or may even cause sand to leave the table (see figure below).

[11] Critical in the sense that it is neither stable nor unstable, but near phase transition.

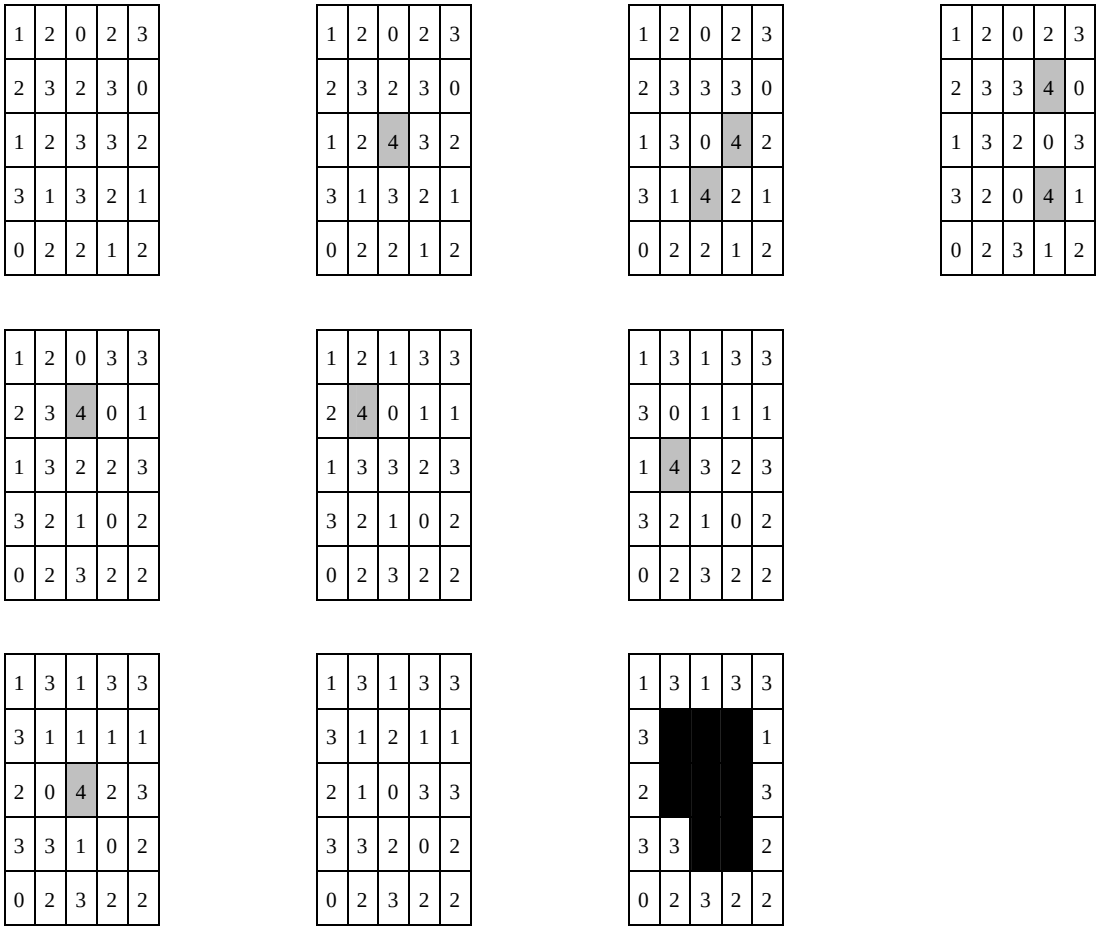


Figure2.24: Illustrating of toppling avalanche in a small sandpile. A grain falling at the site with height 3 at the center of the grid leads to an avalanche composed of nine toppling events, with a duration of seven update steps. The avalanche has a size s=9. The black squares indicate the eight sites that toppled. One site toppled twice. (Bak, 1997, p.53)

In the end new grains added to the pile will result in average in the same number of grains rolling down the pile and falling off the table. In order to achieve such balance between sand added to, and sand leaving the table communication within the system is required. Such state is the self-organized critical (SOC) state.

The number of avalanches of size *s* can be expressed by the simple power law

$$N(s) = s^{-\tau}$$

(where the exponent τ defines the slope of the curve) and results in a quasi straight line if plotted on a double-logarithmic paper.

The power law states the following: small avalanches appear more often than big ones.

The addition of grains of sand has transformed the system from a state in which the individual grains follow their own local dynamics to a critical state where the emergent dynamics are global (Bak, 1997). The individual elements obeying their own simple rules have through interaction lead to a unique, delicately

balanced, poised, global situation in which the motion of any element might affect any other element in the system.

Accordingly the sandpile model shows how an open system has naturally organized itself into a critical scale-free state without any external organizing force, thus a simple model for complexity in nature has been developed.

2.7 Conclusion

In this chapter we have looked at a number of examples illustrating basic principles of pattern formation in natural and artificial systems. The essence is that sophisticated patterns can emerge on the basis of simple rules that are based on local interactions. There is no need for a global blueprint. Cellular automata, Lindenmeyer systems (L-systems), fractals and SOC are convenient formalisms to model pattern formation processes.

Another central factor in pattern formation is — almost trivially — the availability of many cells, and that all the cells are processed in parallel: there must be no central control.

Bibliography

Bak, P. (1997). *How Nature Works*. Oxford University Press

Barnsley, M. (1988). *Fractals Everywhere.* Academic Press

Beckmann, F. S. (1980). *Mathematical Foundations of Programming.* Addison-Wesley

Berlekamp, E. R., J. H. Conway, R. K. Guy (1982). *What is Life?* Chapter 25 in Winning Ways (Volume 2), Academic Press

Flake, G. W. (1998). *The Computational Beauty of Nature.* A Bradford Book

Gardner, M. (1970). *Mathematical Games. The fantastic combinations of John Conway's new solitaire game "life"*. In Scientific American, vol. 223, no. 4, p. 120-123

Gerola, H. and P. Seiden (1978). *Stochastic star formation and spiral structure of galaxies*. Astrophys. J., 223, p. 129

Greenberg, J. M., B. D. Hassard, and S. P. Hastings (1978). *Pattern formation and periodic structures in systems modeled by reaction-diffusion equations*. Bull. Am. Math. Soc., 84, p. 1296

Mandelbrot, B. (1983). *The Fractal Geometry of Nature.* Freeman

Meinhardt, H. (1995). *The Algorithmic Beauty of Sea Shells.* Springer

Peitgen, H.-O., H. Jürgens, D. Saupe (1992). *Chaos and Fractals*, Springer

Prusinkiewicz, P. (1990). The Algorithmic Beauty of Plants. Springer

Schewe, P. F (ed.) (1981). *Galaxies, the Game of Life, and Percolation.* Physics News, Amer. Inst. Phys. Pub. R-302, p. 61

Wolfram, S. (1984a). *Computation Theory of Cellular Automata*. Commun. Math. Phys. 96, p. 15-57

Wolfram, S. (1984b). *Universality and Complexity in Cellular Automata.* Physica D, Vol. 10, p. 1-35

Chapter 3: Distributed intelligence

In the last chapter we concluded that pattern formation in natural systems occurs as a consequence of simple local rules. We had a look at plants, at artificial creatures in the game of life, and at seashells. We now look at the emergence of behavioral patterns, which can be interpreted by an external observer as some kind of "distributed intelligence". In this chapter we will proceed by inspecting some examples of robots and natural agents. We start with an experiment in collective robotics. We then discuss self-organizing phenomena in insect societies. Next, we briefly present Craig Reynolds's famous *boids*. Finally, we discuss some "guiding heuristics for decentralized thinking", as outlined by Mitchel Resnick.

3.1 An experiment: the Swiss robots

The Didabots are cleaning up

In what follows we summarize experiments conducted by Maris and te Boekhorst (1996) who studied a collective heap building process by a group of simple robots, called Didabots (see figure 3.2 (a) below). Instead of predefining "high-level" capacities, Maris and te Boekhorst exploit the physical structure of the robots and the self-organizing properties of group processes. The main idea behind the experiments is that seemingly complex patterns of behavior (such as heap building) can result from a limited set of simple rules that steer the interactions between entities (e.g., robots) and their environment. This idea has, for example, been successfully applied to explain the behavior of social insects (see below).

Now have a look at figure 3.1.

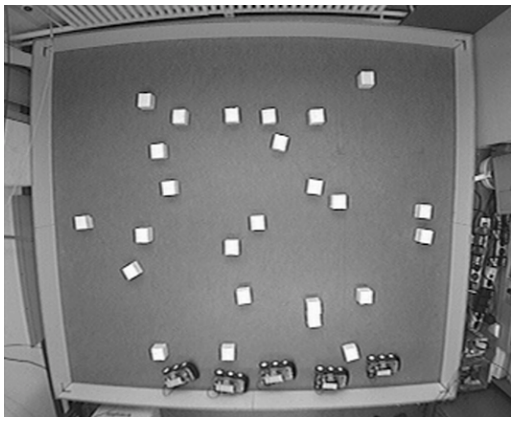


Figure 3.1: Didabots in their arena. There is an arena with a number of Didabots, typically 3 to 5. All they can do is avoid obstacles.

The Didabots present in the arena are equipped with infrared sensors that can be used to measure proximity. They show high activation if they are close to an object and low or zero activation if they are far away. The range of the infrared sensors is on the order of 5 cm, i.e., relatively short range. The sensors are located on the left and on the right side of the robots (see picture 3.2 (b) below). All the Didabots in this experiment

can do is avoiding obstacles. They are programmed with the following simple control rule: If there is sensory stimulation on the left, turn (a bit) to the right, if there is sensory stimulation on the right, turn (a bit) to the left.

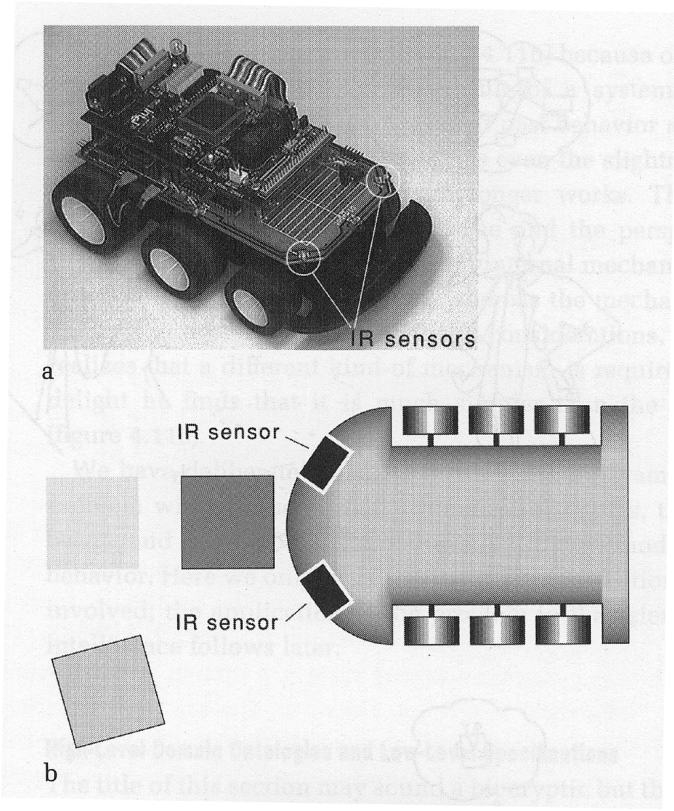


Figure 3.2 (a) Picture of a Didabot. (b) Infrared- Sensor configuration of Didabot.

Now look at the sequence of pictures shown in figure 3.3. Initially the cubes are randomly distributed. Over time, a number of clusters start to form. In the end, there are only two clusters and a number of cubes along the walls of the arena. These experiments were performed many times. The result is very consistent — there are always a few clusters and a few cubes left along the walls. What would you say the robots are doing?

“They are cleaning up”; “They are trying to build clusters of cubes”; “They are making free space”. These are typical answers, and they are fine if we are aware of the fact that they represent the observer’s perspective. They describe the behavior. The second answer even attributes an intention by using the word “trying”. Since we are the designers, we can say very clearly what the robots were programmed to do: to avoid obstacles!

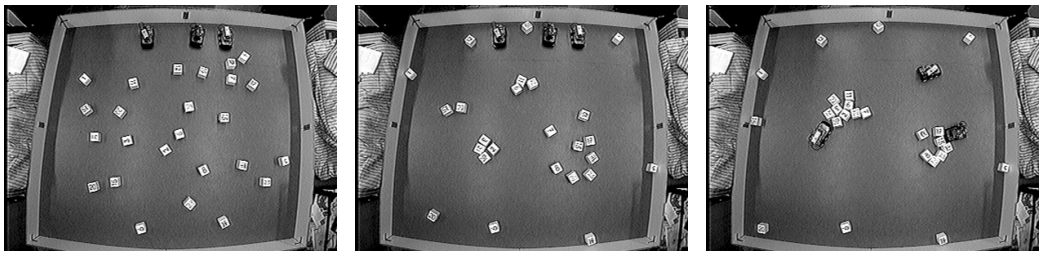


Figure 3.3: Example of heap building by Didabots. Initially the cubes are randomly distributed. Over time, a number of clusters start to form. In the end, there are only two clusters and a number of cubes along the walls of the arena.

The complexity of the behavior is a result of a process of self-organization of many simple elements: The robots with their simple control rule. The Didabots use the sensors on the front left and front right parts of the robot. Normally, they move forward. If they are too close to an obstacle, i.e., the obstacle is within reach of one of the sensors, they simply make a turn to the other side. If they encounter a cube head on, neither the left nor the right sensor detects an obstacle and the Didabot simply continues to move forward. At the same time, it pushes the cube. However, it pushes the cube because it does not "see" it, not because it was programmed to push it. For how long does it push the cube? Until the cube either moves to the side and the Didabot loses it, or until it encounters another cube to the left or the right. It then turns away, thus leaving both cubes together. Now there are already two cubes together, and the chance that yet another cube will be deposited near them has increased. Thus, the robots have changed their environment, which in turn influences their behavior. While it is not possible to predict exactly where the clusters will be formed, we can predict with high certainty that only a small number of clusters will be formed in environments with the geometrical proportions used in the experiment.

The kind of self-organization displayed by the Didabots in this experiment is also called self-organization without structural changes: If at the end of the experiments, the cubes are again randomly distributed and the Didabots are put to work on the same task, their behavior will be the same — nothing has changed internally. This is also the kind of self-organization displayed by physical systems, see for instance the famous Bénard cells (when a heat gradient is applied to a liquid, the individual molecules organize into "rolls"). As soon as the energy input is switched off, the system gets back to its original state. We talk about self-organization with structural changes, whenever something changes within the agent in order that the future behavior of the agent will be different. Such processes of self-organization with structural changes are found in the artificial chimp societies of Hemelrijk (see chapter 5.2). They are also found in the ontogenetic development of the brain. It is crucial that the organism changes over time, since otherwise it could not improve its behavior.

Similar principles as the ones observed in the Didabot experiments can also be found in natural agents such as ants and primates. Whereas in ants seemingly sophisticated group decisions may raise suspicion and induce a search for simpler mechanisms, this is not the case for primates (i.e., monkeys and apes). Let us now look at a number of examples.

3.2 Collective intelligence: ants and termites

Self-organization in a "super-organism"

In his article in the NZZ (see references) Rudiger Wehner describes societies of social insects composed of thousands of individuals, which have "cognitive abilities" that by far transcend the abilities of each of the individual members. This happens, as if the society is ruled by the invisible hand of a central organizer.

The distribution of brood and nourishment in the comb of honey bees is not random, but forms a regular pattern, which is organized in such a way, that the central brooding region is close to a region containing pollen and one containing nectar (providing protein and carbohydrates for the brood). Despite the fact that, due to the intake and outtake of pollen and nectar, this pattern is changing all the time on a local scale, observed from a more global scale, the pattern stays stable. By performing experiments, it has been discovered that this is not the result of an individual bee being aware of the global pattern of brood- and food-distribution in the comb, but of three simple local rules, which each individual bee follows. Please note, that the individual bee does not know whether and how the cells of the comb are filled with nectar and pollen, but it only follows the three simple rules stated below. In other words, these three rules are sufficient to create the global pattern.

1. Deposit brood in cells next to cells already containing brood.
2. Deposited nectar and pollen in discretionary cells but empty the cells closest to the brood first.
3. Extract more pollen than nectar.

By following these three local rules bees create a global distribution-pattern. Thus the distribution-pattern is an emergent phenomenon resulting from the application of local rules and an example of a process of self-organization in biology.

Another example of the combined application of local rules leading to a global result is the process of food-allocation and food-collection in honeybee colonies. The decision how many bees are collecting food, and where they are collecting it, depends on the time they have to wait when delivering the food at the entrance of the comb to the bees in charge of depositing such food in the respective cells. Based on the number of cells already filled, these bees need more time to find an empty cell and consequently the "search-bees" have to wait longer. This in turn leads them to search qualitatively more valuable food, which is usually more difficult to find and consequently, to spend more time searching for food.

Thus no central coordinator is needed to organize the search for food, and its storage. The parallel application of simple local rules solves the complex problem in a much more flexible and efficient way. Social insects are individuals, which by "working together in parallel" create a super-organism capable of solving even the most complex problems without any central organizer.

This idea has been taken up by social scientists and economists, who use computer simulations to study complex social and economic behavior. They design autonomous agents, which follow simple local rules in a specific environment and by copying ideas from biology and evolution they succeed in growing artificial societies bottom up.

The article on self-organization in a “super-organisms” by Rüdiger Wehner was distributed in class. Exact reference: Wehner, R. (1998). Selbstorganisation im Superorganismus. Kollektive Intelligenz sozialer Insekten. NZZ Forschung und Technik, 14, Januar 1998, S.61. This collective intelligence unites biologists, computer scientists, and economists in an interdisciplinary endeavor.

Referring back to our robot experiments in the previous section, we have another instance of sorting behavior. Sorting behavior is also observed in ants.

Deneubourg’s model of sorting behavior in real ants

The examination of an ant’s nest yields that brood and food are not randomly distributed, but that there are piles of eggs, larvae, cocoons, etc. How can ants do this? If the contents of the nest are distributed onto a surface, very rapidly the workers will gather the brood into a place of shelter and then sort it into different piles. Deneubourg and his colleagues show that this sorting behavior can be achieved without explicit communication between the ants.

The model works as follows. Ants can only recognize objects if they are immediately in front of them. If an object is far from other objects, the probability of the ant picking it up is high. If there are other objects present the probability is low. If the ant is carrying an object, the probability of putting it down increases if there are similar objects in its environment. Here are the formulas:

$$p(\mathit{pick\ up}) = \left(\frac{k^+}{(k^+ + f)} \right)^2$$

where f is an estimation of the fraction of nearby points occupied by objects of the same type and k^+ is a constant. If f=0, i.e., there are no similar objects nearby, the object will be picked up with certainty. If $f=k^+$, then *p(pick up)*=1/4 and if f approaches 1 *p(pick up)* decreases. The probability of putting down an object is

$$p(\mathit{put\ down}) = \left(\frac{f}{(k^- + f)} \right)^2$$

where f is the same as before and k^- is again a constant. p(put down) = 0 if f = 0, i.e., if there are no similar objects nearby the probability of putting the object down approaches 0. The more objects of the same type that are nearby, the larger is *p(put down)*. The development of the clusters for real ants and for a simulation is shown in figure 3.4. Sorting is achieved by these simple probabilistic rules. There is no direct communication between the ants. The sorting behavior is an emergent property.

(a) (b)

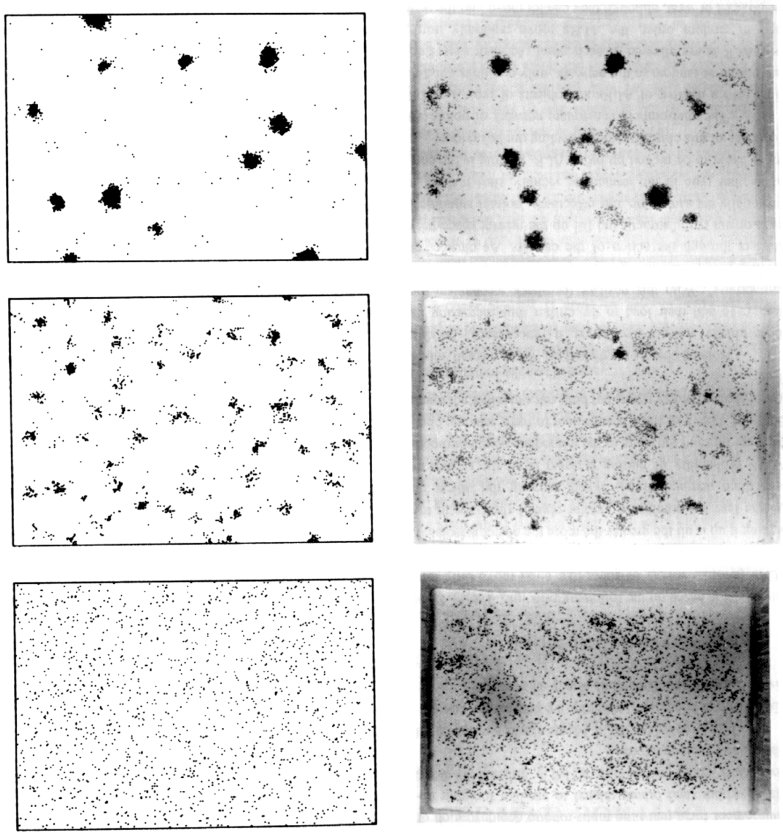


Figure 3.4. Development of clusters of objects in a society of ants. (a) Simulation. (b) Real ants. The simulation is based on local rules only. The simulated ants can only recognize objects if they are immediately in front of them. If an object is far from other objects, the probability of the ant picking it up is high. If there are other objects present the probability is low. If the ant is carrying an object, the probability of putting it down increases as there are similar objects in its environment. This leads to the clustering behavior shown.

While many people would agree that artificial life-like models have explanatory power for ant societies, they would be skeptical about higher animals or humans. Charlotte Hemelrijk and Rene te Boekhorst, two primatologists at the University of Zurich, who are interested in artificial life and autonomous agents. They are convinced that this kind of modeling technique can also be applied to societies of very high-level mammals like chimpanzees or orangutans. Hemelrijk uses computer simulations to study emergent phenomena in societies of artificial creatures, which, for her, are abstract simulations of orangutans. In an instructive paper entitled “Cooperation without genes, games, or cognition”, Hemelrijk (1997) demonstrates that cooperation in the sense of helping behavior is entirely emergent from interactive factors. Often, what seems to be a *tit-for-tat* strategy, as suggested by game theorists, turns out to be a side effect of interactions between agents. A tit-for-tat strategy is one where the individuals keep track of what has happened and only give back as much as they have received (see chapter 6.2). The more parsimonious explanations based on local rules of interaction also obviate explanations resorting to high-level cognition. For example, participants in a conflict are thought to keep track of the number of situations in which they have received help from, and they have given help to another individual. For more detail, the reader is

referred to Hemelrijk (1997). Along similar lines, te Boekhorst did a simulation of artificial "orangutans" demonstrating that travel band formation in orangutans can be explained in very simple ways (te Boekhorst and Hogeweg, 1994). This kind of research is predominantly done at the simulation level since often high-level operators like "recognize dominance rank" are used which cannot be translated to real robots in a straightforward manner.

Ants find their way to a food source

In their experiments on ants, Deneubourg and Goss (1989) tried to find an answer to the question if the complexity of social interactions might be attributed to the individuals or to their interactions. For instance, colonies of certain species of ants appeared to be able to select the nearest food source among several that were present at varying distances from the nest. Attributing the complexity of this phenomenon to the individual ants would imply that individual ants compare the distances to several food sources and on the basis of this knowledge choose the nearest food source. This would entail ample cognitive calculations. Instead, Deneubourg and Goss clarified this choice as a consequence of the pheromonal marking and following system of the ants. Ants mark their path with pheromone when they leave the nest to search for food as well as on their journey back to the nest. At crossings where several paths intersect, they choose the most heavily marked direction. Ants return sooner from nearer food sources and as a consequence, shorter paths are marked more intensively than those leading to sources further away. Self-reinforced differentiation of the degree of marking of a path is also called an *autocatalytic* process, a particular instantiation of self-organization (remember the autocatalytic processes in pattern formation in seashells). Such autocatalytic processes have been invoked to elucidate several other aspects of insect behavior as well, e.g., the observed strict spatial distribution of honey, pollen and youngsters in the comb of bees (see Wehner, 1998, which is referenced above) and the way a comb is built. In all cases autocatalytic effects form an alternative view to the idea that patterns are controlled centrally by a blueprint or higher cognitive capabilities of the individuals.

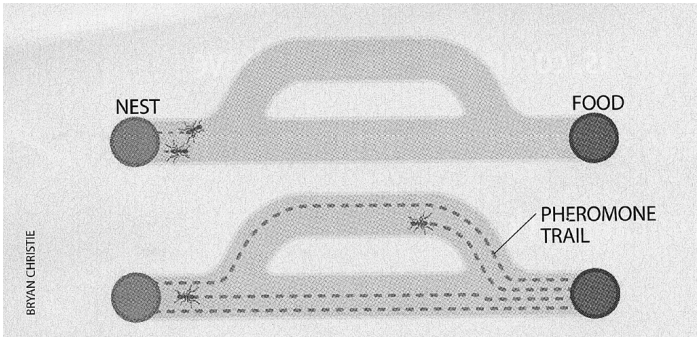


Figure 3.5 Pheromone trails enable ants to search for food efficiently: Two ants leave the nest at the same time (top), each taking a different path and marking it with pheromone. The ant that took the shorter path returns first (bottom). Because this trail is now marked with twice as much pheromone, it will attract other ants more than the longer route will.

3.3 The simulation of distributed systems: Starlogo

Starlogo grew out of the programming language Logo, which was originally developed for preschool children. Logo was simple enough in order that children could easily program simple robots, known as “turtles”. Initially, the turtles were real robots capable of moving around on a flat surface. They were equipped with a pen, which could be in one of two positions, either up, or down. When it was down the turtle drew a line on the ground while moving. The Logo turtles were designed so that children could playfully and intuitively learn concepts from geometry, mathematics, and engineering that are often considered hard to understand (see Papert, 1980). For example they could explore ideas of feedback, geometric figures like polygons and circles, and infinitesimality. Later, the turtles were “virtualized” and became a means for drawing on a computer screen rather than on the physical ground (see also the “turtle graphics” discussed in chapter 2.3 in the context of Lindenmayer systems). The commands that could be given to the turtles, like move forward, turn, pen-down remained the same. The language Logo has become very popular over the years and children like to play with it.

Starlogo is an extension of the traditional Logo language. First, there are many more turtles in Starlogo than in Logo. In a typical Starlogo program, there are literally hundreds or thousands of turtles. In other words, it is a massively parallel language. Second, in Starlogo turtles have been equipped with sensors. While traditional Logo turtles were mainly used for drawing purposes, Starlogo turtles have to behave in ways that strongly depend on their environment, in particular their local environment. The behavioral rules, which can be defined for the turtles, make it possible to model true turtle-environment interactions. Third, Starlogo offers the possibility to define so-called patches, which can be used to model the properties of the environment. For example, pheromones deposited by the ants will automatically evaporate, or food that is eaten by the turtles grows back at a particular rate. The patches are similar to the Cas, which were introduced in chapter 2. In summary, there are two types of rules, those for the turtles (or, more generally, the actors) and those for the environment. We will see precisely the same type of distinction later on, when we will discuss the Sugerscape model (chapter 5.1).

The basic tutorial for Starlogo has been distributed in class.

For further information refer to the StarLogo homepage:lcs.www.media.mit.edu/groups/el/Projects/starlogo

3.4 Flocking — the BOIDS

The *boids* are among the most famous creatures in field of artificial life. They were invented in the mid-80s by the computer animator Craig Reynolds. In Culver City in California where he lived, he would observe flocks of blackbirds. He wondered how he could get virtual creatures to flock in similar ways. His hypothesis was that simple rules were responsible for this behavior. It was clear to him that the boids would have to be agents: they would have to be situated, viewing the world from their own perspective, rather than from a global one. Their behavior is controlled by a certain number of local rules. He came up with the following set (Reynolds, 1987):

(1) Collision avoidance: avoid collision with nearby flockmates

(2) Velocity matching: attempt to match velocity with nearby flockmates

(3) Flock centering: attempt to stay close to nearby flockmates.

The first rule, collision avoidance defines the tendency to steer away from an imminent impact. Static collision avoidance is based on the relative position of the flockmates and ignores their velocity. Conversely, velocity matching, set out in the second rules, is based only on speed. The third rule is about flock centering. It makes a boid want to be near the center of the flock. Because of the situated perspective of the boid, "center of the flock" means the perceived center of gravity of the nearby flockmates. If the boid is already well within the flock, the perceived center of gravity is already at it's position, so there is no pull further towards the center. However, if the boid is at the periphery, flock centering will cause it to deflect somewhat from its path towards the center. Together, these three rules lead to surprisingly realistic flocking behavior (see figure 3.6).

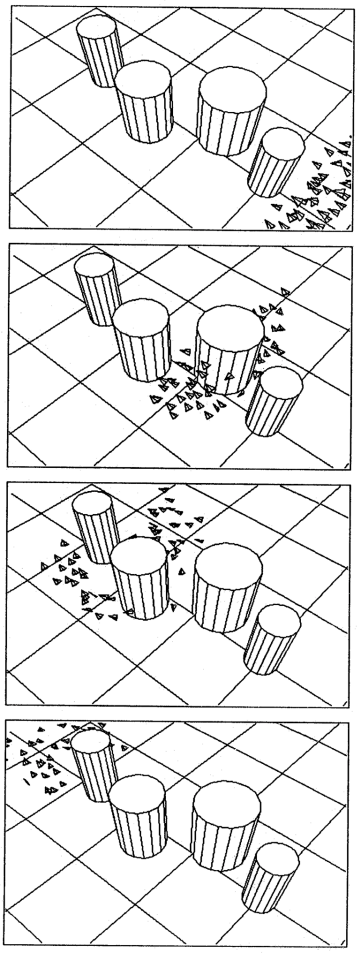


Figure 3.6: Craig Reynolds's "boids" engaged in flocking behavior. They encounter a cluster of pillars. Amazingly enough, the flock simply splits and rejoins after it has passed the pillars. Note that "splitting" is not contained in the set of rules. It is truly emergent: the result of several parallel processes while the boids are interacting with their environment. Even Reynolds was surprised by this remarkable and beautiful (although fully explainable) behavior.

Reynolds was interested in what would happen when the flock encountered obstacles. Would the boids continue to flock? Would they all move past the obstacle on one side? Or would they split? The latter happened (as can be observed in nature). Note that “splitting” was not programmed into the boids. Both, the flocking behavior and the splitting behavior are truly emergent phenomena. What is there, is just a number of internal parallel processes (obstacle avoidance, velocity matching, and flock centering), which are based on the boids’ situated view of the environment. The flocking behavior is very robust though. This is due to the local distributed nature of the mechanism. Another wonderful example of how sophisticated behavior emerges from simple rules.

Researchers in artificial life claim that their creatures are behaving creatures in their own right. Boids are digital creatures as such — they are not only models of real birds. “Flocking in boids is true flocking, and may be counted as another empirical data point in the study of flocking behavior in general, right up there with flocks of geese and flocks of starlings.” (Langton, 1989, p. 33).

Rodney Brooks, Pattie Maes, Maja Mataric, and Grinell Moore used rules almost exactly like Reynolds to achieve flocking behavior in real robots. At an IROS conference in 1990 (International Conference on Intelligent Robots and Systems), they suggested to use of a swarm of robots to prepare the lunar surface for a manned mission. Maja Mataric implemented flocking on her robots using a variation of Reynolds’ rules. Her robots, like the boids, exhibit robust flocking behavior. Again, flocking is emergent from local rules. Let us finish with a quote by Dan Dennett, a champion of the philosophy of mind: “Maja’s robots are flocking, but that’s not what they think they are doing.” (Dennett, 1997, p.251) What they think they are doing is applying Reynolds’ rules. This is another example of the notorious frame-of-reference issue. For those interested in collective robotics, Maja Mataric has investigated the field for many years. A thorough review would be beyond the scope of this class. The interested reader is referred to some of the review papers (Mataric, 1995, 1997).

As seen above there are two possibilities to reach a given behavior, either one simulates such behavior on a computer as Reynolds did with the Boids or, one builds robots which have to be able to behave in the given way in the real world, i.e., exhibit robust flocking behavior as Mataric did.

In general it is easier to simulate a behavior on a computer than to build robots and make them behave accordingly. In a simulation not only the agents but also the environment is predefined by the designer consequently the agents are familiar with the environment. Building robots, which are able to show a certain behavior in the real world, requires the designer to build the robots so that they can adapt to different environments.

Let us now look at rule number two “velocity matching” as an example. It is relatively easy to simulate flocking on a computer by designing agents, that are able to distinguish other agents from different objects in their environment, which always know where other agents are, and in which direction and at which velocity the other agents are moving. Building robots makes one realize that due to the limited technical means available (sensor, motor and computing technology), the aforementioned tasks are very difficult to achieve. One solution to this problem is to redesign the rules used in the simulation. By altering rule 2, in order that the robots only need to know the direction in which other robots are moving, one circumvents the

difficult task to regularly check place and distance of all other robots. Such a simplified rule resulting from the difficult implementation of a behavior in the real world can then be transferred back into the computer simulation.

Online resources are found at:

- www.cs.toronto.edu/~dt/siggraph97-course/cwr87/ (by Craig Reynolds)
- www.cse.unsw.edu.au/~conradp/java/Boids/example.html (a very nice Java applet)

3.5 Guiding heuristics for decentralized thinking

To conclude our discussion of distributed intelligence, and in order to make decentralized phenomena easier to grasp, this section provides a brief summary of the major points discussed in this chapter (adapted from Resnick, 1997).

(i) Positive feedback is not always negative

Positive feedback has a "negative image" problem. Whereas people see negative feedback as something, which helps to keep things under control, positive feedback is often seen as destructive. Below some examples of "negative" effects of positive feedback:

— Screeching sounds that result when a microphone is placed near to a speaker

— Population growth

— A vicious circle (German: Teufelskreis) is based on positive feedback. For example, reduction of service in public transportation, leads to such a vicious circle: Reduction in service leads to frustration of users, leading to decreases in users, which in turn necessitates further reduction of services, which ...

However, positive feedback is not always negative, but can have positive effects. It can help to create and expand patterns and structures. Actually it underlies a large number of phenomena in nature and society.

Some positive examples are:

— The emergence of Silicon Valley as an example of the geographic distribution of cities and industries. After a few high-technology companies had started, the required infrastructure in that area developed, and this made even more high-tech companies move to the region leading to even better infrastructure. This started a snowball effect.

— Winning standards/operating systems → "Macintosh problem". Due to good marketing strategies, availability and user friendly software, Windows became popular. Its popularity lead even more user to make use of Windows, with even more software, and turned it into the most successful software of all times. Of course, this development is seen as negative by many people.

— The formation of ant trails is influenced by the concentration of pheromone on the trail, the more pheromone the more ants take the respective trail (section 3.2).

— In the robot-clustering task (section 3.1) the positive feedback is given by the fact that the more cubes there are in a cluster, the higher is the probability, that yet another cube will be deposited nearby.

Often there is an interaction of positive and negative feedback. An instance of this interaction is the evaporation of pheromones in the formation of ant trails. As long as there is enough food in a given location most ants take the shortest trail to reach this source and the concentration of pheromones on this trail is constantly high, when the food source is exploited less ants choose the respective trail and the pheromone starts to evaporate which makes the ants switch to a different trail.

(ii) Randomness can help create order

Randomness is not just "disorder". Random perturbations – often created by a system itself - can have an important role in self-organizing systems being the "seed" needed to start the formation of patterns or structures:

— Traffic jams: As long as cars are distributed evenly on a road and all cars are driving at the same speed no traffic jams are formed. But even small fluctuations in traffic density and slightly different velocities of the cars can serve as "seed" for traffic jams; positive feedback then accentuates these density fluctuations, making the seed grow into full-fledged traffic jams;

— Randomness is required to achieve adaptivity in pheromone trails. If the ant societies are to remain adaptive, there must always be a random component in the ant's choice behavior (see the shortcut problem, chapter 4). This is important if this idea is to be applied to ant-based control, i.e., to message routing in telecommunication networks (chapter 4).

— Randomness together with positive feedback leads to phenomena like rhythmic clapping of an audience. Initially most clapping is unsynchronized but if accidentally some people are clapping simultaneously this often leads - accentuated by positive feedback - to rhythmic clapping.

(iii) A flock is not a big bird

One important and critical point in describing decentralized systems and self-organizing phenomena is the distinction of different levels. Interacting objects at one level lead to new objects in another level, objects in the first level often behave differently from the resulting object in the next level. The different phenomena and the different levels in which such phenomena occur should not be confused.

— People tend to confuse the behaviors of individuals within a group with the behaviors of groups as a whole, e.g., interactions among individual birds give rise to flocks, but the flock does not follow the same rules as its respective members.

— In many cases, the individuals in one level behave differently from objects in another level: watching traffic jams they tend to move backward, even though all of the cars within these jams are moving forward;

— In the Sugerscape model (see chapter 5), there are diagonal migration patterns even though the agents can only move up/down and left/right;

— The “leader” in a flock is always changing, but the flock as a group remains and does not change its behavior.

(iv) A traffic jam is not just a collection of cars

Beside the confusion about different levels, confusion might also arise based on the definition of objects. Although an object often consists of different individual parts, looking at it only as collection of parts might lead to misunderstandings. It is important to realize that some objects (“emergent objects” resulting from the interaction of lower-level objects) have an ever-changing composition, but as a whole form the same object, with its own set of rules.

— Over an individual’s lifetime, old cells die and new cells are born, but the individual remains the same.

— Ants that make up an “ant bridge” change continuously.

— Water making up the shape of a fountain is always different, but the shape is preserved.

(v) The hills are alive (the environment has an independent dynamics)

People often focus on the behaviors of individual objects, overlooking the influence of the environment that surrounds these objects and interacts with them. Especially in decentralized and self-organizing systems it is important to take the role of the environment into consideration since the same behavior of the individual objects leads to different patterns in different environments.

— The heap building process of the Didabots (section 3.1 above) can be influenced by the way the cubes are distributed at the beginning of the experiment (randomly but evenly distributed or in clusters);

— The path taken by ants from their nest to a food source is influenced not only by the capacity of the ants, but also by the complexity of the surroundings, i.e., the environment.

3.6 Conclusion

In this chapter we looked at some examples of distributed intelligence. We saw that behavioral patterns emerge from the interaction of individuals, which are equipped with simple local rules, and which are organized without a central organizer or coordinator. This process of self-organization can be observed in robots (heap building process of the Didabots), in computer simulations (flocking of the boids), and in natural agents (organization of nests in insect societies). The concept of self-organization has also been taken up by modern economists, which grow artificial societies in order to explain market phenomena. Starlogo represents a massively parallel computer language, which is particularly well suited for simulating decentralized self-organizing systems and system-environment interactions. Finally we outlined guiding heuristics for decentralized thinking and at the same time tried to prevent some misunderstanding that often appear when looking at decentralized phenomena and self-organizing systems.

Bibliography

te Boekhorst, I.J.A., and Hogeweg, P. (1994). Effects of tree size on travelband formation in orangutans: data analysis suggested by model study. In R. Brooks, and P. Maes (eds.): *Artificial Life IV Proceedings.* Cambridge, Mass.: MIT Press, 119-129.

Deneubourg, J. L. and Goss, S. (1989). Collective patterns and decision-making. *Ethology, ecology and Evolution,* **1**, 295-311.

Dennett, D.C. (1997). Cog as a thought experiment. *Robotics and Autonomous Systems,* **20**, 251-256.

Hemelrijk, C. K., (1997). Co-operation without games, genes or cognition. In P. Husbands and I. Harvey (eds.) Fourth *European Conference on Artificial Life.* Cambridge, Mass.: MIT Press, 511-520.

Langton, Ch. (1989). Artificial Life. In Ch. Langton (ed.). *The proceedings of an interdisciplinary workshop on the synthesis and simulation of living systems.* Redwood City: Addison-Wesley.

Maris, M., and te Boekhorst, R. (1996). Exploiting physical constraints: heap formation through behavioral error in a group of robots. In *Proc. IROS'96, IEEE/RSJ International Conference on Intelligent Robots and Systems.*

Mataric, M. (1995). Designing and understanding adaptive group behavior. *Adaptive Behavior*, **4**, 51-80.

Mataric, M. (1997). Learning social behaviors. In R. Pfeifer, and R. Brooks (eds.). *Robotics and Autonomous Systems*, Special Issue on "Practice and Future of Autonomous Agents", 191-204.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas.* New York: Basic Books.

Resnick, M. (1997). *Turtles, termites, and traffic jams. Explorations in massively parallel microworlds.* Cambridge, Mass.: MIT Press.

Reynolds, C.W. (1987). Flocks, herds, and schools: a distributed behavioral model. *Computer Graphics*, **21**, 25-34.

Wehner, R. (1998). *Selbstorganisation im Superorganismus. Kollektive Intelligenz sozialer Insekten*. NZZ Forschung und Technik, 14. Januar 1998, 61.

Chapter 4: Some applications of distributed intelligence — Ant Algorithms

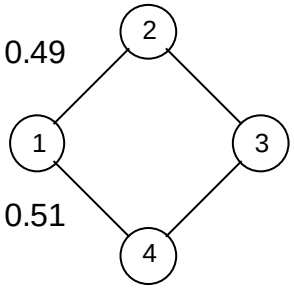
In the previous chapter we looked at a number of examples of distributed systems. Among others, we saw that ants are capable of finding the shortest path to a food source via a process of self-organization, mediated through pheromone trails (chapter 3. section 3.2). Through such stigmergic interactions[1], i.e. interactions mediated by modifications of the environment (depositing pheromones), they solve complex optimization problems. In addition to solving these problems, their behavior is robust, i.e. the "solution" is immune to noise. The ants also remain adaptive, i.e. if the environmental situation changes, for example if a path is no longer present, or a shortcut becomes available, they find different solutions. These characteristics have lead to the development of artificial systems, which were inspired by the observation of colonies of social insects in nature. These systems consist of several agents (e.g. artificial ants) with simple basic capabilities, which give rise to a highly complex structure as a resulting emergent behavior.

4.1 Ant Based Control

The remarkable properties of ants have encouraged researchers at Hewlett-Packard and British Telecom to try to apply these concepts to the problem of load balancing and message routing in telecommunications networks. Their network model is populated by agents (artificial ants), which make use of the trail-laying principles, i.e. they deposit pheromone on each node they visit during their trip through the network. The routing of calls is then decided based on the distribution of pheromone. Load balancing is essentially the construction of phone-call routing schemes that distribute the changing load over the system and minimize lost calls. Lost calls are those that never reach their destination (the caller gets only a beep signal).

Schoonderwoerd, Holland, Bruten, and Rothkrantz (1997) developed the following basic idea. Electronic ants are continuously generated at any node in the network and are assigned random destination nodes. On their way to the respective destination node ants move around in the network and leave their (electronic) "pheromone trails". They do this by updating the so-called pheromone tables in the routers (routing tables). Every node has a pheromone table for every possible destination in the network and the destinations' neighboring nodes. Thus a node with k neighbors in a network with n nodes has a pheromone table with $(n-1)$ rows, where each row corresponds to a destination node, and has k entries. The pheromone table contains probabilities (representing the amount of pheromone), which are regularly updated as soon as an ant reaches a node (see figure 1 below). Updating the probabilities thus represents the secretion of a pheromone.

[1] Stigmergy is a form of indirect communication through the environment, either by physically changing, or by depositing something on the environment.



dest. node / next node	2	4
2	0.95	0.05
3	0.49	0.51
4	0.05	0.95

Figure 1: A simple network configuration and the corresponding pheromone table for node 1. Ants traveling from node 1 to node 3 have a 0.49 probability to choose node 2 and a 0.51 probability to chose node 4 as their next node.

Since ants at every step have accurate recent information about their trip from the source node to their current node, the entries in the pheromone tables are updated with reference to the source node. Thus ants directly influence those ants traveling towards their source node and only indirectly those traveling in the same direction.

The probabilities *p* in the pheromone table are updated according to the following formulas:

The entry corresponding to the node from which the ant just came is increased

$$p_{new} = \frac{p_{old} + \Delta p}{1 + \Delta p}$$

All other entries are decreased $$p_{new} = \frac{p_{old}}{1 + \Delta p}$$

Here Δp is the probability change given by the age of the ants (see below). Note that the influence of a given Δp is much greater on small, rather than large probabilities (p_{old}). Thus the entries of rarely used nodes, those nodes with small probabilities, increase faster if traversed by ants.

In order to distinguish the length of the different routes taken ants get older with every time step while moving along the network or while being delayed at congested nodes. The older an ant gets the smaller the amount of pheromone it is able to lay and thus the smaller its influence on the update of the pheromone tables. The value Δp used to change the entries in the pheromone tables is reduced in accordance with the age of an ant.

Ants knowing their source and destination node choose their path according to the probabilities stated in the pheromone tables. Further details of how this is done, are given in the paper. The pheromone tables are then used to route an incoming phone call. Note that calls operate independently of the ants. The route of an incoming call is decided according to the probabilities in the pheromone table. The node with the largest probability will usually be the next node visited on its way to its destination node. Calls influence the loads on the nodes and thus introduce delays, and thereby influence the age of the ants visiting the same node.

This leads to a complex interaction between the electronic ants and the calls that are routed over the network.

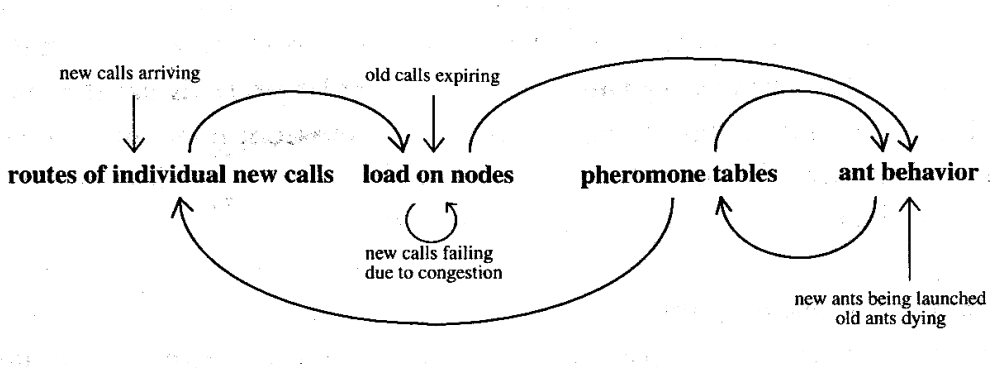


Figure 2: Relationship between calls, node utilization, pheromone table, and ants. An arrow indicates the direction of influence.

The pheromone mechanism of the natural ants had to be changed in order to have different pheromones in the routing tables for each destination node. When trying to map ideas from nature to technology, it is normal that nature's solutions cannot be applied directly, but need to be appropriately modified. The authors also compare the ant-based control method to other methods (shortest path routing, and a software agent based method). Although there are some tradeoffs, ant-based control scores well in these comparisons. One problem with ant-based control is that whenever there is a significant change in the network, it takes some time before the ants "discover" and mark the new routes with pheromones.

The details can be found in the Schoonderwoerd et al. (1997) paper (see references).

Comment: The article was distributed in class.

4.2 Ant Algorithms for Optimization Problems

Based on the way ant colonies work, new algorithms, called ant algorithms or ant colony optimization algorithms, have been developed. These algorithms are especially suited for finding solutions to difficult discrete optimization problems. A colony of artificial ants cooperates to find good solutions, which are an emergent property of the ants' cooperative interaction. Based on their similarities with ant colonies in nature, ant algorithms are adaptive and robust and can be applied to different versions of the same problem as well as to different optimization problems.

The main traits of artificial ants are taken from their natural model. These main traits are: (1) artificial ants exist in colonies of cooperating individuals, (2) they communicate indirectly by depositing (artificial) pheromone (stigmergic communication), (3) they use a sequence of local moves to find the shortest path from a starting, to a destination point (i.e. the optimal solution to a given problem), and (4) they apply a stochastic decision policy using local information only (i.e. they do not look ahead) to find the best solution. If necessary in order to solve a particular optimization problem, artificial ants have been enriched with some additional capabilities not present in their natural counterparts.

In ant systems (ant algorithms) an ant colony of finite size searches collectively for a good solution to a given optimization problem. Each individual ant can find a solution or at least part of a solution to the optimization problem on its own but only when many ants work together can they find the optimal

solution. Since the optimal solution can only be found through the global cooperation of all the ants in a colony, it is an emergent result of such this cooperation. While searching for a solution the ants do not communicate directly but indirectly by adding pheromone to the environment. Based on the specific problem an ant is given a starting state and moves through a sequence of neighboring states trying to find the shortest path. It moves based on a stochastic local search policy directed by its internal state (private information), the pheromone-trails, and local information encoded in the environment (together public information). Ants use this private and public information in order to decide when and where to deposit pheromones. In most applications the amount of pheromone deposited is proportional to the quality of the move an ant has made. Thus the more pheromone, the better the solution found. After an ant has found a solution, it dies, i.e. it is deleted from the system.

Applications of such ant algorithms can be divided into two classes: ant algorithms for static, and ant algorithms for dynamic combinatorial optimization problems. In static problems the key-points of the problem are defined at the beginning and do not change while the problem is being solved. In dynamic problems the problem changes as a function of itself, thus the algorithms used to solve such problems must be able to adapt “online” to the changes.

Examples of applications to the first class of problems, i.e. to static combinatorial optimization problems are:

(1) Traveling Salesman Problem: In this problem, a salesman must find the shortest route while visiting a given number of cities, each city exactly once.

(2) Quadratic Assignment Problem: Problem of assigning *n* facilities to *n* locations so that the costs of the assignment are minimized.

(3) Job-Shop Scheduling Problem: where – given a set of machines and a set of jobs - operations must be assigned to time intervals in such a way that no two jobs are processed at the same time on the same machine and the maximum of the completion times of all operations is minimized.

(4) Vehicle Routing Problem: In this problem, the object is to find minimum cost vehicle routes such that (a) every customer is visited exactly once by exactly one vehicle, (b) for every vehicle the total demand does not exceed the vehicle capacity, (c) the total tour length of each vehicle does not exceed a given limit, and (d) every vehicle starts and ends its tour in the same position (the depot).

(5) Shortest Common Supersequence Problem, where – given a set of strings over an alphabet – a string of minimal length that is a supersequence of each string of the given set has to be found (a supersequence *S* of string *A* can be obtained from *A* by inserting zero or more characters in *A*).

(6) Graph-Coloring Problem: This is the problem of coloring of a graph so that the number of colors used is minimal but no elements of the same color are adjacent.

(7) Sequential Ordering Problem, which consists of finding a minimum weight Hamiltonian path[2] on a directed graph with weights on the arcs and on the nodes, subject to precedent constraints among the nodes.

The main focus of applications to the second class of problems, to dynamic combinatorial optimization problems is on communication networks, in particular on routing problems. Routing answers the question of how to direct data traffic (e.g. phone calls) through a network, i.e. which node to choose next for a data packet entering the network. Routing mainly consists of building, using and updating routing-tables.

Implementations for communication networks can be divided in two classes

(a) Connection-Oriented Network Routing, where all packets of the same session follow the same path selected by a preliminary setup phase (see the Ant Base Control algorithm as explained in section 4.1 above), and

(b) Connectionless Network Routing where data packets of the same session can follow different paths (Internet-type networks).

More details on ant algorithms can be found in Dorigo, M., Di Caro, G., Gambarella, L.M., 1999.

4.3 Conclusion

Ant Algorithms are good examples of the application of swarm intelligence. They show that algorithms inspired by the observation and application of basic principles of a particular natural phenomenon can lead to the discovery of good solutions to diverse optimization tasks. One of the main characteristics of these algorithms is the fact that good solutions are an emergent property of the cooperative interaction of simple agents. Another characteristic is the indirect (stygmergetic) communication (indirect communication mediated by changes in the environment) used by the agents.

Beside the examples shown above, there are a variety of current research efforts using swarm-intelligence based approaches. Insect swarms are – among others - studied to devise different techniques for controlling groups of robots which have to cooperatively transport heavy goods, to find more efficient methods to assign jobs in factories, to solve manufacturing problems, to find information over the World Wide Web and in other large networks and to analyze financial data. In addition thereto there is still a large number of potential applications to be explored.

[2] A Hamiltonian path is a path in which every node is visited only once (see Traveling Salesman Problem above).

Bibliography

Dorigo, M., Maniezzo, V., Colorni, A., *Ant System: Optimization by a Colony of Cooperating Agents*, in: *IEEE Transactions on Systems, Man and Cybernetics- Part B: Cybernetic, Vol. 26, No. 1*, February 1996, 29-41

Dorigo, M., Di Caro, G., Gambarella, L.M., *Ant Algorithms for Discrete Optimization*, in: *Artificial Life 5*, 137-172, 1999 Massachusetts Institute of Technology

Schoonderwoerd, R., Holland, O.E.; Bruten, J.L., Rothkrantz, L.J.M., *Ant-Based Load Balancing in Telecommunications Networks*, in: *Adaptive Behavior Vol. 5*, No. 2, 169-207, 1997, Massachusetts Institute of Technology

Chapter 5: Agent-based simulations

The term “agent-based” refers to a particular type of simulation. Agent-based simulations have two essential components, agents and environment. An agent’s behavior is the result of simple rules based on local interactions. The environment has certain autonomy, i.e. it has a certain level of independence from what the agents do, but it can also be influenced by the agents’ behavior. The interaction of agents among each other, as well as the interaction of agents with the surrounding environment is modeled. This is in contrast to more traditional simulations where often systems of differential equations are integrated. Examples of agent-based simulations are Josh Epstein and Bob Axtell’s “Sugarscape” model (Epstein and Axtell 1996), John Casti’s “Would-be worlds” (Casti, 1997), Hemelrijk’s simulations of artificial monkeys (Hemelrijk, 1998a, 1998b, 1999), and, of course, the StarLogo simulations that were described in chapter 3 (Resnick, 1994).

5.1 The Sugarscape model

The Sugarscape (SSC) model consists of an environment, a landscape, with a particular distribution of various resources (e.g. sugar) that the agents need for their survival. Some regions are rich in sugar, some poor. Agents have certain traits like vision (how far they can see), or metabolism (how fast they consume resources). Agents can “sense” their surrounding (local) environment in order to decide in which direction to move, and they can “eat” the sugar they find on their way. With every movement an agent burns a particular amount of sugar, equal to its metabolic rate. Agents are considered dead once they have burnt up all their sugar. A remarkable number of phenomena emerge from the interaction of these simple agents.

The Sugarscape scenario can be made arbitrarily complex by introducing, for example, seasons, pollution, reproduction, additional resources, trade, markets, legislation (e.g. inheritance), credits (borrowing and lending), diseases, etc. In this way, a kind of laboratory for the social sciences can be constructed. New kinds of questions can be asked and old questions can be answered in new ways. An example is the field of “agent-based economics”, which investigates economic questions without making assumptions about being near equilibrium.

We now look at a few examples in more detail.

Sugarscape is a grid world. Figure 1a shows the distribution of sugar in the grid world, figure 1b has an initial distribution of agent added.

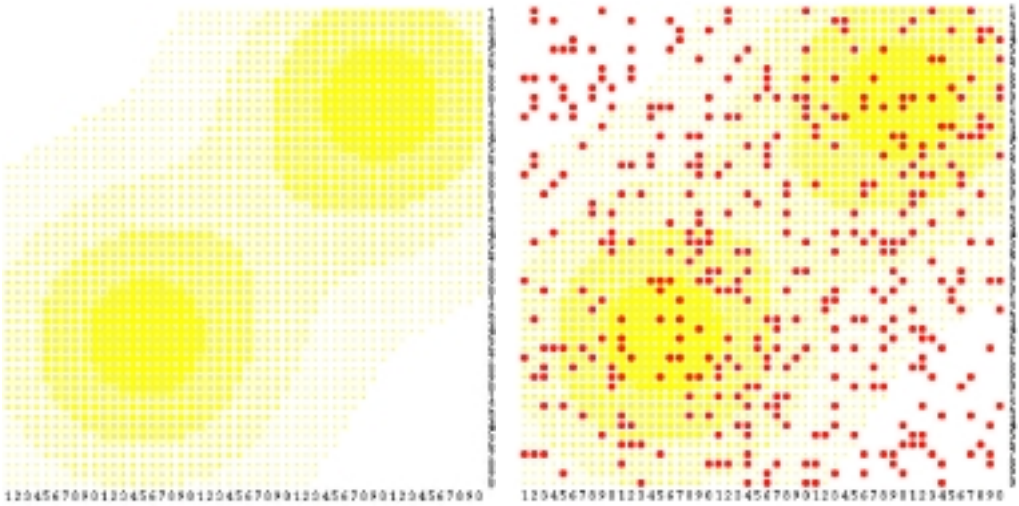


Figure 1: (a) Distribution of sugar. (b) Initial distribution of agents.

Rules

There are two kinds of rules, for the environment and for the agents.

Environment rules:

— 2-dimensional grid (lattice) (50x50)

— Specified for each location (x,y):

- current sugar level

- capacity (i.e. the maximum possible amount of sugar in that particular location)

— Distribution: peaks North East and South West — terraces

— Sugar levels: 0 to 4

SSC grow back rule G_α*:* At each lattice position, sugar grows back at a rate of α units per time interval up to the capacity at that position. If the sugar grows back instantaneously, the rule is called G_∞.

Agent rules:

Agents are characterized by a set of fixed and variable states.

Fixed: Metabolism (amount of sugar used per unit time), field of vision.

Variable: Amount of sugar. Agents are given some initial endowment of sugar, which they carry with them as they move around in the Sugarscape. Sugar, which is collected, but not metabolized, is added to the agent's sugar store. There is no limit to how much sugar an individual agent may accumulate.

Figure 2 illustrates the agent's field of vision. It can only see in the directions of the arrows, it does not see the shaded areas.

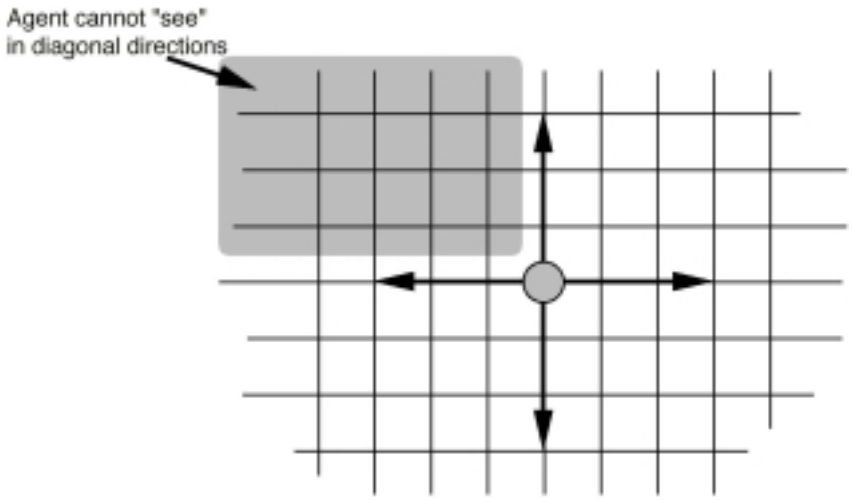


Figure 2: Agent vision. In this example, the agent can only see north, east, south, and west, but not northeast, etc.

Agents move according to the following “agent movement rule”:

Agent movement rule M:

— Look as far as vision permits in the four principal lattice directions and identify the unoccupied site(s) having the most sugar.

— If the biggest sugar value appears on multiple sites then select the nearest one.

— Move to this site (the agent can only move in four directions, i.e. North, East, South, West, but not North-East, North-West, etc.).

— Collect all the sugar in this new position.

The sugar level of the agent is incremented by the amount of sugar available on the new grid point, and decrements by its metabolic rate.

Figure 3 shows the evolution of the Sugarscape model under rules ({G_∞} {M}).

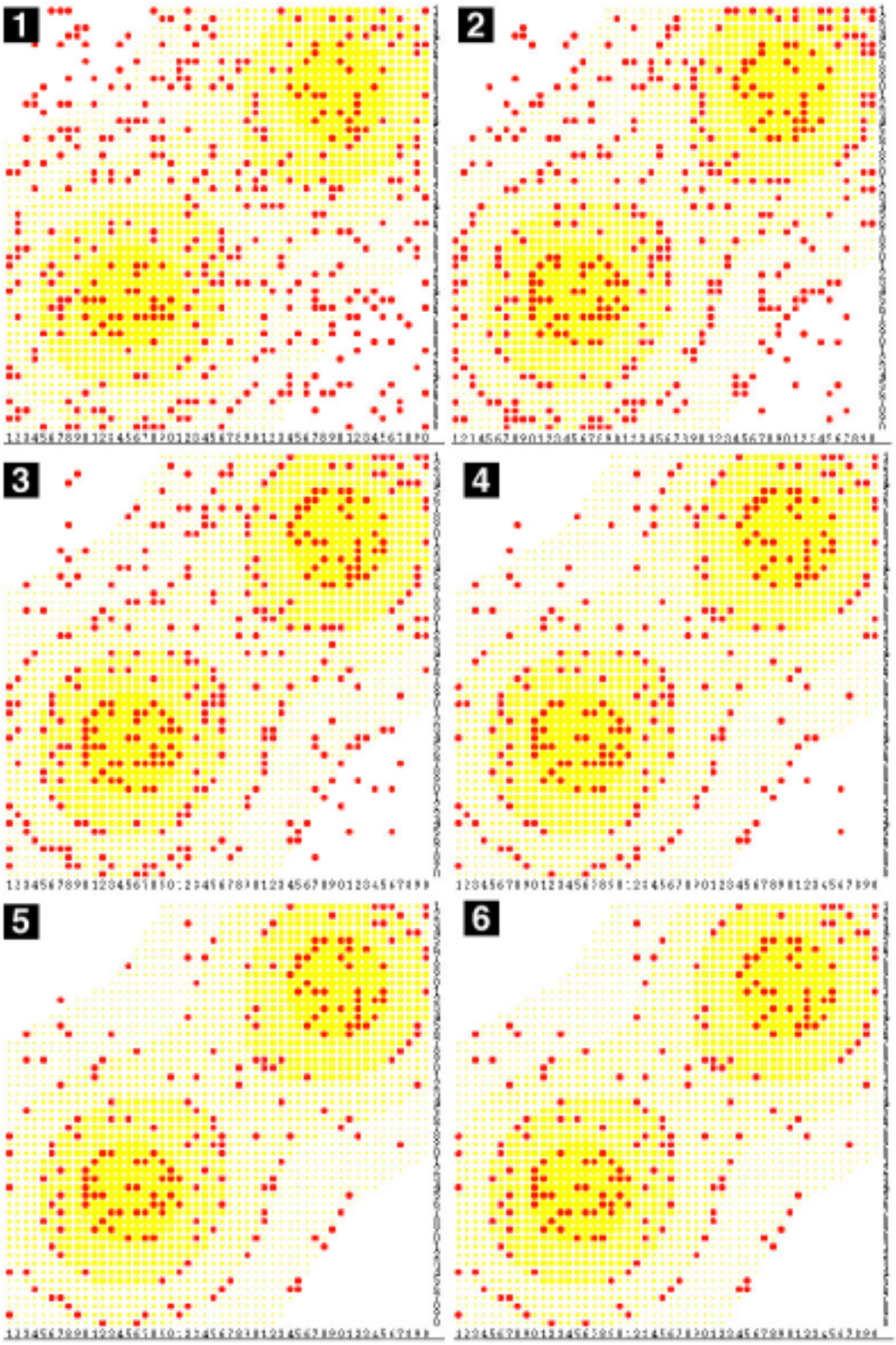


Figure 3: The evolution under rules ({G∞} {M}). It can be clearly seen that the agents start forming regular patterns, i.e. they gather along the edges of the terraces.

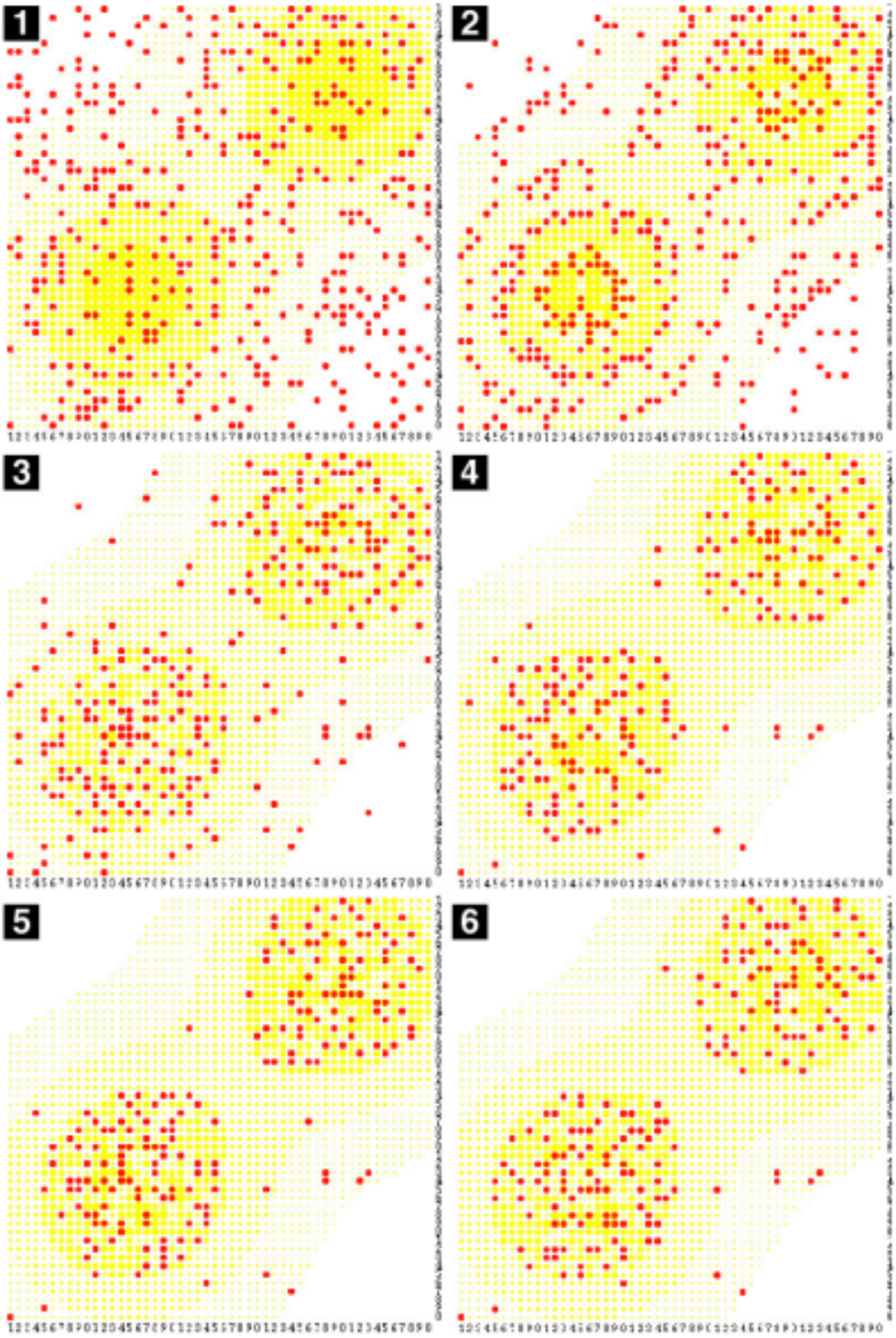


Figure 4: Development with the grow back rule G_1 that states that everything grows back to capacity within one time step.

Two relatively separate communities are forming, each sitting on top of one of the peaks in the Sugarscape.

Figure 5 shows the population development over time for rules ({G_1} {M}). It converges to an asymptotic value of a value somewhat above 200. This value is called the carrying capacity, i.e. the number of individuals that this particular environment can support. Figure 6 displays the carrying capacity as a function of mean vision for three different values of metabolism (m=1, 2, 3). Trivially, for small metabolism the carrying capacity is higher. There is also a slight increase as mean vision increases: Because agents can see further, they are, in a sense, more fit to survive.

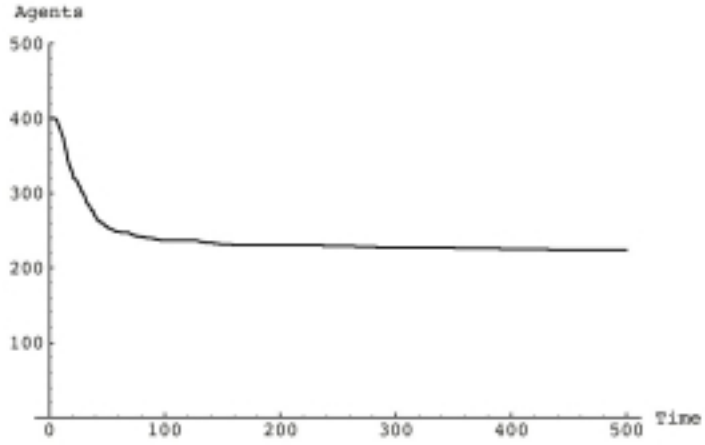


Figure 5: Population development over time.

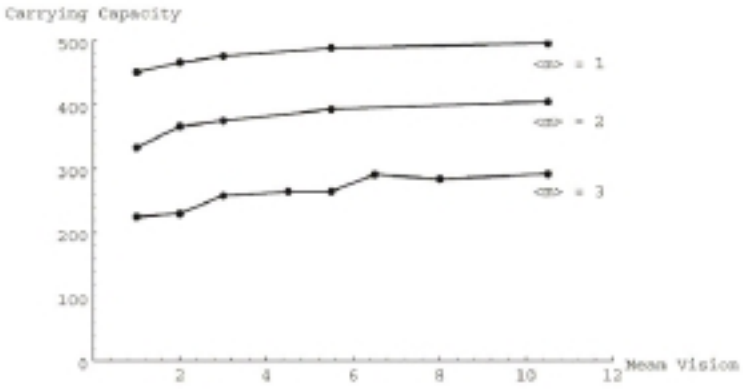


Figure 6: Carrying capacities as a function of mean agent vision for three different values of metabolism.

Wealth distribution:

So far, agents have had indefinite life span as long as they had sufficient sugar supply. If we limit the age, we have to define an "agent replacement rule" R[a,b]:

Agent replacement rule $R_{[a,b]}$: When an agent dies it is replaced by an agent of age 0 having random genetic attributes, random position on the Sugarscape, random initial endowment, and a maximum age randomly selected from the range [a,b]. Figure 7 shows the development of the wealth distribution over time using

agent replacement rule $R_{[60,100]}$. A wealth distribution emerges from the local rules defined earlier: A very small portion of the population owns the better part of the wealth available. This is an example of a macroscopic pattern arising from local interaction of agents. The term *self-organization* is also used for this type of phenomenon. What evolves is in fact a society of economic inequality.

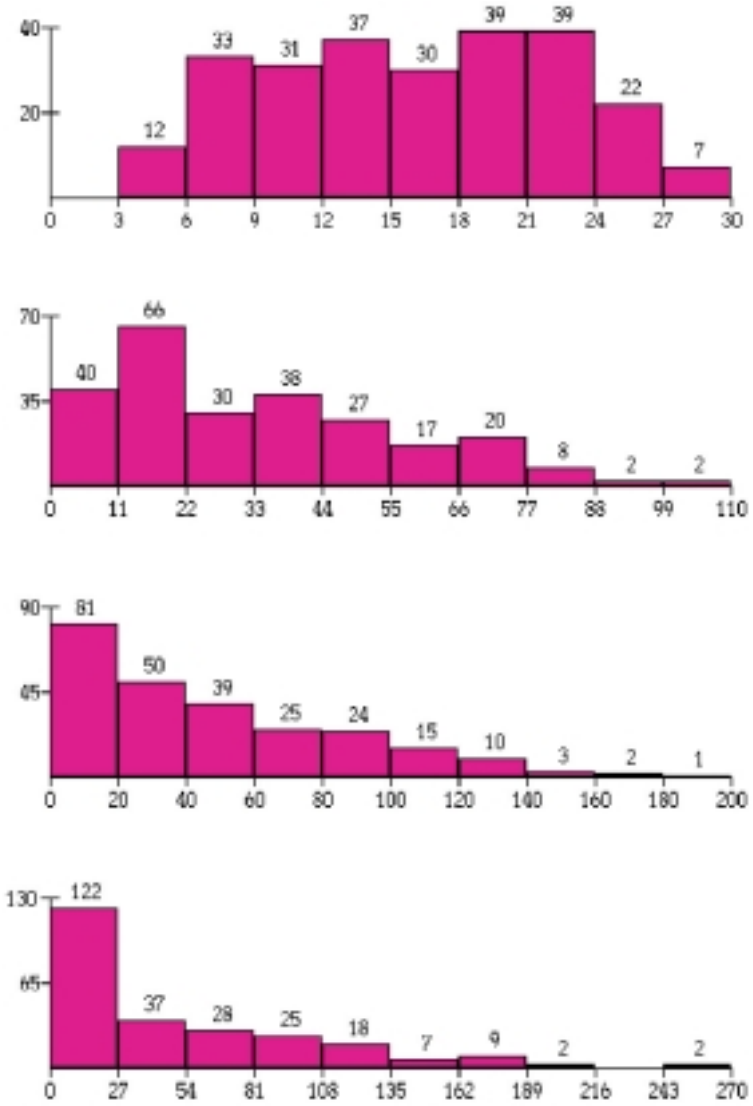


Figure 7: Wealth histogram under rules ($\{G_1\}$, $\{M, R_{[60,100]}\}$) from a random initial distribution of agents.

Migration:

If we start with a distribution of agents in one corner, as shown in figure 8, we get waves propagating through the landscape. It is interesting to note that the waves propagate from lower left to upper right in a diagonal motion even though individual agents can only move north, east, south, and west. Diagonal movement is something not available to individual agents. The diagonal waves are emergent from the local rules.

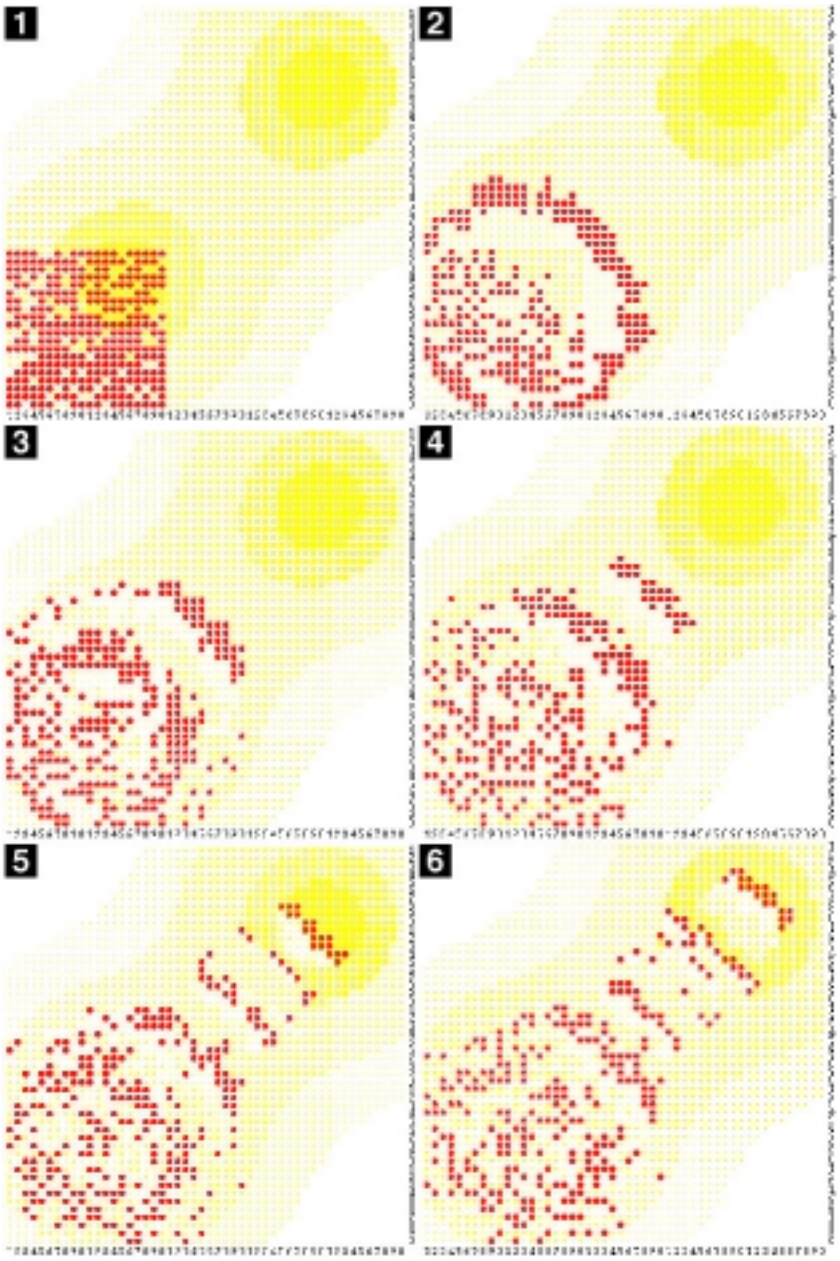


Figure 8: Emergent diagonal waves of migrators under Rules ({G_1}, {M}) from an initial distribution of agents in a square in the lower right corner.

If seasons are introduced, the migration patterns change. Seasons can be simulated by having different grow-back rates for sugar.

Pollution can be introduced by having (a) a pollution formation rule, (b) a pollution diffusion rule, and (c) and agent movement rule modified for pollution.

Pollution formation rule $P_{\alpha\beta}$: When sugar quantity is gathered from the Sugarscape, a certain amount of pollution, collection pollution, is produced, i.e. αs. When sugar amount m is consumed (metabolized), pollution is also produced, consumption pollution, βm. The total pollution on a grid point at time t, p^t, is the sum of the pollution present at the previous tine, plus the pollution resulting from production and consumption activities, that is,

$$p^t = p^{t-1} + \alpha s + \beta m. \quad (1)$$

Agent movement rule M modified for pollution:

— Look as far as vision permits in the four principal lattice directions and identify the unoccupied site(s) having the maximum sugar to pollution ratio.

— If the maximum sugar to pollution ratio appears on multiple sites, then select the nearest one.

— Move to this site.

— Collect all the sugar at this new position.

Pollution diffusion rule D_α:

— After α time periods and at each site, compute the pollution flux - the average pollution level over all von Neumann neighboring sites.

— Each site’s flux becomes its new pollution level.

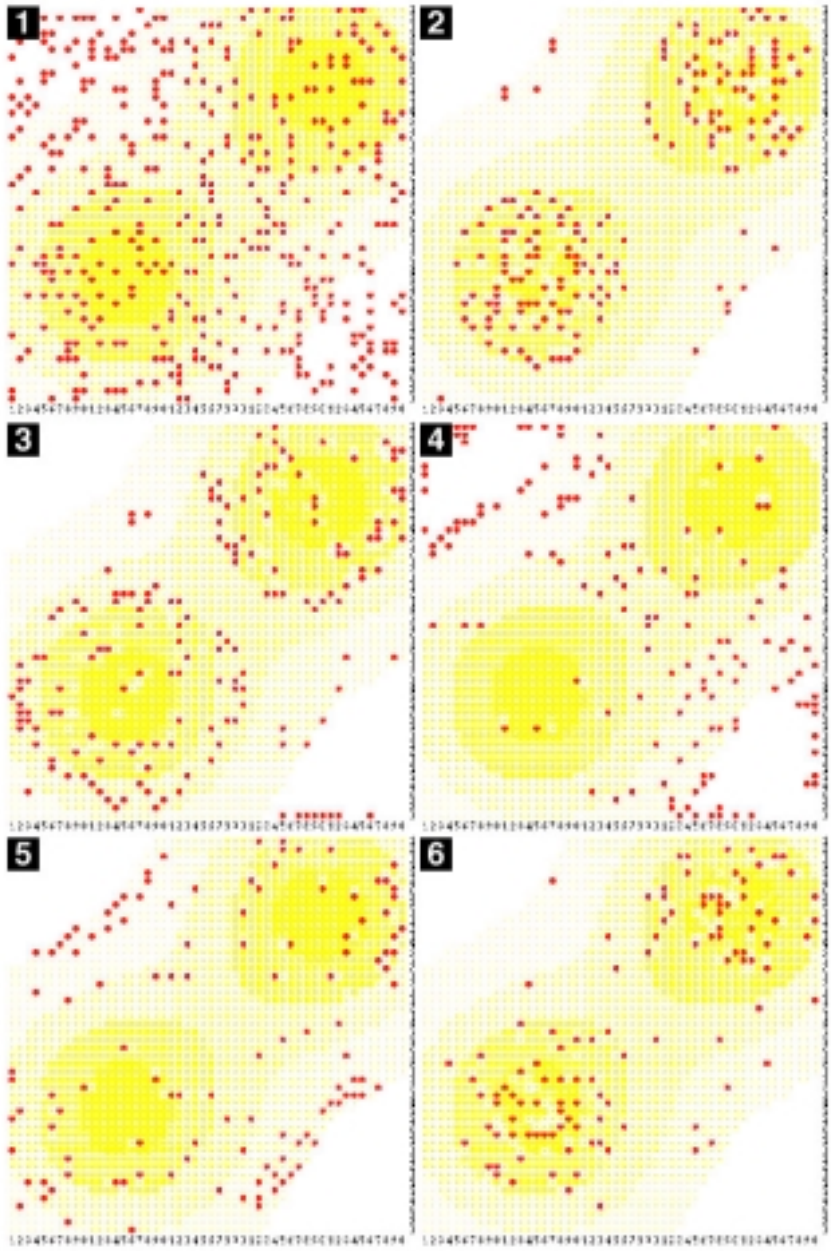


Figure 9: Agent migration over time under this pollution diffusion rule system ({G_1, D_1}, {M, P_{11}}). Initially, everything is the same. Over time, pollution starts to build up and agents leave the polluted regions. The result is also a reduced carrying capacity.

In summary, this case study demonstrates the surprising sufficiency of simple local rules to produce emergent structures. The interesting part is, that the rules are local and appear quite far from the social, collective phenomena they entail. Good examples are the migration patterns (the diagonal waves) and the skewed wealth distribution. The Sugarscape model can be used as some sort of *laboratory in silico*, in which we can "grow" fundamental social structures, that help us learn more about what type of micro mechanisms are sufficient for generating macrostructures of interest.

Increasing the complexity

In principle the Sugarscape model can be made arbitrarily complex. Let us just look at a few examples. We can introduce sexual reproduction and study evolution in our laboratory. In order to be able to reproduce,

agents must fulfill two conditions: (1) they must be of old enough and (2) they must have a sufficient amount of sugar, in order that they can share it with their offspring. In addition we need a rule of sexual reproduction.

Agent sex rule S:

— Select a neighboring agent at random.

— If the neighbor is fertile and of the opposite sex and at least one of the agents has an empty neighboring site (for the baby), then a child is born.

— Repeat for all neighbors.

All agents, including babies, use the same agent movement rule M. The sex of each child is determined randomly. The child's genetic makeup (metabolism, vision, maximum age, etc.) is determined from parental genetics through Mendelian rules. A simple example for vision and metabolism is given in table 1. Assume that one parent has (v, m), the other (V, M):

Table 1: Crossover of genetic attributes in sexual reproduction.

	Metabolism	
Vision	m	M
v	(m,v)	(M,v)
V	(m,V)	(M,V)

One of the results of these experiments is demonstrated in figure 10, showing mean vision and metabolism.

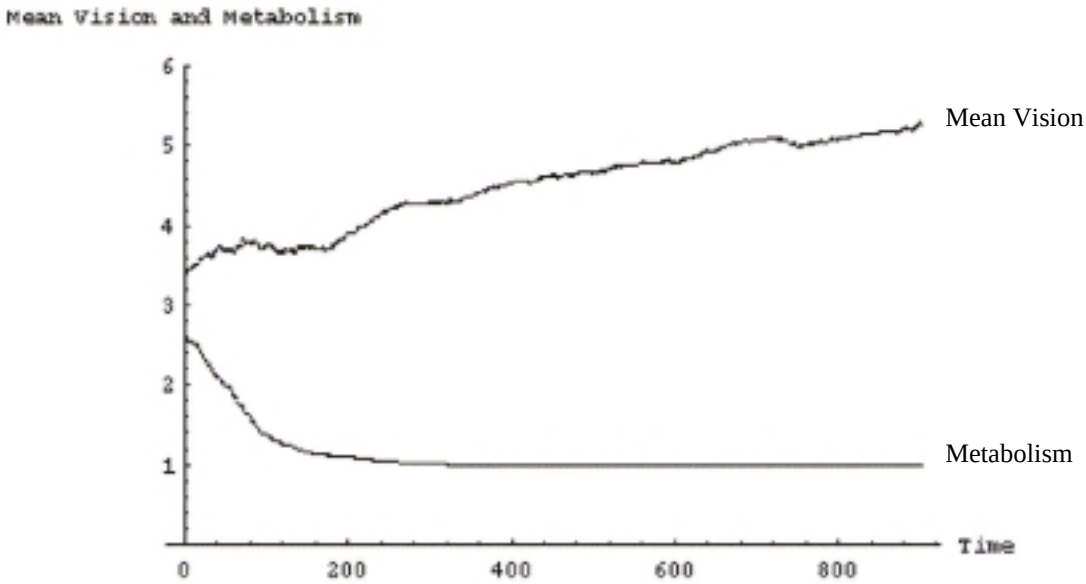


Figure 10: Evolution of mean agent vision and metabolism under rule set ({G_1}, {M,S}).

Figure 10 shows that mean vision increases and mean metabolism decreases. One interpretation of this is that the average fitness in the society has increased. But these fitter agents might be the cause of their own extinction (e.g. through overgrazing and explosive reproduction). Then, even though they have high vision, the overall fitness of the society would be low. A sustainable co-evolution with the environment is a necessary condition for "fitness". Experiments with Sugarscape suggest that fitness should be conceived as another emergent property, not as something—such as vision—that can be determined by inspection of isolated individuals.

Let us add inheritance to the social system.

Agent inheritance rule I:

When an agent dies its wealth is equally divided among all its living children.

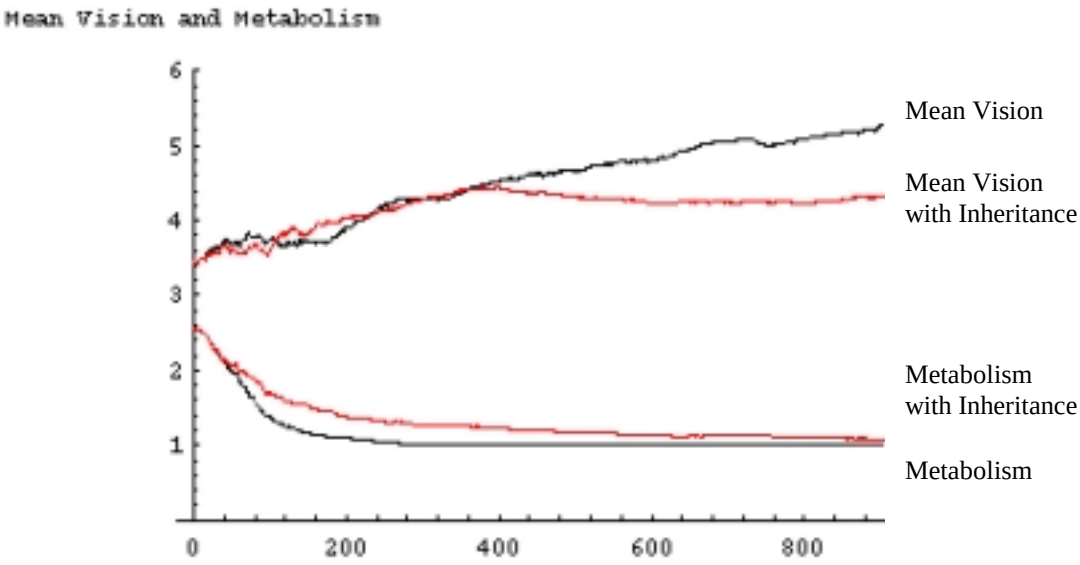


Figure 11: Evolution of mean vision and metabolism under rule set ({G_1}, {M,S,I}).

Note that the selective pressures for vision is mitigated by the inheritance rule I! In other words, there is an interaction of social rules (legislation), and biological traits of individuals. To illustrate the explosive content potentially contained in these innocuous simulations, let us quote Epstein and Axtell (1996, p. 68):

> "Interestingly, some 'Social Darwinists' oppose wealth transfers *to the poor* on the ground that the undiluted operation of selective pressures is 'best for the species.' Conveniently, they fail to mention that intergenerational transfers of wealth *from the rich to their offspring* dilute those very pressures."
>
> [Social] inequality grows under inheritance.

Sugar and spice: the beginnings of agent-based economics

Let us now introduce an additional commodity, spice. At each point on the grid there are now two values, one for sugar and one for spice. A similar distribution as for sugar is assumed for spice: a "hill" in the Northwest and Southeast. Each agent not only requires sugar for its metabolism, but also spice. Each agent

keeps two separate accumulations, one for sugar and one for spice, and has two distinct metabolisms, one for each good. Agents die if either their sugar or their spice accumulation falls to zero.

The agent welfare function

A "rational" agent having, say, identical metabolic rates for sugar and spice, but with a large accumulation of sugar and small amounts of spice, should look for sites having more spice than sugar. One way to model this is to have the agent compute how "close" it is to starving to death due to the lack of either sugar or spice. Imagine an agent with metabolisms m_1, and m_2 and accumulations w_1, and w_2. The amount of time until death from starvation given no further resource gathering, is simply

$$\tau_1 = w_1 / m_1; \tau_2 = w_2 / m_2 \tag{2}$$

The relative size of these two quantities is a measure of the relative importance of finding sugar or spice. The welfare function is

$$W(w_1, w_2) = w_1^{m_1/m_\tau} w_2^{m_2/m_\tau} \tag{3}$$

where $m_\tau = m_1 + m_2$. This function is important and can be used by the agent for deciding to which field to move.

Multi-commodity agent movement rule M:

— Look as far a vision permits in each of the four lattice directions.

— Find the nearest position producing maximum welfare, considering only unoccupied lattice positions.

— Move to the new position.

— Collect all the resources at that location.

Rules of trade

Assume now that an agent is low in sugar but has got a lot of spice, and another nearby agent has a lot of sugar but a small amount of spice. They can both benefit by trading. This immediately raises a number of issues. When will agents trade? How much will they trade? And at what price will exchange occur? There are a variety of ways in which to proceed, depending on the particular economic theory one favors. Here, we only give a very crude characterization. For more details, the interested reader should refer to the excellent book by Epstein and Axtell (1996).

Let us just briefly mention the trade rule. The two key quantities in the trade rule are the MRS, the marginal rate of substitution, and the price. An agent's MRS of spice for sugar is the amount of spice the agent considers to be as valuable as one unit of sugar, that is, the value of sugar in units of spice. It can be shown that this MRS for the welfare function (3) is:

$$MRS = \frac{dw_1}{dw_2} = \frac{\frac{\partial W(w_1, w_2)}{\partial w_1}}{\frac{\partial W(w_1, w_2)}{\partial w_2}} = \frac{m_1 w_2}{m_2 w_1} = \frac{\tau_2}{\tau_1} \quad (4)$$

If MRS<1, for example, then the agent thinks of itself as being relatively poor in spice. If $MRS_A > MRS_B$, then agent A considers sugar to be relatively more valuable than does agent B, and so A is a sugar buyer and a spice seller, while agent B is the opposite. As long as the MRSs are not the same there is potential for trade. The directions of trade are summarized in table 2:

	$\boldsymbol{MRS_A > MRS_B}$		$\boldsymbol{MRS_A < MRS_B}$	
Action	***A***	***B***	***A***	***B***
Buys	***Sugar***	***Spice***	***Spice***	***Sugar***
Sells	***Spice***	***Sugar***	***Sugar***	***Spice***

Table 2: Relative MRSs (marginal ate of substitution) and the directions of resource exchange.

The bargaining rule to determine the local price is:

$$p(MRS_A, MRS_B) = \sqrt{MRS_A MRS_B} \quad (5)$$

Agent trade rule T:

— Agent and neighbor compute their MRSs; if these are equal then end, else continue.

— The direction of exchange is as follows: spice flows from the agent with the higher MRS to the agent with the lower MRS while sugar goes in the opposite direction.

— Calculate price according to (5).

— The quantities to be exchanged are as follows: if p>1 then p units of spice for 1 unit of sugar; if p<1 then 1/p units of sugar for 1 unit of spice.

— If this trade will (a) make both agents better off (increases the welfare of both agents), and (b) not cause the agents' MRSs to cross over one another, then the trade is made and return to start, else end.

Markets of bilateral traders

Assume that we start with a population of 200 immortal agents, with the welfare function (3), behavioral rules M and T, uniform distributions of metabolism for sugar and spice (between 1 and 5), and initial endowments randomly distributed between 25 and 50, for both sugar and spice. Figure 12 shows the time series for the average trade price with vision set to 1, figure 13 the logarithm of the standard deviation (SD) for vision set to 1, figure 14, the SD for vision randomly distributed between 1 and 15. As we can see, though there is still a lot of variation in price, the standard deviation reaches a relatively low value. As one would expect, with higher mean vision, this value gets lower.

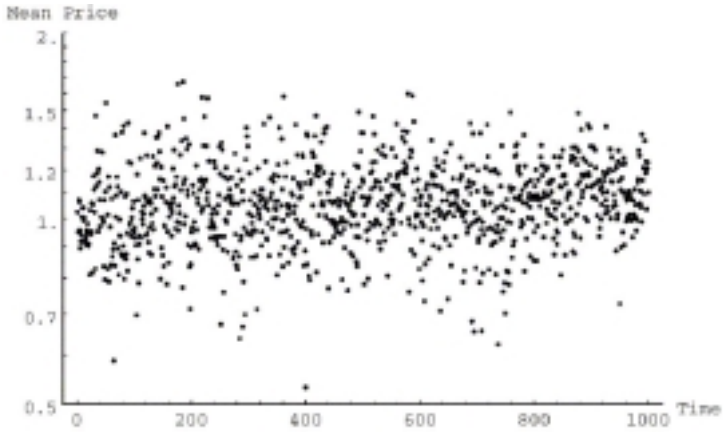


Figure 12: Typical time series for average trade price with agent vision set to 1.

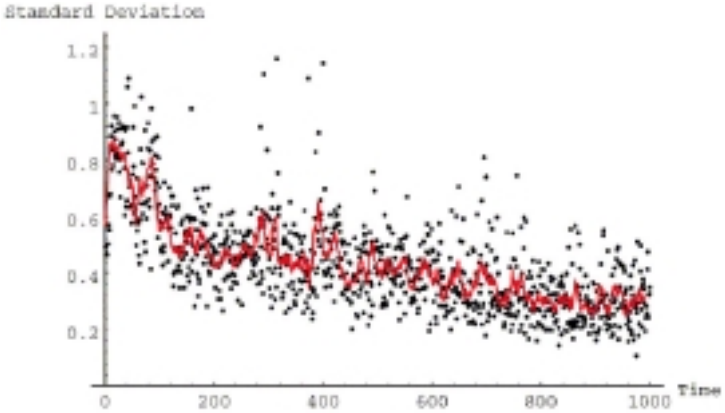


Figure 13: Typical time series for the standard deviation of the logarithm of average trade price with vision set to 1.

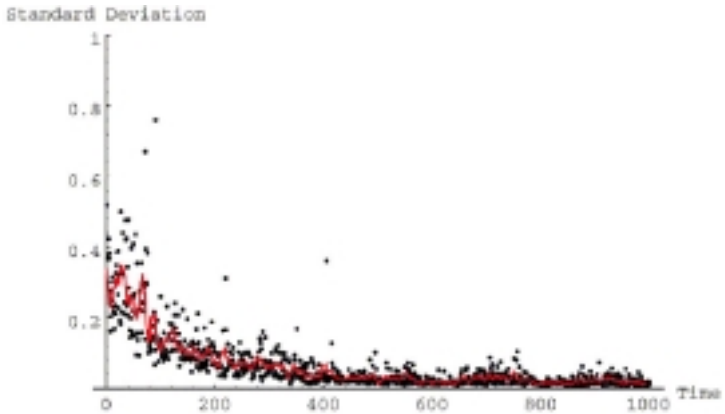


Figure 14: Typical time series for the standard deviation of the logarithm of average trade price with vision randomly distributed between 1 and 15.

If we now decide to have mortal agents, $R_{[a,b]}$, where the age of the agent is uniformly distributed between a and b, we get the time series of figures 15 and 16.

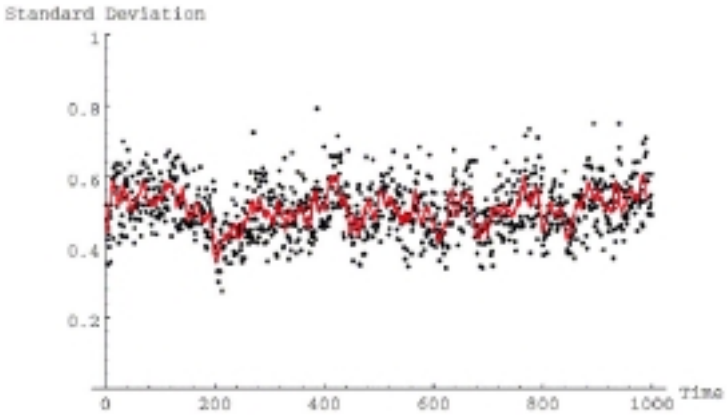


Figure 15: Typical time series for the standard deviation in the logarithm of average trade price under rule system ({G1}, {M, $R_{[60,100]}$, T) (with vision randomly distributed between 1 and 5).

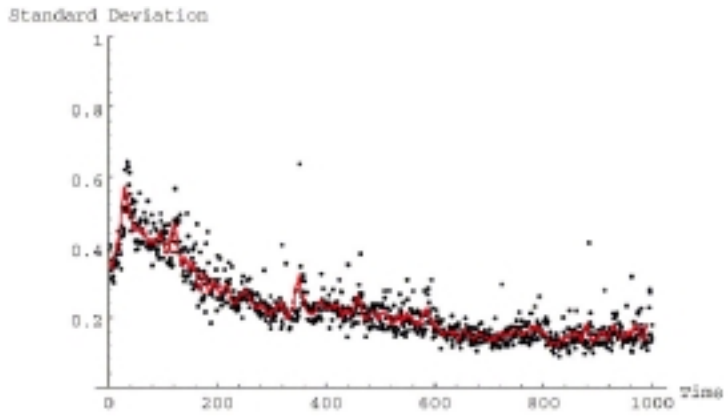


Figure 16: Typical time series for the standard deviation in the logarithm of average trade price under rule system ({G1}, {M, $R_{[960,1000]}$, T).

Figure 15 demonstrates that the variance does not decrease over time. In other words, where trade price is concerned, this economy is *far from equilibrium*. The longer the lifetime the more the economy approaches equilibrium, with maximum values for immortal agents. These considerations suggest that the assumption of an economy near equilibrium may not always be justified.

Conclusion

The Sugarscape model illustrates the idea of a virtual laboratory in which virtual societies can be created and examined. Although we have to be careful with our claims about the relation of such virtual worlds to real worlds, we can conveniently investigate the effects of assumptions that are made. Examples are assumptions about vision, metabolism, welfare function, and lifetime of agents. This type of computational study opens up new ways of doing science, which makes it possible to study new issues. An example is the relation between biological evolution and social factors, like legislation. Remember the effect of inheritance on vision.

5.2 Emergence of structure in societies of artificial animals

Another illustration of virtual laboratories is inspired by primatology. Charlotte Hemelrijk has investigated the emergence of structure in societies of primates in the real world and in simulation. In her simulations she has been able to show that spatial distributions and hierarchical structures can emerge from the local interactions of agents. There is no need to postulate a representation of the hierarchical structure in the individuals' brains (Hemelrijk, 1998a, 1998b, 1999).

The emergence of hierarchies in societies of artificial chimps

Primates are known for their high cognitive capacities, which are thought being manifested in their social behavior, in particular their formation of coalitions. Coalitions are a part of their dominance interactions. Dominance interactions consist of threats and attacks that usually take place between two individuals only. Sometimes, however, a third individual intervenes by attacking one of the partners, thereby supporting the other. This is called coalition formation. The assumption is that primates are highly strategic in their decisions, for example, when they form coalitions and with whom they form them. They are even thought to repay received support. In order to be able to do so, they are presumed to keep records of the frequency of support received from every partner. Yet, in her individual-based computer simulations, Hemelrijk (1998a, 1998b) made a first step towards showing how complex patterns of coalition formation may emerge in the absence of sophisticated cognitive reflections. Inspired by a simulation by Hogeweg (1988) and Hogeweg and Hesper (1979), she implemented a world in which creatures—artificial chimps—dwelled. These creatures were able to move and to see each other. Furthermore, if creatures perceived someone nearby, they engaged into dominance interactions, otherwise they followed rules of moving and turning (figure 5.17) that kept them aggregated (because real primates are group-living).

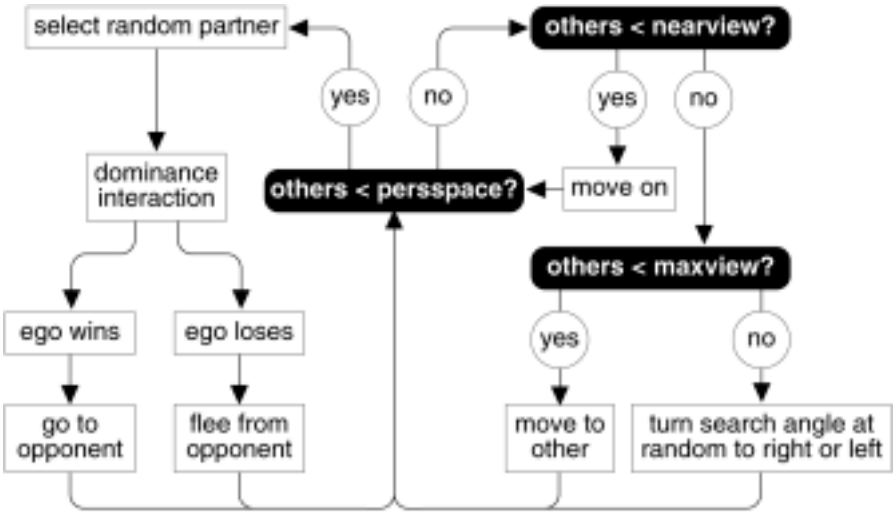


Figure 5.17: Flow chart for the behavioral rules of the artificial chimps designed by Hemelrijk (1998a, 1998b). The left side of the figure contains dominance rules: after winning, Ego approaches the opponent, after losing it flees from it. The right side concerns aggregation rules: creatures look for others at increasingly larger distances. If they see nobody at all, they turn over a search angle to search for others.

Note that interactions among these artificial chimps are just triggered by the proximity of others not by record keeping or other strategic considerations. Creatures were not even endowed with rules to support others in fights. Yet, support was recorded as an emergent event. It occurred if creatures happened to attack others that appeared to be already involved in a dominance interaction with someone else. Dominance interactions in the model incorporated the so-called 'loser- winner effect'. This effect has been established in many animal species, such as insects, reptiles, birds, mammals and humans. It implies that the effects of losing (and winning) are self-reinforcing. This means that after losing a fight the chance to loose the next fight is larger (even if the opponent is weak). The winner effect is the converse. By running the model, several forms of emergent social behavior were noted. This is shown in figure 5.18.

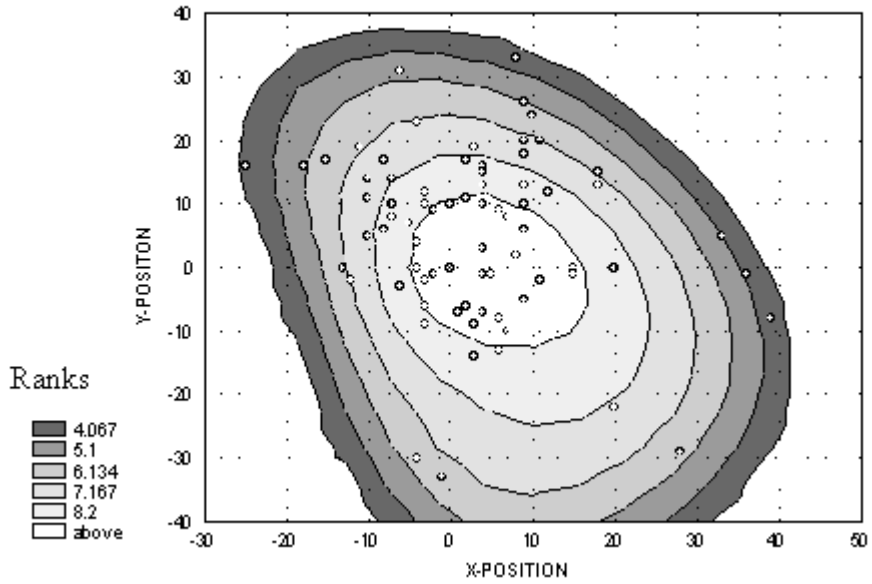


Figure 5.18: Emergent hierarchies in artificial chimps. Spatial-social structure with concentric rings of chimps of different rank categories. The outer rings are occupied by lower ranking creatures.

A dominance hierarchy arose, and a social-spatial structure, with dominants in the center and subordinates at the periphery (figure 5.18). Remarkably, exactly this same social-spatial structure has been described for several primate species. Furthermore, support in fights appeared to be repaid, despite the absence of a motivation to support or keep records of them. This was a consequence of the occurrence of a series of cooperation that consisted of two creatures alternatively supporting each other to chase away a third. These originated because by fleeing from the attack range of one opponent the victim ended up in the attack range of the other opponent. This typically ended when the spatial structure had changed such that one of both cooperators attacked the other. In particular these series were observed in loose groups, because entities were less disturbed and distracted by others. Additionally, chimps that were more aggressive appeared to

co-operate more, due to their longer attack range. These were more easily trotted on by others and more difficult to get away from, resulting in series of repeated attacks.

Thus, the model shows how complex social interaction patterns may arise from local interactions only. It follows that this may also apply to real animals and that interaction patterns need not be genetically or cognitively predefined in the individuals' brains. Furthermore, the model points to some new questions concerning real primates, which would not be asked, if social behavior would be approach from a cognitive perspective only. An example would be: Is cooperation (such as repayment of support) more general in loose than cohesive groups and more prevalent among strongly than mildly aggressive animals? For additional questions, see Hemelrijk's publications.

A note on terminology

Recall the examples discussed in chapter 3. The kinds of phenomena demonstrated in the Didabot experiments and the ants are examples of *self-organization without structural changes*: the Didabots, for example, did not change during the heap building process. If we start a new experiment with the blocks randomly distributed in the environment, the Didabots would behave exactly the same way as in the previous trial. They would try to avoid obstacles. What we observe in Hemelrijk's simulations is *self-organization with structural changes:* The internal state of the individuals, i.e. their dominance value, changes over time, which in turn changes how the agents interact with other agents. Such processes of self-organization with structural changes are especially relevant, for example, during brain development.

5.3 Schelling's segregation model

In the models described above, emergent phenomena are the result of agent and environment rules (in the case of Sugarscape) or a consequence of dominance interactions (in Hemelrijk's model). In Hemelrijk's models, the agents displayed certain clustering patterns. Such clustering phenomena may also arise due to social avoidance of and preference for certain others, i.e. if the agent rules include factors pertaining to social preference. The Harvard economist, Thomas Schelling (Schelling 1969), developed a similar sociological model in the late '70s. But again, what is observed does not appear to reflect the preferences of a single individual. Using a model of individuals that prefer to be surrounded by a certain minimum percentage of similar individuals, Schelling was able to show, how social avoidance of a minority status, even if slight, appears to be amplified at the level of the group and community structure. He notes that "micro motives" of individuals may lead to unexpected "macro patterns" that are not necessarily desired by any of the individuals, such as in ghetto formation and segregation of the sexes at parties.

Epstein and Axtell present the following somewhat modified version of Schelling's model. They use a 50 x 50 lattice world in the form of a torus. A torus in this context means that the points on the left and right edges of the lattice are considered neighbors (the same holds for the points on the top and on the bottom). 500 cells remain empty and 2000 cells are filled with an about equal number of red and of blue agents.

Agents are steered by the following behavioral rule, *Schelling's agent movement rule*:

— Agents perceive a von Neumann neighborhood of 4 cells.

— At each time step, an agent computes the fraction of neighbors of its own color.

— If this fraction greater than or equal to its own preference, the agent remains where it is.

— If the fraction is below its preference, the agent chooses an acceptable site in a random location.

At the start of each simulation run, agents are placed in random locations. During a run this simulation shows how at every time-step the dissatisfied individuals move. This goes on until everybody is happy. This appears to arise only when the segregation is much more extreme than reflected by the individual's preferences: Most individuals appear to be surrounded by many more individuals of the same type than they had wished for. This comes about because the movement of discontent agents may dissatisfy formerly content agents that just reached their preference limit. Consequently, the latter move too, and most agents will end up being surrounded by more of the same type than they strictly required. These results are compatible with Schelling's original model although he used a finite boundary, a Moore neighborhood of 8 cells and discontent agents selected the nearest acceptable site instead of a random one.

An entertaining description of this model is given also by Mitch Resnick (see pages 81-88, Resnick, 1997), where instead of males and females, the agents are turtles and frogs that dwell on a pond, sitting on pads of water lilies, much like Schelling's agents lived in grid points on a lattice. Resnick also shows how the model can be implemented in StarLogo.

5.4 Conclusion

We have seen that the term "agent-based" refers to a particular type of simulation model, which includes two essential components, agents and environment. An agent's behavior is determined by simple rules based on local interactions. The environment has certain autonomy, i.e. it has a certain level of independence from what the agents do, but it can also be influenced by the agents' behavior. The interaction of the agents among each other, as well as the interaction of the agents with their environment is modeled separately and independently from each other (contrary to more traditional kinds of simulation where often systems of differential equations are used). Agent-based models can be used in different areas of science. They are often easier to realize than time-consuming and expensive field studies and the results can be used in different research fields.

Bibliography

Casti, J.L. (1997). *Would-be worlds. How simulation is changing the frontiers of science*. New York: John Wiley and Sons.

Epstein, J.M., and Axtell, R. (1996). *Growing artificial societies. Social science from the bottom up*. Washington, D.C.: Brookings Institution Press; Cambridge, Mass.: MIT Press.

Hemelrijk, C.K. (1998a). Risk sensitive and ambiguity reducing dominance interactions in a virtual laboratory. In R. Pfeifer, B. Blumberg, J.-A. Meyer, and S.W. Wilson (eds.). *From Animals to Animats. Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*. Cambridge, Mass.: MIT Press, 255-262.

Hemelrijk, C.K. (1998b). Spatial centrality of dominants without positional preference. In C. Adami, R.K. Belew, H. Kitano, and C. Taylor (eds.). *Artificial Life VI. Proceedings of the Sixth International Conference on Artificial Life.* Cambridge, Mass.: MIT Press, 307-315.

Hemelrijk, C.K. (1999). An individual-oriented model of the emergence of despotic and egalitarian societies. *Proceedings of the Royal Societ. London B*, 266, 361-369.

Hogeweg, . (1988). MIRROR beyond MIRROR, Puddles of LIFE. *Artificial life, SFI studies in the sciences of complexity*. Redwood City, Ca.: Addison-Wesley, 297-316.

Hogeweg, P., and Hesper, B. (1979). Heterarchical, selfstructuring simulation systems: concepts and applications in biology. *Methodologies in systems modeling and simulation.* Amsterdam: North-Holland, 221-231.

Hogeweg, P., and Hesper, B. (1983). The ontogeny of interaction structure in bumblebee colonies: a MIRROR model. *Behav. Ecol. Sociobiol*., **12**, 271-283.

Resnick, M. (1997). *Turtles, termites, and traffic jams. Explorations in massively parallel microworlds.* Cambridge, Mass.: MIT Press.

Schelling, T. C. 1969 Models of segregation. *American Economic Review, Papers and Proceedings*, **59**, 488-493.

Chapter 6: Artificial Evolution

Artificial evolution is a classic topic of artificial life. Researchers in artificial evolution typically pursue one of the following goals: They are interested in understanding the principles of biological evolution; in the spirit of the synthetic methodology, the understanding of biological phenomena can be greatly enhanced by trying to mimic or model aspects of the biological system under examination. The other main goal is to use methods from artificial evolution as optimization tools, or more generally as design methods. In many areas, evolutionary algorithms have proved highly useful, especially for "hard" problems. In quite a number of areas they have produced solutions that humans could not easily have derived. Humans have their inescapable biases because they have grown up in a particular environments and societies. Evolutionary methods can be seen as ways to overcome these biases because computers only have those biases that have been explicitly programmed into them. A beautiful illustration of this point is the "tubing problem" that was described by Rechenberg in the 60s'. It is illustrated in figure 6.1.

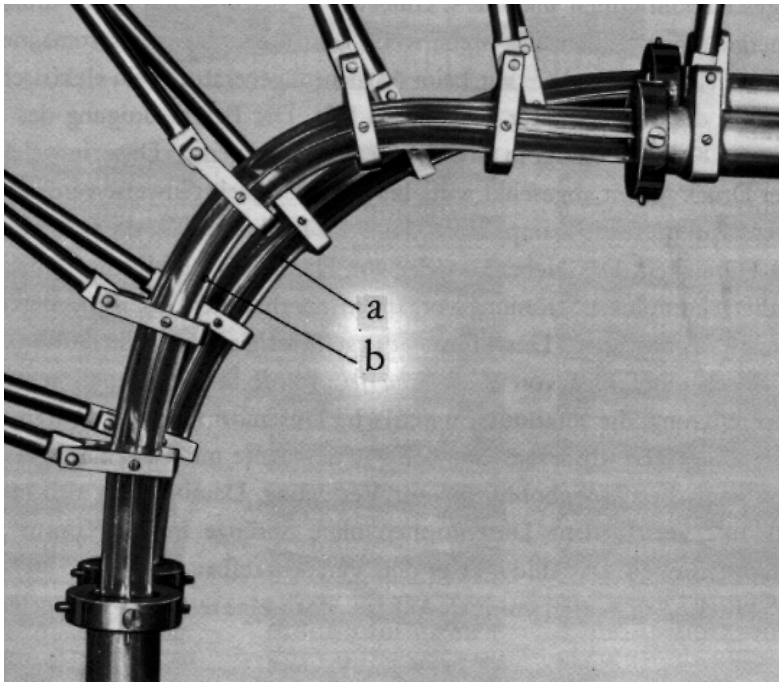


Figure 6.1: Rechenberg's "tubing problem". (a) the standard solution, and (b) the optimal solution.

The fuel flows into the tube from the left. The fuel should leave the tube (top right) such that the resistance of the fluid is minimized. The question is how the two tubes should be connected, i.e. what is the shape of the connecting tube? The standard answer of most people is a quarter circle. Using an evolutionary method called "evolution strategy" or ES - a technical term for a particular class of algorithms (see below) - Rechenberg (1994) could show, that the optimal solution for this problem, has a kind of "hunch" on the top, a solution that one would not easily think of. Apparently this "hunch" helps minimizing turbulence.

6.1 Introduction: Basic Principles

In chapters 1 through 5 we have seen many examples of emergence — of behavior and other patterns. The essential aspects have always been (i) there are many components in the system (cells, actors, grid-points), and (ii) there is interaction via local rules. In this chapter on artificial evolution we also work with systems containing many elements, these elements are called populations. Evolution only makes sense in the context of entire populations — it is not defined for individuals.

The field of artificial evolution started in the 1960s with developments of Ingo Rechenberg in Germany, of John Holland, and L.J. Fogel in the United States. Holland's breed of evolutionary algorithms is called *genetic algorithms* or GAs (DeJong, 1975; Goldberg, 1989, Holland, 1975, Mitchell, 1997), Fogel's *evolutionary programming* or EP (Fogel, 1962; Fogel, 1995), and Rechenberg's *evolution strategies* or ESs (Rechenberg, 1973; Schwefel, 1975). Holland was interested in adaptation in natural systems, Fogel and Rechenberg more in exploiting evolutionary algorithms for optimization. They all share a strong belief in the power of evolution. While from an algorithmic perspective there are important differences between these three types of procedures, for our purposes these differences are not essential. For a comparison, the interested reader is referred to Bäck and Schwefel (1993).

There is a vast literature on evolutionary algorithms of various sorts. But for the better part, the principles upon which they are based are relatively similar and can be easily categorized. We start with an example, which helps introducing the basic concepts and then provide an overall scheme.

Example: Biomorphs

This description is based on "The blind watchmaker" by Richard Dawkins. Biomorphs are tree like structures used as a graphical representation of a number of simple genes. They are a fun way to demonstrate the power of evolution: random mutation followed by non-random selection. Remember the turtle graphics that we encountered in the context of Lindenmeyer systems. We will now use the same idea to illustrate artificial evolution.

Artificial evolution always starts from a genome, thus the problem to be solved must somehow be encoded in the genome. The "problem" in our case is simply to draw interesting and appealing figures. In the turtle graphics we could specify, for example, the length of the lines to be drawn, the angle the two lines make at a joint, and the depth of recursion. That would make a genome with 3 genes, the first and the second are real numbers, the third (the depth of recursion) is an integer. In fact, Dawkins used 9 genes, 4 for the length of the lines and 4 for the angles and also a recursion depth. In this way, more complicated shapes can be produced because not all lines have to be of the same length. Since the idea is to mimic (in some very abstract sense) cell division, and since cells always divide into two cells, the number of lines coming off a joint is always two. The drawing program can be seen as the process of ontogenetic development that transforms the genotype (the genetic material, i.e. the numbers contained in the genome) into the phenotype, i.e. the figure on paper. The drawings themselves are called Biomorphs by Dawkins, a term which was originally coined by zoologist and painter Desmond Morris for the vaguely animal-like shapes

in his surrealist paintings. (Desmond Morris claims that his Biomorphs "evolve" in his mind and that their evolution can be traced through successive paintings.)

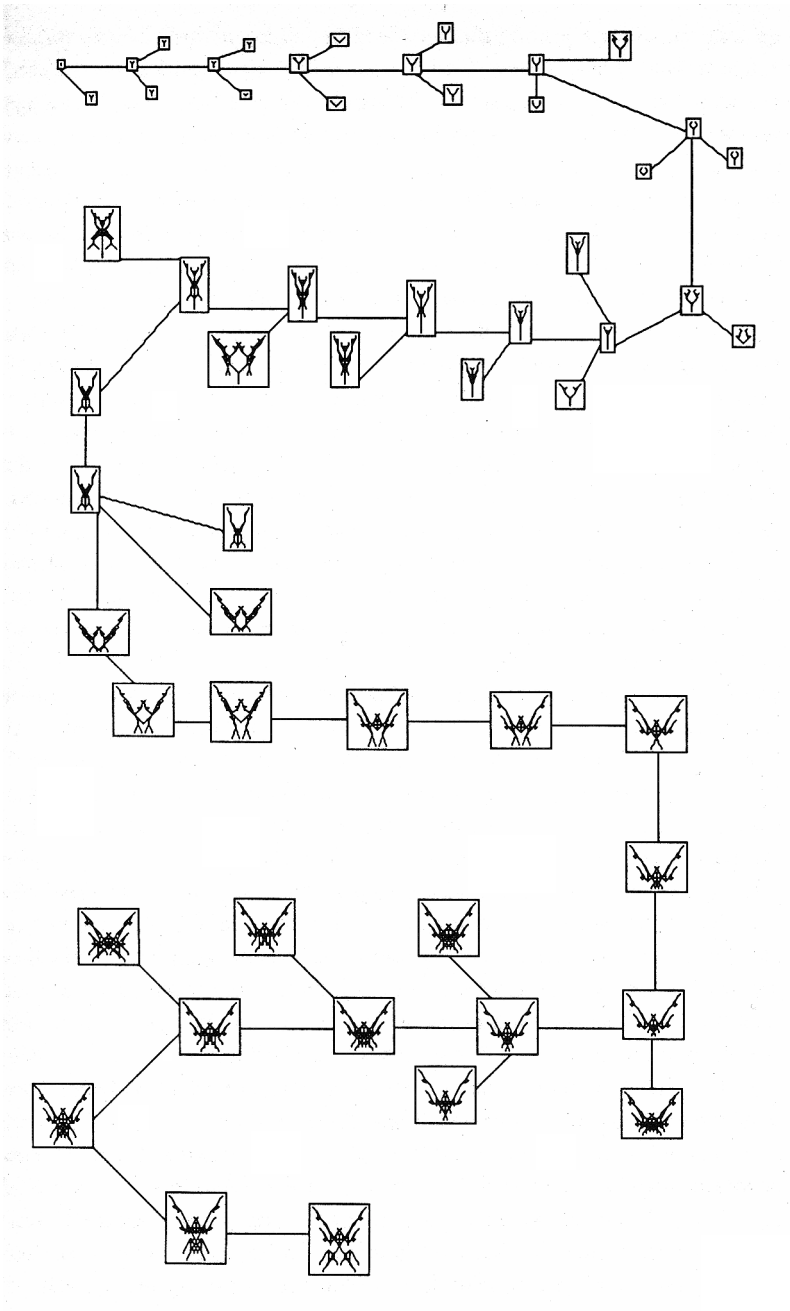


Figure 6.2: Starting from one Biomorph (represented by a dot) Dawkins evolves an insect-like individual after only 29 generations.

Initially, a set of numbers that constitute the genes are randomly generated (within certain limits). From these the genotypes, the phenotypes are produced by applying the drawing program (this is called DEVELOPMENT). The user selects from this set of Biomorphs the one that he or she likes most (this is called SELECTION). This Biomorph is then taken as the starting point for further evolution, and it is allowed to reproduce. In other words, its genotype is mutated and another set of Biomorphs is drawn, etc., until there is a Biomorph that the user likes. This type of evolution where the user of the program decides

which Biomorph is allowed to reproduce, is also called "animal breeder selection", since it resembles precisely what animal breeders do. Note the historical nature of this process: it contains, and in some sense accumulates, the user's decisions. This is why the term "cumulative selection" is sometimes used. The genome is changed through mutation, i.e. random changes in the numbers that represent the various features in the genome (like angles and length of segments).

A simulation starting from scratch, where the user can play "God" is shown in:

http://suhep.phy.syr.edu/courses/mirror/biomorph/

In addition to Dawkins's 9 genes, this demonstration includes 6 extra genes, yielding a total of 15 genes. Very briefly, the first 8 control the overall shape of the Biomorph: four are for encoding the angles on four subsequent steps and four are for the length of the segment to be drawn (similar to Dawkins); gene 9 encodes the depth of recursion (like Dawkins); genes 10, 11 and 12 encode the red, green and blue color components of the color in which the Biomorph is drawn. Gene 13 determines the number of segments in the Biomorph; Gene 14 controls the size of the separation of the segments; and gene 15 the primitive used to draw the Biomorph (line, oval, rectangle, filled oval or filled rectangle). The simulation takes off with dots, but after a few generations fascinating forms start to appear.

In the following simulation Biomorphs are generated at random by clicking the "zap" button. Through "select" the current Biomorph is selected and small random mutations of this Biomorph are displayed in the other windows. The principle is the same as in the first example.

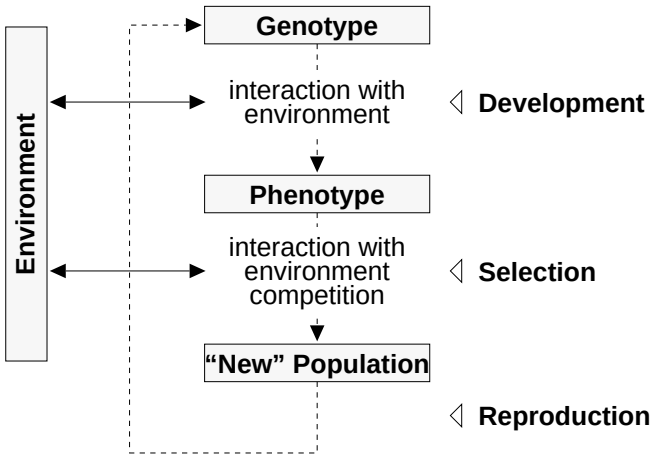
http://www.math.ruu.nl/people/beukers/dawkins/dawkins.html

The Basic Evolutionary Cycle

(adapted from Pfeifer and Scheier, 1999, chapter 8, p. 232, The Evolutionary Process)

Figure 6.3a gives an overview of the evolutionary process. Evolution always works with populations of individuals. In nature these are creatures, in artificial evolution they are often solutions to problems. Each individual agent carries a description of some of its features (color of its hair, eyes, skin, body size, limb size, shape of nose, head etc.). This description is called its genome. The term genotype refers to the set of genes contained in the genome. It is used to express the difference between the genetic setup and the final organism, the phenotype. The genome consists of a number of genes, where in the simplest case one gene describes one feature. Genes are identified by their position within the genome. If the individual members of the population to be investigated are of the same species, they all have the same numbers of genes and the genes are at the same location in the genome. But the values of the genes can differ. The values of all genes of an individual are determined before it starts to live and never changes during its life.

a.



b.

encoding	development	selection	reproduction
• binary • many-character • real-valued	• no development (phenotype = genotype) • development with and without interaction with the environment	• "roulette wheel" • elitism • rank selection • tournament • truncation • steady-state	• mutation • crossover

Figure 6.3: Overview of the process of evolution. (a) The main components. The genotype is translated into a phenotype through a process of development. The phenotypes compete with one another in their ecological niche, and the winners are selected (selection) to reproduce (reproduction), leading to new genotypes. (b) Evolutionary algorithms can be classified according to a number of dimensions: encoding scheme, nature of developmental process, selection method, and reproduction (genetic operators). Mitchell (1997, pp. 166-175) discusses the pros and cons of these various methods in detail.

Through a process of `development`, the genotype is translated into a phenotype. In this process, genes are expressed, i.e. they exert their influence on the phenotype, in various ways. The precise ways in which the genes are expressed are determined by the growing organism's interaction with the environment. In artificial evolution, the phenotype and the genotype are indeed often the same. If we are aware of the fact that we are dealing with algorithms, rather than with a model of natural evolution, there is absolutely nothing wrong with this. Then, the phenotype competes in its ecological niche for resources with other individuals of the same or other species. The competition in the Biomorph example consists in somehow trying to please the experimenter's sense of aesthetics. The winners of this competition are selected by the experimenter, which leads to a new population. The members of this new population have higher average fitness than in the previous one. The individuals in this new population can now reproduce. It is characteristic of evolutionary approaches that they work with populations of individuals rather than individuals only. There are many variations of how selection can be done; a list is provided in figure 6.3b

So far we have discussed only asexual reproduction where an individual duplicates only its own genotype, possibly with some small random mutation. In the Biomorph example, there was only one individual involved in reproduction, i.e. the reproduction was asexual. There is also sexual reproduction where two individuals exchange parts of their genotype to produce new genotypes for their offspring. The most

common type of sexual reproduction is called crossover (see below). It is often used in combination with mutation. Finally, there is a reproduction process. Reproduction, like selection, comes in many variations.

Development, selection, and reproduction are closed in themselves: they receive a certain input and deliver some output. For example, the `development` process receives a genotype as input and eventually produces a phenotype as output. But the phenotype cannot influence the genotype. Since we are dealing with algorithms, it would be no problem to have the phenotype influence the genotype. But that has not been systematically explored, presumably because that would correspond to a so-called Lamarckian position. According to Lamarck, learned properties of an organism can be genetically passed on to the offspring.

The scheme of Figure 6.3 can be used to classify different evolutionary approaches: some comprise all these components in non-trivial ways, some do not have development — in fact, most evolutionary algorithms do not — and some have development, but without interaction with the environment. Additional classification can be made according to the way the features are encoded in the genome, the type of selection, and the kind of reproduction performed. We will only provide a short review and give examples of the most common kinds. There are many excellent textbook that give systematic reviews of the various types of algorithms (e.g. Goldberg, 1989; Mitchell, 1997). Before we look at the different approaches in more detail let us consider some theoretical points.

Variations on Evolutionary Methods: Theoretical Issues

(from Pfeifer and Scheier, 1999, chapter 8, page 236 ff)

As we have already mentioned, there are many variations on evolutionary methods. Like neural networks, evolutionary algorithms are fascinating and seem to exert an inescapable attraction urging the user to play around and tinker with them. Here we present a few variations; for systematic reviews, see, for example, Mitchell 1997.

Encoding Scheme: The most widely used encoding scheme is what we have seen in our earlier example, binary encoding in terms of bit strings. A few others appear elsewhere in this script, such as many-character encoding, as in Eggenberger's Artificial Evolutionary System, or the graph structures used by Karl Sims (6.8 below). The rule here is to use whatever is best suited. Choice of encoding scheme is not a matter of religious devotion to one scheme over all others.

Development: As we mentioned, development is often entirely neglected --- it may not be necessary at all. In its absence, selection is performed directly on the genotype. There are also trivial forms of development in which the representation in the genome is directly mapped onto the organism's features without interaction with the environment. We have seen this in the example above. More complex, but still lacking interaction with the environment, is Sims' approach. One model that capitalizes on ontogenetic development is Eggenberger's Artificial Evolutionary System.

Selection: One gets the impression that researchers in the field have tried virtually any method of selection that even remotely promised to improve their algorithms' performance. All methods have their pros and cons --- discussing them in any depth is well beyond the scope of this chapter. One might be tempted

simply to take the best individuals and ignore the others. However, that would lead to a quick loss of diversity. Those individuals currently not doing as well as the others may have properties that will prove superior in the long run. Thus, a great deal of attention has been devoted to getting just the right mix of individuals. The problem is sometimes termed the *exploration-exploitation trade-off*. The goal is to search the space in the region of the good individuals but still to explore other regions, because the currently best may turn out to be only locally optimal.

Holland (1975) proposed using a method in which an individual's probability of being selected is proportional to its fitness. This is also called *roulette wheel selection*: Spin the roulette wheel and select the individual where it stops. The size of the segment of the roulette wheel for an individual is proportional to its fitness. *Elitism* is often added to various schemes, meaning that the best individuals are copied automatically to the next generation. In *rank selection*, individuals are chosen with a probability corresponding to their rank (in terms of fitness), rather than their actual fitness value. *Tournament selection* is based on a series of comparisons of two individuals: through some random procedure that takes their fitness into account, one individual "wins" and is selected for reproduction. Finally, there is a distinction between *generational* and *steady state selection*. As mentioned above, rather than producing an entirely new population at the same time, in steady-state selection, only a small part of the population changes at any particular time, while the rest is preserved.

Reproduction: The most-often-used genetic operators are mutation and crossover. We have seen both in the example above. Although evolutionary methods are easy to program and play around with, their behavior is difficult to understand. It is still subject to debate how they work and what the best strategies for reproduction and selection are. Let us turn to natural evolution for a moment. Once good partial solutions have been found for certain problems, they are kept around and are combined with other good solutions. Examples are eyes and visual systems: Once they had been "invented", they were kept around and perhaps slightly improved. In evolutionary algorithms, there is a similar idea of good "building blocks" that are combined to increasingly better solutions.

Crossover is designed to combine partial solutions into complete ones with high fitness. There are a number of conjectures about why crossover leads to fast convergence while maintaining a high chance of reaching the global optimum. One is the schema theorem and, related to it, the building block hypothesis (e.g., Goldberg 1989). Schemas are particular patterns of genes that, depending on the algorithm chosen, proliferate in the population. The details need not concern us here; there is an ongoing debate as to the relevance of this theorem to evolutionary methods. The related topic of how useful crossover really is and how it contributes to resolving this trade-off is also still subject to debate (e.g., Srinivas and Patnaik 1994).

The preceding discussion can best be summarized as follows: There is no one best encoding scheme, selection strategy, or genetic operator. It is all a question of having the right balance suited for the particular issues one intends to investigate.

6.2 Different Approaches (GAs, ES, GP)

The most frequent approaches in the field of artificial evolution are the Genetic Algorithms (GAs), the Evolution Strategy (ES), and Genetic Programming (GP) (also called Evolutionary Programming, EP) as briefly mentioned above. We start our discussion with the most popular version, the GAs. John Holland in the United States who was interested mostly in models of biological evolution invented them.

Genetic Algorithms

Genetic algorithms follow a particular scheme, which is common to all. It is illustrated in figure 6.4.

Basic algorithm:

- Initialize population P

- Repeat for some length of time

    • Create an empty population, P'

    • Repeat until P' is full:

        - Select two individuals from P based on some fitness criterion

        - Optionally "mate", and replace with offspring

        - Optionally mutate the individuals

        - Add the two individuals to P'

    • Let P now be equal to P'

Figure 6.4: Basic scheme of genetic algorithm (from Flake, 1998).

Typically, the initial population P is generated randomly. Two individuals are selected on the basis of their fitness using a selection strategy, e.g., roulette wheel selection. Then a genetic operator, e.g., crossover, is applied, creating two offspring, which replace the parents. Almost always, there is mutation. *Mutation* is performed with a certain probability. Figure 6.5 below illustrates selection and reproduction. If this procedure is applied, the subsequent population has a higher average fitness than the previous one, which is, of course, the purpose of the optimization algorithm. Crossover, mutation, and various selection strategies will be discussed below. Note that normally in GAs there is no process of development: the genotype is identical to the phenotype and the selection is performed directly on the genotype. This means that fitness evaluations have to take place on the basis of the genotype, which is biologically speaking, unrealistic.

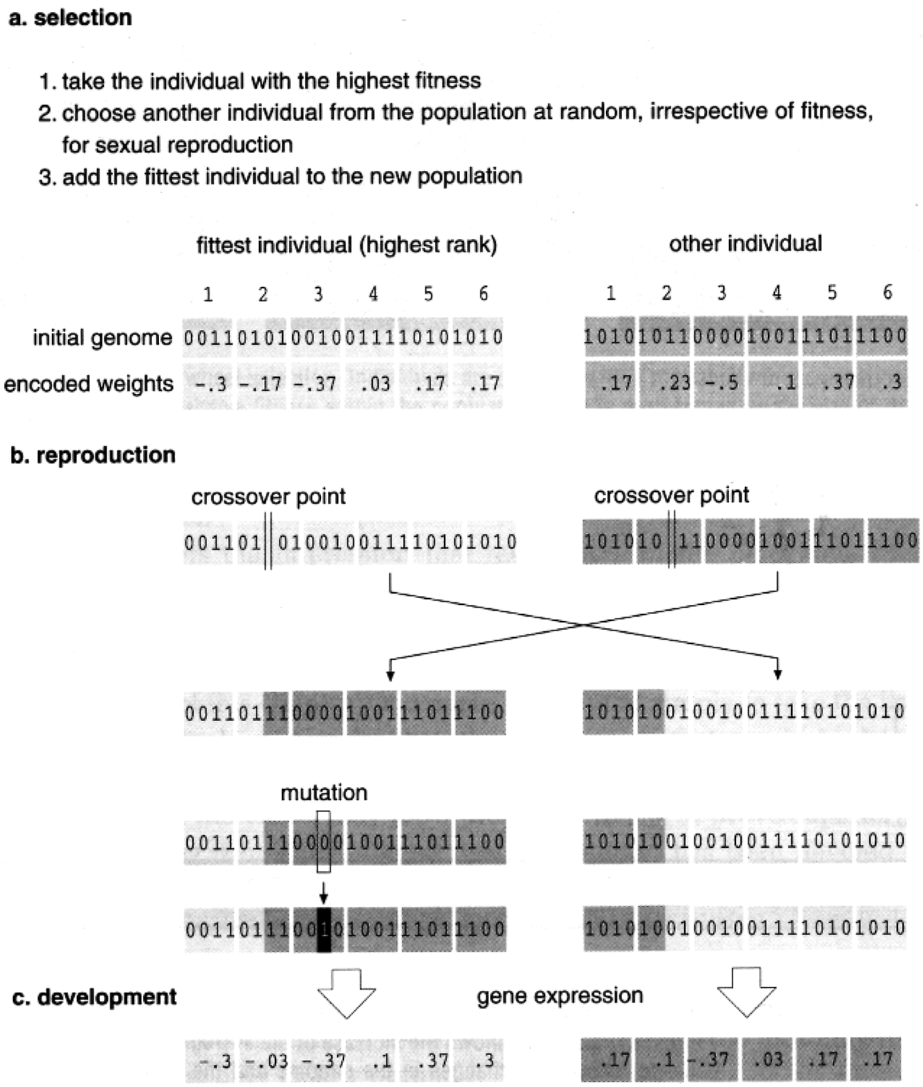


Figure 6.5: Selection and reproduction. a) Selection: After their final fitness values have been determined, individuals are selected for reproduction. b) Reproduction: The crossover point is chosen at random. The entire population is subject to a small mutation: c) Development: After reproduction, the new genome is expressed to become the new individual.

Example: The iterated prisoner's dilemma

The following description of the iterated prisoner's dilemma is based on Mitchell, 1997.

The prisoner's dilemma has been studied extensively in game theory, economics, and political science, because it can be seen as an idealized model for real-world phenomena such as arms races. It can be formulated as follows: two individuals, Alice and Bob, are arrested for having committed a crime together and are being held in two separate cells, with no possibility of communication. Alice is offered the following deal: If she confesses and agrees to testify against Bob, she will receive a suspended sentence with probation (German: bedingt mit Bewährung), and Bob will be put in jail for 5 years. However, if at the same time, Bob confesses and agrees to testify against Alice, her testimony will be discredited, and each will receive 4 years of prison for pleading guilty. Alice is told that Bob is being offered precisely the same

deal. Both Alice and Bob know that if neither testifies against the other, they can be convicted only on a lesser charge for which they will have to spend only 2 years in jail.

Now, what should Alice do? Should she "defect" against Bob (i.e. should she confess and testify against Bob) and hope for the suspended sentence, risking a 4-year sentence if Bob also defects? Or should she "cooperate" with Bob (i.e. should she deny and not testify against Bob), hoping that Bob will also cooperate, in which case both will get only 2 years in prison. However, if Bob does not cooperate but defects, Alice will get 5 years and Bob nothing (i.e. only a suspended sentence with probation).

The game can be described more abstractly. Each player independently decides which move to make, i.e. whether to cooperate or defect. A "game" consists of each player making a decision (a "move"). The possible results of a single game are summarized in a payoff matrix, as shown in figure 6.5:

	Player B		
player A		Cooperate	Defect
	cooperate	3, 3	0, 5
	defect	5, 0	1, 1

Figure 6.6: Payoff matrix for the prisoner's dilemma game (points representing reduction in years to be spent in jail).

Here, the goal is to get as many points as possible (few years in prison would correspond to many points). For example, in the figure, the payoff in each case can be interpreted as 5 minus the number of years in prison. If both players cooperate, each gets 3 points (corresponding to the 2 years in prison). If player A defects and player B cooperates, then player A gets 5 points (suspended sentence with probation) and player B gets 0 points (5 years in prison), and vice versa if the situation is reversed. If both players defect, each gets 1 point. What is the best strategy to use in order to maximize one's own payoff? If you suspect that your opponent is going to cooperate, then you should surely defect. If you suspect that your opponent is going to defect, then you should defect too. No matter what the other player does, it is always better to defect. The dilemma is that if both players defect each gets a worse score than if they cooperate. If the game is iterated, i.e. if the two players play several games in a row, both players' always defecting will lead to a much lower total payoff than the players would get if they cooperated. How can reciprocal cooperation be induced? This question takes on special significance when the notions of cooperating and defecting correspond to actions in, say, a real-world arms race (reducing or increasing one's arsenal).

Note that if the two players only play on single game, they have no information about the other's strategy. However, if they play several games and they have the information of the opponent's strategy in prior games, they have information on which they can base their decisions.

Encoding in genome

Whenever we want to solve a problem using a genetic algorithm we first have to define the encoding in the genotype. The encoding used here, is based on Axelrod (1987). Suppose that the memory of each player contains one previous game. There are four possibilities for the previous game:

- CC (case 1),
- CD (case 2),
- DC (case 3),
- DD (case 4)

where C denotes "cooperate" and D denotes "defect." Case 1 is where both players cooperated in the previous game, case 2 when player A cooperated and player B defected, and so on. A strategy is simply a rule that specifies an action in each of these cases. For example, TIT FOR TAT as played by player A is as follows:

- If CC (case 1), then C.
- If CD (case 2), then D.
- If DC (case 3), then C.
- If DD (case 4), then D.

If the cases are ordered in this way, the strategy can be expressed compactly by simply listing the right-hand sides, i.e. CDCD: Position 1 represents the answer to case 1 (i.e. C), position 2 the answer to case 2 (i.e. D), etc. It is a general principle in genetic algorithms to interpret the genes by position in the genome.

In Axelrod's tournaments the strategies were based on three remembered previous games. For example, CC DC DD means that player A played C in the first game, player B played C in the first game. In the second game, player A played D (i.e. he defected), while player B cooperated, and in the third game, both defected. In total there are 64 possibilities:

- CC CC CC (case 1),
- CC CC CD (case 2),
- CC CC DC (case 3),
- CC CC DD (case 4),
- …
- DD DD DC (case 63),
- DD DD DD (case 64).

A strategy in this case tells the player how to play, given the three games that the player remembers. In other words, for each case the player has to specify what will be the next move, i.e. C or D. Thus, a strategy can be encoded compactly as a string of 64 C's or D's, one for each case. Since for each case there are two possibilities, there are 2^{64} different strategies, i.e. 2^{64} strings of C's and D's (just think of 0 and 1, then this is like a string of binary digits). Actually, Axelrod used a 70-letter string: he also encoded what the players had played in the last three games (yielding 2^{70} different strategies, which is roughly 10^{21} - a very large

number indeed!). Which one is the best one? Exhaustive search is clearly no longer possible. So, we can try a genetic algorithm.

For every GA an initial population has to be specified. Axelrod used a population of 20 such randomly generated strategies. The fitness of each strategy in the population was determined as follows: Axelrod organized Prisoner's Dilemma tournaments. He solicited strategies from researchers in a number of disciplines. Each participant had to submit a computer program that implemented a particular strategy. These programs played the iterated Prisoner's dilemma with three remembered games. The tournament also contained a strategy that simply made random moves. Some of the strategies submitted were rather complicated and used sophisticated modeling techniques in order to determine the best move. However, it turned out that the winner (the strategy with the highest average score) was the simplest of the submitted strategies: TIT FOR TAT, a strategy that we have encountered before in the context of agent-based simulation. This strategy cooperates in the first game and then, in subsequent games, does whatever the other player did in its move in the previous game. In other words, it offers cooperation and reciprocates it. But if the other player defects, he will be punished until he begins to cooperate again.

Axelrod found that 8 of these strategies were sufficient to predict the performance of a strategy on all 63 entries. This set of 8 strategies served as the "environment" for the evolving strategies in the population. Each individual in the population played iterated games with each of the 8 fixed strategies, and the individual's fitness was taken to be its average score (of these eight games).

The results were as follows: Often the TIT FOR TAT strategy was evolved. The problem with this result is that it is based on the assumption of a stable environment. But individuals would not maintain the same strategy if they see what strategies the others use. In other words, the environment, which consists of the strategies of other individuals, will necessarily change. If we make this much more realistic assumption, the conclusions are less straightforward.

Evolution Strategy

The concept of Evolution Strategy (ES) was developed back in the '60s by Bienert, Rechenberg and Schwefel. It merely concentrates on the usage of the fundamental mechanisms of biological evolution for technical optimization problems. In the beginning ES was not a specific algorithm to be used with computers but a method to find optimal parameter settings in laboratory experiments. A simple algorithmic method based on random changes of experimental setups was used to decide how the parameter settings should be changed. Adjustment took place in discrete steps only and was not population based. Thus the first and simplest evolution process is a $(1+1)$-ES, where one parent and one offspring are evaluated and compared. Subsequently the one that is more fit becomes the parent of the next generation. ES imitates, in contrast to genetic algorithms, the effects of genetic procedures on the phenotype, i.e. on the object to be optimized (e.g. the tube shown in fig. 6.1).

Fitness always depends on the solution an individual offers to a certain problem in the respective problem-specific environment. In order to measure fitness, characteristic data of an individual have to be collected and evaluated. The respective parameters are then optimized in an evolution-based process. In the

beginning these parameters were represented by integer variables. Nowadays with computer implementations the parameters are arranged in vectors of real numbers on which operators for recombination (cross-over) and mutation are defined. The data-structure of a single individual is realized in the form of two vectors of real numbers. One is called object-parameter, the other one strategy-parameter. While object-parameters contain the variables to be optimized, strategy parameters control mutation of the object-parameters. The fact that the optimization takes place on both, the object-parameters as well as the strategy-parameters is seen as one of the major qualities of ES (often called self-adaptation).

There are three main criteria used to characterize an ES. They are: size of population, number of offspring in each generation and whether the new population is selected from both, parents and offspring, or just from offspring only.

Based on these criteria there are basically four different types of evolution processes. They differ in how the parents for the next generation are selected and whether to or not recombination is used. The four types are (μ,λ)-ES, $(\mu+\lambda)$-ES, $(\mu/\rho, \lambda)$-ES, $(\mu/\rho+\lambda)$-ES. They characterize ES with increasing level of imitation of biological evolution. The letter μ stands for the total number of parents, ρ marks the number of parents, which will be recombined, and λ stands for the number of offspring. Only in the two latter cases offspring are "produced" using mutation and recombination.

The two notations (..+λ) and (.., λ) describe the selection mechanism, in $(\mu+\lambda)$-ES and $(\mu/\rho+\lambda)$-ES the fittest μ individuals out of the union of parents and offspring are selected (so-called plus notation), in (μ,λ)-ES and $(\mu/\rho, \lambda)$-ES only the fittest μ offspring form the next generation of parents (comma notation). Thus in the latter case λ has to be greater than μ.

The first and simplest evolution process is a *(1+1)*-ES used in laboratory experiments. The parent individual "produces" one offspring by mutation (random change). The offspring is assigned a fitness value. If the offspring proves to be better than its parent it becomes the next generation's parent otherwise the parent stays the same. This ES does not incorporate a population typical for evolution but only one parent and one offspring.

As mentioned above, ES is especially suited for real-world optimization problems because typically they involve real numbers. ES works with fewer individuals, and all dimensions are mutated simultaneously. Furthermore the range of the mutation itself is subjected to the evolutionary process.

An example of how the evolution strategy can be applied to a real-world problem is how to find the optimal shape of a lens. The shape of the lens has to be appropriately parameterized (encoded in the genome) for the evolution strategy to work. An animation can be found at the following Web site:

http://lautaro.fb10.tu-berlin.de/user/michael/english/lens/lens.html

Genetic Programming

In genetic programming (GP) the goal is to evolve entire computer programs. The first and most straightforward approach would be — as in the standard GA approach — to directly encode a program into a fixed-length bit-string. This would be easy to do, since programs are represented as bit-strings anyhow. However, this would not be suitable mainly for the following two reasons.

First, most bit-strings do not represent valid computer programs. So, assuming, for example, that before mutation, there is in fact a bit-string that does represent a valid program, random mutation and crossover would almost certainly lead to non-functioning programs. If a program is non-functioning, it has zero fitness (or no fitness) and it does not deliver any feedback on the progress of evolution. Thus, it is important that the genetic operators always yield running programs. These programs may often not do much useful work, but since they are running programs, their fitness can be calculated and they influence the evolutionary process. One could object by saying that this is not really a problem, since any non-valid individual could simply be rejected by testing for its validity before accepting the mutation. The GP algorithm would then merely apply its genetic operators (i.e. mutation and recombination/crossover) until it has generated a sufficient number of valid offspring. But for sure such an approach would be extremely inefficient.

Second, genomes as we have discussed so far are limited to a fixed length. Therefore, in a fixed bit-string representation that length would limit the maximum potential complexity of the program. In order to alleviate this problem, the designer might simply specify a very long genome. However, this would significantly increase the computational complexity of the GP algorithm and therefore slow down the rate of progress. A similar problem is encountered when evolving neural networks. Typically, the weights of the network are encoded in the genome and thus the length of the genome determines the maximum size of the neural network (see also below).

In order to cope with these two problems, GP represents a program in a tree structure consisting of a root node, links to further nodes, which can in turn have links to yet further nodes, and so forth, recursively. If the nodes have no further links to other nodes, they are called leaves. In such a tree structure nodes represent processor operations, the instructions such as add, multiply, and divide, whereas the links represent their parameters, i.e. program data. In this scenario, leaves represent parameters and since they are terminals, they encode either constants, variables, registers, or external inputs.

Trees can be conveniently represented as list structures, which are linear strings of symbols. This is how programs are represented in the programming language LISP. The tree structure shown in upper-left part of Figure 6.4 corresponds to add(mul(r1,3),sin(r2,v1)) in linear representation [or (add (mul r1 3) (sin r2 v1)) in LISP notation]. This is a convenient representation on top of which operations can be defined which lead to other tree structures representing valid programs.

In the first design step, the designer decides on the set of operators, e.g. mul, add, div, pow, etc., the set of possible constants, e.g. +1, -1, +2, -2, and so on, and the set of possible variables, registers as well as inputs. Then, the mutation operator has to be defined in order to ensure that the outcome is again a tree structure representing a valid program. The easiest option is to replace any constant, register, variable, or

input by another, and similarly to replace operators by valid operators. When exchanging operators by other operators, care must be taken if the operators require a different number of parameters. Possible options are to simply discard parameters (in the case of fewer parameters) or to add neutral constants (in the case of more parameters).

When applied to tree structures, the recombination operators work as follows. As usual, the algorithm first selects two individuals. Then, the recombination operator selects a node or leaf of each individual's genome. Finally, the operator exchanges the entire sub-trees that are associated with each selected node or leaf respectively. It might be interesting to note that the recombination operator does not have any problem with the numbers of parameters of an operator, since each selected node or leaf originates from one ancestor and thus it will automatically have the correct number of parameters.

Let us illustrate the GP algorithm with the example presented in figure 6.7. Here, we have two genomes with the list representations x1=add(mul(r1,3),sin(r2,v1)) and x2=add(add(v1,sin(4,v2)),-2). First, the selected parents are copied. Then, the algorithm applies the recombination operator at the two randomly chosen positions (as indicated in figure 6.5) leading to two new offspring with the linear representations add(mul(r1,3),sin(r2,add(v1,sin(4,v1)))) and add(v1,-2). After applying mutation, the first genome changes to add(mul(r1,3),sin(r2,add(v1,sin(4,v3)))).

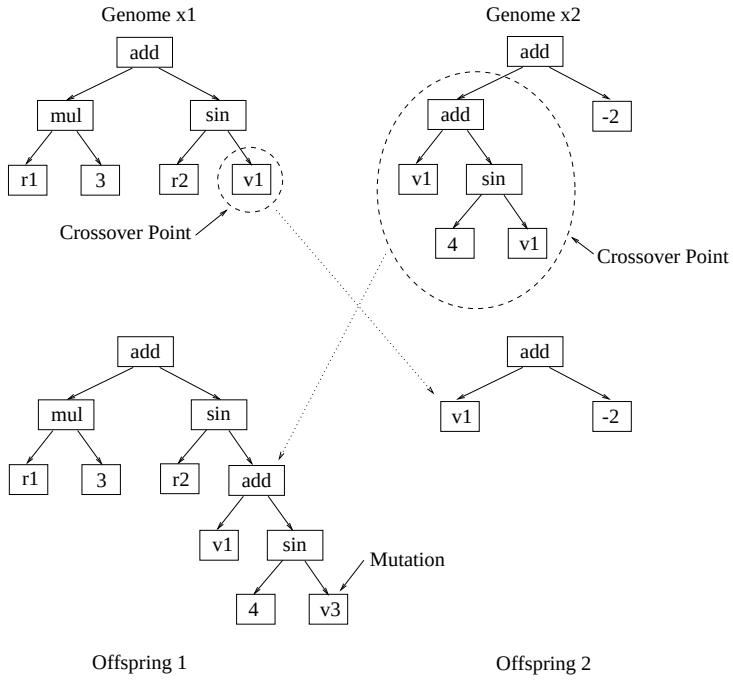


Figure 6.7: Illustration of crossover and mutation on trees.

By exchanging complete sub-trees, the involved individuals are allowed to grow and shrink. The result (the grown or the shrunk individual) is passed on to the selection process that does its work based on fitness evaluation. Growing of trees (genomes) is a fundamental problem in GP, since the search space may grow dramatically. In order to deal with this problem, the designer normally adds an additional term to the fitness function. Let us assume that we have a fitness function f(x). Then, the augmented fitness function f'(x) is defined as f'(x) = f(x) + c*l(x), with l(x) representing the genome's "length." This length might be, for

example, the number of nodes including the leaves. The main strategy is that a shorter genome ‘x1’ has a better fitness f'(x1) than a longer genome ‘x2’, even though both genomes have identical fitness values in terms of the simple fitness function f(x), i.e. f(x1) = f(x2). The constant *c* plays a critical role in the augmented fitness function f'(x). The designer has to find a good compromise between the genome length and the actual fitness values f(x).

This short example can only provide the basic idea of how the genetic operators can be defined to yield valid programs. The story of GP is more complex than described here — we cannot do full justice to it in such a short summary. We simply wanted to introduce the basic ideas: Applying evolution algorithms to the automatic generation of computer programs, where the individuals have variable length genomes, which can grow and shrink. This feature endows GPs with enormous power, the power to evolve any thinkable machine learning system (e.g. Banzhaf et al., 1998).

It should be noted that the applicability of GPs is by no means restricted to the evolution of computer programs. GP can be applied to any problem that can be reasonably represented by tree structures, a very large class of problems indeed. Thus, applications abound not only in the field of computer programming (acyclic graph evaluation, caching algorithms, sorting algorithms, computer animation, computer security, data compression, etc.), but also in biochemistry, in data mining in general, control of autonomous agents, pattern recognition, signal processing, electronic circuit design etc. For a more comprehensive list of applications, see Banzhaf et al. pp. 342-345.

6.3 Morphogenesis

Often in the field of genetic algorithms, as mentioned earlier, there is no process of development: the phenotype is taken to be identical to the genotype as in the iterated prisoner’s dilemma. In this section we extend this view by discussing mainly two approaches: Karl Sims’s virtual creatures and Peter Eggenberger’s “Artificial Evolutionary System.”

Karl Sims’s Virtual Creatures

(from Pfeifer and Scheier, 1999, chapter 8, page 245 ff)

Let us now look at a complex example, Karl Sims' virtual creatures (1994a, 1994b). These creatures hold an inescapable fascination---and they are a lot of fun. A number of factors underlie this fascination: First, Sims evolves morphology and neural control. This relieves the human designer of having to come up with a fixed design for the entire low-level specification: The designer commitments are pushed one step further back. We examine these below. And second, Sims was one of the first to use a 3-D world of simulated physics in the context of virtual reality applications. Simulating physics includes considerations of gravity, friction, collision detection, collision response, and viscous fluid effects (e.g. in simulated water).

The details of such simulations are not essential for our purposes; what is essential is that the creatures must perform and compete against each other in this virtual world. As we will see, evolution generates some fascinating morphologies for agents that occupy these virtual worlds, and because of the simulated physics, these agents interact in many unexpected ways with the environment.

In developing his agents, Sims needed to specify the representation of the genotype, the process of development, the selection process, and the reproduction strategy.

Representation of the Genotype

Again, we apply the scheme of figure 6.3 to describe Sims' approach. The genotype specifies a directed graph, which is considerably more complex than the bit strings used in the standard evolutionary algorithm. The genotype specifies how the phenotype has to be generated. This requires an interpreter for the genotype that knows what the specifications mean and how the various parts are to be expressed in the phenotype. Figure 6.8 gives an idea of what this might look like.

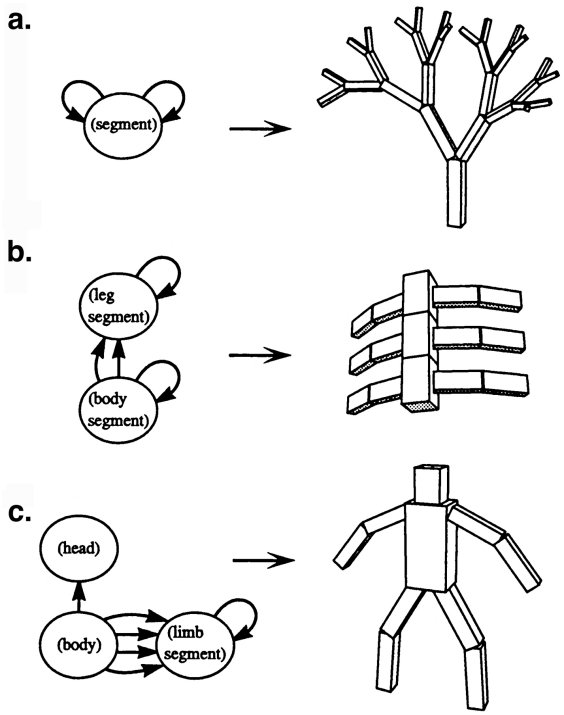


Figure 6.8: Generating a phenotype from a genotype in Sims's approach. (a) A tree-structure: The graph indicates that each segment spawns two other segments. (b) A body and a six-legged creature. (c) A humanlike body.

Development

The Sims system uses procedural development. The phenotype consists of a structure of three-dimensional rigid parts described by a directed graph (see figure 6.8). The developmental process requires an interpreter for these graph structures, depicted on the left side of each panel in the figure. The graph in figure 6.8a, for example, states that two segments are to be simultaneously attached to the existing segment. The shape, length, joint types, angles at which the joints are attached, and various other parameters are all subject to variation by the genetic algorithm; that is, they are subject to mutation and crossover. One of these parameters is the number of times a particular schema is to be applied. In this case, as can be seen on the right of figure 6.8a, this number is four, which leads to the standard, treelike structures shown. The examples in figures 6.8b and 6.8c are somewhat more complicated, but the principle is the same. The

developmental procedure always maps the same genotype onto the same phenotype: There is no interaction with the environment during development. The genotype also encodes information about sensors, effectors, and neurons that connect the sensors and effectors. The tactile sensors can be put onto all the structures' faces. Light sensors can also be defined in the genotype. Each different morphology (body, limbs, position of sensors) requires a different neural controller to match the morphology's requirements. Figure 6.9 shows how the sensors, effectors and neural connections are encoded.

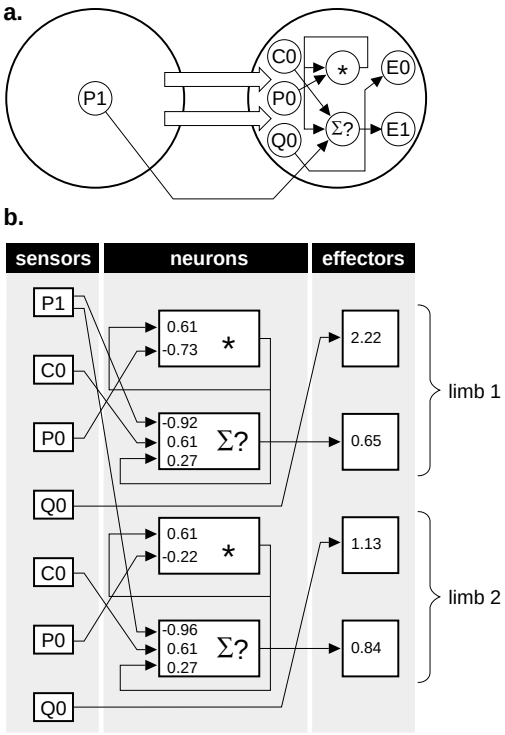


Figure 6.9: Encoding of the sensors, effectors, and neural connections.

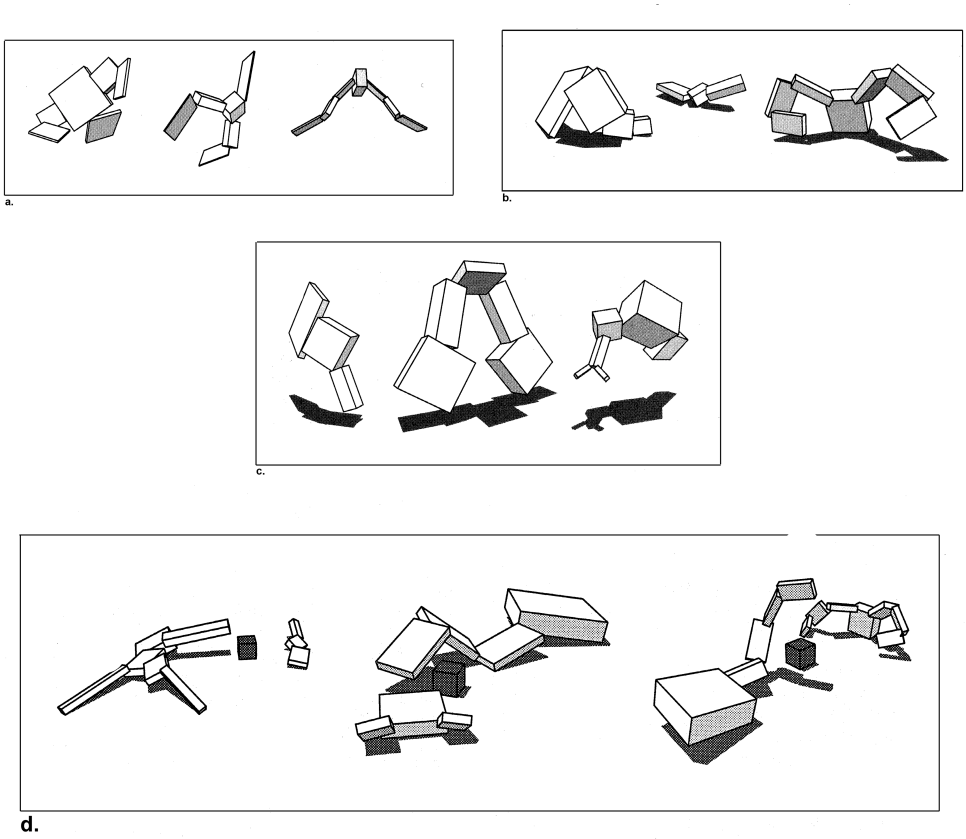


Figure 6.10: Selection of evolved creatures (a) for swimming, (b) for walking, (c) for jumping, and (d) for getting control over a cube.

Fitness and Selection

Once the phenotypes have been generated, they have to perform in the simulated world of physics. Swimming, walking, jumping, following, and getting control over a cube have been used to evaluate the creatures' fitness. The creature that can swim the longest distance within a given period of time is selected, and similarly for walking and jumping. Figure 6.10a shows creatures that have been evolved for swimming. Figures 6.10b and 6.10c show creatures evolved for walking and jumping, respectively. In the case of following, the average speed at which a creature moves toward a light source is taken as the fitness criterion. In another set of experiments, inspired by nature, the creatures must compete directly for a particular resource. In nature, creatures always have to compete in their ecological niche. In Sims' experiments, they have to try to get control over a cube. The creatures' final distances from the cube were used to calculate their fitness scores (the closer to the cube, the higher the score). Again, the details are not essential for our purposes. Figure 6.10d shows examples of creatures evolved by direct competition. The simulations typically use a population size of 300. Selection is by truncation, meaning that the population is "truncated", so that only the agents in the upper, say, 20 percent survive for reproduction. Furthermore, each surviving individual generates a number of offspring proportional to its fitness.

Reproduction

The creatures are then subjected to a reproduction process that includes mutation and crossover. Both operations are more complicated than in the case of simple GAs in which the genotype is simply a bit string. Here, graph structures have to be manipulated appropriately to yield structures that the developmental process can interpret. (For details on the reproduction process, the reader is referred to Sims' original papers 1994a, 1994b).

What can we learn from this example? First, the example shows that it is indeed possible to evolve creatures even if the morphology is not given a priori. Second, the creatures that evolved were surprising and funny. Especially if we look at their ways of locomotion, we find that they can be truly innovative. For example, one creature in figure 6.8b moves by continuously flipping over. Such unexpected things can happen because the search space, that is, the space of possible creatures, is enormous, and the more possibilities there are, the more chances that among them are creatures that can adapt to the demands of the environment. The third lesson follows directly from this point: The larger the search space, the more computation is required. Computation required to evolve these creatures is immense. Not only must we consider the space of connection weights in the neural network we must also consider the space required by possible morphologies. We know that by introducing constraints, we can cut down computation by orders of magnitude. However, and this is the fourth lesson to be gleaned from the example, the more constraints, the fewer the degrees of freedom, and the less surprise. This is similar to the exploration-exploitation trade-off. If everything is entirely unconstrained, we are certain not to get any convergence, that is, there are no creatures with good fitness values. Note that in spite of the fact that morphology is not given, those that result are still very constrained. The possible morphologies are composed of particular types of segments joined in a limited number of ways. This certainly helps evolution to converge, but it also forces it in a particular direction. The final message that we would like to take home from the example above, is that certain kinds of locomotion that can be evolved are not found in natural systems. One example is a creature that sort of rolls over. We see that evolution, at least artificial evolution, is by no means confined to the organisms we find in nature: It potentially exploits anything that is possible. And this is, among many other things, what makes it so fascinating.

Cell Growth from Genome-Based Cell-to-Cell Communication

(from Pfeifer and Scheier, 1999, chapter 8, page 250 ff)

Although vaguely inspired by nature, Sims is not trying to imitate specific natural systems. Natural systems always include a process of development. Although Sims has to translate genotype into phenotype, this process is entirely deterministic: If the genotype is given, the phenotype is determined. Peter Eggenberger, a medical doctor and theoretical physicist, is interested in modeling development from a biological perspective. He wants to create a computer simulation starting from what is currently known about the mechanisms of cell growth. His ultimate goal is to evolve an entire organism: its morphology, its neural substrate, its sensors, and its motor system. In particular, he wants to study the interaction of genetic and environmental factors in development. Of course, it will be a while before this can be achieved. As a first

test, Eggenberger's Artificial Evolutionary System was used to grow simple shapes and controllers for existing robots (Eggenberger 1996, 1997). Because many authors have tackled the latter task, we focus on the first, the evolution of shapes.

The Artificial Evolutionary System is based on the notion of genome-based cell-to-cell communication. What is encoded in the genome in this system is not the organism's structure, but rather the growth processes. Here is how it works: All the cells are placed on the points of a 3-D grid, which is then immersed in a solution of transcription factors: proteins produced by different cells. The concentrations of these transcription factors determine what a cell is going to do next. So let us briefly look at how individual cells "work".

Every cell contains a genome consisting of so-called regulatory genes and structural genes. The regulatory genes determine whether a particular structural gene is turned on. If turned on, the structural genes each perform their predefined functions, namely

- producing a transcription factor (dumping a transcription factor into the environment)
- forming a receptor (forming a receptor on the surface of the cell)
- forming a so-called cell adhesion molecule (CAM) on the surface of a cell
- cell division
- cell death
- searching for partner (searching for matching CAM in the cell environment)

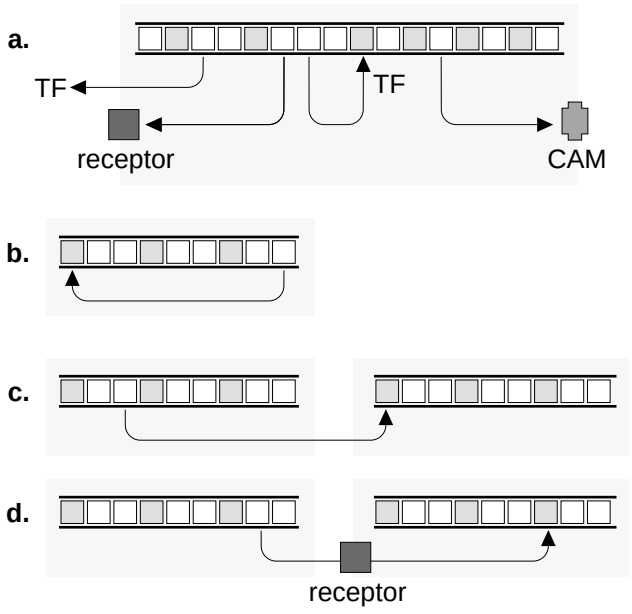


Figure 6.11: Basic mechanisms of the Artificial Evolutionary System. (a) Some basic functions of a gene: production of transcription factor, formation of receptor cell, creation of a CAM, used by other cells to make connections. This is required to build neural networks. (b) A transcription factor influences a regulatory gene within the same cell. (c) A transcription factor diffusion to another cell. (d) A transcription factor with affinity to a receptor.

Figure 6.11 illustrates some of these functions. When a transcription factor is produced, its concentration is highest at the location of the cell where it was produced. Further away on the grid, the concentration of this particular factor is lower because of diffusion. Regulatory genes are activated whenever the concentration of a transcription factor at a particular cell's location is high enough. Whether activation occurs depends on concentration and affinity. Affinity is calculated on the basis of geometric properties: The geometric properties of the transcription factors are compared to the geometric properties of the regulatory gene or the receptor protein on the surface of the cell. These geometric properties are represented in the genotype as sequences of four digits --- 1, 2, 3, and 4 --- meant to model the four bases of DNA: adenine, thymine, guanine, and cytosine. Figure 6.12 shows the encoding scheme. The typical genome used in the simulations consisted of 8 units. Each unit consisted of two regulatory and two structural genes, for a total of 32 genes (or one regulatory unit and two structural genes, as shown in figure 6.12).

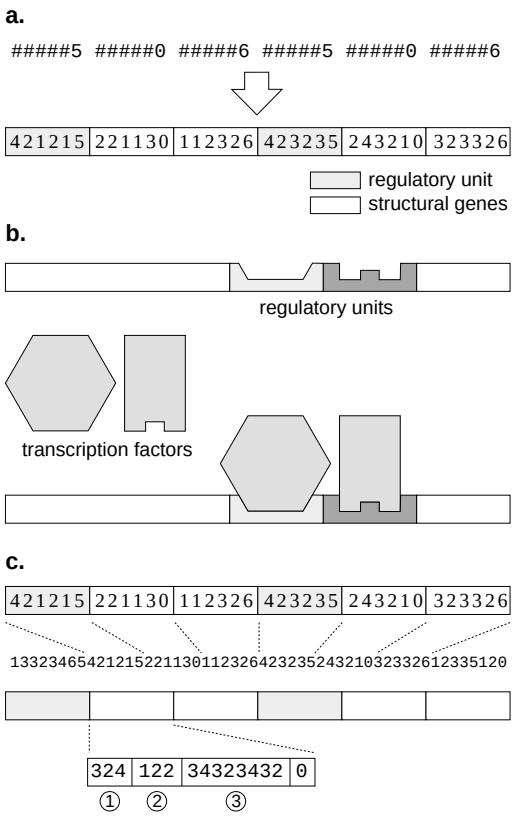


Figure 6.12: Implementation of the genotype. (a) Generation of initial genotype. Panel (b) depicts the matching process. (c) Details of encoding.

Encoding of the Genotype

At the beginning of a simulation run, a sequence of the 4 bases is generated at random (figure 6.102a). In each cycle, the cells “read” the concentrations of the transcription factors on the 3-D grid where they are located. Depending on the affinity of these transcription factors with the regulatory genes of the cell and their concentration, the regulatory genes are activated and the structural genes turned on. Figure 6.12b shows how a structural gene is activated. Activation in turn causes the structural genes to perform their

function. We have discussed the production of a transcription factor and the formation of a receptor cell. CAMs are used to form connections between cells, connections needed to grow neural networks. CAMs are used together with the function "searching for partner". If a cell "searches for a partner", it looks in the cell environment for a matching CAM. The search radius is encoded in the genome (see figure 6.12c). If a match is found, a connection is established. If the gene for cell division is turned on, the neighboring grid points are searched for an empty space. If all are occupied, the cell does not divide. In other words, cells inside the organism can no longer divide. If the gene for cell death is on, the cell is removed from the grid.

Development, Fitness, and Selection

In this setup, development results from highly complex dynamics. The organism's structure is not predefined in the genome. Figure 6.13 illustrates some of the shapes that have been grown. The goal in these examples was to grow organisms of a fixed size with a T-shape. The fitness function in these examples therefore has two components: (1) the number of cells in the final organism, and (2) a measure of "T-shapeness". The fitness function was set to 0 if the number of cells in the organism was more than 4,000. The "T-shapeness" measure was implemented as follows: A 3-D model of a T-shape was defined in Cartesian coordinates, and whenever a cell happened to be placed within this shape, the organism's fitness value was increased, otherwise it was decreased. Each generation consisted of 40 individuals. The final organism emerged after 72 generations.

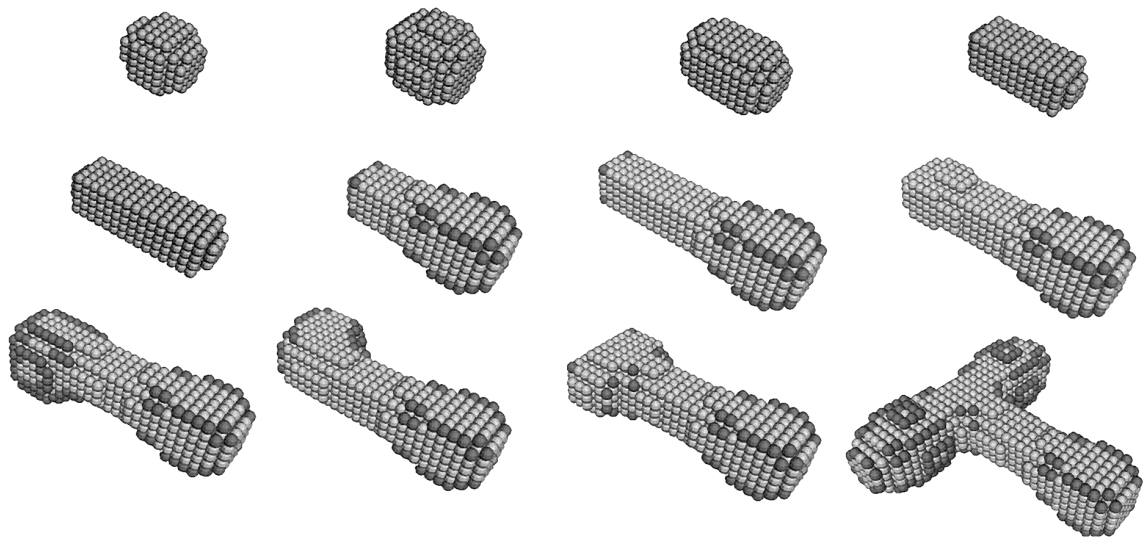


Figure 6.13: Phylogenetic development of an organism using the Artificial Evolutionary System. The goal was to evolve T-shapes. The shapes of the best individuals are shown after every six generations. The final shape had emerged after 72 generations.

The procedure, in general, works as follows. Start with a population of 3-D grids; these will eventually host a population of organisms. In each of these grids, put one cell on a grid point. The genome for this cell is initialized to a random sequence of letters, as shown in figure 6.12a. Depending on the initial concentrations of transcription factors, certain genes will be activated. Let the cells do their work, that is, divide, producing transcription factors, a receptor, a CAM, and so forth. Calculate the new concentrations of the transcription factor for each grid point according to the laws of diffusion. This leads to a changed

organism and changed levels of transcription factors. Repeat this cycle for all organisms a preset number of times. (We have to be careful not to confuse an organism, that is, a collection of cells, with the population of organisms.)

What can we learn from this example? First, it demonstrates a fascinating way of growing entire organisms without predefining their final structure in the genome. This makes the length of the genome independent of the organism's size. Second, the example shows a way biological insights can be translated into a simulation model in a natural way. For example, having the same genome for all the organism's cells and having cell differentiation, as the result of which genes are active, is a biologically motivated assumption. Any resulting organism has emerged because of a complex dynamic process. If this process is influenced, for example, by introducing additional transcription factors, the organism's shape changes. Third, the process of development in this example is more realistic than in the other models discussed so far. The designer does not pre-code the organism's shape. Fourth, as always, computation is expensive. The search space is very large. It is a real challenge to find appropriate constraints. And fifth, at the moment, the organisms have only shape. It would be more realistic if they also displayed interesting behavior. After all, behavior is the business we are interested in.

In summary, although this model is only a beginning, it opens up the possibility of experimenting with shapes.

6.4 Evolution of Hardware

Another interesting question is how the ideas of artificial evolution can be used to evolve hardware. Currently there are technological limitations mainly regarding time and costs, which prevent the application of such ideas in many areas. Nevertheless there are interesting developments in this field such as the configuration of Field Programmable Gate Arrays (FPGA) using Evolutionary Algorithms. An FPGA is a Very Large Scale Integration (VLSI) Silicon Chip containing a large array of components and wires. Switches distributed throughout the chip determine how each component behaves and how the components are connected to the wires. Configuring these switches determines the behavior of an FPGA. The specific arrangement is stored in its configuration memory. Since the FPGA can be interfaced to a host computer its configuration memory can be written to by an Evolutionary Algorithm (EA) (a Genetic Algorithm, Evolution Strategies or Evolutionary Programming (see 6.2 above)) running on the host computer. By setting the switches the EA creates a physically real electronic circuit, which can be tested and evaluated according to its real-world performance.

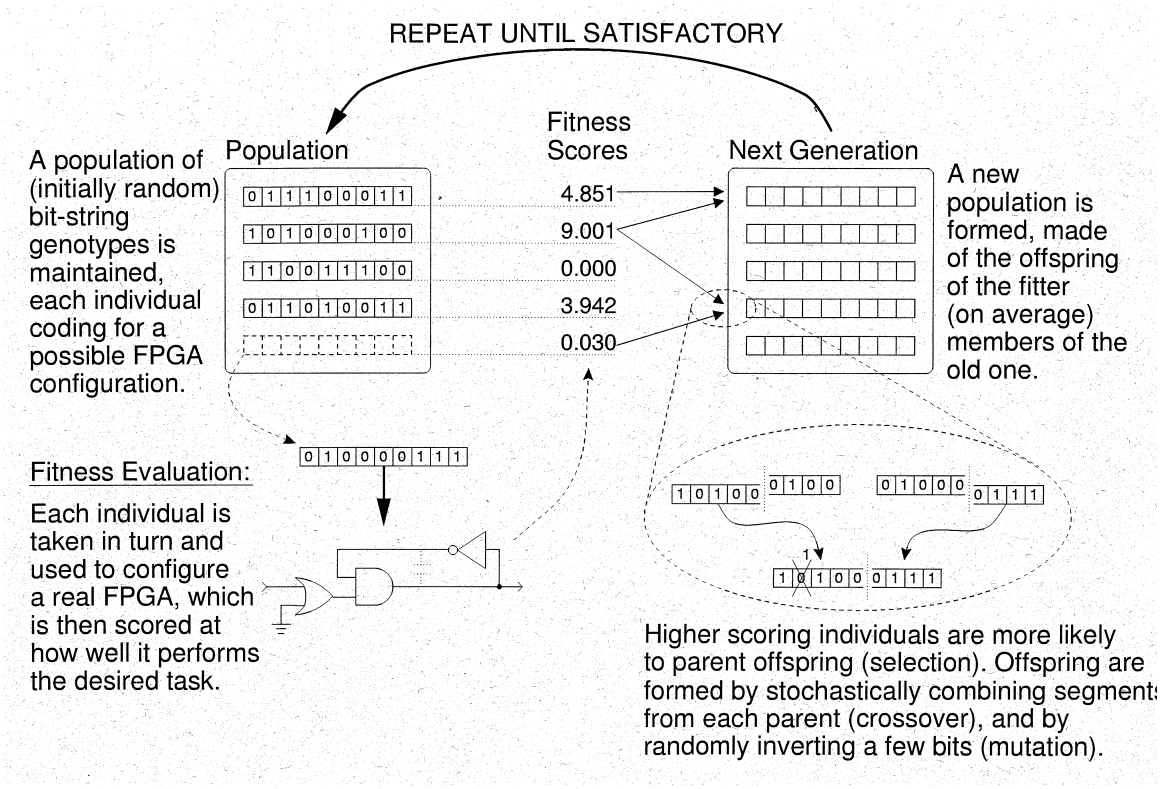


Figure 6.14: Evolving an FPGA configuration using a simple genetic algorithm.

Adrian Thompson of the University of Sussex uses the above setup to design electronic circuits. The circuit has a single input and a single output. Its task is to distinguish between 1kHz and 10kHz audio tones (a possible first step to pattern recognition and signal processing). Its fitness function consists in maximizing the difference in average output voltage caused by the two different input signals. In the experimental setting no synchronizing clock is used (consequently the input period cannot be timed) therefore evolution can exploit the rich natural unconstrained dynamics of the silicon to achieve the task. By automatically designing electronic circuits using an evolutionary algorithm major differences between the circuits designed according to conventional goal-oriented methods and those configured using evolutionary methods could be discovered. Perfect behavior was achieved using only a 10x10 array out of the whole 64x64 array to implement an electronic circuit (see Figure 6.15) whereas conventional design would need a much larger area to achieve the same performance. Within the 10x10 array not all cells contributed to the circuits behavior and among them are cells which are not even connected to the main part of the circuit but do still influence the behavior of the system (see cells shaded in gray in figure 6.15). These components must be interacting with the rest of the system by other means (e.g. electromagnetic coupling or power-supply loading). The detailed physical properties of the silicon the spatial relationships of the components and their interactions have been exploited by evolution in a fascinating way. A circuit evolved with an EA can use the electronic resources more effectively than an equivalent circuit designed by conventional methods.

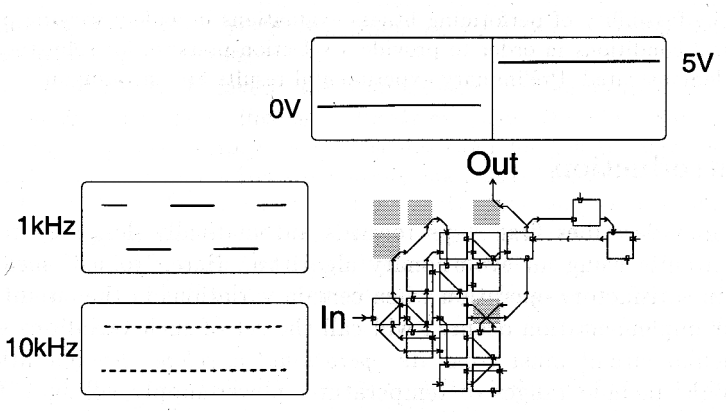


Figure 6.15: The tone-discriminator evolved on a single FPGA to distinguishing between two input signals. Note the cells shaded in gray which are not connected to the main part of the circuit but do influence the circuits behavior.

Since "unconstrained" circuits do not fulfill the present norms and expectation of the industry Thompson is concentrating on the evolution of robust circuits. Robustness is especially important for the use of so evolved circuits in real-world applications. Robustness means in this context that the circuit is able to operate in a satisfactory way even when variations in its environment or implementation occur.

6.5 Conclusion

The field of artificial evolution has grown into a large research field of its own. By copying certain aspects of biological systems artificial evolution helps to understand biological evolution and to solve optimization problems using new and unusual approaches. In this chapter we could not cover all parts of artificial evolution, topics we have not mentioned include more biologically inspired approaches such as Tom Ray's Tierra system. Besides we only touched the broad field of co-evolution when discussing the prisoner's dilemma in section 6.2.

Our knowledge of biological evolution and tied to it our knowledge of artificial evolution is not yet complete but it is rapidly evolving. We hope to at least have conveyed the basic ideas of this fascinating research field.

Bibliography

Axelrod, R. (1987). The evolution of strategies in the iterated prisoner's dilemma. In L.D. Davis (ed.). *Genetic Algorithms and Simulated Annealing.* London: Pitman, 32-41.

Bäck, T. B., and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, **1**, 1-23.

Bäck, T.B. (1996). *Evolutionary Algorithms in Theory and Practice*, New York, Oxford University Press.

Banzhaf, W., Nordin, P., Keller, R.E., and Francone, F.D. (1998). *Genetic Programming — An Introduction.* San Francisco, CA: Morgan Kaufmann.

Dawkins, R. (1986). *The Blind Watchmaker*, Longman.

De Jong, K.A. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. Ph.D. dissertation, University of Michigan, Ann Arbor.

Eggenberger, P. (1997). Creation of neural networks based on developmental and evolutionary principles. *Proc. ICANN'97*.

Eggenberger, P. (1997). Evolving morphologies of simulated 3d organisms based on differential gene expression. *Proc. Fourth European Conference on Artificial Life*.

Eggenberger, P. and Dravid, R. (1999). An evolutionary approach to pattern formation mechanisms on lepidopteran wings.

Eggenberger, P. (1999). *Evolution of three-dimensional artificial organisms: simulations of developmental processes*. Unpublished PhD Dissertation, Medical Faculty, University of Zurich.

Flake, G.W. (1998). *The computational beauty of nature.* Cambridge, Mass.: MIT Press.

Fogel, D.B. (1995). Computer simulation of natural evolution. In *Evolutionary computation: Toward a new philosophy of machine intelligence*. In D.B. Fogel (Ed.) (pp. 67-119). Piscataway, NJ: IEEE Press.

Fogel, L.J. (1962). Autonomous automata *Industrial Research 4*, 14-19.

Goldberg, D. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, Mass.: Addison-Wesley.

Holland, J.H. (1975). *Adaption in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Koza, J.R. (1992). *Genetic Programming: On the programming of computers by natural selection.* Cambridge, MA: MIT Press.

Koza, J.R. (1994). *Genetic Programming II: Automatic discovery of reusable programs.* Cambridge, MA: MIT Press.

Mitchell, M. (1997). *An introduction to genetic algorithms*. Cambridge, Mass.: MIT Press.

Pfeifer, R., Scheier, C., (1999). *Understanding Intelligence*, The MIT Press, Cambridge, Massachusetts

Rechenberg, I. (1973). *Evolutionsstrategie Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, Stuttgart, Frommann-Holzbog.

Rechenberg, I. (1994). *Evolutionsstrategie '94*. Stuttgart: Frommann.

Schwefel, H.-P. (1977). *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Basel, Birkenhäuser.

Sims, K., (1994a).Evolving virtual creatures. *Computer Graphics*, 28, 15-34.

Sims, K. (1994b). Evolving 3D morphology and behavior by competition. In R. Brooks and P. Maes (eds.), *Artificial Life IV Proceedings*. Cambridge, MA: MIT Press, 28-39.

Srinivas, M., and Patnaik, L.M. (1994). Genetic algorithms: A survey. *IEEE Computer, 27*, 17-26.

Thompson, A. (1998). On the Automatic Design of Robust Electronics Through Artificial Evolution. In M. Sipper, D. Mange, A. Perez-Uribe (Eds.), Evolvable Systems: From Biology to Hardware, *Second International Conference, ICES 98*, Lausanne, Switzerland, September 1998 Proceedings. Lecture Notes in Computer Science 1478, Springer.

Thompson, A. (1997). Artificial Evolution in the Physical World. In T. Gomi (Ed.), *Evolutionary Robotics: From Intelligent Robots to Artificial Life (ER'97*), AAI Books, 1997.

Thompson, A. and Layzell, P. (2000). Evolution of Robustness in an Electronic Design, In *Proc. third International Conference on Evolvable Systems*, Springer Verlag 2000. In Press. www.cogs.susx.ac.uk/users/adrianth/ .

Chapter 7: Self-replication

Krachend trifft die glatte Schärfe.
Wahrlich, brav getroffen!
Seht, er ist entzwei!
Und nun kann ich hoffen,
Und ich atme frei.
Wehe! wehe!
Beide Teile
Stehn in Eile
Schon als Knechte
Völlig fertig in die Höhe.
Helft mir, ach, ihr hohen Mächte!

Johann Wolfgang von Goethe, 1797

```
main(a){printf(a="main(a){printf(a=%c%s%c,34,a,34);}",34,a,34);}
```

Dario Dariol

Note: The strange line above is called a 'Quine' [from the name of the logician Willard van Orman Quine, via Douglas Hofstadter]. It is a C program that, when executed, will print out an exact copy of its own source code. This is a very small example of a self-replicating program. Can you figure out how it works?

7.1 Introduction to Self-Replication

Evolution works by making many copies of some organisms with a few random changes and by selecting the 'best' ones. The process is repeated over and over again, and leads to a refinement of the species. There has been lots of work on artificial evolution, and 'copying' is always taken for granted, being realized by centralized control program, whose task is just to make copies of the genome. In Nature it is not that easy, since all organisms are responsible for their own copying.

The prospect of self-replicating machines offers unimaginable potential benefits for mankind. The concept is pretty simple: Build a machine capable of just two processes, (a) build a copy of yourself, and (b) do something, e.g. mine for ore or explore other planets. This leads to a powerful conclusion. Once one machine is built and started, there will be soon two, then four, then eight, etc. With half of the machines replicating and half performing a task, pretty soon there will be a huge number of them all working hard. This is the closest there is to 'something out of nothing'.

This chapter consists of three main parts. The first part discusses the conceptual side of self-replication. We will have a look at NASA's proposed Self Replicating Lunar factory and John von Neumann's Kinematic Beast. The second part details some attempts to create an artificial Self Replicating entity in some form of Cellular Automata. CA's are a good medium for self-replicating entities, since they can model complex, non-linear systems and are mathematically tractable. There is also a discussion of Tierra, which is more like a one-dimensional CA. And finally, before the conclusions, comes a look at the mechanical side of self-replication.

7.2 Theoretical Aspects

NASA SRS Concept Team

In 1980, a team at NASA, the Self-Replicating System (SRS) Concept Team looked into viable applications of self-replication for space exploration and colonisation. The concepts they proposed were staggering, much closer to science fiction than to what we might call real science. Ideas of giant lunar factories starting off from one single egg and spreading like a virus across the lunar surface (see fig 1) or sending reproductive probes out into the galaxy to multiply and explore. It is a fantastic idea, but it is far ahead its time, since the necessary science and technology to make it reality are still to be discovered.

The proposal for a lunar factory was very detailed. A single spherical 'egg' would be sent to the moon, from which a number of small robots would emerge. There would be robots to mine, to transport, to process and to use different materials. The first job would be to begin the construction of a solar array, in order to be able to power the whole system. Some robots would search for the best location, while others would begin with the construction of a communication network. The plant would spread from a centre with mining robots levelling the ground and paving robots providing a smooth, stable surface. Then the central computer would be moved to the centre of this area. Soon areas of chemical processing, parts fabrication, assembly and control would be constructed, which would lead to even more solar panels. After about one year, a large factory would be ready to begin producing whatever would be desired, perhaps more 'seeds' for other satellites in the solar system or other areas of the moon. Some parts could not be manufactured by the system, due to either lack of materials or a too sophisticated manufacturing processes. These parts could be produced on earth and shipped to the moon. The estimation was that 4%-10% of the required parts would have to be sent. This seemed acceptable to the team. Unfortunately, the project was abandoned in 1983 due to lack of government funding. No wonder.

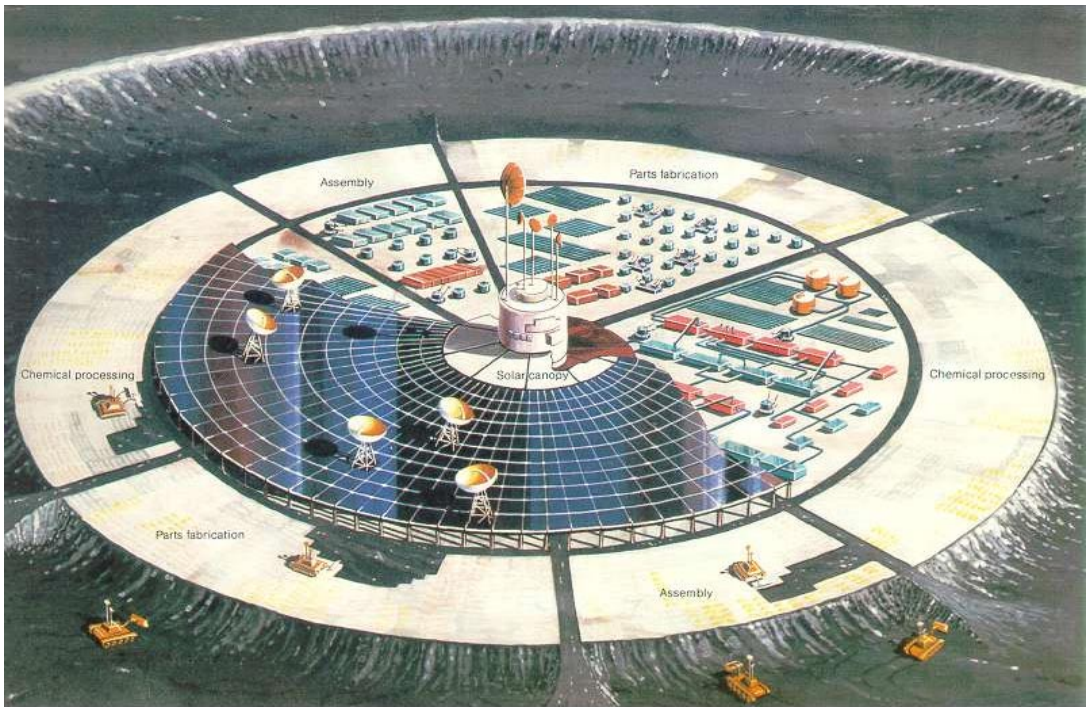


Fig 1: Artists impression of self-replicating lunar factory.

Von Neumann Kinematic Beast

One of the first self-replicating machines appeared in the imagination of the great mathematician, John von Neumann. This fantastic machine was a hypothetical robot, consisting of a computer with valves and other processing elements (this was in the '40s, many years before integrated silicon chips). In addition to just an electronic brain, the hypothetical robot would be equipped with: a manipulating hand-like element, a fusing element to connect two items, a cutting element to separate two items, a sensing element to recognise different parts and a many 'girders' (Deutsch: Stahlträger), these were rigid structures, to provide not only a chassis for the robot but also a means of information storage. Obviously the 'beast' would need an appropriate environment, for example a large lake containing millions upon millions of elements. The organism would consist of three sub-units.

Von Neumann wanted to formalise the process of self-replication. He believed that biological organisms could be seen as machines --- very sophisticated machines, but machines nonetheless. He believed that the important part of an organism was not the matter from which it is made, but rather the information and more importantly still, the complexity of the interactions of the information. The organism would consist of three sub-units:

1) A general construction machine, **A**. If a machine, **X**, is desired, this could take a description of it, **Φ(X)** and make it.

$$\mathbf{A + \Phi(X) \rightarrow X}$$

Where '+' means a machine composed of the left and right components and '→' means construction.

2) A general copying machine, **B**. This would make a copy of the instruction tape.

$$\mathbf{B + \Phi(X) \rightarrow \Phi(X)}$$

3) A control machine, **C**. When combined with **A** and **B**, this would activate them in the right order to produce **X** and a copy of **Φ(X)** and then connect them to each other and separate them from the original machine **(A + B + C + Φ(X))**

$$\mathbf{A + B + C + \Phi(X) \rightarrow X + \Phi(X)}$$

Now, if we take **X** to be a machine **A + B + C** and give it the name **D** we get:

$$\mathbf{A + B + C + \Phi(A + B + C) \rightarrow A + B + C + \Phi(A + B + C)}$$

So as you can see, this is a self-reproducing machine. Now lets continue with this logic and see where we end:

4) Let's make a machine $\mathbf{E = D + \Phi(D)}$

   Therefore, we can also say:

   $$\mathbf{E = A + B + C + \Phi(A + B + C)}$$

   So, **E** can replicate itself. $\mathbf{E \rightarrow E}$

5) Now lets take a new instruction tape, $\mathbf{\Phi(D + F) = \Phi(A + B + C + F)}$

   Where **F** is the instructions for building another arbitrary machine.

6) Now we can make a machine $\mathbf{E_F = D + \Phi(D + F)}$

   $$\mathbf{E_F = A + B + C + \Phi(A + B + C + F)}$$

   So we see:

   $$\mathbf{E_F \rightarrow A + B + C + F + \Phi(A + B + C + F)}$$

7) Which means ultimately:

   $$\mathbf{E_F \rightarrow E_F + F}$$

So we have finally found a description of a machine that can replicate itself while building an extra machine in the process. What that machine could be would depend on the application. When the technology to build $\mathbf{E_F}$ arrives, the possibilities are endless and potentially dangerous.

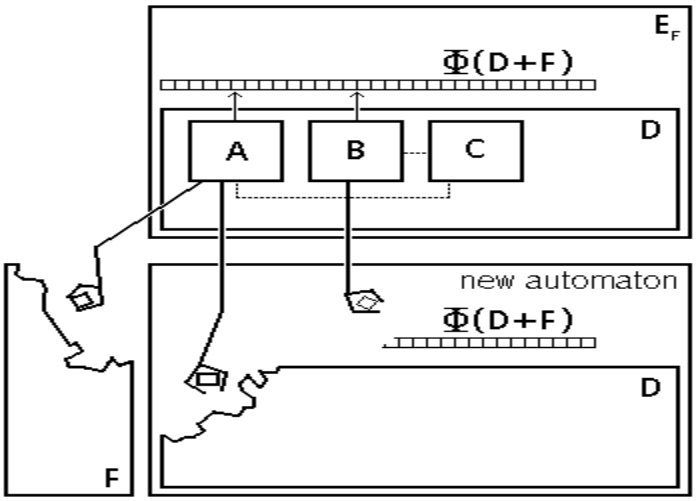


Fig 2: Schematic view of von Neumann's theoretical self-replicating machine.

7.3 SR Cellular Automata and related examples

Von Neumann Universal Constructor

The problem with the Kinematic Beast was that it was completely hypothetical. It could never be implemented. The logic used by von Neumann was sound, but it did not lead to clues of how to build such a system. The two main flaws with it are that it assumes a huge lake filled with parts and also that the parts are black boxes. Where did they come from? How are they made? What are they made of? The beast was destined to stay hypothetical. Soon after, in the 1950s when von Neumann with the help of Stanislaw Ulam invented Cellular Automata (see chapter 2), Neumann began working on an implementation of self-replication. Although CA's have been found to have a huge variety of potential applications in many areas of science because of their ability to describe non-linear dynamics and their tractability, they were originally designed to be the universe in which self-replicating 'creatures' could exist. These creatures would be made of information and live in a two-dimensional universe with their state changing over time according to the rules of the system and eventually make a copy of themselves. This is a good example of *emergence*, where local rules produce surprising and coordinated global behaviour.

The way Neumann's CA works is very complicated. The cellular automata had an incredible 29 states and an organism was composed of many sub-organisms in the shape of a box spanning 80 cells by 400 (see fig 3). This was the creature's main body and control centre but was still only a quarter of the entire organism. The rest of the cells were in a huge tape of 150'000 cells in a line attached to it. This was the blueprint for constructing a duplicate organism. Each cell would update according to the rules of the system and the states of its local neighbours. Soon the organism would start reading the tape and executing the instructions on it. It would extend an arm and begin manufacturing a child. After a while the child would be complete, the tape would be copied and finally the 'umbilical cord' would be dissolved, and left back would be the original organism in its original state and a perfect copy of itself. Now both organisms are ready to self-replicate.

Remember, that all this was done a few years before the discovery of DNA by Watson and Crick in 1953. Unfortunately, the self-reproducing automata of von Neumann were too large and too complex to be implemented. They could not be completed before von Neumann's death in 1957.

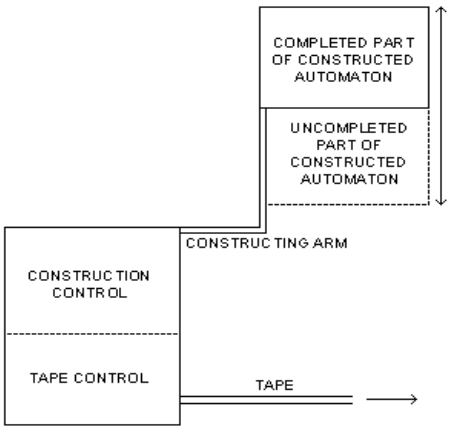
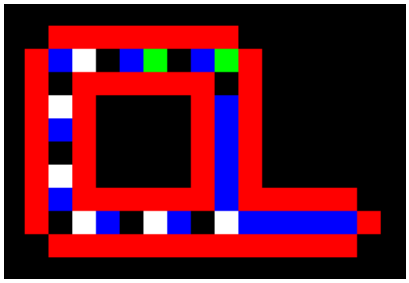


Fig 3: Schematic diagram of von Neumann's self-reproducing automaton

Langton Loops

In 1968, an engineer, E. F. Codd reduced the number of states from 29 to 8. This was more manageable but still very complex. Codd introduced the concept of ‘sheaths’. The sheaths were two layers of a particular state enclosing a single ‘wire’ of information flow. This was important in that the information could be contained and protected from the world, analogous to the walls of a cell.

The problem of complexity in the self-reproducing automaton of von Neumann and of Codd is a serious problem. In 1979, Christopher Langton set out to create self-replication in a CA. He realised that such a structure need not be capable of universal construction like those of von Neumann and Codd; it just needs to be able to reproduce its own structure. He used the same substrate as Codd, i.e. a CA with 8 states per cell and a von Neumann neighbourhood. He also kept the sheaths. In fact, he began with an element designed by Codd known as a periodic emitter. His ‘creatures’ however were very simple. Consisting of a single loop, replication would occur by extending an arm, which would bend round to create a daughter loop, then dissolve the umbilical and start again. One aspect of the system that Langton emphasized is the use of information in two modes, interpreted and un-interpreted. This is biologically analogous to translation and transcription respectively. The transcription of the information is accomplished by the information being copied at the umbilical junction and the translation is using the data to extend or bend the constructing arm.



```
  2 2 2 2 2 2 2 2
2 1 7 0 1 4 0 1 4 2
2 0 2 2 2 2 2 2 0 2
2 7 2           2 1 2
2 1 2           2 1 2
2 0 2           2 1 2
2 7 2           2 1 2
2 1 2 2 2 2 2 2 1 2 2 2 2 2
2 0 7 1 0 7 1 0 7 1 1 1 1 1 2
  2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Fig 4: Initial configuration of a Langton Loop. Note the red ‘sheath’ (state 2) that protects the flowing information. The data flows round the loop anti-clockwise. It is copied at the junction of the arm and one copy is sent round the loop, to keep the structure ‘alive’, and the other copy is sent down the arm to be interpreted into extending and bending it.

The way the replication is achieved is as follows. Inside the sheath the data is flowing round in an anti-clockwise direction. An arm ‘bud’ is formed at one corner of the loop and extended by the data hitting it, the sheath is extended automatically. The data is composed of space (black), information carrier medium (blue), extend arm data (white) and bend arm data (green). The red is the sheath, which protects the data. The two green data bend the arm 90°. It takes 151 time steps for a loop to reproduce. When the arm hits itself, a process of decaying the umbilical occurs to separate the loops and leaves them both ‘fertile’ i.e. both are capable of reproducing again.

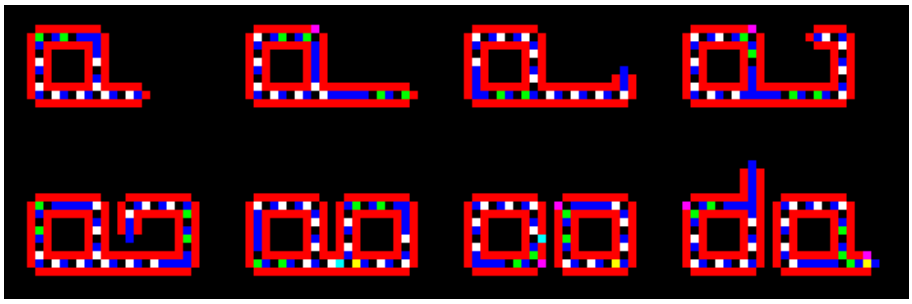


Fig 5: Self-replication process of a Langton Loop. The parent loop extends an 'arm' and bends it round to form a daughter loop. Then the umbilical dissolves to separate the loops. Now both loops are ready to self-replicate again.

Sayama SDSR Loops and Evoloops

The main problem with the Langton Loops is its lack of robustness. They assume a single seed loop of an exact configuration on an infinite CA with no obstacles. If there are any other structures in the universe or if the CA is bounded or wrapped then after a while the loops will become corrupt and the information flow will escape the sheath which will result in a big mess and loss of 'life'. Also, if the structures cannot reproduce in the desired direction, they die and their lifeless structures remain. It is similar to the way coral reefs have living organisms on the outside but the inside is made of the corpses of old, dead coral.

In 1999 Hiroki Sayama from the University of Tokyo extended the rule table for the Langton Loops to increase the robustness. In this system, if an extending arm hits another structure it is either absorbed, or tries to delete the obstacle and continue. Also, the structures can dissolve, which means they decompose gracefully, in order to leave space for other organisms. This robustness allows for a continuous life in a bounded or wrapped CA universe. The Structurally Dissolvable Self Replicating (SDSR) and the Evoloops can also be of bigger size, allowing for different species and evolution. Usually the evolutionary pressure leads to smaller loops simply because they can replicate faster.

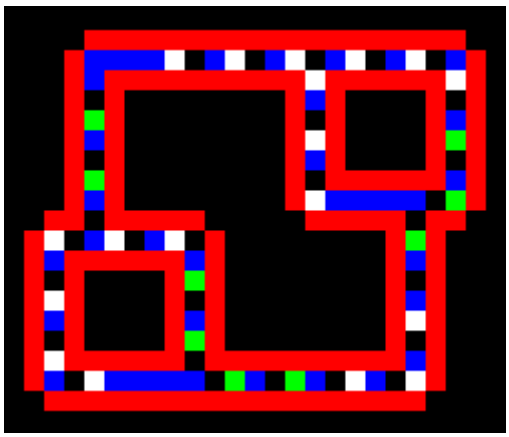


Fig 6: Stable configuration of two colliding SDSR Loops

Tempesti Loops

The self-replicating CA structures described above are very interesting, but they are just half of the story. The whole point of artificial self-replication is to provide a service for mankind from a small 'seed'. They need to be able to reproduce, but they need also to be capable of a second function. Langton abandoned the universal computation and universal construction capabilities of Von Neumann and Codd for the sake of simplicity and size. His loops were simply able to replicate. The loops of Giovanni Tempesti can also self-replicate, but after doing so, perform finite computations. The loops replicate *and* write the letters 'LSL' (Logic Systems Laboratory) inside themselves.

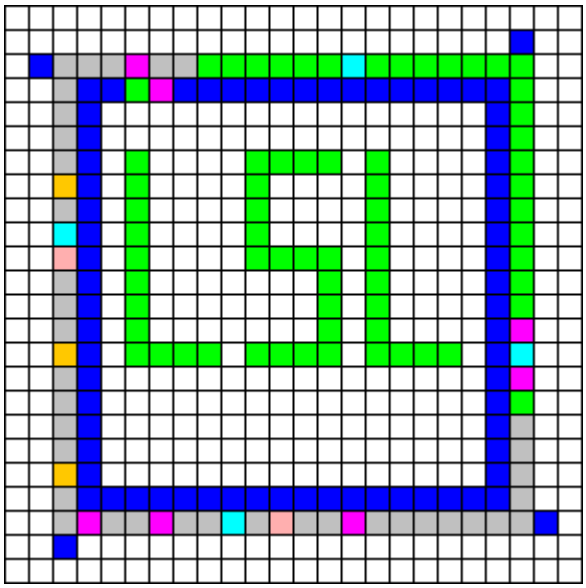


Fig 7: A Tempesti loop after it has spawned four children and constructed the letters 'LSL'. Both the information for self-replication and that for building the letters flow around the outside of the loop. Notice the concept of the sheath is still here but only the inner sheath. The rules are robust enough for the data to be subjected to the 'void'. The inner sheath merely acts as a guide for the data flow, not protection.

Tierra

Biologist Thomas Ray developed a system of self-replicating computer programs competing and evolving in what is claimed to be the first truly open-ended simulation. The creatures are composed of a string of instructions from a limited set of assembly language operands. Each 'creature' is a process running on a virtual CPU which means that it is just a computer program living in a very abstract environment. The idea comes from a game of competing computer programs called 'Core War', made famous by Dewdney. The universe for these things is the domain of the computer, competing for space (computer memory) and food (CPU cycles). The universe was seeded with a single organism (hand coded by Ray), which just had the ability to reproduce. It had a length of 80 instructions and it took over 800 instruction cycles to replicate. The virtual machine that executed the programs was designed to allow a small error rate (which meant mutations while copying, analogous to natural mutation). After a while the organism would have reproduced and the daughter would begin to reproduce also. Soon the space would be filled with replicating programs, some slightly different from their parents (see fig 8.a). Once the space was filled by 80%, the organism started competing for space and CPU cycles. A 'reaper' program was included to kill some of the organisms, with an artificial nod and wink to natural catastrophes. Now this is where the analogy to natural selection makes its entrance. The organism that copied faster had obviously more children, which was an advantage. Soon mutations only 79 instructions long proliferated – after a while even shorter organisms. Evolution had begun optimising the code. And actually, this is what evolution is good at. The shorter organisms dominated for a while, then something unexpected and seemingly impossible happened. An organism of only 45 instructions was born and started doing very well soon (see fig 8.b). Ray was confused, since he thought, that in this system an organism would need a minimum number of instructions to self-replicate, and 45 was certainly not enough. Yet these organisms were doing just as well as the larger ones, with lengths bigger than 70 instructions. The numbers of the longer and shorter organisms seemed to be linked. Then it dawned on him what had happened, evolution is very good at exploiting its environment and so after a while, some organisms had become parasites. A struggle was ensuing not unlike the celebrated foxes and rabbits idea. These parasites did not have any self-replication code of their own but they somehow had managed to tap into the abilities of the unaware hosts, not unlike real viruses. Of course, when the number of hosts thinned considerably, the parasites began starving, their number dropped and the hosts began to recover. The typical story of negative feedback of population numbers in ecosystems, albeit an artificial one. Then another interesting thing happened. A very long organism that had developed immunity to the parasites emerged. It could 'hide' from them (see fig 8.c and 8.d). Soon the parasites evolved into a 51 instruction long parasite, which could find the immune organism, and so the evolutionary arms race continued. Also, hyperparasites evolved which could exploit the parasites. Then these hyperparasites could be seen to 'cooperate', this means that they would exploit each other leading to the evolution of 'social cheaters', which would exploit them both. And so the system continued with its evolution of competing and cooperating self-replicating organisms, leading to surprise after surprise for the creator.

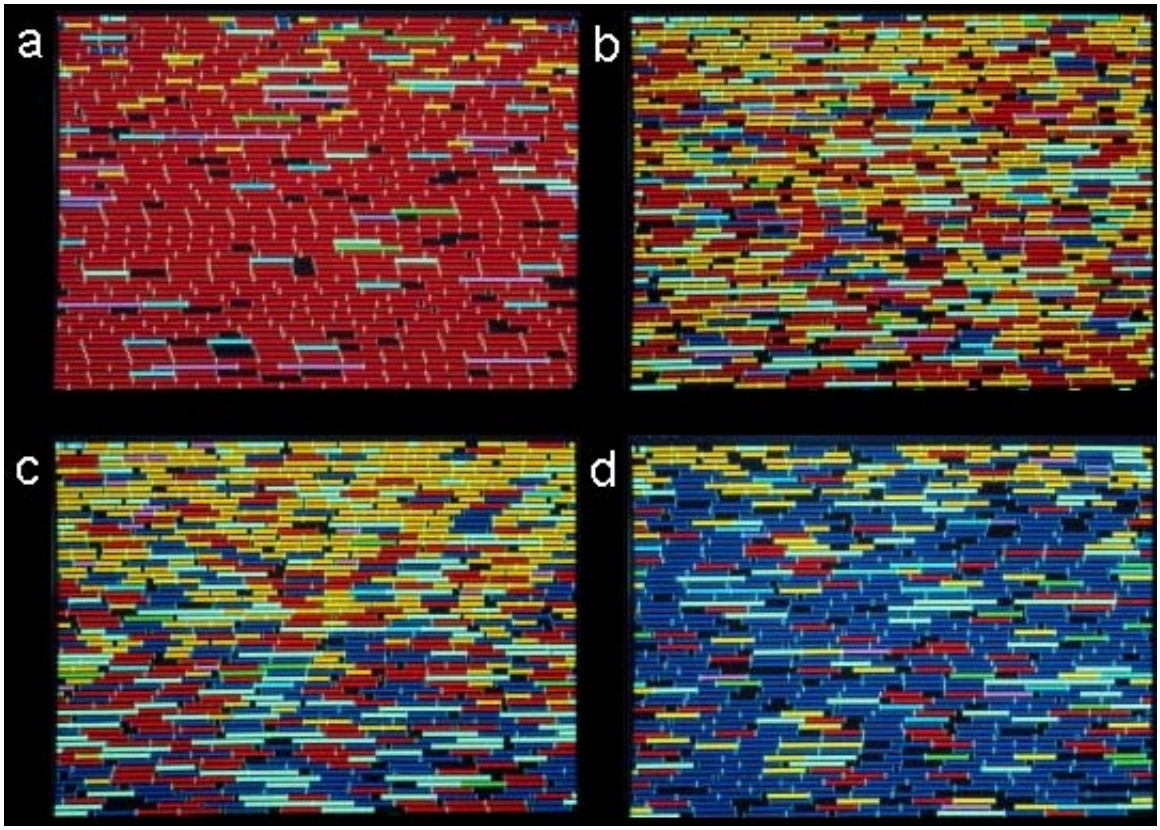


Fig 8: Tierra universe, growing in ecological complexity. Picture (a) shows the Tierra environment at quite an early stage. The universe is primarily filled with red bars; these are the descendents of the original hand-coded, eighty-instruction organism. Some other colours show mutated organisms, still able to survive and already we see a small but growing population of the yellow parasitic creatures. In (b) we see that the parasites are doing extremely well, exploiting the unaware red hosts. However, we can detect the emergence of the deep blue creatures, immune to the yellow parasites. In (c) we can observe the blue immune creatures, driving out the parasites. The red hosts are still there due to the parasites needing them. The immune creatures are by far the most successful in this run and so we see them proliferating greatly in (d).

7.4 Mechanical Self-Replication

Penrose Mechanical SR

This was the first example of mechanical self-replication ever realised. In this system, a number of identical units with particular shape were designed and mechanical motions. If many units are put into a box and shaken nothing interesting happens. However, if a 'seed' (two connected units) is placed into the box and the box is shaken, pretty quickly the seed uses the other independent units to reproduce. Soon most of the units are connected to another in the same configuration as the seed. The replication proceeds as follows (see fig 9):

a) The units are composed of four layers of mechanical components with particular shape and motions. The first two layers are responsible for locking together with other units and releasing them during the process of replication. The two components are for the right and left sides. The third layer is a stopper, which only allows units to approach and connect when it is receptive; i.e. during replication. The bottom mechanism allows a maximum of four units (two 'organisms') to be in close proximity. This stops the units clumping together in a crystal-like line. The 'seed' is the two connected units in the centre. The other two units are raw material with which the seed will reproduce. Both sides of the seed are in a receptive mode.
b) The left side of the seed has made contact with the raw material. Note that the top layer has released its hold, but the second layer is still secure.
c) The right side has made contact also. Now the second layer releases its grip. Note the bottom layer makes sure that the sides are not receptive until the reproduction is complete.
d) Now there is nothing holding the two halves together so they separate to show two structures, identical to the initial seed and primed, ready to reproduce again.

Very quickly the individual units get assimilated and the box is filled with dual-unit structures, which are direct copies of the ancestral seed. This example shows that non-trivial mechanical self-replication is possible.

Although this system is one-dimensional Penrose also designed a two-dimensional example, and it is easy to imagine a three-dimensional possibility, although the details would be of much greater complexity.

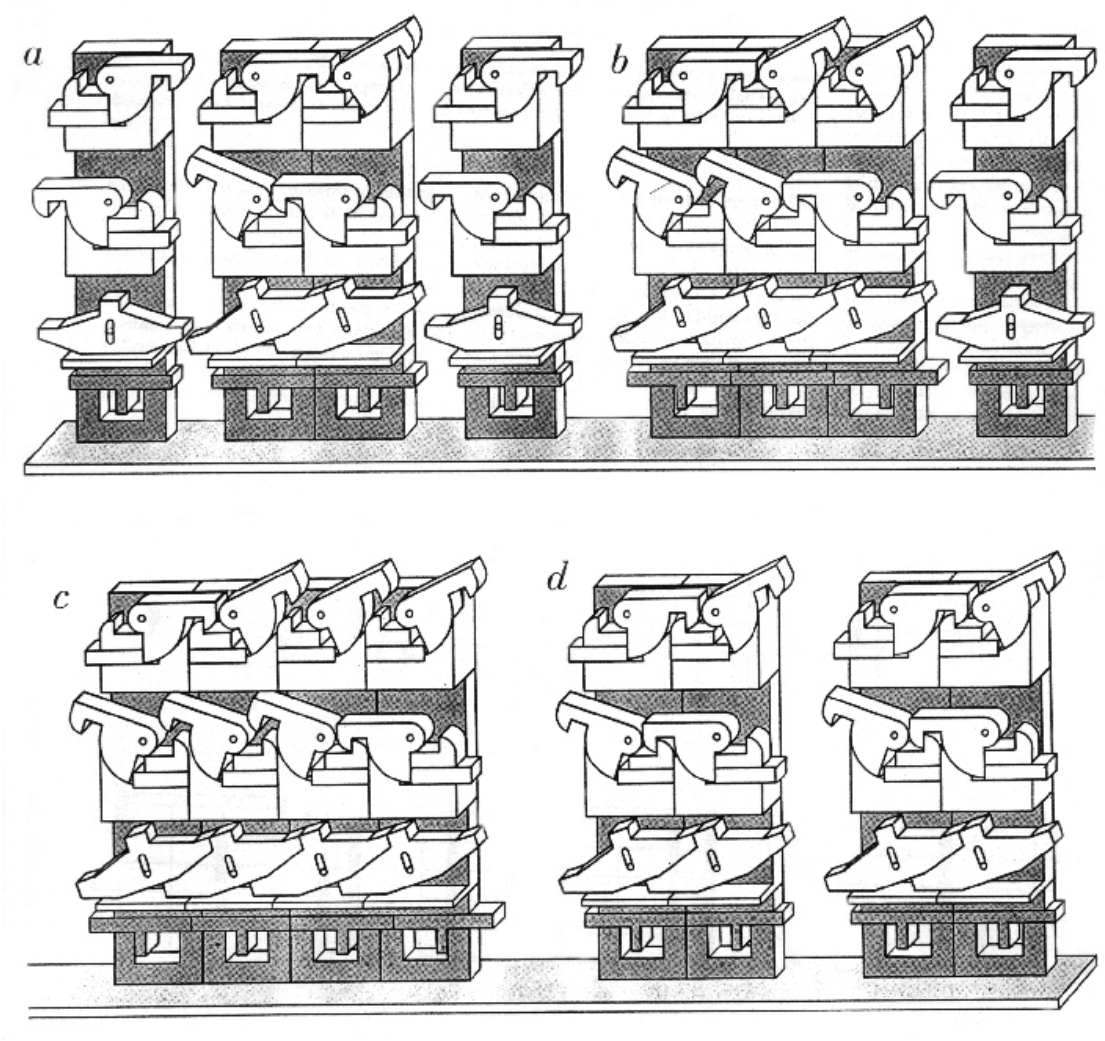


Fig 9: The Penrose mechanical Self-Replicator

7.4 Conclusion

In this chapter we have seen several examples of Self-Replicating systems. From the logical aspects to mechanical construction, from evolvability to ecology, the systems presented here are the first few stumbling steps towards the further understanding of such systems, whether real or artificial. This leads to the main conclusion, the point of this work. As we understand more about such processes, there are two areas of benefit. Firstly, the dream of 'something-for-nothing' has already been mentioned. A small 'seed' which could replicate exponentially and provide a huge service to mankind; millions of probes exploring planets or the galaxy, or hoards of nano-machines, cleaning our insides and building our desires. Secondly, a greater understanding of biological self-replication could help in many areas of medicine. One can imagine cures for cancer, regeneration of lost limbs or growth promotion of damaged or deformed brains.

The important point to grasp with this type of Self-Replicating system is that a copy is made without the need for a global copier. The copying is performed by the structure being copied, using only local rules and

exploiting self-organisation. This is how nature works. You can imagine how long it would take for an organism to grow, if each cell of the organism would have to be produced by a single 'factory' cell. Without even considering transportation problems, the idea of making one cell at a time to make up the 10^{14} cells in the human body is just silly. In computer simulations of self-replication, this is not an advantage due to the serial nature of the processors, but this is a moot point. It is of great importance in terms of parallel processing, self-replicating machines and biology.

7.5 Bibliography

1. Dewdney, A. K. (1984). In the game called Core War hostile programs engage in a battle of bits, *Scientific American*, Vol 250, No. 5, pp 15-19.
2. Langton, C. G. (1987). Artificial life, in *Artificial Life: Proceedings of an interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, volume VI of *SFI Studies in the Sciences of Complexity*, pp. 1-47, Los Alamos, New Mexico, Addison-Wesley.
3. NASA, (1980) in Chapter 5: Replicating Systems Concepts: Self-Replicating Lunar Factory and Demonstration, pp. 198-335. http://www.islandone.org/MMSG/aasm/
4. Penrose, L. S. (1959). Self-Reproducing Machines, in *Scientific American*, Vol 200, No. 6, pp. 105-114.
5. Ray, T. S. (1990). An approach to the synthesis of life, in Langton, C. G., Taylor, C., Farmer, J. D., and Rasmussen, S. eds., *Artificial Life II: Proceedings of the Workshop on Artificial Life*, volume X of *SFI Studies in the Sciences of Complexity*, pp. 371-408, Santa Fe, New Mexico, Addison-Wesley.
6. Sayama, H. (1998). Introduction of structural dissolution into Langton's self-reproducing loop, in Adami, C., Belew, R. K., Kitano, H., and Taylor, C. E. eds., *Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life*, pp. 114-122, Los Angeles, California, MIT Press.
7. Tempesti, G. (1995). A new self-reproducing cellular automaton capable of construction and computation, in Morán, F., Moreno, A., Merelo, J. J., and Chacón, P. eds., Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life (ECAL '95), volume 929, Lecture Notes in Artificial Intelligence of Lecture Notes in Computer Science, pp. 555-563, Granada, Spain, Springer-Verlag.
8. Von Neumann, J. (1966). The Theory of Self-Reproducing Automata. U. of Illinois Press, edited and completed by A. W. Burks.

Chapter 8: Conclusions

Artificial Life (AL) is a new and still rapidly growing research field. AL combines multidisciplinary attempts to explain complex phenomena in nature using new procedures, typically a synthetic i.e. bottom-up approach and computer simulations. Thus AL helps to understand such complex phenomena that could not be explained using familiar linear models but requested new methods and means.

AL has three main goals: studying biological issues, abstracting principles of intelligent behavior and develop practical applications based on these findings. In doing so AL uses computational techniques to understand biological issues and biological techniques to solve computational problems.

Referring to the connection between AL and biology AL can be characterized as the study of human-made systems that posses some essential characteristics of living systems.

Another way to describe AL is that it tries to give solutions to high-level problems by understanding low-level rules. Computer simulations show that complex pattern and complex behavior on a high level are emergent results of the application of simple local rules to small units and, of the simultaneous interaction of several of such units on a lower level. Expressed with few words: the sum is often more than its parts.

Another important finding of AL is that there is no need for central organization or control to reach such complex behavior as the behavior emerges from the low-level rules itself.

Examples of pattern formation in natural and artificial systems using simple rules are cellular automata, Lindenmeyer systems (L-systems), and fractals which can be used, to explain the growth-process of plants and the formation of patterns on seashell and, to develop artificial creatures in the game of life.

A similar result – the emergence of complex behavioral patterns – can be seen in robots and natural agents. In this context it is also called “distributed or swarm intelligence”. Through a process of mutual interaction, self-organization and interaction with the environment artificial and natural agents achieve impressive results they would never be able to achieve on their own. Good examples are self-organizing phenomena in insect societies (how bees organize their nest or how ants find the shortest path to a food source), the flocking of boids (a computer simulation of birds), or the heap building process of some robots (Didabots). Self-organizing artificial societies and agent-based models can also be used to understand and solve complex real-world problems such as distribution of resources, market predictions and data-mining. Again no central control is needed.

Another important topic in the broad research field of AL is artificial evolution. There are tree main types of evolutionary algorithms: genetic algorithms, evolutionary programming, and evolution strategies. These types have been developed at about the same time but with different intentions in mind. The application of evolutionary algorithms i.e. to get a wide variety of new solutions and offspring respectively, to evaluate their fitness and to allow only the best solution or the fittest offspring to reproduce leads to new and unexpected results. Methods from artificial evolution as optimization procedures, or more generally as design methods have proven very useful. Using evolutionary algorithms solutions have been found that humans could not easily have derived.

Conclusions 8.2

One limitation in AL is that it mostly uses computer simulations and models. Thus its findings aren't restricted to those of traditional biology but to the possibilities of computers. Despite the fact that simulations and models used in AL lead to new and unexpected results that can often not be predicted since too many interactions happen simultaneously they still depend on how the simulations or models are designed and what data is fed to the computer. Simulations and models always have to simplify the world and cannot fully reflect real-world situations. Another limitation is that results have to be interpreted and such interpretation depends strongly on the intention and background of the interpreter. One way to set aside such restrictions is to connect AL and its findings to the real world e.g. by building robots which have to interact with, and learn from, the physical world rather than by designing agents and environment on a computer.

AL is a rapidly growing branch of science that introduces new means and methods to find new solutions to existing problems. Thus it helps to a better theoretical understanding and practical application of existing biological knowledge and to formulate new ideas that can be applied to other fields of science and real-world problems.