DISSERTATION

THE EVOLUTION OF EMERGENT COMPUTATION IN CELLULAR AUTOMATA

Submitted by

Rajarshi Das

Department of Computer Science

In partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 1998

COLORADO STATE UNIVERSITY

June 30, 1998

We hereby recommend that the dissertation THE EVOLUTION OF EMERGENT COMPUTATION IN CELLULAR AUTOMATA prepared under our supervision by Rajarshi Das be accepted as fulfilling in part requirements for the degree of Doctor of Philosophy.

Committee on Graduate Work

Richard E. Eykholt

[illegible]

Charles W Anderson

Melanie Mitchell

[illegible]

Adviser

[illegible]

Department Head

ABSTRACT OF DISSERTATION

THE EVOLUTION OF EMERGENT COMPUTATION IN CELLULAR AUTOMATA

How does an evolutionary process interact with a decentralized, distributed system in order to produce globally coordinated behavior? Using a genetic algorithm (GA) to evolve cellular automata (CAs), it is shown that emergent coordination occurs when evolution takes advantage of the underlying medium's potential to form embedded particles. The particles, typically walls or defects between homogeneous domains, are designed by the evolutionary process to resolve global conflicts in the system. Descriptions of typical solutions discovered by the GA, and the discovered coordination algorithm in terms of embedded particles dynamics are presented. The particle-level description is also employed to analyze the evolutionary pathway through which the solutions were discovered. The results have implications both for understanding emergent collective behavior in natural systems and for the automatic programming of decentralized spatially extended multiprocessor systems.

Rajarshi Das
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523
Summer 1998

ACKNOWLEDGEMENTS

The number of people and organizations who have helped me along my way towards finishing this dissertation is indeed overwhelming. The following list is by necessity only partial.

First, my parents for their constant encouragement, patience, and moral support from the distant shores of India.

I am deeply indebted to my advisor, L. Darrell Whitley, for the confidence he showed in me and for his encouragement without which I would not have pursued this dissertation research. His help was invaluable in completing the research and in producing this document.

At the Santa Fe Institute, my advisors, Melanie Mitchell and James P. Crutchfield provided the foundation for this research and I express my deepest gratitude to them for their insights, guidance, and encouragement. I respect and trust their judgements and think of them as a good friends, collaborators and confidants.

The other members of my dissertation committee, Charles Anderson, Wim Bohm, and Richard E. Eykholt were extremely helpful. They were always interested in what I was doing, provided alternative perspectives, and helped improved the quality of this work.

The Computer department at the Colorado State University was simply wonderful. I received generous help and support in many different ways from Jenny Pruett, Keith Mathias, and Scott Gordon.

A large part of this research was conducted at the Santa Fe Institute (SFI). With its highly encouraging and stimulating academic atmosphere, SFI provided the ideal environment for my research. At SFI, I particularly profited from numerous discussions with Erica Jen, James Hanson, Cris Moore, Terry Jones, Una-May O'Reilly, Wim Hordijk, and Megumi Harada.

Many members of the staff at the Colorado State University and at SFI have offered their guidance and help with every issue related to the department, institute or the university. I am especially thankful to Ginger Richardson, Andi Sutherland, Marita Prandoni, and Ann Bohm for their generous support.

I have saved the best for last. Over the course of my entire graduate studies, my sister Rupa, has been my truest friend. Our travels together around the world provided the enthusiasm and rejuvenation to complete this work.

This research was supported in part by grants to the Santa Fe Institute, including funding from the John D. and Catherine T. MacArthur Foundation; and the National Science Foundation.

CONTENTS

Chapter 1

INTRODUCTION

Natural evolution has created many systems in which the collective actions of simple, locally-interacting components give rise to sophisticated forms of coordinated global information processing. Although the individual components in these systems have their own internal dynamics and are constrained to interact locally through restricted communication pathways, the system as a whole is able to display spontaneous emergent coordination.

A spectacular example of such behavior is displayed in the slime-mold amoebae *Dictyostelium discoideum*, which are found in soil throughout the world. For a significant portion of their life cycle, the amoebae occur as free-living unicellular organisms. However, when their food supply is exhausted, they undergo a fundamental transformation and spontaneously aggregate to form motile multicellular entities consisting of up to 10^6 cells. The aggregation process is caused by the the spatial and temporal coordination of several different cellular activities occurring independently in each amoeba. This remarkable display of globally-coordinated collective behavior arising out of local interactions within a homogeneous population of identical cells has been the focus of intensive research for the past several decades [Bon67, Seg84, Dev89].

The collective behavior displayed by slime-mold amoebae is not an isolated example. Globally coordinated behavior has been observed and studied in a number of other unicellular and multicellular organisms. For example, insect colonies often act as a coherent unit when they exhibit intricate nest-building and efficient foraging strategies; even though there is no central control-apparatus directing the behavior of individual insects. Since insects have a limited capacity for storing and processing information due to their small brains and short life spans, it is remarkable that evolution has succeeded in discovering a number of mechanisms allowing insects to display coordinated group behavior and form well organized colonies [HW93].

Other often-cited examples of emergent coordination include the efficacious group behavior observed in a flock of birds or a school of fish which increases the chance of survival of an individual organism when the group is attacked by a predator. In a different biological realm, emergent coordination is also observed in the brain where the synchronized oscillations of neural assemblies may play a significant role in encoding information [LD94, LI95, Hop95].

From a computer science standpoint, especially in the field of distributed and decentralized computer systems, the above observations are germane for two important reasons: (i) One of the most vexing problems in computer science today is the design and operation of large distributed systems where a central control is neither available nor desirable. Given the preceding observations, it is natural to ask: How has evolution overcome the problem of designing decentralized distributed systems capable of displaying coordinated behavior? (ii) In order to attain global coordination in a multicomponent system, the system must have the capacity to process and propagate information. However, the information processing occurring in these biological systems appears to be very different from the structured and algorithmic techniques of information processing prevalent in today's multiprocessor computer systems. This raises the question: How do we relate the behavioral dynamics of a natural multicomponent system to the standard model of computation in computation theory? In the following discussion, we look at both these issues in greater detail.

How does evolution design decentralized distributed systems?

The importance of globally-coordinated behavior in decentralized systems has been recognized for decades in computer science. From the earliest days of analog and digital computer design, the functioning of an entire computing device has been critically dependent on the global synchronization of individual processing units. One of the earliest mathematical articulations of a similar problem—firing squad synchronization—was given by Myhill 1957 and is still actively studied [Yun94]. The continuing and difficult problems in the contemporary design of multiprocessing systems still require that the issue of global coordination be directly addressed. Typically, the design choice is to use a central controller which determines the behavior of the individual components. However, such centralized systems can often suffer serious disadvantages when compared to decentralized distributed systems [CM94]. In a centralized system, a major share of the system's resources is preempted by the central controller that could have otherwise gone to other agents in the system. As one of the consequences, centralized systems are liable to fail easily if the central controller is incapacitated. Also, the central controller can often be a bottleneck for fast information processing or for information transmission. Depending on the processing load, serious degradation of the response time of the entire system can occur. In contrast, the sharing of resources can be much more equitable in a decentralized system. Such systems can process information independently at different locations and thus they may be more efficient, if properly designed. In addition, decentralized systems can be robust; the presence of one or more local faults may cause only a gradual degradation in the system's overall performance.

Although decentralized systems offer the possibility of a number of such advantages, it is difficult to design a collection of individual components and their interactions such that information processing occurs in a globally coordinated fashion. This is hardly surprising: it is not well understood how complex global coordination emerges from the individual actions of the simple components in natural systems. In most cases, the individual components display nonlinear behavior in their internal dynamics in addition to having nonlinear local interactions with other nearby components. Thus, these kinds of spatially-extended systems—that is, multicomponent nonlinear systems in one or more spatial dimensions with many degrees of freedom—are very difficult to analyze.

Given the widespread appearance of collective behavior in decentralized and spatially extended systems in nature, evidently natural evolution has successfully overcome the problem of designing global coordination in distributed systems In the examples discussed earlier, evolution has effectively taken advantage of spatially local nonlinear dynamics to produce entities which, on the one hand, consist of potentially independent subsystems, but whose behavior and survival, on the other hand, rely on emergent coordination. These observations leave us with unanswered, but significant questions: by what mechanisms does evolution discover the methods of emergent coordination? More specifically, how does evolution take advantage of nature's inherent dynamics to engender collective behavior in decentralized distributed systems? Furthermore, with an in-depth understanding of these issues, might it be possible to mimic evolution and employ it in a directed fashion to design decentralized and distributed computer systems?

How does natural systems perform computation?

As indicated in the preceding discussions, the survival of many natural multicomponent systems depend on their ability to display sophisticated forms of coordinated global information processing. But what are the basic computational elements that allow such systems to process information? Our current notions of "computation" and how a physical device can be used to realize computation are affected by our familiarity with contemporary digital computer technology. However, most natural systems are continuous, stochastic, and spatially-extended; thus it is very hard to relate their behavior to the functioning of a digital computer.

Nevertheless, in some situations, the observations of the behavior of a natural decentralized, multicomponent system might suggest that some form of computation is taking place. Insect colonies, economic systems, the immune system, and the brain have all been cited as examples of systems in which "emergent computation" occurs (e.g., see [For90, LTFR92]). However, discovering, detecting and understanding the underlying logic by which the computation is performed is typically very difficult.

In this work "emergent computation" refers to the appearance in a system's temporal behavior of information-processing capabilities that are neither explicitly represented in the system's elementary components or their couplings nor in the system's equation of motion or its initial and boundary conditions. More precisely, emergent computation signifies that the global information processing can be interpreted as implementing (or approximating) a computation [DMC94]. In this dissertation, the interest is in phenomena in which many locally-interacting processors, unguided by a central control, result in globally-coordinated information processing that is more powerful than that implemented by individual components or linear combinations of components.

Although information is processed in parallel in the above systems, the mechanisms of computation can be contrasted with the current parallel computation techniques that are better-understood. In a typical approach to parallel computation, a problem is subdivided into parts which are then sent to different processors. These processors solve the subproblems in parallel and return the partial solutions to a central unit which combines them to produce a coherent result. Even in the popular "parallel distributed processing" approach [MRH86], there is some degree of centralization. The system's overall information processing capability is crucially dependent on the "hidden units". However, because the hidden units are typically connected to all the input and output units, they have direct access to system-wide information. In contrast, in a truly decentralized distributed system, components have access to a limited portion of the entire system's state, and no component is computationally more powerful than any other component. In order to design many-processor systems manifesting globally-coordinated parallel information processing, methods are required which enable the processing, transmission and storage of information to be distributed throughout the system. More specifically, a design methodology should allow spatially distant components to communicate with each other without pre-designed direct communication channels, or through the mediation of a central information-processing entity. In essence, all of the components should facilitate the control, storage, and transmission of information, and should be able to switch from one subtask to another with ease.

How can a natural distributed system satisfy these criteria? The answer to this question hinges mainly on how information processing mechanisms is represented in natural systems. More specifically, given the spatio-temporal dynamics of a system, we must discover the locus of the basic computational elements and analyze how they facilitate the processing of information.

Overview of Approach: Evolving Cellular Automata with a Genetic Algorithm

To study the two questions posed in the preceding discussion, this work adopts a simplified framework: a population of idealized but behaviorally-rich distributed dynamical systems—one-dimensional cellular automata (CAs)—is coupled to an idealized evolutionary process—a genetic algorithm (GA). An individual cellular automaton consists of a large number of processing entities with their own local dynamics. In this scheme, survival of an individual cellular automaton is determined by its ability to perform a given computational task that requires global coordination.

Once the genetic algorithm has evolved high-performance cellular automata for a task, one can study their spatio-temporal behavior to determine how global information-processing capabilities arise from local interactions of the cellular processors. However, until recently, there was no formal method to relate the spatio-temporal behavior of a CA to its information processing capability. Recent progress in understanding the "intrinsic" information processing in spatially-extended systems such as cellular automata has provided a new set of tools for the analysis of computation in similar spatially-extended systems [CH93, HC92]. Crutchfield and Hanson use these tools to analyze the spatio-temporal dynamics of a number of CAs, many of which had been studied by other researchers using different techniques. By underscoring the feasibility of applying the framework formalized by Crutchfield and Hanson to the CAs discovered by the GA, this work opens new avenues to study computation in evolved systems. In particular, our study provides a detailed analysis of the basic computational elements embedded in the behavior of an individual cellular automaton which are ultimately responsible for increased fitness. This work extends the earlier work by Crutchfield and Hanson in assigning functionality to the discovered computational elements in the behavior of a CA and delineating how the different computational elements and their functionalities allow the CA to perform a given task. In addition, these tools also facilitate the investigation of the interactions between these behavioral mechanisms and the evolutionary processes which drive a cellular automata population to increasingly sophisticated computational strategies. Given the success of the GA in finding high-performance CAs for the given tasks, the results of this research have implications both for understanding emergent collective behavior in natural systems and for the automatic programming of decentralized spatially extended multiprocessor systems.

1.1 Summary of the Chapters

The following sections give a brief and informal summary of the subsequent chapters in this thesis.

Cellular Automata

Cellular Automata (CAs) are arguably the simplest example of decentralized, spatially extended systems. In spite of their simple definition they exhibit rich dynamics that over the last decade have come to be widely appreciated. The combination of their simple definition and rich behavior makes them a compelling choice for the study of how evolution interacts with multi-component pattern-forming systems.

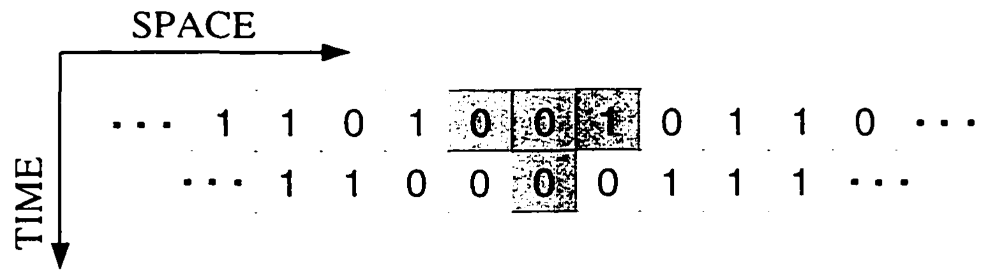


Figure 1.1: Two successive configurations of a two state, one-dimensional CA are shown. Each cell has information about itself, and its immediate neighbors on either side. The new state at each site is determined by a majority vote among the three cells in each local neighborhood. The CA's new configuration is shown in the lower row in the figure.

A CA consists of a collection of time-dependent discrete variables, called the local states, arrayed on a lattice of sites (or cells). An example of a one-dimensional CA, with two possible states per site, is shown in the top row of Figure 1.1. The set of all local states at a given time-step is called the configuration of a CA. The CA starts out with an initial configuration (IC), and at each time-step the configuration is changed as each cell in the CA uses the same rule to update its own local state. A CA rule can be expressed as a look-up table that lists, for each local neighborhood, the new local state for that neighborhood's central cell. For example, in the CA depicted in Figure 1.1, each cell has information about itself, and its immediate neighbors on either side. In the figure, the CA follows a majority rule, where the new state at each site is set to 1 if there are two or more 1s in the cell's local neighborhood; otherwise it is set to 0. The CA's new configuration is shown in the lower row in Figure 1.1. Such a depiction of CA configurations over successive time-steps is called a space-time diagram.

Cellular automata have been studied extensively as mathematical objects, as models of natural systems, and as architectures for fast, reliable parallel computation [Gut90, TM87, Wol86]. However, the difficulty of understanding the emergent behavior of CAs or of designing CAs to have desired behavior has up to now severely limited their applications in science and engineering, and their use in general computation.

The ability of a CA to directly process and store information is controlled by its neighborhood size and the number of states per site. The neighborhood size not only imposes an upper bound on the speed of information propagation in the system, but it also limits the amount of information regarding the entire system's state that is accessible from each cell in one time step.

Computational Tasks

This work focuses on two computational tasks that are simple to define but nevertheless require spatially global information-processing in CAs. The first task is a density classification problem, where a CA with two states per site (0 and 1) must categorize bit-strings according to the density of 1s in the string. After accepting an arbitrary bit-string as an initial configuration, a successful CA for the density classification task should reach the fixed point configuration containing all 1s if the fraction of 1s in the IC is more than 1/2. Otherwise, the CA should permanently settle on the all 0s configuration. Since density is a global property of a configuration, whereas a small-neighborhood CA relies only on local interactions mediated by the cell neighborhoods, it is not immediately apparent how to design a CA look-up table for the density classification task.

The second task requiring global coordination is similarly motivated. In this task, the goal is to find a look-up table for a CA such that, starting from an arbitrary IC, the CA reaches a final configuration that oscillates between the all-0 and the all-1 configurations. Without the use of a central controller instructing all the components when to change their state, it is not immediately clear how to design a CA that results in this type of global synchrony starting from random ICs.

Genetic Algorithms

The search for high-performance CAs for the above tasks is difficult for a number of reasons. First, the number of possible CAs grows very rapidly with increasing number of states per site and with increasing neighborhood size. Even with two state per site, exhaustive search is impractical when the neighborhood size of CAs is more than five. Calculus based methods which require continuity and derivative information are also not applicable because of the discrete nature of the CA search space. These observations suggest the need for a different technique which can search through a large number of possibilities for solutions in an efficient fashion and which does not require derivative or other auxillary problem-specific knowledge.

In this work, we investigate whether an evolutionary process can discover high-performance CAs for the computational tasks discussed above. Our choice of an evolutionary process as the search engine is not accidental. An evolutionary process can be viewed as a massively parallel method to search among a huge number of possible "solutions." Moreover, at an abstract level, the underlying mechanisms of an evolutionary process can be remarkably simple: natural selection of entities that reproduce with variation. These features provided us the main inspiration to employ an evolutionary approach in this work.

As an abstract computational model of an evolutionary process, a genetic algorithm (GA) was used to evolve CAs. GAs [Hol75] are a class of computational models of evolution which have gained popularity as efficient stochastic search algorithms [Gol89]. More recently, increasing interest has been shown among evolutionary biologists in using variations of GAs to study macroevolutionary phenomena such as the generation of morphological novelty in natural evolution [NP94, EA94, Joh94].

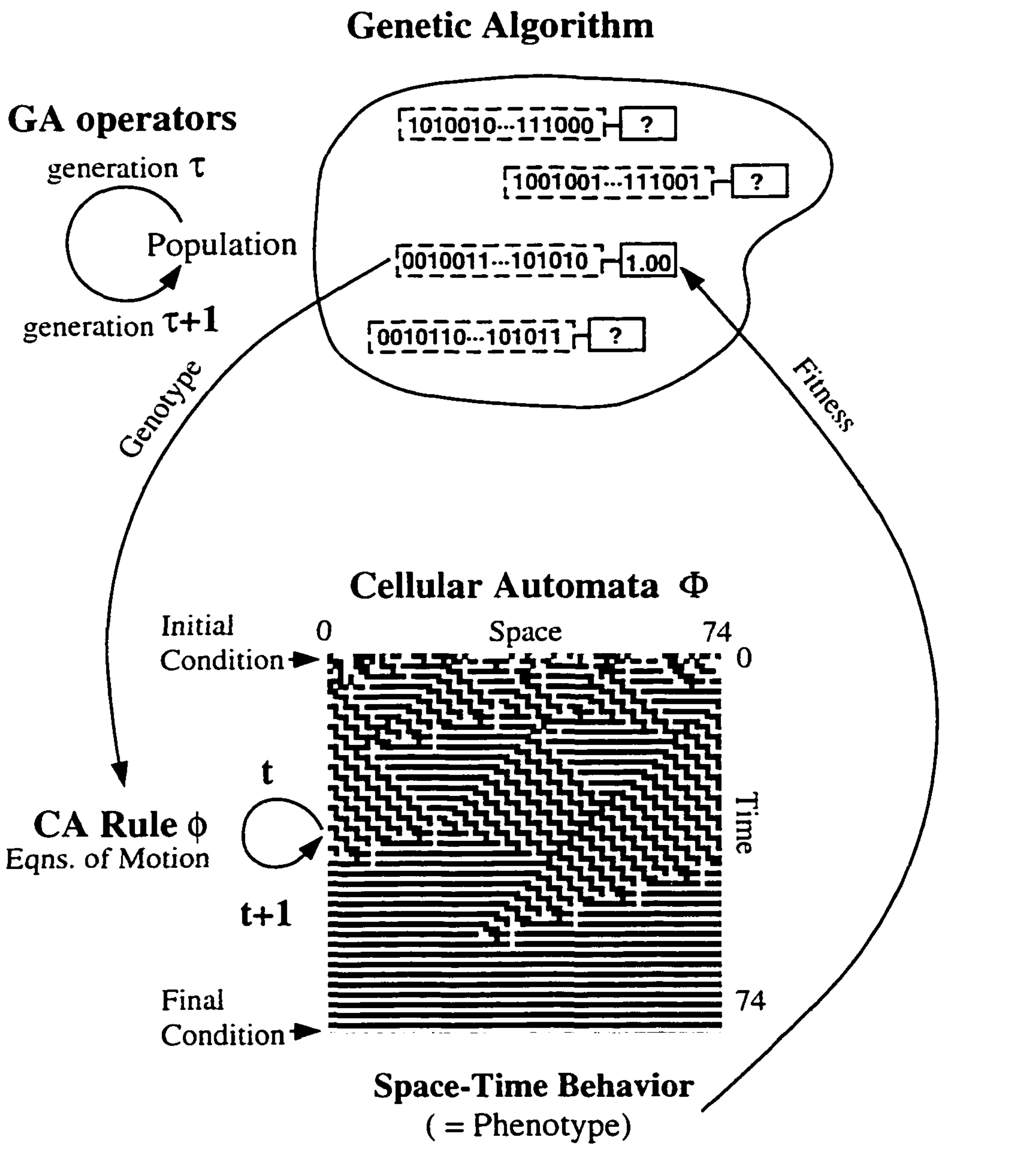


Figure 1.2: Schematic diagram depicting the experimental setup for GA evolving CAs.

In the experiments to be described here, a GA starts with a randomly generated population of CA look-up tables, and evaluates the CA rules in the population by estimating each rule's success in performing a given task. CAs which are more successful in performing the task are preferentially selected, and a new population of CAs is created by applying the genetic operators of crossover and mutation on the genomic representation of the more successful CAs in the existing population. This process, when repeated for a number of iterations, often results in the discovery of high-performance CAs.

The experimental design (Figure 1.2) used in this work for evolving CAs is interesting for a number of reasons. First, it consists of two nonlinear dynamical systems, a GA and a CA, operating at different time scales, which are coupled to each other. Due to the coupling, the dynamics in one system influences the dynamics of the other. Thus, a complete delineation of the results obtained from the experimental setup not only requires an in-depth knowledge of how each of the components work, but also how they interact with each other.

Although the operators in the GA act on the CA look-up tables (the genotype), the fitness of a look-up table is determined by the spatio-temporal behavior of the corresponding CA (its phenotype). Since the mapping from the genotype to the phenotype is typically nonlinear in a CA, and because the elements in the genotype often display complex nonlinear interactions, the process through which a GA finds high-performance CAs is not readily apparent. In addition, since a CA's behavior (i.e., its phenotype) is often very complex and difficult to characterize rigorously, the above problems are seriously compounded.

Evolved CAs

The results from the experiments to be described here show that for both computational tasks, the GA was able to find high-performance solutions. While the look-up tables of these CAs were very different from one another, and although they exhibited very different spatio-temporal behaviors, there was one striking similarity among all the high-performance CAs. In each case, the spatio-temporal behavior of a high-fitness rule was dominated by distinct patterns which interacted with each other in a seemingly complicated fashion. As indicated earlier, understanding the behavioral mechanisms that give rise to increased fitness is of crucial importance in this work. Thus, the fundamental question that needs to be answered is: How do these patterns and their dynamics aid the CA in performing the computational tasks? In other words, what is the emergent logic through which a CA is performing the desired computation?

Computational Mechanics Analysis of Evolved CAs

In order to understand the computation performed by the successful CAs, this work adopts the "computational mechanics" framework for CAs developed by Crutchfield and Hanson [CH93, HC92]. This framework describes the "intrinsic computation" embedded in the temporal development of the spatial configurations in terms of *domains*, *particles*, and *particle interactions* without any functionality or utility attached to the descriptions. A domain is, roughly, a homogeneous region of space-time in which the same "pattern" appears. The notion of a domain can be formalized by describing the domain's regularities using the minimal deterministic finite automaton (DFA) that accepts all and only those spatial configurations that are consistent with the regularity.

Once the domains have been detected in the spatio-temporal behavior of a high-fitness CA, nonlinear transducers can be constructed to filter the domains out of the configuration, leaving just the deviations from those regularities. In each of the high-performance CAs, the resulting filtered space-time diagram reveals the propagation of domain boundaries. These boundaries are called *particles* when they remain spatially localized over time. These "embedded" particles are one of the main mechanisms for carrying information over long space-time distances. This information might indicate, for example, the partial result of some local processing which has occurred elsewhere at an earlier time. Operations on the information carried by the particles are performed when the particles interact. The collection of domains, domain boundaries, particles, and particle interactions for a CA represents the basic information-processing elements embedded in the CA's behavior—the CA's "intrinsic" computation.

The identification of the computational structures embedded in a CA's behavior makes it possible to delineate the particle-level emergent logic or "strategy" which allows the successful CAs to perform the computational tasks. Since a high-fitness CA for a given computational task is performing some "useful" computation, i.e., accomplishing a desired mapping from input to output, we can assign specific functionality to each of the discovered computational structures according to the role they play in performing the computational task. It is in this light that the work presented here on CAs goes beyond earlier work by Crutchfield and Hanson.

The results from the analysis based on the computational mechanics framework also facilitates a comparison between different CAs at the functional level. As mentioned earlier, different high-performance CAs use different look-up tables and exhibit very dissimilar spatio-temporal behavior. However, careful analysis of space-time behavior shows that, at the particle-level, the different GA-discovered strategies used by these rules to perform a given computational task are often very similar.

Once the computational structures and their emergent logic in a high-fitness CA has been identified one may also ask: How are such computational structures discovered by the GA in the first place? Computational mechanics analysis of the behavior of the ancestors of the high-performance CAs gives a detailed picture of the interplay between the evolutionary dynamics and the computational structures that are embedded in the CA's behavior. We show that although the existence of some computational structures in the behavior of a high-fitness CA can be directly attributed to the amplification process resulting from the selection pressures in evolution, the discovery of some other computational structure is often due to pure chance.

The results presented in this work demonstrate how evolution can engender emergent computation in a spatially-extended system. For both computational tasks, evolution was able to take advantage of the underlying medium's potential to form embedded particles and in the end was able to design high-performance CAs. These discoveries are encouraging, both for using GAs to model the evolution of emergent computation in nature and for the automatic engineering of emergent computation in decentralized multicomponent systems.

Chapter 2

CELLULAR AUTOMATA: AN OVERVIEW

Scientists often use simplifying models to gain insights into physical systems. Generally, these models involve some form of mathematical apparatus, such as differential equations. As an example, fluid dynamics can be modeled with the help of partial differential equations where space and time are both assumed to be real and continuous. The physical properties (such as temperature or pressure) being studied in these models are also considered to be real and continuous and they are described as a function of the two independent variables, space and time.

As models in the continuous domain, partial differential equations have been strikingly successful in facilitating the analysis of a wide variety of physical systems. When spatially extended physical systems are modeled with partial differential equations, the simplifying assumption is that the system is continuous at a microscopic level. Although we know that such may not be the case (for example, box of gas, atoms in a crystal), the assumption often does not compromise the validity of the overall results. A fundamentally different approach would be to assume a system's configuration to be discrete in all respects. Cellular Automata (CAs) belong to this diametrically opposite class of models in which a fully discrete representation is used as the basis for modeling spatially extended systems. In a CA, space, time and the properties associated with each point in space exhibit only discrete values. In addition, the equations of motion governing the CA allow for only spatially local interactions. The rule, which is applied in parallel throughout the system, is also spatially and temporally invariant. Thus, as an abstract model class, the underlying assumptions in a CA are simple and easy to describe.

In spite of their discrete nature, CAs display a wide range of behaviors, many of which are qualitatively similar to behaviors observed in systems which are in the continuous domain. Like other abstract models, CAs have been used extensively to study the relationship between local interaction rules and the resulting global spatio-temporal behavior. However, because of their discrete nature and due to the spatially-local governing equations of motion, CAs are ideally suited for fast simulations on digital computers. Due to the simplicity with which they can be coded, CAs furnish a natural testing ground for theoretical predictions about spatial coherence and pattern formation in spatially extended systems. Consequently, CAs have become a popular choice as models for physical systems in a wide spectrum of scientific disciplines such as fluid dynamics, growth processes, spin systems, and reaction-diffusion systems [FHP86, Vic84, Tof84].

Prior to this, the study of CAs can be traced back to the work of mathematicians John von Neumann and Stanislaw Ulam in which CAs were invented to aid the study of biological self-reproduction. Subsequent generations of scientists have analyzed the breadth of phenomena exhibited by CAs, and have not only applied CAs in scientific modeling, but have also investigated CAs as an abstract mathematical apparatus, and have used CAs to perform sophisticated forms of parallel computation [TM87, Gut90].

2.1 Definitions

In this work, we focus exclusively on CAs in one spatial dimension. A one-dimensional *cellular automaton* consists of a linear array of N *sites* or *cells* indexed $0, \ldots, i, \ldots, N-1$. At any given time t, each site in the lattice is associated with a discrete variable s_t^i, called the *local state*, which is chosen from a finite alphabet $\mathcal{A}$ of k discrete symbols: $s_t^i \in \{0, 1, \ldots, k-1\} \equiv \mathcal{A}$. At discrete time intervals, the same *local update rule* ϕ operates in parallel (i.e., synchronously) throughout the lattice on local regions of a given radius r. This local update rule is denoted by the mapping

$$\phi(s_t^{i-r}, \cdots, s_t^i, \cdots, s_t^{i+r}) = \phi(\eta_t^i) = s_{t+1}^i, \tag{2.1}$$

where the string of local states $s_t^{i-r}, \ldots, s_t^i, \cdots, s_t^{i+r}$ of $2r+1$ symbols represents the *parent neighborhood* η_t^i and s_{t+1}^i is the resulting *child symbol.* The local update rule is usually represented as a *rule table.* here also denoted ϕ. which is simply an ordered list of (η, s) pairs. such that $\phi(\eta) = s$. In contexts in which the space index i and the time index t are not relevant. in this report we will simply use η and s with no sub- or superscripts to refer to a generic neighborhood configuration and a generic child symbol respectively.

Since a site in a CA can be in any one of k states, the number of possible parent neighborhoods equals k^{2r+1}. For each of these parent neighborhoods. the child symbol can again attain any one of the possible k states. Thus. for a given values of k and r. there are $k^{k^{2r+1}}$ possible rule tables ϕ.

The collection of all local states at a given time t in a lattice of size N is called the *configuration*: $\mathbf{s}_t = s_t^0 s_t^1 \cdots s_t^{N-1}$. $\mathbf{s}_t \in \mathcal{A}^N$ is the state of the CA considered as a dynamical system. where $\mathcal{A}^N$ is the set of all possible configurations and is the CA's state space. $\mathbf{s}_0$ is called the initial configuration (IC), It is also useful to define $\mathcal{A}^*$ as the union of all possible configurations $\mathcal{A}^N$ for $N \geq 0$.

The *global equation of motion* Φ maps a configuration at one time step to the next: $\mathbf{s}_{t+1} = \Phi(\mathbf{s}_t)$. where it is understood that the local function ϕ is applied simultaneously to all lattice sites. Starting from an initial configuration $\mathbf{s}_0$, the configuration at time t is

$$\mathbf{s}_{t+1} = \Phi(\Phi(\cdots\Phi(\mathbf{s}_0)\cdots)) = \Phi^t(\mathbf{s}_0). \tag{2.2}$$

To complete the description of a CA with finite lattice size N, it is also necessary to specify the boundary conditions. In this work, we will always use periodic boundary conditions, i.e., $s_t^i = s_t^{(i+N) \bmod N}$.

In addition to studying how the global mapping Φ operates on individual configurations in $\mathcal{A}^N$, it is also useful to examine how a CA processes entire sets of spatial configurations. The *ensemble operator* $\mathbf{\Phi}$ operates on a set of configurations or substrings of configurations by applying Φ separately to each member of the set[1] That is,

$$\mathcal{L}_{t+1} = \mathbf{\Phi}\mathcal{L}_t = \mathbf{\Phi}^t\mathcal{L}_0, \tag{2.3}$$

where $\mathcal{L}_0 \subseteq \mathcal{A}^N$ is any set of initial configurations and the ensemble of configurations are operated on by $\mathbf{\Phi}$ according to

$$\mathcal{L}_{t+1} = \{\mathrm{s}_{t+1} : \mathrm{s}_{t+1} = \Phi \mathrm{s}_t \quad \forall \mathrm{s}_t \in \mathcal{L}_t\}. \tag{2.4}$$

We allow a single ensemble to contain configurations with many different lattice sizes. Thus, the above formulation imposes no restrictions on the lattice size.

Most CAs are *dissipative* systems, i.e., under the global equation of motion Φ, the fraction of the state space $\mathcal{A}^N$ that can be reached by starting from a particular state decreases with time. This can be formally represented as

$$\mathbf{\Phi}\mathcal{L} \subseteq \mathcal{L}. \tag{2.5}$$

In the algebraic analysis of CAs, a distinction is often made between *linear* and *nonlinear* CAs [Jen90]. A rule ϕ is linear if it satisfies the additivity condition: for any two parent neighborhoods η_i and η_j

$$\phi(\eta_i) \oplus \phi(\eta_j) = \phi(\eta_i \oplus \eta_j) \tag{2.6}$$

where $\oplus$ denotes k-ary modulo addition, i.e., sum modulo k, the alphabet size. As a direct consequence of the above relationship, linear CAs obey a linear superposition principle in the global equation of motion. Thus, in a linear CA Φ

$$\Phi(\mathrm{s}^a) \oplus \Phi(\mathrm{s}^b) = \Phi(\mathrm{s}^a \oplus \mathrm{s}^b) \tag{2.7}$$

[1]The notation for CAs presented here and in the rest of this work follows that of Hanson and Crutchfield [Han93, HC95].

where $\mathbf{s}^a$ and $\mathbf{s}^b$ are any two configurations. Due to this decomposition, the behavior of linear CAs can be predictable. For example, consider a linear binary CA on a periodic lattice of length N following the local update rule $\phi(\eta^i) = s^{i-1} \oplus s^{i+1}$. Using the local rule ϕ, we can determine the dependence of the local state s_t^i on past lattice configurations.

$$\begin{aligned}
s_t^i &= s_{t-1}^{i-1} \oplus s_{t-1}^{i+1} \\
&= s_{t-2}^{i-2} \oplus s_{t-2}^{i} \oplus s_{t-2}^{i} \oplus s_{t-2}^{i+2} \\
&= s_{t-2}^{i-2} \oplus s_{t-2}^{i+2} \\
&= (s_{t-3}^{i-3} \oplus s_{t-3}^{i-1}) \oplus (s_{t-3}^{i+1} \oplus s_{t-3}^{i+3}) \\
&= (s_{t-4}^{i-4} \oplus s_{t-4}^{i-2}) \oplus (s_{t-4}^{i-2} \oplus s_{t-4}^{i}) \oplus (s_{t-4}^{i} \oplus s_{t-4}^{i+2}) \oplus (s_{t-4}^{i+2} \oplus s_{t-4}^{i+4}) \\
&= s_{t-4}^{i-4} \oplus s_{t-4}^{i+4} \\
&= \ldots \\
&= s_{t-2^j}^{i-2^j} \oplus s_{t-2^j}^{i+2^j} \quad \text{where } j = 2^k \text{ and } k \text{ is an integer with } 0 < k < log N..
\end{aligned}$$

Thus, in this example, the local state of a site value can be predicted 2^j time-steps in advance without actually simulating the CA. Since the superposition principle cannot be applied in nonlinear CAs, they are less predictable and are therefore much more difficult to analyze.

In some CA studies, a special symbol $\hat{\sigma} \in \mathcal{A}$ is designated to be the *quiescent* symbol with the following constraint on the rule table: $\phi(\hat{\sigma}, \hat{\sigma}, \ldots, \hat{\sigma}) = \hat{\sigma}$. Here the sites associated with the quiescent symbol $\hat{\sigma}$ (usually denoted 0) are interpreted as "inactive" cells, and all other sites denote cells that are "excited". Note that only a restricted set of CAs can have such quiescent states.

2.2 Elementary Cellular Automata

Wolfram [Wol84c] describes studies of one dimensional CAs with $k = 2$ and $r = 1$. Wolfram refers to these CAs as *elementary* cellular automata (ECA)[2]. In an ECA, there are 8 possible parent neighborhoods. Since the child symbol for each of the 8 parent neighborhoods can be either a 0 or a 1, there exists a set of 256 ($= 2^{2^3}$) distinct rules defining different ECAs.

Wolfram introduced a canonical ordering of the parent neighborhoods in an ECA rule table. In the top row, all the 8 parent neighborhoods are given. Below each parent neighborhood is given the state of the child symbol (or the output bit) a_i in the next time interval, where $a_i \in 0, 1$ with $i = 0, 1, \ldots, 7$. Thus, a CA rule ϕ can be specified in the following form:

Parent Neighborhood: $\eta_i =$	$s_t^{i-1}s_t^is_t^{i+1}$	000	001	010	011	100	101	110	111
Child Symbol	s_{t+1}^i	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Wolfram's labeling scheme assigns the integer $R = \sum_{i=0}^{i=7} a_i 2^i$ to each ECA rule table ϕ. The rule number for an ECA is thus an integer value between 0 and 255. As an example of an ECA, the bottom row in the following table lists the output bits in the rule table of ECA 232.

Parent Neighborhood: $\eta_i =$	$s_t^{i-1}s_t^is_t^{i+1}$	000	001	010	011	100	101	110	111
Child Symbol in ECA 232	s_{t+1}^i	0	0	0	1	0	1	1	1

Assuming the lexicographic ordering of the parent neighborhoods to be fixed, ECA 232 can be simply represented as (00010111) [3].

[2]Wolfram's choice of the word *elementary* in denoting two state CAs with $r = 1$ is unfortunate. CAs with $k = 2$ and $r = 1/2$ are even more *elementary* in their design, and as a result, they are also more amenable to algebraic analysis [Moo95].

[3]We will use the same lexicographic ordering of the parent neighborhoods in the rule table throughout this work. However, note that Wolfram uses the reverse ordering to represent ECAs [Wol84c].

Although there are 256 ECAs, there is some behavioral redundancy. In particular, ECAs can be grouped under several equivalence relations. For any binary CA rule ϕ, there exists a rule ϕ^c, such that $\phi^c(\eta) = \mathcal{C}(\phi(\mathcal{C}\eta))$, where the symmetry operator $\mathcal{C}$ flips each value (i.e., 1s are replaced with 0s, and 0s with 1s) in a binary sequence. $\mathcal{C}$ is its own inverse. The rules are equivalent in the sense that the global equation of motion can be easily transformed from one rule to another. Thus, $\Phi(s_0) = \mathcal{C}(\Phi^c(\mathcal{C}s_0))$. As an example, ECA 2 (i.e., 01000000) and ECA 191 (i.e., 11111101) fall in the same equivalence class.

Similarly, for any rule ϕ, it is possible to construct another rule ϕ^r such that $\phi(\eta) = \phi^r(\mathcal{R}\eta)$, where the symmetry operator $\mathcal{R}$ reverses the ordering in a binary sequence. As a result, $\Phi(s_0) = \mathcal{R}(\Phi^r(\mathcal{R}s_0))$. As an example, ECA 2 (i.e., 01000000) and ECA 16 (i.e., 00001000) fall in the same equivalence class. A rule is said to be symmetric if $\phi = \phi^r$. An example of a symmetric rule is ECA 18 (i.e., 01001000).

A third equivalence relation can be derived by applying the operators $\mathcal{C}$ and $\mathcal{R}$ successively, in either order, on rule ϕ. In such a rule ϕ^{cr}, the following relationship holds in the global equations of motion: $\Phi(s_0) = \mathcal{C} \circ \mathcal{R}(\Phi^{rc}(\mathcal{C} \circ \mathcal{R}s_0))$. As an example, ECA 2 (i.e., 01000000), ECA 191 (i.e., 11111101), ECA 16 (i.e., 00001000), and ECA 247 (i.e., 11101111) fall in the same equivalence class.

Once the ECAs are grouped into the complementation and reflection-symmetric equivalence relations as described above, 88 unique ECA equivalence classes can be identified. Thus, the notion of equivalence relations leads to a fundamental simplification in studying CA behavior. Nevertheless, the number of distinct equivalence classes of rules is always more than $\frac{1}{4}(2^{2^{2r+1}})$, where r is the radius of the CAs [4].

[4] Back of the envelope calculation shows that the exact number of equivalence classes for CAs with radius $= r$ is given by $\frac{1}{4}(2^{2^{2r+1}}) + \frac{1}{4}(2^{(2^{2r}+2^r)}) + \frac{1}{4}(2^{2^{2r}}) + \frac{1}{2}(2^{(2^{2r-1}+2^{r-1})})$.

2.3 Phenomenology

In spite of the apparently simple nature of the ECAs. they exhibit a remarkable range of behavior. While some ECAs produce behavior that is extremely ordered or rigidly periodic. others produce behavior that is very disordered.

Wolfram [Wol84c] detailed a survey of all ECAs and in loose analogy with continuous dynamical systems. he proposed that the behavior of all CA rules can be categorized into four classes as follows:

Class I Almost all initial configurations relax after a transient period to the same fixed configuration which is homogeneous (e.g.. all 1s). ECAs such as 0. 16. and 40 are examples of Class I CAs.

Class II Almost all initial configurations relax after a transient period to either (i) some fixed inhomogeneous configurations or (ii) some configurations consisting of temporally periodic structures that are spatially isolated. The final configuration. which might be temporally periodic. depends on the initial configuration. Examples of Class II CAs are ECAs 4. 55. and 73.

Class III Almost all initial configurations result in "chaotic" and aperiodic (both temporal and spatial) configurations after an initial transient period. The term "chaotic" here (and in the rest of this work) refers to apparently unpredictable spatio-temporal behavior. A change in a single site value in such CAs can affect the entire lattice in a finite number of time-steps. ECAs such as 18. 45. and 90 are examples of Class III CAs.

Class IV Some initial configurations produce complex patterns of localized structures which may persist for long periods of time. Examples of Class IV CAs include ECA 110. Compared to the other three classes, Class IV rules are relatively rare. This class was never well-defined.

In a finite lattice of size N, there are a finite number (k^N) of configurations. Thus, all CAs with a finite lattice must produce periodic behavior after time $t \geq k^N$. In contrast, the temporal periodicity displayed in a Class II CA refers to periodic behavior whose periodicity is much less than k^N. On the other hand, Class III CAs display temporally "aperiodic" behavior with periodicity close to k^N.

The behavior of a one-dimensional CA can be graphically represented in a space-time diagram. Figures 2.1 (a)-(d) depict typical behavior obtained from the four different Wolfram classes of CAs. Each space-time diagram plots 100 successive configurations on a lattice of size $N = 100$ with periodic boundary conditions. Time advances down the page. A site value of 1 is colored black, while sites with 0 values are colored white. A similar pictorial representation of the space-time histories of one dimensional CAs will be adopted in the rest of this work.

This kind of two-dimensional space-time representation of one-dimensional processes is not uncommon in nature. The intricate patterns observed on the shells of molluscs is a notable example. In these molluscs, the shell grows along one edge only, where material is deposited by a row of interacting and pigment-producing cells. Pigment deposited at the growing edge forms patterns that are never subsequently altered. Thus, a pattern on the shell is a space-time history of the activations of the pigment producing cells [Mei95].

While the diversity in the spatio-temporal behavior displayed by CAs is readily apparent in Figure 2.1, their behavioral classification is inherently qualitative since it is based on visual inspection of space-time patterns exhibited over long time scales. The behaviors of Class I, Class II, Class III, and Class IV cellular automata are meant to roughly correspond, respectively, to fixed points, limit cycles, deterministic chaos, and bifurcation points in continuous-state dynamical systems. Class IV CAs often exhibit a large number of spatially isolated transients that can propagate across space-time in an irregular fashion. Wolfram and others have hypothesized that Class

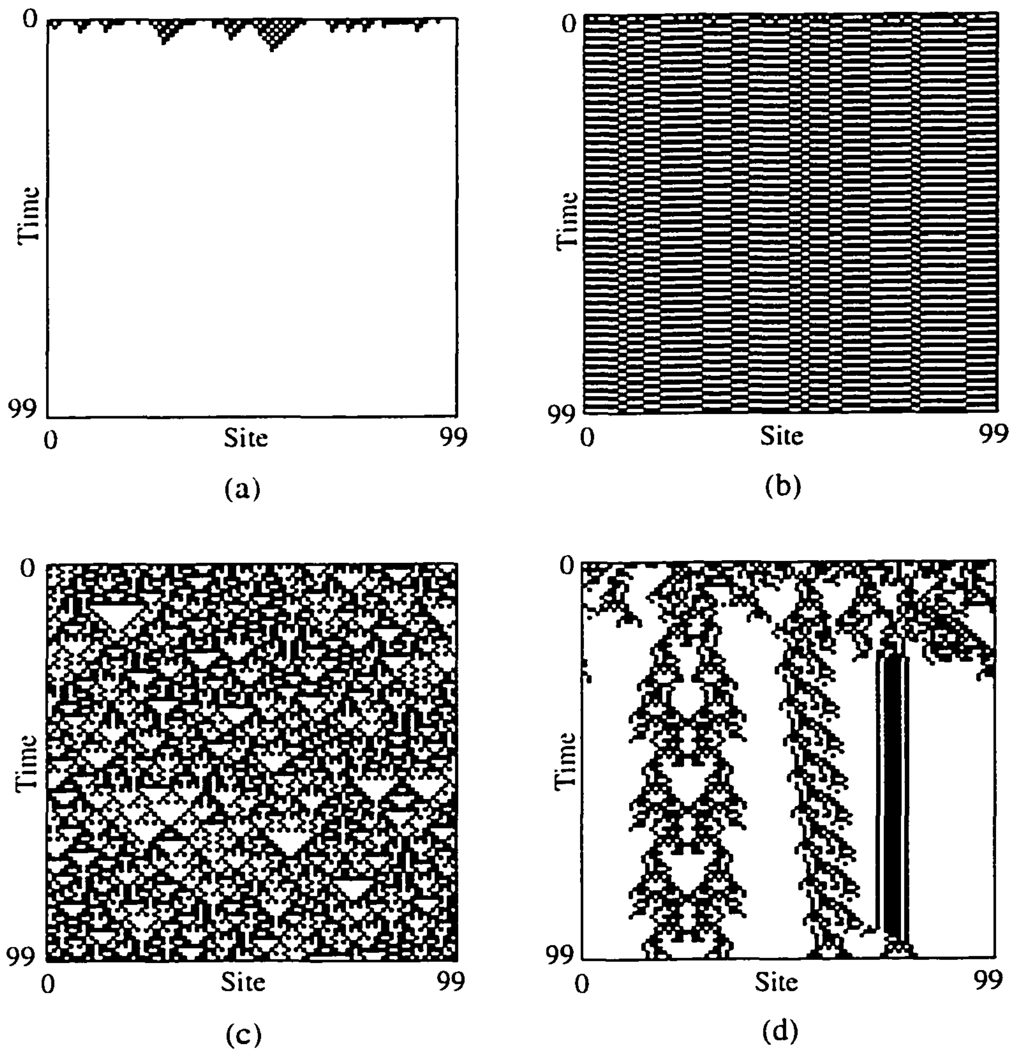


Figure 2.1: Examples of space-time diagrams exhibited by CAs in Wolfram's four behavioral classes. Top to bottom, left to right, they are: (a) Class I: ECA 40; (b) Class II: ECA 55; (c) Class III: ECA 90; (d) Class IV: (k = 2, r = 2) CA 1771476584. The initial configuration has been randomly generated in each case.

IV automata are capable of supporting computation, even universal computation. We revisit this issue in Section 2.5.

Despite its popularity, Wolfram's classification scheme lacks scientific rigor. Culik et. al. have shown that it is undecidable whether all finite configurations of a given CA relax to the quiescent configuration [CHY90]. As a corollary, they also show that the Wolfram class of a given CA is undecidable. More importantly, the very idea of classifying the entire behavioral repertoire of a CA into a single category is questionable. By selecting appropriate ICs, it can be shown that for a given CA, elements of more than one Wolfram class can simultaneously coexist in the lattice. How do we classify such CAs? Such issues question the very basis of Wolfram's classification scheme. For a detailed discussion on this topic, the reader may refer to [HC92].

2.4 Quantitative Analysis of Finite Cellular Automata Behavior

As mentioned earlier, in a CA with a finite lattice size the number of possible global configurations s is necessarily finite. Thus, one approach to the study of CA behavior involves the enumeration of all possible configurations, where each configuration is represented as a vertex in a discrete "state transition diagram" [Wol84a, WL92]. The directed edges of this graph correspond to the global equation of motion operator Φ which acts on s_t. Since a CA follows deterministic laws, any configuration at time t can have one and only one successor configuration s_{t+1} in the next time-step. The out-degree of each vertex in the graph is thus equal to one. Most CAs are dissipative in nature, and therefore vertices in the graph typically have an in-degree of more than one.

Configurations resulting from the iterations of a CA on a finite lattice can be divided into three groups according to the circumstances under which they are generated. One class of configurations consists of those that can exist only as initial configurations and never reappear in any subsequent iteration. These configurations are called *Garden of Eden* states. Formally, $\Phi^{-1}(\mathrm{s})$ is not defined if s is a Garden of Eden state. In the second group are those configurations that never appear except in the first τ time steps, $t = 1, 2, \ldots \tau$. Such configurations are called *transient* states. The third group of configurations are those which are visited repeatedly and appear in cycles. Once a cellular automata arrives at a configuration of the third type defined above, all configurations visited thereafter also belong to the same cycle. Configurations in this group are called *recurrent* states. Formally, $\Phi^{t+P}(\mathrm{s}) = \Phi^{t}(\mathrm{s})$ if s is a state in a cycle of period P.

The state transition diagram for a CA typically consists of a number of disconnected directed trees. Under the global equation of motion Φ, a deterministic CA can never leave one directed tree and enter another directed tree.

The study of CA behavior using the state transition diagram is severely limited in its scope for several reasons [HC92]. It is readily apparent that the construction of the state transition diagram for large lattices is impractical since the number of configurations grows exponentially with the lattice size as k^{N}. Thus the analysis of CA behavior with the help of a state transition diagram is only possible for very small lattices of the order of $N \sim 10^{1}$, which is miniscule compared to the lattices of the order of $N \sim 10^{4}$ to 10^{6} that are regularly used in the study of fluid dynamics.

The state transition diagram of a CA rule can be highly sensitive to the lattice size. As a result, the state transition diagram for a CA with a given lattice size might not generalize to other lattice sizes. A classic example is ECA 90, which typically displays spatially and temporally aperiodic behavior (recall Figure 2.1 (c)), and is considered to be a Class III CA exhibiting chaotic behavior. However, for any IC in a lattice size of size N, where $N = 2^{i}$ with $i = 1, 2, \ldots$, ECA 90 relaxes to the fixed point 0^{N} in 2^{i-1} time steps.

In trying to represent a spatial configuration as a single vertex, the state transition diagram representation ignores information about the internal spatial structures that may be present in the configuration. Since a CA is a spatially-extended system, configurations often consist of a pattern of symbols. Even under the simplistic notion of a "pattern", two configurations containing the same sequence of cell values (possibly subject to linear translations) contain the same "pattern". In the state transition diagram however, two such similar states can be arbitrarily distant from each another; even lie in different trees. Similarly, two configurations which are identical to each other except for a single cell value in the lattice can be maximally distant from each other in the state-transition diagram. Since one of the hallmarks of spatially extended systems is to spontaneously form spatio-temporal patterns and local regularities, the notion of a "pattern" seems to be of utmost significance in the study of such systems. In Chapter 6 we return to this topic and address some of the issues briefly raised in this section.

2.5 Computation in Cellular Automata

The notion of "computation" in CAs has different interpretations, although the notions are not necessarily independent of each other. In the most common portrayal, the CA is depicted as a mapping device which performs some "useful" computational task. Here, the "input data" is represented by CA's initial configuration. The IC is iterated for a predetermined number of iterations or until the CA has relaxed to the "goal pattern", which may be a time-invariant configuration. This final configuration represents the "output data". Thus, the CA rule table, which governs the input to output mapping, can be interpreted as the "program". Algorithms based on CAs have been used for multiplying integers [Atr65], in sorting binary numbers [Nis75], for real-time recognition of formal languages [Pec83], for image processing tasks such as filtering or skeletonization [PD84], and in solving multidimensional mazes [PD84], to name just a few applications.

A related but distinct notion of CA computation is that a CA is able to perform universal computation when given particular initial configurations. Universal computation is, by definition, the emulation of a Universal Turing machine (UTM). A UTM is a Turing machine which can simulate any other Turing machine when given an initial configuration which includes the description of the Turing machine to be simulated.

Two different approaches have been adopted to simulate a UTM in a CA. In one approach, Berlekamp, Conway, and Guy used a two-stage construction proof to demonstrate that the Game of Life—a well studied two-dimensional CA—can support universal computation [BCG82]. In the first stage, they show that it is possible to implement logic gates such as AND, OR, and NOT with the help of coherent dynamical structures—called gliders—that are supported by the CA. The information-carrying gliders consist of non-quiescent states and propagate against a quiescent background. In the second stage of the construction, the logic gates are used to implement an embedded serial computer in the CA, which might be a UTM or a general-purpose RAM computer. We should note that, although it is possible to construct CAs which are universal, in general it is undecidable whether a given CA is universal.

Although the above method is sound in principle, it has major limitations as an approach to perform parallel computation in a CA. Here, a CA simulates a UTM only when given particular initial configurations. Constructing such an initial configuration is itself a difficult process, since only a vanishingly small fraction from the set of all possible initial configurations can realize a UTM simulating a given TM. As a result, this methodology delineates the computational capability that can be achieved in principle by a CA, but tells us little about the capacity to perform computation that can be attained in practice in a CA. Moreover, in Berlekamp et. al.'s implementation of a UTM in a CA, only a small fraction of the sites in the lattice actually take part in the computational process to simulate the serial computation in the UTM. Thus, in this approach, not only are we constraining a massively parallel system to simulate a serial computer, but in addition, we are employing only a tiny fraction of the available parallel architecture for the simulation.

Lindgren and Nordahl have used a different approach to embed a UTM in a one-dimensional CA [LN90]. In their construction, the CA lattice simulates the one-dimensional tape and the position of the read-write head of a UTM. This embedding is made possible by ensuring that the state of each site s^i in the lattice is chosen from a set of finite alphabets $\mathcal{A}$, where $\mathcal{A}$ is the union of the set of tape symbols and the set of head states of the UTM to be simulated. Unfortunately this approach suffers from serious limitations similar to those of Berlekamp et. al.'s approach. The construction scheme assumes a UTM with a single head, and in essence, simulates a serial computer in a parallel machine. In addition, due to the assumption of a single head, a large proportion of the CA's rule table entries are left unspecified. These entries would have otherwise encoded the local interactions between two or more heads in a multiple headed Turing machine. Thus, the CA simulates a UTM only for certain initial configurations where the underlying UTM has a single head.

In an attempt to relate CA behavior to its computational capability, Wolfram conjectured that all Class IV CAs possess the capacity for universal computation [Wol84c]. The space-time diagram of a Class IV CA may consist of interacting coherent structures which move against a quiescent background. Given suitable initial configurations, Wolfram claimed that such Class IV CAs can simulate a UTM. However, given the informality of the definition of Class IV CAs, and the impossibility in proving that a given rule can or cannot perform universal computation, this hypothesis is virtually impossible to verify.

Crutchfield, Hanson, and Young have put forth a different notion of computation in a dynamical system, which they refer to as *intrinsic computation* [CY89, HC92]. Intrinsic computation alludes to the computational properties that are generic or typical in the system's behavior. This can be contrasted to the "in-principle" or "best-case" computational capacity of a system. Conway's proof of universal computation in the Game of Life is a notable example where the computational capacity has been engineered into some subset of the CA's behavioral repertoire.

When applied to spatially extended systems such as CAs, Crutchfield and Hanson's methodology treats the CA as a parallel computer by attempting to address the parallel computational properties of the system. Intrinsic computation does not necessarily measure any "useful" computation being performed by a CA. In fact, it intentionally does not depend on any externally imposed notion of utility of a computation being accomplished. Instead, intrinsic computation attempts to identify generic computational structures that exist over long spatial and time scales in the behavior of a CA. These entities may interact with each other and can play a significant role in the information processing occurring in the system. Implicit in the dynamics of these computational structures are notions such as information manipulation, transmission, and storage. In their studies of CAs, Crutchfield and Hanson attempt to discover the underlying components supporting information processing that are embedded in a given CA's generic behavior without attributing any functionality to the description of these components.

In this dissertation, we employ a genetic algorithm to evolve cellular automata to perform computational tasks requiring globally-coordinated information processing. Thus, in our scheme, any evolved CA with superior performance in a given computational task is also doing some "useful" computation. Useful computation is our interpretation of a CA mapping input configurations to output configurations so as to accomplish some task useful to us (and useful to the CA since since it can be eliminated from the population for performing poorly). The fitness function imposes the "usefulness" on the CAs.

To understand how a CA is performing a given computational task, we use the framework of Crutchfield and Hanson to delineate the computational elements embedded in the space-time behavior. Since the CA is accomplishing some useful computation, we can study the role of each of the discovered computational elements in performing the overall computational task; i.e., we can assign specific functionality to the computational structures. In this way, a semantics of utility—in terms of survival and the performance of a computational task—is introduced out of CA's intrinsic computational abilities.

ECA 57 (11101100) provides a context for the preceding discussion in drawing the distinction between intrinsic computation and useful computation. Space-time diagrams of ECA 57 starting with two different randomly generated ICs are illustrated in Figure 2.2. The figure shows that a checkerboard pattern occurs in ECA 57's spatio-temporal behavior. Also, the presence of back and white "gliders" that move with a constant velocity against a checkerboard background can be noticed. Another important observation here is that, starting from *any* initial condition, the configuration after N steps consists of either only black gliders or only white gliders that move against a checkerboard background (where N is the size of the lattice). Thus any final configuration falls in one of the two regular languages $(11)^+ \cup (01)^+$ or $(00)^+ \cup (01)^+$.

In delineating the intrinsic computation occurring in this CA, Crutchfield and Hanson's approach would identify and formally characterize the properties of the checkerboard pattern, the gliders, and the interactions between the gliders. The identification of the computational structures is made with the help of formal languages by detecting the regularities in the space-time behavior (such as the checkerboard background) and the deviations from the regularity (such as the black and the white gliders). However no attempt is made to associate any semantics with these information-processing structures and their interactions.

In this particular example, the computational structures are visually obvious from the CA's space-time diagram, but in general this may not be the case. These computational structures are generic properties of ECA 57's behavior since they appear in the space-time diagram irrespective of the choice of the initial configuration.

By focusing on these computational elements this approach identifies the locus of information processing in the CA's behavior (for example, when a back glider collides with a white glider leading to mutual annihilation) and shows how information processing may occur in parallel in the system.

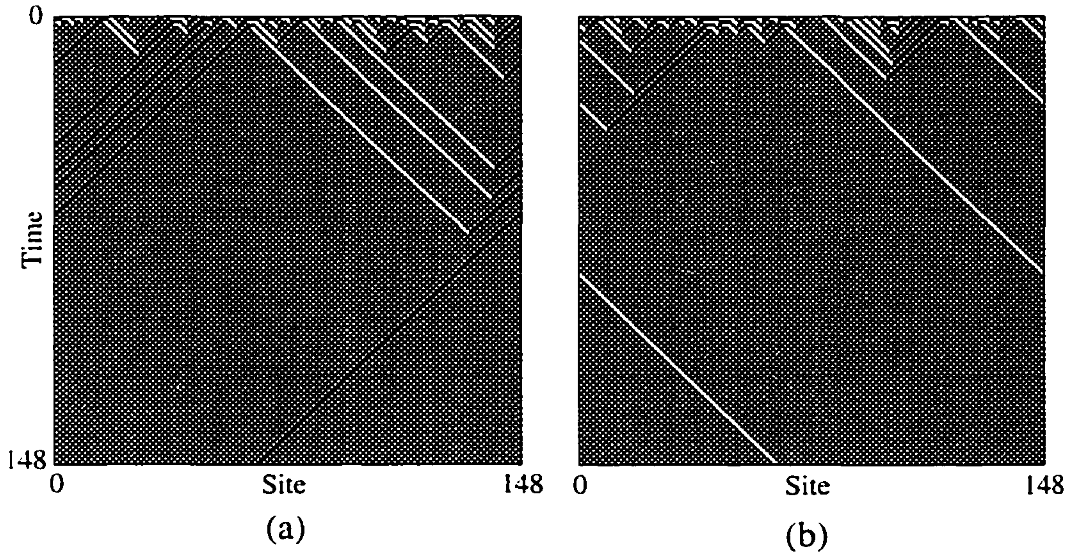


Figure 2.2: Space-time diagram of ECA 57 starting with two different randomly generated ICs. The final configurations in (a) and (b) consist of black glider and white glider respectively.

It turns out however, that from a computation theoretic standpoint, it is possible to ascribe some computation in ECA 57's space-time behavior. Investigations of ECA 57's behavior discloses that a final configuration has one or more black (or white) gliders if and only if in the initial condition there are more (or fewer) 1-blocks of even length than 0-blocks of even length. Since a finite automata will require a counter to determine the excess of 1-blocks of even length over 0-blocks of even length (or vice versa), the computational task can be defined in terms of a particular context-free grammar. Thus ECA 57 performs a mapping from a context free language to one of the two regular languages $(11)^+ \cup (01)^+$ or $(00)^+ \cup (01)^+$ and consequently does some non-trivial classification. Once Crutchfield and Hanson's approach is used to discover and quantify the computational structures in ECA 57's behavior, the role of the different computational elements in accomplishing this classification can be ascertained, and the functionality of the gliders and the checkerboard can be delineated.

The preceding discussion presents an example of a CA whose global behavior can be interpreted as implementing a specific computation. But what if we wanted to design a CA that performs some other specific computation that is useful to us? Our work uses genetic algorithm to design CAs whose usefulness is measured in terms of how well the CAs perform a given task. To understand the behavior of high-performance CAs discovered by evolution we adopts the computational mechanics framework.

Chapter 3

THE COMPUTATIONAL TASKS

Two different computational tasks for cellular automata will be considered in this dissertation: a density classification task and a synchronization task. The choice of these two tasks was influenced by several factors. Although the two tasks are easy to define. it is not apparent how to design one-dimensional. small radius ($r \ll N$), two-state CAs To perform them. As detailed in Chapter 2. each site in a CA has access only to spatially local information and can only influence the behavior of its neighbors. The tasks are fundamentally difficult for a CA to perform because each task involves complete global coordination requiring long-range information processing and transmission.

3.1 Density Classification

In this task. the density $\rho(\mathrm{s})$ refers to the fraction of 1s in a configuration s. For an IC s_0, the shorthand notation ρ_0 will be used for the density $\rho(\mathrm{s}_0)$.

The particular density classification problem studied in this work is defined as follows. If the initial density ρ_0 in a given IC s_0 is more than a critical density ρ_c, then within M time steps, a successful CA for this task should relax to a fixed-point configuration 1^N; otherwise, the CA should relax to a fixed-point configuration 0^N. M is a parameter of the task that depends on the lattice size N. On a spatially periodic lattice of length N, this task can be formally defined as

$$\mathrm{T}_{\rho_c}(\mathrm{s}_0, N, M) = \left\{ \begin{array}{ll} \Phi^{M+i}(\mathrm{s}_0) = 1^N & \text{if } \rho_0 > \rho_c \\ \Phi^{M+i}(\mathrm{s}_0) = 0^N & \text{if } \rho_0 < \rho_c \end{array} \right\} \forall\, \mathrm{s}_0 \in \{0,1\}^N \;\&\; \forall\, i \in \{0,1,\ldots,\infty\}. \tag{3.1}$$

By choosing appropriate lattice size N, the behavior of the CA for $\rho_0 = \rho_c$ need not be defined.

For the density classification task $\mathbf{T}_{1/2}$, the critical density ρ_c is fixed at 1/2 and the lattice size N is restricted to odd values. A successful CA for the $\mathbf{T}_{1/2}$ task can be thought of as a mapping device which accepts an IC as an input and classifies it according to its density ρ_0 into one of the two "goal" fixed-point configurations, 0^N or 1^N. From this perspective, the global equation of motion Φ of the CA over M time-steps can be interpreted as the program which governs the mapping from an input to an appropriate output configuration.

For the $\mathbf{T}_{1/2}$ task, the *performance*, $\mathcal{P}_K^N(\phi)$, of a given CA, ϕ, on a lattice of size N is defined to be the fraction of K randomly chosen initial configurations on which ϕ produces the correct final configuration within M time steps. All performance measurements given hereafter are made with $K = 10^4$, $M \approx 2N$, and $N \in \{149, 599, 999\}$. In general, the performance of a CA on these three values of N gives a good idea how the performance scales with lattice size.

The $\mathbf{T}_{1/2}$ task is interesting because it is closely related to image processing tasks such as filtering or logical convolutions [PD84]. In these tasks, a "kernel" function is used to define a two-dimensional local neighborhood around each point in an image. To process an entire image, a logical transform, such as thresholding, is simultaneously applied to the neighborhood of each point in the image. Often, a series of different kernels and logical transforms are successively applied on an image for finer processing. In a simpler context, our study of one-dimensional CAs to perform simple tasks such as $\mathbf{T}_{1/2}$ is a preliminary step in the understanding and design of CAs which can perform image processing tasks in two or more dimensions.

The density classification task is also of interest in studies of reliable computation. Issues related to the avoidance of accumulation of errors in arbitrarily large computations using unreliable components are of fundamental importance in computer science. One of the goals in such studies is to design a spatially-homogeneous distributed medium that can store information even though each element in the medium makes an error with some constant probability at each time step. In the model analyzed in [GKL78], a single bit of information—a zero or a one—was encoded in the state of a CA configuration consisting of 0^N or 1^N respectively. The aim was to design a CA rule such that any perturbation in the 0^N (1^N) configuration will die out, with the CA returning to the 0^N (1^N) configuration. Under such a rule, the CA would reliably store information in spite of random error in rule update. In the absence of new errors in rule update, a CA which is maximally reliable would restore any configuration which is less than $N/2$ in Hamming distance from 0^N (1^N) back to the fixed-point configuration of 0^N (1^N). Such a CA would therefore also exhibit high performance for the $\mathbf{T}_{1/2}$ task.

Although the $\mathbf{T}_{1/2}$ task is simple to define, it is nontrivial for a small-radius ($r \ll N$) CA. Since density is a global property of a configuration, whereas a small-radius CA relies only on local interactions mediated by the cell neighborhoods, it is not apparent how to design a CA for the $\mathbf{T}_{1/2}$ task. The task is difficult because it requires the implementation of a prespecified global mapping in terms of the local mapping in a CA rule table. Moreover, since 1s (say) can be distributed throughout the lattice, the CA must transfer information over large space-time distances ($\approx N$). As an example, consider an IC with $\rho_0 < 1/2$ such that the addition of a single 1 in the configuration increases the value of ρ_0 to more than 1/2. As a result of this infinitesimal modification—which could occur anywhere in the lattice—the desired fixed-point configuration is changed from 0^N to 1^N. Therefore, as a necessary precondition to the change in the fixed-point configuration, information about the new site value has to be propagated throughout the lattice and affect each site value in the final configuration.

For a Turing machine that accepts the IC as the input tape, the minimum amount of memory required for the $\mathbf{T}_{1/2}$ task is proportional to $log(N)$, since the equivalent of a counter register is required to track the excess of 1s in a serial scan of the IC. In other words, the task requires computation which corresponds to the recognition of a non-regular language [MCH94b]. In fact, it can be shown that no finite radius CA can perform this task perfectly across all lattice sizes (see Appendix A). For a fixed lattice size, even performing this task well (with, say, $\mathcal{P}_K^N(\phi) > 0.5$) requires more powerful computation than can be performed by a single cell or any linear combination of cells.

A study of the $\mathbf{T}_{1/2}$ task [MCH94b] shows that any CA ϕ that performs $\mathbf{T}_{1/2}$ perfectly, must possess two symmetries:

1. If an IC $\mathbf{s}_0$ is spatially reversed on the lattice, then there is no change in the classification obtained by ϕ. Thus the global equation of motion engendered by ϕ must meet the condition

$$\Phi^{M+i}(\mathbf{s}_0) = \Phi^{M+i}(\mathcal{R}\mathbf{s}_0) \;\; \forall \; \mathbf{s}_0 \in \{0,1\}^N \;\; \& \;\; \forall i \in \{0,1,\ldots,\infty\}. \quad (3.2)$$

where the operator $\mathcal{R}$ reverses the order of the bits in a configuration $\mathbf{s}$.

2. If all the bits in an IC $\mathbf{s}_0$ are flipped (i.e., 1s are replaced with 0s, and 0s with 1s), then ϕ must give the opposite classification. Thus the global equation of motion engendered by ϕ must meet the condition

$$\Phi^{M+i}(\mathbf{s}_0) = \mathcal{C}\Phi^{M+i}(\mathcal{C}\mathbf{s}_0) \;\; \forall \; \mathbf{s}_0 \in \{0,1\}^N \;\; \& \;\; \forall i \in \{0,1,\ldots,\infty\}. \quad (3.3)$$

where the operator $\mathcal{C}$ flips the bits in a configuration $\mathbf{s}$.

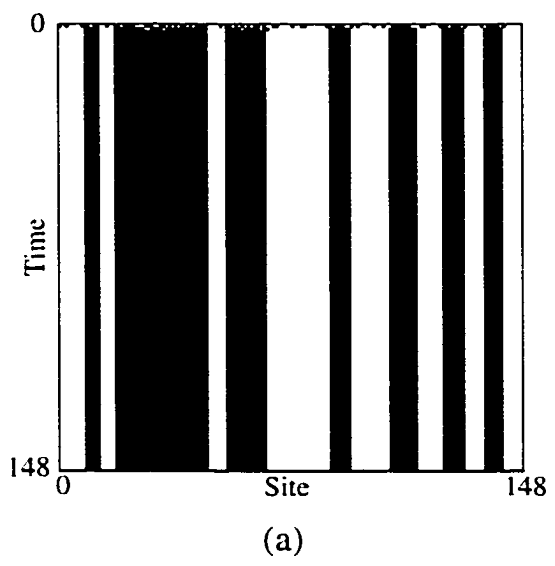
A CA can satisfy the two symmetry constraints either in its local interactions (i.e., in the individual neighborhood mappings in the rule table ϕ) or in its global mapping (defined by the global equation of motion Φ), or both.

A successful CA that performs the $\mathbf{T}_{1/2}$ task must obey some additional constraints on its global equation of motion Φ.

$$\text{If } \rho(\mathbf{s}) < \rho_c, \text{ then } \rho(\Phi^i(\mathbf{s})) < \rho_c \;\; \forall \; i \in \{0,1,\ldots,\infty\} \quad (3.4)$$

$$\text{If } \rho(\mathbf{s}) > \rho_c, \text{ then } \rho(\Phi^i(\mathbf{s})) > \rho_c \;\; \forall \; i \in \{0,1,\ldots,\infty\} \quad (3.5)$$

The validity of the above constraints can be proven by contradiction. Suppose there exists a rule ϕ_p which correctly classifies all ICs. For a given configuration s, if $\rho(\mathrm{s}) < \rho_c$, then by definition, under ϕ_p's global equation of motion, $\rho(\Phi^M(\mathrm{s})) = 0.0$. Now, also suppose that ϕ_p violates the first constraint, and thus, $\rho(\Phi^t(\mathrm{s})) > \rho_c$ for some $t \geq 1$. Then, by definition $\rho(\Phi^M(\Phi^t(\mathrm{s}))) = 1.0$. This leads to a contradiction, since ϕ_p classifies the configuration s with $\rho(\mathrm{s}) < \rho_c$ into the fixed-point configuration of 1^N. Thus, a perfect rule for the $\mathrm{T}_{1/2}$ task must satisfy the first constraint. Using similar logic, the validity of the second constraint can be shown.



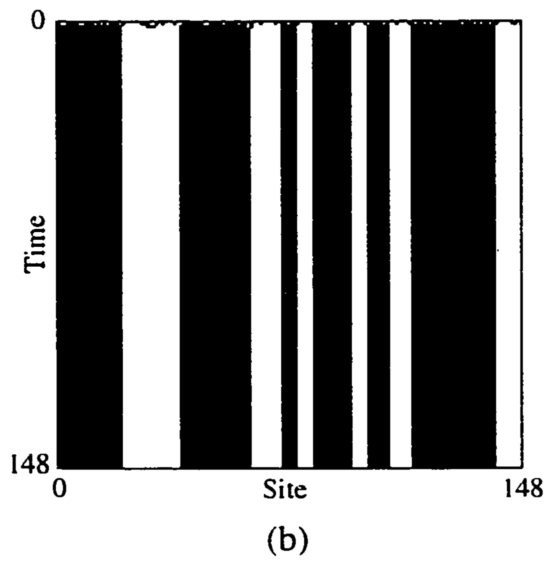


Figure 3.1: Two space-time diagrams for the majority rule CA, ϕ_{maj}. In (a), $\rho_0 \approx 0.45$, and in (b), $\rho_0 \approx 0.55$. It should be noted that both in (a) and (b), the CA has failed to reach the correct goal configuration of 0^{149} and 1^{149} respectively.

As a starting benchmark, consider a naive candidate solution ϕ_{maj} for $\mathrm{T}_{1/2}$ task. ϕ_{maj} uses majority voting among the site values in the parent neighborhood to determine the corresponding output bit. For an $r = 3$ CA, ϕ_{maj}'s look-up table is defined as follows

$$s_{t+1}^i = \phi_{majority}(\eta_t^i) = majority[s_t^{i-3}, s_t^{i-2}, s_t^{i-1}, s_t^i, s_t^{i+1}, s_t^{i+2}, s_t^{i+3}] \qquad (3.6)$$

In words, this rule says that for each neighborhood η^i of seven adjacent cells, the new state of the central cell is decided by a majority vote among itself and its six neighbors. From equation 3.6, it can be determined that the candidate rule ϕ_{maj} satisfies the two symmetry constraints (equations 3.2 and 3.3) in its parent neighborhood mappings. Typically, ϕ_{maj} also obeys the two constraints on its global equation of motion as defined in equations 3.4 and 3.5.

In spite of the above arguments in its favor, ϕ_{maj} fails to perform the $\mathbf{T}_{1/2}$ task. The performance $\mathcal{P}_{10^4}^{N}(\phi_{majority})$ was measured to be 0.000 for each of the three lattice sizes $N = 149, 599$, and 999.

Figure 3.1 helps to illustrate why ϕ_{maj} is not successful. The figure depicts its space-time behavior, starting from two ICs with different ρ_0. In each case, the CA fails to relax to the correct goal configuration. After a very brief transient period, the CA configuration is frozen in a temporally invariant but spatially inhomogeneous configuration consisting of blocks of 0s or 1s. Under ϕ_{maj}, any block of 1s (or 0s) of length greater than r cannot decrease in length. Among all possible ICs in $\{0, 1\}^N$, the fraction of ICs without blocks of 1s *and* 0s of length greater than 3 tends to 0 with increasing N. Thus, for almost all ICs the application of ϕ_{maj} rule results in unclassified final configurations consisting of blocks of both 1s and 0s.

Interestingly, there exist trivial rules whose performance $\mathcal{P}_k^N(\phi) = 0.5$ for any odd lattice size N. In other words, these trivial rules have superior performance than ϕ_{maj}. Consider a rule ϕ_0, in which all neighborhoods are mapped to the output symbol 0. As a result, starting from any IC, ϕ_0 relaxes to the fixed-point configuration of 0^N in a single time step. Since ICs are randomly sampled to measure the performance a rule, ϕ_0 correctly classifies half of the ICs. Similarly, rule ϕ_1—in which all neighborhoods are mapped to the output symbol 1—has performance $\mathcal{P}_k^N(\phi_1) = 0.5$.

As mentioned earlier, it is already known that there is no finite radius CA that performs $\mathbf{T}_{1/2}$ perfectly. But are there CA rules that can perform significantly better than ϕ_0 or ϕ_1 on $\mathbf{T}_{1/2}$? If there are such rules, to what extent do they satisfy the constraints inherent in the task? More importantly, what are the mechanisms of information processing and information transmission in such CAs? The subsequent chapters address these issues in detail.

3.2 Synchronization

The goal in this task is to find a CA look-up table ϕ that, given any initial configuration s_0, within M time steps reaches a final configuration that oscillates between the 0^N and 1^N configurations on successive time steps. The synchronization task $\mathbf{R}$ is formally defined as

$$\mathbf{R}(s_0, N, M) = \left\{ \begin{array}{l} \Phi^M(s_0) = 1^N \text{ and } \Phi(1^N) = 0^N \\ or \\ \Phi^M(s_0) = 0^N \text{ and } \Phi(0^N) = 1^N \end{array} \right\} \forall s_0 \in \{0, 1\}^N \qquad (3.7)$$

M, the desired upper bound on the synchronization time, is a parameter of the task that depends on the lattice size N. This is perhaps the simplest nontrivial synchronization task for a CA.

Implicit in the definition of a CA is a globally synchronous update clock. That is, a CA's local states are updated at the same time across the lattice. But, since each site has a "processor" ϕ which determines local behavior and site-to-site interactions, the effect of the underlying global update need not be manifested directly in globally synchronous configurations (e.g., ECA rule 90's behavior is "chaotic").

One of the earliest mathematical articulations of a similar nontrivial spatial synchronization problem in a distributed system—the firing-squad synchronization problem (FSSP)—uses a globally synchronous update clock. As in the $\mathbf{R}$ task, in spite of the global update mechanism, it is the site-to-site interactions among the individual processors that makes the FSSP interesting and difficult. Although FSSP was first proposed by Myhill in 1957, it is still being actively studied [Yun94].

In the FSSP one considers a finite, but arbitrarily long ordered line of identical cells, numbered from 1 to N. The state of a cell at time $t+1$ depends on its own state and the state of its immediate neighbors at time t. All the cells are updated synchronously. Given an IC in which all cells, except the leftmost cell (the "General"), are in the quiescent state, the task in the FSSP is to design a CA rule such that all the cells will enter a special state ("fire") for the first time at the same time step. By constraining the rule table, it is guaranteed that the General remains in the leftmost cell, and the number of Generals remains constant at one. Thus the resulting CA is spatially inhomogeneous. It is easy to show that the minimal-time solution to the FSSP requires at least $2N - 2$ time steps. Among the various solutions to the FSSP, Mazoyer's approach requires the least number (6) of states per cell [Maz87]. Various generalizations of the FSSP have been proposed in the literature, including multi-dimensional versions of the problem [Cul93].

Although the FSSP and the **R** task are similarly motivated, there are three important differences.

1. In the solution to the FSSP proposed by Mazoyer, the number of states per cell k is 6, and the number of possible neighborhood configurations is $6^{(2r+1)} = 216$ for $r = 1$. The **R** task deals with two-state CAs, and thus the number of possible neighborhood configurations is $2^{(2r+1)} = 8$ for $r = 1$. In this work, we mainly focus on CAs with $r = 3$, which allow 128 different neighborhood configurations.

2. The FSSP considers only those ICs in which all cells, except the general, are in the quiescent state. There is no such restriction on the ICs for the **R** task. A successsful CA for the **R** task must attain global synchronization starting from all possible ICs.

3. In the FSSP, all the cells enter the firing state for the first time at the same time step. Thus local information is sufficient to determine whether the system has attained synchronization or not. In other words, if a cell is in the fire state, it "knows" that the whole system is in synchrony. In the **R** task, there is no special fire state, As a result, the cells in

the system have no way of "knowing" whether or not the whole system has attained synchrony. Instead, information about synchronization is a property of the system's global configuration.

For the **R** task, the *performance* $\mathcal{P}_K^N(\phi)$ of a given CA ϕ on a lattice of size N is defined to be the fraction of K randomly chosen initial configurations on which ϕ produces the correct final sequence of configurations after at most M time steps. As in the case of the density classification task $\mathbf{T}_{1/2}$, all performance measurements given for **R** are made with $N \in \{149, 599, 999\}$, $K = 10^4$, and $M \approx 2N$.

Like $\mathbf{T}_{1/2}$, **R** is nontrivial since synchronous oscillation is a global property of a configuration, whereas a small-radius (e.g., $r = 3$) CA employs only local interactions mediated by the sites' neighborhoods. Thus, while the locality of interaction can directly lead to regions of local synchrony, it is substantially more difficult to design a CA that will guarantee that spatially distant regions are in phase. Since regions that are not in phase can be distributed throughout the lattice, a successful CA must transfer information over large space-time distances ($\approx N$). In order to produce a globally synchronous configuration a CA must to remove phase defects—borders separating regions that are locally synchronous.

For reference, we consider a naive candidate solution ϕ_{osc}, a $r = 3$ CA whose look-up table is defined by:

$$\phi_{osc}(\eta) = \begin{cases} 1 & \text{if } \eta = 0000000 \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

The output-bit settings in ϕ_{osc} ensure that $\Phi(1^N) = 0^N$ and $\Phi(0^N) = 1^N$. The performance $\mathcal{P}_{10^4}^N(\phi_{osc})$ was measured to be 0.54, 0.09, and 0.02, for $N = 149$, 599, and 999, respectively. This shows that although ϕ_{osc} is able to reach the globally synchronous state for roughly half the ICs in a lattice with $N = 149$, its performance decreases dramatically for larger lattice sizes.

ϕ_{osc} is an unsuccessful synchronizer precisely because it is unable to remove phase defects. This is readily apparent in Figure 3.2, a space-time diagram from

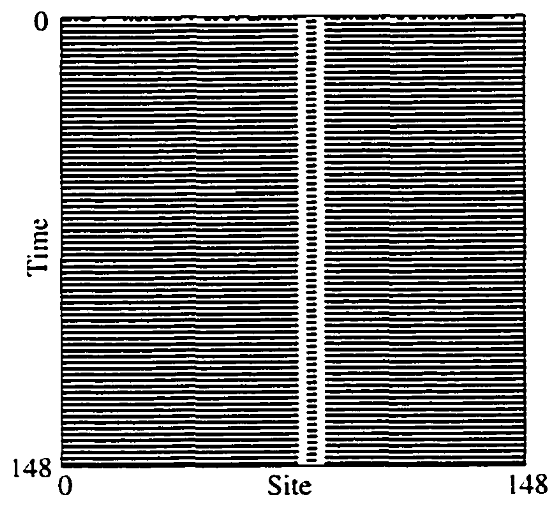


Figure 3.2: Typical space-time diagram produced by ϕ_{osc}. It should be noted that ϕ_{osc} is unable to remove the two phase defects—borders separating regions that are locally synchronous.

ϕ_{osc}. In the figure, there are two phase defects which separate regions that are locally synchronous but out of phase with respect to each other. Such phase defects occur when the IC has at least one block of 0s of length greater than the neighborhood size $(2r+1)$. Among all possible ICs in $\{0,1\}^N$, the fraction of ICs without blocks of 0s of length greater than or equal to 7 tends to 0 with increasing N. Thus, the performance of ϕ_{osc} drops sharply with increase in the lattice size.

The inferior performance of both ϕ_{maj} and ϕ_{osc} for their respective tasks are a result of similar limitations in their spatio-temporal behavior. In each case, the global equations of motion result in noninteracting sub-regions within the lattice. Since these sub-regions persist indefinitely over time, there is no information transmission across the system. Thus, for both tasks $\mathbf{T}_{1/2}$ and $\mathbf{R}$, more sophisticated CAs must be found which must at least be capable of engendering long-range information transmission. This information then can be used to produce the desired global configurations.

Chapter 4

GENETIC ALGORITHMS: AN OVERVIEW

From the very dawn of the computer age, computer scientists have been deeply interested in biological systems as guiding metaphors for developing computer systems with biological properties such as intelligence and self-reproduction. At the present time, a resurgence of interest is being witnessed in the use of biological paradigms for the development of computer systems. One such approach has developed into the field of "neural networks" [HKP91]. Researchers in this field aim to build intelligent systems by studying the neurons and the neuronal connections in the brain as the main inspiring metaphor.

A very different metaphor is adopted in the field of "evolutionary computation" which derives inspiration from the paradigm of biological evolution. Genetic algorithms (GAs), which are non-deterministic search strategies based on concepts abstracted from evolution, provide the most notable example of this evolutionary computation approach. GAs were first developed by John Holland and his colleagues in the 1960s. In *Adaptation in Natural and Artificial Systems* [Hol75], Holland presented GAs as an abstraction of natural evolution and provided a theoretical framework to study the mechanisms that guide adaptation in a GA. Since the mid 1980s, GAs have garnered wide scientific interest and have been successfully applied in a large number of complex computational problems in science and engineering [Gol89, Dav91, Mit92].

4.1 The Evolutionary Paradigm

What are the factors that make it appealing to use evolution as a paradigm to help in solving difficult computational problems? One of the primary reasons is that some of the main mechanisms of biological evolution can be easily adapted to find solutions to complex computational problems. In many such problems, the goal is to find the optimal (or a near-optimal) solution among a vast number of possible solutions. For example, consider the problem of turbine blade design. Given that there are a large number of control parameters which determine the shape, size, and other physical properties of a turbine blade, how does one search for the set of parameters that result in the most energy efficient turbine blade? A similar scenario is encountered in the problem of computational drug design where the aim is to find a protein, enzyme or a antibody with a set of desired properties. The properties of such biochemicals are primarily determined by the underlying amino-acid sequence. Since there exists an enormous number of possible amino-acid sequences, it is not apparent how to efficiently search through all the possibilities to determine the right protein.

In addition to the large search space, problems such as the ones described above are difficult for several other important reasons. In these problems, the fitness of a candidate solution is determined by a large number of parameters or features, and it is often difficult to determine which features are responsible for superior performance. Moreover, the effect of one feature on the overall fitness may depend strongly upon what other features are present in the candidate solution. As a result, the fitness measure is often a complicated, nonlinear and high-dimensional function of the underlying parameters. The presence of local optima and sudden discontinuities in such functions make them very difficult to search using traditional gradient-based algorithms. Finally, in some problems, the fitness measure might be a noisy function and may also vary over time and space. In order to find the optimum. such problems require algorithms which are adaptive and responsive to the continual changes occurring in the search space.

In trying to understand these difficulties and obtain a wider perspective, one notices that biological evolution encounters strikingly similar impediments. And yet, evolution is able to produce increasingly fit organisms in uncertain and complex environments which may vary over time and space. Biological evolution (also referred to as Darwinian evolution or natural evolution) works on a population of organisms in which the fittest survive and are able leave behind offspring into the next generation. The fitness of an organism depend on its *phenotype*, i.e., the set of characteristics or attributes that help an organism to survive in an environment. These characteristics are in turn determined by the *genes* residing in the chromosome of the organism. Each gene can attain several different forms or alternatives—called *alleles*—resulting in differences in the set of phenotypic characteristics associated with that gene. The entire genetic makeup of an organism is called the *genotype* of that organism. Since higher order organisms contain on the order of 2^{10000} genes, the number of possible genotypes is indeed astronomical. This gives an idea of the enormous size of the space through which natural evolution has to perform its search. Moreover, since the effects of genes are in general not additive, the search for fitter organisms becomes considerably more difficult. Due to the interactions and the interdependence among the genes—a phenomenon known as *epistasis*—the mapping from a genotype to its phenotypic characteristics is very complicated. As a result, it is difficult to apportion credit to the individual alleles that are actually responsible for superior fitness. In the presence of all these impediments, how does natural evolution discovers fitter organisms?

In his book *Adaptation in Natural and Artificial Systems*, Holland argued that natural evolution is able to overcome these problems by primarily searching for "coadapted" sets of interdependent alleles which together result in improved performance of the corresponding phenotype. Thus instead of searching for a single highly fit genotype by testing candidate solution one at a time, biological evolution searches for coadapted sets of alleles among the individuals in a population. The presence of a large population allows the search to proceed in parallel along many different paths. At the same time, genetic recombination and mutation help in discovering new sets of coadapted alleles. Holland formalized the concept of a coadapted set of alleles under the term *schema* and used it to provide a theoretical framework for his genetic algorithm.

4.2 A Simple Genetic Algorithm

A GA maintains a population of *chromosomes*, and employs a set of genetics-inspired operators to create a new population from the existing one. Each chromosome in a population represents a candidate solution and is encoded in a string of bits. Under a binary representation scheme, each position (*locus*) in a string has two possible values (*alleles*), 0 and 1. Borrowing vocabulary from biology, the chromosome of a particular candidate solution defines the *genotype* of that individual. Each individual in the population is also associated with a fitness, i.e., a measure of how well the individual solves the problem at hand. The fitness of an individual is determined by its features or characteristics, i.e., its *phenotype*.

As an example, consider the problem of maximizing the function $F(x) = x^3 - 5x^2 + 13$, where x is an integer $\in [0, 31]$. Under a binary encoding scheme, the entire domain of the solution space can be represented with unsigned binary integers of length five, since there are 2^5 candidate solutions. Thus a candidate solution $x = 15$ can be represented with the binary string 01111. In this problem, the fitness of the chromosome 01111 is $F(01111) = 225$.

A simple GA uses three type of operators which operate on chromosomes in a population:

Selection: The individuals in the population undergo a fitness based *selection* process, where strings encoding fitter solutions are allowed to reproduce preferentially into the next generation. Thus, fitter strings on average send more offspring into the next generation than less fit ones.

Crossover: The crossover operator, which loosely mimics biological recombination, proceeds with a pair of parent chromosomes. A crossover locus within the chromosome is chosen at random, and the subsequences on either side of the locus are exchanged between the two parents. For example, consider the following two parent chromosomes P_1 and P_2 undergoing the crossover operation.

$$P_1 = 11011|1101$$
$$P_2 = 00000|0000$$

When the crossover site falls after the fifth locus (represented with the separator symbol |), the crossover operation results in the two child chromosomes C_1 and C_2:

$$C_1 = 110110000$$
$$C_2 = 000001101$$

Typically, the crossover operation is performed with a certain probability p_c for every pair of mated parent chromosomes.

Mutation: In the case of binary strings the mutation operator flips the bits in the strings with a very low probability p_m. For example, when the chromosome 110110000 undergoes a mutation at the third locus, it results in the chromosome 111110000.

For a given problem, a simple GA works by sequentially following the steps given below. (Although a bit-string representation for candidate solutions has been adopted here, an extension to a larger alphabet size is relatively straightforward.)

1. The initial population consists of P chromosomes. Each chromosome is made up of a string of L bits and it represents a candidate solution x in the search space. Each allele value in a chromosome is randomly generated, i.e., the allele value can be a 0 or a 1 with equal probability. The generation counter is initially set to zero.

2. The fitness $F(x)$ of each chromosome x in the population is evaluated.

3. Steps (i)-(iii) are repeated until the total number of offspring produced is equal to the population size P:

   (i) A pair of parent chromosomes are selected from the current population. The probability of selecting a chromosome is directly proportional to its fitness. Since the chromosomes are selected with replacement, a chromosome with above average fitness can be selected as a parent on more than one occasion.

   (ii) The crossover operator is applied to the two parent chromosomes with a probability p_c. It results in two new offspring.

   (iii) The mutation operator is applied to each locus in both child chromosomes with a very small probability p_m. The resulting child chromosomes are placed in the new population.

4. The current population is replaced by the new population. The generation counter is simultaneously incremented by one.

5. The process is stopped if the number of generations has exceeded a predetermined limit; else the evolutionary cycle continues by returning to step 2.

In practice, when applying a GA to solve a real world problem, researchers typically use more complicated versions of the genetic algorithm than the simple algorithm presented here. Nevertheless, most genetic algorithms follow the same basic steps outlined above. In the recent past, GAs have been used in a wide range of engineering problems requiring numerical as well as combinatorial optimization. [For93, BB91, Sch89, Gol89, Dav91]. GAs have also been used as models of biological evolution to study immune systems, economic models, and ecological systems [For90, HM91, Mit92].

The apparent success of GAs in these diverse fields raises the question: What features in a GA are responsible for its apparent success? Although the basic steps in a GA are simple to describe, the behavior of a GA is often very complicated. As a result, there is considerable debate regarding the underlying mechanisms which lead to the discovery of high performance solutions in a GA.

The earliest effort to explain the behavior of a GA is due to Holland in which he emphasized the notion of a *schema* or a building block. One of Holland's main assumptions is that the genotype of a high performance solution is made of good building blocks, i.e., sets of alleles which together confer higher fitness on the chromosome in which they are present. Holland proposed that the success of a GA in discovering good solutions lies in the GA's ability to efficiently search for good building blocks or schemata. A schema represents a set of bit-strings that share some common feature. A schema can be denoted as a bit string consisting of the symbols 0, 1, and #, where # is the wild card symbol. As an example, the schema s = 11### represents all bit strings of length five which begin with a pair of 1s. The number of defined bits (i.e., locii with a non-# symbol) in a schema defines the *order* of the schema. The *defining length* of a schema is the distance between the outermost defined bits. Holland focused on these two properties in his schema analysis.

It is easy to show that any given bit string of length L is an instance of 2^L distinct schemata. For example, the string 01 is an instance of the following four schemata: 01, 0#, #1, and ##. Thus, a GA with a population of P L bit-strings can contain up to $P \times 2^L$ schemata. More importantly, when a GA evaluates the fitness of the P chromosomes, it *implicitly* estimates the average fitness of a much larger number of schemata at the same time. The selection process allows fitter schemata from one generation to send more copies of themselves into the next generation. It is important to note that while the selection operator is only applied at the chromosome level, selection at the schema level occurs automatically. By recombining the existing high-performance schemata in a population, the crossover operator provides the primary vehicle through which new schemata with even better performances are discovered in a GA. In his GA, Holland envisioned that the main role of the mutation operator was to guarantee genotypic diversity in a finite population. While the mutation operator certainly plays this role, research shows that this operator can also aid in the discovery process by making a sequence of small adjustments to the chromosome where each step in the sequence leads to an improvement in the fitness [SCED89]. In other words, the mutation operator may often allow a GA to "hill climb" to the optimum.

Taking into account the effects of the various operators in a GA, Holland provided a theoretical framework—known as the Schema Theorem—to delineate how a GA biases the number of instances of a schema in a population from one generation to the next. Given a schema, its current fitness in the population, and the number of instances of the schema in the current population, the theorem provides a lower bound on the number of copies that can be expected to be found in the next generation. In essence, the Schema Theorem says that the GA will allocate an exponentially increasing number of samples to low-order short-defining length schemata with above average fitness. This allows the crossover operator to recombine short low-order schemata and discover new high-fitness schemata which subsequently undergo the selection process themselves.

The Schema Theorem has been the focus of much research in the GA community over the past decade. However, several researchers have criticized the schema theorem for its simplistic assumptions which are rarely true in realistic problems. Others have studied GAs using different techniques such as Markov models [Vos93] and statistical mechanics approaches [PBS94]. Nevertheless, Holland's schema theorem remains the most well-studied framework to analyze the mechanisms that drive a GA [FM93b, FM93a, MHF].

4.3 Using GAs to Study Natural Evolution

In this dissertation, we use a GA to design CAs that can perform a given computational task. It needs to be reemphasized that the GA is provided information about *what* is to be done rather than *how* to perform the tasks. In addition to emphasizing the use of a GA in automatically discovering CAs, this work also focuses on the evolutionary dynamics in a GA that lead to the discovery of high-performance CAs. Thus, we are also interested in GAs as computational models to understand biological evolutionary processes.

Among the problems which have historically been of the greatest interest in the study of evolutionary theories, the origin and nature of morphological novelty in organisms undergoing biological evolution occupies an important place. An example of morphological novelty in evolution is the development of fins for locomotion in early vertebrates. There are several schools of thought on the guiding mechanisms in evolution that are responsible for the origin of novel biological structures and the subsequent adaptation of those structures [Wil93, Boc95, Cru94]. The neo-Darwinist school [Boc95] maintains that the order and form observed in biological organisms is a result of (i) fitness-based selection of individuals, and (ii) genetic variations that lead to diversification. Proponents of this school suggest that the above two processes are sufficient to gradually alter one form of structure into a different form.

A rival school of thought, mainly lead by Stephen J. Gould and Nigel Eldredge, asserts that while the processes of selection and genetic variation are important, major changes in structure are often solely due to historical accidents [GE93]. It is argued that although an existing structure may play a major role in an organism's ability to survive, at the time when the structure appeared it might have conferred no survival benefits. In their theory of *punctuated equilibria*, Gould and Eldredge have claimed that biological evolution is not gradual, but episodic, with long periods of stasis interrupted by bursts of evolutionary activity.

More recently, a different perspective has been offered by a third hypothesis of thought which seeks to delineate the "principles of organization" in biological evolution [Goo92, FB94, HW93]. This school maintains that despite the extreme genetic diversity of organisms, the interactions between the genes are constrained such that the range of possible biological forms is fundamentally limited to a relatively small set of structural prototypes. In the course of evolution, the different structural prototypes act as "attractors" waiting to be filled. Which structural prototype is finally achieved through the course of evolution is primarily determined by selection, historical accidents or both.

With these different viewpoints in mind, this work attempts to gain deeper insights into the origin of structural and functional novelty in systems undergoing evolution. In particular, we seek to understand the origin and nature of computational elements in a CA which are selected by a GA to perform the computational tasks. Also, our goal is to understand the evolutionary pathway through which a GA manipulates the discovered computational elements to achieve improved performance.

Chapter 5

EVOLVING CELLULAR AUTOMATA WITH A GENETIC ALGORITHM

Our approach in using GAs to evolve CAs to perform a computation task can also be viewed as an automatic programming technique. Similar automatic techniques have been used in the past to design classifier systems and neural networks in machine learning [WS92, Gru92, MTH89, WDD91], and to construct high-performance sorting networks [Hil90]. A study of these works shows that while the GA is often successful in designing the above complex systems, much remains to be discovered and learned. In particular, further research is necessary in determining the features in these problems that either help or impede the GA in discovering and adapting high-performance solutions. In this work, we attempt to understand these issues in the context of a GA evolving CAs.

5.1 Previous Work

A part of the work described in this dissertation is the extension of earlier work by Mitchell, Crutchfield, and Hraber [MHC93, MCH94a, MCH94b] on evolving CAs. Mitchell et al. were initially interested in this problem to verify earlier experiments by Packard where the latter used a GA to evolve one-dimensional CAs. Packard designed his experiment to test the "edge of chaos" hypothesis which claimed that a GA evolving CA rules to perform a nontrivial computational task will tend to select rules near conjectured phase transitions in rule space. Behavior that was between ordered and chaotic was expected from the high-performance rules. In these experiments, the fitness of a CA was determined by its ability to perform the density classification task $\mathbf{T}_{1/2}$. In [Pac88], Packard claimed that in his experiments the GA indeed found rules near this phase transition and offered these results to support the edge of chaos hypothesis. However, Mitchell, Crutchfield, and Hraber re-examined Packard's claims, and provided both theoretical arguments and experimental evidence which did not support the edge of chaos hypothesis [MHC93]. Mitchell, et al., claimed that Packard's results were an artifact of the particular GA he used in his experiments. In addition, Mitchell, et al., found no evidence that rules near the conjectured phase transition are computationally more capable in solving the density classification task.

5.2 Details of the Experimental Setup

The GA used in these experiments was modeled after earlier work on density classification by Mitchell, Crutchfield, and Hraber [CM94, MHC93]. The GA begins with a population of P randomly generated "chromosomes"—bit strings encoding CAs by listing each CA's output bits in lexicographic order of neighborhood configuration. For binary $r = 3$ CAs, the chromosomes are of length 128 ($= 2^{2r+1}$). The size of the space the GA searches is thus 2^{128}—far too large for any kind of exhaustive search.

Like the performance measure $\mathcal{P}_K^N(\phi)$, the *fitness* of a chromosome ϕ is estimated by fixing the lattice size, selecting a random sample of initial configurations, interpreting the chromosome as a CA look-up table, and iterating this look-up table on each IC in the sample until the desired behavior is reached or the CA has iterated for a maximum of M time steps. The fitness value $F_I(\phi)$, is the fraction of I ICs on which the CA produces the correct final configuration or sequence of configurations. In the case of the density classification task $\mathbf{T}_{1/2}$, the correct final configuration is 0^N if $\rho_0 < 1/2$; otherwise the final configuration is 1^N. The initial density ρ_0 is never exactly $1/2$, since N is chosen to be odd. No partial credit is given for a partially correct final configuration. In the synchronization task $\mathbf{R}$, the desired dynamics in final configurations must consist of oscillations between 0^N and 1^N. No partial credit is given for incompletely synchronized final configurations.

Our fitness measure, $F_{100}(\phi)$, differs from $\mathcal{P}^N_{10^4}(\phi)$ in the following ways: (1) For computational tractability, $F_{100}(\phi)$ was always computed on a lattice with $N = 149$, and only $I = 100$ ICs were chosen rather than 10^4; (2) Rather than selecting the ICs from an unbiased distribution (i.e., equal probability across the entire search space as for $\mathcal{P}^N_{10^4}(\phi)$) the ICs were chosen from a special distribution over ρ, where ρ denotes the fraction of 1s in a configuration. In this biased distribution, the probability of choosing an IC with density ρ was equal for all $\rho \in [0.0, 1.0]$. This uniform distribution over ρ is highly skewed with respect to the unbiased distribution over the search space, which is binomially distributed over $\rho \in [0.0, 1.0]$ and thus very strongly peaked at $\rho = 1/2$. Preliminary experiments indicated that the GA was able to quickly find CAs with $\mathcal{P}^N_{10^4}(\phi) > 0.5$ only when the ICs were chosen over from a uniform distribution over ρ.

In each generation the GA goes through the following steps. (i) A new set of 100 ICs is generated from the uniform distribution over ρ. (ii) $F_{100}(\phi)$ is calculated for each ϕ in the population. (iii) The E highest fitness ("elite") CAs are copied without modification to the next generation. (iv) The remaining $(P - E)$ CAs for the next generation are formed by single-point crossovers between pairs of elite CAs chosen randomly with replacement. The offspring from each crossover are each mutated m times, where a mutation consists of flipping a randomly chosen bit in a chromosome. This defines one generation of the GA; it is repeated G times for one GA run.

$F_{100}(\phi)$ is a random variable since its value depends on the particular set of 100 ICs selected to evaluate ϕ. Thus, a CA's fitness varies stochastically from generation to generation. For this reason, we choose a new set of 100 ICs at each generation. The entire population, including the set of elite CAs, is re-evaluated on a new IC set.

Also, because $F_{100}(\phi)$ is only a rough estimate of performance, we more stringently measured the quality of the GA's solutions by calculating $\mathcal{P}^N_{10^4}(\phi)$ with $N \in \{149, 599, 999\}$ for the best CAs in the final generation of each run.

For our initial experiment, we chose a population size $P = 100$, the number of elite chromosomes in the population $E = 20$, and the number of mutations per chromosome $m = 2$. A smaller population size often leads to premature convergence and loss of genotypic diversity. On the other hand, running a GA with a much larger population is computationally expensive. M was chosen from a Poisson distribution with mean 320 (slightly greater than $2N$). Varying M prevents selecting CAs that are adapted to a particular M.

For each of the two computational tasks, we performed 50 GA runs of 100 generations each. These runs were followed by number of other experiments, each consisting of 50 GA runs, in which some parameters were modified. In the next two sections, we present the phenomenological details of the results obtained from these experiments.

5.3 Density Classification

The GA uses operators such as crossover and mutation which act on the local mappings comprising a CA look-up table (the "genotype"). Selection is performed according to the dynamical behavior of CAs over a sample of ICs (the "phenotype"). As is typical in real-world evolution, it is very difficult to understand or predict the phenotype from studying the genotype. So, once the GA has constructed successful complex systems for the task, we are faced with problems familiar to evolutionary biologists: how and why did the GA evolve such systems? What underlying mechanisms gave rise high fitness in the successful systems?

In beginning to answer these questions, we must first address the following issues:

1. How well did the evolved rules perform the density classification task $\mathbf{T}_{1/2}$? What "strategy" allowed the rules to attain superior performance?

2. How efficient was the GA in finding high-performance rules for this task? Why was the GA successful or unsuccessful in finding such rules?

To investigate these issues. we studied the behavior of the evolved CAs as well the behavior of the GAs used to discover high-performance rules from several different perspectives.

5.3.1 Performance of the Evolved CAs

Table 5.1 shows the performances $\mathcal{P}^N_{10^5}$ of the some of the rules evolved by the GA. measured over three different lattice sizes $N = 149$. $N = 599$ and $N = 999$. For comparison purposes, the performances of the majority rule CA ϕ_{maj}, and ϕ_{GKL} are also presented in this table. ϕ_{GKL} was hand-designed by Gacs, Kurdyumov and Levin to study reliable computation in one-dimensional spatially-extended systems [Gac85].

In a majority of the experiments, the GA discovered CAs with behavior and performances similar to $\phi_{exp(1)}$ or $\phi_{exp(2)}$. These rules displayed fitness $F_{100}(\phi)$ between 0.9 and 1.00. Such CAs use one of two strategies: (1) Relax to the fixed point of all 0s unless there is a sufficiently large ($\sim 2r+1$) block of adjacent (or almost adjacent) 1s in the IC. If so, expand that block till the fixed point of 1^N configuration is reached. (2) Relax to the fixed point of all 1s unless there is a sufficiently large block of adjacent (or almost adjacent) 0s in the IC. If so, expand that block till the fixed point of 0^N configuration is reached. A CA exhibiting either one of these strategies is referred to as a "block-expanding rule". The block-expanding rules evolved by the GA do not count as sophisticated examples of computation in CAs: all the computation is done locally in identifying and then expanding a "sufficiently large" block. Table 5.1 also indicates that the performance of the block-expanding rules like $\phi_{exp(1)}$

CA name	Symbol	Rule Table (Hexidecimal)	$\mathcal{P}_{10^5}^{149}$	$\mathcal{P}_{10^5}^{599}$	$\mathcal{P}_{10^5}^{999}$
Majority rule	ϕ_{maj}	000101170117177F 0117177F177F7FFF	0.000	0.000	0.000
Block-expanding rule (1)	$\phi_{exp(1)}$	0505408305C90101 200B0EFB94C7CFF7	0.652	0.515	0.503
Block-expanding rule (2)	$\phi_{exp(2)}$	10101300150E086D 79DFFFFF7FFFFBFF	0.669	0.533	0.505
Particle-based rule (1)	$\phi_{par(1)}$	0504058605000F77 037755877BFFB77F	0.776	0.740	0.722
Particle-based rule (2)	$\phi_{par(2)}$	1000022441170231 155F57DD734BFFFF	0.745	0.723	0.709
Particle-based rule (1a)	$\phi_{par(1a)}$	0500171700005567 073135777FFFF77F	0.818	0.770	0.758
GKL rule	ϕ_{GKL}	005F005F005F005F 005FFF5F005FFF5F	0.816	0.771	0.757

Table 5.1: Measured values of $\mathcal{P}_{10^5}^{N}$ for different $r = 3$ rules: the majority rule (ϕ_{maj}), five rules discovered by the GA in different runs, and the GKL rule (ϕ_{GKL}) with $N = 149$, $N = 599$, and $N = 999$. The standard deviation of $\mathcal{P}_{10^5}^{N}$, when calculated 100 times for the same rule, is approximately 0.005. Each hexadecimal digit can be expanded (left to right, top row followed by bottom row) to recover the 128-bit string giving the CA look-up table outputs bits. In the resulting binary string, the outputs bits are arranged in lexicographic order of neighborhood, with the leftmost bit of the 128-bit string giving the output bit for neighborhood 00000000, and so on. Space-time diagrams of $\phi_{exp(1)}$ and $\phi_{exp(2)}$, are illustrated in Figure 5.1. Similarly, Figure 5.4 illustrates the CAs $\phi_{par1(1)}$ and $\phi_{par1(2)}$. $\phi_{par(1a)}$ was obtained by continuing the run which discovered $\phi_{par(1)}$ for 50 more generations with $\mathcal{P}_{10^2}^{N}$ as the fitness measure.

and $\phi_{exp(2)}$ decreased dramatically for larger N since the size of block to expand was tuned by the GA for $N = 149$. Among all possible ICs in $\{0, 1\}^N$, the fraction of ICs without blocks of 0s (or 1s) of length greater than $\sim 2r + 1$ tends to 0 with increasing N. Thus, the performance of such block-expanding strategies asymptotes to 0.5 with increasing lattice size since they are able to correctly classify exactly half the ICs. The highest measured $\mathcal{P}^{149}_{10^4}(\phi)$ and $\mathcal{P}^{599}_{10^4}(\phi)$ for a block-expanding rule was 0.685 and 0.560 respectively.

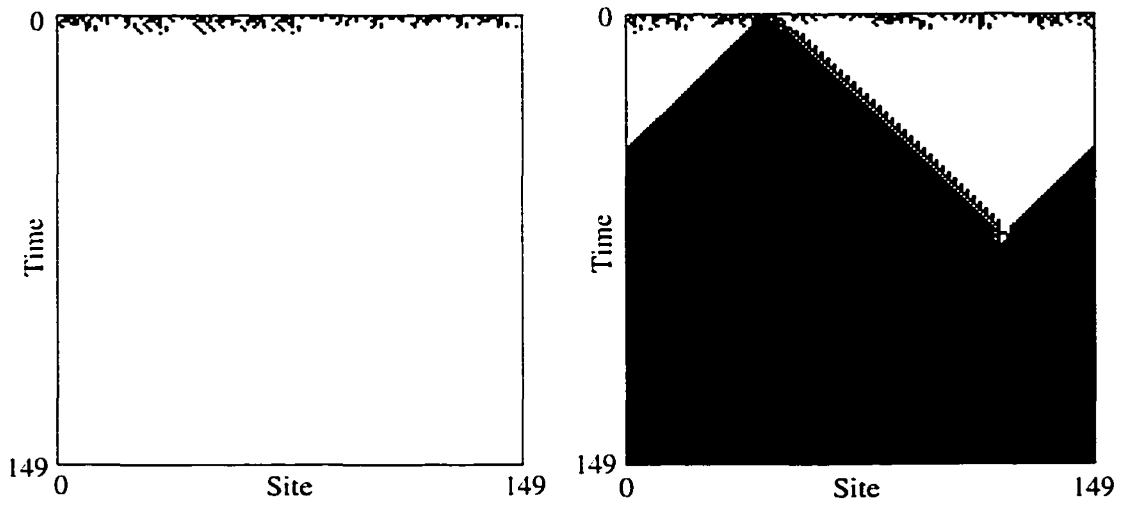
In contrast to the less sophisticated block-expanding rules evolved on most runs, in a small but significant fraction of the runs, the GA discovered rules similar in behavior and performance to CAs like $\phi_{par(1)}$ or $\phi_{par(2)}$. While the fitness F_{100} of these rules was between 0.95 and 1.0, as shown in Table 5.1, these rules displayed significantly higher performance than the block-expanding rules. More over, the performance of these rules decreased gradually as a function of the lattice size N, indicating a marked improvement in generalization. For reasons which will become clear later on, we describe such high-performance rules as "particle-based" rules.

Among all the GA discovered rules the highest measured $\mathcal{P}^N_{10^4}$ obtained from $\phi_{par(1a)}$ which exhibits performance nearly identical to that of ϕ_{GKL}. $\phi_{par(1a)}$ was obtained by continuing the run which discovered $\phi_{par(1)}$ for 50 more generations with $\mathcal{P}^N_{10^2}$ as the fitness measure.

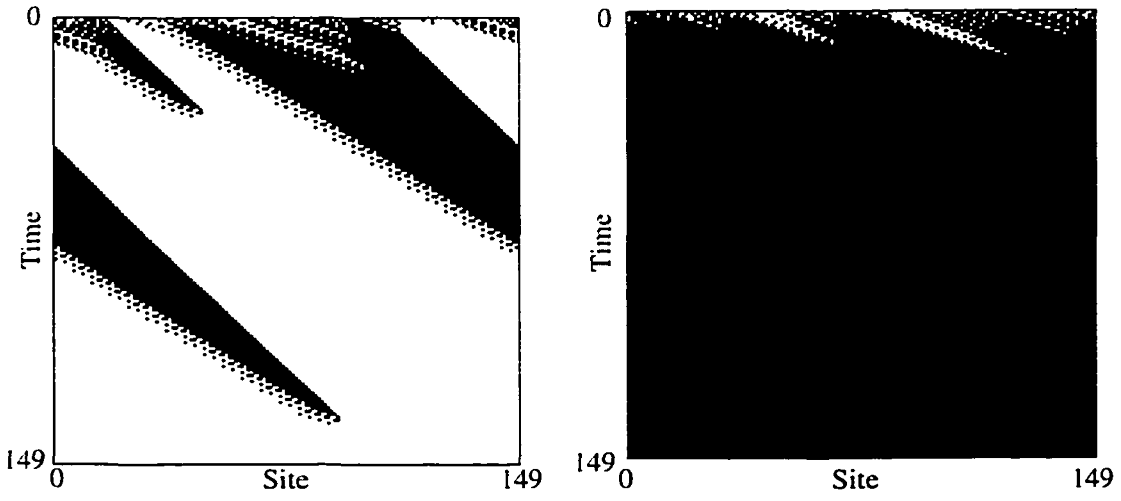
In the following two sections, we present the results of several phenomenological studies of the block-expanding rules and the particle-based rules.

Block-expanding rules

Sample space-time diagrams from CAs implementing block-expanding strategies are illustrated in Figure 5.1. In both Figures 5.1 (a) and (b), the the space-time diagrams on the left have $\rho_0 < 1/2$ while the diagrams on the right have $\rho_0 > 1/2$. In all the four instances, the correct final configuration has been reached. In Figure 5.1 (a), the rule $\phi_{exp(1)}$ quickly goes to the 0^N configuration unless there is a



(a)



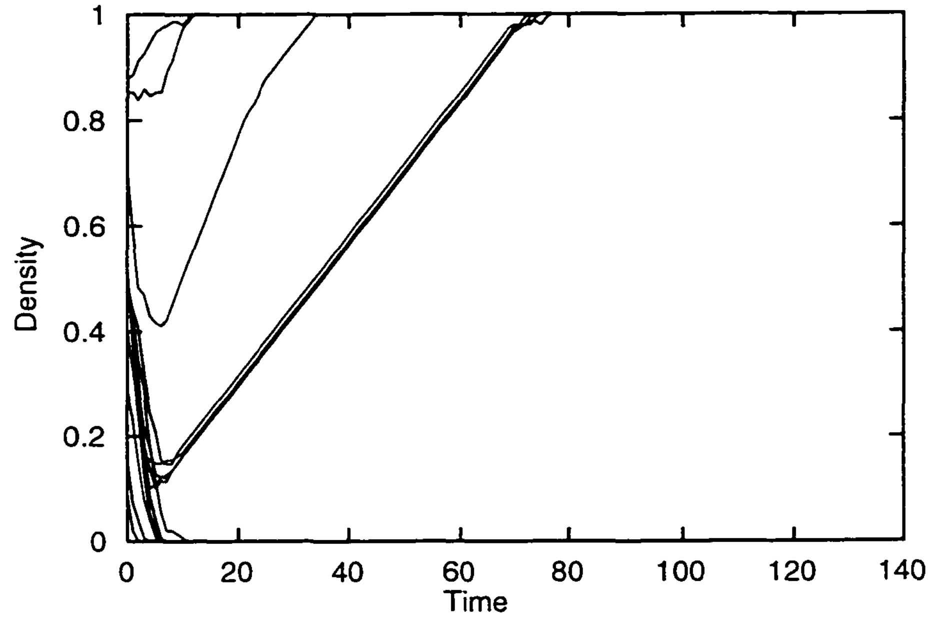
(b)

Figure 5.1: A Typical space-time behavior observed in block-expanding rules starting with random initial conditions. (a) $\phi_{exp(1)}$ following strategy 1: the CA relaxes to the 0^N configuration (diagram on the left), unless there is a large 1-block of length six in the IC. In the latter case, the CA reverts to the all 1s configuration (diagram on the right). (b) $\phi_{exp(2)}$ following strategy 2: the CA relaxes to the 1^N configuration (diagram on the right), unless there is a large 0-block of length eight in the IC. In the latter case, the CA reverts to the all 0s configuration (diagram on the left).

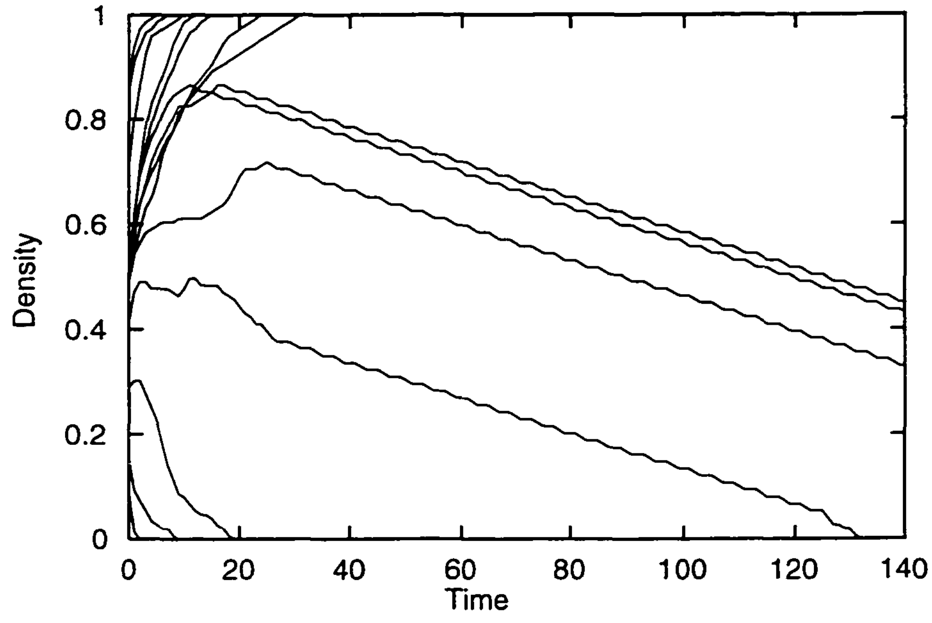
1-block of length six or more. The situation in Figure 5.1 (b) is exactly the reverse. Here the rule $\phi_{exp(2)}$ relaxes to the 1^N configuration only when there is no 0-block of length eight.

The block-expanding strategy can be analyzed in terms of how a rule reaches the fixed point configurations. Figure 5.2 (a) and (b) shows configuration density vs. time plots (i.e., $\rho(\mathbf{s}_t)$ vs. t) obtained from $\phi_{exp(1)}$ and $\phi_{exp(2)}$ respectively. In each case, ten randomly generated ICs with different ρ_0 values have been used. In Figure 5.2 (a) it can be seen that there is a fundamental asymmetry in how $\phi_{exp(1)}$ operates on the set of ICs with $\rho_0 < 1/2$ as opposed to the set of ICs with $\rho_0 > 1/2$. The figure shows that unless ρ_0 is close to 1.0, the application of such a rule immediately results in a sharp drop in $\rho(\mathbf{s})$. (This is possible because most of the look-up-table entries in such rules are fixed to 0.) However, for configurations with a sufficiently large block of 1s, the CA is able to arrest this sharp drop in density, and revert the trend to eventually reach the all-1s configuration. ICs with no such 1-blocks result in the CA quickly reaching the all-0s configuration. As shown in Figure 5.2 (b), the reverse situation is true in $\phi_{exp(2)}$. Such plots not only help in explicating the strategy of the underlying rules, but they also highlight the symmetries inherent in the $\mathbf{T}_{1/2}$ task.

Block-expanding strategies 1 and 2 rely on the appearance or absence of blocks of 1s or 0s in the IC to be good predictors of ρ_0. For example, high-ρ ICs are more likely to have blocks of adjacent 1s than low-ρ ICs. The size of blocks that are expanded was tuned by the GA to be a good predictor of high or low density for $N = 149$ given the distribution of ICs on which the rules were tested. The limitations of this approach is further detailed in Figure 5.3 which shows the performance of block-expanding rules $\phi_{exp(1)}$ and $\phi_{exp(2)}$ as a function of ρ_0 for $\mathbf{T}_{1/2}$ task. Figure 5.3 indicates that most misclassifications occur for $\rho_0 \approx \rho_c$. More importantly, the performance of a rule on ICs with $\rho_0 > 0.5$ and ICs with $\rho_0 < 0.5$ is not symmetric, and this asymmetry increases with increasing lattice size. From the three different



(a)



(b)

Figure 5.2: $\rho(s_t)$ vs. t plots from block-expanding rules: (a) $\phi_{exp(1)}$ following strategy 1, (b) $\phi_{exp(1)}$ following strategy 2. Ten randomly generated ICs with different ρ_0 values have been used generate the graphs in each figure.

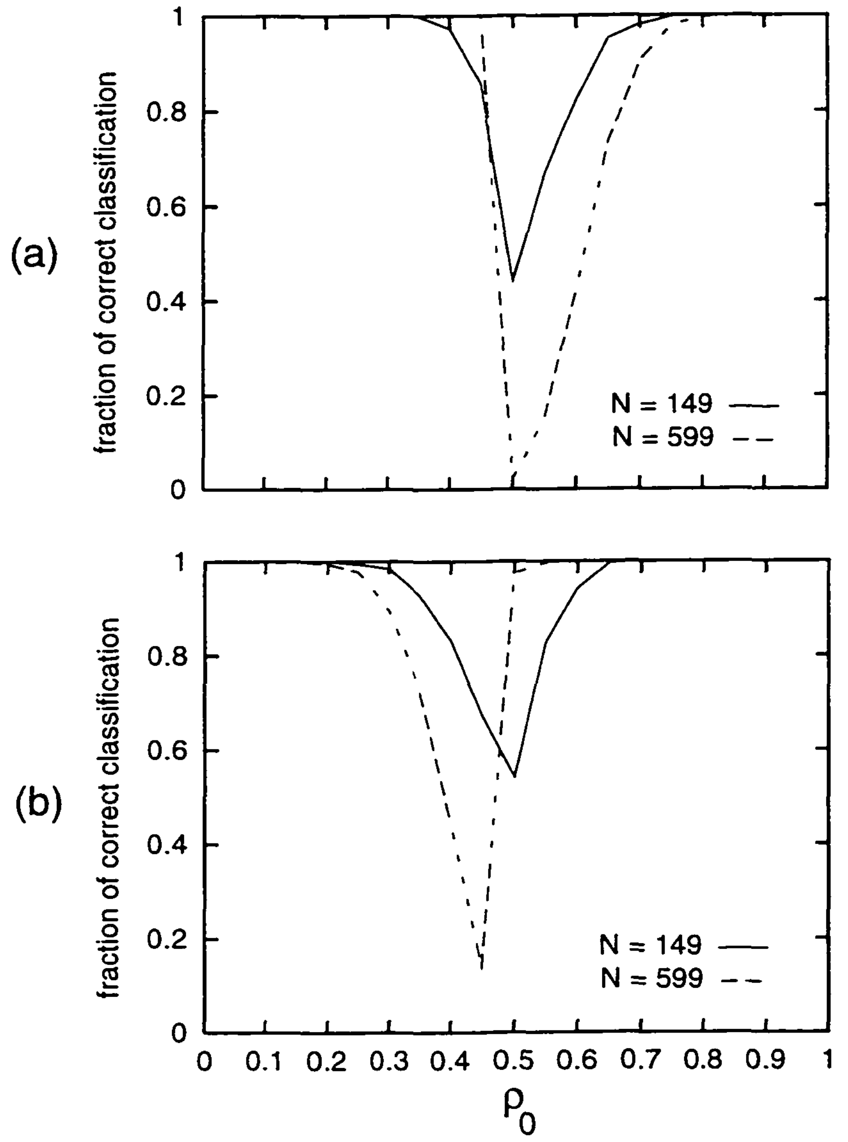


Figure 5.3: Performance of a typical block-expanding rule using (a) $\phi_{exp(1)}$ and (b) $\phi_{exp(2)}$, as a function of ρ_0 for $T_{1/2}$ task. Performance plots are presented for two different lattice sizes: $N = 149$, and 599. 20 equally spaced bins of ρ_0 values, with 10^3 ICs per bin, were used to generate the plots.

phenomenological perspectives presented here, it is clear that the failure of the block-expanding strategy can be attributed to its approach which fails to respect the underlying symmetries of the $\mathbf{T}_{1/2}$ task.

Particle-based rules

As indicated in Table 5.1, the particle-based rules like $\phi_{par(1)}$ and $\phi_{par(2)}$ exhibits superior performance compared to the block-expanding rules. Typical space-time diagrams from four such rules (each from a different GA run) are illustrated in Figure 5.4 (a-b) and in Figure 5.5 (a-b). Even a casual comparison of behavior of $\phi_{par(1)}$ and $\phi_{par(2)}$ with those of the block-expanding rules in Figure 5.1 suggests a clear difference. In all high-performance CAs for the density classification task it can be seen that although the configurations eventually converge to fixed points, there is a transient phase during which local regions are formed that display coherent and periodic spatio-temporal dynamics. The regions exhibiting these "patterns" may interact with each other. But how are we to understand the "strategy" employed by such rules in reaching the correct final configuration? Also, what is the role of the "patterns" in helping the CAs to achieve high performance? We attempt to answer these questions in our analysis in the subsequent chapters.

The $\rho(\mathbf{s}_t)$ vs. t plots for the particle-based CA $\phi_{par(1)}$, illustrated in Figure 5.6, highlights the fact that the CA's behavior is very different from typical block-expanding rules (for comparison, see Figure 5.2). In general, if the density of a configuration is less (more) than 1/2, then the density of subsequent configurations either remains the same or decreases towards 0.0 (increases towards 1.0).

Figure 5.7, which presents the performance of $\phi_{par(1)}$ (and for comparison, ϕ_{GKL}) as a function of ρ_0, further highlights the difference between a high-performance particle-based rules and the block-expanding rules (for comparison, see Figure 5.3). Although most misclassifications in this case also occur for ICs with $\rho_0 \approx \rho_c$, $\phi_{par(1)}$ appears to respect the underlying symmetries of this task since

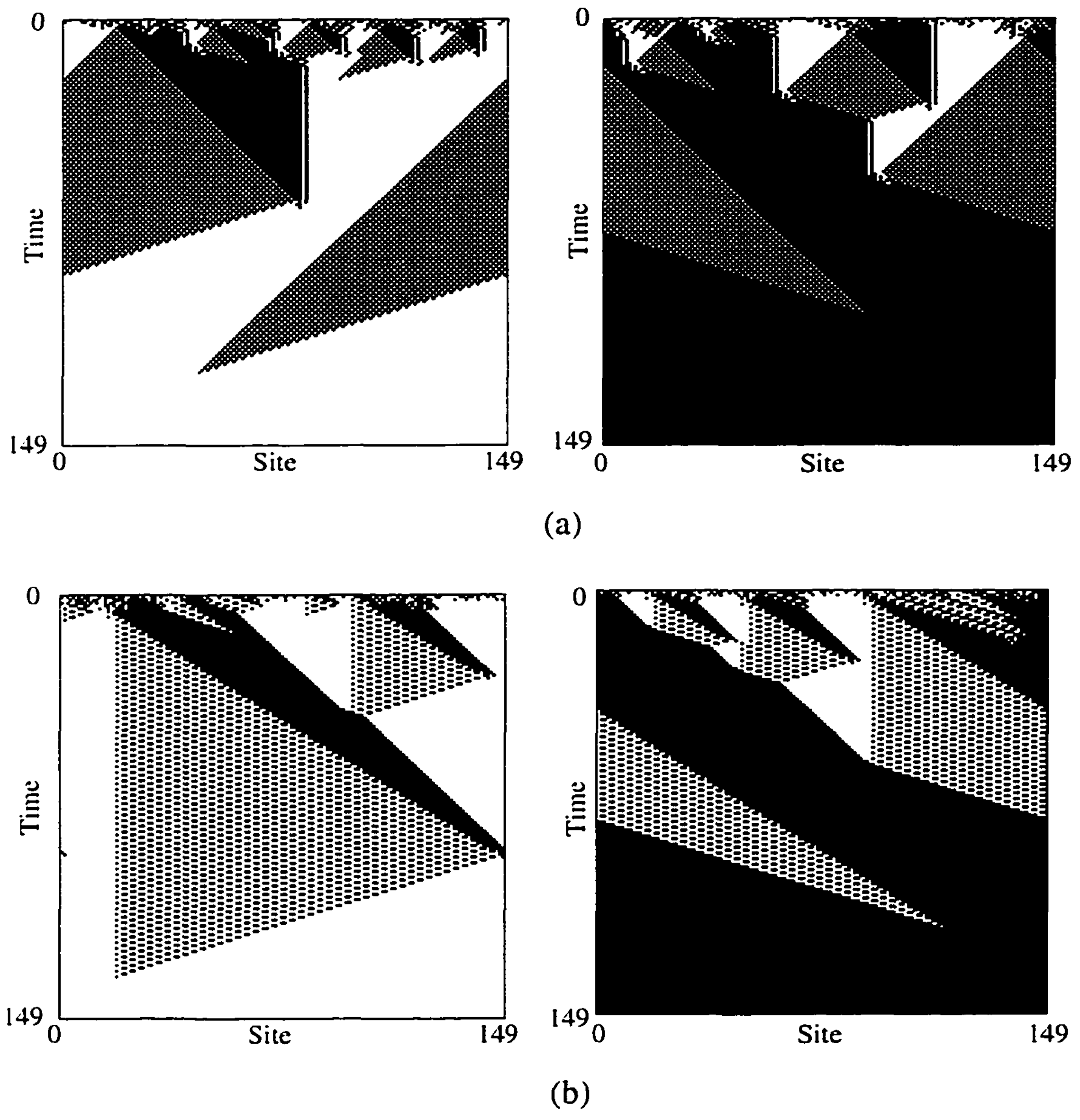
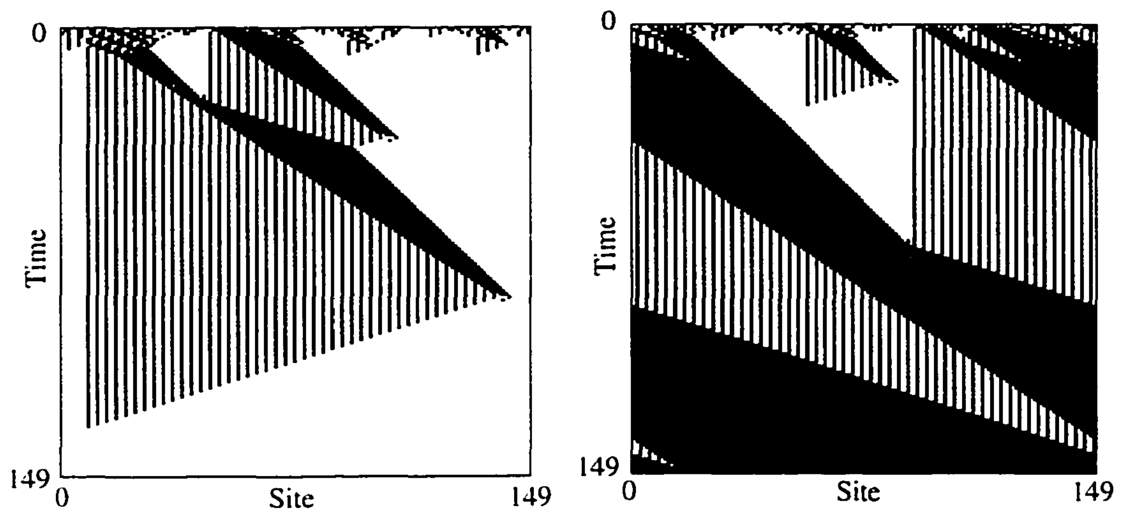
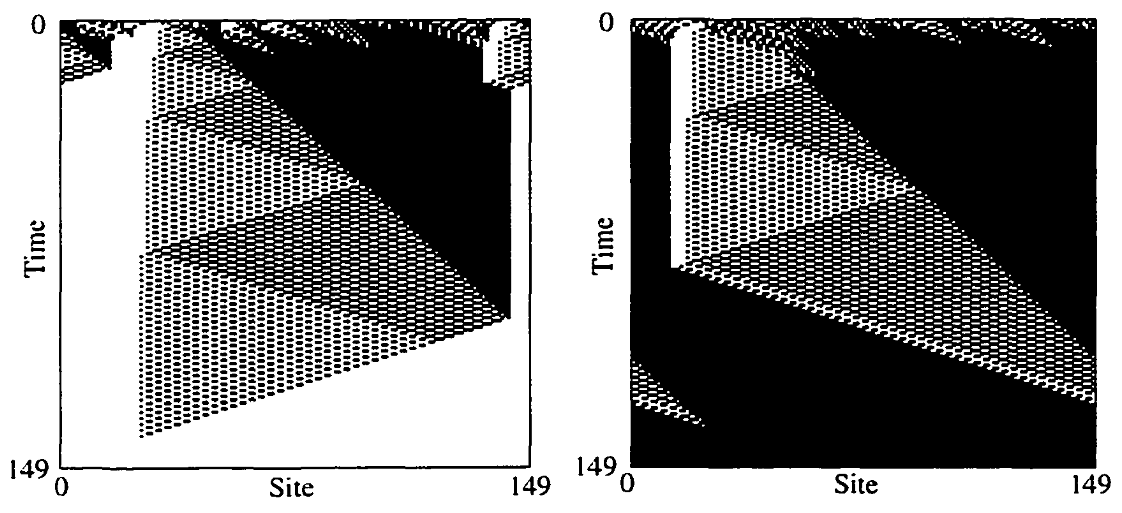


Figure 5.4: (a-b) Space-time behavior observed in two high-performance CAs, $phi_{par(1)}$ and $phi_{par(2)}$ discovered for the $T_{1/2}$ task. Randomly generated ICs have been used. The space-time diagrams on the left have $\rho_0 < 1/2$ while the space-time diagrams on the right have $\rho_0 > 1/2$. It should be noted that the correct final configuration has been reached in all cases.



(a)



(b)

Figure 5.5: (a-b) Space-time behavior observed in two high-performance CAs discovered for the $T_{1/2}$ task. Randomly generated ICs have been used. The space-time diagrams on the left have $\rho_0 < 1/2$ while the space-time diagrams on the right have $\rho_0 > 1/2$. It should be noted that the correct final configuration has been reached in all cases.

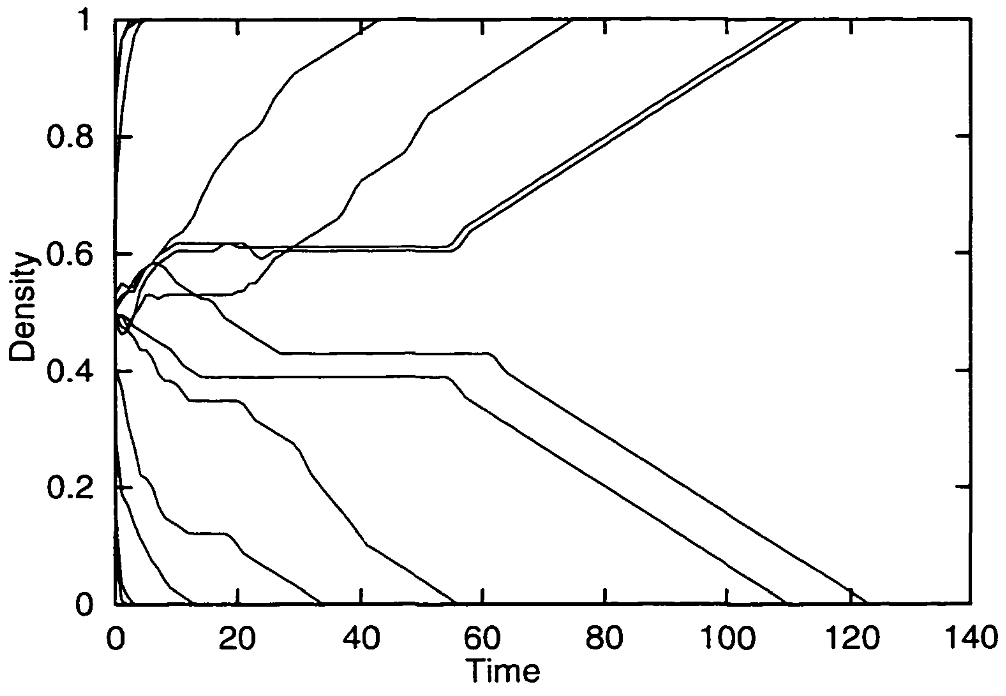


Figure 5.6: $\rho(s_t)$ vs. t plots for the high-performance rule $\phi_{par(1)}$. Ten randomly generated ICs with different ρ_0 values have been used to generate the graphs.

the performance of the rule for ICs with $\rho_0 > 0.5$ and ICs with $\rho_0 > 0.5$ is symmetric. More over, this symmetry is maintained as the lattice size is increased. Interestingly, the performance of $\phi_{par(1)}$ as function of ρ_0 mirrors that of the hand-designed rule ϕ_{GKL}. Although the rules tables of $\phi_{par(1)}$ and ϕ_{GKL} are different (see Table 5.1), we would like to understand if these rules are nevertheless applying the same strategy to perform the $T_{1/2}$ task.

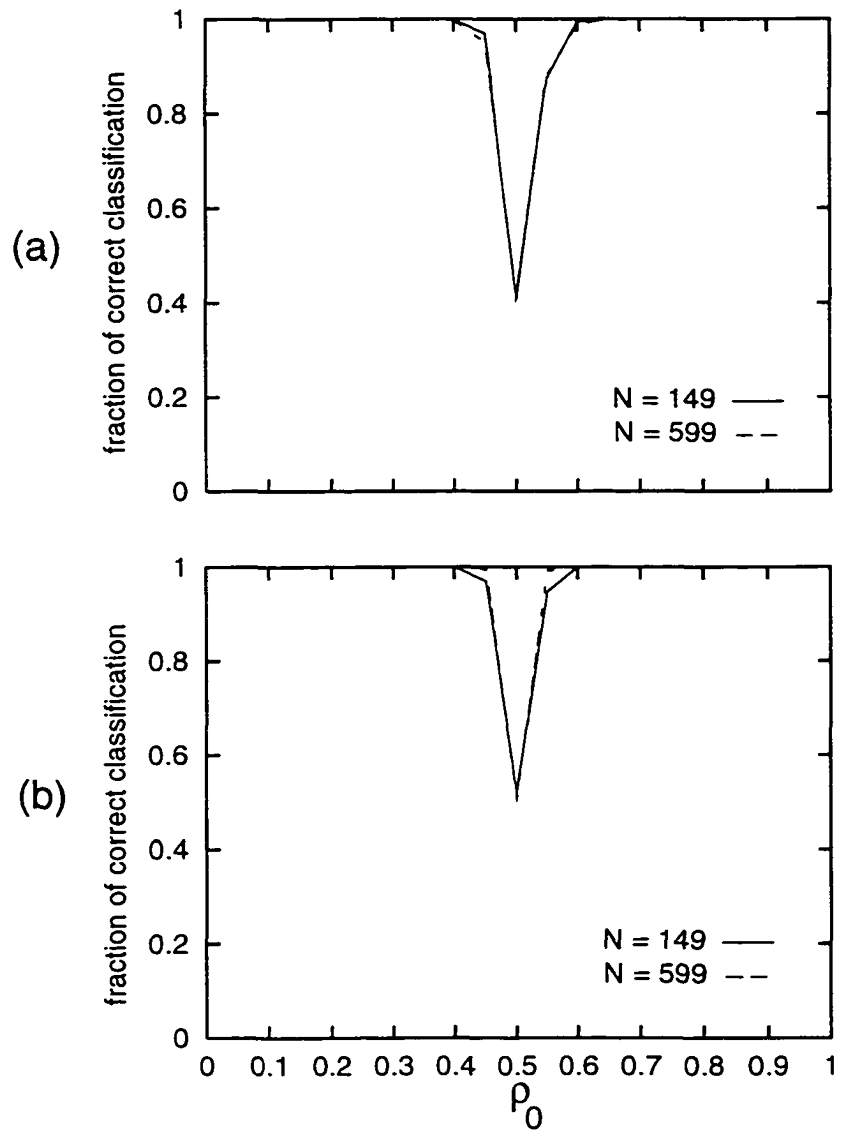


Figure 5.7: Performance of (a) $\phi_{par(1)}$, and (b) the GKL rule ϕ_{GKL} as a function of ρ_0 for $\mathbf{T}_{1/2}$ task. Performance plots are presented for two different lattice sizes: N = 149, and 599. 20 equally spaced bins of ρ_0 values, with 10^3 ICs per bin, were used to generate the plots.

5.3.2 Efficiency of the GA in Finding High-performance CAs

Case I: With Unbiased Initial Population

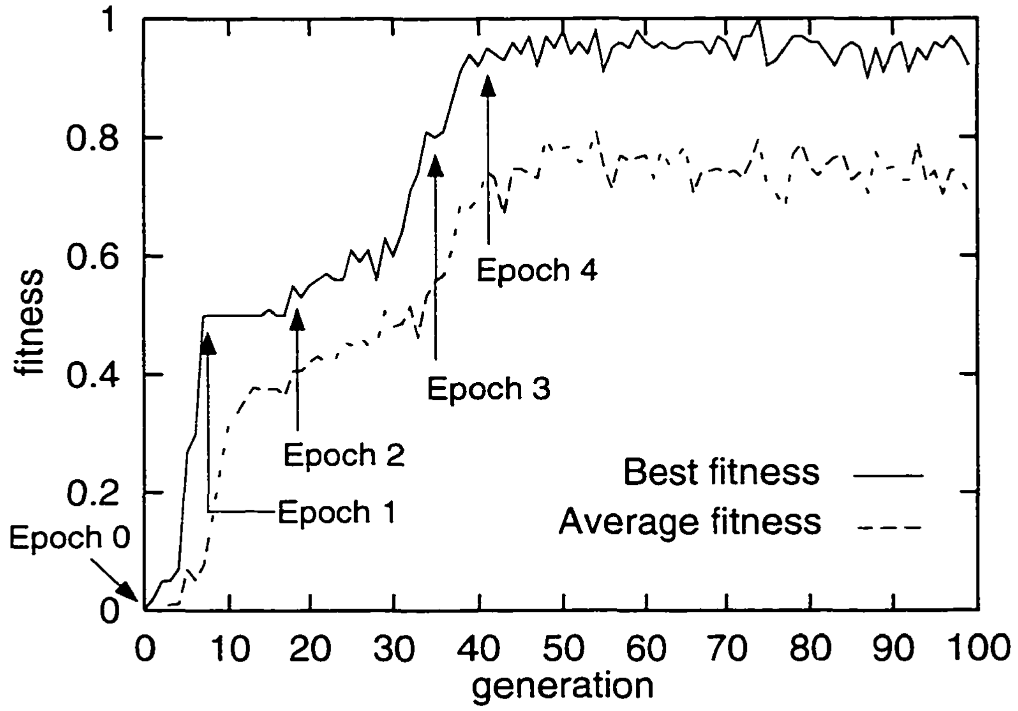


Figure 5.8: Best and average fitness versus generations for a typical run starting with a randomly generated initial population. The onset of the five epochs of innovation are depicted.

In our preliminary experiments, the initial population was produced randomly where each bit in the chromosome was set to 1 with a probability of 1/2. Figure 5.8 shows the best and average fitness of the population in each generation for a typical run starting with such an initial population. In most runs, one immediately notices that the best fitness (and the average fitness) of the population increases in sharp jumps between periods of relative stasis. The rise in fitness can be divided into "epochs" where a significantly better rule for the $T_{1/2}$ task is discovered by the GA at the start of a new epoch. In Figure 5.8, the initial generations of these epochs are labeled. At the start of each epoch, the average fitness $F_{100}(\phi)$ of a typical rule

	GA (unbiased init. pop.)	GA (no crossover, unbiased init. pop.)
Runs reaching Epoch 3	38/50	41/50
Runs used in average	38/50	41/50
$T1$	4.7 (3.7)	4.3 (4.8)
$T2$	21.7 (24.5)	34.7 (21.4)
$T3-T2$	9.6 (5.0)	10.4 (4.9)
Runs finding PB rules	0/50	0/50

Table 5.2: Results of evolving CA rules for $\mathbf{T}_{1/2}$ task under two different experimental setups: (i) GA with randomly generated initial population: (ii) GA randomly generated initial population but without crossover. and The table shows the fraction of runs reaching Epoch 3, fraction of runs used to compute averages, mean generations to onset of Epoch 1 ($T1$), mean generations to onset of Epoch 2 ($T2$), and mean length of Epoch 2 in generations ($T3-T2$) for those runs reaching Epoch 3 by generation 99. Standard deviations are shown in parentheses. The last row shows the none of the runs found particle-based rules.

was measured to be 0.00 for Epoch 0, 0.50 for Epoch 1, 0.54 for Epoch 2, and 0.85 for Epoch 3, and 0.92 for Epoch 4. The beginning of the last epoch, i.e., Epoch 4, was defined to be the the generation beyond which no substantial and sustained rise in the best or average fitness of the population was witnessed.

It is easy to explain why rules at the start of Epoch 0 has zero fitness. Since the initial population is randomly generated, it is likely that none of the CAs in the initial population is able to correctly classify even a single IC, and thus the fitness of a rule is zero. As shown in Figure 5.8, the fitness quickly rises to 0.50 in a few generations to start Epoch 1. Why is the fitness exactly 0.50 in Epoch 1? The rules in Epoch 1 always arrives at the fixed point configuration of 1^N (or 0^N) irrespective of the ρ_0 value of ICs: thus, they are able to correctly classify half the ICs. In this particular experimental setup, all the rules discovered in the subsequent Epochs exhibited block-expanding strategy.

The second column in Table 5.2 summarizes the results of evolving CAs from 50 different runs under this experimental setup. The table shows the fraction of runs reaching Epoch 3, fraction of runs used to compute averages, mean generations to onset of Epoch 1 ($T1$), mean generations to onset of Epoch 2 ($T2$), and mean length of Epoch 2 in generations ($T3 - T2$) for those runs reaching Epoch 3 by generation 99. Standard deviations are shown in parentheses.

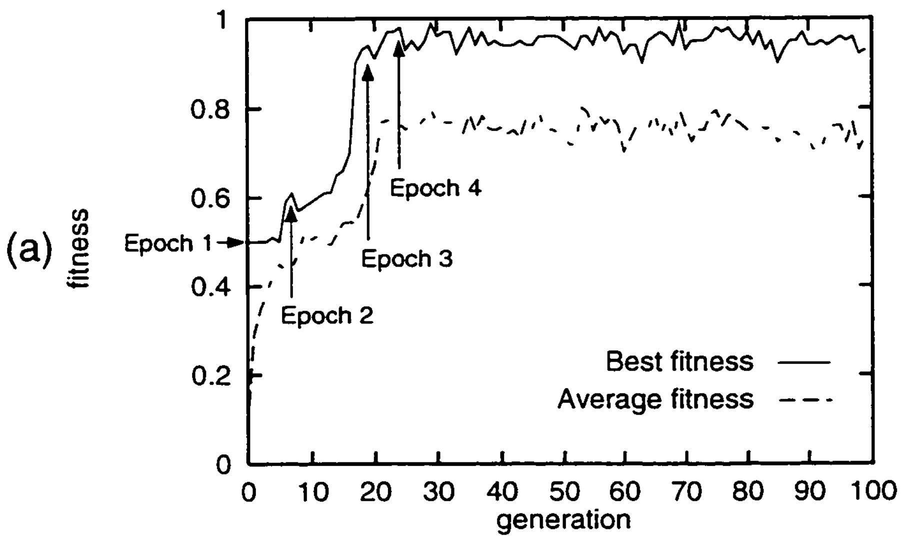
While 38 of the 50 runs under this experimental setup were able to reach Epoch 3 with rules having $F_{100}(\phi) \approx 0.85$, nevertheless, as shown in the bottom row of the Table 5.2, the GA was unable to find high-performance particle-based rules with $\mathcal{P}_{10^4}^{599}(\phi) > 0.57$ in any of the 50 runs.

The third column of Table 5.2 shows the results obtained from an identical experimental setup where all the parameters remained unchanged except for the crossover operator which was completely turned off. There exists a remarkable degree of similarity between the second and third column of Table 5.2. This suggests that in this experimental setup, the crossover operator is not playing a major role in discovering fitter rules, and therefore, it is the mutation operator which is driving the search process.

Case II: With Biased Initial Population

In an attempt to improve the efficiency of the GA, all subsequent runs started with a biased initial population where the probability of choosing a chromosome with a proportion of λ 1s, was equal for all $\lambda \in [0.0, 1.0]$. In this new experimental setup, the initial population was expected to contain rules with λ values equal (or very close) to 0.0 and 1.0. Such rules reach the fixed point configuration of 0^N (or 1^N) irrespective of the ρ_0 value of ICs, and thus, they have a fitness of 0.5. In short, this modification was aimed to allow the GA to skip over Epoch 0 and to begin searching for rules with $F_{100}(\phi) \geq 0.5$.

Figure 5.9 shows the best and average fitness in each generation for two typical runs starting with a biased initial population. As expected, the best fitness in the initial population was equal to 0.5, and so the mean generations to onset of Epoch 1 ($= T1$) was always zero. In some atypical runs, the onset of Epoch 2 was delayed as illustrated in Figure 5.10(a). However, after the onset of Epoch 2, the onset of



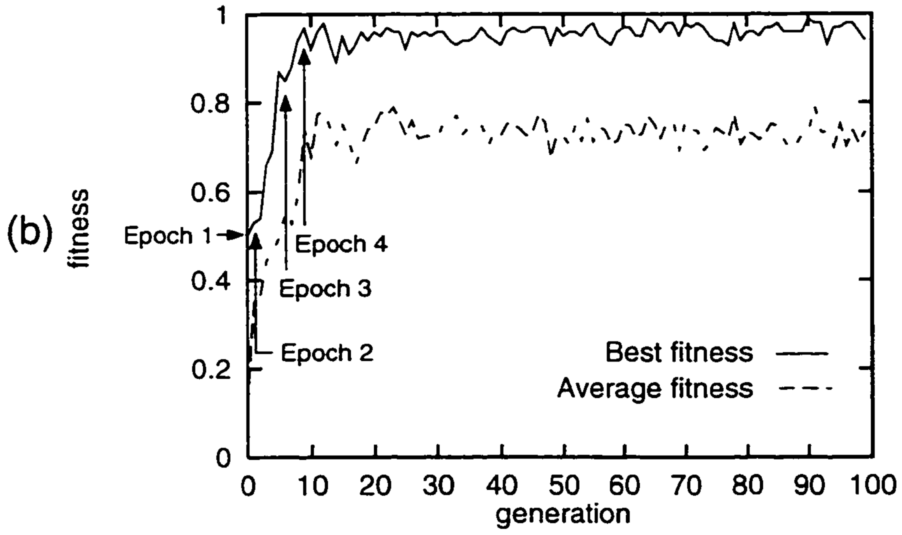
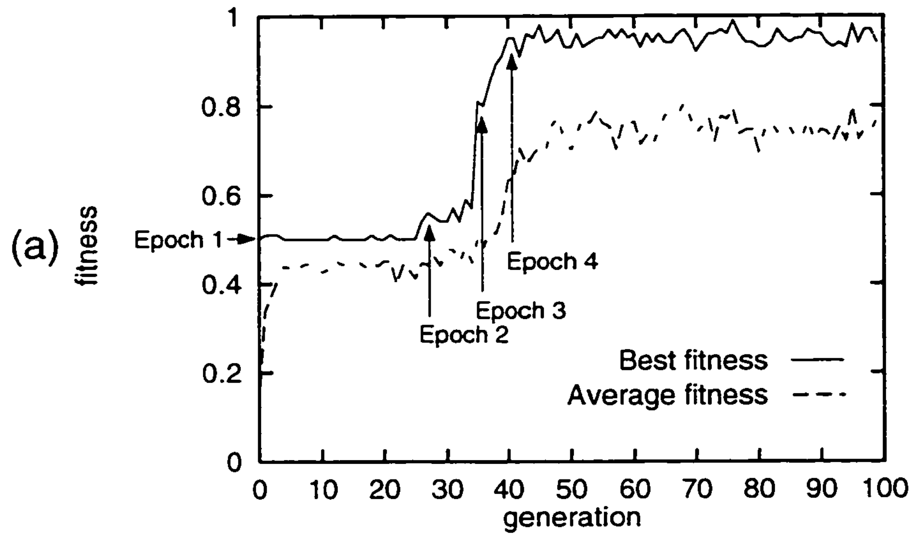


Figure 5.9: Best and average fitness versus generations for two typical runs starting with a uniform initial population. The onset of the four epochs of innovation are depicted. In figure (b) Epoch 1 lasts for zero generations.



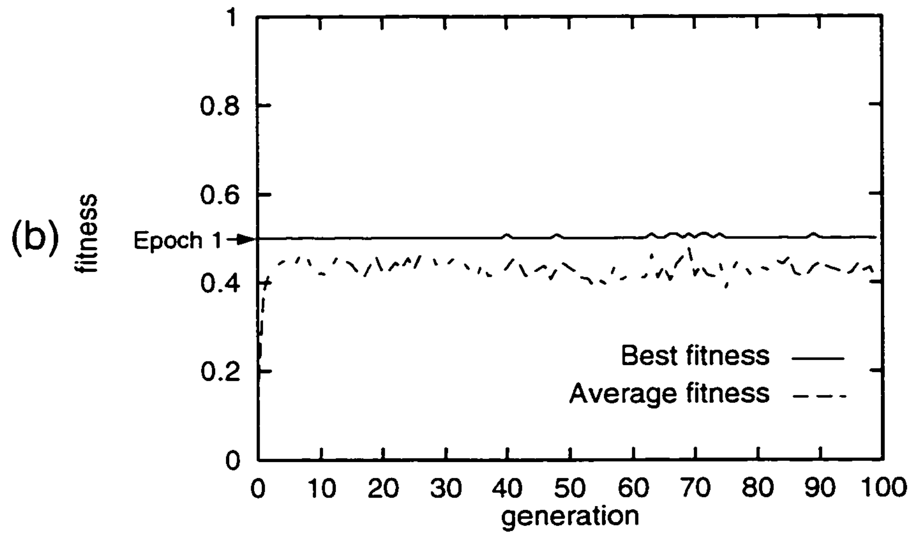


Figure 5.10: Best and average fitness versus generations for two runs starting with a uniform initial population. The onset of the four epochs of innovation are depicted. These runs are atypical in terms of the duration of Epoch 1. However, when the crossover operator is not used, then the typical run displays similar characteristics.

	GA (biased init. pop.)	GA (no mutation, biased init. pop.)	GA (no crossover, biased init. pop.)
Runs reaching Epoch 3	46/50	16/50	16/50
Runs used in average	45/50	16/50	16/50
T1	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
T2	3.8 (4.3)	2.0 (1.3)	57.4 (22.4)
T3 - T2	4.1 (2.9)	2.7 (1.5)	14.2 (9.1)
Runs finding PB rules	1/50	0/50	0/50

Table 5.3: Results of evolving CA rules for $\mathbf{T}_{1/2}$ task under three different experimental setups: (i) GA, (ii) GA without mutation, (iii) GA without crossover. In all these three cases, a biased initial population was used. The table shows the fraction of runs reaching Epoch 3, fraction of runs used to compute averages, mean generations to onset of Epoch 1 ($T1$), mean generations to onset of Epoch 2 ($T2$), and mean length of Epoch 2 in generations ($T3 - T2$) for those runs reaching Epoch 3 by generation 99. Standard deviations are shown in parentheses.

subsequent epochs proceeded normally. In an even more atypical run, no rule with $F_{100}(\phi) \geq 0.54$ was found in any generation, and thus the GA was stuck in Epoch 1 for the entire duration of the run (Figure 5.10(b)).

The second column in Table 5.3 summarizes the results of evolving CAs from 50 different runs with this experimental setup. Compared to the previous experimental setup with unbiased initial population, in this situation a higher proportion of the runs reach Epoch 3. Not only is $T1$ reduced to zero, but $T2$, the number of generations to the onset of Epoch 2, and $T3 - T2$, the duration of Epoch 2 are both significantly reduced. However, only one of the runs is able to find high-performance particle-based rules.

Interestingly, when only the mutation operator is switched off, both T2 and T3 are further reduced (third column in Table 5.3), although the proportion of runs reaching Epoch 3 drops sharply. On the other hand, when the crossover operator is switched off in place of the mutation operator, both T2 and T3 increases, while proportion of runs reaching Epoch 3 again drops sharply (fourth column in Table 5.3). This suggests that, in the initial stages of the search during the discovery of Epoch 2 and Epoch 3 rules, the crossover operator is playing a more important role than the mutation operator. How do we explain these observations?

The answer lies in the manner in which block-expanding rules can be implemented in a chromosome. Consider a rule from the initial population (Epoch 1) with a λ value equal to zero, such that it rule always reaches the fixed-point configuration of 0^N. The fitness F_{100} of such a rule increases beyond 0.5 when it is initially able to classify some high-density ICs. This occurs when the all 1s neighborhood and all the seven neighborhoods with six 1s are mapped to the output bit of 1. This ensures that the 1^N is a fixed point configuration, and isolated 0s in a region of all 1s are eliminated. It needs to be underscored here that the entire constellation of these output bits has to set concurrently to obtain an increase in fitness. The fitness of such a rule further increases as more neighborhoods consisting of a majority of 1s are mapped to the output bit of 1.

As an example, let us construct a block-expanding rule starting with a rule with λ value equal to zero. As described in the preceding paragraph, we first set the output bits of the neighborhoods { (0111111), (1011111), (1101111), (1110111), (1111011), (1111101), (1111110), (1111111) } to 1 to guarantee the stability of any all-1s region. (Under the lexicographic ordering these neighborhoods are numbered, respectively, 63, 95, 111, 119, 123, 125, 126, and 127.) To expand a block of 1s, it is necessary for the boundaries between the all-0s region and the all-1s region at each end of the block move away from each other. For example, block-expanding rule may transform a configuration in the following way:

Configuration at time t: ...0000000011111110000000....

Configuration at time $t+1$: ...00000000111111111000000.....

Here the block of 1s increases in size by one site every time step. The right moving boundary between the all-1s region and the all-0s region can be implemented by setting the output bits of the neighborhoods { (1111100), (1111000), (1110000) } (numbered 124, 120, and 122, respectively) to 1 , while the fixed boundary on the left is constructed by mapping the output bits of the neighborhoods { (0001111), (0011111) } (numbered 15 and 31, respectively) to 1. In order to guarantee that a block of 1s ultimately reaches the 1^N configuration, it is necessary to set two additional neighborhoods { (1100011), (1110001) } (numbered 99 and 113, respectively) to 1. The resulting CA implements a block-expanding strategy which relaxes to all 0s, unless the IC has a block of 1s of length equal to six or more.

The preceding discussions show that under the lexicographic ordering we have used in our experiments, most of the neighborhoods with a majority of 1s, which are necessary for implementing a block-expanding rule, are clustered around both ends of the chromosome representing the rule-table. Thus, starting from a biased initial population, which is likely to contain some chromosomes with $\lambda = 0.0$ and some chromosomes with $\lambda = 1.0$, it is possible for the crossover operator to combine such rules to create better rules implementing the block-expanding strategy. Once such block-expanding rules are found, the GA can fine-tune such block-expanding rules, mostly through hill-climbing with the help of the mutation operator, to expand blocks of length $\sim 2r + 1 = 7$. This explains why the crossover operator plays a more significant role than the mutation operator in discovering Epoch 2 and Epoch 3 rules.

This situation can be contrasted with the experimental setup where the initial population was generated randomly without any bias, and as a result there was no structure in the chromosomes in the initial population which the crossover operator could exploit. Thus, as shown in Table 5.2, the effects of crossover is negligible when the initial population is unbiased.

For comparison purposes, Table 5.4 presents the performance of Monte Carlo search technique for the $T_{1/2}$ task. The table indicates that the effectiveness of the Monte Carlo search is even more limited than the GA in terms of finding Epoch 3 or Epoch 4 block-expanding rules.

	GA (biased init. pop.)	Monte Carlo
Runs reaching Epoch 3	46/50	18/50
Runs used in average	45/50	18/50
T1	0.0 (0.0)	0.0 (0.0)
T2	3.8 (4.3)	44.1 (19.8)
T3 - T2	4.1 (2.9)	5.1 (5.0)
Runs finding PB rules	1/50	0/50

Table 5.4: Results evolving CA rules for $\mathbf{T}_{1/2}$ task under two different experimental setups: (i) GA, (ii) Monte Carlo. The table shows the fraction of runs reaching Epoch 3, fraction of runs used to compute averages, mean generations to onset of Epoch 1 ($T1$), mean generations to onset of Epoch 2 ($T2$), and mean length of Epoch 2 in generations ($T3 - T2$) for those runs reaching Epoch 3 by generation 99. Standard deviations are shown in parentheses.

Case III: With Reordered Rule-table and a New Fitness Function

The observation that most of the output bits necessary to implement the block-expanding strategy are loosely clustered near either ends of the rule-table under the lexicographic ordering, and that the crossover operator can exploit such structures in the chromosome, suggests a possible reordering of the look-up table. In this new experimental setup, the rule table was reordered so that the output bits of neighborhoods with the same number of 1s were clustered together. As shown in the third column of Table 5.5, this new ordering scheme succeeded in reducing the time the GA spent in Epoch 1, T1, to only one generation. Thus in all the runs, the GA reached found Epoch 2 rules rules with $F_{100}(\phi) \geq 0.54$ in generation 1. This can be contrasted with Table 5.2, where the GA on an average took 21.7 generations to reach Epoch 2.

Despite this success, the reordering of the rule-table did not improve the GA's chances of finding high-performance particle-based rules. The main impediment in finding such rules was that it was easy for the GA to find block-expanding rules which represent local optima in the rule space. And the primary reason why block-expanding rules have low performance is because the strategy does not respect the inherent properties of the $\mathbf{T}_{1/2}$ task which is symmetric with respect to 0s and 1s.

	GA	GA with reordered LUT	GA with reordered LUT new F
Runs reaching Epoch 3	46/50	50/50	50/50
Runs used in average	45/50	50/50	50/50
T1	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
T2	3.8 (4.3)	0.96 (0.19)	0.98 (0.14)
T3 - T2	4.1 (2.9)	3.18 (1.72)	5.70 (1.87)
Runs finding PB rules	1/50	1/50	10/50

Table 5.5: Results of evolving CA rules for $\mathbf{T}_{1/2}$ task under three different experimental setups: (i) GA (ii) GA with reordered chromosome, (iii) GA with reordered chromosome and the new fitness function (see text for details). In all these three cases, a biased initial population was used. The table shows the fraction of runs reaching Epoch 3, fraction of runs used to compute averages, mean generations to onset of Epoch 1 ($T1$), mean generations to onset of Epoch 2 ($T2$), and mean length of Epoch 2 in generations ($T3 - T2$) for those runs reaching Epoch 3 by generation 99. Standard deviations are shown in parentheses.

To encourage this symmetry, a new fitness measure F'_{100} was defined. The new fitness measure gives credit only when a CA reaches the correct final configuration with the constraint that $\rho(s_t)$ is less (more) than the critical density ρ_c if the initial density ρ_0 is less (more) than ρ_c, for $t = 1 \ldots M$. In effect, F'_{100} penalizes CAs which cross the ρ_c threshold to reach the correct final configuration. The new fitness measure "induces" the CA to avoid block-expanding strategies which fail to meet this new constraint. As indicated in Table 5.5, with the new fitness measure F'_{100} the GA is able to find high-performance particle-based rules in 20

The results presented in this section emphasizes that a detailed understanding of the underlying search space, its inherent symmetries, and the strengths and weaknesses of the genetic operators, can lead to the design of GAs which are more efficient in finding high-performance solutions.

5.4 Global Synchronization

5.4.1 Performance of the Evolved CAs

Table 5.6 presents the performance of some of the rules evolved by the GA, measured over three different lattice sizes $N = 149$, $N = 599$ and $N = 999$. For

CA name	Symbol	Rule Table (Hexidecimal)	$\mathcal{P}_{10^5}^{149}$	$\mathcal{P}_{10^5}^{599}$	$\mathcal{P}_{10^5}^{999}$
Hand-designed	ϕ_{osc}	8000000000000000 0000000000000000	0.000	0.000	0.000
Epoch 0 rule	$\phi_{E=0}$	B81D41F839EADC08 C985ADC398508D06	0.000	0.000	0.000
Epoch 1 rule	$\phi_{E=1}$	F8A19CE6B65848EA D26CB24AEB51C4A0	0.327	0.073	0.029
Epoch 2 rule	$\phi_{E=2}$	F8A1AE2FCE6BC1E2 C26CB24E3C226CA0	0.565	0.327	0.274
Particle-based rule (1)	$\phi_{par(1)}$	FEB1C6EAB8E0C4DA 6484A5AAF410C8A0	1.000	1.000	1.000
Particle-based rule (2)	$\phi_{par(2)}$	EFDFF7DFC57BEB6D D790F166C600C000	1.000	1.000	1.000

Table 5.6: Measured values of $\mathcal{P}_{10^5}^{N}$ for different $r = 3$ rules: ϕ_{osc}) and five rules discovered by the GA in different runs with $N = 149$, $N = 599$, and $N = 999$. The standard deviation of $\mathcal{P}_{10^5}^{N}$, when calculated 100 times for the same rule, is approximately 0.005. Space-time diagrams of $\phi_{E=2}$ is illustrated in Figure 5.11 (a). Similarly, Figure 5.12 (a-b) illustrates the CAs $\phi_{par1(1)}$ and $\phi_{par1(2)}$.

comparison purposes, the performances of the hand-designed rule, ϕ_{osc}, is also presented in this table.

In a majority of the runs for the **R** task, the best CAs in the final generation had performance $\mathcal{P}^N_{10^4}$ between 0.65 and 0.99. Space-time diagrams from two such rules are illustrated in Figure 5.11. Starting from the ICs shown in the figures, these CAs are able to reach global synchronization. However, the space-time diagrams contain stable phase-defects—borders separating regions that are out of phase with respect to each other. We will show later on in Chapter 8 that the presence of stable phase defects prohibit these CAs from achieving maximal performance. Moreover, when such a CA is tested on larger lattice sizes, the presence of phase defects leads to drastically inferior performance.

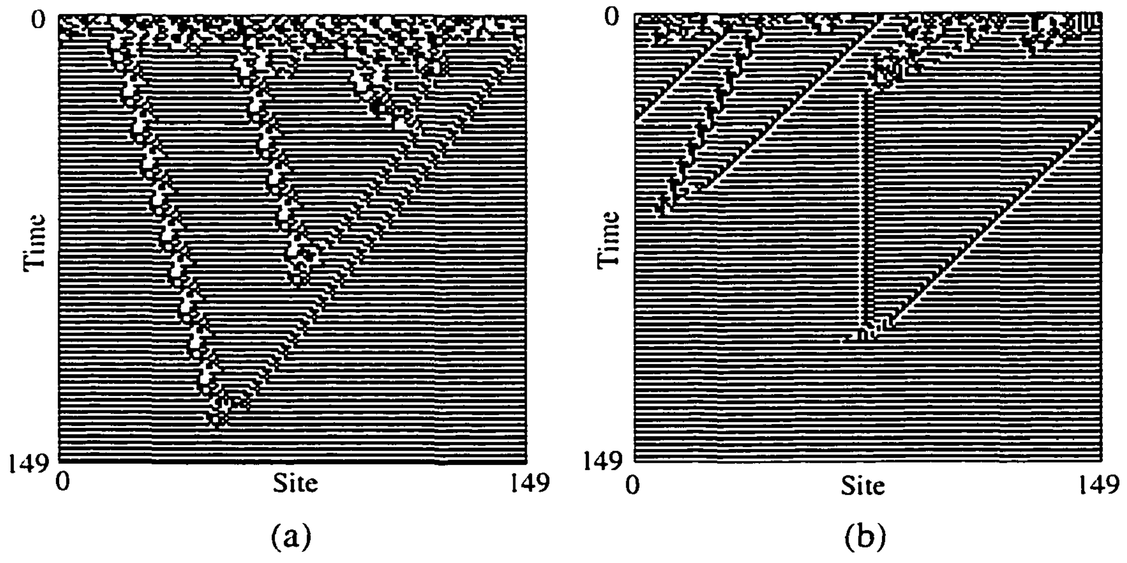


Figure 5.11: Typical space-time diagrams from two different rules with $\mathcal{P}^{149}_{10^4} \approx 0.9$. Several different types of phase defects can be observed. Randomly generated ICs have been used.

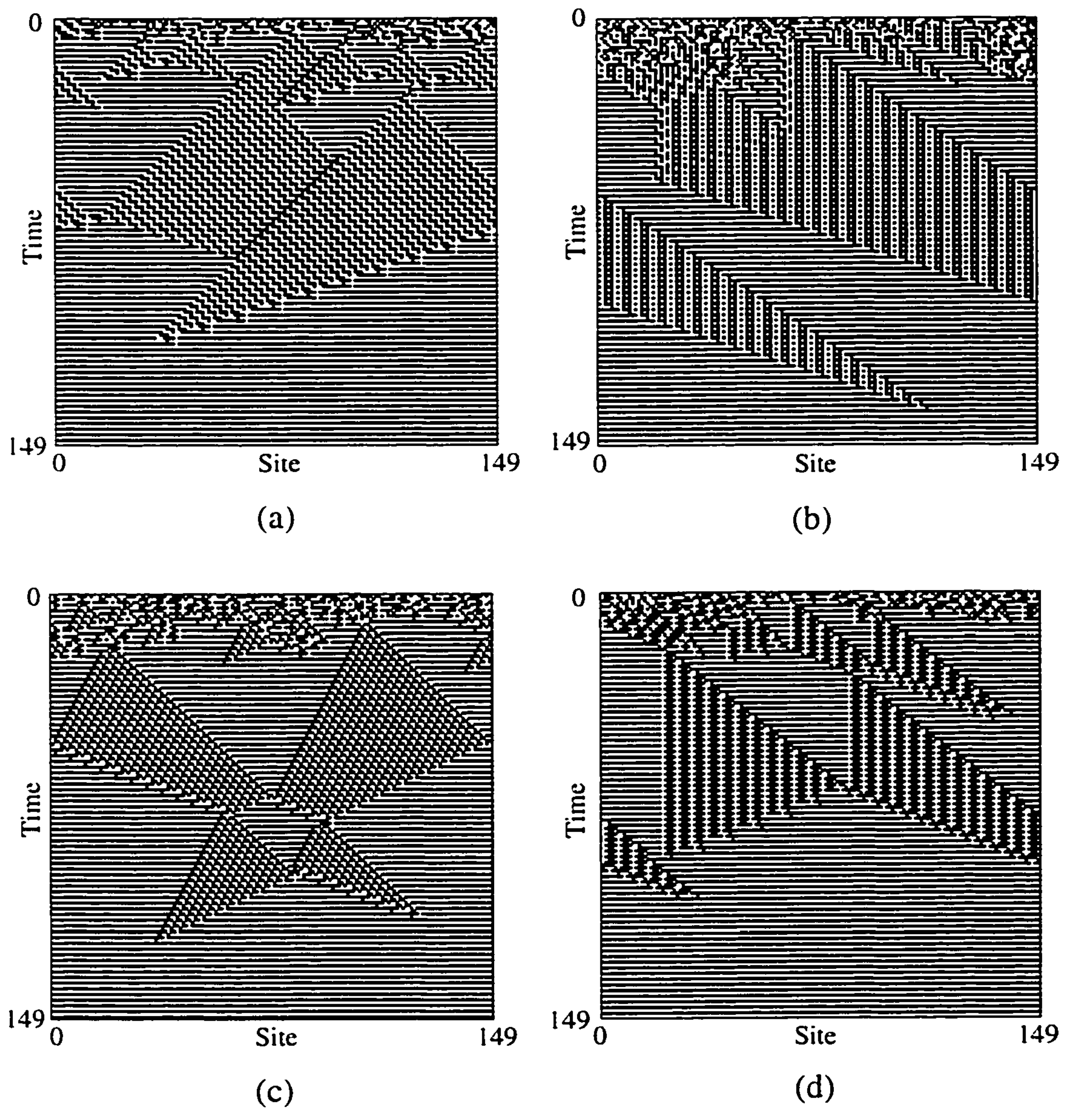


Figure 5.12: Typical space-time diagrams from four sophisticated rules (each from a different GA run) are illustrated. Randomly generated ICs have been used. All these rules demonstrated performance $\mathcal{P}^N_{10^4} = 1.0$ for lattice sizes $N = 149$, 599, and 999.

In some of the runs however, the GA discovered successful CAs ($F_{100} = 1.0$ and $\mathcal{P}^N_{10^4} = 1.0$)[1]. The performances of the more sophisticated CAs remained fixed at 1.0 even when tested on larger lattices of size 599 and 999. These results provide an interesting contrast with those found in the the density-classification task in which perfectly-performing CAs were not discovered by the GA (and are in fact impossible– see Appendix A).

Typical space-time diagrams from four sophisticated rules (each from a different run) are illustrated in Figure 5.12 (a)-(d). Unlike the space-time diagrams of the less sophisticated rules, we do not notice any stable phase defects occurring between regions that are locally synchronous. Instead, we observe an abundance of "organized patterns" in the space-time which grow and shrink over time until the CA reaches global synchronization The behavioral similarities between the high-performance rules for the $\mathbf{T}_{1/2}$ task and the $\mathbf{R}$ task are indeed striking. So in this case too, we face a similar set of questions: How do understand the emergent computation these more successful CAs are performing? How does the GA find such high-performance rules?

5.4.2 Efficiency of the GA in Finding High-performance CAs

Case I: With Unbiased Initial Population

In our preliminary experiments, the initial population was generated randomly where each bit in the chromosome was set to 1 with a probability of 1/2. The other GA parameters were set as described in Secton 5.2. In the initial population, all rules had fitness equal to 0.00. Table 5.7 shows that a large majority of the runs found

[1]Interestingly, when the GA was restricted to evolve CAs with $r = 1$ and $r = 2$, all the evolved CAs had $\mathcal{P}^N_{10^4} \approx 0.0$ for $N \in \{149, 599, 999\}$. (Better performing CAs with $r = 2$ can be designed by hand.) Thus $r = 3$ appears to be the minimal radius for which the GA can successfully perform the task.

	GA	GA, with mutation only	GA, with crossover only
Runs with $F_{100}(\phi) > 0.9$	47/50	39/50	0/50
Runs used in average	47/50	39/50	43/50
T1	4.9 (6.3)	12.5 (14.8)	3.6 (3.6)
T2 - T1	10.5 (3.7)	12.2 (9.6)	-

Table 5.7: Results of evolving CA rules for $T_{1/2}$ task under three different experimental setups: (i) GA, (ii) GA without mutation, (iii) GA without crossover. In all these three cases, a randomly generated initial population was used. The table shows the fraction of runs evolving rules with $F_{100}(\phi) > 0.9$, fraction of runs used to compute averages, mean generations to onset of Epoch 2 ($T1$), and mean length of Epoch 1 in generations ($T2 - T1$) for those runs reaching Epoch 2 by generation 99. Standard deviations are shown in parentheses.

relatively high-performance rules with fitness $F_{100}(\phi) > 0.9$. Under the actions of the evolutionary operators, the best fitness (and the average fitness) of the population rose sharply in between periods of relative stasis. As in the case of the $T_{1/2}$ task, the rise in fitness could be divided into epochs, where a rule with significantly higher fitness is found at the beginning of each epoch. The performances of typical rules in each epoch are presented in Table 5.6.

The role of the mutation operator and the crossover operator in discovering these rules was investigated by disabling each operator. The third column of Table 5.7 presents the results obtained from an identical experimental setup where all the parameters remained unchanged except for the crossover operator which was completely turned off. The results show that the mutation only GA not only takes a longer time to find Epoch 1 and Epoch 2 rules, but it also finds rules with $F_{100}(\phi) > 0.9$ less frequently. On the other hand, a GA with no mutation never finds high-fitness rules, although it is able to reach Epoch 1 rules much faster (fourth column in Table 5.7).

Case II: With Biased Initial Population

When the GA used a biased initial population, several significant differences were observed (Table 5.8). First, all the runs were able to find rules with $F_{100}(\phi) > 0.9$. Moreover, the time to reach Epoch 1 and Epoch 2 rules decreased dramatically.

	GA biased I. Pop.	GA, no mutation, biased I. Pop.	GA Random I. Pop.
Runs with $F_{100}(\phi) > 0.9$	50/50	47/50	47/50
Runs used in average	50/50	47/50	47/50
T1	0.2 (0.5)	0.4 (1.1)	4.9 (6.3)
T2 - T1	3.7 (1.8)	6.7 (2.6)	10.5 (3.7)

Table 5.8: Results of evolving CA rules for task **R** under three different experimental setups: (i) GA with uniformly distributed initial population; (ii) GA with uniformly distributed but without crossover, and (iii) GA with randomly generated initial population. The table shows the fraction of runs evolving rules with $F_{100}(\phi) > 0.9$, fraction of runs used to compute averages, mean generations to onset of Epoch 1 ($T1$), mean length of Epoch 1 in generations ($T2 - T1$) for those runs reaching Epoch 2 by generation 99. Standard deviations are shown in parentheses.

Interestingly, when the mutation operator was disabled, the GA with the crossover operator was still able to find high-fitness rules quickly and efficiently. These results suggest that the constellation of bits responsible for increased fitness in an Epoch 1 (or an Epoch 2 rule) can be easily assembled by the crossover operator.

To explain these observations, we note that the fitness of a Epoch 0 rule increases to a value more than 0.00 only when all the eight neighborhoods with six or more 0s (1s) are mapped to to 1 (0), and seven out of eight neighborhoods with six or more 1s (0s) are mapped to 0 (1). These mapping ensure that the 1^N configuration and the 0^N configuration are mapped to each other under the global map of the CA, and small perturbations from these configurations are quickly removed from the lattice. Under the lexicographic ordering, most of these bits necessary for implementing an Epoch 1 rule are clustered at the left and right ends of the chromosomes. Thus it is relatively easy for the crossover operator to bring them together from two separate CAs to create an Epoch 1 rule.

5.5 How do we Analyze the Particle-based Rules?

The preceding two sections provide some understanding of how and when the GA is able to evolve high-fitness rules efficiently. However, most of the discussion has been limited to rules implementing simple strategies like block-expanding for the $T_{1/2}$ task. As mentioned earlier, on a small number of runs, the GA found particle-based rules with even higher performance. Why do such rules have higher performance? Why is it more difficult for the GA to find such particle-based rules?

In beginning to answer these questions, we first have to identify the underlying mechanisms in the CA's dynamics which are responsible for superior performance. To understand how computation is performed by the more sophisticated CAs, we adopt the "computational mechanics" framework for CAs developed by Crutchfield and Hanson [CH93, HC92]. This framework describes the "intrinsic computation" embedded in the temporal development of the spatial configurations in a CA. Since intrinsic computation attempts to identify generic computational structures that play a governing role in the "raw" information processing occurring in the system, this framework is well suited for studying the behavior of the more successful CAs from our experiments. Before proceeding with this analysis, we first describe the computational mechanics framework in detail in Chapter 6. Once we have familiarized ourselves with the techniques and tools in this framework, we use them to explain the emergent computation in the high-performance CAs. We present these discussions in Chapter 7 and 8.

Chapter 6

COMPUTATIONAL MECHANICS FOR SPATIAL SYSTEMS

6.1 Computational Mechanics

The term computational mechanics, introduced by Crutchfield in [Cru91, Cru92], refers to the study of computational structures in dynamical systems.

The framework uses apparati developed in theoretical computer science, such as formal languages and automata, to discover, characterize, and quantify the intrinsic computation capabilities in a dynamical system. In other words, the computational mechanics approach attempts to identify regularities or structural features that are embedded in the generic behavior of a system, but can nevertheless play a key role in the information processing, storage and transmission within the system. The framework provides a new perspective for studying dynamical systems and has the potential to discover novel forms of computation which may be embedded in the behavior of natural or artificial spatially-extended systems.

When the computational mechanics apparatus is applied to spatially-extended systems such as CAs, the intrinsic parallel computational properties of the system are identified. The framework essentially describes the spatio-temporal behavior of the system as the internal dynamics of a parallel computer.

To use the computational mechanics framework, a system's spatial configuration is first converted into a sequence of discrete symbols. Since any measuring device has a finite level of resolution, in practice this conversion is inevitable in any measurement. In principle, however, techniques from symbolic dynamics can be used to make the conversion. One of the benefits of studying a CA as an idealized spatially-extended system is that in a CA the system's state variables are already discrete, and hence the discretization process is unnecessary.

Once the sequence of discrete symbols representing the spatial configurations of a system are available, tools from the computational mechanics apparatus can be applied. First, the formal languages or automata representing the pattern bases are inferred from the discrete symbols sequences. For a given pattern basis, a transducer is constructed that performs pattern recognition for that particular basis. The transducer identifies the patterns in the configurations that conform to the pattern basis. Such regions—"domains"—form the basic units of computational structures in the system's behavior. The system can then be studied from the standpoint of these domains, i.e., in terms of the interactions between the different computational structures inherent in the dynamics of the system.

The following discussion provides an informal introduction to the main tool in computational mechanics, namely *machine reconstruction* for pattern discovery. It also shows how to use the discovered pattern bases to study computation in one dimensional CAs. For a more comprehensive introduction, the reader may refer to [Cru91, CY89, CY90, HC92, CH93]

6.2 Machine Reconstruction

Machine reconstruction is an inductive technique which accepts sample data produced by a dynamical system and infers a finite state machine representing the computational structures embedded in the system's behavior. This procedure disregards the statistical properties of the data, and instead focuses on the topological properties inherent in the data. Although this section deals with the machine reconstruction from the data generated by a deterministic dynamical system, the extension of the framework to the reconstruction of stochastic systems is straightforward.

6.2.1 The Reconstruction Procedure

The machine reconstruction process is achieved in three steps: (i) gathering the data, (ii) building a tree from the data, and (iii) building a machine from the tree. Although it was mentioned earlier, it is worth re-emphasizing that the above three steps can be applied hierarchically to build increasingly sophisticated models of the system [Cru91].

Gathering the Data

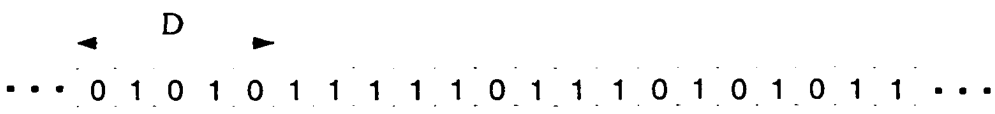


Figure 6.1: Gathering the Data: A sample data sequence is scanned by a moving window of length D.

To gather the data, a measuring device is used to map the states of a dynamical system onto symbols in some finite alphabet $\mathcal{A}$. For continuous dynamical systems, a measuring distortion is necessarily introduced since the measuring device partitions the state space into discrete cells of size ϵ according to the precision of the measurement. The machine reconstructed from such data is called an *ϵ-machine* [CY89]. However, in the research described here, discrete dynamical systems have been employed; thus there is no measurement distortion. We restrict our attention to binary alphabets (i.e., $\mathcal{A} = \prime, \infty)$ in this work.

In a spatially-extended system, the measuring device can gather data in a variety of ways. When it is fixed at a given spatial location, the device can gather temporal data by measuring the time series of a particular characteristic associated with the location. On the other hand, the device can gather spatial data, by sampling a characteristic associated with every site in the system at a given instant in time. Since all parts of the system are equally important to us, we choose the latter approach.

A given data sequence of length N is scanned with a moving window of size D—with $D \ll N$—to create a series of subwords of length D. In each iteration, the window is moved one symbol to the right to produce a new subword. For example, scanning the sample data stream in Figure 6.1 with a moving window of size $D = 5$ results in subwords 01010, 10101, 01011, and so on.

D is an upper bound on the size of the structural features that can be detected in the data stream by the machine reconstruction technique. Usually, D is set to a large value to extract the maximum amount of structure in the data. On the other hand, reconstruction with a large D is susceptible to statistical fluctuations in the data due to the limited sample size.

Constructing the Tree

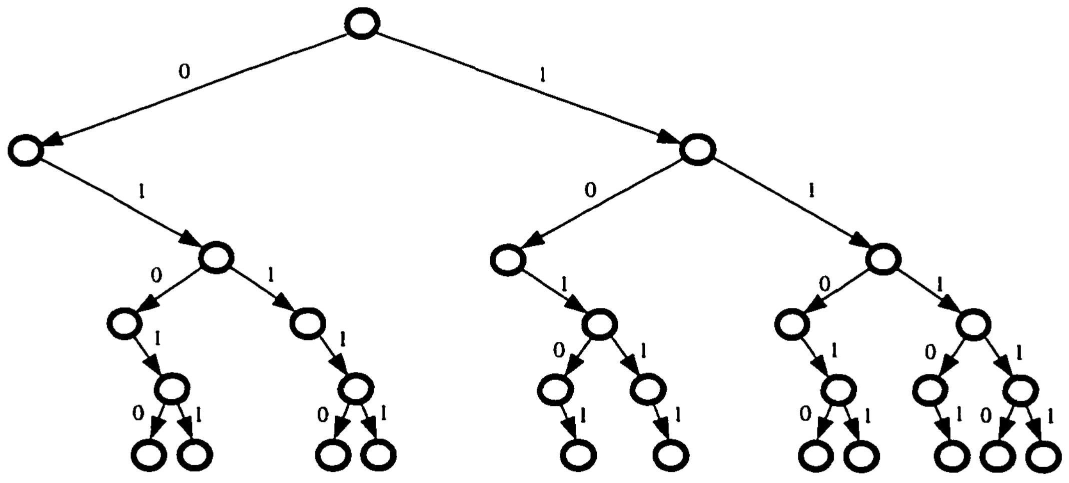


Figure 6.2: Constructing the tree: The tree with depth $D = 5$ constructed from the data sequence in Figure 6.1.

In this step, the set of subwords produced from the data sequence is used to construct a tree of depth D. First, the subwords are aligned such that the first symbol in the subword corresponds to the top level of the tree, the second symbol corresponds to the second level and so on. At any given level, if a symbol in the subword is a 0, then the left branch is followed, else if the symbol is a 1, then the right branch is followed. This process is repeated for each subword in the data sequence (Figure 6.2). For machine reconstruction, the number of times a node is visited is disregarded, although such a measurement is of fundamental importance in the reconstruction of stochastic systems.

Building the Machine

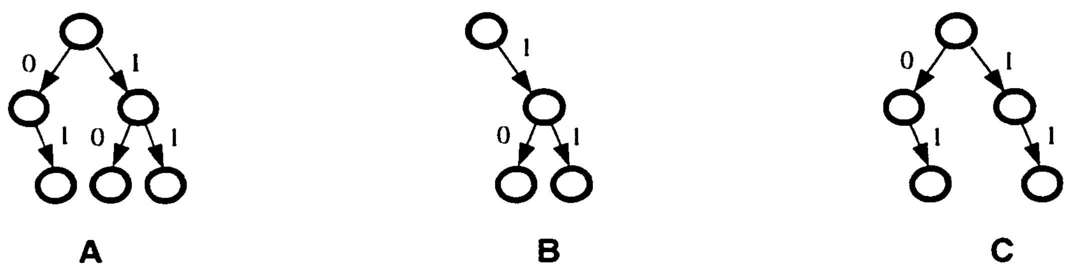


Figure 6.3: The three distinct morphs of depth $L = 2$ identified from the tree in Figure 6.2.

Given the tree constructed from a data sequence, building the machine involves identifying the nodes in the tree which are *future equivalent*. Two nodes are said to be future equivalent if the branching structure of the subtree rooted at each of the nodes is identical. In order to characterize subtrees, the idea of a *morph* is used. A morph is a tree of depth L, where $L < D$. A morph defines an equivalence class of subtrees which have the same branching structure expressed in the morph. The tradeoff in choosing the morph depth L is that a small value of L increases the occurrences of each morph in a tree whereas a large value of L increases the number of possible morphs. A reasonable compromise is to choose a morph depth

$$L = \lfloor \frac{D-1}{2} \rfloor \tag{6.1}$$

where $\lfloor x \rfloor$ is the largest integer smaller than or equal to x [Cru91]. In Figure 6.2 the tree depth D is 5, and hence the morph depth L is set to 2.

Once L is chosen, the tree is analyzed to determine the distinct morphs of depth L, and each morph is assigned a unique index. For example, the three distinct morphs with $L = 2$ identified in Figure 6.2 are shown in Figure 6.3; they are labeled A, B, and C.

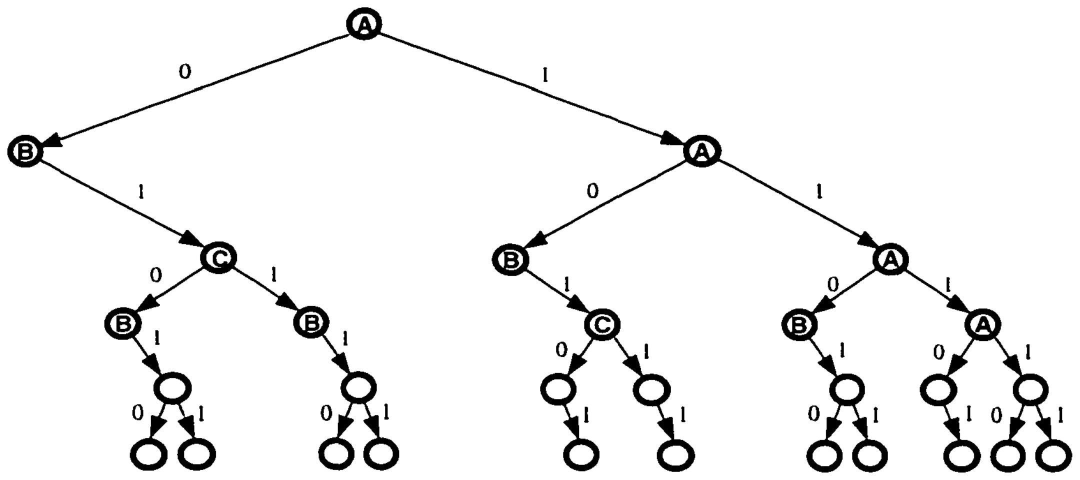


Figure 6.4: The tree in Figure 6.2, with nodes labeled by matching morph indices from Figure 6.3.

Once the morphs have been identified and indexed, the nodes in the tree are labeled by the index of the corresponding morph. However, the nodes in the bottom L levels of the tree are left unlabeled since the subtree rooted at those nodes are of depth less than L. An example of this labeling scheme is shown in Figure 6.4.

Since the nodes representing the root of the subtrees characterized by a unique morph are future equivalent, the morphs themselves define the states of the reconstructed machine. The states of the reconstructed machine are assigned the same index of the matching morph, and the transitions between the states are determined from the constructed tree. In the machine, a directed edge from state u to state

v with label σ is added when there is at least one branch in the tree which leads from morph u to morph v and is labeled as σ. Following the above steps, the reconstructed machine from Figure 6.4 is shown in Figure 6.5.

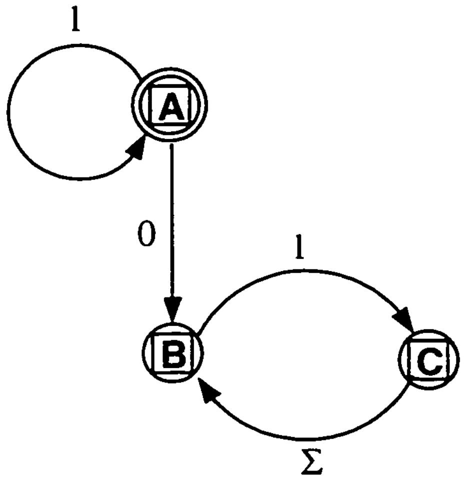


Figure 6.5: The machine: Reconstructed machine from the tree in Figure 6.4. In this figure Σ represents either symbol 0 or 1.

This concludes the informal introduction to machine reconstruction. However, there are a variety of theoretical and practical issues related to machine reconstruction, which have not been broached in this section, but are discussed in detail in [Cru91].

6.3 Computation in One-Dimensional CAs

Hanson and Crutchfield have made detailed investigations of several one-dimensional CAs using the computational mechanics framework [HC92, Han93, CH93, HC95]. The following discussion gives an informal introduction to their approach.

Hanson and Crutchfield's approach is based on the idea that a CA can be viewed as a regular-language processor. The key notion here is that any configurations (or subconfigurations) of local states in a one-dimensional CA lattice is viewed as words in some regular language [Hur87, Wol84b]. Since a given word can belong to an infinite number of regular languages, the aim is to determine the finite state machine that best describes the properties of the configurations produced by a CA.

However, a given finite state machine is not only able to characterize a single configuration (or subconfiguration): it is also a functional description of the structural properties shared by a set of configurations. Thus, a finite state machine is actually a specification of a particular *pattern*, and all symbol sequences generated by the machine are examples of that pattern. It is this viewpoint that sets the stage for understanding the pattern dynamics in a CA. Instead of simply asking how a CA maps one configuration into another (i.e., $\Phi(\mathbf{s}_t) = \mathbf{s}_{t+1}$), it is now possible to investigate how a CA maps an ensemble of configurations: $\mathbf{\Phi}\mathcal{L}_t = \mathcal{L}_{t+1}$, where $\mathcal{L}_i$ is a regular language describing the CA configurations at time i. It is important to keep in mind that $\mathcal{L}$ is independent of the CA lattice size, which considerably facilitates the analysis of the behavior of a CA.

Using the above framework it is possible to identify different types of mappings on $\mathcal{L}$. When $\mathcal{L}$ is mapped onto itself—$\mathbf{\Phi}\mathcal{L} = \mathcal{L}$—then it is defined to be an *invariant language* with respect to $\mathbf{\Phi}$. On the other hand, when $\mathbf{\Phi}_p\mathcal{L} = \mathcal{L}$ for some finite integer p, then $\mathcal{L}$ is defined to be a *periodic language*. Temporal invariance or periodicity in $\mathcal{L}$ does not necessarily imply that the configurations in $\mathcal{L}$ are themselves temporally invariant or periodic.

6.3.1 Regular Domains

A central feature in the analysis of computation in CAs is the notion of the *regular domain* [HC92]. A regular domain is a region in space-time which is "computationally homogeneous". Formally, a regular domain $\Lambda = \{\mathcal{L}_1, \mathcal{L}_2, \ldots\}$ is a set of regular languages representing spatial configurations (or sub-configurations) in a CA with the following two properties:

(i) Temporal invariance: $\Phi\Lambda = \Lambda$, i.e., Λ is mapped onto itself; and

(ii) Spatial homogeneity: The process graph of Λ is strongly connected.

A process graph is a directed graph representing a deterministic finite automata (DFA) with one or more start states. A process graph is said to be strongly connected if it consists of a set of recurrent states such that it is possible to reach any state from any other state in the directed graph. If a process graph is strongly connected, it ensures that spatial translations of words in the language are also words in the same language. However, spatial homogeneity does not imply that the configurations in the domain are spatially uniform or periodic.

The computational homogeneity in a domain can be expressed in the CA configurations in several different ways. Since CAs are deterministic systems, spatial uniformity or periodicity in the configurations necessarily implies temporal uniformity or periodicity in the configurations. However, temporal uniformity or periodicity in the configurations can still result in configurations that are spatially disordered (e.g., the ECA rule 204, which implements the identity map $\Phi(\mathrm{s}) = \mathrm{s}$ for all s, has $\Lambda = \mathcal{A}^*$). With this in mind, it is possible to distinguish a number of different types of domains for which the configurations in a CA have the following properties:

(i) Spatial and temporal uniformity.

(ii) Spatial periodicity and temporal uniformity or periodicity.

(iii) Spatial disorder and temporal uniformity or periodicity.

(iv) Spatial and temporal disorder. Such domains are called "chaotic domains".

6.3.2 Domain Discovery

So far in this section, the existence of regular domains in CAs have been assumed. But how can one discover and identify the domains from the spatio-temporal behavior of a given CA? It is at this juncture where machine reconstruction is used on CA configurations to automatically discover regular domains. For the process of machine reconstruction, data is gathered at a single time-step from a subregion in a large lattice, with the CA starting from a random initial configuration. By choosing different subregions in the configuration, multiple regular domains in the space-time diagram can be identified. Once machine reconstruction has been used to discover candidate languages, the candidates can be tested to ensure that they satisfy the regular domain conditions of temporal invariance and spatial homogeneity.

6.3.3 Domain Filtering

After the regular domains have been identified, the next step in the analysis of pattern dynamics in a CA involves the design of a *domain transducer.* A domain transducer T is an automaton that serves the function of a pattern filter. Since domains are computationally homogeneous, they represent regularities in the computational behavior that are completely understood. Regions in space-time behavior that are deviations from the regularities represent features of the CA behavior whose computational roles are yet to be delineated. A domain transducer filters out all the known regularities in the space-time plot in order to highlight the embedded irregularities. The underlying goal here is to iteratively apply the computational mechanics tools on the deviations from the regularities to discover more sophisticated forms of regularities in the data.

A domain transducer is a finite state machine which accepts words in the domain languages $\{\Lambda^0, \Lambda^1, \ldots\}$ that occur in the space-time plots of a given CA. For each spatial configuration in a given space-time plot, the transducer scans the configuration in a particular direction and assigns the same symbol i to any site that belongs to words in domain Λ_i. The resulting filtered space-time diagram depicts behavior of the domain boundaries or *walls.* A wall which separates two domains of the same language is called a *defect.* Defects with zero thickness are called *dislocations.*

Although a transducer can scan a configuration in either direction, in order to simplify the discussion, in all subsequent sections it is assumed that the transducer under consideration is moving from left to right.

Given the process graphs for the domains $\{\Lambda^0, \Lambda^1, \ldots\}$, the design process of the transducer can be fully automated. The construction of a transducer involves three steps which deal with three different aspects of the filtering operation: (i) synchronization, (ii) arriving at a wall, and (iii) re-synchronization.

Initially, before reading the first cell in a configuration, a transducer is in the start state which represents its total ignorance of the current context. In order to determine if a cell is a wall or a domain (and also to specify the domain), the transducer has to read a finite number of cells to synchronize itself. During this synchronization process, the transducer reads in new symbols and emits the null symbol λ. The set of synchronizing states in the transducer can be generated by converting a nondeterministic finite automata (NFA) which consists of all the process graphs $\{\Lambda^0, \Lambda^1, \ldots\}$, to a deterministic finite automata (DFA). The resulting DFA allows the transducer to begin parsing a configuration, and to arrive at one of the states in the original process graphs within a finite number of steps.

As long as a region in a configuration is in a domain Λ^i, a transducer is in one of the recurrent states in the process graph of Λ^i. A wall is encountered when a transducer reads a symbol for which there is no corresponding edge in the process graph. Walls are dealt with as follows. For every disallowed transition in the original process graphs, a new edge is placed between the process graphs such that the resulting machine can read in and then classify any word in $\mathcal{A}^*$. But how is the destination node for a new edge determined? A naive solution is to try to re-synchronize by forcing the new edge to return to the start state of the transducer. However, the fact that the transducer was in a particular process graph before encountering the wall does provide some contextual information and it can be used to re-synchronize in an optimal fashion. Suppose the transducer is in state V_i of the process graph of domain Λ^i when it reads a symbol a for which there is no legal transition edge. From the set of legal words allowed in domain Λ^i which end in V_i, a minimal length suffix w is found such that the word wa is in one of the domain languages $\{\Lambda^0, \Lambda^1, \ldots\}$. The destination of the new edge from V_i is the node in the U_j where wa is a minimal length legal word allowed in Λ^j which ends in U_j. To uniquely identify the transition made on encountering a wall, the transducer emits a symbol which is a function of the origin and destination states of the new edge. For a detailed analysis of the re-synchronizing process, the reader may refer to [CH93].

In a periodic lattice, the cells which are initially assigned the null symbol λ during the synchronization process are reassigned new symbols during the second pass of the transducer. It can be shown that the transducer needs at most two complete passes over a configuration to assign new symbols to each site.

6.4 Domains in Elementary CAs

Using the computational mechanics tools discussed earlier, this section gives some examples of domain analysis in one-dimensional ECAs.

6.4.1 ECA Rule 40

Figure 6.6 shows a typical space-time diagram from ECA rule 40. After the initial transients have died down, the configurations are spatially and temporally uniform, and the single regular domain consists of all sequences of contiguous 0s, i.e., $\Lambda^0_{40} = 0^*$. This is verified by first showing that the domain is temporally invariant ($\Phi(0^*) = 0^*$), and then determining that the process graph for 0^* is strongly connected. As depicted in Figure 6.7(a), there is a single recurrent state in the process graph, which trivially satisfies the homogeneity condition. The transducer for ECA rule 40, $T_{\Lambda^0_{40}}$, is shown in Figure 6.7(b). In this and in all subsequent figures, thick directed edges represent state transitions within regular domains and thin lines depict the transitions due to synchronization, walls, and re-synchronizations. Here and in later figures, the transducer maps all domains to the symbol 0, and all walls to the symbol 1.

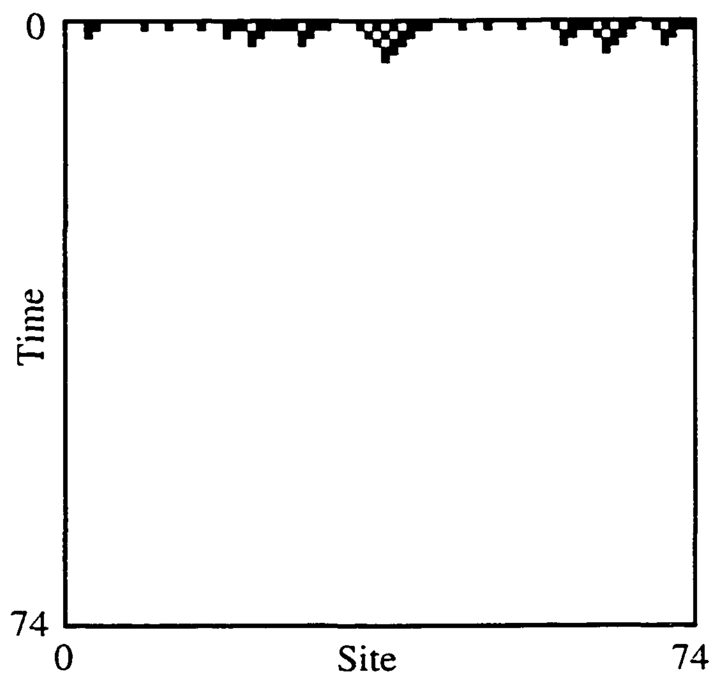


Figure 6.6: (a)Space-time diagram of ECA rule 40 starting with a random initial configuration (After [Han93].)

6.4.2 ECA Rule 55

A typical space-time diagram from ECA rule 55 is shown in Figure 6.8(a). In this case the domain that is visually most apparent consists of contiguous blocks of cells of a single value which oscillates between 0 and 1 in alternate time steps. Thus the domain has a temporal periodicity of two. The process graph for this domain is denoted as Λ^0_{55} and it is shown in Figure 6.9(a)[1]. Being a regular domain, Λ^0_{55} is temporally invariant, and at any instant in time, its process graph is strongly connected, albeit trivially. The corresponding transducer $T_{\Lambda^0_{55}}$ is presented in Figure 6.9(b). When $T_{\Lambda^0_{55}}$ is applied to the space-time diagram in Figure 6.8(a), it produces the filtered diagram as depicted in Figure 6.8(b). A comparison of the two figures

[1]A less obvious domain in ECA rule 55 is $\Lambda^1_{55} = (00)0^*(11)1^*$. Unlike Λ^0_{55}, this domain has a temporal periodicity of one, and is also temporally invariant, and, spatially homogeneous. Indeed, the two domains are nested, with $\Lambda^0_{55} \subset \Lambda^1_{55}$.

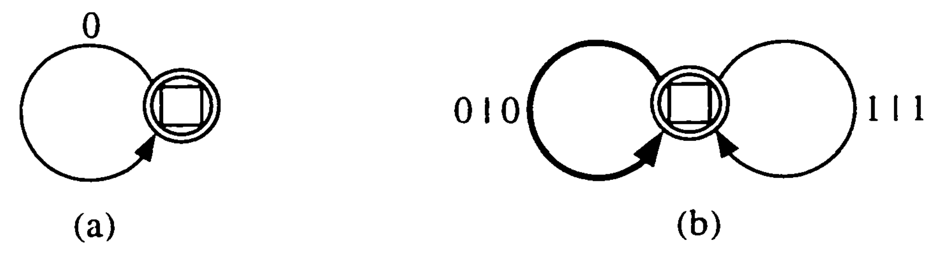
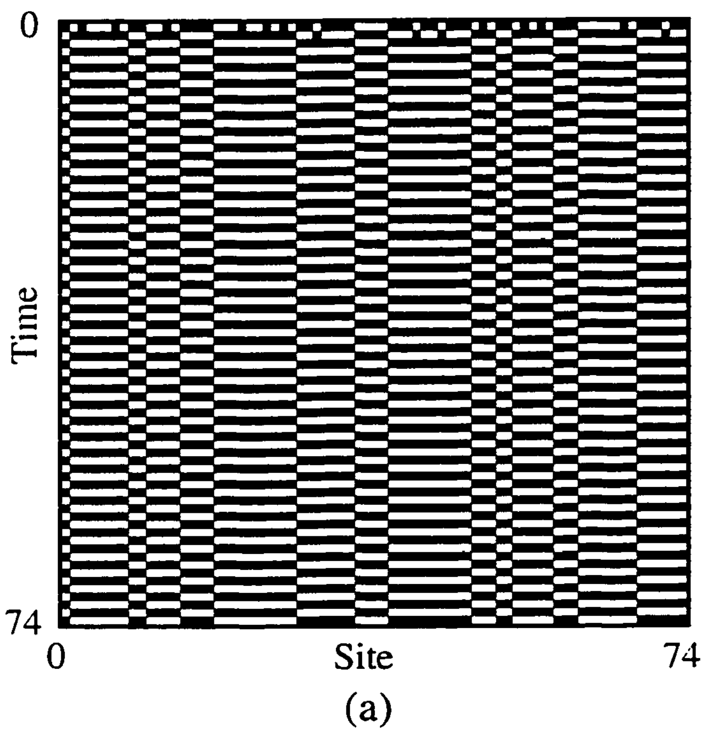


Figure 6.7: A minimal DFA representing $\Lambda^0_{40} = 0^*$. (b) Domain transducer $T_{\Lambda^0_{40}}$. (After [Han93].)

clearly indicates that the domain walls identified by $T_{\Lambda^0_{55}}$ in Figure 6.8(b) match with the visually apparent domain walls in Figure 6.8(a). In essence, starting from any random initial configuration, ECA rule 55 forms noninteracting subregions which are in the domain Λ^0_{55} and which are out of phase with the adjacent domains.

Given the filtered space-time diagram in Figure 6.8(b) with the prominent walls and domains, can the framework of computational mechanics be reapplied to further understand the space-time behavior of ECA rule 55? The answer is indeed yes. Figure 6.10(a) depicts the process graph for the domain characterizing the space-time diagram in Figure 6.8(b). It shows that when the domains and walls are represented with the symbols 0 and 1 respectively, the resulting filtered space-time diagram from ECA rule 55 can be described by another domain $\Lambda^0_{T_{\Lambda_{55}}} = (0^*100)^*$. The transducer constructed from this process graph is shown in Figure 6.10(b). When the transducer is applied to the space-time diagram in Figure 6.11(a), the plot shown in Figure 6.11(b) is produced. The key point here is that there are no walls in Figure 6.11(b), signifying that a complete computational characterization of ECA rule 55's behavior—i.e., a complete delineation of the regularities in terms of intrinsic computational structures—has been achieved.



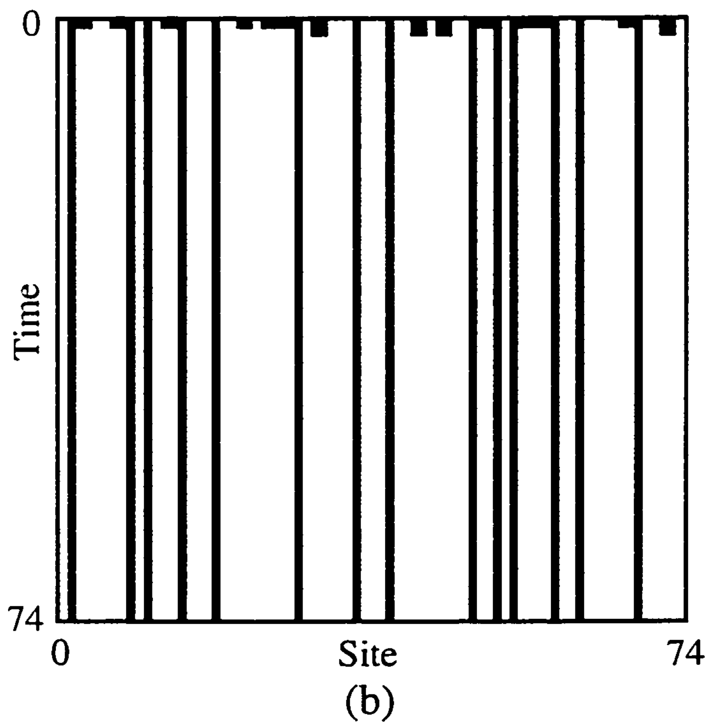


Figure 6.8: (a) Space-time diagram of ECA rule 55 starting with a random initial configuration. (b) The same space-time diagram as in (a), but after filtering with the spatial transducer $T_{\Lambda^0_{55}}$. All domains have been mapped to 0 (white) and all walls to 1 (black). (After [Han93].)

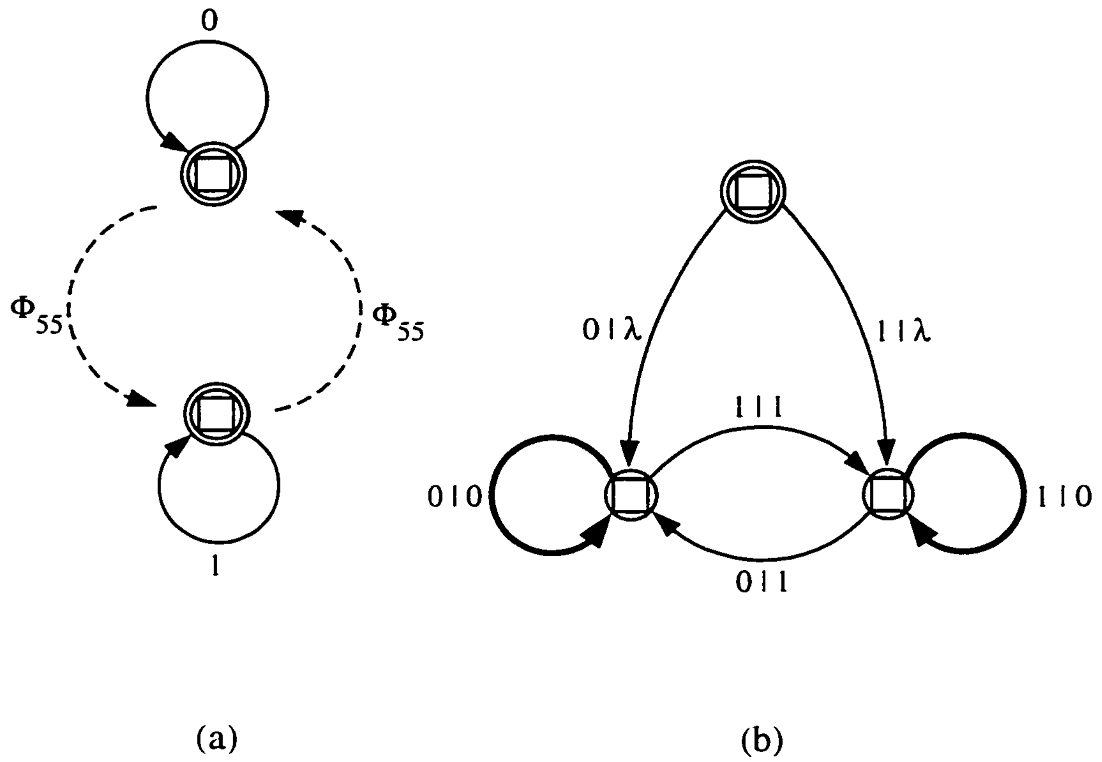


Figure 6.9: (a) The process graph representing Λ^0_{55}. (b) Domain transducer $T_{\Lambda^0_{55}}$. (After [Han93].)

6.4.3 ECA Rule 54

Compared to ECA rule 40 or 55, the space-time diagram generated by ECA rule 54 is considerably more complicated (Figure 6.14(a)). However, it is possible to visually identify regions within the space-time configuration that seem to have some form of regularity. In particular, regions in the configuration obeying the regularity display spatial and temporal periodicity of four. However, the regular domain Λ^0_{54} which precisely characterizes this regularity has a temporal periodicity of two and a spatial periodicity of four, with $\Phi(1000)^* = (0111)^*$ and $\Phi(0111)^* = (1000)^*$. The process graph for Λ^0_{54} is presented in Figure 6.12, and the corresponding transducer $T_{\Lambda^0_{54}}$ is portrayed in Figure 6.13. After filtering the space-time diagram in Figure 6.14(a) using $T_{\Lambda^0_{54}}$, the diagram in Figure 6.14(b) is obtained.

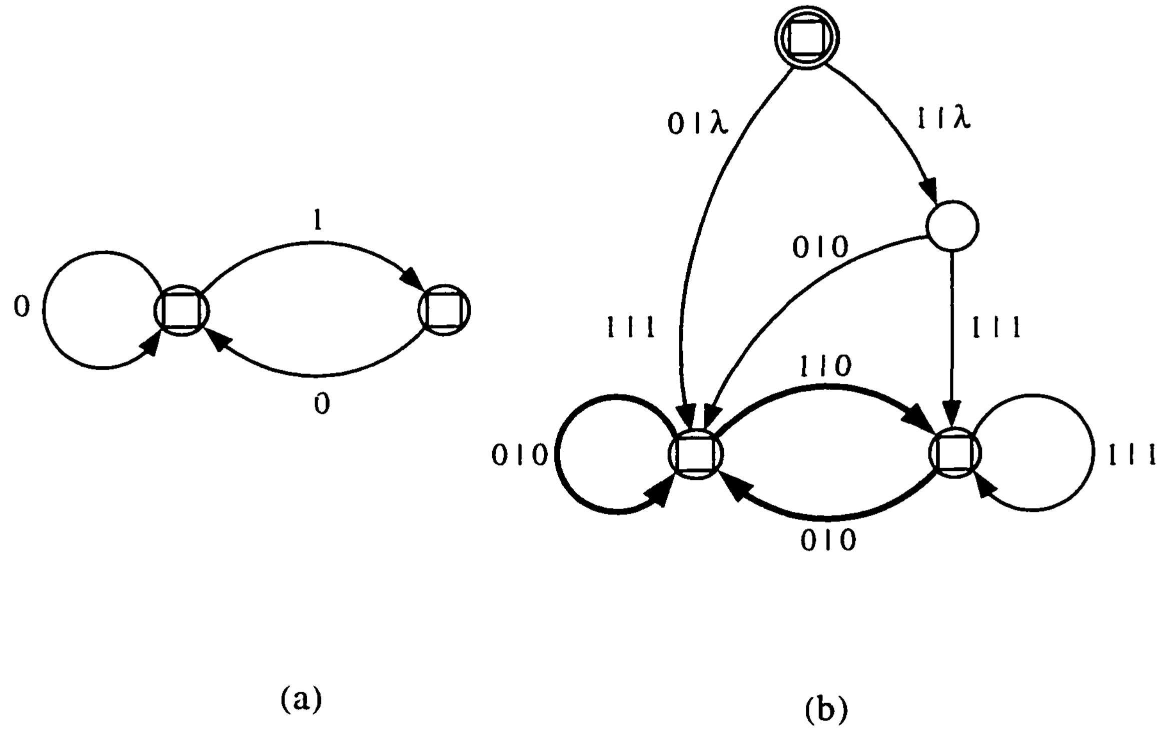
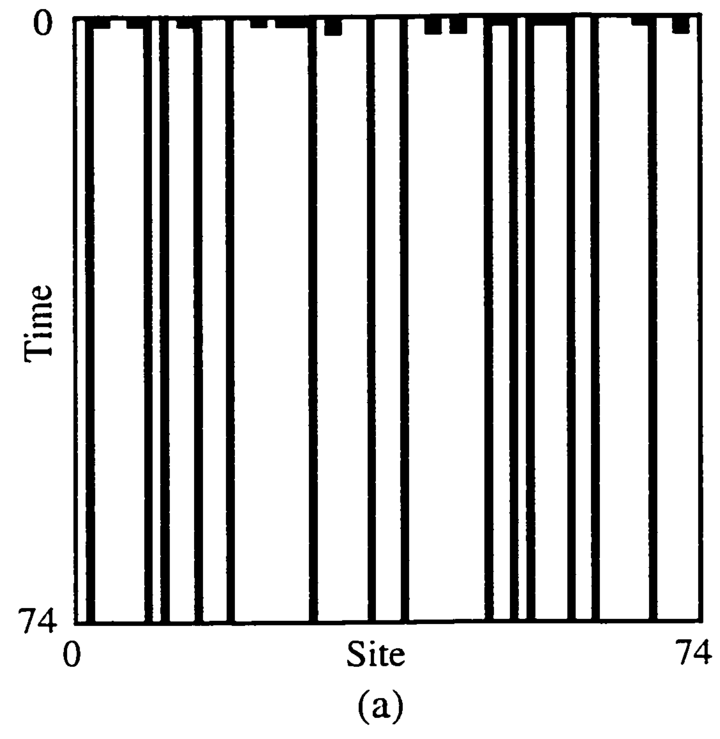


Figure 6.10: (a) The process graph representing $\Lambda^0_{T_{\Lambda 55}} = 0^*100$. (b) Domain transducer for the process graph in (a). (After [Han93].)

The most interesting feature in the filtered diagram in Figure 6.14(b) is the rich dynamics displayed by the walls. Moreover, some of the walls exhibit a structure that is not only spatially coherent but also temporally periodic. Such walls which travel with a constant speed across space-time are labeled as *particles*. A careful inspection of the figure shows that it is possible to classify the particles into four different types. An example of each type of particle is shown in detail in Figure 6.15 and its characteristics are given in Table 6.1. The temporal periodicity of a particle is determined by the time interval between occurrences of the same spatial structure in the particle.

While the counterparts of the particles α and β in the original space-time diagram (Figure 6.14(a)) are visually recognizable, the particles γ and δ are harder to



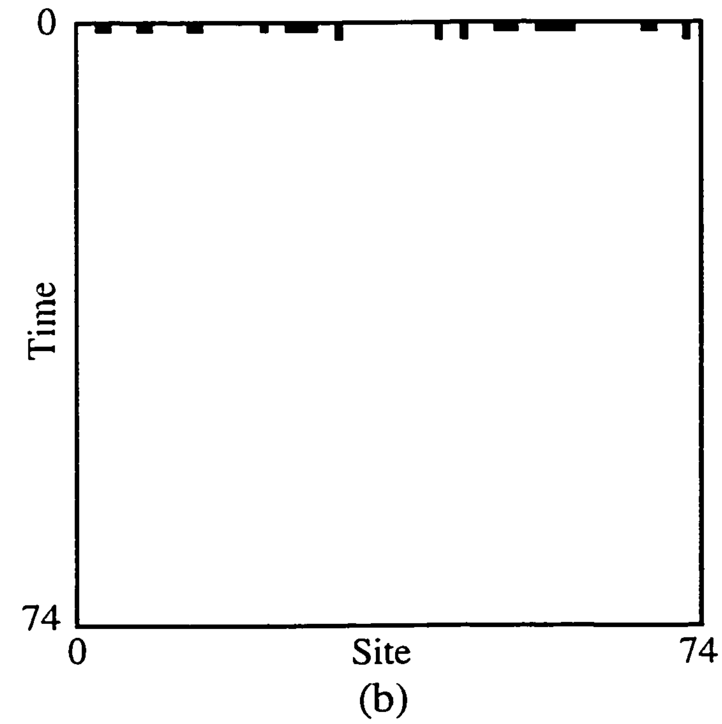


Figure 6.11: (a) The same filtered space-time diagram of ECA 55 as shown in Figure 6.8 (b). (b) The same space-time diagram as in (a), but after the domains $\Lambda^0_{T_{\Lambda^0_{55}}} = (0^*10)^*$ have been identified. All domains have been mapped to 0 (white) and all walls to 1 (black). (After [Han93].)

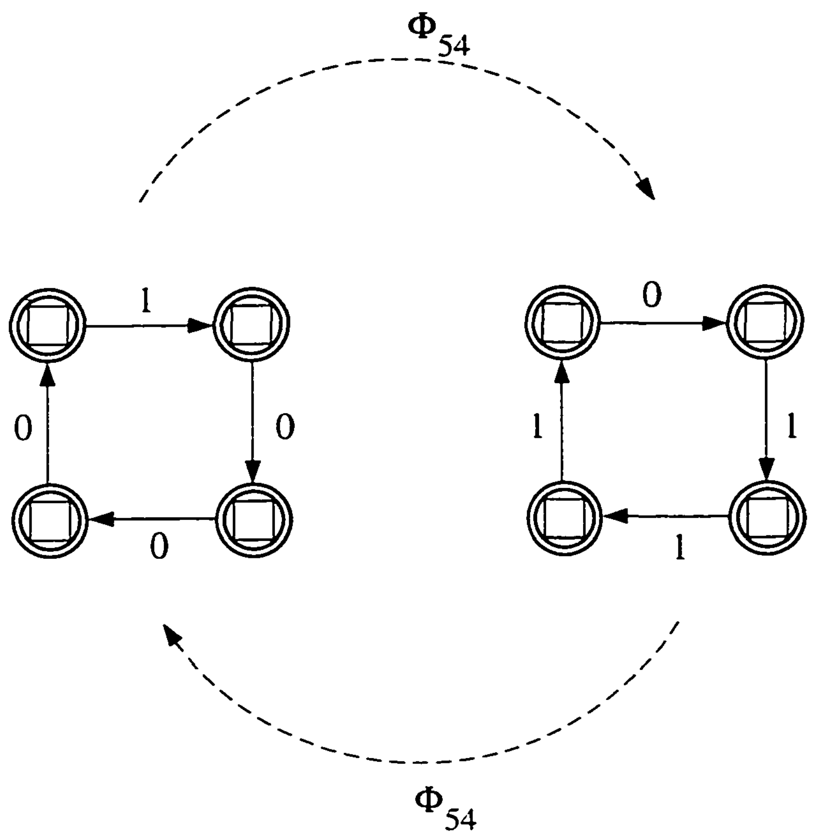


Figure 6.12: The process graph representing Λ^0_{54}. (After [Han93].)

visually identify in the unfiltered diagram. This is primarily because these two latter particles are walls with zero thickness—i.e., they are dislocations that introduce a small phase shift in the domain Λ^0_{54}.

In addition to the different kinds of particles observed in Figure 6.14 (b), several different types of interactions among the particles are recognizable:

(i) $\gamma + \beta \rightarrow \delta$,

(ii) $\beta + \delta \rightarrow \gamma$,

(iii) $\gamma + \delta \rightarrow \beta$,

(iv) $\gamma + \beta + \delta \rightarrow \emptyset$,

(v) $\gamma + \alpha \rightarrow \gamma + \alpha + \delta$,

(vi) $\alpha + \delta \rightarrow \gamma + \alpha + \delta$,

$\emptyset$ indicates a domain with no particles.

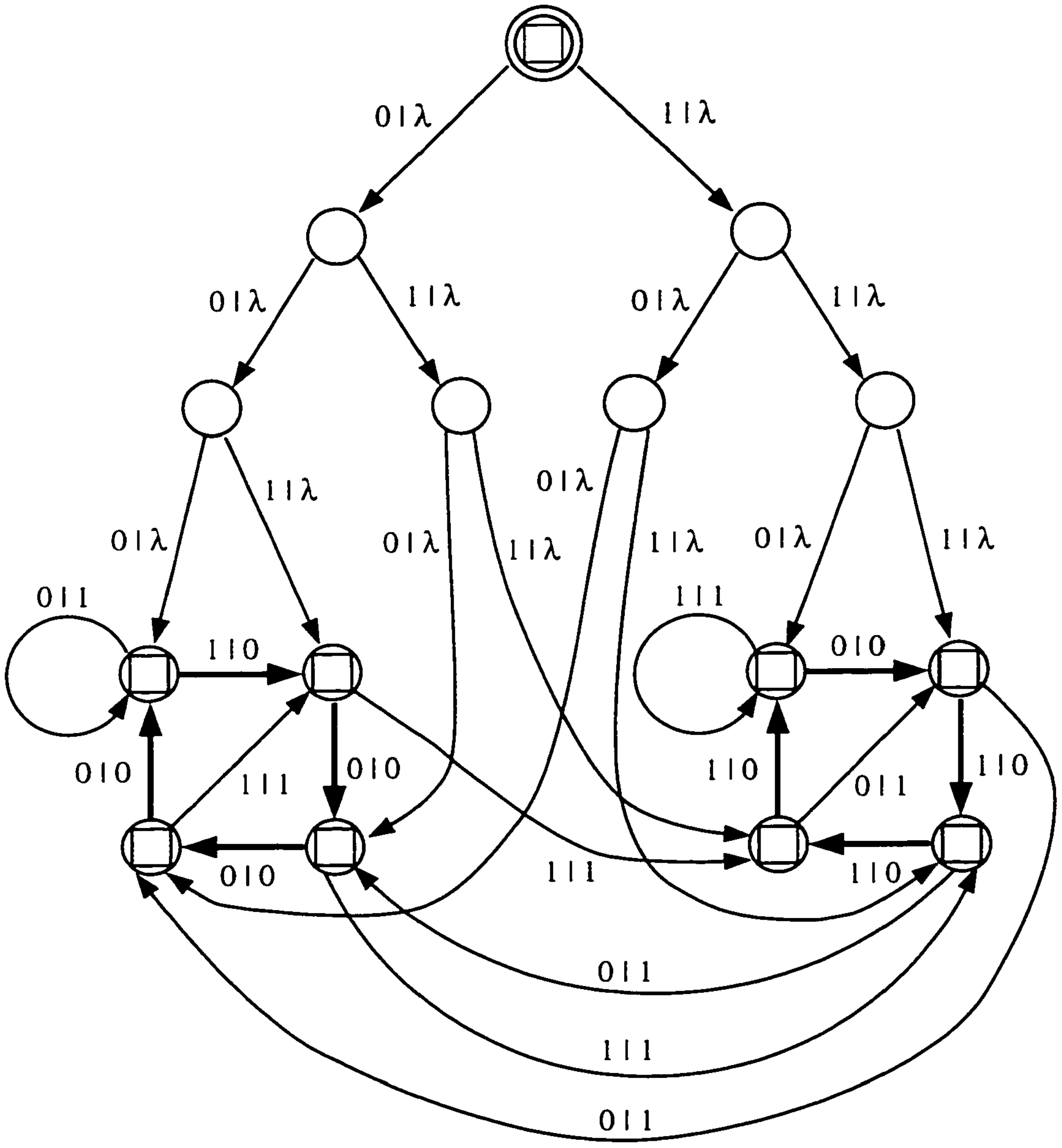
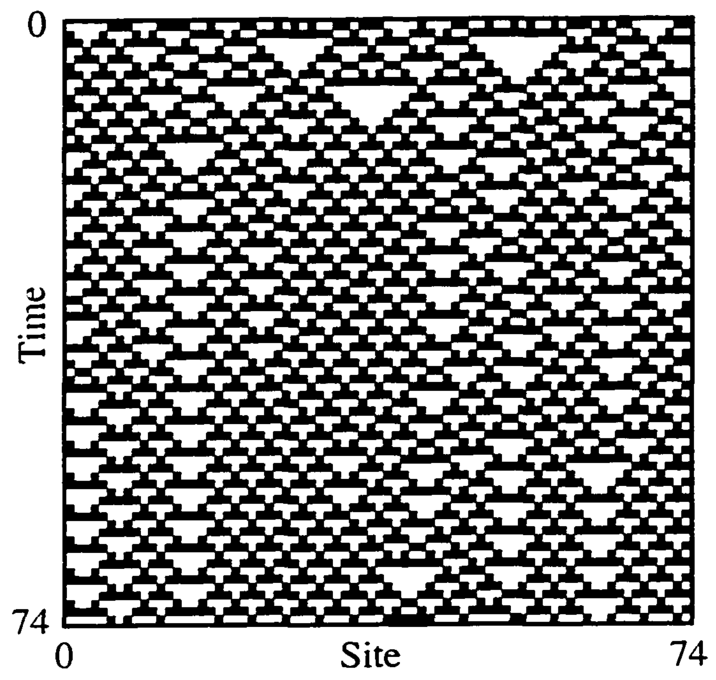
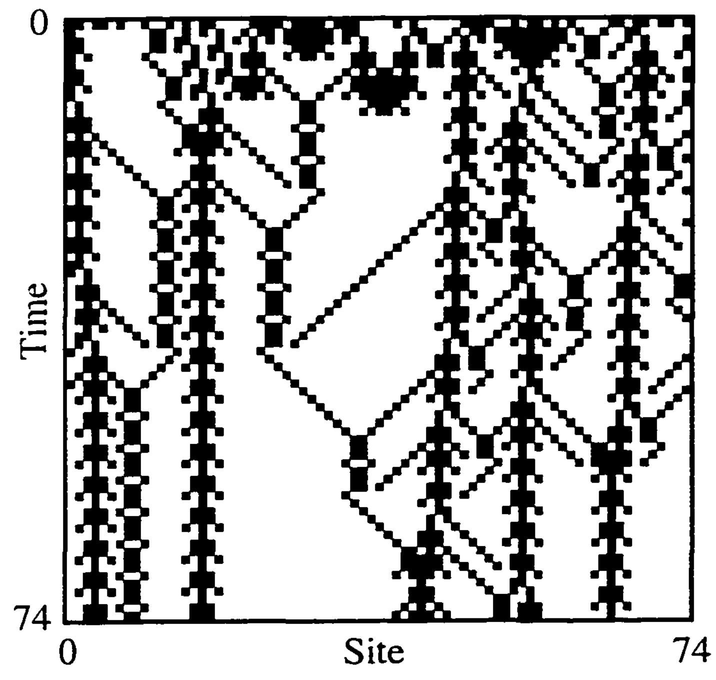


Figure 6.13: Domain transducer for the process graph Λ^0_{54}. (After [Han93].)



(a)



(b)

Figure 6.14: (a) Space-time diagram of ECA rule 54 starting with a random initial configuration. (b) The same space-time diagram as in (a), but after filtering with the transducer $T_{\Lambda^0_{54}}$. All domains have been mapped to 0 (white) and all walls to 1 (black). (After [Han93].)

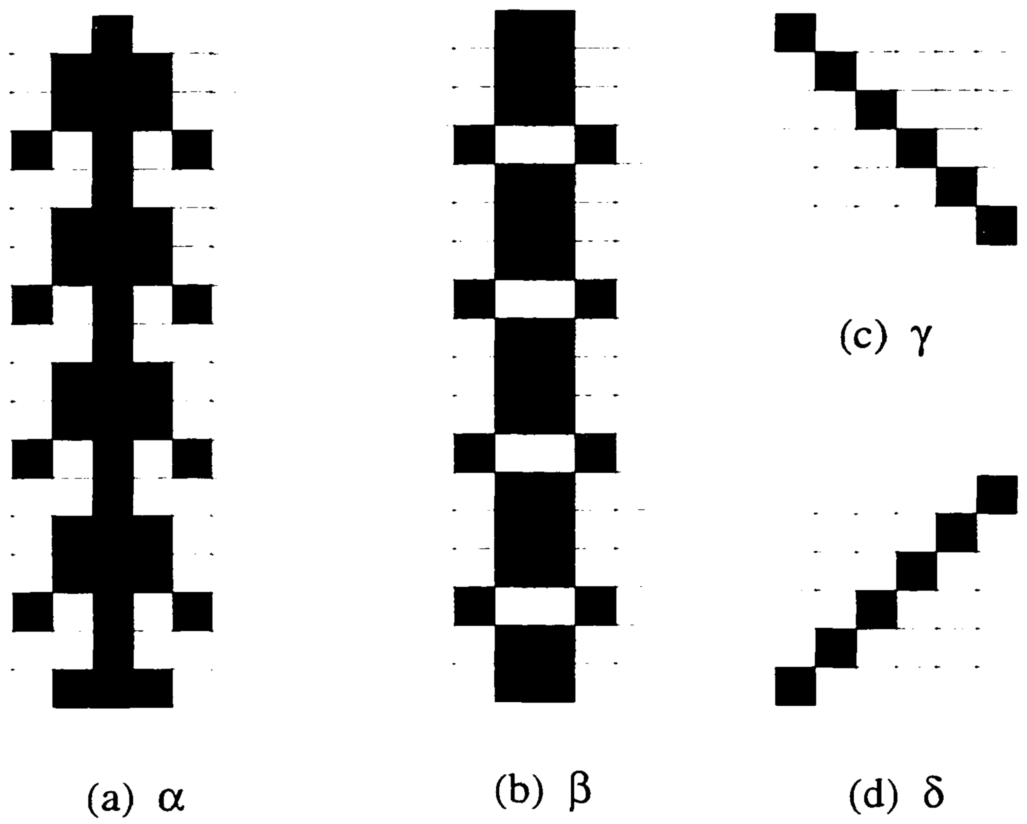


Figure 6.15: Details of the four kinds of walls (or particles) which occur in the space-time diagram produced by ECA rule 54. (After [Han93].)

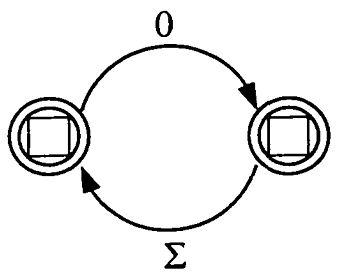
It is interesting to note the different types of interactions. While reaction (iv) represents a three particle annihilation, the number of particles increase in the reactive reactions (v) and (vi), but decrease in the reactions (i), (ii), and (iii). Also, there are obvious symmetries in the behavior displayed by particles γ and δ. A more in-depth investigation of ECA rule 54's behavior should include a study of these particle interactions. Indeed, Boccara et al. have performed such a particle-level analysis and have characterized the long time behavior of ECA rule 54 [BNR91].

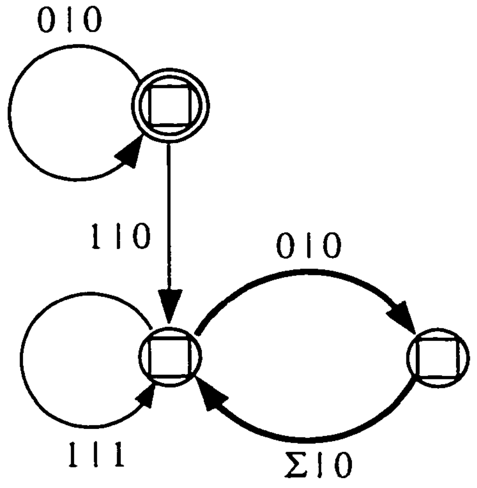
6.4.4 ECA Rule 18

So far, in all the examples of space-time diagram in this section, the underlying regular domains were visually identifiable. This is always because there was some form of spatial and temporal uniformity or periodicity in the domains. In contrast,

Particle	Temporal Periodicity	Spatial Displacement	Velocity
α	4	0	0
β	4	0	0
γ	1	1	1
δ	1	-1	-1

Table 6.1: The characteristics of the four particles embedded in the space-time behavior of ECA rule 54. By convention, a displacement to the left is assigned a negative value. The velocity of a particle is determined by dividing its spatial displacement by its temporal periodicity.



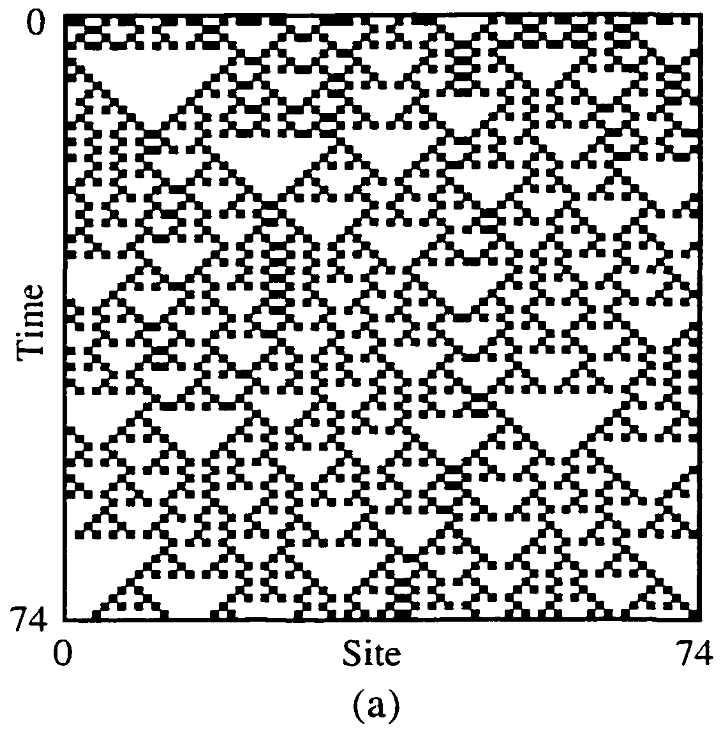


(a) (b)

Figure 6.16: (a) The process graph representing Λ^0_{18}. (b) Domain transducer $T_{\Lambda^0_{18}}$. The symbol Σ denotes either of the symbols 0 or 1. (After [Han93].)

the regular domain present in the space-time diagram of ECA rule 18 has configurations which are spatially disordered (Figure 6.17 (a)). Thus not only the domains, but also the domain boundaries are difficult to identify by casual inspection.

Using machine reconstruction techniques Crutchfield and Hanson have shown that the regular domain in ECA rule 18 can be characterized as $\Lambda^0_{18} = (0(1 \cup 0))^*$ [CH93]. Thus, configurations in this domain have every other site set to 0, while the rest of the sites can be set to either a 1 or a 0. This is shown in the process graph of Λ^0_{18} in Figure 6.16 (a). The domain boundaries between adjacent domains represent



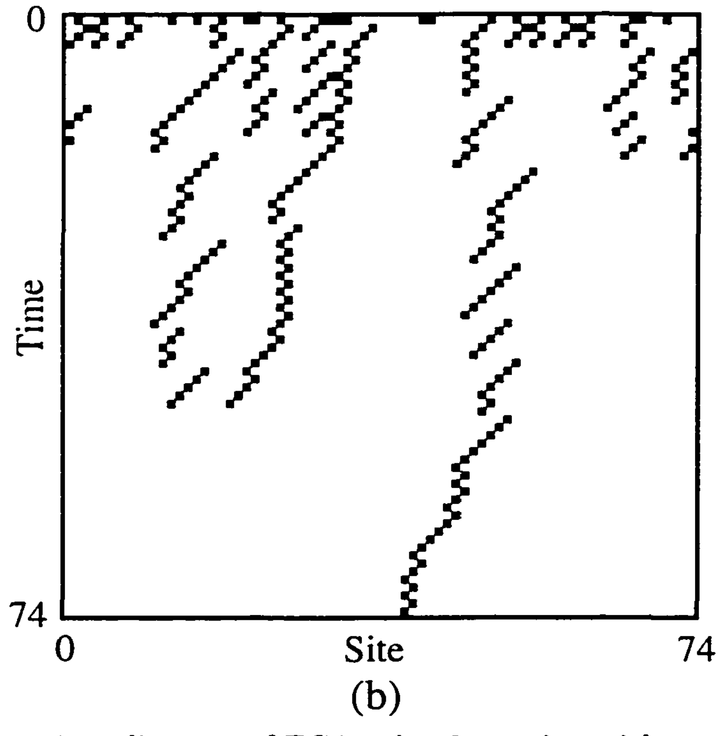


Figure 6.17: (a) Space-time diagram of ECA rule 18 starting with a random initial configuration. (b) The same space-time diagram as in (a) after it has been filtered with the transducer $T_{\Lambda^0_{18}}$. All domains have been mapped to 0 (white) and all walls to 1 (black). (After [Han93].)

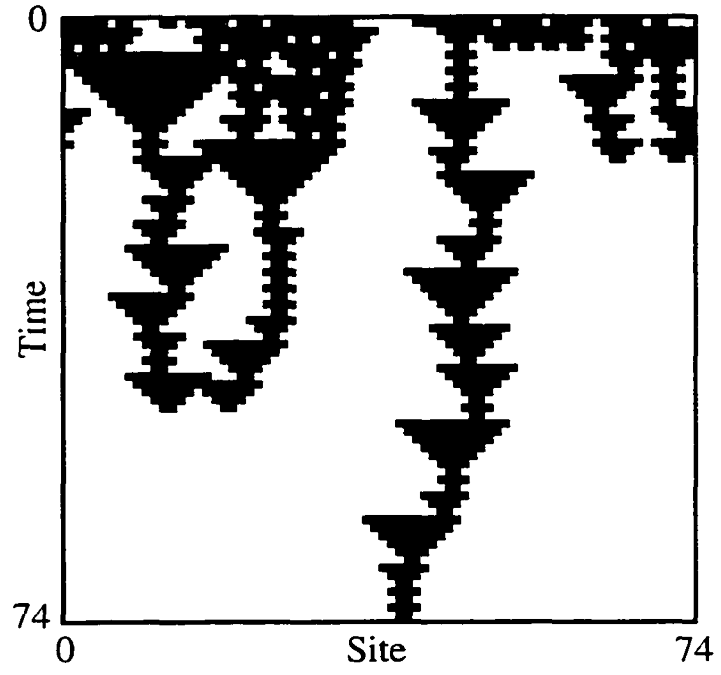


Figure 6.18: The same space-time diagram as in Figure 6.17 (a), but after it has been filtered with a "symmetrized" transducer which recognizes strings in the language $1(00)^*1$ as defects. All domains have been mapped to 0 (white) and all walls to 1 (black). (After [Han93].)

dislocations that are spatial phase slips. Dislocation occurs whenever there is an even number of 0s between two 1s in a sequence. The dislocations move as if they are performing random walk in space-time. When they meet in pairs they annihilate each other.

Figure 6.16 (b) shows the domain transducer $T_{\Lambda^0_{18}}$ constructed from the process graph Λ^0_{18}. The filtered plot of the original space-time diagram is presented in Figure 6.17 (b). In the figure, the domain boundaries between adjacent domains represent dislocations that are spatial phase slips in Λ^0_{18}. The dislocations, which occur whenever there is an even number of 0s between two 1s in a sequence, move as if they are performing random walk in space-time and when they meet in pairs, they annihilate each other.

Since the transducer $T_{\Lambda^0_{18}}$ scans each configuration from the left to the right, it introduces a spatial asymmetry in the filtered plot shown in Figure 6.16 (b). More

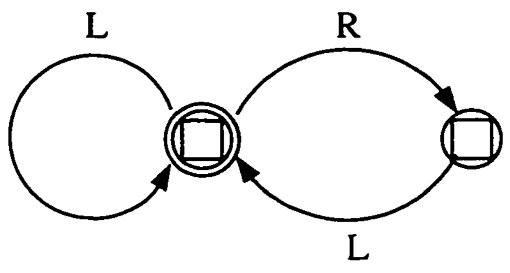


Figure 6.19: Machine reconstructed from the dynamics of a single dislocation in Λ^0_{18}. The symbols L and R denote motion to the left and the right respectively. (After [Han93].)

specifically, the dislocations seem to be discontinuous as they occasionally "jump" to the right. To overcome this problem, a "symmetrized" transducer can be built that recognizes strings in the language $1(00)^*1$ as defects. The output from a such a transducer is shown in Figure 6.18.

6.5 Computation with Particles

One of the important conclusions to draw from the previous sections is that because the regular domains are computationally homogeneous, there is no nontrivial information processing or information transmission occurring within the domains. Additionally, the amount of information stored in a domain is proportional to the number of recurrent states in its process graph. Thus the regions within the domains are severely restricted in terms of their information processing capability.

On the other hand, walls which maintain their spatial and temporal coherency can be the main mechanisms through which information is processed in the spatio-temporal behavior of a CA. It is readily apparent that a particle can carry information across long space-time distances. This information might indicate the result of some local processing that has occurred elsewhere at an earlier time. As a simple example, a particle might help to propagate information about a phase slip introduced due to local interactions elsewhere in an earlier time step.

An interesting question to ask at this juncture is, can an individual particle process information all by itself ? Clearly, the individual particles in ECA rule 54 that display periodic behavior in their spatial structures and that move with a constant velocity are limited by their very nature in terms of their computational capability. In contrast, particles in ECA rule 18 display apparently irregular behavior. Figure 6.17 (b), shows that each move to the left shifts the position of the particle by a single cell, whereas moves to the right may be of any odd-valued distance. To make a quantitative estimate of the computational capability of the particle, the framework of computational mechanics can again be used. After a dislocation has been detected with the transducer $T_{\Lambda^0_{18}}$, the motion of the particle is recoded as a symbol string, where a move to the left (right) is interpreted as the symbol L (R). The machine reconstruction technique can be applied to this symbol sequence; and the resulting machine is shown in Figure 6.19. The machine makes it clear that the dislocation in ECA rule 18 can move to the left any number of times, but a move to the right must be followed by a move to the left. This severely constrains the computational capability of a particle in ECA rule 18. A similar approach using machine reconstruction techniques can be adopted to determine the computational capability of arbitrary particles.

The collection of domains and particles represents the basic computational structures in the spatio-temporal behavior of a CA. Such structures can interact with one another and give rise to information processing in the system. In general, logical operations on the information carried by the particles is performed when the particles meet. This is readily observed in ECA rule 18 where particles pairwise annihilate, and in ECA rule 54 where a small number of particles exhibit rich interaction dynamics.

It is important to note that the computational structures discovered in a CA by the computational mechanics framework is independent of any "useful" computation the CA may or may not be performing. In the following two chapters, an attempt will be made to use the computational mechanics framework to study the spatio-temporal behavior of the high-performance CA rules discovered by a GA. The goal is to characterize and quantify the basic elements of computation in the CA's behavior. and to determine how these computational structures are used by the CA in achieving superior performance in tasks requiring global coordination.

Chapter 7

DENSITY CLASSIFICATION TASK

The main goal in this chapter is to understand the spatio-temporal behavior of the high-performance CAs for the density classification task $\mathbf{T}_{1/2}$ using the framework of computational mechanics. In contrast to that of the less fit block-expanding rules described in Chapter 5, the space-time behaviors of the high-performance rules are dominated by visually identifiable patterns. The aim here is to characterize the pattern dynamics in these CAs in terms of domains, particles and particle interactions, and delineate how they aid the CAs in achieving high-performance. In addition, the particle-level analysis is employed to portray the evolutionary pathway through which the GA discovered the high-performance CAs. The goal is to study the ancestors of the high-performance CAs and then delineate the evolution of the computational structures that are embedded in the dynamics of the CAs.

This chapter also investigates a related issue. The range of the spatial and temporal regularities in the patterns displayed by different high-performance rules is indeed remarkable, but the particle-level analysis indicates higher-level similarities among all the high-performance rules that the GA discovered. Another goal in this chapter is to elucidate such similarities.

For these investigations, several representative high-performance rules with different look-up tables have been selected and a detailed analysis of their behavior is performed. The three rules are:

(i) ϕ_A = 0x 05040587 05000F77 03775583 7BFFB77F,

(ii) ϕ_B = 0x 10564424 0140149D 1F7746F7 EF5B3F7F, and

(iii) ϕ_C = 0x 01000131 0D60052F 1F7BD7B5 F5FFFD7F.

Each CA ϕ is given as a hexadecimal string which, when translated to a binary string, gives the output bits of ϕ in lexicographic order ($\eta = 0^7$ on the left). Although the analysis presented here deals with these rules, the main conclusions drawn from the analysis are sufficiently general in their scope, and can be easily extended to understand the behavior of other high-performance rules for the density classification task.

7.1 Analysis of ϕ_A

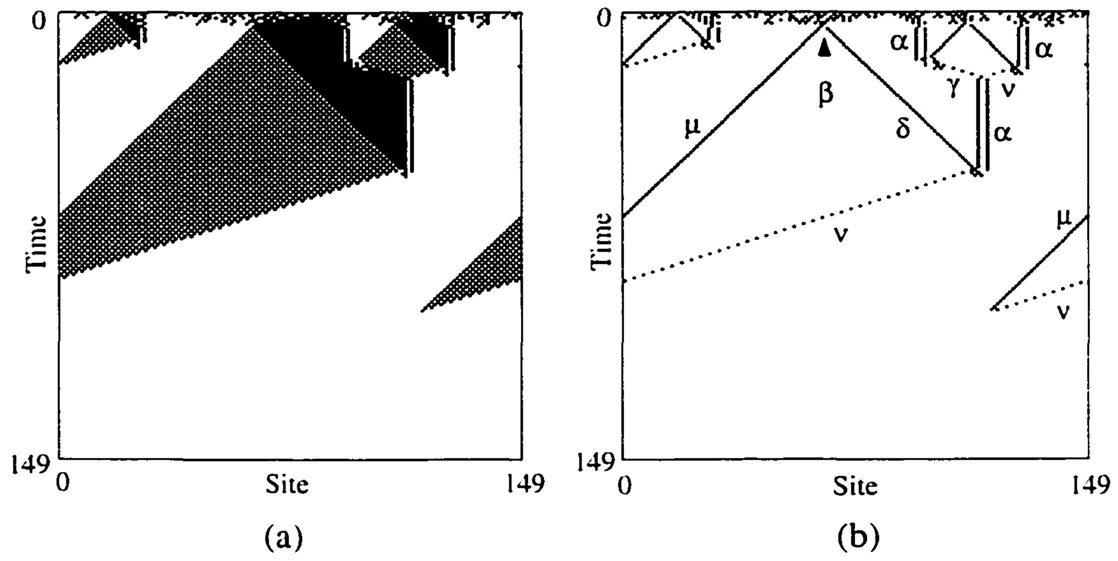


Figure 7.1: (a) Space-time diagram of ϕ_A starting with a random initial condition with $\rho_0 = 0.4765$. (b) The same space-time diagram after filtering with the transducer T_A which maps all domains to white and all defects to black. Greek letters label particles described in the text.

Figure 7.1(a) gives a space-time diagram for ϕ_A, the highest performing CA discovered by the GA in the experiments. The unbiased performance measure $\mathcal{P}^N_{10^4}(\phi_A) = 0.769$, 0.766, and 0.757 for $N = 149, 599$, and 999 respectively. The figure shows that the space-time behavior of ϕ_A is dominated by three types of patterns: regions with the all-1s pattern, regions with the all-0s pattern, and regions with the checkerboard pattern. These three patterns can be characterized as regular

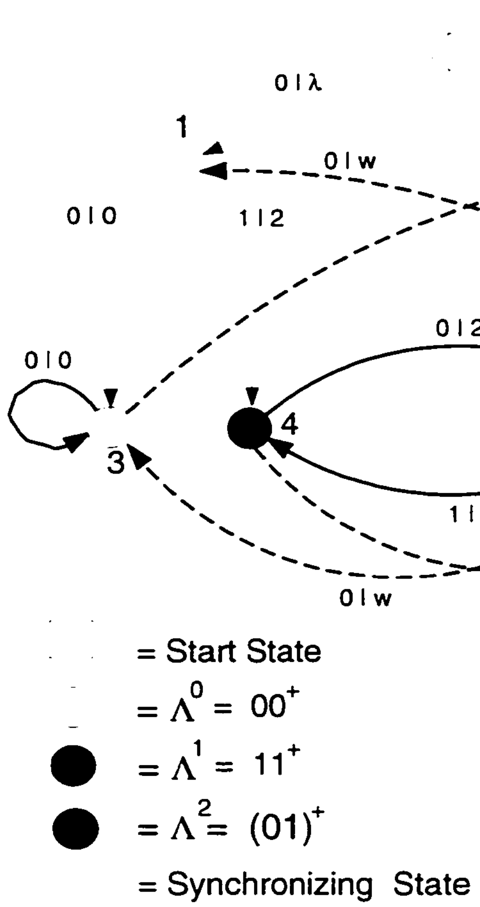


Figure 7.2: The spatial transducer T_A which accepts words in all three domains Λ^0_A, Λ^1_A, and Λ^2_A. The same transducer can be used to filter space-time plots generated by ϕ_{GKL} (described in the text).

domains since each satisfies the temporal invariance and spatial homogeneity conditions. The domains, denoted here as Λ^0_A, Λ^1_A, and Λ^2_A, consists of the languages 00^+, 11^+, and $(01)^+$ respectively. In this chapter, Λ^0_A, Λ^1_A, and Λ^2_A will also be denoted by symbols W, B, and # respectively.

Figure 7.2 shows the spatial transducer T_A, which accepts words in all the three domains, and, when applied to the space-time diagram in Figure 7.1(a), produces the plot in Figure 7.1(b). In Figure 7.1(b), the filtering helps in determining the location as well as the spatial and temporal features of the particles in the space-time diagram. In addition, various types of particle interactions can be distinguished in the figure. The domains, particles and particle interactions observed in the filtered space-time plots of ϕ_A are presented in Table 7.1.

Domains		
$W = 00^+$	$B = 11^+$	$\# = (01)^+$

Particles				
Symbol	Domain Boundary	Temporal Periodicity	Spatial Displacement	Velocity
α	BW	1	0	0
β	WB	-	-	-
γ	$B\#$	1	3	3
δ	$\#B$	1	1	1
μ	$W\#$	1	-1	-1
ν	$\#W$	1	-3	-3

Particle Interactions			
Decay	$\beta \to \mu + \delta$		
Annihilative	$\mu + \nu \to \emptyset$	$\gamma + \delta \to \emptyset$	
Reactive	$\delta + \alpha \to \nu$	$\alpha + \mu \to \gamma$	$\gamma + \nu \to \alpha$

Table 7.1: The domains, particles, and particle interactions which dominate the spatio-temporal behavior of ϕ_A. The domains, particles, and particle interactions embedded in the behavior of ϕ_{GKL} are identical to those presented in the above table.

Why does ϕ_A perform relatively well on the $\rho_c = 1/2$ task? In Figure 7.1(a) it can be seen that, although the patterns eventually converge to fixed points, there is a transient phase during which a spatial and temporal transfer of information about the density in local regions takes place. This local information interacts with other local information to produce the desired final state. Roughly, ϕ_A successively classifies "local" densities with a locality range that increases with time. In regions where there is some ambiguity, a "signal" is propagated. This is seen either as a checkerboard pattern propagated in both spatial directions or as a vertical black-to-white boundary. These signals indicate that the classification is to be made at a larger scale.

The following section elucidates the above intuitive reasoning in detail. Moreover, in the following, the computational mechanics approach to CAs is used to analyze the strategy of ϕ_A—the rule with highest $\mathcal{P}^N_{10^4}$ found by the GA—and describe the generational progression by which ϕ_A was evolved under the GA.

7.2 The Evolution of Computation

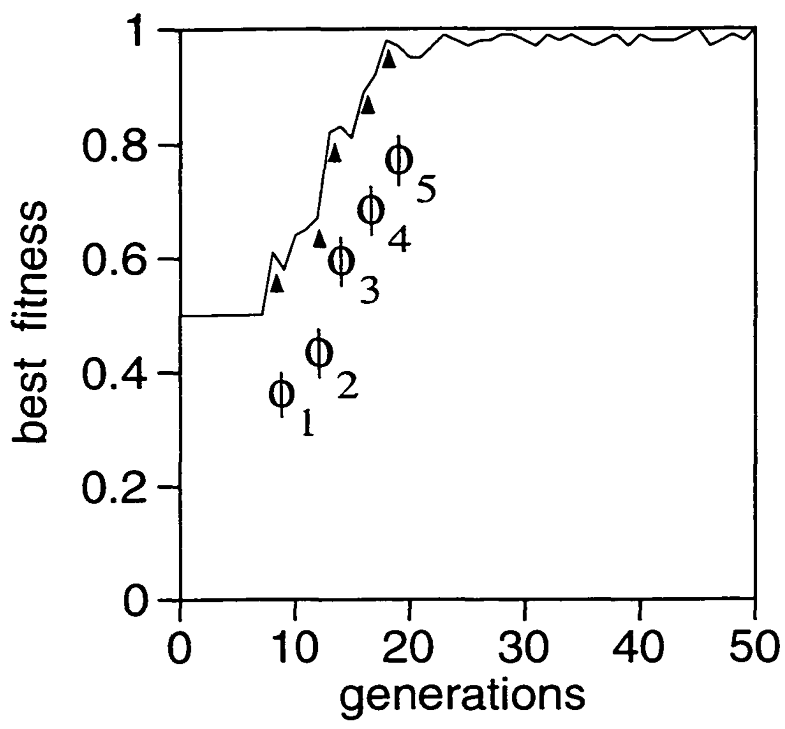


Figure 7.3: Plot of the fitness of the most fit rule in the population versus generation in the run producing ϕ_A. The arrows in the plot indicate the generations in which the GA discovered each new significantly improved strategy.

Figure 7.3 plots best fitness (F_{100}) in the population versus generation for the first 50 generations of the run in which ϕ_A was discovered. It can be seen that, before the GA discovers high fitness rules, the fitness of the best CA rule increases in rapid jumps after periods of relative stasis. Qualitatively, the rise in performance can be divided into several "epochs", each beginning with the discovery of a new, significantly improved strategy for performing the $\rho_c = 1/2$ task. In Figure 7.3, the initial generations of these epochs are labeled with the name of the best rule found at that generation.

Epoch 0: Trivial solutions

For the first seven generations, the highest F_{100} value equal to 0.5 is achieved by a number of rules that map almost all neighborhoods to the same symbol. Rules in which almost all neighborhoods map to 1 (0) quickly settle to the all-1s (all-0s) configuration irrespective of ρ_0. Since the ICs are divided evenly between $\rho < \frac{1}{2}$ and $\rho > \frac{1}{2}$, such rules are able to correctly classify exactly half of the ICs and thus have a fitness of 0.5.

Epoch 1: Discovery of block-expanding rule: expansion of 1-blocks.

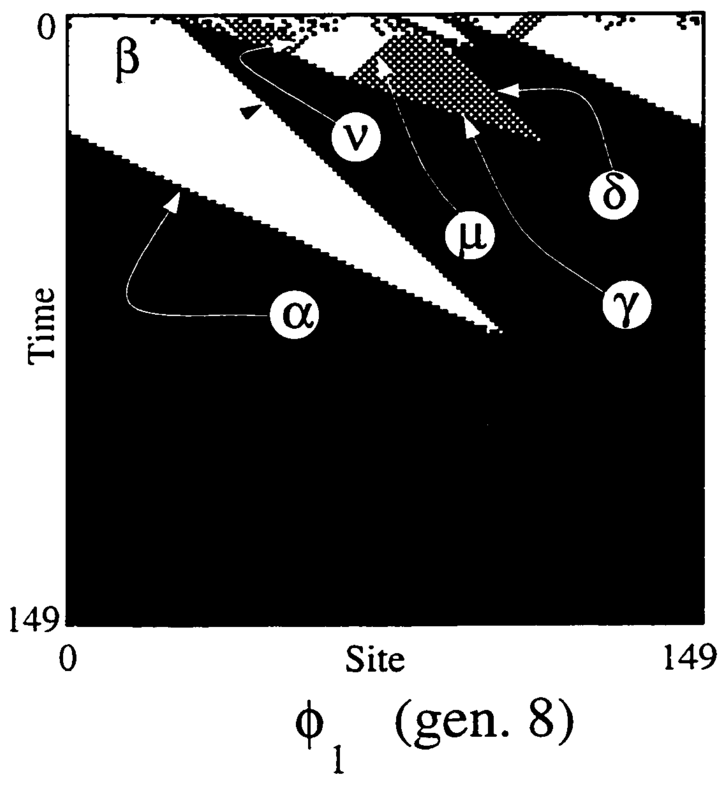


Figure 7.4: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 8, ϕ_1. Randomly generated ICs have been used in Figures 7.4 through 7.8.

In generation 8, a rule (ϕ_1) is discovered that has significantly improved performance ($F_{100} \approx 0.61$). Its strategy is illustrated by the space-time diagram in Figure 7.4. ϕ_1 was created by a crossover between a rule that maps most neighborhoods to 0 and a rule that maps most neighborhoods to 1. A closer inspection of the behavior of this rule reveals that it always relaxes to the all-1s configuration except when ρ_0 is close to zero, in which case it relaxes to the all-0s configuration, yielding $F_{100} > 0.5$. An analysis of particle interactions explains ϕ_1's behavior. After a

Particle	Wall Type	Velocity (Gen. = 8)	Velocity (Gen. = 18)
α	WB	1	0
β	BW	2	0
γ	$B\#$	3	3
δ	$\#B$	1	1
μ	$W\#$	-1	-1
ν	$\#W$	-1	-3

Table 7.2: The velocities of the six particles generated by the best rule in generation 8 (cf. Figure 7.3(b)) and in generation 18 (cf. Figure 7.3(f)).

very short transient phase. ϕ_1 essentially creates three regular domains. $B = 11^+$. $W = 00^+$. and $\# = (10)^+$. This behavior can be understood readily from ϕ_1's rule table: out of the 28 neighborhood patterns with five or more 1s (0s), 27 (22) result in an output bit of 1 (0). Therefore. in any IC, small "islands" of length 1, 2 or 3 containing only 1s or 0s in are quickly eliminated, resulting in locally uniform domains. To maintain a # domain, the neighborhood patterns (1010101) and (0101010) must produce a 1 and 0 respectively, as is done in ϕ_1.

After the B, W, and # domains are created, the subsequent behavior of ϕ_1 is primarily determined by the interaction of the particles representing the domain walls. ϕ_1 has three domains W, B, and #, and six particles, one for each domain wall type: $BW : \alpha$, $WB : \beta$, $B\# : \gamma$, $\#B : \delta$, $W\# : \mu$, and $\#W : \nu$. The velocities of these particles (i.e., the slope of the domain wall corresponding to the particle) are 2, 1, 3, 1, -1, and -1 respectively. There are two annihilative interactions: $\alpha + \beta \rightarrow \emptyset$ and $\gamma + \delta \rightarrow \emptyset$. Two interactions are reactive: $\alpha + \mu \rightarrow \gamma$ and $\gamma + \nu \rightarrow \alpha$. All these interactions can be seen in Figure 7.4. Several particle interactions never occur; they are prevented due to the direction and magnitude of the particle velocities. For example, the particles ν and μ, which are associated with boundary between the W and # domains, have the same velocity and never interact. Similarly, although δ moves in the same direction as α, the former has only half the speed of the latter, and the two particles never have the chance to interact. For similar reasons, the # domain and its four associated particles (γ, δ, μ, and ν) play no appreciable role in the determination of the CA's final configuration. This is easily verified by complementing the output bit of one of the two neighborhood patterns necessary to maintain the # domain (i.e., the neighborhoods (1010101) and (0101010)) and observing that very little change occurs in the rule's fitness.

Thus for this epoch we can focus exclusively on the boundaries between the B and the W regions. Because β's velocity is less than that of α, α soon catches up with β. The interaction of these two walls results in domain B, eliminating domain W between them. Therefore when an island of W is surrounded by B domains, the size of the former shrinks until it vanishes from the lattice. Conversely, when an island of B is surrounded by W domains, the B domain grows until it occupies the whole lattice. However, an island of B must occupy at least five adjacent cells in order to expand. Thus if an initial configuration contains a block of five 1s *or* results in the creation of such a block when the rule is applied over subsequent time steps, then the CA inevitably relaxes to the all-1s configuration. Otherwise it relaxes to the all-0s configuration.

The explanation of ϕ_1's behavior in terms of particles and particle interactions may seem a complicated way to describe simple behavior. But the usefulness of the computational mechanics framework for explicating computation will become clearer as the space-time behavior becomes more complex. In the succeeding generations, it will become clear how the velocity of the particles and their interactions play the most crucial role in determining the computational behavior (and thus the fitness) of a CA rule.

Epoch 2: Refinement of block-expanding strategy

The next four generations produce no new epochs (i.e., no significantly new strategies) but a modest improvement in the best fitness. During this period three changes in behavior were observed, all of which appear in the generation 12 rule ϕ_2 (see Figure 7.5). First, for an island of B to expand and take over the whole lattice, it now must contain at least seven cells. Since a seven-cell island of B is less likely than a five-cell island in low-ρ ICs, more low-ρ ICs are correctly classified.

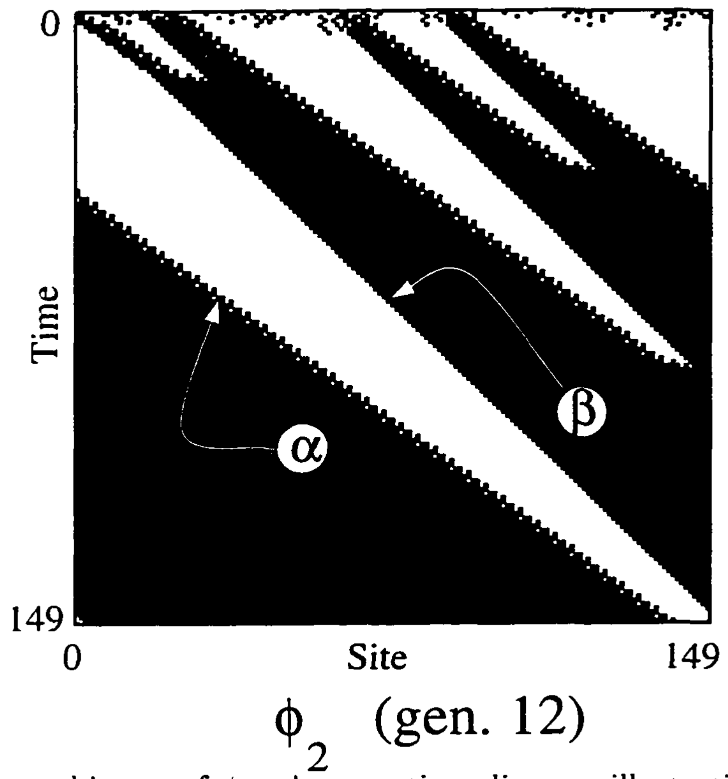


Figure 7.5: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 12, ϕ_2.

Second, the velocity of the $\#W$ boundary, ν, is modified from -1 to -3, allowing the annihilative reaction: $\mu + \nu \rightarrow \emptyset$. Thus, unlike in ϕ_1, an island of $\#$ domain when surrounded by the W domain cannot persist indefinitely. Third, the velocity of α is decreased to $\frac{3}{2}$. Since the velocity of the WB boundary, β remains constant at 1, it now takes longer to eliminate W domains. (This last modification does not result in a significant change in fitness.) Unlike the innovation that resulted in ϕ_1, where crossover played the major role, these modifications (and the ones that follow) are primarily due to the mutation operator. This is because most of above modifications can be attributed to small and isolated changes in the look-up table occurring independently of each other.

Epoch 3: Expansion of 0-blocks

In generation 13, a new epoch begins with the discovery of ϕ_3, with a sharp jump in F_{100} to 0.82 corresponding to a significant innovation. ϕ_3's behavior is

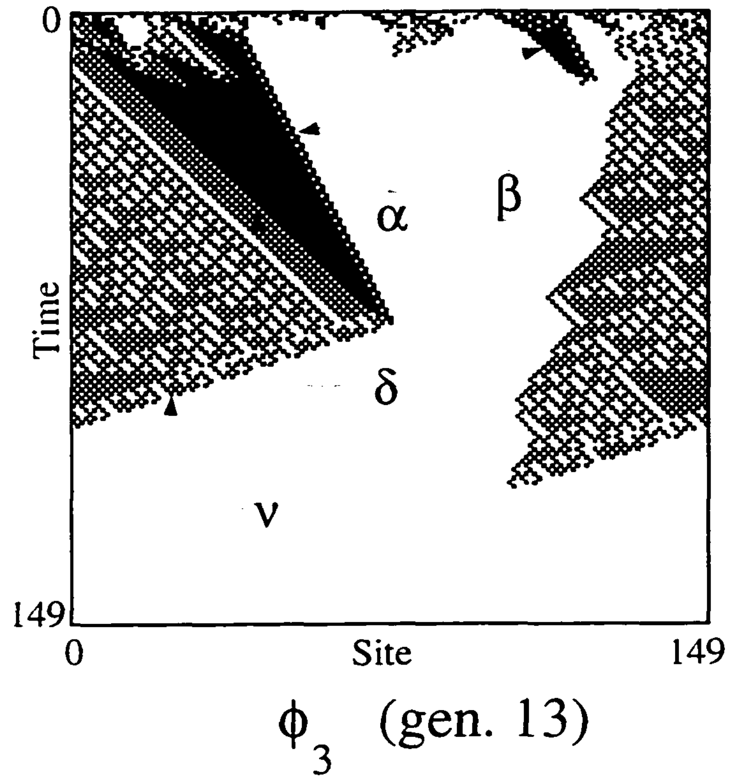


Figure 7.6: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 13, ϕ_3.

illustrated in Figure 7.6 (Here # is used to refer to a more complicated variation of the checkerboard domain #). While the velocity of β is held constant at 1, α now moves with velocity $\frac{1}{2}$ in the same direction. Since the velocity of β is more than that of α, an island of W can now expand when surrounded by B, with the condition that W has to be at least six cells in length.

This means that the rule still defaults to the all-1s configuration, but if there is a sufficiently large island of W in a low-ρ IC (a fairly likely event), the W island expands and the IC is correctly classified. However, misclassification is now possible for $\rho_0 > 0.5$ due to the (less likely) chance formation of an island of W of length six. This aspect of the strategy is again similar to the block-expanding strategy.

In addition to the above changes, a new interaction is allowed to take place for the first time. The decrease in velocity of the particle α to $\frac{1}{2}$ not only results in the removal of small islands of B but it also allows δ to interact with α. The interaction results in the particle ν with a velocity of -3, creating the necessary symmetry with γ, which has velocity +3 (see Table 7.2).. From this juncture the # domain and its four associated particles play a major role in determining the fitnesses of the fittest CA rules.

Epoch 4: The Fundamental Innovation

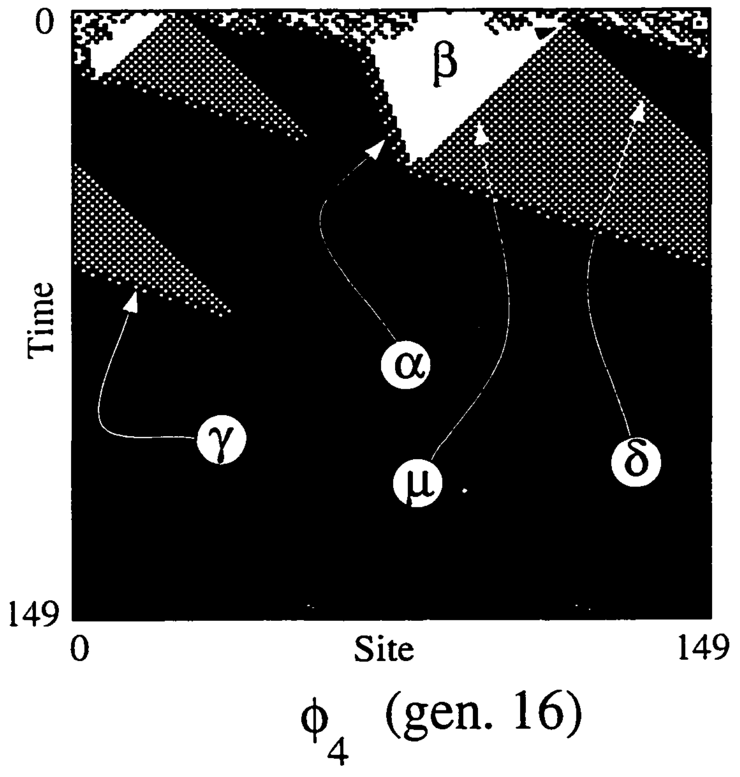


Figure 7.7: Evolutionary history of ϕ_4: A space-time diagram illustrating the behavior of the best rule in generation 16, ϕ_4.

After a brief stasis over three generations, a fundamentally new development results in the improved performance seen in ϕ_4, the best rule in generation 16 (F_{100} = 0.89). In Figure 7.7, the following behavior can be seen. Particle β, which exists at WB boundary, now spontaneously decays to create a # domain with two boundaries, one on each side: μ moving to the left with a velocity of -1, and δ moving to the right with a velocity of 1. Since μ is moving to the left and can interact with right moving α to create γ, the rule will stop the growth of W islands even if their length is greater than neighborhood size. This prevents the error of expanding W blocks when $\rho_0 > 0.5$. In other words, whenever a B island exists with a W island situated to its left, the two domains compete for space in the lattice according to their relative size, with the # domain acting as the mediator. This is a fundamental innovation over the previous epochs in using particles to effect non-local computation.

Also, the velocity of α is further reduced from $\frac{1}{2}$ to $\frac{1}{3}$. However, an asymmetry still remains: because the BW domain boundary, α, is moving to the right with a positive velocity, the magnitude of the velocity difference between the domain boundaries BW (α) and $W\#$ (μ) is more than the velocity difference between BW (α) and $\#B$ (δ). As a result of this asymmetry, it takes longer to remove islands of B than to remove islands of W of the same size, a feature which often leads to misclassifications.

Epoch 5: Final Refinement

The above asymmetry is rectified by the discovery of ϕ_5 in generation 18, in which the velocity of α is set to zero (Figure 7.8), yielding $F_{100} = 0.98$. From generation 19 till the end of the GA run, little change is seen in either the fitness of the best rule in the population or its behavior. ϕ_5's behavior is very similar to that of ϕ_A which was discovered later in this run.

Interestingly, ϕ_A's behavior is very similar to the behavior of the well-known Gacs-Kurdyumov-Levin (GKL) CA (denoted here as ϕ_{GKL}), which was invented to study reliable computation in one-dimensional spatially-extended systems [Gac85]. The look-up-table of GKL CA in hexadecimal is:

$$\phi_{GKL} = \text{050005FF 050005FF 05FF05FF 05FF05FF}$$

and its performance $\mathcal{P}^N_{10^4}(\phi_{GKL})$ was measured to be 0.816, 0.766, and 0.757 for $N = 149$, 599, and 999, respectively. As detailed in Chapter 3, there is a strong similarity between the reliable computation problem as posed by Gacs et al. and

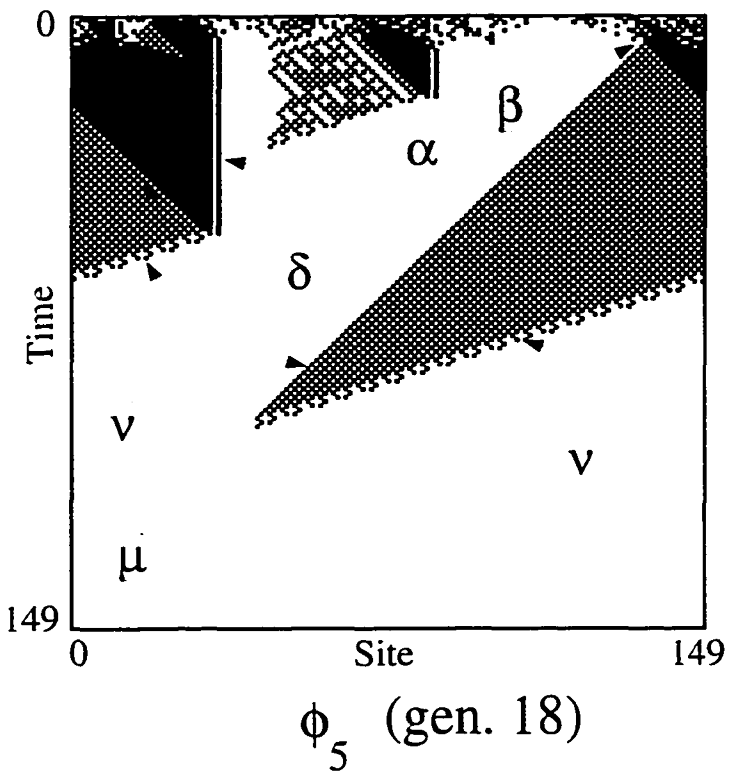


Figure 7.8: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 20, ϕ_5.

the $T_{1/2}$ task. An example of the space-time behavior displayed by ϕ_{GKL} is shown in Figure 7.9(a). Analysis shows that the three regular domains which govern the the behavior of ϕ_{GKL} have a one-to-one correspondence with the domains in ϕ_A (i.e., Λ^0_A, Λ^1_A, and Λ^2_A). As a result, the transducer for ϕ_A, T_A, can be used to study the particle dynamics in ϕ_{GKL}. Figure 7.9(b) depicts the space-time pattern in Figure 7.9(a) after filtering the latter with T_A. Particle-level study shows that the particles and particle interactions in ϕ_{GKL} are indeed very similar to those in ϕ_A. However, the look-up-tables ϕ_A and ϕ_{GKL} are not identical to each other and the distance between them in Hamming space is 31. In essence, the "particle logic" that makes ϕ_{GKL} a superior reliable computing medium is very similar to the strategy which allows ϕ_A to achieve high-performance in the $T_{1/2}$ task.

What are the details of this particle logic? The strategy used by ϕ_{GKL} is illustrated in Figure 7.10(a). In the figure, the IC consists of a B island and a W

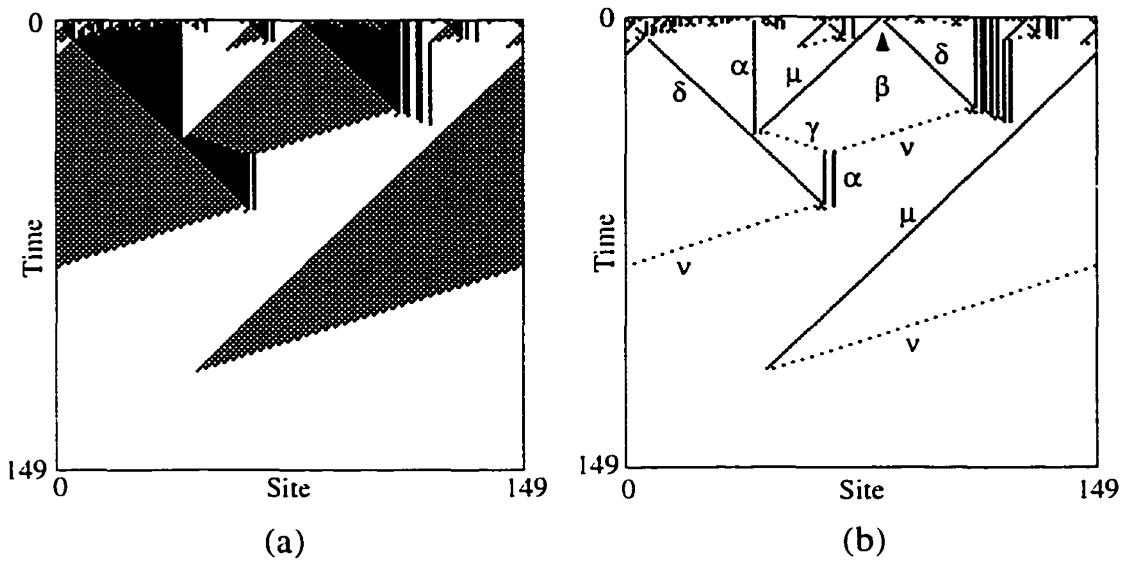


Figure 7.9: (a) Space-time diagram of ϕ_{GKL} starting with a random initial condition with $\rho_0 = 0.4899$. (b) The same space-time diagram after filtering with the transducer T_{GKL} ($= T_A$) that maps all domains to white and all defects to black. Greek letters label particles described in the text.

island set next to each other. Initially, the BW boundary represented by particle α occurs at the left (and the right) boundary of the diagram. The size of the B island is slightly more than that of the W island, and thus the overall density of the IC is slightly more than 1/2. The behavior of ϕ_{GKL} CA starting from this simple IC helps to explicate the basic steps in ϕ_{GKL}'s particle logic.

The first reaction occurs at the WB boundary β. The BW boundary is stable and it remains unchanged. The unstable "particle" β produces two particles μ and δ. The two particles have velocities that are equal in magnitude but opposite in direction. It should be noted that the above behavior results in the origination and subsequent growth of the # domain which replaces the W island and the B island at the same rate of 1 site per time step. Since we have periodic boundary conditions, both particles move towards the α particle at the BW boundary. However, because the W island is smaller than the B island, particle μ is able to interact with α before δ has a chance to do so.

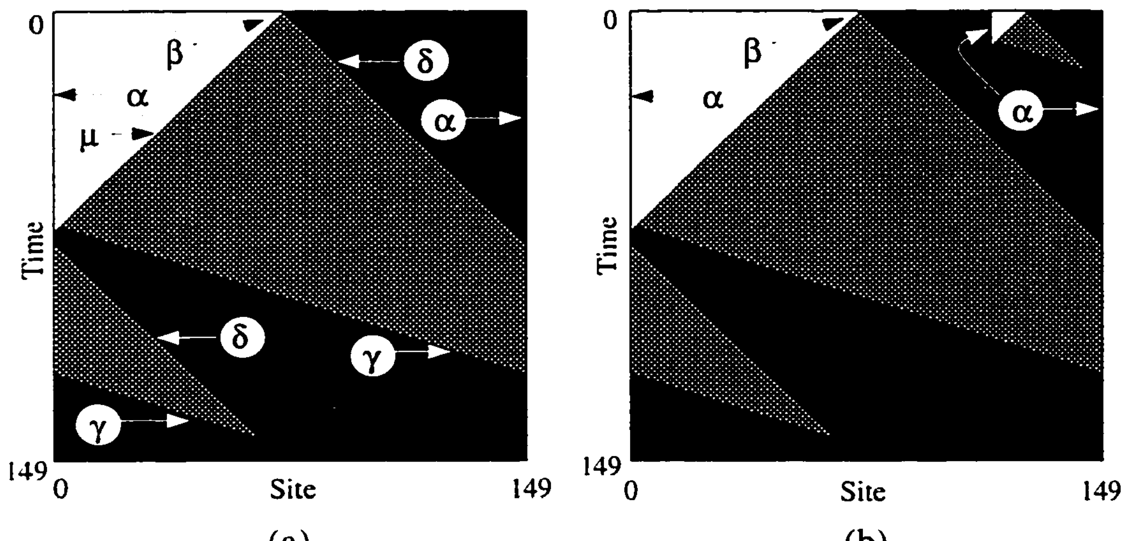


Figure 7.10: (a) Space-time diagram of ϕ_{GKL} starting with a IC consisting of a B island and a W island. $\rho_0 > 1/2$. (b) Space-time diagram of ϕ_{GKL} starting with a IC similar as in (a) with an additional small W island. $\rho_0 < 1/2$.

The particle interaction $\alpha + \mu$ results in particle γ. The δ particle on the other hand is unaffected and it continues in its previous trajectory. At this juncture, the W island has been entirely removed from the lattice, and the configuration consists of a small B island surrounded by the $\#$ domain. Due to the velocities of the particles γ and δ, the B island now starts to grow at the rate of 2 sites per time step. In a short time, the B island takes over the entire lattice, and the correct fixed point configuration is reached.

In more complicated ICs, the same sequence of particle interactions occur at different regions in the lattice and result in a correct final configuration. However, ϕ_{GKL} often fails to correctly classify ICs. Why is ϕ_{GKL} not a perfect rule for the density classification task? And under what conditions does ϕ_{GKL} fail? We use Figure 7.10(b) to answer these questions.

The IC shown in Figure 7.10(b) is identical to the IC in Figure 7.10(a), except for the presence of a second W island. Due to the presence of this new island, the density of the IC is now slightly less than 1/2. Nevertheless, as the figure shows,

ϕ_{GKL} incorrectly classifies the IC by reaching the all-1s configuration as before. In explaining this error, we note that despite the dissimilarities in the two ICs, the resulting spatio-temporal behaviors are remarkably similar. The only significant difference in Figure 7.10(b) is the presence of the extra W island. However, the new W island is removed from the lattice before it has a chance to affect the particles of the two larger islands. Once this happens, the information about the smaller W island is forever lost, and ϕ_{GKL} behaves as if the extra W island never existed. Thus, the CA arrives at the erroneous answer. In more complicated ICs, smaller islands which are hidden inside larger islands are often removed from the lattice before these islands have had a chance to influence the particle logic of the larger islands.

This simple scenario presented in Figure 7.10(b) highlights the fundamental reason that prevents ϕ_{GKL} from achieving 100% performance in the $\mathbf{T}_{1/2}$ task. It also underscores the $\mathbf{T}_{1/2}$ task's main dichotomy. In order to achieve superior performance, a CA must force the density of the configuration to move closer to 1.0 or 0.0. However, by this very act, the CA is forced to loose information about the current configuration. As we have shown above, this results in inferior performance (recall the discussions in Appendix A).

7.3 Analysis of ϕ_B

Figure 7.11(a) gives a space-time diagram for another high-performance CA, ϕ_B, for the $\mathbf{T}_{1/2}$ task. The unbiased performance measure $\mathcal{P}^{149}_{10^4}(\phi_B) = 0.729$. Analysis of Figure 7.11(a) shows that the space-time behavior of ϕ_B is dominated by three types of domains: The domains, denoted here as Λ^0_B, Λ^1_B, and Λ^2_B, consists of the languages 000^+, 11^+, and $(001)^+$ respectively. Since the temporal invariance and spatial homogeneity conditions are satisfied, Λ^0_B, Λ^1_B, and Λ^2_B are indeed regular domains. As in the previous section, Λ^0_B, Λ^1_B, and Λ^2_B will be denoted here as W, B, and # respectively.

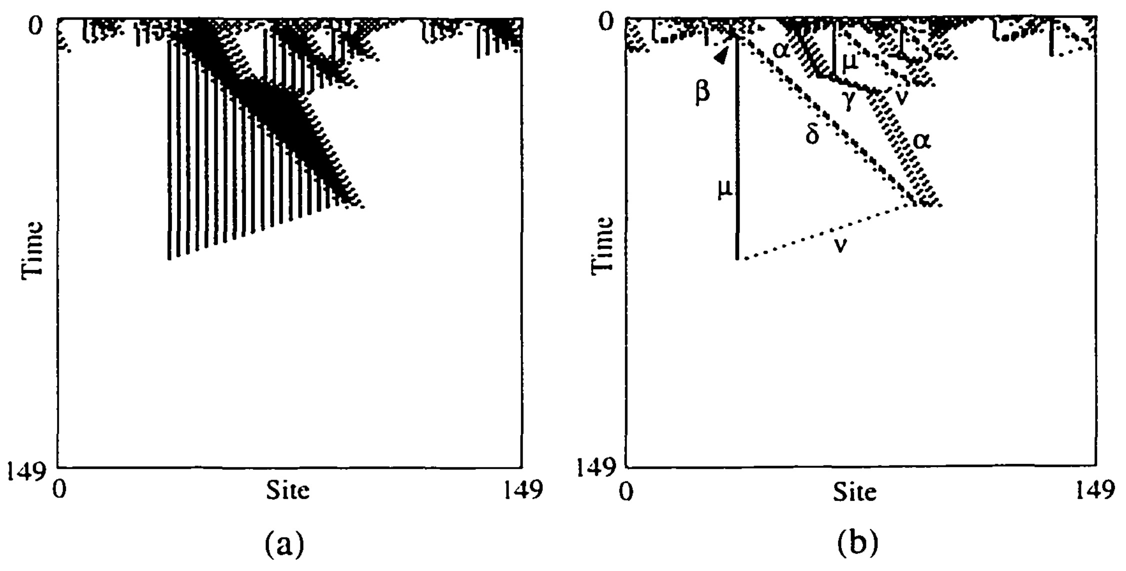


Figure 7.11: (a) Space-time diagram of ϕ_B starting with a random initial condition with $\rho_0 = 0.4966$. (b) The same space-time diagram after filtering with the transducer T_B that maps all domains to white and all defects to black. Greek letters label particles described in the text.

The transducer T_B, which which accepts words in the domain languages Λ^0_B, Λ^1_B, and Λ^2_B, is presented in Figure 7.12. Figure 7.11(b) depicts the space-time plot after T_B has been applied to the space-time diagram in Figure 7.11(a). Table 7.3 lists the domains, particles and particle interactions observed in the filtered space-time plots of ϕ_B.

The most interesting feature to note in Table 7.3 is its striking similarity to Table 7.1 for ϕ_A. The table shows that the interactions between the particles in ϕ_B are identical to those in ϕ_A (although the particle velocities in ϕ_B and ϕ_A are different from each other). As in ϕ_A, the particle β, which exists at WB boundary, spontaneously decays to create the # domain with two boundaries, one on each side: μ on the left and δ on the right (Figure 7.11(b)). Given the similarity in the number of domains, the number of particles, and their interactions, it is possible to claim that the strategy used by ϕ_B to solve the density classification problem is very similar to the strategy used by ϕ_A or ϕ_{GKL}.

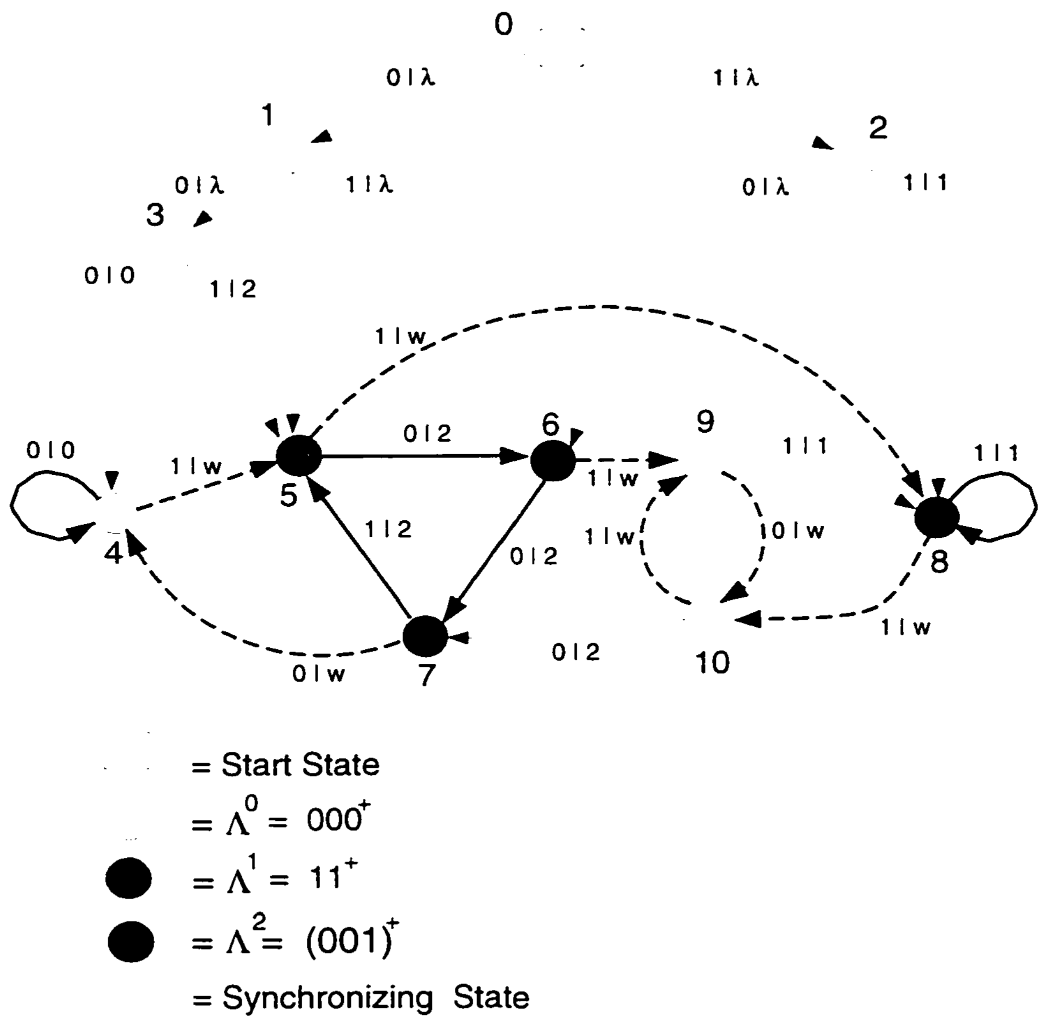


Figure 7.12: The spatial transducer T_B, which accepts words in domain Λ_B^0, Λ_B^1, and Λ_B^2.

But why then is ϕ_B's performance for the $T_{1/2}$ task inferior to the performance of ϕ_A ? The answer to this question lies in the inherent symmetries of the $T_{1/2}$ task, which requires treating W or B domains in the same fashion without any bias. For example, as mentioned in the previous section, W and B islands of the same size should be removed at the same rate to ensure higher performance. Given that the particle interactions are identical in ϕ_A and ϕ_B, a simple way to check for such symmetries in the particle dynamics is to analyze the relationships between the particle velocities. If the WB boundary (β) is unstable, and the BW boundary α has a zero velocity (or vice versa), then symmetry constraints require the following:

Domains		
$W = 000^+$	$B = 11^+$	$\# = (001)^+$

Particles					
Symbol	Domain Boundary	Temporal Periodicity	Spatial Displacement	Velocity	Velocity w.r.t. α
α	BW	2	1	$\frac{1}{2}$	0
β	WB	-	-	-	-
γ	$B\#$	1	3	3	$2\frac{1}{2}$
δ	$\#B$	3	3	1	$\frac{1}{2}$
μ	$W\#$	0	0	0	$-\frac{1}{2}$
ν	$\#W$	1	-3	-3	$-3\frac{1}{2}$

Particle Interactions			
Decay	$\beta \rightarrow \mu + \delta$		
Annihilative	$\mu + \nu \rightarrow \emptyset$	$\gamma + \delta \rightarrow \emptyset$	
Reactive	$\delta + \alpha \rightarrow \nu$	$\alpha + \mu \rightarrow \gamma$	$\gamma + \nu \rightarrow \alpha$

Table 7.3: The domains, particles, and particle interactions that dominate the spatio-temporal behavior of ϕ_B.

(i) The velocity of μ ($W\#$ boundary) and δ ($\#B$ boundary) should be equal in magnitude but opposite in direction.

(ii) The velocity of ν ($\#W$ boundary) and γ ($B\#$ boundary) should be equal in magnitude but opposite in direction.

As shown in Table 7.1, ϕ_A meets these conditions.

In order to check for the above symmetry constraints, Table 7.3 also gives the particle velocities with respect to α. (It should be noted that this imposes only a linear transformation in the CA's space-time behavior and does not fundamentally affect the particle dynamics.) As shown in the "velocity w.r.t. α" column, the magnitude of the particle velocities in ν and γ are not equal: this is the fundamental reason behind ϕ_B's inferior performance when compared to ϕ_A.

7.4 Analysis of ϕ_C

The space-time diagram from another high-performance rule discovered by the GA, ϕ_C, is shown in Figure 7.13(a). The unbiased performance measure $\mathcal{P}^{149}_{10^4}(\phi_C)$ for the $\mathrm{T}_{1/2}$ task equals 0.744. Analysis of the figure shows that the space-time

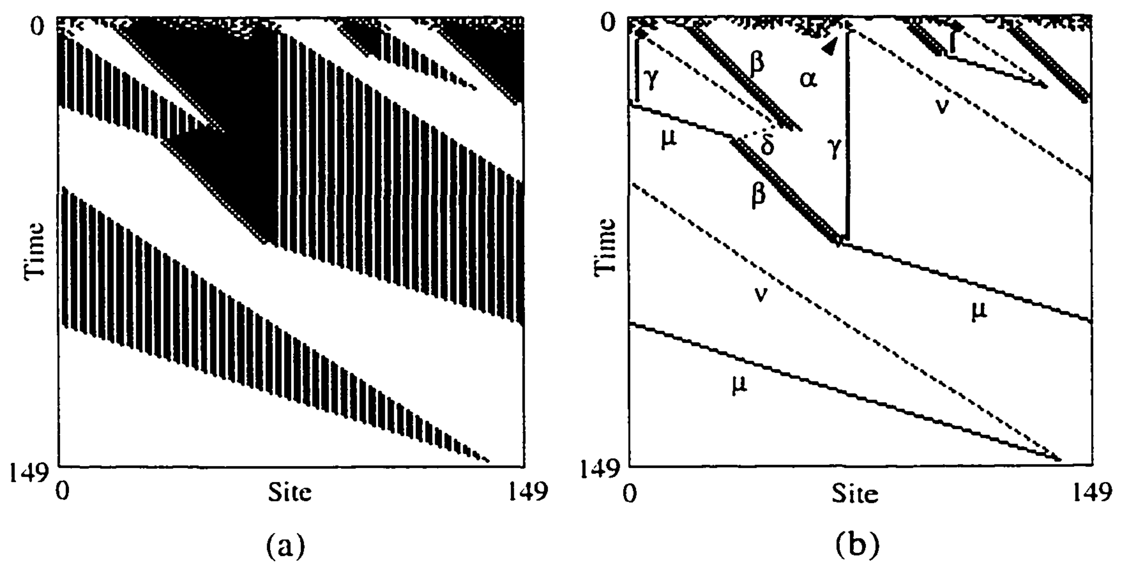


Figure 7.13: (a) Space-time diagram of ϕ_C starting with a random initial condition with $\rho_0 = 0.4362$. (b) The same space-time diagram after filtering with the transducer T_C that maps all domains to white and all defects to black. Greek letters label particles described in the text.

behavior of ϕ_C consist of three types of domains. The domains, denoted here as Λ^0_C, Λ^1_C, and Λ^2_C, consists of the languages 00^+, 111^+, and $(110)^+$ respectively. All three domains Λ^0_C, Λ^1_A, and Λ^2_C are regular domains since each satisfies the temporal invariance and spatial homogeneity conditions. As before, Λ^0_C, Λ^1_C, and Λ^2_C will be denoted here as the W, B, and # domains respectively.

Figure 7.14 shows the spatial transducer T_C, which accepts words in domain languages Λ^0_C, Λ^1_C, and Λ^2_C. It should be noted that T_C is the mirror image of the transducer T_B given in the previous section. Essentially, each input symbol has been flipped (i.e., 0 to 1, and 1 to 0) in T_B to create T_C. This is because the languages in the three domains in ϕ_B and in ϕ_C are Boolean complements of each other.

Figure 7.13(b) depicts the space-time plot after T_C has been applied to the space-time diagram in Figure 7.13(a). The domains, particles and particle interactions observed in the filtered space-time plots of ϕ_C are presented in Table 7.4.

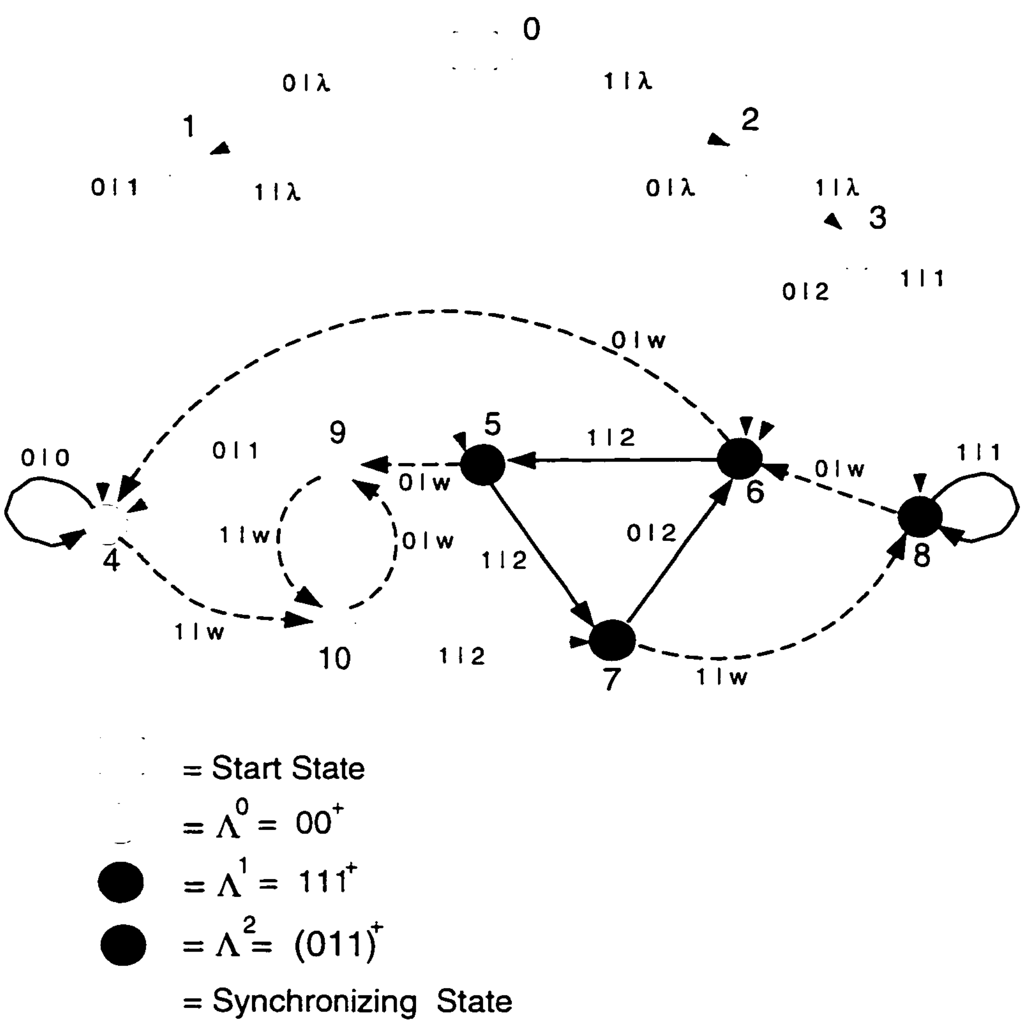


Figure 7.14: The spatial transducer T_C which accepts words in domain Λ_C^0, Λ_C^1, and Λ_C^2

Is the strategy used by ϕ_C similar to that in ϕ_A (or ϕ_B) ? As shown in Table 7.4, the number of domains and the number of particles in ϕ_C are identical to those in ϕ_A or ϕ_B. Nevertheless, one seemingly important difference in ϕ_C is that particle α is unstable, while β is stable. In addition, the set of particle interactions apparently bears little resemblance to the particle interactions in ϕ_A. However, a closer inspection reveals that the essential particle-based mechanisms used by ϕ_C for the $\mathbf{T}_{1/2}$ task are very similar to the strategies discussed in the earlier sections in this chapter. Indeed, the particle interactions listed in Table 7.4 can be mapped

Domains		
$W = 00^+$	$B = 111^+$	$\# = (110)^+$

Particles					
Symbol	Domain Boundary	Temporal Periodicity	Spatial Displacement	Velocity	Velocity w.r.t. β
α	BW	-	-	-	-
β	WB	1	1	1	0
γ	$B\#$	1	0	0	-1
δ	$\#B$	1	-3	-3	-4
μ	$W\#$	1	3	3	2
ν	$\#W$	2	4	2	1

Particle Interactions			
Decay	$\alpha \rightarrow \gamma + \nu$		
Annihilative	$\mu + \nu \rightarrow \emptyset$	$\gamma + \delta \rightarrow \emptyset$	
Reactive	$\beta + \gamma \rightarrow \mu$	$\nu + \beta \rightarrow \delta$	$\mu + \delta \rightarrow \beta$

Table 7.4: The domains, particles, and particle interactions that dominate the spatio-temporal behavior of ϕ_C.

to the particle interactions in ϕ_A by simply exchanging the particle pairs α and β, γ and δ, and μ and ν.

Table 7.4 also helps in explaining why ϕ_C's performance is inferior to that of ϕ_A. As shown in the table, particles μ and δ have velocities with different magnitudes, which in turn inhibits ϕ_C's performance.

This chapter has detailed the use of the computational mechanics framework to understand the spatio-temporal behavior in high-performance CAs for the density classification task. The above analysis has shown that although the high-performance CAs typically use different look-up tables, and although the patterns dominating the behavior of the CAs can be very different, nevertheless, the underlying strategies engendered by the particle dynamics can be similar. Moreover, the GA, while explicitly operating on the look-up tables, can be thought of as implicitly modulating the particle dynamics to evolve high performance CAs for the $T_{1/2}$ task.

Chapter 8

SYNCHRONIZATION TASK

As in the previous chapter, the main aim here is to use the framework of computational mechanics to understand the spatio-temporal behavior of CAs exhibiting high-performance for the synchronization task **R**. The major unifying theme among the high-performance CAs for the synchronization task was the presence of visually distinct patterns in the space-time diagrams. The aim here is to delineate the pattern dynamics in these CAs in terms of the embedded domains, particles and particle interactions, and to show how they give rise to high performance in the CAs. Once the computational structures have been identified, the second goal in this chapter is to describe the temporal stages in the evolutionary process which lead to the discovery of a high-performance CA. Finally the third goal is to determine the common themes underlying the particle-level strategies used by the high-fitness CAs to perform the synchronization task.

In this chapter, three representative rules with different look-up tables have been selected from the set of rules with high performance for the **R** task. These rules are then analyzed in detail. The three rules are:

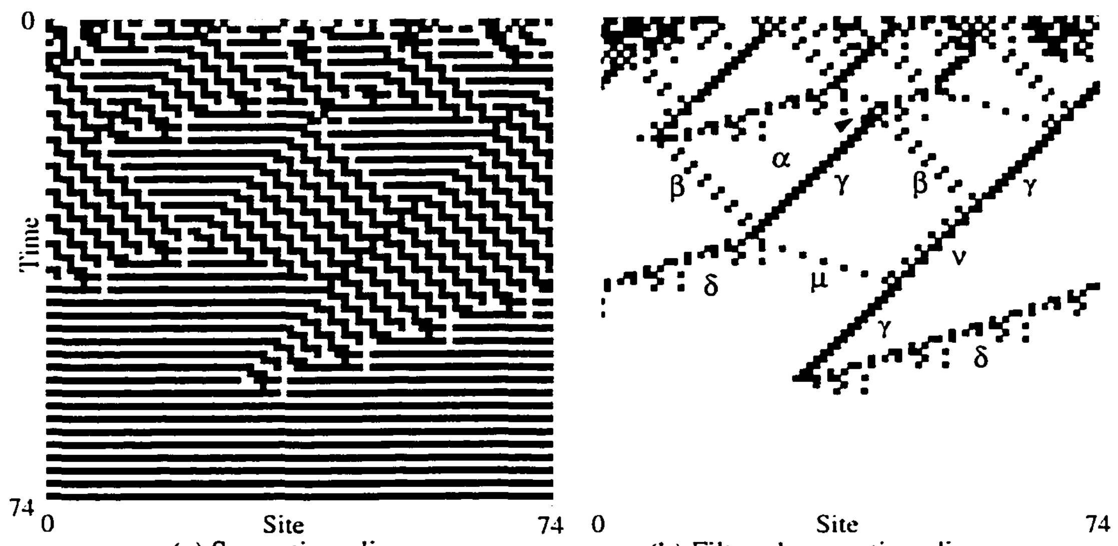
(i) ϕ_A = 0x FEB1C6EA B8E0C4DA 6484A5AA F410C8A0,

(ii) ϕ_B = 0x FE5CEEF4 EDD2E0AA 60F1EE40 FCB6B280, and

(iii) ϕ_C = 0x BFCD3CC2 A8EFFE8C D9C8FEAE 3CEEFA60.

While the analysis presented here focuses on these three rules, the main conclusions drawn from the analysis are general in their scope: in most cases the analysis can be easily extended to other high-performance rules without much difficulty or modification.

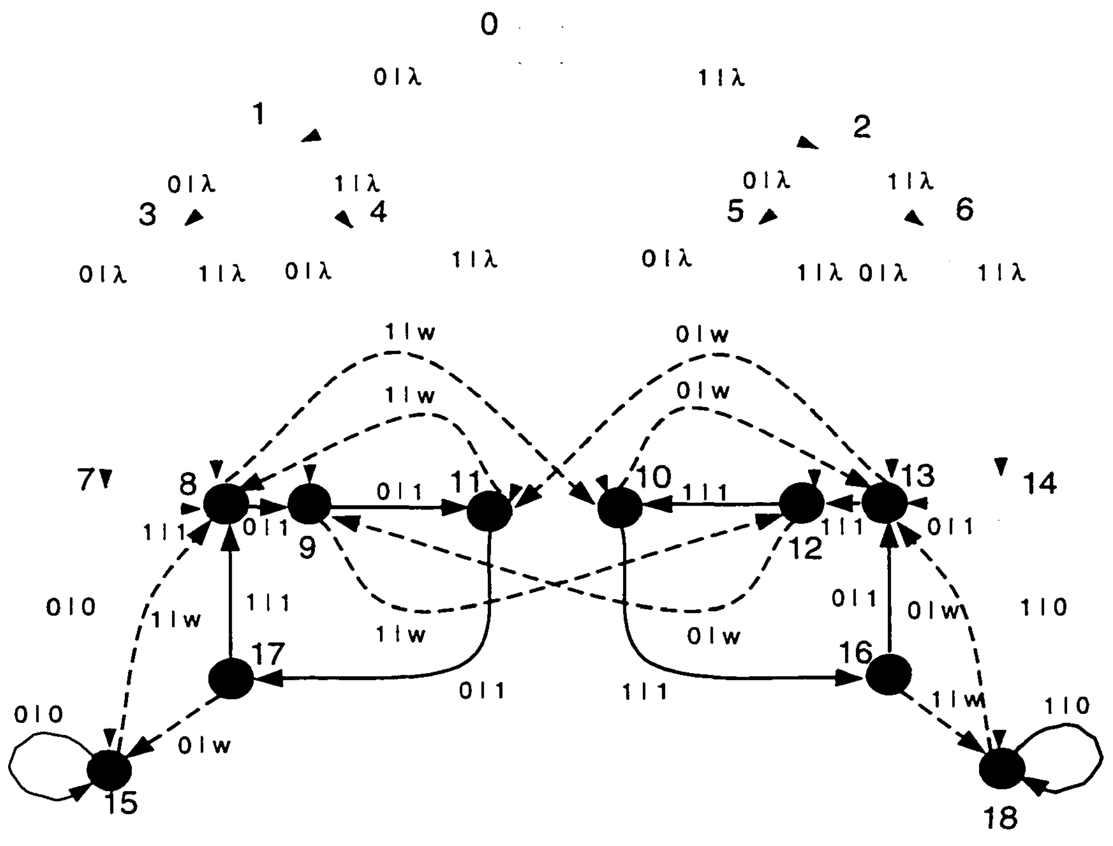
8.1 Analysis of ϕ_A



(a) Space-time diagram. (b) Filtered space-time diagram.

Figure 8.1: (a) Space-time diagram of ϕ_A starting with a random initial condition. (b) The same space-time diagram after filtering with the transducer T_A which maps all domains to white and all defects to black. Greek letters label particles described in the text.

Figure 8.1(a) gives a space-time diagram for one of the GA-discovered CAs with 100% performance, here called ϕ_A. In the example given in the figure, global synchronization occurs at time step 58. The figure also shows that the space-time behavior of ϕ_A is dominated by two types of distinct patterns: regions where the all-1s pattern alternates with the all-0s pattern; and regions of jagged black diagonal lines alternating with jagged white diagonal lines. Both these patterns can be characterized as regular domains since each satisfies the temporal invariance and spatial homogeneity conditions. One of the domains, denoted here as Λ^0_A, consists of two languages, 0000^+ and 1111^+, such that $\Phi(0000^+) = 1111^+$ and $\Phi(1111^+) = 0000^+$. Λ^0_A is the synchronous domain, characterizing regions in the configurations where the cells are oscillating in synchrony. Λ^0_A will also be denoted as S. For the CA to reach global synchrony, the entire lattice has to be occupied by the S domain. The second domain, denoted here as Λ_A^1, also has a temporal periodicity of two, but has a spatial periodicity of four. Λ_A^1 consists of two languages $(0001)^+$ and $(1110)^+$, such that $\Phi((0001)^+) = (1110)^+$ and $\Phi((1110)^+) = (0001)^+$. In subsequent discussions, Λ_A^1 will also be denoted as D. Unlike the languages in Λ_A^1, the configurations in Λ_A^1 have a temporal periodicity of four.



. = Start State
● = Λ^0 = 0000^+, 1111^+
● = Λ^1 = $(0001)^+$, $(1110)^+$
= Synchronizing State

Figure 8.2: The spatial transducer T_A which accepts words in domains Λ_A^0 and Λ_A^1.

Figure 8.2 shows a spatial transducer T_A, which accepts words in domain Λ_A^0 and Λ_A^1. When T_A is applied to the space-time diagram in Figure 8.1(a), it produces the plot shown in Figure 8.1(b). In the figure, the filtering not only determines the location of the particles in the space-time diagram, but it also helps in readily identifying the spatial and temporal features of the particles. In addition, various types of particle interactions can be distinguished. The domains, particles and particle interactions observed in the filtered space-time plots of ϕ_A are presented in Table 8.1.

Domains	
S alternates between 0000^+ and 1111^+, i.e. $0000^+ = \Phi(1111^+)$, $1111^+ = \Phi(0000^+)$	D alternates between $(0001)^+$ and $(1110)^+$, i.e. $(0001)^+ = \Phi((1110)^+)$, $(1110)^+ = \Phi((0001)^+)$

Particles				
Symbol	Domain Boundary	Temporal Periodicity	Spatial Displacement	Velocity
α	$S\overline{S}$	-	-	0
β	DS	2	2	1
γ	SD	2	-2	-1
δ	DS	4	-12	-3
μ	SD	2	6	3
ν	$D\overline{D}$	2	-2	-1

Particle Interactions			
Decay	$\alpha \to \gamma + \beta$		
Annihilative	$\gamma + \delta \to \emptyset$	$\mu + \beta \to \emptyset$	
Reactive	$\beta + \gamma \to \nu$, if $d \bmod 4 = 1$	$\nu + \delta \to \beta$	$\mu + \nu \to \gamma$
Reversible	$\beta + \gamma \to \delta + \mu$, if $d \bmod 4 \neq 1$	$\mu + \delta \to \gamma + \beta$	

Table 8.1: The domains, particles, and particle interactions that dominate the spatio-temporal behavior of ϕ_A. $S\overline{S}$ and $D\overline{D}$ represent phase defects within the S and D domains respectively. The reactions between particles β and γ depend on the relative distance d between them, where $0 \leq d \leq 2r$.

How does ϕ_A perform the synchronization task? From any random initial configuration, ϕ_A produces local regions of synchronization, i.e., regions which are in the S domain. However, in many cases, adjacent S domains are out-of-phase with respect to each other. Wherever such phase defects occur, ϕ_A resolves them by propagating particles—the boundaries between the S domains and the D domain—in opposite directions. Encoded in ϕ_A's look-up table are interactions involving these particles that allow one or the other competing synchronized region to annihilate the other and to itself expand. Similar sets of interactions continue to take place among the remaining synchronized regions until the entire configuration has one coherent phase.

In the next section this intuitive description is made more rigorous. In particular, the computational mechanics framework is used to describe the evolutionary path by which the GA discovered ϕ_A.

8.2 The Evolution to Synchronization

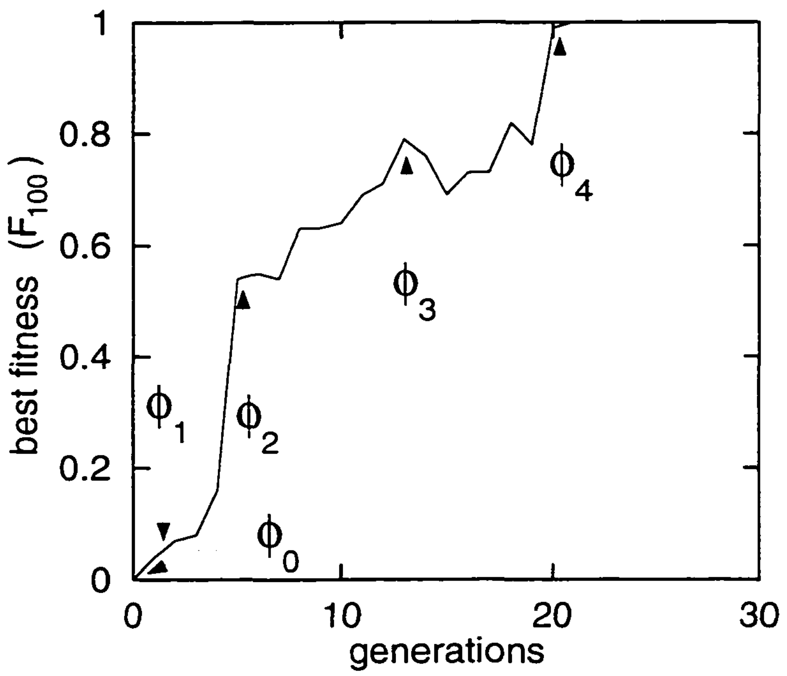


Figure 8.3: Evolutionary history of ϕ_A: F_{100} versus generation for the fittest CA in each population. The arrows indicate the generations in which the GA discovered each new significantly improved strategy. In Figures 8.4 through 8.8, the space-time diagrams illustrating the behavior of the best ϕ at each of the five generations marked in this plot are shown.

Figure 8.3 plots the best fitness in the population versus generation for the first 30 generations of the run in which ϕ_A was evolved. The figure shows that, over time, the best fitness in the population is marked by periods of sharp increases. Qualitatively, the overall increase in fitness can be divided into five epochs. The first epoch starts at generation 0 and each of the following epochs corresponds to the discovery of a new, significantly improved strategy for performing the synchronization task. Similar epochs were seen in most of the runs resulting in CAs with 100% performance. In Figure 8.3, the beginning of each epoch is labeled with the fittest CA in the population at that generation.

Epoch 0: Growth of Disordered Regions.

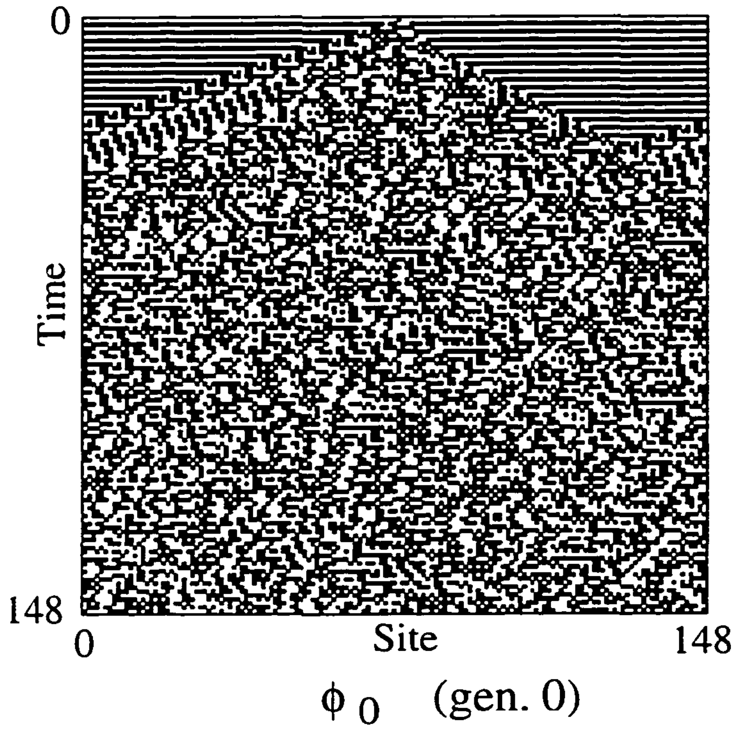


Figure 8.4: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of a rule in generation 0, ϕ_0. The IC consists of a single 1 in the center of a field of 0s. Note that the disordered region grows until it occupies the whole lattice.

To perform the synchronization task, a CA ϕ must have $\phi(0^7) = 1$ and $\phi(1^7) = 0$. These mappings insure that local regions will have the desired oscillation and will be in domain S with $0^+ = \Phi(1^+)$, and $1^+ = \Phi(0^+)$. Since the existence of the S domain is guaranteed by fixing just two bits in the chromosome, approximately 1/4 of the CAs in a random initial population have S.

However, S's stability under small perturbations depends on other output bits. For example, ϕ_1 is a generation 0 CA with these two bits set correctly, but under ϕ_1

a small perturbation in S leads to the creation of a disordered region. This is shown in Figure 8.4, where the IC contains a single 1 at the center site. In the figure, the disordered region grows until it occupies the whole lattice. This behavior is typical of CAs in generation 0 that have the two end bits set correctly. Increasing the number of perturbation sites in S leads to a simultaneous creation of disordered regions all over the lattice, which subsequently merge to eliminate synchronous regions. Thus, CAs like ϕ_0 have zero fitness unless there is at least one test IC with all-0s (or all-1s) configuration.

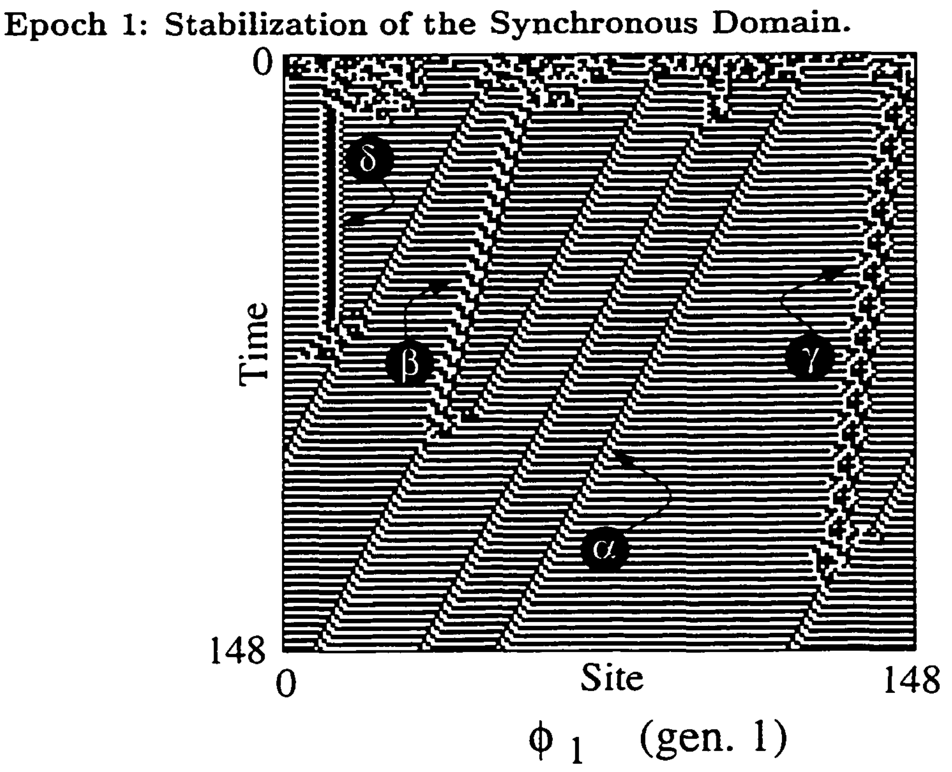


Figure 8.5: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 1, ϕ_1. Randomly generated ICs have been used in Figures 8.5 through 8.8. Note that unlike ϕ_0, rule ϕ_1 has succeeded in stabilizing the S domain.

The best CA at generation 1, ϕ_1, has $F_{100} \approx 0.04$, indicating that it successfully synchronizes on only a small fraction of the ICs. Although this is only a small increase in fitness, the space-time behavior of ϕ_1 (Figure 8.5) is very different from that of ϕ_0. Unlike ϕ_0, ϕ_1 eliminates disordered regions by expanding (and thereby stabilizing) local synchronous domains. The stability of the synchronous domain is due to the fact that ϕ_1 maps all the eight neighborhoods with six or more 0s to 1, and seven out of eight neighborhoods with six or more 1s to 0. Under the lexicographic ordering, most of these bits are clustered at the left and right ends of the chromosome. This means it is easy for the crossover operator to bring them together from two separate CAs to create CAs like ϕ_1.

Figure 8.5 shows that under ϕ_1, the synchronous regions fail to occupy the entire lattice. A significant number of constant-velocity particles (here, boundaries between adjacent S domains) persist indefinitely and prevent global synchronization from being reached. Due to the temporal periodicity of the S domain, the two adjacent S domains at any boundary can be either in-phase or out-of-phase with respect to each other. The in-phase and the out-of-phase defects between two S domains are denoted as SS and $S\overline{S}$ respectively. A more detailed analysis of ϕ_1's space-time behavior shows that it supports one type of stable $S\overline{S}$ particle, α, and three different types of stable SS particle: β, γ, and δ, each with a different velocity. Examples of these particles are labeled in Figure 8.5, and their properties and interactions are summarized in Table 8.2. (For notational convenience, the same set of Greek letters is used to represent different particles in different rules.)

For most ICs, application of ϕ_1 quickly results in the appearance of these particles, which then go on to interact, assuming they have distinct velocities. A survey of their interactions indicates that the α particle dominates: it persists after collision with any of the SS particles. Interactions among the three SS particles do take place, resulting in either a single β or a pair of α's. Thus, none of the interactions are annihilative: particles are produced in all interactions. As a result, once a set of particles comes into existence in the space-time diagram, it is guaranteed that at least one particle persists in the final configuration. For almost all values of initial ρ, ϕ_1's formation of persistent particles ultimately prevents it from attaining global synchrony. Only when the initial ρ is very close to 0.0 or 1.0 does ϕ_1 reach the correct final configuration. This accounts for its very low fitness.

Cellular Automata		Particles and Interactions			
Chromosome	Generation $(\mathcal{P}^{149}_{10^4}, \mathcal{P}^{599}_{10^4}, \mathcal{P}^{999}_{10^4})$	Label	Domain Boundary	Temporal Periodicity	Velocity
$\phi_1 =$	1	α	$S\overline{S}$	2	-1/2
F8A19CE6		β	SS	4	-1/4
B65848EA	(0.00.	γ	SS	8	-1/8
D26CB24A	0.00.	δ	SS	2	0
EB51C4A0	0.00)	$\beta + \alpha \to \alpha, \gamma + \alpha \to \alpha, \delta + \alpha \to \alpha$			
$\phi_2 =$	5	α	$S\overline{S}$	2	-1/2
F8A1AE2F		β	$S\overline{S}$	6	0
CF6BC1E2	(0.33.	$\beta + \alpha \to \emptyset$			
D26CB24C	0.07.				
3C266E20	0.03)				
$\phi_3 =$	13	α	$S\overline{S}$	4	-3 /4
F8A1AE2F		β	$S\overline{S}$	6	0
CE6BC1E2	(0.57,	γ	$S\overline{S}$	12	1/4
C26CB24E	0.33,	δ	$S\overline{S}$	2	1/2
3C226CA0	0.27)	$\beta + \alpha \to \emptyset, \gamma + \alpha \to \emptyset, \delta + \alpha \to \emptyset$			

Table 8.2: Ancestors of ϕ_A: Particles and their dynamics for the best CAs in early generations of the run that found ϕ_A. The table shows only the common particles and common two-particle interactions that play a significant role in determining fitness.

Epoch 2: Suppression of In-Phase Defects.

Following the discovery of ϕ_1, the next sharp increase in fitness is observed in generation 5, when the best CA in the population, ϕ_2, has $F_{100} \approx 0.54$. The rise in fitness can be attributed to ϕ_2's ability to suppress in-phase (SS) defects for ICs with very low or very high ρ.

In addition to the suppression of SS boundaries, the space-time behavior of ϕ_2 is dominated by two new and different $S\overline{S}$ particles, labeled α and β (Table 8.2; examples are labeled in Figure 8.6). Moreover, because α and β annihilate each other, ϕ_2 is able to reach synchronous configurations even on some ICs with intermediate ρ. However, since the velocity difference between α and β is only 1/2, the two particles might fail to annihilate each other before the maximum of M time steps have elapsed.

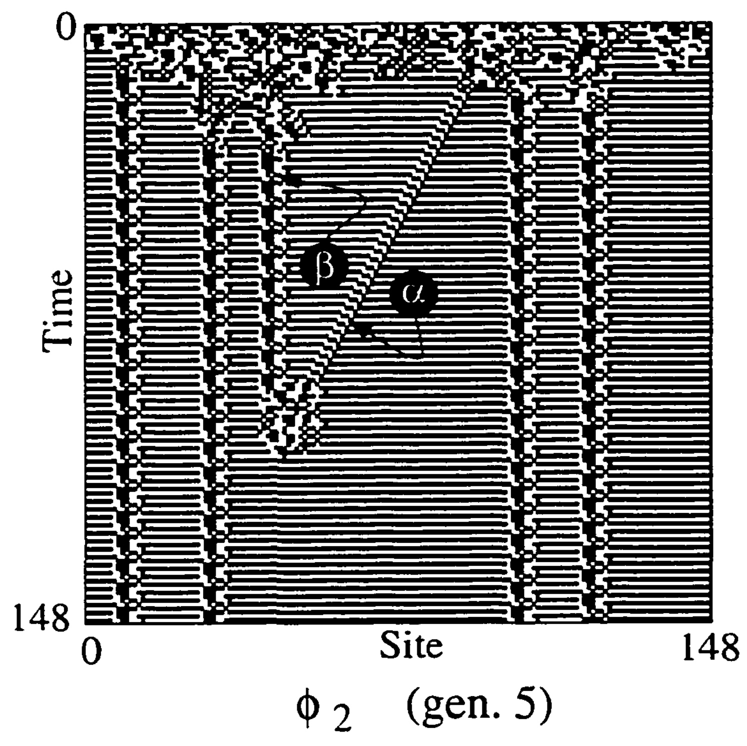


Figure 8.6: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 5, ϕ_2. Note that SS particles do not occur under ϕ_2.

In spite of these improvements, ϕ_2 still fails on a large fraction of its fitness tests. Often the same type of particle occurs more than once in the configuration. In the absence of particles of a different type, they persist, since particles of the same type have the same velocity. Global synchrony is achieved (possibly in more than M time steps) only when the number of α particles and β particles in any configuration are equal. Experiments on ϕ_2 show that the probability of occurrence of β is about twice that of α, so their numbers are often unequal.

From the standpoint of the genetic operators acting on the CA rules, a small change in the relevant entries in ϕ is sufficient to significantly modify the properties of the domain boundaries. As a result, it is the mutation operator that seems to play the primary role in this and subsequent epochs in discovering high-performance CAs.

Epoch 3: Refinement of Particle Velocities.

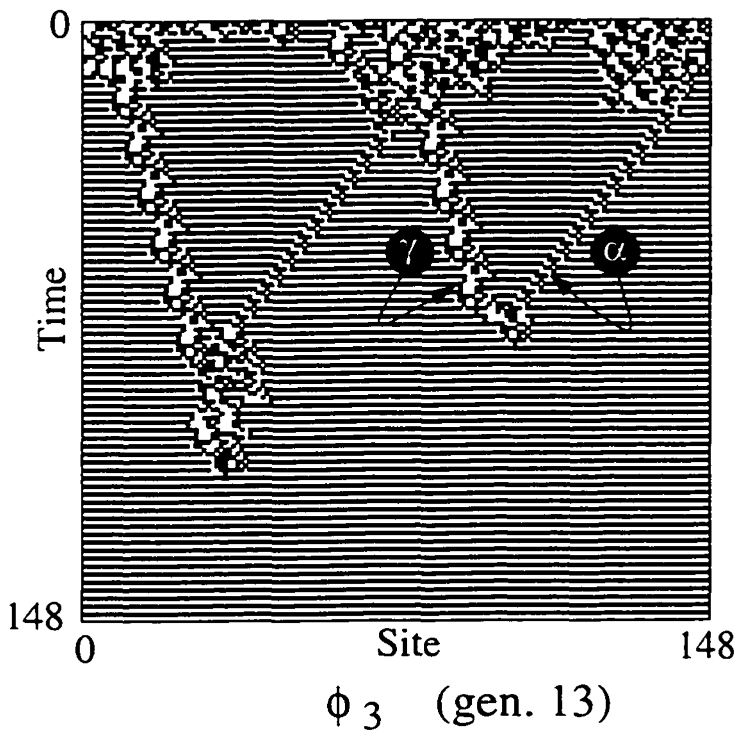


Figure 8.7: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 13, ϕ_3. As a result of the refinement of the $S\overline{S}$ particle velocities, ϕ_3 has attained higher fitness.

A much improved CA, ϕ_3, is found in generation 13 with a fitness of 0.79. Its typical behavior is illustrated in Figure 8.7. ϕ_3 differs from ϕ_2 in two respects, both of which result in improved performance. First, as noted in Table 8.2, the velocity difference between α and γ, the two most commonly occurring particles produced by ϕ_3, is larger (1 as compared to 1/2 in ϕ_2), so their annihilative interaction typically occurs more quickly. This means ϕ_3 has a better chance of reaching a synchronized state within M time steps. Second, the probabilities of occurrence of α and γ are almost equal, meaning that there is a greater likelihood they will pairwise annihilate, leaving only a single synchronized domain.

In spite of these improvements, it is easy to determine that ϕ_3's strategy will ultimately fail to synchronize on a significant fraction of ICs. As long as $S\overline{S}$ particles exist in the space-time diagram, there is a nonzero probability that a pair of $S\overline{S}$ defect sites would be occupied by a pair of identical particles moving in parallel.

In the absence of other particles in the lattice, such a particle pair could exist indefinitely, preventing global synchrony. Thus a completely new strategy is required to overcome persistent parallel-traveling particles.

Epoch 4: The Final Innovation.

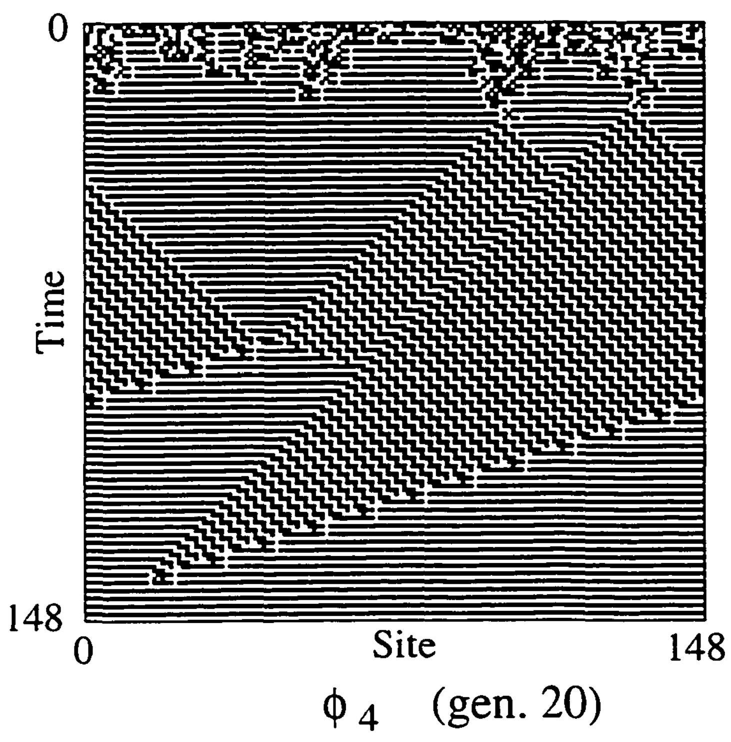


Figure 8.8: Evolutionary history of ϕ_A: A space-time diagram illustrating the behavior of the best rule in generation 20, ϕ_4. The main innovation here is the creation of the D domain.

Fitness increases between generations 13 and 19 reflect only refinement of the basic strategy used in ϕ_3; nothing fundamentally new is discovered. However, in the 20th generation a final dramatic increase in fitness is observed when ϕ_4 is discovered. ϕ_4 has $F_{100} \approx 0.99$ and displays significantly different space-time behavior (Figure 8.8). Following the discovery of ϕ_4 and until the end of the run in generation 100, the best CAs in each generation have $F_{100} = 1.00$. Also, no significant variation in the space-time behavior is noticeable among the best CAs in this run. In particular, ϕ_4's strategy is very similar to that of ϕ_A, a perfected version of ϕ_4

that appeared in the last generation. Here the earlier intuitive description of ϕ_A's strategy is made more rigorous.

As can be seen in Figure 8.1(a), after the first few time steps the space-time behavior of ϕ_A is dominated by two distinct types of domains, S and D, described in the previous section.

Using a transducer that recognizes the S and D regular languages, Figure 8.1(a) can be filtered to display the propagation of the particles embedded in the space-time behavior of ϕ_A (Figure 8.1(b)). As shown in Table 8.1, ϕ_A supports five stable particles, and one unstable "particle", α, which occurs at $S\overline{S}$ boundaries. α "lives" for only one time step, after which it decays into two other particles, γ and β, respectively occurring at SD and $D\overline{S}$ boundaries. β moves to the right with velocity 1, while γ moves to the left at the same speed.

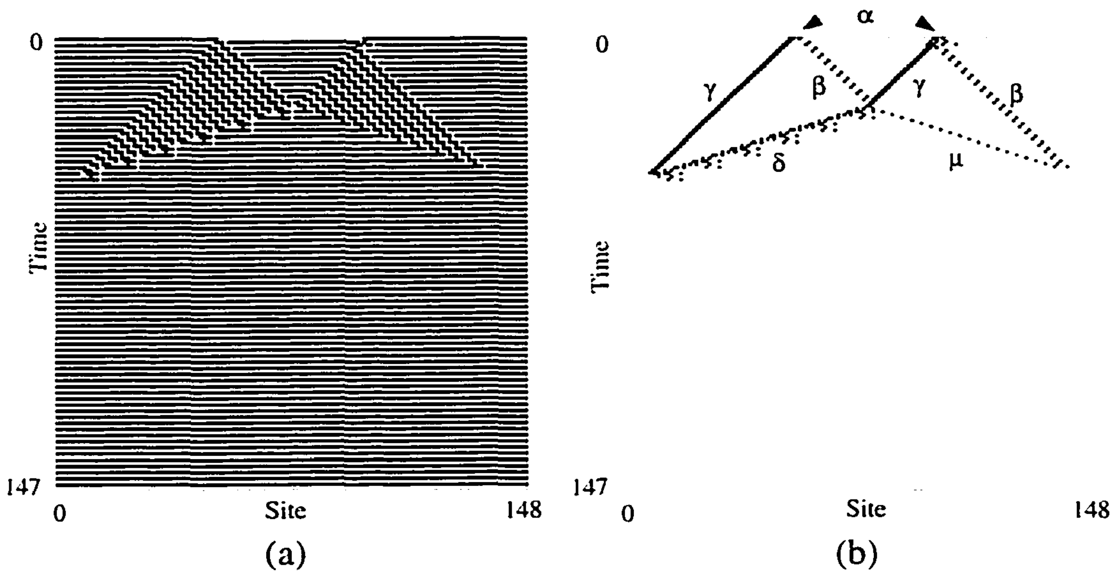


Figure 8.9: (a) Space-time diagram of ϕ_A starting with two S domains which are out of phase with each other. ϕ_A's strategy can be described as a competition between the two S domains according to their relative size, with the D domain acting as the mediator. (b) The same space-time diagram in (a) after filtering with the transducer T_A which maps all domains to white and all defects to black.

The following simple scenario illustrates the role of the unstable particle α in ϕ_A's synchronization strategy. Let ϕ_A start from a simple IC consisting of a pair of

$S\overline{S}$ domain boundaries which are at a small distance from one another. An example of such an initial condition is shown in Figure 8.9(a). Each $S\overline{S}$ domain boundary forms the particle α, which exists for only one time step and then decays into a β-γ pair, with β and γ traveling at equal and opposite velocities. In this example, two such pairs are formed, and the first interaction is between the two interior particles: the β from the left pair and the γ from the right pair. As a result of this interaction, the two interior particles are replaced by δ and μ, which have velocities of -3 and 3, respectively. Due to their greater speed, the new interior particles can intercept the remaining β and γ particles. Since the pair of interactions $\gamma + \delta \rightarrow \emptyset$ and $\mu + \beta \rightarrow \emptyset$ are annihilative, and because the resulting domain is S, the configuration is now globally synchronized (Figure 8.9(b)). One necessary refinement to this explanation comes from noticing that the β-γ interaction depends on the inter-particle distance d, where $0 \leq d \leq 2r$. If $d \bmod 4 \neq 1$, then the interaction $\beta + \gamma \rightarrow \delta + \mu$ takes place. But if $d \bmod 4 = 1$, then the reaction $\beta + \gamma \rightarrow \nu$ occurs. The particle ν is essentially a defect in the D domain, which regenerates β and γ when it reacts with any other particle.

The fundamental innovation of ϕ_A over ϕ_3 is the formation of the D domain, which allows two globally out-of-phase S domains to compete according to their relative size and so allows for the resolution of global phase frustration.

The particle interactions in the filtered space-time diagram in Figure 8.1(b) (starting from a random IC) are somewhat more complicated than in this simple example, but it is still possible to identify essentially the same set of particle interactions ($\beta + \gamma \rightarrow \delta + \mu$, $\beta + \gamma \rightarrow \nu$, and $\gamma + \delta \rightarrow \emptyset$) that effect the global synchronization in the CA.

8.3 Analysis of ϕ_B

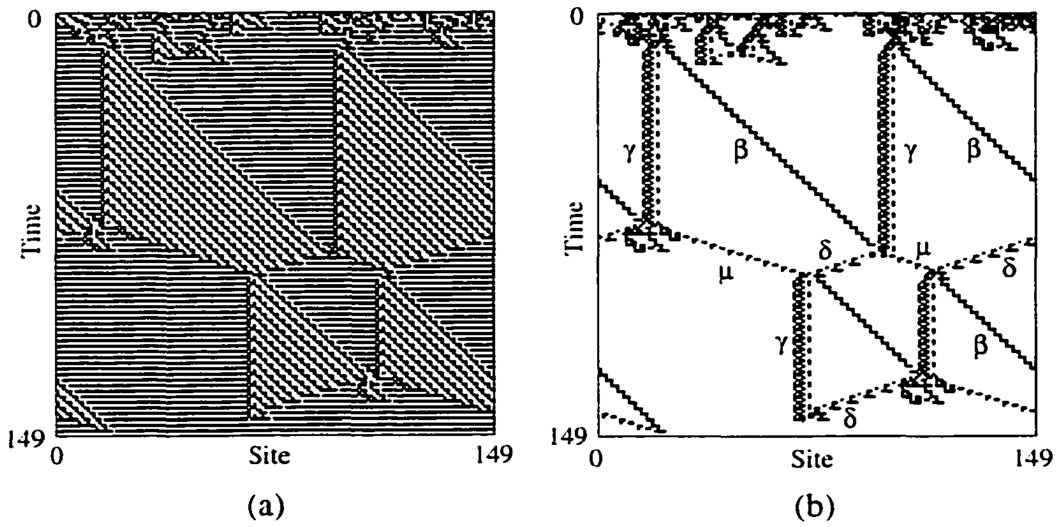


Figure 8.10: (a) Space-time diagram of ϕ_B starting with a random initial condition. (b) The same space-time diagram after filtering with the transducer T_B which maps all domains to white and all defects to black. Greek letters label particles described in the text.

Figure 8.10(a) gives a space-time diagram for another GA-discovered CA with 100% performance, here called ϕ_B. Analysis of Figure 8.10(a) shows that two different regular domains dominate the space-time behavior of ϕ_B. One of the domains, denoted here as Λ^0_B, is the familiar synchronous domain S described in the previous section. The second domain, denoted here as Λ^1_B, has a temporal periodicity of two and a spatial periodicity of four. Λ^1_A consists of two languages, $(0001)^+$ and $(0011)^+$, such that $\Phi((0001)^+) = (0011)^+$ and $\Phi((0011)^+) = (0001)^+$. The configurations in Λ^1_B however, have a spatial and temporal periodicity of four. Both Λ^0_B and Λ^1_B are regular domains since each satisfies the temporal invariance and spatial homogeneity conditions. As in the previous section, Λ^0_B and Λ^1_B will also be represented as S and D respectively.

The transducer T_B, which accepts words in domains Λ^0_B and Λ^1_B, is presented in Figure 8.11. After T_B has been applied to the space-time diagram in Figure 8.10(a),

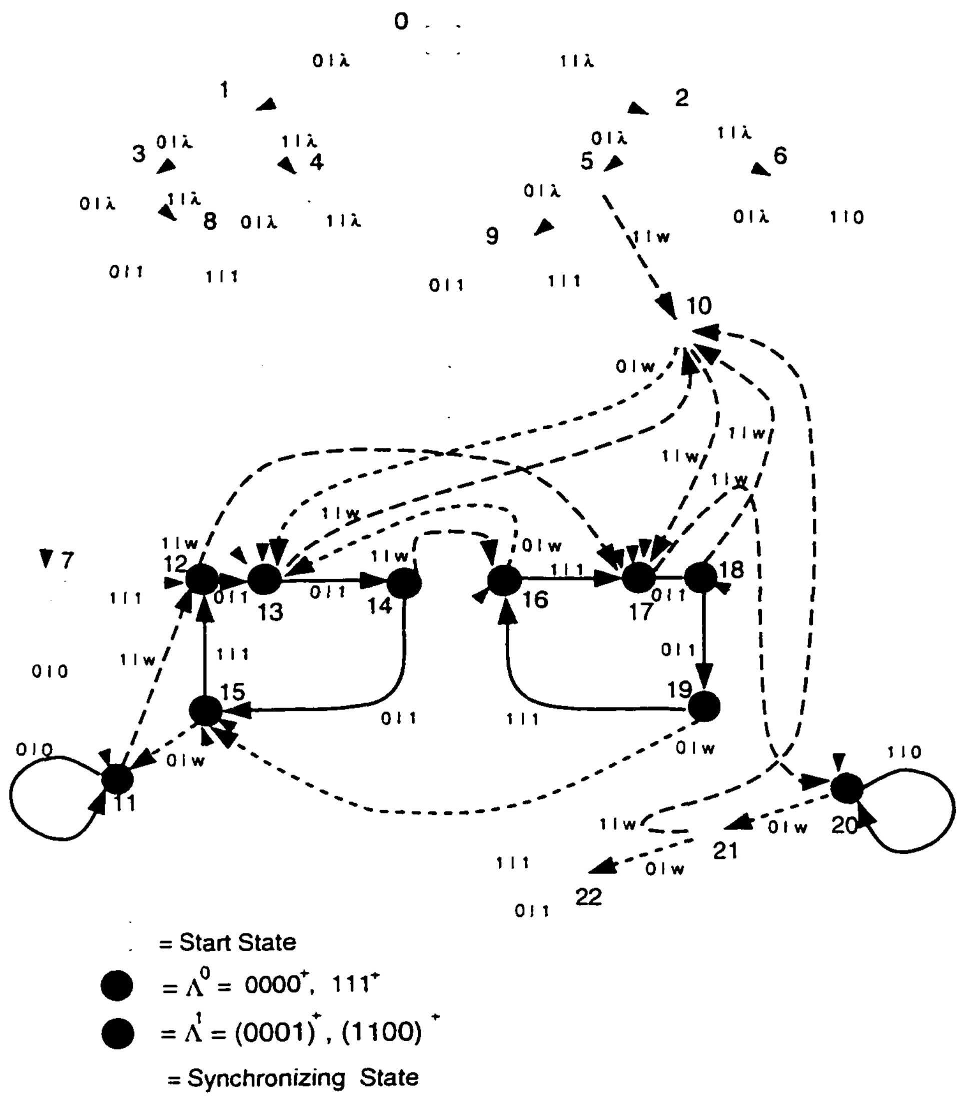


Figure 8.11: The spatial transducer T_B, which accepts words in domains Λ_B^0 and Λ_B^1.

the resulting plot is shown in Figure 8.10(b). The domains, particles and particle interactions observed in the filtered space-time plots of ϕ_B are presented in Table 8.3.

The most striking feature to be observed in Table 8.3 is its similarity to the domain, particle, and particle interaction table for rule ϕ_A. Indeed the number of domains, the number of particles, and the number and types of particle interactions are identical in the two tables. Most important of all, as in ϕ_A, the out-of-phase defect $S\overline{S}$ is unstable in rule ϕ_B, and the defect spontaneously gives rise to the D domain along with the particles γ and β. This allows two adjacent out-of-phase

Domains	
S alternates between 0000^+ and 111^+, i.e. $0000^+ = \Phi(111^+)$, $111^+ = \Phi(0000^+)$	D alternates between $(0001)^+$ and $(0011)^+$, i.e. $(0001)^+ = \Phi((0011)^+)$, $(0011)^+ = \Phi((0001)^+)$

Particles				
Symbol	Domain Boundary	Temporal Periodicity	Spatial Displacement	Velocity
α	$S\overline{S}$	-	-	0
β	DS	2	2	1
γ	SD	4	0	0
δ	DS	2	-6	-3
μ	SD	2	6	3
ν	$D\overline{D}$	8	0	0

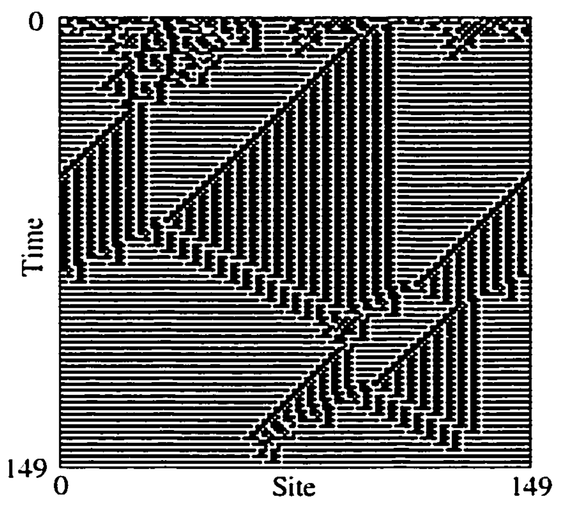
Particle Interactions			
Decay	$\alpha \rightarrow \gamma + \beta$		
Annihilative	$\gamma + \delta \rightarrow \emptyset$	$\mu + \beta \rightarrow \emptyset$	
Reactive	$\beta + \gamma \rightarrow \nu$, if $d \bmod 4 = 3$	$\nu + \delta \rightarrow \beta$	$\mu + \nu \rightarrow \gamma$
Reversible	$\beta + \gamma \rightarrow \delta + \mu$, if $d \bmod 4 \neq 3$	$\mu + \delta \rightarrow \gamma + \beta$	

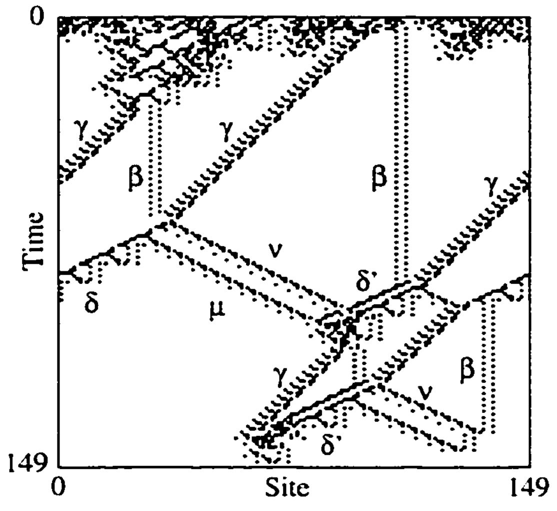
Table 8.3: The domains, particles, and particle interactions that dominate the spatio-temporal behavior of ϕ_B. The reactions between particles β and γ depend on the relative distance d between them, where $0 \leq d \leq 2r$.

synchronous domains to compete for space in the lattice with D acting as the mediating domain. Because the velocity difference between γ and β is less in ϕ_B than that in ϕ_A, the former takes more time to reach global synchrony. However, for most random initial configurations, the time required to reach synchrony is much less than $M \approx 2N$, where M is the maximum number of time steps for which the CA is allowed to operate, and N is the lattice size. Thus the performance of ϕ_B is not affected and it remains at 100%. In short, the strategy used by ϕ_B to perform the synchronization task is very similar to the strategy used by ϕ_A.

8.4 Analysis of ϕ_C

The space-time diagram from another high-performance rule, ϕ_C, is shown in Figure 8.12(a). Like ϕ_A and ϕ_B, ϕ_C also has 100% performance for the synchronization task. Analysis shows that two different regular domains dominate its space-time behavior. One of the domains, denoted here as Λ^0_C (and also as S), is the familiar





(a) (b)

Figure 8.12: (a) Space-time diagram of ϕ_C starting with a random initial condition. (b) The same space-time diagram after filtering with the transducer T_C which maps all domains to white and all defects to black. Greek letters label particles described in the text.

synchronous domain S described in the previous sections. The second domain, denoted here as Λ_C^1 (and also as D), has a temporal periodicity of two and a spatial periodicity of four. Λ_A^1 consists of two languages $(0001)^+$ and $(0011)^+$, such that $\Phi((1110)^+) = (1100)^+$ and $\Phi((1100)^+) = (1110)^+$. The configurations in Λ_C^1 however, have a spatial and temporal periodicity of four. Since both domains satisfy the temporal invariance and spatial homogeneity conditions, they are indeed regular domains.

Figure 8.13 shows the spatial transducer T_C, which accepts words in domains Λ_C^0 and Λ_C^1. It should be noted that T_C is the mirror image of the transducer T_B used in the previous section. Essentially, each input symbol in T_B has been flipped (i.e., a 0 to a 1, and a 1 to a 0) to create T_C. This is the case because the languages in the two domains in ϕ_B and in ϕ_C are Boolean complements of each other.

Figure 8.12(b) gives the space-time plot after T_C has been applied to the space-time diagram in Figure 8.12(a). The domains, particles, and particle interactions observed in the filtered space-time plot of ϕ_C are presented in Table 8.4.

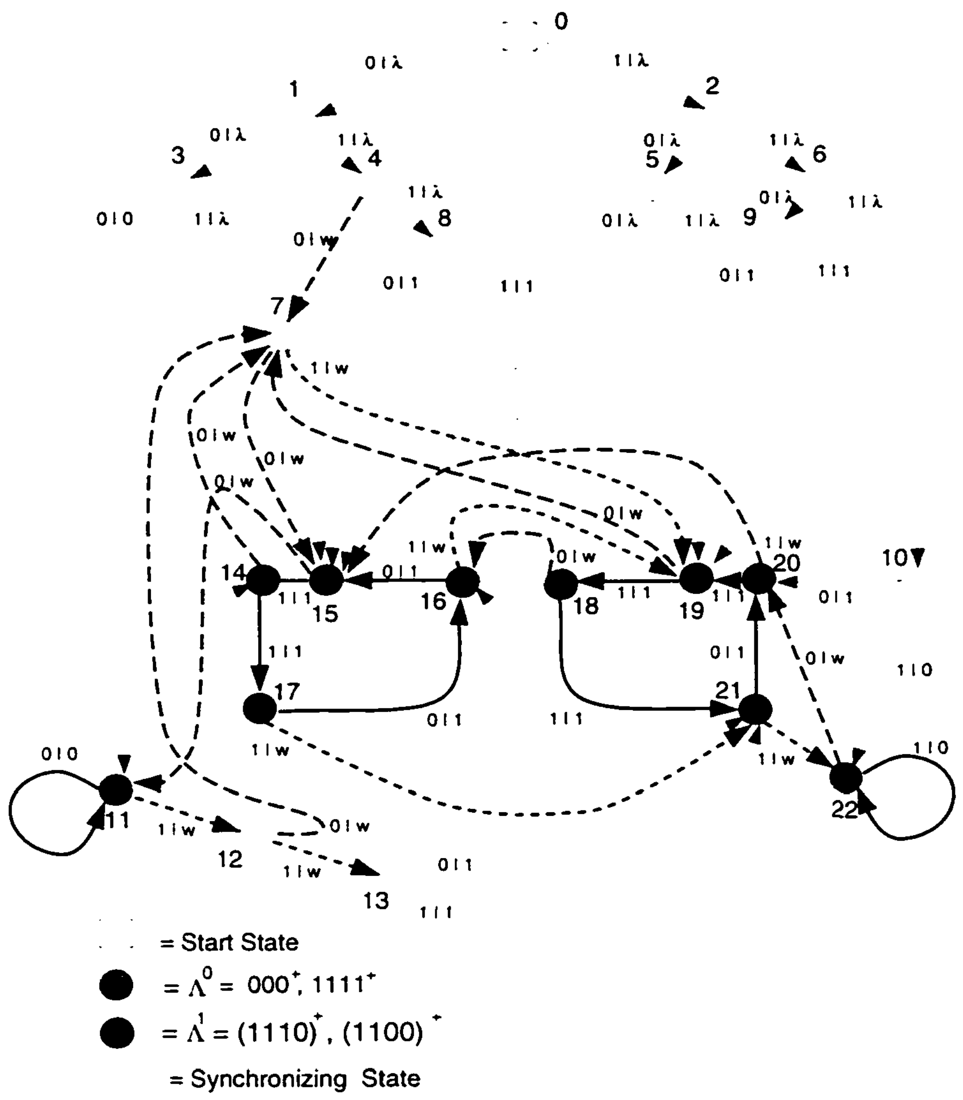


Figure 8.13: The spatial transducer T_C, which accepts words in domains Λ_C^0 and Λ_C^1.

An analysis of the particle logic implemented in ϕ_C makes it clear that although ϕ_C uses a different domain, and even though the particle velocities are different, the basic strategy is very similar to the particle logic used by ϕ_A and ϕ_B. As in ϕ_A, the out-of-phase defect $S\overline{S}$ spontaneously decays to γ and β which then interact with each other to create a set of particle interactions that result in global synchrony.

As in the density classification task, the GA discovered a variety of rules exhibiting high-performance for the synchronization task. Although the look-up tables for the high-performance CAs are very different from each other, and in spite of the

Domains	
S alternates between 000^+ and 1111^+, i.e. $000^+ = \Phi(1111^+)$, $1111^+ = \Phi(000^+)$	D alternates between $(1110)^+$ and $(1100)^+$, i.e. $(1110)^+ = \Phi((1100)^+)$, $(1100)^+ = \Phi((1110)^+)$

Particles				
Symbol	Domain Boundary	Temporal Periodicity	Spatial Displacement	Velocity
α	$S\overline{S}$	-	-	0
β	DS	2	0	0
γ	SD	4	-4	-1
δ ($\dot{\delta}$)	DS	6	-12	-2
μ	SD	2	4	2
ν	$D\overline{D}$	2	4	2

Particle Interactions			
Decay	$\alpha \to \gamma + \beta$		
Annihilative	$\gamma + \delta \to \emptyset$	$\gamma + \dot{\delta} \to \emptyset$	$\mu + \beta \to \emptyset$
Reactive	$\beta + \gamma \xrightarrow{d \bmod 4=0} \delta + \mu$	$\beta + \gamma \xrightarrow{d \bmod 4=1} \dot{\delta} + \mu$	$\beta + \gamma \xrightarrow{d \bmod 4=2} \delta + \mu + \nu$
	$\beta + \gamma \xrightarrow{d \bmod 4=3} \dot{\delta} + \mu + \nu$	$\mu + \dot{\delta} \to \gamma + \beta$	$\mu + \delta \to \gamma + \beta$
	$\nu + \beta \to \beta$		

Table 8.4: The domains, particles, and particle interactions that dominate the spatio-temporal behavior of ϕ_C. The reactions between particles β and γ depend on the relative distance d between them, where $0 \leq d \leq 2r$.

visually distinct patterns produced by the CAs, this chapter has shown that the embedded particle logic in these distinct rules are very similar to each other. The analysis is based on the computational mechanics framework, which facilitates the discovery and the quantification of the particle dynamics that lead to global synchronization in the system.

Chapter 9

CONCLUSION

This dissertation addressed two different but related issues regarding emergent computation in distributed multicomponent systems. Natural evolution has produced many systems which are composed of relatively simple components limited to local interactions. Yet, these systems display the capacity for globally coordinated information processing. Using CAs as abstract models of the class of such systems, we wanted to understand how information processing is embedded in the dynamical behavior of the systems. The goal was to discover, detect, and then quantify the underlying computational structures present in the system's spatio-temporal behavior. With this aim in mind, this research focused on how an evolutionary process interacts with a decentralized multicomponent system to produce emergent computation. Using a GA as an abstract computational model of an evolutionary process, we evolved CAs to perform two different computational tasks both of which required some form of global coordination. Since each CA consists of a number of potentially independent processors, the survival of a CA depended crucially on the amount of global coordination these processors were able to achieve. The emphasis in our work was to understand the mechanisms through which evolution may take advantage of a system's inherent dynamics to give rise to globally coordinated information processing. In other words, we wanted to study the evolutionary mechanisms that lead to the discovery and subsequent adaptation of the computational structures in the CAs.

Our results show that the GA is able to find CAs that have high-performance for the computational tasks. More importantly, analysis using the computational mechanics framework helped to discover and analyze the basic elements of computation embedded in the spatio-temporal behavior of the CAs that lead to improved performance. These elementary computational structures—namely, the domains and the particles—facilitated the emergent computation in the CAs. Our results clearly demonstrate that in all the high-performance CAs, the long-range information transmission that lead to global coordination was made possible by the underlying domains and particles in the CA's spatio-temporal behavior. Moreover, the results presented here indicate that as an evolutionary process the GA is not only able to discover particle-based computation in CAs, but it is also capable in manipulating the discovered computational structures to eventually find CAs with even better performance.

After having presented our findings in the preceding chapters, in this concluding note we return to some of the above issues in an attempt to present a context for the results obtained in this research and to outline the agenda for future research.

9.1 Understanding Information Processing in Spatially Extended Systems

Most of the analysis of CA behavior presented in this dissertation is based on the computational framework developed by Crutchfield and Hanson. This framework uses regular language and their associated finite automata as the mathematical underpinning to analyze the spatio-temporal behavior of CAs. The important notion of a regular domain is developed to characterize computationally homogeneous space-time regions. Other computational structures such as domain-boundaries, walls, or particles are defined as points where the domain pattern breaks down. Such particles can carry information over large space-time distances and the interaction among two or more particles may result in zero or more new particles. Logical operations on the information being carried by the particles occurs when they interact. The result of the logical operation is encoded in the state of the new particle (if any) that is produced by interacting particles. Thus, the domains, particles, and particle interactions are the key elements facilitating long-range information processing in a CA.

The preceding chapters have provided a detailed particle-level analysis of the high-performance CAs. The analysis not only delineated why the CAs exhibit superior performance in their respective tasks, but it also helped in comparing the particle-level "strategies" in different CAs. Our research indicates that, for a given task, all the high-performance CAs discovered by the GA employed similar strategies in their particle dynamics, despite the dissimilarities present in the entries of their look-up table, and the spatio-temporal details of their domains and particles. Without the aid of the computational mechanics framework, this fact would not have been discovered.

The notion of using particle-like structures to perform computation in a CA is not new. However, our result—a GA discovers particle-based in CAs—does highlight several salient and unique features in this regard. First, rather than designing a CA and its associated particles by hand, we use the GA as an automatic discovery mechanism to design CAs. Note that the GA has information only on *what* is to be designed, rather than information on *how* to design the high-performance CAs. This is a fundamentally different and new approach when compared to past efforts which were intrinsically laborious and time-consuming. Second, in using particle-like entities to perform computation most efforts up to now have designed particles as a coherent set of non-quiescent states moving against a homogeneous background of quiescent states. As indicated in Chapter 2, only a restricted set of CAs can have quiescent states. Thus, by choosing to focus on this restricted set, it is possible that simpler and more efficient ways of performing particle-based in CAs with no quiescent states are being ignored. In our work, the particles are defined at domain boundaries, where the domain themselves may contain non-quiescent and periodic states. Since the "density" of sites with non-quiescent is higher in such CAs, one of the interesting consequences is that these CAs may exhibit a much greater "density of computation" in its spatio-temporal behavior. Also, under similar circumstances (i.e., in the neighborhood size and in the number of states per site), CAs with no quiescent states may display more sophisticated computation in their particle-level dynamics than CAs with quiescent states. Since we have not imposed such restrictions on the CAs that are evolved by a GA, our approach has the potential to discover new and more efficient CA-solutions to the previously studied problems.

The analysis of information processing in CAs presented in this work is a preliminary step in understanding the behavior of more complicated natural and artificial systems. Extensions of this type of analysis are possible along several directions. One direction, which is being currently investigated, is the application of the computational mechanics framework to study the behavior of spatially-extended systems in more than one dimension. Many of the tools in the computational mechanics framework used to analyze one-dimensional CAs can be directly applied to study the behavior of two-dimensional CAs. Naturally, the dynamics of domains and particles in such systems can be expected to be much richer. For example, in a two-dimensional CA a domain-boundary can be associated with fundamentally new properties, such as the curvature of the boundary. These properties in turn can influence and enrich the particle-dynamics.

Research is also being done on stochastic spatially-extended systems. For example, consider one-dimensional probabilistic CAs with two states per site, where each entry in the look-up table defines the probability of obtaining the output bit of 1. One question that immediately arises is: Are domains and particles observed in such CAs? Interestingly, the answer is indeed yes, although new phenomena such as domain boundaries exhibiting stochastic dynamics may be observed. A similar setup also makes it feasible to study deterministic systems under the presence of small noise, i.e., systems where random error in rule update occurs at each site in the lattice with a small probability. This in turn raises the question: How robust are the domains and particles in the presence of noise? When noise is added, is a CA still able to perform the computation for which it was designed? Further experiments in this work has shown that for each of the two computational tasks, the high-performance CAs are robust enough to maintain their final configuration(s) in the presence of small noise ($\approx 2\%$ error rate) for a time-period that is much larger than N, the lattice size. Indeed, the high-performance CAs for the synchronization task can achieve global coordination even for nonzero error rate. In contrast, the performance of the highly-fit CAs for the density classification task degrade rapidly with increasing noise. Such phenomena are also being currently investigated.

Finally, another avenue of future research is to extend the computational mechanics to study continuous dynamical systems. As a preliminary step in this direction, work is in progress to apply computational mechanics to coupled map lattices (CML). Although CMLs use discrete-space and discrete-time representation, the variable associated with each site is real-valued. However, a symbol sequence can be generated from the spatial configuration in a CML by using a generating partition to discretize the value associated with each site. The resulting symbolic dynamics can be analyzed using the computational mechanics framework described in this work.

9.2 Evolution of Emergent Computation

The work presented here describes the use of a GA as an evolutionary process for optimization. This may raise some questions regarding the effectiveness of an evolutionary process in searching for optimality in phenotypic traits. However, there is growing evidence that adaptation via natural evolution may result in precise condition of an individual's phenotypes that lead to optimality. In particular, it has been quantitatively demonstrated that evolution can be reach optimality by tuning precisely and continuously one or more phenotypic traits. Our experiments show that the GA is able to adapt the phenotypic traits to obtain CAs that can perform a given computational task. These results open up the possibility for using GAs as an automatic discovery tool to design other spatially-distributed parallel computers. Success in designing such computers can have significance in the field of parallel computation.

Our interest in GAs is not limited to its application as an optimization technique. but also in its use as a computational model for biological evolution. In particular. we are interested in understanding how a GA succeeds in discovering high-performance CAs. As an important step in this direction. it is necessary to explain some of the characteristic features observed in the GA's behavior when it is used to evolve CAs.

The genetic operators in a GA act on the chromosomes encoding the phenotypic characteristics of the CAs. As observed in nature. our experiments show that of the multitude of gene mutations and recombinations that occur in a population in each generation. many are so minor as to be neutral in their effect on the corresponding phenotypic traits. Thus. such genotypic changes neither favor nor disfavor survival of an individual. The vast majority of genetic changes whose effects are significant enough to be readily detected are also harmful. Such changes are quickly removed from the population by natural selection. In contrast, when a new mutant or a novel recombination of preexisting sets of alleles happen to confer higher than average fitness to a chromosome, the new alleles tend to spread throughout the population in subsequent generations. In time. most of the chromosomes in the population contain these alleles, and they become the genetic norm.

This raises the question: Which alleles are responsible for superior performance in CAs? As our computational mechanics analysis has shown, high-performance in a CA can be traced back to the domains, particles and particle interactions observed in the CA's spatio-temporal behavior. Thus, a key step is to determine which alleles are responsible for the occurrence of these computational structures. Only then we can begin to understand the dynamics in a GA that leads to the discovery of these alleles.

Our analysis of the genotype in CAs shows that "constellations" of bits (or sets of "coadapted" alleles) in the chromosome are responsible for the stability and occurrence of the domains and particles observed in a CA's spatio-temporal behavior. For a given computational structure, a small change in its alleles sometimes results in only a minor change in the computational characteristics associated with that structure. However, in many cases even the smallest genotypic change in the constellation of bits results in a fundamental modification of the structure and produces a "new" structure with entirely different computational properties. Finally, in some situations, the mutation of single bit causes the structure to become unstable, and completely cease to appear in the CA's spatio-temporal behavior.

Each computational structure imposes a certain set of constraints on the allele values in the chromosome. To "implement" a set of computational structures, a number of such constraints on the allele values in the chromosome have to be satisfied. However, this may result in conflicts when two different structures try to enforce different allele values at the same locus. Viewed from another angle, this means that the presence of a given structure might guarantee the simultaneous existence or non-existence of another structure in the CA's behavior. This has important consequences in evolution. If a GA happens to discover a computational structure that confers high-fitness in CAs, the alleles responsible for the existence of the structure may become the genetic norm, and prohibit the presence of other potential structures in the future. In other words, the discovery may restrict the range and direction of potential evolutionary pathways.

This observed phenomena in our work bears similarity with the school of thought that seeks to determine the "principles of organization" in evolution [Goo92, FB94, HW93]. It claims that potential evolutionary pathways for a given phenotypic trait is fundamentally constrained by the genetic and structural constraints present in the organism itself. There are two schools of thought on how evolution discovers new phenotypic traits. Neo-Darwinists maintain that selection in the presence of genetic variation is sufficient to create new structures by gradually modifying the existing ones [Boc95]. On the other hand, the theory of punctuated equilibrium proposes that evolution is episodic, with periods of stasis and sudden burst of evolutionary activity. The theory emphasizes the role of happenstance in evolution [GE93].

In our model, we observe different evolutionary phenomena that also bears analogy with these different schools of thought. In most runs, the best and average performance in the population displays episodic behavior. There may exist long periods of stasis when no improvement is observed in the performance of the CAs in the population. During such periods, evolution of the population is mainly influenced by random drifts in the gene pool. When a constellation of high-fitness alleles is discovered, the entire population undergoes a major change with the discovered alleles becoming the new genetic norm. On other occasions, our GA exhibits gradual improvement in the performance of the CA's in the population. In such situations, the GA is able to incrementally modify the computational structures that result in incremental improvement in performance. The modifications in the computational structures are preceded by appropriate changes in the corresponding alleles which are in turn brought about by the genetic operators.

In this work, we have used the experimental setup of a "GA evolves CAs" to study how evolution can give rise to collective behavior in a distributed multicomponent system. How does such a research endeavor relate to the study of collective behavior in the biological world? A study of the literature from topics including evolutionary biology, molecular genetics, insect sociology, and bacterial pattern formation, shows that many of the issues discussed in this work are of fundamental importance in these fields. In many situations, the framework of a GA evolves CA (or its variations) may be well suited as abstract models to study the biological phenomena of interest. For example, in trying to explain collective behavior in ants. Wilson first notes that a small change in the behavioral characteristic of the individual insects in a colony can radically alter the behavior of the entire colony [HW93]. Such hypotheses invite analogy with CAs. Wilson goes on to propose a series of hypothesis regarding how such collective behavior might have appeared through evolution. These hypothesis are paraphrased below:

a the individual insect being unaware of the most of what is going on in the colony to which it belongs. responds in an ad hoc manner to stimuli it encounters moment to moment in its immediate environment:

b the responses and the probabilities of their occurrences are programmed genetically so that the mass behavior of a colony is efficient:

c the program is repeatedly altered in the course of evolution to produce individual behavioral patterns that are efficient in meeting the demands of the environment in which the species exists.

Wilson suggests that the reconstruction of colony behavior from a knowledge of the behavior of a single colony member is the central problem of insect sociology. Due to the long evolutionary time scale involved, many of the above hypotheses are difficult, if not impossible to study and verify in practice. Our suggestion here is that abtract computational models—such as the one studied here—may provide a suitable vehicle to study issues related collective phenomena in the biological world.

REFERENCES

[Atr65] A. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Computing*, EC-14:394, 1965.

[BB91] R. K. Belew and L. B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1991.

[BCG82] E. Berlekamp, J. H. Conway, and R. Guy. *Winning ways for your mathematical plays*, volume 2. Academic Press, New York, NY, 1982.

[BNR91] N. Boccara, J. Nasser, and M. Roger. Particle-like structures and their interactions in spatio-temporal petterns generated by one-dimensional deterministic cellular automaton rules. *Physical Review A*. 44:866, 1991.

[Boc95] G. Bochhi. Biological evolution: the changing image. In E. Laszlo, editor, *The new evolutionary paradigm*, pages 33–54, New York, NY, 1995. Gordon and Breach Science Publishers.

[Bon67] J. T. Bonner. *The Cellular Slime Molds*. Princeton University Press, Princeton, NJ, 1967.

[CH93] J. P. Crutchfield and J. E. Hanson. Turbulent pattern bases for cellular automata. *Physica D*, 69:279–301, 1993.

[CHY90] K. Culik, L. Hurd, and S. Yu. Computation theoretic aspects of cellular automata. *Physica D*, 45:357, 1990.

[CM94] J. P. Crutchfield and M. Mitchell. The evolution of emergent computation. Technical Report 94-03-012, Santa Fe Institute, Santa Fe, New Mexico, 1994.

[Cru91] J. P. Crutchfield. Reconstructing language hierarchies. In H. A. Atmanspracher and H. Scheingraber, editors, *Information Dynamics*, page 45, New York, 1991. Plenum.

[Cru92] J. P. Crutchfield. Discovering coherent structures in nonlinear spatial systems. In A. Brandt, S. Ramberg, and M. Shlesinger, editors, *Nonlinear Ocean Waves*, pages 190–216, Singapore, 1992. World Scientific. Also appears in Complexity in Physics and Technology, R. Vilela-Mendes, editor, World Scientific, Singapore (1992).

[Cru94] J. P. Crutchfield. Is anything ever new? Considering emergence. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models, and Reality*, volume XIX of *Santa Fe Institute Studies in the Sciences of Complexity*, Reading, MA, 1994. Addison-Wesley. In press.

[Cul93] K. Culik. How to fire almost any arbitrary pattern on a cellular automaton. In N. Boccara et al., editor, *Cellular Automata and Cooperative Systems*, Netherlands, 1993. Kluwer Academic.

[CY89] J. P. Crutchfield and K. Young. Inferring statistical complexity. *Physical Review Letters*, 63:105, 1989.

[CY90] J. P. Crutchfield and K. Young. Computation at the onset of chaos. In W. H. Zurek, editor, *Complexity, Entropy, and the Physics of Information*, pages 223–269. Addison-Wesley, Redwood City, CA, 1990.

[Dav91] L. D. Davis, editor. *The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[Dev89] P. Devreotes. *Dictyostelium discoideum*: A model system for cell-cell interactions in development. *Science*, 245:1054, 1989.

[DMC94] R. Das, M. Mitchell, and J. P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature—PPSN III*, volume 866, pages 344–353, Berlin, 1994. Springer-Verlag (Lecture Notes in Computer Science).

[EA94] M. Enquist and A. Arak. Symmetry, beauty and evolution. *Nature*, 372:169–172, 1994.

[FB94] W. Fontana and L. Buss. The arrival of the fittest. *Bulletin of Mathematical Biology*, 56:1–64, 1994.

[FHP86] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the Navier-Stokes equation. *Physical Review Letters*, 56:1505, 1986.

[FM93a] S. Forrest and M. Mitchell. Relative building block fitness and the building block hypothesis. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, San Mateo, CA, 1993. Morgan Kaufmann.

[FM93b] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning*, 13:285–319, 1993.

[For90] S. Forrest. Emergent computation: Self-organizing, collective, and cooperative behavior in natural and artificial computing networks: Introduction to the Proceedings of the Ninth Annual CNLS Conference. *Physica D*, 42:1–11, 1990.

[For93] S. Forrest, editor. *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA, 1993.

[Gac85] P. Gacs. Nonergodic one-dimensional media and reliable computation. *Contemporary Mathematics*, 41:125, 1985.

[GE93] S. J. Gould and N. Eldredge. Punctuated equilibrium comes of age. *Nature*, 366:223–227, 1993.

[GKL78] P. Gacs, G. L. Kurdyumov, and L. A. Levin. One-dimensional uniform arrays that wash out finite islands. *Probl. Peredachi. Inform.*, 14:92–98, 1978.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[Goo92] B. Goodwin. Evolution and generative order. In B. Goodwin and P. Sanders, editors, *Theoretical Biology: Epigenetic and evolutionary order from complex systems*, pages 89–100. Johns Hopkins University Press, Baltimore, MY, 1992.

[Gru92] F. Gruau. Genetic synthesis of Boolean neural networks with a cell rewriting developmental process. In L. D. Whitley and J. D. Schaffer, editors, *International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 55–72. Los Alamitos, CA, 1992. IEEE Computer Society Press.

[Gut90] H. A. Gutowitz, editor. *Cellular Automata.* MIT Press, Cambridge, MA, 1990.

[Han93] J. E. Hanson. *Computational Mechanics of Cellular Automata.* PhD thesis, The University of California, Berkeley, CA, 1993.

[HC92] J. E. Hanson and J. P. Crutchfield. The attractor-basin portrait of a cellular automaton. *Journal of Statistical Physics*, 66(5/6):1415–1462, 1992.

[HC95] J. E. Hanson and J. P. Crutchfield. Computational mechanics of cellular automata: An example. *Santa Fe Institute Working Paper*, (10-95), 1995.

[Hil90] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[HKP91] J. Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the theory of neural computation.* Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley, Reading, MA, 1991.

[HM91] J. H. Holland and J. H. Miller. Artificial adaptive agents in economic theory. Technical Report 91-05-025, Santa Fe Institute, Santa Fe, New Mexico, 1991.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[Hop95] J. J. Hopfield. Pattern recognition computation using action potential timing for stimulus reponse. *Nature*, 376:33-36, 1995.

[Hur87] L. Hurd. Formal language characterizations of cellular automaton limit sets. *Complex Systems*, 1:69, 1987.

[HW93] B. Holldobler and E. O Wilson. *The Ants.* Harvard University Press, Cambridge, MA, 1993.

[Jen90] E. Jen. Aperiodicity in one-dimensional ca. *Physica D*, 45:3, 1990.

[Joh94] R. A. Johnstone. Female preference for symmetrical males as a by-product of selection for mate recognition. *Nature*, 372:172–175, 1994.

[LB95] M. Land and R. K. Belew. No perfect two-sattecellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, 1995.

[LD94] G. Laurent and H. Davidowitz. Encoding of olfactory information with oscillating neural assemblies. *Science*, 265:1872 – 1875, 1994.

[LI95] J. E. Lisman and M. A. P. Idiart. Storage of 7 ± 2 short-term memories in oscillatory subcycles. *Science*, 267:1512–1515, 1995.

[LN90] K. Lindgren and M. G. Nordahl. Universal computation in a simple one-dimensional cellular automaton. *Complex Systems*, 4:299–318, 1990.

[LTFR92] C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors. *Artificial Life II.* Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley, Reading, MA, 1992.

[Maz87] J. Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50:183 – 238, 1987.

[MCH94a] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Dynamics, computation, and the "edge of chaos": A re-examination. In G. Cowan, D. Pines, and D. Melzner, editors, *Complexity: Metaphors, Models, and Reality*, pages 497–513, Reading, MA, 1994. Addison-Wesley.

[MCH94b] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361 – 391, 1994.

[Mei95] H. Meinhardt. *The Algorithmic beauty of sea shells.* Springer-Verlag, NY, 1995.

[MHC93] M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

[MHF] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing? In J. D. Cowan, G. Tesauro, and J. Alspector (editors), *Advances in Neural Information Processing Systems 6*. San Mateo, CA: Morgan Kaufmann.

[Mit92] M. Mitchell. Genetic algorithms. In L. Nadel and D. Stein, editors, *1992 Lectures in Complex Systems*, pages 3–87. Addison-Wesley, Redwood City, CA, 1992.

[Moo95] C. Moore. Quasi linear cellular automata. In preparation, 1995.

[MRH86] J. L. McClelland, D. E. Rumelhart, and G. E. Hinton. The appeal of parallel distributed processing. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing*, volume 1, pages 1–44, Cambridge, MA, 1986. MIT Press.

[MTH89] G. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384. Morgan Kaufmann, San Mateo, CA, 1989.

[Nis75] H. Nishio. Real time sorting of binary numbers by one-dimesional cellular automata. *Proceedings of the International Symposium on Uniformly Structured Automata and Logic*, 1975.

[NP94] D. E. Nilsson and S. Pelger. ??? *Proceedings of the Royal Society*, B256:53–58, 1994.

[Pac88] N. H. Packard. Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301, Singapore, 1988. World Scientific.

[PBS94] A. Prugel-Bennett and J. L. Shapiro. Analysis of genetic algorithms using statistical mechanics. *Physical Review Letters*, 72(9):1305–1309, 1994.

[PD84] K. Preston and M. Duff. *Modern Cellular Automata.* Plenum, New York, 1984.

[Pec83] J. Pecht. On the real-time recognition of formal languages in cellular automata. *Acta Cybernetica*, 6:33, 1983.

[SCED89] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 51–60. Morgan Kaufmann, San Mateo, CA, 1989.

[Sch89] J. D. Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA, 1989.

[Seg84] L. Segel. *Modeling dynamic phenomena in molecular and cellular biology.* Cambridge University Press, Cambridge, England, 1984.

[TM87] T. Toffoli and N. Margolus. *Cellular Automata Machines: A new environment for modeling.* MIT Press, Cambridge, MA, 1987.

[Tof84] T. Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D*, 10:117–127, 1984.

[Vic84] G. Y. Vichniac. Simulating physics with cellular automata. *Physica D*, 10:96–116, 1984.

[Vos93] M. D. Vose. Modeling simple genetic algorithms. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, San Mateo, CA, 1993. Morgan Kauffman.

[WDD91] L. D. Whitley, S. Dominic, and R. Das. Genetic reinforcement learning with multilayer neural networks. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 562–569, San Mateo, CA, 1991. Morgan Kaufmann.

[Wil93] E. O Wilson. *The diversity of life.* W. W. Norton & Co., New York, NY, 1993.

[WL92] A. Wuensche and M. Lesser. *The global dynamics of cellular automata.* Santa Fe Institute Studies in the Sciences of Complexity. Addison-Wesley, Reading, MA, 1992.

[Wol84a] S. Wolfram. Algebraic properties of cellular automata. *Communications of Mathematical Physics*, 93:219–258, 1984.

[Wol84b] S. Wolfram. Computation theory of cellular automata. *Communications of Mathematical Physics*, 96:15–57, 1984.

[Wol84c] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[Wol86] S. Wolfram, editor. *Theory and applications of cellular automata.* World Scientific, Singapore, 1986.

[WS92] L. D. Whitley and J. D. Schaffer, editors. *International Workshop on Combinations of Genetic Algorithms and Neural Networks.* IEEE Computer Society Press, Los Alamitos, CA, 1992.

[Yun94] J. B. Yunes. Seven-state solutions to the firing squad synchronization problem. *Theoretical Computer Science.* 127:313 - 332, 1994.

Appendix A

PROOF REGARDING THE DENSITY CLASSIFICATION TASK

Conjecture:

No two-state, finite-radius CA can perform the density classification task $\mathbf{T}_{\rho_c}$ perfectly for periodic lattices of all (odd) sizes N beyond some minimum value N_o.

The task $\mathbf{T}_{\rho_c}$ is defined over a lattice of size N where the value of N is odd. When the radius of a CA, r, is approximately equal to N, information about the state of the global configuration of the system can be stored in the neighborhood configurations of a CA. Such CAs can solve the $\mathbf{T}_{\rho_c}$ task, albeit trivially. On the other hand, for $r \ll N$, only a very limited amount of information regarding the entire system's state is accessible from each cell. This proof focuses exclusively on the more challenging latter constraint.

In the following proof, ρ_c is assumed to be 1/2. Nevertheless, the generalization to other values with $0.0 < \rho_c < 1.0$ is simple and straightforward [1].

Proof by contradiction:

Suppose there exists a perfect CA rule ϕ^p of radius r, which correctly classifies all configurations of length $N \geq N_o \geq 2r + 1$, according to their initial density to 1^N or 0^N. To guarantee the existence of the two fixed points 1^N and 0^N, ϕ^p must satisfy

[1]A number of researchers including Cris Moore, Mats Nordahl, Jonathan Amsterdam (all through personal communications) and Land & Belew [LB95] have proven this conjecture. While these proofs are distinct in their details, they all make use of CA's finite speed of information propagation to construct configurations that cannot be classified correctly. The proof presented here chooses the same overall approach.

the following two conditions in its look-up table: $\phi^p(0^{2r+1}) = 0$ and $\phi^p(1^{2r+1}) = 1$. As discussed in Chapter 3, ϕ^p also has to meet the following constraints on its global equation of motion Φ (for all strings s of length $N \geq N_o$):

$$\text{If } \rho(\mathrm{s}) < \frac{1}{2}, \text{ then } \rho(\Phi_i(\mathrm{s})) < \frac{1}{2} \quad \forall\, i \in \{0, 1, \ldots, \infty\} \tag{A.1}$$

$$\text{If } \rho(\mathrm{s}) > \frac{1}{2}, \text{ then } \rho(\Phi_i(\mathrm{s})) > \frac{1}{2} \quad \forall\, i \in \{0, 1, \ldots, \infty\} \tag{A.2}$$

To prove the conjecture, it is sufficient to construct an IC denoted s which the perfect rule cannot classify correctly and in the process violates Equation A.1 or Equation A.2.

First a new term $\nu(\mathrm{s})$ is introduced, which refers to the difference between the number of 1s and the number of 0s in configuration s. It should be remembered that in any configuration s, when a single 0 is replaced by a 1, then $\nu(\mathrm{s})$ is incremented by two.

Now consider an IC s^a on a periodic lattice of length l with $\rho(\mathrm{s}^a) > 1/2$ and $l \geq N_o \geq 2r + 1$, such that $\rho(\Phi(\mathrm{s}^a)) > \rho(\mathrm{s}^a)$. As a result $\nu(\Phi(\mathrm{s}^a)) - \nu(\mathrm{s}^a) \geq 2$.

Next consider another IC s^b on a periodic lattice of length ml consisting of m copies of s^a concatenated together, where m is a positive integer $\geq 2r + 1$. Obviously, $\nu(\mathrm{s}^b) = m\nu(\mathrm{s}^a)$, and because the boundary conditions for each copy of s^a remain unaffected, $\nu(\Phi(\mathrm{s}^b)) = m\nu(\Phi(\mathrm{s}^a))$. Since $\nu(\Phi(\mathrm{s}^a)) - \nu(\mathrm{s}^a) \geq 2$, we find $\nu(\Phi(\mathrm{s}^b)) - \nu(\mathrm{s}^b) \geq 2m$.

Finally another IC s^d is created by concatenating s^b to s^c, where s^c is a block of 0s such that s^d has exactly one more 0 than 1, that is, $\nu(\mathrm{s}^d) = -1$ and $\rho(\mathrm{s}^d) < 1/2$. For $\nu(\mathrm{s}^d) = -1$, we need s^c to be a block of 0s of length $\nu(\mathrm{s}^b) + 1$ (that is, $\nu(\mathrm{s}^c) = -(\nu(\mathrm{s}^b) + 1)$). It should be noted here that s^c has a length $\geq 2r + 1$ since $\nu(\mathrm{s}^b) + 1 = m\nu(\mathrm{s}^a) + 1$. The new configuration s^d has a length equal to $ml + \nu(\mathrm{s}^b) + 1$.

Given this configuration s^d with $\nu(\mathrm{s}^d) = -1$, the main aim is to determine a lower bound on $\nu(\Phi(\mathrm{s}^d))$ and to compare it with $\nu(\mathrm{s}^d)$. As s^d consists of two substrings s^b and s^c, the analysis can be made simpler by first considering how the global map operates separately on each of the substrings and then making appropriate adjustments for the interactions between the two substrings at their boundaries: that is,

$$\nu(\Phi(\mathrm{s}^d)) = \nu(\Phi(\mathrm{s}^b)) + \nu(\Phi(\mathrm{s}^c)) + \text{interactions at the substring boundaries}$$

Ignoring the effects of the interactions between the two substrings it is apparent that $\nu(\Phi(\mathrm{s}^c)) = \nu(\mathrm{s}^c) = -(\nu(\mathrm{s}^b) + 1)$ since $\phi^p(0^{2r+1}) = 0$.

Given that each site in a radius r CA has information of only r adjacent sites on each side, in a single time step only the r contiguous sites at the extremity of a substring can be influenced by an adjacent substring. Consequently, at each extremity of a substring, the number of 0s (or 1s) can increase or decrease at most by an amount r. Since two adjacent substrings can reciprocally influence one another, the net change in the number of 0s (or 1s) in both the substrings at a single boundary is therefore $2r$. Both substrings s^b and s^c are of length $\geq 2r + 1$ and in a periodic lattice they meet at two boundaries. As the distance between the two boundaries is more than $2r + 1$, in a single time step the interactions at one boundary cannot influence the dynamics at the other boundary. Thus the net gain in the number of 0s (or 1s) due to interactions between the two substrings is at most $4r$, with a change in $\nu(\Phi(\mathrm{s}^d))$ by as much as $8r$. However, in this situation, we want a lower bound for $\nu(\Phi(\mathrm{s}^d))$, and so we are only interested in bits that flip from 1 to 0 at the two boundaries. Since at most $2r$ of the $4r$ bits at the two boundaries are 1s, then $\nu(\Phi(\mathrm{s}^d))$ cannot decrease by more than $4r$ due to the interactions between the substring boundaries. (However, it is possible that $\nu(\Phi(\mathrm{s}^d))$ could increase by more than $4r$.)

From the above discussion, a lower bound for $\nu(\Phi(\mathrm{s}^d))$ can be determined.

$$\begin{aligned}\nu(\Phi(\mathrm{s}^d)) &= \nu(\Phi(\mathrm{s}^b)) + \nu(\Phi(\mathrm{s}^c)) + \text{interactions at the substring boundaries}\\ &\geq \nu(\Phi(\mathrm{s}^b)) - (\nu(\mathrm{s}^b) + 1) - 4r\end{aligned}$$

$$\geq (\nu(\Phi(s^b)) - \nu(s^b)) - 1 - 4r$$

$$\geq 2m - (4r + 1)$$

Since $m \geq 2r + 1$, we find $\nu(\Phi(s^d)) \geq 1$, and consequently $\rho(\Phi(s^d)) > 1/2$. But this is a violation of Equation A.1, contradicting the assumption of a perfect rule.

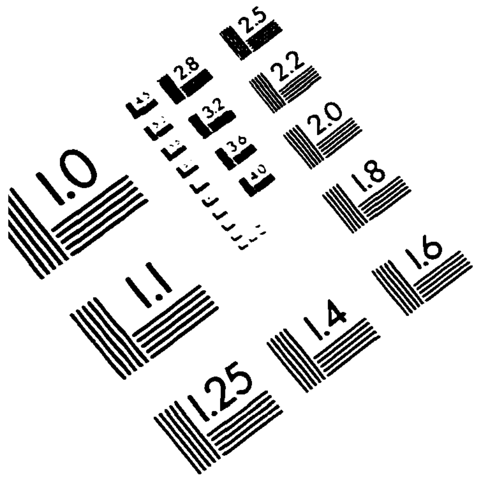
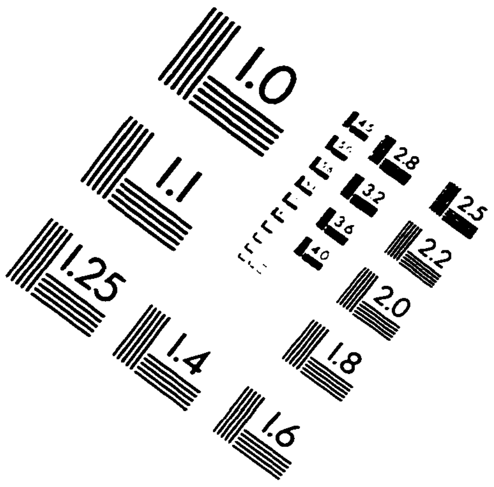
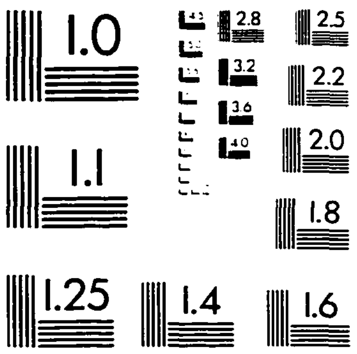
Since the task $\mathbf{T}_{\rho_c}$ is symmetric with respect to 0s and 1s, a similar line of argument proving the nonexistence of a perfect rule can be proposed with $\rho(\Phi(s^a)) < \rho(s^a) < 1/2$ and with s^c consisting of a block of 1s.

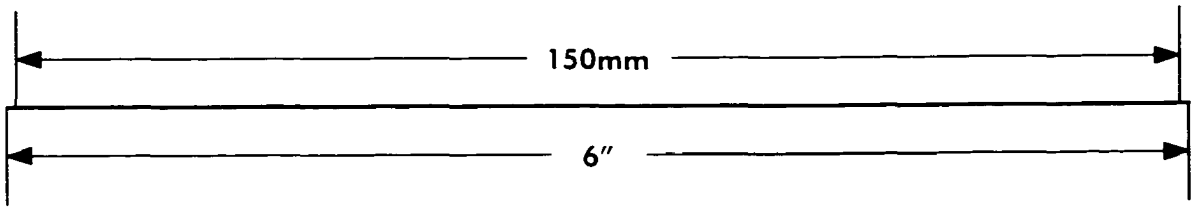
What is the minimum lattice size on which this proof is valid? As indicated earlier, the length of the configuration s^d is $N = ml + \nu(s^b) + 1 = ml + m\nu(s^a) + 1$. We already know that both m and l are $\geq 2r + 1$. The minimum value of $\nu(s^a)$ for which this proof holds is 1. Substituting these values, we find that $N = (2r + 1)^2 + (2r + 1) + 1 = 4r^2 + 6r + 3$. Clearly, this is an upper bound on the smallest possible lattice size. In the case of a CA rule where there exists a configuration s^a of length $2r + 1$ such that $\nu(\Phi(s^a)) - \nu(s^a)$ is strictly greater than 2, a smaller value of m can be chosen to achieve $\nu(\Phi(s^d)) \geq 1$, and thus to obtain a smaller lattice size which is sufficient to satisfy the proof.

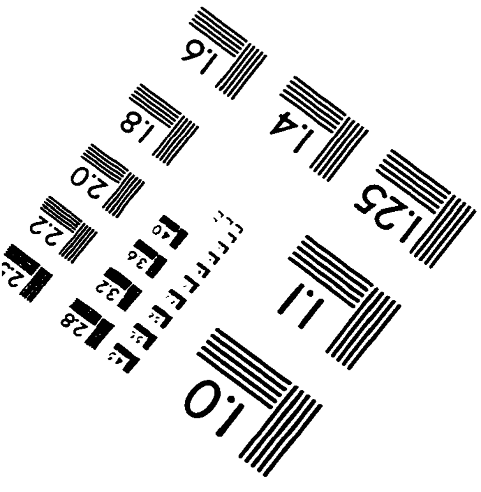
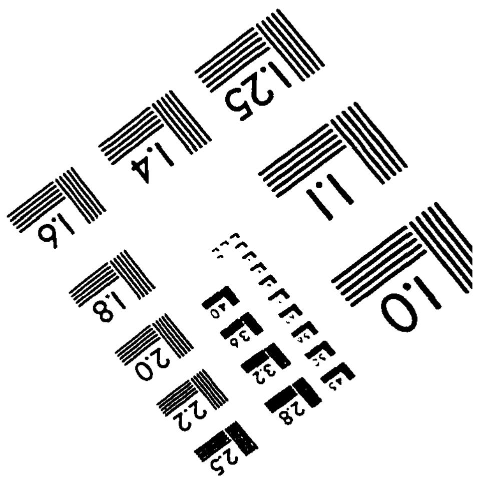
As an example of this proof, let $r = 1$, $l = 2r + 1 = 3$, and $s^a = 101$. The latter must be transformed by a perfect rule such that $\Phi(s^a) = 111$. Here $\nu(\Phi(s^a)) - \nu(s^a) = 2$. Using the smallest value of $m = 2r + 1 = 3$ necessary to satisfy the proof, 3 copies of s^a are concatenated together to form $s^b = 101101101$. Obviously, $\nu(s^b) = 3$ and $\Phi(s^b) = 111111111$ with $\nu(\Phi(s^b)) = 9$. Now s^c is created to consist of a block of four 0s ($\nu(s^b) + 1 = 4$) resulting in $s^d = 1011011010000$ with $\nu(s^d) = -1$ and $\rho(s^d) < 1/2$. Note that the length of $s^d = 4r^2 + 6r + 3 = 13$. Under the global map operator Φ, the minimum number of 1s in $\Phi(s^d)$ is 7 with $\Phi(s^d) = 0111111100000$. Thus $\nu(\Phi(s^d)) = 1$ and $\rho(s^d) > 1/2$, and the operation of the assumed perfect CA violates Equation A.1.

For other values of ρ_c, we can choose s^c (i.e. a block of 0s) of appropriate length and then employ an identical line of argument delineated above to prove that the density classification task $\mathbf{T}_{\rho_c}$, for $0 < \rho_c < 1$ cannot be performed perfectly by any two-state, finite-radius CA on a periodic lattice.

IMAGE EVALUATION TEST TARGET (QA-3)



APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989