SANDIA REPORT

SAND2008-6041
Unlimited Release
Printed September 2008

Mathematical Approaches for Complexity/Predictivity Trade-Offs in Complex System Models: LDRD Final Report

Jackson R. Mayo, Robert C. Armstrong, Michael E. Goldsby, Keith B. Vanderveen, Arnab Bhattacharyya

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2008-6041
Unlimited Release
Printed September 2008

Mathematical Approaches for Complexity/Predictivity Trade-Offs in Complex System Models: LDRD Final Report

Jackson R. Mayo
Visualization & Scientific Computing

Robert C. Armstrong
Scalable Computing R&D

Michael E. Goldsby
Keith B. Vanderveen
Exploratory Computer & Software Engineering

Sandia National Laboratories, P.O. Box 969, Livermore, CA 94551-0969

Arnab Bhattacharyya, Electrical Engineering & Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139-4307

Abstract

The goal of this research was to examine foundational methods, both computational and theoretical, that can improve the veracity of entity-based complex system models and increase confidence in their predictions for emergent behavior. The strategy was to seek insight and guidance from simplified yet realistic models, such as cellular automata and Boolean networks, whose properties can be generalized to production entity-based simulations. We have explored the usefulness of renormalization-group methods for finding reduced models of such idealized complex systems. We have prototyped representative models that are both tractable and relevant to Sandia mission applications, and quantified the effect of computational renormalization on the predictive accuracy of these models, finding good predictivity from renormalized versions of cellular automata and Boolean networks. Furthermore, we have theoretically analyzed the robustness properties of certain Boolean networks, relevant for characterizing organic behavior, and obtained precise mathematical constraints on systems that are robust to failures. In combination, our results provide important guidance for more rigorous construction of entity-based models, which currently are often devised in an ad-hoc manner. Our results can also help in designing complex systems with the goal of predictable behavior, e.g., for cybersecurity.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Many systems of interest to Sandia missions show properties of self-similarity, power-law scaling, and emergent behavior. Examples include critical infrastructures such as the Internet [30], the financial system [31], and electrical grids [10], as well as other phenomena of relevance to national security, such as the formation and behavior of social networks (including terrorist cells) [5] and the spread of communicable diseases [26]. The emergence of unpredictable behavior from a collection of components whose behavior is well-understood (or assumed to be well-understood) is the hallmark of what is known as a *complex system* [22]. Sandia and other research institutions have invested significant resources developing models of complex systems of interest [27], but to date there has been no validated set of principles for constructing these models in a manner that maximizes model rigor and veracity.

Another problem that frequently plagues efforts to model complex systems is that the behavior of their components and interactions may be understood, in principle, but the governing physics is too complicated to simulate effectively, and may not be relevant to the phenomena of interest in the system. An example would be social interactions between people, about which a great deal is known in terms of culturally-specific patterns of interaction, social norms, etc., most of which is abstracted away in models of social networks. Another example concerns cascading failures in electrical grids. Much is known about how transformers fail physically, yet it is believed that not all of the detailed physics of transformer failure is needed to accurately model cascading electrical grid failures [10]. Currently, little is known about how such "abstracting away" of the details of component-component interactions during the modeling process affects the veracity and rigor of the model.

The real danger of abstracting away too much detail of component behavior and component-component interaction can be seen in the context of the evaluation of the security properties of computer systems. To make the job of designing computer hardware and software easier, designers typically employ a paradigm of modular design, in which a component or module will have a simple interface with other components/modules (often a protocol) that hides (or encapsulates) more complicated internal implementation. Frequently, when the security properties of a system are evaluated, these interfaces (which are abstractions of the true component behavior) are taken at face value, as the actual behavior of the components in question. The reality is that hardware

and software components exhibit much richer "interfaces" in the real world than the ones specified in design documents, a fact which hackers have exploited numerous times to break into systems. A simple recent example is the "freezing" of computer DRAM to preserve the contents of volatile memory after a machine is powered off, which allowed security researchers to retrieve cryptographic keys and other data from an otherwise secure system [16].

One of the most important questions concerning the modeling of complex systems is, how much abstraction is safe? Are there general mathematical principles that can be applied to answer this question? To date, most efforts to model complex systems have applied abstraction during the course of model construction in a context-dependent, ad-hoc fashion. The most rigorous modeling efforts undertake some sort of validation of models against real-world observations of the system of interest, and attempt to account in the model for those aspects of the real system having the greatest influence on the system behavior through sensitivity studies. Sensitivity studies, however, require exhaustive study of the system of interest, which may not always be possible with the resources available for the modeling effort. Furthermore, when dealing with systems that exhibit complex, emergent phenomena, it can be difficult to know when enough observations have been collected to build a model that will yield useful results. This problem is especially pronounced when modeling real-world systems or phenomena that have not occurred, such as a massive catastrophic failure of the modern financial system or extensive global warming. It would be enormously helpful to the modeling of many different kinds of complex phenomena to be able to know when a model has captured enough detail of the real-world system to be useful, so that a particular modeling effort can be guided by (and compared against) rigorous, objective criteria, as opposed to the intuition of the modelers.

1.2 Research Goals

One goal of our research was to explore the use of renormalization-group methods to find reduced models for complex systems. Our expectation was that finding methods for computing these reduced models would yield greater insight into the emergence of complexity in the systems being studied, and might also yield models that were easier to understand yet still captured the essence of the phenomena of interest in the original system. One of our hopes is that the preliminary research we have undertaken in this LDRD will allow us to begin to answer the questions raised in the previous section, whether there are universal principles governing the effects abstraction has on models of complex systems. One can view the renormalization-group methods we have explored in this LDRD as a formalized version of the kind of abstraction that goes on in efforts to construct models of real-world complex systems. Because of the long use of renormalization-group methods in physics, there are many mathematical tools we can use to study renormalization-group methods for model reduction of complex systems, making it easier to study than the ad-hoc methods of abstraction now used to build models of complex systems. However, we expect that results we have proved concerning renormalization-group methods will generalize to more ad-hoc methods of abstraction, allowing us to make statements about the veracity and rigor of models of complex real-world phenomena.

A second goal of our research was to explore how the structure of a complex system affects the robustness of that system in response to perturbations in its environment. To make the problem tractable, and allow us to bring well-developed analytical tools to bear on the problem, we chose to study the structure-robustness relationship in the context of Boolean networks. Boolean networks are graphs in which each node has a Boolean state (0 or 1) that changes at the next timestep according to some function of the current state and the state of the node's neighbors [22]. While simple, Boolean networks can model complicated dynamics of interest in fields such as biology, epidemiology, electrical engineering, computer science, etc. [3]. In particular, VLSI chips composed of digital logic circuits can be represented directly as Boolean networks [3]. Our research has explored what constraints on the structure of Boolean networks lead to robustness in the presence of environmental perturbations. We believe that these results can be generalized to show constraints on the structure of real-world complex systems that show robustness in the presence of environmental change, such as social organizations (including terrorist networks), biological organisms, computer networks, etc. Our results can hopefully inform efforts to model such systems. Furthermore, we believe our research results might have direct applicability to the problem of designing logic hardware, software, and protocols that resist faults and attacks.

Chapter 2

Cellular Automata

2.1 Background

Cellular automata provide an especially simple setting to illustrate the emergence of rich phenomena from basic underlying rules. Extensive theoretical and computational results have been previously obtained for cellular automata, showing that these systems exhibit a wide range of behaviors seen in the natural and manmade world, as documented by Wolfram [34]. It has even been suggested that cellular automata and related systems are candidates for expressing fundamental laws of nature, in contrast to traditional mathematical approaches based on differential equations [34]. At a different level, the relevance of such idealized systems to real-world applications is illustrated by a cellular-automaton model for the adoption of a new technology in an economy [24].

The essence of a cellular automaton is its regular structure and communication pattern. A cellular automaton consists of a lattice of *cells*, each of which carries a definite state at any given time. In most cases, the state of a cell is discrete and thus can be represented by an integer. Furthermore, the evolution of the system is carried out in discrete timesteps. As a result, a specification of the underlying dynamics of the system can be exactly reproduced in a computer simulation, provided enough memory and processing time are available. The lattice of cells can exist in a "space" of one, two, three, or more dimensions, although dimensions greater than three are difficult to visualize and also expensive to simulate. Typical cellular automata of interest are in one dimension (such as the fundamental Wolfram cellular automata) or two dimensions (such as the sandpile model to be discussed below). Another two-dimensional cellular automaton is the Game of Life, which has been widely popularized and offers evolution patterns with high visual interest [14].

The most common setting for cellular automata is the simple Cartesian lattice (e.g., a square lattice in two dimensions); this is the case we investigate here, although other regular lattice types are possible. Also, because a finite lattice must be used in any simulation, the treatment of the boundaries of the lattice must be specified. The simplest approach, which we adopt, is to impose "periodic" or "toroidal" boundary conditions; that is, the lattice is conceptually wrapped around so that each cell on a boundary is considered adjacent to the corresponding cell on the opposite boundary. A key advantage of periodic boundary conditions is that all cells are equivalent and the lattice structure is invariant to arbitrary shifts, just as for a hypothetical infinite lattice.

The procedure for "updating" a cellular automaton (evolving to the next discrete timestep) is usually specified via a function that determines the new state of a given cell based on the current state of that cell and its nearest neighbors (for our purposes, $2d$ neighbors in a d-dimensional Cartesian lattice; in some cases, such as the Game of Life, the "diagonal" neighbors are also included). The limitation to nearest-neighbor dependence reflects the locality of interactions in physical space, and also reduces the number of possible cellular automaton specifications. A further assumption is that the updating function is the same for all cells; this makes the system dynamics (like the lattice structure) homogeneous in space, and allows the use of mathematical descriptions such as Fourier analysis. Conceptually, the updating function is applied to all cells in parallel, using the current states, and the new states all take effect simultaneously. This synchronous updating is actually implemented in a standard computer by allocating a separate memory area for the new states, storing in that area the result of the updating function for each cell to avoid overwriting the current states, and finally treating the new memory area as encoding the state of the system at the next timestep (the old states can then be discarded). So far we have described a cellular automaton with deterministic behavior. A cellular automaton can also be defined so that the result of the updating function depends not only on the given cell and its neighbors, but on an additional input that reflects an external perturbation, such as a random number that introduces stochastic behavior. We refer to this external input as a *trigger* and assume that it is also an integer.

Thus, with the usual simplifying assumptions, a cellular automaton model for a d-dimensional lattice is specified by giving an integer function (of $2d+1$ integer variables and possibly an additional trigger), used for updating each cell. If the cells are intended to have a finite set of possible states, then this function must have the property that its result belongs to this set when its inputs do; as a result, proper initial conditions (valid states of all cells at a starting time) will result in evolution that remains in the assigned set. The simplest nontrivial set of states is the Boolean case with two possible values, say 0 and 1. This case already allows some rich dynamics for cellular automata [34], and it also provides a starting point for another prototype system to be discussed in Chapter 3, namely Boolean networks, which keep the properties of cellular automata we have described (with deterministic evolution), except that the entities do not have identical updating rules, and the communication among them does not follow a lattice structure, but rather some arbitrary connectivity. However, for cellular automata, it is useful to compensate for the simplicity of the lattice structure by allowing a larger state space. We next describe such a model, which we adopt as our prototype cellular automaton.

2.2 The Sandpile Model

2.2.1 Definition of the Model

The starting point for our prototype cellular automaton is the Bak–Tang–Wiesenfeld (BTW) sandpile model [4], which is defined on a two-dimensional square lattice. This model represents an idealization of the complex behavior of a pile of sand, which becomes unstable when its slope exceeds a critical value. As a result, if sand is randomly added to a pile in various locations,

"avalanches" eventually occur; such an avalanche continues until the slope is everywhere below the critical value, restoring stability. Depending on the exact configuration at the location and time of the perturbation, the avalanche may be localized or it may sweep over a large part of the system. The sandpile model can thus offer an idealized representation of a process of cascading failure, relevant to many real-world applications. An additional important property of the model is discussed in Section 2.2.2.

As originally defined by BTW, the sandpile model does not completely conform to the definition of a cellular automaton, because the evolution procedure involves making a perturbation, evolving the system until the states no longer change, and then making the next perturbation. This requires monitoring the system as a whole, whereas a cellular automaton should have a consistent local updating rule applied to every cell at every timestep. To fix the discrepancy, we here define an updating rule that receives a random trigger to determine whether a perturbation is applied. This trigger is an independently generated random number for each cell at each timestep, thus maintaining locality. The result is that perturbations are applied at events that form a Poisson process in discrete space-time. If the density of this Poisson process is sufficiently low, each avalanche is likely to run its course with little interference, and the outcome will be similar to the globally monitored case.

A further modification we introduce is in the details of the perturbation. In the existing sandpile model [4], each perturbation adds 1 to the value of the chosen cell, representing the addition of a grain of sand. As a result, the total number of grains of sand in the system would increase with time; this is compensated by allowing sand to disappear at the boundaries of the lattice, as if it were falling off the edge of a table. Because we prefer to work with periodic boundary conditions, we use a perturbation that can both add and remove grains of sand. Remarkably, a very simple perturbation can accomplish this: simply not updating the given cell and instead keeping its state unchanged. As discussed in Section 2.2.2, this choice preserves the main behaviors of the BTW model but leads to additional, richer phenomena.

Our sandpile cellular automaton model is as follows. On a two-dimensional lattice with periodic boundary conditions, the state of each cell (x,y) is $z(x,y) \in \{0,1,\ldots,7\}$; we define our model to maintain this state space, whereas the BTW model does not prescribe a bound on possible states. Our normal updating rule is precisely that of BTW,

$$z(x,y) \leftarrow z(x,y) - 4\lfloor \tfrac{1}{4} z(x,y) \rfloor + \lfloor \tfrac{1}{4} z(x-1,y) \rfloor + \lfloor \tfrac{1}{4} z(x+1,y) \rfloor + \lfloor \tfrac{1}{4} z(x,y-1) \rfloor + \lfloor \tfrac{1}{4} z(x,y+1) \rfloor, \tag{2.2.1}$$

where $\lfloor\ \rfloor$ denotes rounding down to an integer. This rule maintains our state space $\{0,1,\ldots,7\}$, because the sum of the first two terms on the right-hand side is in $\{0,1,2,3\}$, and each of the remaining terms is in $\{0,1\}$. Also, as noted by BTW, the rule results in conservation of the total amount of sand, $Z \equiv \sum_{x,y} z(x,y)$, because when the change in this sum is evaluated, each term $\lfloor \frac{1}{4} z(x,y) \rfloor$ appears once with coefficient -4 and four times with coefficient $+1$. Intuitively, the rule describes an avalanche process in which a cell with $z(x,y) \geq 4$ transfers one unit of the conserved quantity (one grain of sand) to each of its four nearest neighbors.

To introduce perturbations, we allow each update to leave the local state unchanged,

$$z(x,y) \leftarrow z(x,y), \tag{2.2.2}$$

instead of applying Eq. (2.2.1), if the trigger so directs. This occurs with a fixed small probability (an independent random choice for each cell at each timestep). The triggering of this "non-updating" rule for a given cell has no effect on the updating of neighboring cells. As a result of a non-update of (x,y), the sum Z changes by

$$\begin{aligned}\Delta Z &= 4\lfloor \tfrac{1}{4}z(x,y)\rfloor - \lfloor \tfrac{1}{4}z(x-1,y)\rfloor - \lfloor \tfrac{1}{4}z(x+1,y)\rfloor - \lfloor \tfrac{1}{4}z(x,y-1)\rfloor - \lfloor \tfrac{1}{4}z(x,y+1)\rfloor \\ &\in \{-4,-3,\ldots,4\},\end{aligned} \tag{2.2.3}$$

i.e., the negative of the amount by which $z(x,y)$ would otherwise have changed. Thus the cellular automaton is driven by both adding and removing various amounts of the conserved quantity.

2.2.2 Self-Organized Criticality

The BTW sandpile model was introduced to illustrate the then-new concept of self-organized criticality. This refers to systems that, from random initial states, spontaneously tend toward a highly structured dynamic regime. Such a regime involves similar phenomena occurring on a wide range of space and time scales, with a scale-invariant power-law distribution of activity over these scales. A self-organizing cellular automaton, despite the limitation to nearest-neighbor interactions at each step, develops coherent long-range structures over time. The behavior is called "criticality" because it is analogous to the scale-invariant behavior of systems in statistical physics, such as magnets and fluids, at a thermodynamic critical point (second-order phase transition). But unlike thermodynamic critical behavior, which requires fine tuning of parameters such as temperature to reach a phase transition, the key property of self-organized criticality is that a system is attracted to such a regime spontaneously under generic conditions.

The self-organized criticality of the BTW sandpile model was demonstrated through a power-law distribution of the size of regions affected by avalanches [4], a somewhat indirect property. However, a more conventional diagnostic of critical behavior in statistical physics—a power-law correlation function (or power-law spatial Fourier spectrum) of the state fluctuations—is not realized in the BTW sandpile model, with only short-range correlations developing between cell states [7]. Remarkably, our variant sandpile model produces long-range power-law correlations, closely resembling those of thermodynamic critical points, but without the associated fine tuning. This richer scale-invariant behavior is shown through numerical simulations in Chapter 6.

Chapter 3

Boolean Networks

3.1 Background

Boolean networks, like cellular automata, are a particular type of discrete dynamical system. They were first proposed by Kauffman as random models of genetic regulatory networks [21]. They have a simple structure yet rich dynamical behavior and have been used as models of complex dynamical systems in a number of different fields: genetic networks [22], the theory of evolution [22], social science [29], biochemical reaction pathways [9], and others.

A Boolean network can be represented as a directed graph. Each vertex of the graph has a Boolean state value (0 or 1). The ability to specify an arbitrary connectivity pattern among vertices allows Boolean networks to model diverse real-world systems, which may not be well represented by a lattice. The most commonly considered graph topology results in what is called a Kauffman network or N-K network. For such a network, there are N vertices (nodes), and each vertex has K in-neighbors. It is common to select the neighbors and the initial states randomly. The random approach can be used to model natural systems (such as genetic regulatory networks) that surely have a structure that is not completely random, but one that is complex and largely unknown.

Each vertex or node sends its value to all its out-neighbors. Each node also has a transfer function, a Boolean function of its inputs that is typically also randomly chosen. At each timestep, all nodes are updated in parallel by replacing each node's value by the value of the node's transfer function applied to the node's inputs. Asynchronous updating of the nodes has also been investigated [17]. The transfer function of a given node, once chosen, can be left unchanged (the *quenched dynamics*) or changed at each timestep (the *annealed dynamics*). In the majority of the work that has been done on Boolean networks, and in our work, the transfer functions are left unchanged.

The dynamical behavior of the network can be described in terms of its cycles, attractors, and basins of attraction [3]. Assume the transfer functions are fixed. A network of N nodes has 2^N possible states, and the rule for evolving the state values is deterministic. Therefore after a certain number of steps ($\leq 2^N$), the set of state values must start repeating. Each unique cycle of state values into which the network can fall is an attractor, and the set of all initial states that lead to a given cycle is that cycle's basin of attraction.

It has been observed that Boolean networks exhibit three phases, termed quiescent, critical, and chaotic. In this, they show similarity to the percolation problems of statistical physics [3].

In the quiescent phase, there are few cycles, the cycles tend to be short, and most of the state values end up fixed. In the chaotic phase, the average cycle length grows exponentially with N, and most values keep changing. In the critical phase, the cycle lengths and the average number of attractors increase algebraically with N. Kauffman hypothesizes that biological systems operate in the critical phase, "on the edge of chaos" [22].

The phase of the network depends on the number of in-neighbors K, and on the probability with which the transfer functions output true or false values. If the transfer functions are chosen at random from all possible truth tables with 2^K rows, then the network exhibits the quiescent phase for $K < 2$, the critical phase for $K = 2$, and the chaotic phase for $K > 2$. Another way to describe the network is in terms of a collection of the time series of node states, or alternately in terms of their frequency spectra. The state values tend to change rapidly in the chaotic phase, slowly in the quiescent phase, and at an intermediate rate in the critical phase.

In our work, we assign edges randomly in such a way that the graph has a specified average number of in-neighbors per node, but each node need not have the same number of inputs. For example, if the specified average is 2.5, every node would have at least two incoming edges, and approximately half the nodes would have three.

An interesting example of the use of Boolean networks to model a complex system is found in a biochemistry application [9], which used Boolean networks as a coarse-grained approximation of the more detailed differential equation network model of the fission yeast cell cycle control network. The Boolean model abstracted out time information but successfully modeled the reaction kinetics, reproducing the sequence of cycle states and even predicting the effects of mutations.

3.2 Robustness Properties

Because Boolean networks are mathematically tractable yet closely related to real-world systems such as biological regulatory networks and digital circuits, they provide a convenient setting for the study of robustness. From a biological perspective, the problem can be formulated as follows: A specification of a Boolean network (topology and transfer functions) models a genotype. The expression of this genotype is carried out by evolving the Boolean network until it reaches an attractor, which models a phenotype. The biological fitness of this phenotype is evaluated using an objective function, which in the simplest case is itself Boolean (fit or unfit).

For mutation and natural selection to operate efficiently, it is necessary that almost all random changes to the genotype (mutations) result in a phenotype that still satisfies this objective function. This allows mutations to explore the possibility of keeping up with a slow drift in the objective function, or satisfying additional, stricter objective functions, while almost always remaining fit according to the original objective function. Otherwise, if most mutations resulted in an unfit phenotype, then very little promising variation would be available, mutations would not be useful, and species adaptation via natural selection would grind to a halt.

Thus robustness is a necessary condition for biological evolvability, and can serve as a marker

of systems that have an organic character. As discussed in Section 6.2, we have translated the idea of robustness into precise mathematical constraints on the structure of certain Boolean networks and the objective functions they satisfy. This result can then guide the design of models for systems that are known, or desired, to exhibit robustness to failures.

Chapter 4

Renormalization

4.1 Background

The emergent behaviors of a complex system generally become apparent only in a large system and after many timesteps. Indeed, the real-world systems of greatest application value, whether physical, cyber, or social, undergo an enormous number of elementary local interactions in the course of building up their rich global dynamics. The large number of degrees of freedom in these systems implies a severe cost in computational resources for a direct simulation, especially when many simulations are being performed to explore parameter space or to gather statistics. Consequently, almost all practical simulations use some kind of reduced model. Even models accepted as "complete" descriptions of a system are usually well-founded abstractions of deeper processes (e.g., the equations of fluid mechanics summarize the effects of molecular dynamics). For complex systems, the construction of successful reduced models has typically been an ad-hoc process requiring a great deal of insight and experience in the problem domain. One of our goals has been to develop a more systematic and validated approach for creating coarse-grained models of complex systems.

A useful framework for simplifying the description of intractably large systems is available from theoretical physics in the form of "renormalization." This technique seeks to replace the actual system with a model of reduced complexity, lying within some specified class of models. The form of such a reduced model is often taken as a straightforward generalization of the underlying system dynamics, with a number of adjustable parameters included. In the process, many of the original degrees of freedom are omitted through some form of sampling or averaging; those associated with the large-scale phenomena of primary interest are retained. The method is successful when the effect of the omitted degrees of freedom is approximately equivalent to that of substituting new values for the model parameters. These parameters are thus "renormalized" in the reduced description. Renormalization is expected to be most useful when the underlying interactions exhibit locality, so that a block of nearby degrees of freedom behaves approximately as a unit—allowing a summary description of its internal state and higher-level interactions.

This coarse-graining process can be applied iteratively, leading to the "renormalization group" concept, which describes a repeated transformation or "flow" in the model parameter space as degrees of freedom are progressively omitted. At each step, only small blocks of the current degrees of freedom need be considered, and their effective dynamics can be determined more tractably

by taking advantage of the locality of interactions. Thus the renormalization group is a mathematical device for reducing the number of degrees of freedom (zooming out) while preserving some aspects of the dynamics. The original motivation for renormalization-group methods arose in statistical physics, where a large number of interacting degrees of freedom (even with simple interactions, such as those between nearest-neighbor atoms in a crystal lattice) can lead to critical fluctuations on a wide range of length and time scales. Renormalization-group methods have since been applied extensively to phenomena with similar characteristics, as in high-energy particle physics and fluid turbulence. In many such areas, the renormalization group can be treated with considerable rigor, and it can be shown that all but a small number of parameters (describing possible interactions) become negligible upon repeated coarse-graining. Such rigorous justification of reduced models is possible because the systems are defined by highly regular structures, such as functions in Euclidean space.

To determine whether renormalization remains useful in a more general modeling context, we have prototyped representative complex system models that are both tractable and relevant to Sandia mission applications, and quantified the effect of renormalization on the predictive accuracy of these models. The renormalization itself has been be carried out computationally, by simulating the behavior of blocks of degrees of freedom under a variety of conditions and fitting the coarse-grained model parameters that best reproduce this behavior. The stepwise process of the renormalization group makes these model-building simulations tractable because small blocks are used at each stage. We next describe the details of our renormalization method for cellular automata and Boolean networks. Results of our computational tests of this approach are presented in Chapter 6.

4.2 Application to Prototype Models

4.2.1 Cellular Automata

A previous application of the renormalization group to cellular automata [20] considered special cases in which renormalization can be performed rigorously and the resulting coarse model is exact. We wish to generalize renormalization to other kinds of cellular automata for which an empirical, statistical approach is needed. As is usual in the renormalization of lattice systems, we wish to create a model on a coarser Cartesian lattice, each cell of which corresponds to a block of cells in the original lattice. The motivation for this is that the large-scale emergent behavior can be adequately discerned from a coarse history that summarizes the states within each block. The traditional summary description in physics is a block average, but here we use a block sum so that the result remains an integer.

Our approach is to implement the goal of renormalization—finding a coarse model that best reproduces the system's large-scale behavior—as a numerical optimization problem. Such an optimization cannot, of course, explore the space of all conceivable coarse models to find the best one; rather, the investigator must define a limited family of coarse models, with a reasonable number of

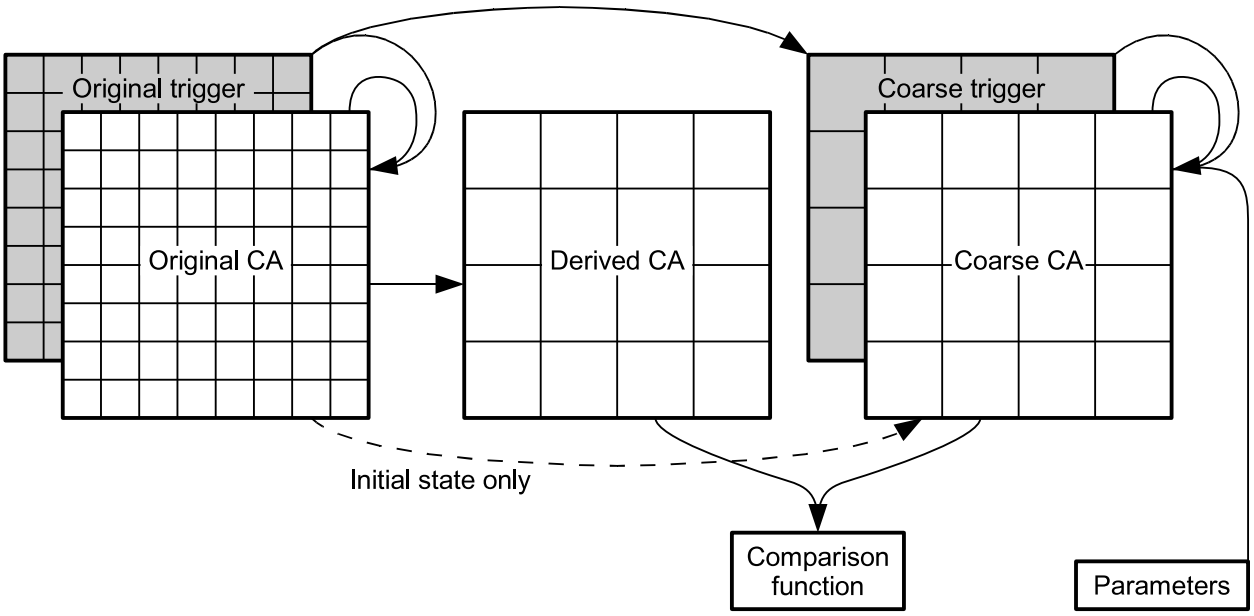


Figure 4.1. Basic scheme for computational renormalization of cellular automata.

free parameters to be optimized. The choice of family may be aided by general principles such as symmetry, or by expert knowledge of the system's behavior. This approach is general and does not require analytic tractability of models. However, it raises the question of how renormalization can make a simulation more computationally feasible if a simulation of the original model is needed for comparison to optimize the coarse model. The answer is that the optimization can be based on a much smaller simulation, in lattice size and in timesteps, than the main simulation for which results are needed.

The computational renormalization process is shown schematically in Figure 4.1. The following items are specified in advance to define a particular renormalization problem:

- the updating rule for the original cellular automaton,
- a parameterized updating rule for the coarse cellular automaton,
- the sizes of the original and coarse lattices to be used for optimization,
- a method of populating the original cellular automaton with random initial states,
- a method of generating trigger values for the original cellular automaton (if applicable),
- a comparison function for quantifying the discrepancy between the coarse predictions and the actual "derived" block sums of the original cellular automaton,
- the number of simulation timesteps to be carried out during optimization, and

- the number of separate simulations, starting from random initial states, to be used in evaluating each candidate coarse model (to average out statistical fluctuations in performance).

Given trial values of the coarse rule parameters, the assigned number of simulations are performed, each consisting of (conceptually) parallel evolution of the original and coarse cellular automata. For each such simulation, random initial states are placed on the original lattice (to favor parameter values that perform well in a broad range of situations), and their block sums are placed on the coarse lattice, so that the coarse cellular automaton starts from an equivalent initial condition. Both cellular automata are then simulated using their respective rules for the assigned number of timesteps; at each timestep, a "derived" lattice of the same size as the coarse one is populated with block sums from the original cellular automaton, representing the true coarse states that should ideally be reproduced. The comparison function accumulates the total discrepancy between the derived and simulated coarse states over all timesteps of all simulations with given parameter values. This discrepancy is then treated as an objective function and minimized with respect to the parameters using a standard numerical optimization algorithm. The parameters are treated as floating-point numbers; although their effect on the time evolution of the coarse model is ultimately discrete, their typically wide range of possible values and the statistical averaging of the simulations result in somewhat smooth overall behavior of the objective function. Nevertheless, gradients are not available and the detailed form of the objective function is not locally smooth, so we use the Nelder–Mead simplex method for optimization.

If a trigger is needed (as for our sandpile model), a lattice is populated at each timestep to provide trigger values for the original cellular automaton, and the block sums are used as trigger values for the coarse cellular automaton. Thus the coarse cellular automaton is being asked to track the large-scale behavior as well as possible with access to a synchronous, appropriately coarse summary of the particular external input values driving the original cellular automaton. This allows the coarse cellular automaton to develop an accurate model of the response to individual perturbations, even if they are stochastic. To reduce the effect of statistical fluctuations on the objective function landscape, the random number generator is initialized with a fixed seed for each new set of trial parameter values. As a result, the series of random initial conditions for the simulations, and any randomness in the trigger values, will be identical for all trial parameter values. Otherwise, independent fluctuations would enter each time the objective function was evaluated, and the landscape would appear extremely jagged, frustrating numerical optimization.

An alternative to optimizing the match of individual simulation histories would be to generate separate ensembles of simulations of the original and coarse cellular automata, and compare only statistical properties of the derived and coarse states, such as their spectra in space or time. This would represent a goal of a different kind of predictivity, favoring coarse models whose behavior "looks right" statistically but need have no correlation with the behavior of the original model in a particular realization. Such an approach is of limited usefulness because matching a chosen set of statistical properties is no guarantee of matching others, and furthermore, accurate individual simulations are often needed for real-world applications. By using linked trigger values for the original and coarse cellular automata and optimizing the match of individual simulation histories, we favor coarse models that reflect the underlying dynamical mechanisms as accurately as possible. To the extent that individual realizations are reproduced correctly, statistical properties will

automatically follow.

In the absence of a perfect coarse model, however, our choice of objective function will have a systematic effect on statistical properties, which is related to the number of simulation timesteps used for optimization. An imperfect coarse model will inevitably lose its correlation with individual histories of the original cellular automaton beyond some timescale, the predictivity horizon. (This concept is illustrated by weather forecasting models, which can give useful predictions only for approximately a two-week period.) If the optimization uses simulations much longer than this timescale, then the coarse model cannot generate any meaningful prediction for most of a simulation and, to reduce its discrepancy, is driven to make a noncommittal prediction that tends to a uniform average state for all cells (analogous to the use of climate averages as the best alternative beyond the horizon of weather forecasts). Thus, the longer the optimization simulations and the more imperfect the coarse model, the more suppressed is the intensity of dynamics in the optimized coarse predictions, resulting in smaller statistical measures of variation compared to those derived directly from the original cellular automaton. On the other hand, if the optimization simulations are very short, such as a single timestep, then other issues arise. The performance being optimized is then tied strongly to the imposed distribution of random initial states, and may not carry over to longer simulations in which the cellular automaton relaxes to its own characteristic distribution and statistical correlations. Also, the coarse model may obtain an excellent match, because only relatively slight changes in block sums over a single timestep need to be predicted, but may not be well optimized to maintain accuracy for as long as possible (which may involve stability properties in addition to single-timestep accuracy). Thus simulations with a duration of the same order as the predictivity horizon are most useful for obtaining optimal coarse models.

For our sandpile model, the details of the renormalization scheme are as follows. The original cellular automaton uses a trigger value of 0 for the normal updating rule (2.2.1) and 1 for the nonupdating perturbation. Each block-sum trigger value for the coarse cellular automaton therefore indicates the number of perturbations in the block; because perturbations are rare, this number is almost always at most 1. The properties of the sandpile model that must carry over to the coarse model are the symmetry among the four directions in the square lattice (isotropy), and the conservation of the sum of states (total amount of sand) in the absence of perturbations. A simple generalization of the rule (2.2.1) that preserves these properties is

$$z(x,y) \leftarrow z(x,y) - 4\,f\big(z(x,y)\big) + f\big(z(x-1,y)\big) + f\big(z(x+1,y)\big) + f\big(z(x,y-1)\big) + f\big(z(x,y+1)\big), \tag{4.2.1}$$

where f is an integer function; the original rule has $f(z) = \lfloor \frac{1}{4} z \rfloor$. Note that $f(z)$ can be interpreted as the number of grains of sand transferred by a block to each of its four neighbors. To give a tractable number of parameters, we take a coarse rule with

$$f(z) = \left\lfloor \frac{z - A_0}{B_0} \right\rfloor, \tag{4.2.2}$$

where A_0 and B_0 are real numbers. Repeated coarse-graining can be performed within this rule family by using an optimized coarse rule as the original rule for a further optimization, thus obtaining a new coarse rule applicable to even larger blocks.

An additive constant in f has no effect on the updating rule, and so addition of an integer

multiple of B_0 to A_0 leaves the rule unchanged. The graph of $f(z)$ for $B_0 \geq 1$ consists of upward steps of $+1$ at approximately equal intervals of z given by B_0. If the range of relevant z values is much greater than B_0, then any change in A_0 should have relatively little effect, merely shifting the phase of these steps. On the other hand, in the original rule, where $0 \leq z \leq 7$, there is only one step, which can be recreated for any $B_0 > \frac{7}{2}$ by appropriate choice of A_0 (e.g., $B_0 = 4$ and $A_0 = 0$). The sandpile model's preservation of a finite state space is due to the nondecreasing property of $f(z) = \lfloor \frac{1}{4} z \rfloor$, which results in the largest values of z never increasing further (and the smallest values never decreasing further). Our generalized coarse rule provides a similar stability when $B_0 > 0$, but we do not impose this as a constraint, instead relying on the objective function to penalize the divergent evolution of an unstable rule.

Whereas the original cellular automaton performs a non-update for a trigger value of 1, we expect that some other strategy is appropriate for the coarse model, because only one of the underlying cells of the block is responding to such a trigger and the others are updating normally. Thus we further generalize the coarse model by performing an update analogous to Eq. (4.2.1) also in the case of positive coarse trigger values (which will almost always be 1), but using different parameters A_1 and B_1 in the function f.

For a coarse model based on 2×2 blocks, we can make an educated guess of reasonable initial parameter values for optimization. In the normal updating case, the number of grains of sand transferred to all block neighbors is 8 if all underlying states are between 4 and 7 (typical block sum $z = 22$), is 4 if two states are between 0 and 3 and two are between 4 and 7 (typical block sum $z = 14$), and is 0 if all states are between 0 and 3 (typical block sum $z = 6$). Thus $f(z)$, the number of grains transferred to each block neighbor, should equal 2 for z near 22, equal 1 for z near 14, and equal 0 for z near 6. A similar argument for the triggered case, where only three underlying cells update normally, shows that in that case $f(z)$ should equal 1 for z near 17 (where on average $\frac{2}{3}$ of the states are between 4 and 7 and $\frac{1}{3}$ are between 0 and 3) and equal 0 for z near 6 (where all states are between 0 and 3). Example parameter values that fit these conditions are

$$(A_0, B_0, A_1, B_1) = (2, 8, 0.67, 10.67). \tag{4.2.3}$$

Such values are not necessarily optimal because the typical block sums quoted for various patterns of underlying cells are also consistent with different patterns, reflecting the loss of information in coarse-graining.

4.2.2 Boolean Networks

The renormalization of Boolean networks follows most of the same principles as for cellular automata and, as discussed in Chapter 5, is performed with the same software framework. However, some differences of interpretation apply. A trigger is not needed because the Boolean networks commonly studied have deterministic time evolution. Since Boolean networks usually associate a different updating rule with each node, we take the specific rule to be encoded in part of a node's state value—a part that does not change with updating (because we use the quenched dynamics), but determines the updating of the Boolean bit in the state. A type of Boolean network is not considered to be a specific assignment of updating rules to nodes, but rather an *ensemble* of such rule

assignments, as part of the generation of random initial states. Correspondingly, a renormalized version of a Boolean network consists not of a specific set of coarse rules, but rather of a *method* that transforms specific rules on the original network into specific rules on the coarse network. It is this method that contains parameters to be optimized. Because the specific rules are part of the state values, this method operates at the point of populating the initial states of the coarse network. Unlike cellular automata, whose states are coarsened simply with a block sum, Boolean networks require coarsening the specific rules at the same time (this applies only when setting initial conditions, since the coarse rules are automatically preserved thereafter). The actual updating rule, which plays the same role as a cellular automaton updating function, is a rule fixed once and for all, with no parameters. This master rule interprets the specific rule part of a node's state, and evaluates the appropriate function of the Boolean bits of its in-neighbors to replace the given node's Boolean bit. Because a Boolean rule for a node with K in-neighbors requires 2^K bits to specify, and we must carry an additional Boolean bit, standard 32-bit integers can store the required state values provided $K \leq 4$.

Whereas lattice systems offer an obvious state-coarsening approach based on regular blocks, the arbitrary connectivity of Boolean networks makes it unclear what groups of nodes should be represented by the coarse network's nodes. We expect that renormalization is most effective when the node groups are highly connected internally and sparsely connected externally, so that they act as coherent entities. Thus we are faced with a graph clustering problem; we have adopted an implementation of a clustering algorithm based on the statistical physics of spin glasses [28]. This algorithm is used for obtaining the coarse network structure, which is held fixed through the parameter optimization process. We must then relate the states of the coarse network to those of the original network. In principle, cluster sums could be used for coarsening the Boolean bits, leading to a coarse model with integer states that is not a Boolean network. But for simplicity we have maintained the Boolean nature of the network when coarsening. Moreover, instead of using the majority vote of all nodes in the cluster to determine the coarse Boolean bit, we have made a different choice that may give better results. Because only the nodes with out-links from a cluster can affect other clusters at the next timestep, we consider the majority Boolean bit of these out-facing nodes to be a more useful summary of the state of the cluster. This distinction does not arise for coarsening of cellular automata into 2×2 blocks because all cells are automatically out-facing.

As mentioned, a method is also required for coarsening specific Boolean rules. For this we adopt a simple strategy. To tabulate the coarse updating function for a cluster, we "excise" the cluster from the original network and evolve it independently starting from the given initial states, but supplying its external in-links with random Boolean bits at each timestep. The key parameter of the method is the number of timesteps for which these random inputs are held fixed before being regenerated. Because the random inputs are a proxy for the behavior of the cluster's environment, the performance of the resulting coarse rule in a whole-network simulation will depend on how well this proxy reflects the typical dynamics of the inputs to a cluster. The optimization will have the opportunity to tune the timescale parameter to the most effective value. Every step in the evolution of the excised cluster contributes to a tabulation of the coarse rule: Each coarse input bit (arriving on a coarse in-link from another cluster) is taken as the majority of its most recent underlying random inputs, thus identifying a row in the truth table, and then the out-facing nodes of the cluster "vote" on what the corresponding result of the coarse Boolean rule should be. The

excised evolution is continued until the truth table accumulates a threshold number of votes for either a 0 or 1 result in every row, and then the majorities determine the final coarse rule. This vote threshold is also an adjustable parameter of the method, but because it mainly governs the trade-off between statistical accuracy and execution time, we assign it a fixed value rather than subjecting it to optimization.

The optimization process is carried out as for cellular automata. The various simulations performed for a given parameter value start from various initial states and thus use different specific rules from the assigned ensemble. For each simulation, after the initial states (including rules) are coarsened, the original and coarse networks are evolved for the given number of timesteps. After each timestep, the coarsening of the original network's states onto the "derived" network (for comparison with the coarse network) applies only to the Boolean bits, because there is no point to re-coarsening the rules. Correspondingly, the comparison function ignores the rules in the coarse network and simply compares the Boolean bits with those of the derived network. The baseline discrepancy measure for a unhelpful coarsening corresponds to predicting half of all bits incorrectly, and discrepancies less than this indicate some success in tracking the original network's behavior. Although the optimization for Boolean networks is currently one-dimensional and thus a variety of other numerical optimization algorithms are available, for simplicity and consistency we continue to use the Nelder–Mead simplex method.

Chapter 5

Software

5.1 General Description

We wish to experiment with complex systems expressed as networks. A network has a topology given as a directed graph. Once established, a graph's topology remains fixed. Each node in the graph contains a state value, and we wish to investigate the evolution of the state values in time. There is an edge from node A to node B in the graph if node A's value can influence the evolution of B's value. Each network type is characterized by a particular evolution rule, by which the new value at a node is computed taking into account the values at each of the node's in-neighbors and perhaps some additional characteristics of the node. For all the networks investigated so far, we update all state values in parallel. Our particular interest lies in deriving networks that are renormalized or coarsened approximations of an original network. The coarsened version of a network is in general parameterized, and we optimize over the values of the parameters to improve the approximation.

Our code is written in C ($\sim$ 2000 lines) but structured so as to give some of the advantages of object orientation. All networks, no matter what their topologies or updating rules, are covered by one type, CX_network, and all updating rules are of one function type, CX_update. A particular advantage of this approach is that we can construct a single objective function, to be called by the optimizer, that works for all networks.

The network data structure is laid out to facilitate this simulated polymorphism. All network state data occupies a one-dimensional array. Any further structure is expressed in the configuration data, which is given merely as a void pointer in the data structure. Likewise, parameters used for the updating rule are given in an array of type CX_parameter (defined as a double), with the length of the array and the meaning of its elements left undetermined. The network data structure also contains a function pointer to the function that returns the list of neighbors of a node, to be filled in with a pointer to a function that takes the particular network topology into account.

In order to introduce a new network type, it is necessary to provide a way to create the network, an update rule for the network, a neighbor list function for the network, a way to derive a renormalized version of the network, and, since storage management is done "by hand" in C, a way to release the network's storage. It may be necessary to provide a separate updating rule for the coarsened network, in the case that the coarse updating rule has optimizable parameters.

We currently support the creation of two network types, the periodic Cartesian lattice and the Kauffman network (a type of Boolean network). The periodic Cartesian lattice may have any number of dimensions with any number of nodes in each dimension; a d-dimensional periodic lattice forms a torus in d dimensions, in which each node has $2d$ neighbors. The Kauffman network, sometimes called an N-K network, has a graph in which each of the N nodes has K in-neighbors. We assign the links randomly.

For the periodic Cartesian network, network creation is handled by the function CX_periodic_cartesian_create in module periodic_cartesian.c and network destruction is handled by the generic CX_network_destroy in network.c. The neighbor list function is CX_periodic_cartesian_neighbor_list in module periodic_cartesian.c. The sandpile update rule, CX_sandpile_update in module sandpile.c, can be used for periodic Cartesian networks. A separate updating rule, CX_sandpile_coarse_update, can be used for a renormalized periodic Cartesian network. The update function uses the state value at the node itself as well as the values at the neighboring nodes. The renormalized network states (block sums) are derived with function CX_periodic_cartesian_derive in module periodic_cartesian.c.

The igraph package [8], a high-quality software package originating in the Hungarian Academy of Sciences for creating and manipulating graphs and released under the GNU Public License, is used for the topology component of a Kauffman network. Creation of a Kauffman network requires two steps. CX_kauffman_create in module kauffman.c creates the N-K topology and returns an igraph_t graph structure. CX_graph_create in module graph.c accepts the igraph structure and returns a CX_network. CX_graph_create installs the igraph_t as its configuration data, allocates the state data array, and installs the proper neighbor list function, CX_graph_neighbor_list in module graph.c. Doing the creation in two steps will make it easy to substitute other graph topologies (such as ones constructed directly by igraph) if this becomes desirable. The update function for Kauffman networks is CX_boolean_update in module boolean.c. In a Boolean network, in addition to a state value (a true/false value), each node has a transfer function that is a Boolean function of the inputs to the node. After an update, the state value at the node is the output of the node's transfer function applied to the node's inputs. In our implementation, the transfer function is encoded in the state data; thus the state data consists of two parts, the state value proper and the encoded transfer function. Renormalization of Kauffman networks is a three-step process. The topology coarsening is done by function CX_graph_spinglass cluster in module graph.c, which returns a graph_t structure. Then CX_graph_create in module graph.c is called to return a CX_network structure with the given topology. Finally, CX_boolean_derive in module boolean.c is called to derive the state values of the coarsened network from the original network. If CX_boolean_derive is called with no parameters, it derives only the 1-bit Boolean value at each node. If it is provided with parameters, it also derives the transfer function at each node (see below for details).

To evaluate the performance of a coarsened network, CX_objective in module objective.c returns the total discrepancy, over a specified number of simulations, between the predicted coarse states and those derived from the original network. In the process, CX_objective also accumulates frequency spectra of node histories and/or spatial spectra of snapshots of the network (the latter meaningful only for a Cartesian lattice). Spectra are computed using fast Fourier transform routines from the GNU Scientific Library [13]; our implementation assumes that the relevant sizes in

space and time are powers of 2. The discrepancy is used as the objective function by CX_optimize in module optimize.c, which calls the Nelder–Mead simplex method for numerical minimization, also provided by the GNU Scientific Library.

5.2 Description of the Individual Modules

The following gives a brief description of the functions in each module:

boolean.c

- CX_state CX_boolean_update(CX_state **list, CX_parameter *p, CX_state trigger);

  Updates a node of a Boolean network, using the state values of a node's in-neighbors and the node's Boolean transfer function to determine the node's new value.

- double CX_boolean_discrepancy(CX_network *net1, CX_network *net2);

  Computes the distance between two Boolean networks with the same layout by counting the number of nodes whose Boolean state bits differ (ignoring the transfer functions that may be encoded in the state data).

- void CX_boolean_derive(CX_network *orig, CX_network *derived, CX_parameter *p);

  Derives the state values of a coarse (derived) network from those of a fine (original) network. The state values have two components, a Boolean transfer function and and a Boolean value. If the parameter argument is null, only the Boolean value is derived. Otherwise, the coarse transfer function is also computed by simulating the evolution of each coarse node (cluster). The fine nodes making up a coarse node are provided with random inputs on the incoming edges from outside the cluster and updated a number of times, noting the count of true and false values in the out-facing nodes after each update step. Using the correspondence between coarse and fine edges, each set of random inputs is interpreted as a set of coarse inputs, corresponding to a row in a truth table for the coarse node. When enough true and false counts have been collected in each row of the truth table, the coarse transfer function for the node is determined. There are two parameters in the parameter array: the number of steps to evolve the fine nodes within a coarse node between assignments of random inputs, and the minimum number of true or false values needed in each row of the coarse node's truth table.

compare.c

- typedef void (*CX_random)(CX_network *net, CX_parameter *p);
  typedef void (*CX_derive)(CX_network *orig, CX_network *derived, CX_parameter *p);
  typedef double (*CX_discrepancy)(CX_network *net1, CX_network *net2);

Function types for functions called by the compare function in order, respectively, to assign random values to a network, derive coarsened states for a network, and compute the difference (or distance) between the states of two networks.

- struct CX_compare_closure;
  double CX_compare(CX_compare_closure *c);

  Using items from the closure data structure, updates the fine network according to the fine update rule, updates the coarse network according to the coarse update rule, derives a coarsened network from the fine network, and compares the coarse network to the derived network using the discrepancy function. The discrepancy is accumulated for an assigned number of timesteps. If the closure contains a non-null sampler function, CX_compare uses it to populate the fine trigger network, which it then coarsens to produce a coarse trigger network (used in cases where an update function requires a trigger argument—see below). If instructed, CX_compare also records node histories, and accumulates spatial spectra of snapshots, for the coarse and derived networks.

graph.c

- CX_network *CX_graph_create(igraph_t *graph);

  Given a graph structure (as created, e.g., by CX_kauffman_create), creates a network having the topology of the graph.

- CX_state **CX_graph_neighbor_list(CX_network *net, unsigned node);

  Returns the list of in-neighbors of a given node of the network (where the given node is considered to be the first "neighbor" in the list.)

- igraph_t *CX_graph_spinglass_cluster(igraph_t *orig, CX_parameter *param);

  Coarsens a given graph structure using the igraph_community_spinglass function of the igraph package. The parameter array contains three parameters: a suggested number of nodes of the original graph per node of the coarsened graph; an edge threshold giving the minimum number of (directed) edges of the original graph needed to justify creation of an edge of the coarsened graph; and an edge limit, the maximum number of (incoming) edges that any node of the coarsened graph may have.

gsl.c

- struct CX_objective_gsl_closure;
  double CX_objective_gsl(const gsl_vector *x, void *v);

  Function called by the GNU Scientific Library optimization routine. It serves as a wrapper for CX_objective function, which it calls.

kauffman.c

- igraph_t *CX_kauffman_create(unsigned n, double k);

  Creates and returns a graph of n nodes, each with an average of k in-neighbors drawn independently and uniformly from the n nodes. Each node has a probability $k - \lfloor k \rfloor$ to have $\lfloor k \rfloor + 1$ in-neighbors instead of $\lfloor k \rfloor$.

- igraph_t *CX_kauffman_destroy(igraph_t *graph);

  Frees the memory used for a Kauffman graph.

network.c

All networks, no matter what their topology, are accommodated by the same data structure having the same components: an array of state values (of type CX_state), an undefined (void *) data structure defining the connectivity, an array of parameters that may be used for the updating rule (of type CX_parameter), and a function pointer to the updating rule to be used for the network.

- CX_network *CX_network_destroy(CX_network *net);

  Frees the memory used by the network (for state data array and configuration data).

- double CX_network_discrepancy(CX_network *net1, CX_network *net2);

  Computes the distance between two networks with the same layout using a node-by-node sum of squares of the differences between the state values.

objective.c

- struct CX_objective_closure;
  double CX_objective(CX_parameter *p, CX_objective_closure *c);

  The objective function used for optimization of the coarse network parameters. The first argument contains the parameters being optimized. Using data and functions specified in the closure argument, the function populates a fine network, coarsens it using the optimization parameters, and computes the discrepancy by calling CX_compare with the CX_compare_closure element of the CX_objective_closure (see module compare.c, above). CX_objective does this for a specified number of simulations and returns the cumulative discrepancy. Also, CX_objective accumulates frequency spectra of the coarse and derived node histories recorded by CX_compare.

optimize.c

- double CX_optimize (CX_parameter *param, unsigned param_count, CX_objective_gsl_closure *c);

  Calls the GNU Scientific Library's Nelder–Mead simplex method to minimize CX_objective, and returns the minimum discrepancy value. The parameter array is used to input the starting values and to output the optimized vales. Collection of spectra, if requested, is suppressed during optimization and performed only when converged parameter values are reached.

periodic_cartesian.c

- CX_network *CX_periodic_cartesian_create(unsigned dim, unsigned *size);

  Creates a network with a periodic Cartesian lattice topology.

- CX_state **CX_periodic_cartesian_neighbor_list(CX_network *net, unsigned node);

  Returns a list of a given node's neighbors in a periodic Cartesian lattice (in which the node itself is considered to be the first "neighbor").

- void CX_periodic_cartesian_derive(CX_network *orig, CX_network *derived, CX_parameter *p);

  Derives coarsened states for a periodic Cartesian lattice network. Each node of the derived network represents a block of nodes of the original network; its state value is the sum of the values of the original nodes in the block. The parameter array provided is transferred to the derived network.

- void CX_periodic_cartesian_display(CX_network *net, char *title);

  Prints the state values of a periodic Cartesian network.

random.c

- void CX_uniform_random(CX_network *net, CX_parameter *p);

  Initializes the state values of a network to values chosen uniformly at random from a range specified by the parameters.

- void CX_boolean_random(CX_network *net, CX_parameter *p);

  Initializes the state values of a network to randomly selected Boolean values, with the probability of 0 given by the parameter.

sandpile.c

- CX_state CX_sandpile_update(CX_state **list, CX_parameter *p, CX_state trigger);

  Updates a network node according to the sandpile rule.

- CX_state CX_sandpile_coarse_update(CX_state **list, CX_parameter *p, CX_state trigger);

  The coarse version of the sandpile rule suitable for a coarsened network. Uses four parameters whose values are subject to optimization.

update.c

- typedef CX_state (*CX_update)(CX_state **, CX_parameter *, CX_state trigger); CX_network *CX_network_update(CX_network *net, CX_update rule, CX_network *trigger);

  The function to call to update any kind of network. The caller must provide the updating rule (a function pointer); a particular rule may also require a trigger network of the same layout whose state values are used in determining how to update each individual node.

Chapter 6

Results

6.1 Computational Analysis of Renormalization

6.1.1 Cellular Automata

We performed computational experiments on the renormalization of the sandpile model described in Section 2.2, using the approach described in Section 4.2.1. We performed six stages of renormalization, each coarsening a 16×16 lattice to an 8×8 lattice composed of 2×2 blocks. The lattice sizes were chosen small enough to allow efficient simulation in the optimization process but large enough to make finite-size effects reasonably unimportant. In the first stage, we took our sandpile model as the original cellular automaton and optimized the four parameters of the coarse sandpile rule starting from the trial values (4.2.3). In each of the remaining stages, the coarse sandpile rule with the previously optimized parameter values was taken as the original cellular automaton, and starting from these values, new parameter values were optimized for a further 2×2 coarsening. The probability of a trigger value of 1 for each cell of the 16×16 lattice was always taken as 0.01, so that two or three perturbations occurred in the entire system on a typical timestep. The discrepancy of derived and coarse states was computed as the sum of squared differences. All objective function evaluations used 100 simulations from random initial conditions for statistical averaging. Because the range of typical state values changes with repeated coarsening, we used a different distribution of random initial states for each stage. Starting with the range of 0 to 7 for the sandpile model, we increased the midpoint of the range by a factor of the cumulative number of original cells coarsened into the current blocks (4, 4^2, 4^3, ...), and increased the width of the range by the square root of this factor. In this way, we estimated the typical range that would result from block sums of approximately independent cell states.

As discussed in Section 4.2.1, the number of timesteps used for the optimization simulations affects the behavior of the resulting coarse model. We measured each coarse model's performance by converting the objective function value into an *error ratio*, the ratio of its discrepancy to the discrepancy of a naïve model that predicts, for each timestep, a uniform average state value for all cells. For very short simulations, this ratio is small, because even keeping the coarse initial state unchanged is a much better prediction than a uniform state. For very long simulations, beyond the predictivity horizon, this ratio is approximately 1, because the optimal coarse model is close to the naïve one. Since the successive coarsenings of the sandpile model represent larger and

Table 6.1. Optimized parameters for six renormalization stages of the sandpile model.

Block size	Timesteps	A_0	B_0	A_1	B_1	Error ratio
2×2	3	1.86	8.26	0.88	10.88	0.58
4×4	8	−4.09	18.05	−2.97	12.71	0.25
8×8	22	−5.21	22.56	−1.13	12.95	0.35
16×16	61	−3.03	24.66	−2.40	12.34	0.59
32×32	170	31.76	44.39	−21.59	11.46	0.52
64×64	475	36.86	91.57	−144.13	81.09	0.38

larger blocks of the original cells, they have a longer and longer predictivity horizon. This is simply because it takes a long time for significant amounts of sand to be transferred between very large blocks through the nearest-neighbor interactions of the basic sandpile model. To optimize the coarse models over a duration of order their predictivity horizon, we adjusted the growth of the simulation durations with repeated coarsening until the optimized models maintained an error ratio of order 0.5. We found that each coarsening into 2×2 blocks increased the predictivity horizon by a factor of approximately 2.8. As a result, the greatest execution time was spent in the last stage of renormalization. The results of the renormalization stages are shown in Table 6.1.

We then used the coarse rule from the second stage, representing 4×4 blocks, to simulate a much larger sandpile model on a 256×256 lattice with a coarse 64×64 lattice. The probability of a trigger value of 1 for each cell of the 256×256 lattice was taken as 4×10^{-5}, so that two or three perturbations occurred in the entire system on a typical timestep. We used only one simulation, with 10^6 timesteps, to evaluate the objective function; in this way we examined the very-long-term behavior of the models. The error ratio was 1.17, i.e., the optimized coarse model had a greater discrepancy than a naïve coarse model, as expected because we were performing a very long simulation with a coarse model that was optimized for times within the predictivity horizon. The coarse cellular automaton maintained vigorous dynamics even when it ultimately become uncorrelated with the original cellular automaton. As a result, although the accuracy of individual realizations was lost, we found that some statistical properties were well reproduced. Figure 6.1 shows the spatial power spectra of the cellular automaton states, averaged over 1000 snapshots throughout the simulation. The spectrum of the predicted coarse states (green points) has the same power-law slope, approximately -1.8, as that of the derived states from the underlying sandpile model (black points). For comparison, we also performed an equally long simulation of the basic (non-renormalized) sandpile model on the coarse 64×64 lattice and plotted its spectrum (red points). The large intensity of this spectrum at the smallest wavelengths (upper right part of the plot) reflects the tendency of the sandpile model to form "checkerboard" patterns of alternating high and low values on the lattice. Only at larger scales (left part of the plot) is the power-law spectrum seen from the basic sandpile model. Our optimized coarse model, however, correctly reflects the result of coarsening with 4×4 blocks, washing out the checkerboard patterns and exhibiting the power-law spectrum in the same range of wavelengths as the spectrum calculated from the sandpile model on the underlying 256×256 lattice. The lower amplitude of the coarse model's spectrum reflects its imperfect predictivity, which results in favoring somewhat more cautious predictions during the

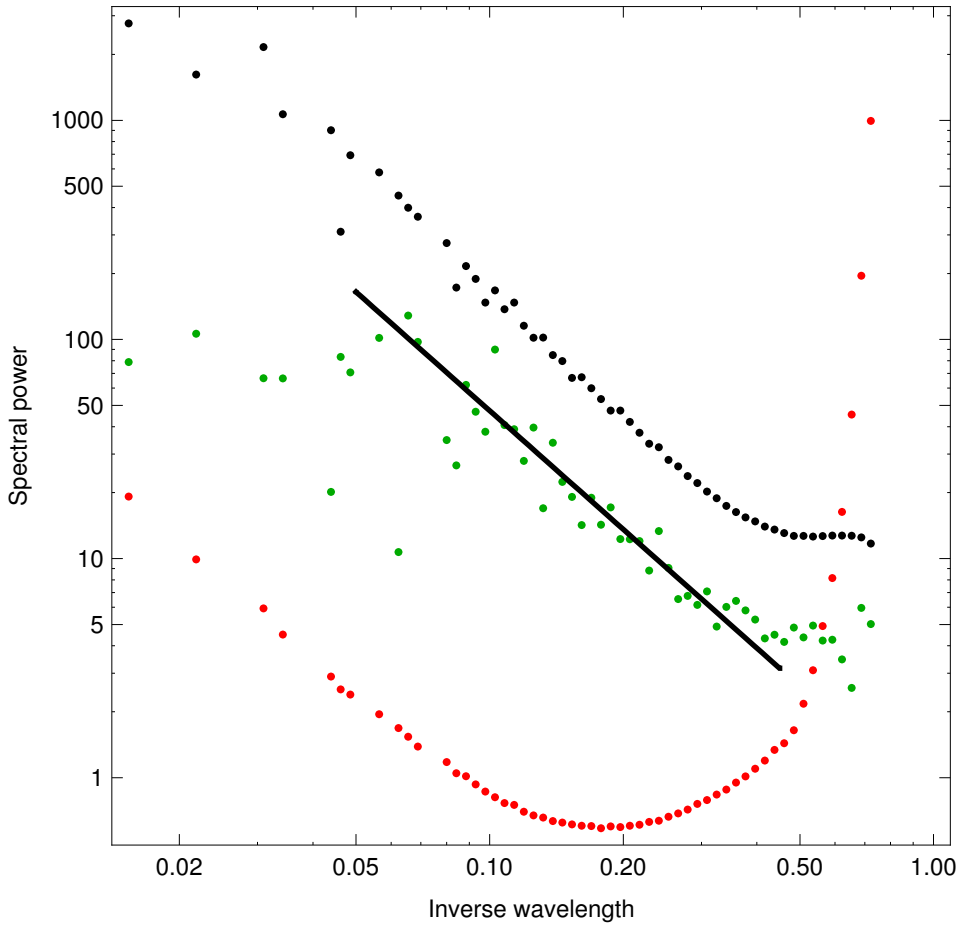


Figure 6.1. Spatial power spectra of the sandpile model and its coarsening. The horizontal scale measures inverse wavelength in units of the coarse 64×64 lattice spacing. Black points: Spectrum of the coarse states derived directly from the sandpile model on the 256×256 lattice. Green points: Spectrum of the coarse states predicted by the renormalized sandpile model on the 64×64 lattice (with straight line estimating power law). Red points: Spectrum for the non-renormalized sandpile model on the 64×64 lattice.

Table 6.2. Optimized parameters for renormalization of five Kauffman networks.

K	Clusters	Timescale	Error ratio
1	82	12	0.11
1.5	87	7	0.40
2	87	2	0.64
2.5	89	1	0.74
3	96	4	0.82

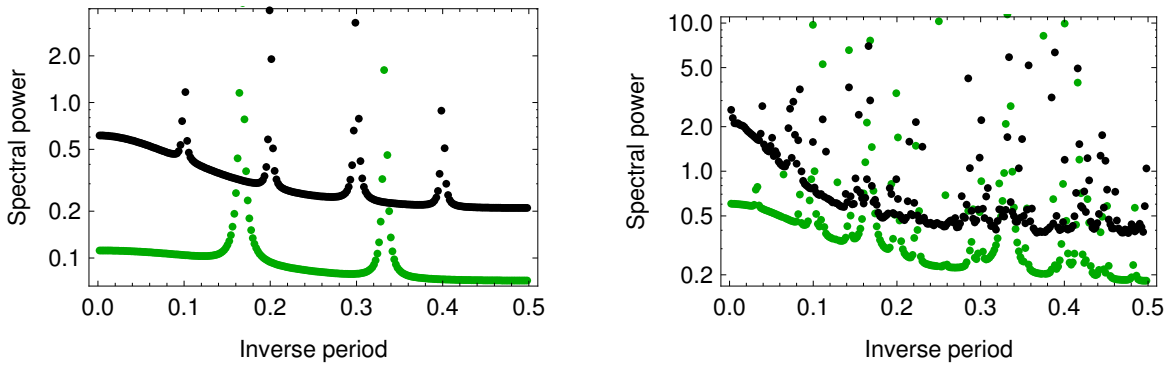
optimization. The power-law spectra observed here demonstrate the claim in Section 2.2.2 that our sandpile model, unlike the original BTW model, exhibits criticality in the conventional physics sense.

6.1.2 Boolean Networks

We also performed computational experiments on the renormalization of Kauffman networks, using the approach described in Section 4.2.2. We created five *N*-*K* network topologies with $N = 500$ nodes and $K = 1$, 1.5, 2, 2.5, and 3 average in-neighbors per node. As is common in studies of Kauffman networks, the specific Boolean rules were drawn uniformly from all possible truth tables and the initial Boolean bit values were drawn uniformly from $\{0, 1\}$. Under these conditions, as described in Chapter 3, Kauffman networks are known to exhibit quiescent, critical, and chaotic phases for *K* less than, equal to, and greater than 2. We wished to determine whether our renormalization method can automatically preserve these differences in emergent behavior upon coarsening the network.

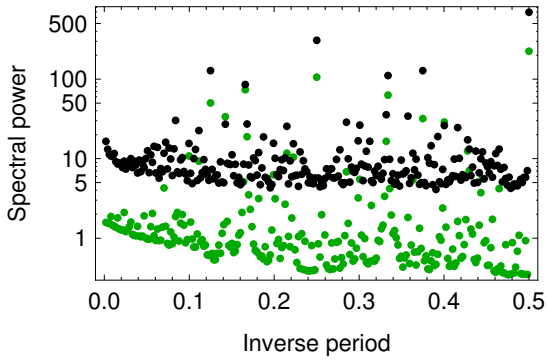
The topological coarsening was performed by instructing the graph clustering algorithm to find as close to 100 clusters as possible but no more; thus approximately 5 nodes were used for each cluster. To give the coarse network the greatest flexibility to produce various emergent behaviors, links were placed in the coarse network whenever any link existed between clusters in the original network, up to a limit of 4 coarse in-links per cluster (with preference for clusters between which multiple links existed in the original network). The resulting coarse Boolean network is not in general a Kauffman network, because the in-neighbors of each coarse node need not be similar in number or randomly distributed. Furthermore, the coarse network's rules were not randomly chosen but determined by the renormalization process of Section 4.2.2 from a given set of rules for the Kauffman network. In this process, the vote threshold was set to 400. We performed optimizations to determine the best values of the timescale parameter (starting with a trial value of 2), and measured the performance of the coarsening of each Kauffman network with 300 simulations over 512 timesteps. Each simulation used a different set of random initial states (rules and Boolean bits), which were coarsened as described in Section 4.2.2. A coarse model's error ratio was computed as the ratio of its discrepancy to that of a model predicting half of all bits incorrectly.

The results of the renormalization of each Kauffman network are shown in Table 6.2. As

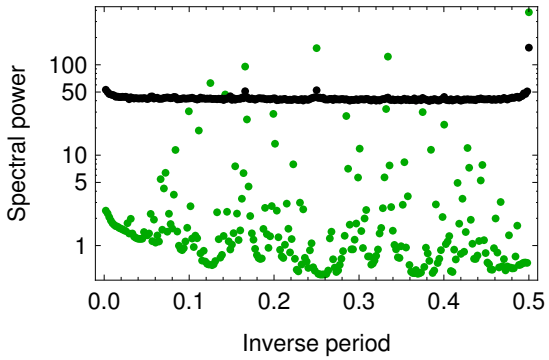


(a) $K = 1$

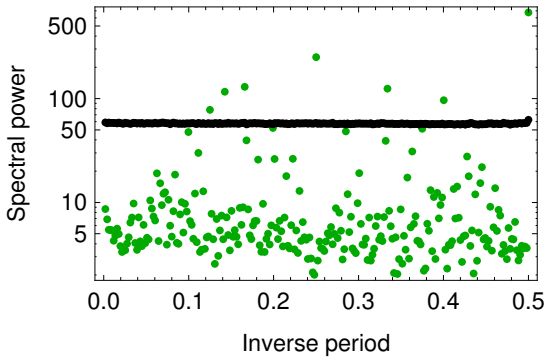
(b) $K = 1.5$



(c) $K = 2$



(d) $K = 2.5$



(e) $K = 3$

Figure 6.2. Frequency power spectra of five Kauffman networks ($N = 500$ nodes) and their coarsenings (~ 100 nodes). Note semilog scales. The horizontal scale measures inverse period in units of the simulation timestep. Black points: Spectrum of the coarse states derived directly from the Kauffman network. Green points: Spectrum of the coarse states predicted by the renormalized Boolean network.

K increases, the optimal timescale parameter mostly decreases, indicating that the evolution of the network becomes more rapid. Also, as K increases, the error ratio increases, indicating that an accurate prediction of the evolution over 512 timesteps becomes more difficult. Both trends are as expected in the transition from the quiescent to the chaotic phase. As a diagnostic to test whether the different phases can be distinguished in the emergent behavior of the coarse model, we collected time series of the Boolean bits of each coarse node and computed their frequency power spectra, averaged over all nodes in each coarse network. Figure 6.2 compares the spectrum of the derived Boolean bits from the underlying Kauffman network (black points) with that of the Boolean bits predicted by the coarse model (green points), for each value of K. The lower amplitude of the green spectra, as for the sandpile model, reflects the imperfect predictivity of the coarse models. The black spectra show that the quiescent, critical, and chaotic phases of the Kauffman network have distinct signatures—a few prominent cycles, a proliferation of cycles, and white noise, respectively. These spectra, moreover, support the concept that the transition between phases is gradual rather than sharp, when a continuum of K values is considered. The green spectra show similar distinctions between phases, indicating that the renormalization process, despite not making direct use of any information about the expected properties of Kauffman networks, has preserved important emergent behaviors.

6.2 Functions Robustly Expressible by Boolean Networks

6.2.1 Introduction

The genotype of an organism is the hereditary information contained in the genome, while the phenotype is the set of properties actually exhibited by the organism and acted upon by natural selection. This section is part of an effort to understand why the distinction between the genotype and the phenotype is advantageous at all. The answer commonly proposed is that the genotype-phenotype distinction promotes *evolvability*, that is, the ability to acquire novel phenotypes through genetic perturbations [33]. The argument is that Darwinian evolution is aided if the structure of the genotype ensures that random genotypic mutations vary the phenotype but keep it viable. Such robustness might not be possible if the phenotype were mutated directly and hence the genotype-phenotype distinction.

Formally, suppose that the phenotype is specified by n Boolean characters $x_1, \ldots, x_n \in \{\pm 1\}$ and that the phenotype is viable exactly when $f(x_1, \ldots, x_n) = 1$ where $f : \{\pm 1\}^n \to \{\pm 1\}$ is a constraint determined by the environment. The optimally fit phenotype is, of course, viable and so contained in the solution set of f (i.e., the set of inputs that evaluate to 1). The genotype is some encoding of $x_1, \ldots, x_n$. The evolvability argument suggests that the genotype encoding should be designed in such a way that random mutations to it keep the corresponding phenotype viable, that is, in the solution set of f, with high probability. Otherwise, since natural selection eliminates unviable phenotypes, evolution toward the optimally fit phenotype would be highly unlikely or, alternatively, require an unreasonably large population. Thus, robustness of the phenotype toward random genotypic mutations is a necessary condition for evolvability.

We have yet to describe how the genotype encodes the phenotype. In nature, this encoding takes the form of a regulatory network. The expression level of each gene is functionally related to the expression levels of some other genes. Thus, if the expression level of one gene is changed, expression levels of other genes are also modified according to the regulatory connections. Quantizing gene expression to only two levels, ON and OFF, we arrive at the notion of a Boolean network, introduced by Kauffman [21, 22, 23]. Formally, a *Boolean network on n variables* is specified by a directed graph on the node set $\{1,\ldots,n\}$ where each node i has K_i incoming edges from other nodes and carries a *state* $x_i \in \{\pm 1\}$ as well as an *update function* $u_i : \{\pm 1\}^{K_i} \to \{\pm 1\}$. The states of the nodes dynamically change in the following way: if at time t, the node states are $\{x_1(t),\cdots,x_n(t)\}$, then at time $t+1$, for each[1] $i \in [n]$,

$$x_i(t+1) = u_i(x_{i_1}(t),\cdots,x_{i_{K_i}}(t)) \tag{6.2.1}$$

where $i_1,\ldots,i_{K_i}$ are the K_i nodes with outgoing edges that end at i. The states at $t=0$ are specified in advance. The set of states of all the nodes in the network at a given time t is said to be the *configuration* at time t. Since the configuration space is finite and the dynamics of the network are deterministic, the network will eventually fall into a previously visited configuration, after which the configuration dynamics becomes periodic. This cyclic trajectory is called an *attractor*. The set of attractors is believed [1, 25, 12, 19, 18, 15] to correspond to the set of phenotypes expressible by the genotype that is represented by the Boolean network.

In this section, we focus our attention on Boolean networks where all the attractors are cycles of length 1, that is, fixed points in the configuration space. The reasons for this restriction are twofold. First of all, in many models for regulatory systems actually found in nature, such as the model for cell determination during flower development analyzed in [12], the attractors reached are always fixed points instead of limit cycles. Secondly, the special case of networks with only fixed points as attractors is easier to specify and analyze technically, and so, is a first step toward an understanding of the more general case. For these reasons, we restrict our attention here to Boolean networks specified by directed acyclic graphs. Henceforth, we assume Boolean networks to be acyclic without comment.

Mutations act on the genotype, or the Boolean network in our model. Mutations could modify the network in various ways, such as changing the adjacency relations (as in [6]), changing the update functions (as in [32]), and duplicating or deleting nodes (as in [2]). In this section, we investigate the case when a mutation on the genotype arbitrarily modifies the update functions of nodes.

Definition 6.2.1. *Given a Boolean network N and a parameter $\varepsilon \in (0,1)$, an* ε-mutation *of N is a random variable denoting a Boolean network N' with the same node set and adjacency relations as N but for which, if u_i and v_i denote the update functions for node i in networks N and N' respectively, then for each node i, $v_i = u_i$ with probability $1-\varepsilon$ and $v_i = -u_i$ with probability ε.*

Certainly, mutation to update functions in a regulatory network is one of the most common types of mutations in nature. Such mutations to the Boolean network can either be genetic or even

[1] Throughout, $[n] \stackrel{\text{def}}{=} \{1,\ldots,n\}$.

be due to environmental effects. Imagine some physical event which probabilistically affects the transmission of information from one node to the other in the network. These random failures on the edges can be equivalently modelled as failures of the update functions at the nodes. Therefore, understanding the effect of update function mutations on the expressed phenotypes is highly relevant.

Note that in acyclic Boolean networks, the final configuration of the network is independent of its initial state. For a string $x \in \{\pm 1\}^n$, we say that a Boolean network N *expresses* x if for all $i \in [n]$, x_i is the state of node i in the final configuration. Now, we can formalize the notion that for an evolvable system, the phenotype needs to be viable with high probability even when the genotype undergoes random mutation.

Definition 6.2.2. *For $\varepsilon \in (0,1)$, a Boolean function $f : \{\pm 1\}^n \to \{\pm 1\}$ is said to be* ε-robustly expressible *if there exists a Boolean network N with nodes $\{1,\ldots,n\}$ whose states correspond to the n arguments of f such that, if $x = (x_1,\ldots,x_n)$ is the configuration expressed by an ε-mutation of N, then $f(x) = 1$ with probability at least $1 - o_n(1)$. f is said to be* robustly expressible *if it is ε-robustly expressible for some constant $\varepsilon \in (0,1)$.*

The primary goal of this work is to advance our understanding of robustly expressible Boolean functions. This is needed for evolvable, adaptable systems to exist in the first place. Said differently, the space of possibilities for self-organized systems must be large enough and accessible enough to adapt or evolve.

6.2.2 Necessary Conditions for Robust Expressibility

We define the ε-biased product measure μ_ε on $\{\pm 1\}^n$ by $\mu_\varepsilon(x_1,\ldots,x_n) = \varepsilon^{n-k}(1-\varepsilon)^k$ where $k = |\{i : x_k = 1\}|$. We may view a function $f : \{\pm 1\}^n \to \{\pm 1\}$ as the characteristic function of a subset of $\{\pm 1\}^n$, that is, the subset $\{x \in \{\pm 1\}^n : f(x) = 1\}$. Then, $\mu_\varepsilon(f)$ denotes the weight assigned by the measure μ_ε to the set characterized by f.

Our main observation is the following.

Lemma 6.2.1. *$f : \{\pm 1\}^n \to \{\pm 1\}$ is ε-robustly expressible by a network of degree d if and only if there exist $\pi, g, \varphi_1,\ldots,\varphi_n$ such that:*

$$f(x_1,\ldots,x_n) = g(x_{\pi(1)} \cdot \varphi_1(), x_{\pi(2)} \cdot \varphi_2(x_{\pi(1)}), \ldots, x_{\pi(n)} \cdot \varphi_n(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n-1)})) \tag{6.2.2}$$

where:

(i) $\pi : [n] \to [n]$ *is a permutation,*

(ii) *for $i \in [n]$, $\varphi_i : \{\pm 1\}^{i-1} \to \{\pm 1\}$ is a Boolean function depending on at most d inputs, and*

(iii) $g : \{\pm 1\}^n \to \{\pm 1\}$ *such that $\mu_\varepsilon(g) \geq 1 - o(1)$.*

Proof. To prove one direction, suppose f is ε-robustly expressed by a Boolean network N of degree d. Since N is acyclic, there exists a permutation $\pi : [n] \to [n]$ such that there is an edge between node i and node j in N only if $\pi^{-1}(x) \leq \pi^{-1}(y)$. For every $i \in [n]$, let φ_i denote the update function associated with node $\pi(i)$ in the network. Note that for any i, the function φ_i can only take as arguments at most d elements of the set $\{x_{\pi(j)}\}_{j \leq i}$. Now, in an ε-mutation of N, each $x_{\pi(i)}\varphi_i(\cdots)$ is independently 1 with probability $1-\varepsilon$ and -1 with probability ε. Let $g(s_1,\ldots,s_n) = f(x_1,\ldots,x_n)$ where inductively, $x_{\pi(i)} = s_{\pi(i)}\varphi_i(x_{\pi(1)},\ldots,x_{\pi(i-1)})$ for each $i \in [n]$. One can explicitly verify now that Equation (6.2.2) holds for this choice of g. By definition of ε-robust expressibility, $g : \{\pm 1\}^n \to \{\pm 1\}$ is such that $\mu_\varepsilon(g) \geq 1 - o(1)$.

The proof in the other direction is similar. Given the permutation π and the functions $\varphi_1,\ldots,\varphi_n$, simply define a Boolean network N where π gives the ordering of the nodes and the φ_i's specify the update functions of the nodes. Then, the condition on g ensures that f is robustly expressed by the network. ∎

Theorem 6.2.2. *If f is robustly expressible by a Boolean network of constant degree, it is correlated with a function computable by a perceptron of constant degree.*

Corollary 6.2.3. *Any function robustly expressible by a Boolean network of constant degree can be learned in polynomial time (and logarithmic sample complexity).*

6.2.3 Sufficient Conditions for Robust Expressibility

Definition 6.2.4 (Sequential Cover)**.** *A bipartite graph $G = (V_1, V_2, E)$ with $|V_1| = m$ and $|V_2| = n$ is* sequentially coverable *if there exists a sequence of vertices $v_1,\ldots,v_k \in V_2$ for some $k \leq n$ such that the following two conditions hold:*

(i) Every vertex $v \in V_1$ is a neighbor of some v_i

(ii) Let $G_0 = G$. For $i \in [k]$, inductively define G_i as the induced graph on $G_{i-1} \backslash (\{v_i\} \cup \mathcal{N}(v_i))$. Each v_i is a vertex of degree exactly 1 in G_{i-1}.

The sequence $v_1,\ldots,v_k$ is called a sequential cover of size *k for G.*

A bipartite graph is thus sequentially coverable if the vertices of V_2 can be ordered in such a way that at most one vertex of V_1 is covered at a time. Note that $m \leq n$ necessarily if the graph is sequentially coverable.

Theorem 6.2.5. *If a function $f : \{\pm 1\}^n \to \{\pm 1\}$ has a sign-representation $\operatorname{sgn}(p(x_1,\ldots,x_n))$, such that $p(x)$ is a degree-d polynomial with constant coefficients and such that its term-variable bipartite graph, G_p, has a sequential cover of size $\Omega(n)$, then f is robustly expressible by a Boolean network of degree $d-1$.*

Proof. Let f be a function of the variables $X = \{x_1, \ldots, x_n\}$. We construct a Boolean network N that ε-robustly expresses f for some constant positive ε. Suppose that the term-variable graph G_p is sequentially covered by the sequence of variables $x_{i_1}, \ldots, x_{i_k} \in X$, where each i_j is a distinct element of $[n]$. Let $x_{i_{k+1}}, \ldots, x_{i_n}$ denote the rest of the variables (in some arbitrary order). For $j \in [k]$, let T_j denote the unique term covered by the variable x_{i_j}. Observe that for $j \in [k]$, T_j can only contain the variables $\{x_{i_\ell}\}_{\ell \geq j}$ and always contains x_{i_j}. In the Boolean network N, let the update function for the node associated with x_{i_j} be $u_{i_j} = \mathrm{sgn}(T_j/x_{i_j})$ for $j \in [k]$ and let u_{i_j} be an arbitrary element of $\{\pm 1\}$ for $j \in \{k+1, \ldots, n\}$. It is clear that N is an acyclic Boolean network.

We now show that f is ε-robustly expressed by N for some $\varepsilon \in (0,1)$. Observe that with probability at least $1 - 2^{-\Omega(n)}$, at most $2\varepsilon n$ mutations occur. There are a total of $\Omega(n)$ terms in p. The terms which correspond to mutated nodes are strictly negative, while those which are not are strictly positive, because of our choice of update functions. Since all the coefficients of p are constant, for a small enough constant ε, at most $2\varepsilon n$ mutations will not be enough to make p evaluate to a negative real. Hence, N expresses f with probability at least $1 - 2^{-\Omega(n)}$. ∎

Corollary 6.2.6. *There is a robustly expressible family of functions $f_n : \{\pm 1\}^n \to \{\pm 1\}$ that cannot be robustly expressed by a static assignment. In fact, for any constant $\varepsilon > 0$ and for any static assignment of $\{x_1, \ldots, x_n\}$, the probability that the assignment expressed by an ε-mutation of the assignment satisfies f_n is at most $2^{-\Omega(n)}$.*

Proof. For each $n \geq 1$, consider the function $f_n : \{\pm 1\}^n \to \{\pm 1\}$ where $f_n(x_1, \ldots, x_n) = \mathrm{sgn}(x_1 + x_1x_2 + x_1x_2x_3 + \cdots + x_1x_2 \cdots x_n - \frac{n}{4})$. Noting that the term-variable graph of $p(x) \stackrel{\text{def}}{=} x_1 + x_1x_2 + x_1x_2x_3 + \cdots + x_1x_2 \cdots x_n$ is sequentially covered by the sequence $x_n, \ldots, x_1$, it follows by a probabilistic argument similar to the one in the proof of Theorem 6.2.5, that the function f_n is ε-robustly expressible for a small enough constant ε.

On the other hand, we next show that f_n cannot be robustly expressed by any static assignment. Fix a static assignment for f_n, and consider an ε-mutation of it. Then, each x_i is an independent random variable that acquires $-1/1$ with probability $1 - \varepsilon$ and $1/-1$ with probability ε. For $i \in \{1, \ldots, n\}$, let $y_i = x_1x_2 \cdots x_i$. Now, $p(x) = \sum_i y_i$, and therefore, $|\mathbb{E}[p(x)]| = \sum_i |\mathbb{E} y_i| \leq \sum_i (1 - 2\varepsilon)^i \leq \frac{1-2\varepsilon}{2\varepsilon}$, a constant. We need to bound the concentration around this mean. Note that the y_i's are not independent; instead, they are generated by a Markov process. That is, $\Pr[y_i = a_i | y_{i-1} = a_{i-1}]$ can be specified by a 2-by-2 stochastic matrix, either $\begin{pmatrix} 1-\varepsilon & \varepsilon \\ \varepsilon & 1-\varepsilon \end{pmatrix}$ or $\begin{pmatrix} \varepsilon & 1-\varepsilon \\ 1-\varepsilon & \varepsilon \end{pmatrix}$. The eigenvalue gaps of these two matrices are 2ε and $2(1-\varepsilon)$ respectively. By a concentration bound on the sum of elements generated by a Markov chain with eigenvalue gap δ, given in Theorem 4.23 of [11], we have that $\Pr[|\mathbb{E}[\sum_i y_i] - \sum_i y_i| > n/8] \leq 2^{-\Omega(\delta n)}$. So, with probability at least $1 - 2^{-\Omega(n)}$, $\sum_i y_i < n/4$ and f_n is not satisfied. ∎

We will say that a sign-representation is *acyclic* if the term-variable graph contains no cycle. This allows us to present a more natural class of functions that are robustly expressible.

Theorem 6.2.7. *If a Boolean function $f : \{\pm 1\}^n \to \{\pm 1\}$ has an acyclic constant-degree sign-representation with constant coefficients and no degree-1 terms, then f is robustly expressible by a Boolean network of constant degree.*

Proof. We show that f has a sign-representation whose term-variable graph has a sequential cover of size $\Omega(n)$, thus proving our claim using Theorem 6.2.5. Let G be the term-variable bipartite graph for the given sign-representation for f. Since G is a forest by assumption, there must exist some (at least 2) degree-1 vertices. Furthermore, because there are no degree-1 monomials in the sign-representation, all the degree-1 vertices represent variables, not terms. We construct S, a sequential cover of G, as follows. Initially S is empty. Select some degree-1 vertex v in G and append it to S. Next, remove v from G and also all the vertices adjacent to v. Note that these adjacent vertices must represent terms. The modified graph is still a forest and must have some degree-1 vertices. Again, the degree-1 vertices must represent variables, not terms. Hence, we can repeat the process, appending a degree-1 vertex to S, remove it and its adjacent vertices from G, and so on. We stop when no vertices remain that represent terms.

It is clear that S is a sequential cover. We only need to show that S is of size $\Omega(n)$. This is so because each time a vertex is added to the sequential cover, we remove the unique term the associated variable is contained in, and this removal can make only a constant number of other vertices isolated (because each term is of constant degree). Hence, in order for all the variable vertices to either be in S or be isolated after the removal process, at least $\Omega(n)$ vertices need to be in S. ∎

We remark that given our other conditions, the "no degree-1 term" condition cannot be removed entirely. For instance, $f(x) = \text{sgn}(-x_1 + x_2 + (x_1 + x_2)(x_3 + \cdots + x_n))$ has an acyclic sign-representation which contains degree-1 monomials, but it is not robustly expressible, because with constant probability $x_1 = 1$ and $x_2 = -1$ which makes $f(x) = -1$ regardless of the values of $x_3, \ldots, x_n$.

Corollary 6.2.8. *There are at least $n^{\Omega(n)}$ functions on n variables that are robustly expressible.*

Proof. Use Cayley. ∎

Corollary 6.2.9. *There is a family of functions $f_n : \{\pm 1\}^n \to \{\pm 1\}$ such that it is robustly expressible by a constant degree Boolean network but does not have a robust static assignment.*

Proof. For each n, consider $f_n(x_1, \ldots, x_n) = \text{sgn}(x_1x_2 + x_1x_3 + \cdots + x_1x_n)$. f_n satisfies the conditions of Theorem 6.2.7 and hence is robustly expressible by a constant degree Boolean network. On the other hand, for any static assignment, x_1 could be assigned to the complement of $\text{sgn}(x_2 + \cdots + x_n)$ with constant probability, so that the assignment would not satisfy f_n. ∎

Algorithmic Complexity of Constructing Robust Boolean Networks

Theorem 6.2.10. *Suppose $f : \{\pm 1\}^n \to \{\pm 1\}$ is known to be ε-robustly expressible. Then, if f has a constant-size decision tree, a robust Boolean network expressing f can be constructed in polynomial time with probability $1 - \frac{1}{\text{poly}(n)}$, using PAC queries and uniformly distributed examples. Also, if f is chosen uniformly at random from the set of all functions with logarithmic decision tree depth, then a robust Boolean network expressing f can be constructed in polynomial time with probability $1 - \frac{1}{\text{poly}(n)}$, using PAC queries and uniformly distributed examples.*

Proof. Show how to find robust Boolean network from decision tree representation of function. Use Ehrenfeucht-Haussler and Blum for first statement (class of size s decision trees can be PAC learned in $n^{\log s}$ time steps). Use Jackson-Servedio for second statement. ∎

Open Problems While this is a promising start toward characterizing the necessary conditions for self-organization, adaptability, and evolvability, further work is necessary particularly in generalizing the conditions placed on the structure of Boolean networks. The acyclic graph requirement for the networks considered in this section is necessary to ensure that there is a stationary fixed point. While most evolvable systems *do* have stationary fixed points, they arrive at them in a more general way than this drastic requirement. Future work should focus on relaxing this retriction.

Chapter 7

Discussion

7.1 Significance

We have demonstrated the usefulness of renormalization-group methods for constructing reduced models of complex systems that preserve emergent behavior, and have provided mathematical constraints on the structure of complex systems that are robust to perturbations. Although we have worked with idealized complex system models—cellular automata and Boolean networks—our approaches are extensible to more general entity-based models. In combination, our results provide important guidance for more rigorous construction of entity-based models, which currently are often devised in an ad-hoc manner. Our results can also help in designing complex systems with the goal of predictable behavior, e.g., for cybersecurity. This work has potential applications to Sandia modeling and simulation efforts in the ERN, HSD, and DS&A SMUs, such as energy infrastructure and financial networks. By providing a clearer relation between model construction and emergent behavior, we can more efficiently identify appropriate entity-based models for a given domain.

One way of applying this work is through renormalization of a known detailed model that is not feasible to simulate in full. The systematic approach of renormalization offers a controlled study of the abstraction process that is involved in constructing almost all real-world models. A reduced model not only allows greater opportunities for extensive simulation, but offers a more intuitive picture in terms of higher-level entities. Our results for cellular automata indicate that as the desired level of description of a system becomes coarser, the predictivity horizon increases. Thus, if we are interested in the large-scale behavior of the system, a coarse model can track this behavior accurately, well past the time at which the smaller-scale predictions of a detailed model may become inaccurate.

Another way of applying this work, in the absence of a known detailed model, is through model construction based on known or desired emergent behavior. In particular, many real-world systems exhibit properties of self-organization, scale invariance, and robustness. Our results, along with those in other literature, provide known families of idealized complex systems that manifest such behaviors. Entity models obtained by renormalization of these idealized systems are good candidates for modeling real-world systems in such a way that their key emergent behaviors are reproduced. Furthermore, engineering design of complex systems can benefit from attempting to build actual entities that exhibit the same responses as those in a model with desirable types of

emergent behavior.

7.2 Future Directions

Several further technical investigations are suggested by this work. A limitation of our current renormalization process is that coarsening occurs in space but not in time. Because a coarsened model generally has slower evolution, it should be possible to construct a coarsened model whose timestep corresponds to more than one timestep of the original model. This would ameliorate the increasing cost of successive renormalizations, as seen in the sandpile model. However, because time is one-dimensional whereas space is typically two- or three-dimensional, most of the efficiency gain from a reduced model has already been obtained through spatial coarsening.

In a broader setting, the relation of the idealized models we have considered to more general entity-based models would benefit from further study. We can argue that any entity-based model that can be simulated in a digital computer is necessarily equivalent to some Boolean network, but the effect on tractability of viewing the model in this way remains to be determined. From the other direction, it is important to extend the renormalization of Boolean networks so that the resulting coarse model may be more general than a Boolean network. In this way, Boolean networks can be a source of useful entity-based models for real-world applications, and our results on the robustness of Boolean networks can inform the selection of such entity-based models.

Finally, an important property of some real-world systems that is not reflected in our idealized models is dynamic adaptation of networks, i.e., connectivity that changes in time. This presents interesting challenges to the renormalization concept. If the emergent behavior of dynamic adaptive networks can be studied with systematic computational and theoretical tools analogous to those we have developed here, then much better guidance on the construction of adaptive entity-based models will be available. This will further advance the capabilities of modeling and simulation for addressing vital problems.

References

[1] R. Albert and H. Othmer. The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in *Drosophila melanogaster*. *J. Theor. Biol.*, 233:1–18, 2003.

[2] M. Aldana, E. Balleza, S. Kauffman, and O. Resendiz. Robustness and evolvability in genetic regulatory networks. *J. Theor. Biol.*, 245:433–448, 2007.

[3] M. Aldana, S. Coppersmith, and L. P. Kadanoff. Boolean dynamics with random couplings. In E. Kaplan, J. E. Marsden, and K. R. Sreenivasan, editors, *Perspectives and Problems in Nonlinear Science*. Springer, New York, 2003.

[4] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: An explanation of $1/f$ noise. *Phys. Rev. Lett.*, 59:381–384, 1987.

[5] A.-L. Barabási. *Linked: The New Science of Networks*. Perseus, Cambridge, MA, 2002.

[6] S. Braunewell and S. Bornholdt. Reliability of genetic networks is evolvable. *Phys. Rev. E*, 77:060902, 2008.

[7] J. M. Carlson and J. Doyle. Highly optimized tolerance: A mechanism for power laws in designed systems. *Phys. Rev. E*, 60:1412–1427, 1999.

[8] G. Csárdi and T. Nepusz. The igraph software package for complex network research. *InterJournal Complex Systems*, page 1695, 2006.

[9] M. Davidich and S. Bornholdt. From differential equations to Boolean networks: A case study in modeling regulatory networks. `http://arxiv.org/abs/0807.1013v2`, 2008.

[10] I. Dobson, B. A. Carreras, V. E. Lynch, and D. E. Newman. Complex systems analysis of series of blackouts: Cascading failure, critical points, and self-organization. *Chaos*, 17:026103, 2007.

[11] D. P. Dubhashi and A. Panconesi. Concentration of measure for the analysis of randomised algorithms. Draft Monograph, 2008. `http://www.dsi.uniroma1.it/~ale/Papers/master.pdf`.

[12] C. Espinosa-Soto, P. Padilla-Longoria, and E. Alvaraez-Buylla. A gene regulatory network model for cell-fate determination during *Arabidopsis thaliana* flower development that is robust and recovers gene expression profiles. *Plant Cell*, 16:2923–2939, 2004.

[13] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*. Network Theory Limited, Bristol, England, second edition, 2006.

[14] M. Gardner. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life". *Sci. Am.*, 223:120–123, Oct. 1970.

[15] T. S. Gardner, D. di Bernardo, D. Lorenz, and J. J. Collins. Inferring genetic networks and identifying compound mode of action via expression profiling. *Science*, 301:102–105, 2003.

[16] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. *Proc. 2008 USENIX Security Symposium*.

[17] I. Harvey and T. Bossomaier. Time out of joint: Attractors in asynchronous random Boolean networks. In P. Husbands and I. Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life (ECAL97)*, pages 67–75. MIT Press, 1997.

[18] S. Huang, G. Eichler, Y. Bar-Yam, and D. E. Ingber. Cell fates as high-dimensional attractor states of a complex gene regulatory network. *Phys. Rev. Lett.*, 94:128701, 2005.

[19] S. Huang and D. Ingber. Shape-dependent control of cell growth, differentiation and apoptosis: Switching between attractors in cell regulatory networks. *Exp. Cell Res.*, 261:91–103, 2000.

[20] N. Israeli and N. Goldenfeld. Coarse-graining of cellular automata, emergence, and the predictability of complex systems. *Phys. Rev. E*, 73:026203, 2006.

[21] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed nets. *J. Theor. Biol.*, 22:437–467, 1969.

[22] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993.

[23] S. A. Kauffman. *At Home in the Universe*. Oxford University Press, 1995.

[24] F. Kun, G. Kocsis, and J. Farkas. Cellular automata for the spreading of technologies in socio-economic systems. *Physica A*, 383:660–670, 2007.

[25] L. Mendoza and E. Alvarez-Buylla. Genetic regulation of root hair development in *Arabidopsis thaliana*: A network model. *J. Theor. Biol.*, 204:311–326, 2000.

[26] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Phys. Rev. Lett.*, 86:3200–3203, 2001.

[27] S. Rankin. Mathematical sciences in the FY 2007 budget. *AAAS Report XXXI*, 2007.

[28] J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Phys. Rev. E*, 74:016110, 2006.

[29] T. C. Shelling. Dynamic models of segregation. *J. Math. Sociol.*, 1:143–168, 1971.

[30] C. Song, S. Havlin, and H. A. Makse. Self-similarity of complex networks. *Nature*, 433:392–395, 2005.

[31] K. Soramäki, M. L. Bech, J. Arnold, R. J. Glass, and W. E. Beyeler. The topology of interbank payment flows. *Physica A*, 379:317–333, 2007.

[32] A. Szejka and B. Drossel. Evolution of canalizing Boolean networks. *Eur. Phys. J. B*, 56:373–380, 2007.

[33] A. Wagner. Robustness, evolvability, and neutrality. *FEBS Lett.*, 579:1772–1778, 2005.

[34] S. Wolfram. *A New Kind of Science*. Wolfram Media, Champaign, IL, 2002.

DISTRIBUTION:

1	MS 0899	Technical Library, 9536 (electronic)
1	MS 0123	D. Chavez, LDRD Office, 1011


Sandia National Laboratories